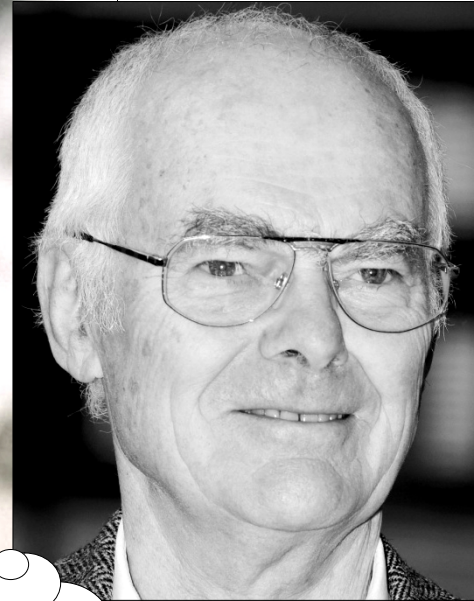


Nakamichi-Draconeye-Kaidanji=The_Zennish_Microdeduplicator



Leprechaun returns ... the teraripper (from Greek *teras* 'monster', thus, **monstrous ripper**) is back to **shamrock** your searches — Prof. Rudolf Bayer's B-tree order 3 implemented by machinely yours Sanmayce is to replace Railgun 'Trolldom' i.e. memmem() approach in Nakamichi parser.

All matches — 4,6,8 bytes long, for a start — are to be findable without any hashing, just in a few RAM fetches. Simply, all the matches (put as keys in the leaves) in the B-tree are to be paired with their last seen offset, hence the immediateness.

¹ shamrock is a 3-leaf clover – a symbol of Ireland



Nakamichi

Author: Georgi 'Sanmayce' Marinov, e-mail: sanmayce@sanmayce.com, Twitter: <https://twitter.com/Sanmayce>
Last update: 2021-Aug-30

Enfun!

Nakamichi_Ryuugan-ditto-1TB_btree.c:

```
00,001 // Nakamichi is 100% FREE LZSS SUPERFAST decompressor.
00,002 // Home of Nakamichi: www.sarnmayce.com/Nakamichi/index.html
00,003
00,004 // Downloadable at:
00,005 // The 'Large Text Compression Benchmark' main revision from 2019-Aug-06:
00,006 // https://gist.github.com/Sarnmayce/33e5047d45cdcb8e7711cd7d3ed52c7f/raw/d72e7126c8fbfde07c0d727dcb353b0267b8196c/Nakamichi_Ryuugan-ditto-1TB.c
00,007 // Other older ones:
00,008 // https://community.centminmod.com/data/attachment-files/2019/07/8164_Nakamichi_2019-Jul-01.zip
00,009 // https://community.centminmod.com/threads/a-lzss-microdeduplicator-tagetting-huge-texts-with-c-source.16427/#post-75533
00,010
00,011 // 2019-Nov-28: Jesteress was replaced by 'Pippip':
00,012 // https://www.codeproject.com/Articles/716530/Fastest-Hash-Function-for-Table-Lookups-in-C
00,013 // https://softwareengineering.stackexchange.com/questions/401167/searching-hashing-wikipedia-at-each-position-for-keys-1-31-long-the-fastest
00,014
00,015 // How to compile? Windows?
00,016 /*
00,017 Intel(R) Parallel Studio XE 2015
00,018 Copyright (C) 1985-2014 Intel Corporation. All rights reserved.
00,019 Intel(R) Parallel Studio XE 2015 Composer Edition (package 108)
00,020 Setting environment for using Microsoft Visual Studio 2010 x64 cross tools.
00,021
00,022 C:\Program Files\Intel\Composer XE 2015>
00,023
00,024 E:\Nakamichi_Kanshiketsu_source_binaries_booklet_2021-Aug-28\Kanshiketsu_AES_sourcecode>_MakeEXE_Nakamichi_ICL.bat
00,025
00,026 E:\Nakamichi_Kanshiketsu_source_binaries_booklet_2021-Aug-28\Kanshiketsu_AES_sourcecode>icl /TP /O3 Nakamichi_Ryuugan-ditto-1TB_btree.c /FeNakamichi_DD-192.exe -
D_N_YMM -D_N_prefetch_4096 -D_N_HIGH_PR
00,027 IORITY -D_icl_mumbo_jumbo_/Facs -DHashInBITS=24 -DHashChunkSizeInBITS=24 -DRAMpoolInKB=5120 -DBtreeHEURISTIC -D_WIN32_ENVIRONMENT_ -DLongestLineInclusive=128 -
DSpeedUpBuilding=1000 -D_NDD -DLITE -D_N
00,028 alone
00,029 Intel(R) C++ Compiler XE for applications running on Intel(R) 64, Version 15.0.0.108 Build 20140726
00,030 Copyright (C) 1985-2014 Intel Corporation. All rights reserved.
00,031
00,032 Nakamichi_Ryuugan-ditto-1TB_btree.c
00,033 Microsoft (R) Incremental Linker Version 10.00.40219.01
00,034 Copyright (C) Microsoft Corporation. All rights reserved.
00,035
00,036 -out:Nakamichi_DD-192.exe
00,037 Nakamichi_Ryuugan-ditto-1TB_btree.obj
00,038
00,039 E:\Nakamichi_Kanshiketsu_source_binaries_booklet_2021-Aug-28\Kanshiketsu_AES_sourcecode>icl /TP /O3 Nakamichi_Ryuugan-ditto-1TB_btree.c /FeNakamichi_SHA3-192.exe -
D_N_YMM -D_N_prefetch_4096 -D_N_HIGH_
00,040 PRIORITY -D_icl_mumbo_jumbo_/Facs -DHashInBITS=24 -DHashChunkSizeInBITS=24 -DRAMpoolInKB=5120 -DBtreeHEURISTIC -D_WIN32_ENVIRONMENT_ -DLongestLineInclusive=128 -
DSpeedUpBuilding=1000 -D_NSHA3 -DLITE
00,041 -D_N_alone
00,042 Intel(R) C++ Compiler XE for applications running on Intel(R) 64, Version 15.0.0.108 Build 20140726
00,043 Copyright (C) 1985-2014 Intel Corporation. All rights reserved.
00,044
00,045 Nakamichi_Ryuugan-ditto-1TB_btree.c
00,046 Microsoft (R) Incremental Linker Version 10.00.40219.01
00,047 Copyright (C) Microsoft Corporation. All rights reserved.
00,048
00,049 -out:Nakamichi_SHA3-192.exe
00,050 Nakamichi_Ryuugan-ditto-1TB_btree.obj
00,051
00,052 E:\Nakamichi_Kanshiketsu_source_binaries_booklet_2021-Aug-28\Kanshiketsu_AES_sourcecode>icl /TP /O3 Nakamichi_Ryuugan-ditto-1TB_btree.c /FeNakamichi_PRV-192.exe -
```

```
D_N_YMM -D_N_prefetch_4096 -D_N_HIGH_P
00,053 RRIORITY -D_icl_mumbo_jumbo_ /Facs -DHashInBITS=24 -DHashChunkSizeInBITS=24 -DRAMpoolInKB=5120 -DBtreeHEURISTIC -D_WIN32_ENVIRONMENT_ -DLongestLineInclusive=128 -
DSpeedUpBuilding=1000 -D_NPRV -DLITE -D
00,054 _N_alone
00,055 Intel(R) C++ Compiler XE for applications running on Intel(R) 64, Version 15.0.0.108 Build 20140726
00,056 Copyright (C) 1985-2014 Intel Corporation. All rights reserved.
00,057
00,058 Nakamichi_Ryuugan-ditto-1TB_btree.c
00,059 Microsoft (R) Incremental Linker Version 10.00.40219.01
00,060 Copyright (C) Microsoft Corporation. All rights reserved.
00,061
00,062 -out:Nakamichi_PRV-192.exe
00,063 Nakamichi_Ryuugan-ditto-1TB_btree.obj
00,064
00,065 E:\Nakamichi_Kanshiketsu_source_binaries_booklet_2021-Aug-28\Kanshiketsu_AES_sourcecode>icl /TP /O3 Nakamichi_Ryuugan-ditto-1TB_btree.c
/FeNakamichi_Vanilla_LITE_Kaidanji_ICC_64bit.exe -D_N_YMM -D_N_p
00,066 refetch_4096 -D_N_HIGH_PRIORITY -D_icl_mumbo_jumbo_ /Facs -DHashInBITS=24 -DHashChunkSizeInBITS=24 -DRAMpoolInKB=5120 -DBtreeHEURISTIC -D_WIN32_ENVIRONMENT_ -
DLongestLineInclusive=128 -DSpeedUpBuildin
00,067 g=1000 -DLITE -D_N_alone -DKaidanji
00,068 Intel(R) C++ Compiler XE for applications running on Intel(R) 64, Version 15.0.0.108 Build 20140726
00,069 Copyright (C) 1985-2014 Intel Corporation. All rights reserved.
00,070
00,071 Nakamichi_Ryuugan-ditto-1TB_btree.c
00,072 Microsoft (R) Incremental Linker Version 10.00.40219.01
00,073 Copyright (C) Microsoft Corporation. All rights reserved.
00,074
00,075 -out:Nakamichi_Vanilla_LITE_Kaidanji_ICC_64bit.exe
00,076 Nakamichi_Ryuugan-ditto-1TB_btree.obj
00,077
00,078 E:\Nakamichi_Kanshiketsu_source_binaries_booklet_2021-Aug-28\Kanshiketsu_AES_sourcecode>icl /TP /O3 Nakamichi_Ryuugan-ditto-1TB_btree.c
/FeNakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_ICC_64bit.exe -D_
00,079 N_YMM -D_N_prefetch_4096 -D_N_HIGH_PRIORITY -D_icl_mumbo_jumbo_ /Facs -DHashInBITS=24 -DHashChunkSizeInBITS=24 -DRAMpoolInKB=5120 -DBtreeHEURISTIC -
D_WIN32_ENVIRONMENT_ -DLongestLineInclusive=128 -DSp
00,080 eedUpBuilding=1000 -DLITE -D_N_alone -DKaidanji -DKanshiketsu
00,081 Intel(R) C++ Compiler XE for applications running on Intel(R) 64, Version 15.0.0.108 Build 20140726
00,082 Copyright (C) 1985-2014 Intel Corporation. All rights reserved.
00,083
00,084 Nakamichi_Ryuugan-ditto-1TB_btree.c
00,085 Microsoft (R) Incremental Linker Version 10.00.40219.01
00,086 Copyright (C) Microsoft Corporation. All rights reserved.
00,087
00,088 -out:Nakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_ICC_64bit.exe
00,089 Nakamichi_Ryuugan-ditto-1TB_btree.obj
00,090
00,091 E:\Nakamichi_Kanshiketsu_source_binaries_booklet_2021-Aug-28\Kanshiketsu_AES_sourcecode>icl /TP /O3 Nakamichi_Ryuugan-ditto-1TB_btree.c
/FeNakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_DD-128AES_ICC_64b
00,092 it.exe -D_N_YMM -D_N_prefetch_4096 -D_N_HIGH_PRIORITY -D_icl_mumbo_jumbo_ /Facs -DHashInBITS=24 -DHashChunkSizeInBITS=24 -DRAMpoolInKB=5120 -DBtreeHEURISTIC -
D_WIN32_ENVIRONMENT_ -DLongestLineInclusiv
00,093 e=128 -DSpeedUpBuilding=1000 -DLITE -D_N_alone -DKaidanji -D_NaquaHash -DKanshiketsu
00,094 Intel(R) C++ Compiler XE for applications running on Intel(R) 64, Version 15.0.0.108 Build 20140726
00,095 Copyright (C) 1985-2014 Intel Corporation. All rights reserved.
00,096
00,097 Nakamichi_Ryuugan-ditto-1TB_btree.c
00,098 Microsoft (R) Incremental Linker Version 10.00.40219.01
00,099 Copyright (C) Microsoft Corporation. All rights reserved.
00,100
00,101 -out:Nakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_DD-128AES_ICC_64bit.exe
00,102 Nakamichi_Ryuugan-ditto-1TB_btree.obj
```

```
00,103
00,104 E:\Nakamichi_Kanshiketsu_source_binaries_booklet_2021-Aug-28\Kanshiketsu_AES_sourcecode>icl /TP /O3 Nakamichi_Ryuugan-ditto-1TB_btree.c
/FeNakamichi_Vanilla_LITE_Kaidanji_DD-128AES_ICC_64bit.exe -D_N_
00,105 YMM -D_N_prefetch_4096 -D_N_HIGH_PRIORITY -D_icl_mumbo_jumbo_ /FAcs -DHashInBITS=24 -DHashChunkSizeInBITS=24 -DRAMpoolInKB=5120 -DBtreeHEURISTIC -D_WIN32_ENVIRONMENT_
-DLongestLineInclusive=128 -DSpee
00,106 dUpBuilding=1000 -DLITE -D_N_alone -DKaidanji -D_NaquaHash
00,107 Intel(R) C++ Compiler XE for applications running on Intel(R) 64, Version 15.0.0.108 Build 20140726
00,108 Copyright (C) 1985-2014 Intel Corporation. All rights reserved.
00,109
00,110 Nakamichi_Ryuugan-ditto-1TB_btree.c
00,111 Microsoft (R) Incremental Linker Version 10.00.40219.01
00,112 Copyright (C) Microsoft Corporation. All rights reserved.
00,113
00,114 -out:Nakamichi_Vanilla_LITE_Kaidanji_DD-128AES_ICC_64bit.exe
00,115 Nakamichi_Ryuugan-ditto-1TB_btree.obj
00,116
00,117 E:\Nakamichi_Kanshiketsu_source_binaries_booklet_2021-Aug-28\Kanshiketsu_AES_sourcecode>dir *.exe
00,118
00,119 08/29/2021 12:04 AM 83,456 Nakamichi_DD-192.exe
00,120 08/29/2021 12:05 AM 83,456 Nakamichi_PRV-192.exe
00,121 08/29/2021 12:05 AM 84,480 Nakamichi_SHA3-192.exe
00,122 08/29/2021 12:05 AM 84,480 Nakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_DD-128AES_ICC_64bit.exe
00,123 08/29/2021 12:05 AM 83,456 Nakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_ICC_64bit.exe
00,124 08/29/2021 12:05 AM 83,456 Nakamichi_Vanilla_LITE_Kaidanji_DD-128AES_ICC_64bit.exe
00,125 08/29/2021 12:05 AM 82,432 Nakamichi_Vanilla_LITE_Kaidanji_ICC_64bit.exe
00,126
00,127 E:\Nakamichi_Kanshiketsu_source_binaries_booklet_2021-Aug-28\Kanshiketsu_AES_sourcecode>
00,128 */
00,129
00,130 // How to compile? Linux?
00,131 /*
00,132 E:\Nakamichi_Kanshiketsu_source_binaries_booklet_2021-Aug-28\Kanshiketsu_AES_sourcecode>type _MakeELF_Nakamichi_GCC.sh
00,133 #!/usr/bin/bash
00,134
00,135 gcc -O3 -static -mavx2 -fomit-frame-pointer Nakamichi_Ryuugan-ditto-1TB_btree.c -o Nakamichi_DD-192.elf -D_N_YMM -D_N_prefetch_4096 -D_N_alone -DHashInBITS=24 -
DHashChunkSizeInBITS=24 -DRAMpoolInKB=51
00,136 20 -DBtreeHEURISTIC -D_POSIX_ENVIRONMENT_ -DLongestLineInclusive=128 -DSpeedUpBuilding=320 -DLITE -D_NDD
00,137 gcc -O3 -static -mavx2 -fomit-frame-pointer Nakamichi_Ryuugan-ditto-1TB_btree.c -o Nakamichi_PRV-192.elf -D_N_YMM -D_N_prefetch_4096 -D_N_alone -DHashInBITS=24 -
DHashChunkSizeInBITS=24 -DRAMpoolInKB=5
00,138 120 -DBtreeHEURISTIC -D_POSIX_ENVIRONMENT_ -DLongestLineInclusive=128 -DSpeedUpBuilding=320 -DLITE -D_NPRV
00,139 gcc -O3 -static -mavx2 -fomit-frame-pointer Nakamichi_Ryuugan-ditto-1TB_btree.c -o Nakamichi_SHA3-192.elf -D_N_YMM -D_N_prefetch_4096 -D_N_alone -DHashInBITS=24 -
DHashChunkSizeInBITS=24 -DRAMpoolInKB=
00,140 5120 -DBtreeHEURISTIC -D_POSIX_ENVIRONMENT_ -DLongestLineInclusive=128 -DSpeedUpBuilding=320 -DLITE -D_NSHA3
00,141
00,142 x86_64-w64-mingw32-gcc -O3 -m64 -static -mavx2 -fomit-frame-pointer Nakamichi_Ryuugan-ditto-1TB_btree.c -o Nakamichi_DD-192.exe -D_N_YMM -D_N_prefetch_4096 -D_N_alone
-DHashInBITS=24 -DHashChunkSizeIn
00,143 BITS=24 -DRAMpoolInKB=5120 -DBtreeHEURISTIC -D_WIN32_ENVIRONMENT_ -DLongestLineInclusive=128 -DSpeedUpBuilding=320 -DLITE -D_NDD
00,144 x86_64-w64-mingw32-gcc -O3 -m64 -static -mavx2 -fomit-frame-pointer Nakamichi_Ryuugan-ditto-1TB_btree.c -o Nakamichi_PRV-192.exe -D_N_YMM -D_N_prefetch_4096 -D_N_alone
-DHashInBITS=24 -DHashChunkSizeI
00,145 nBITS=24 -DRAMpoolInKB=5120 -DBtreeHEURISTIC -D_WIN32_ENVIRONMENT_ -DLongestLineInclusive=128 -DSpeedUpBuilding=320 -DLITE -D_NPRV
00,146 x86_64-w64-mingw32-gcc -O3 -m64 -static -mavx2 -fomit-frame-pointer Nakamichi_Ryuugan-ditto-1TB_btree.c -o Nakamichi_SHA3-192.exe -D_N_YMM -D_N_prefetch_4096 -
D_N_alone -DHashInBITS=24 -DHashChunkSize
00,147 InBITS=24 -DRAMpoolInKB=5120 -DBtreeHEURISTIC -D_WIN32_ENVIRONMENT_ -DLongestLineInclusive=128 -DSpeedUpBuilding=320 -DLITE -D_NSHA3
00,148
00,149 gcc -O3 -m64 -static -mavx2 -fomit-frame-pointer Nakamichi_Ryuugan-ditto-1TB_btree.c -o Nakamichi_Vanilla_LITE_Kaidanji_GCC_64bit.elf -D_N_YMM -D_N_prefetch_4096 -
D_N_alone -DHashInBITS=24 -DHashChunk
00,150 SizeInBITS=24 -DRAMpoolInKB=5120 -DBtreeHEURISTIC -D_POSIX_ENVIRONMENT_ -DLongestLineInclusive=128 -DSpeedUpBuilding=1000 -DLITE -DKaidanji
00,151 gcc -O3 -m64 -static -mavx2 -fomit-frame-pointer Nakamichi_Ryuugan-ditto-1TB_btree.c -o Nakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_GCC_64bit.elf -D_N_YMM -
```



```
D_N_prefetch_4096 -D_N_alone -DHashInBITS=24
00,152 -DHashChunkSizeInBITS=24 -DRAMpoolInKB=5120 -DBtreeHEURISTIC -D_POSIX_ENVIRONMENT_ -DLongestLineInclusive=128 -DSpeedUpBuilding=1000 -DLITE -DKaidanji -DKanshiketsu
00,153 gcc -O3 -m64 -static -mavx2 -maes -fomit-frame-pointer Nakamichi_Ryuugan-ditto-1TB_btree.c -o Nakamichi_Vanilla_LITE_Kaidanji_DD-128AES_GCC_64bit.elf -D_N_YMM -
D_N_prefetch_4096 -D_N_alone -DHashInBIT
00,154 S=24 -DHashChunkSizeInBITS=24 -DRAMpoolInKB=5120 -DBtreeHEURISTIC -D_POSIX_ENVIRONMENT_ -DLongestLineInclusive=128 -DSpeedUpBuilding=1000 -DLITE -DKaidanji -
D_NaquaHash
00,155 gcc -O3 -m64 -static -mavx2 -maes -fomit-frame-pointer Nakamichi_Ryuugan-ditto-1TB_btree.c -o Nakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_DD-128AES_GCC_64bit.elf -
D_N_YMM -D_N_prefetch_4096 -D_N_alone
00,156 -DHashInBITS=24 -DHashChunkSizeInBITS=24 -DRAMpoolInKB=5120 -DBtreeHEURISTIC -D_POSIX_ENVIRONMENT_ -DLongestLineInclusive=128 -DSpeedUpBuilding=1000 -DLITE -DKaidanji
-D_NaquaHash -DKanshiketsu
00,157
00,158 x86_64-w64-mingw32-gcc -O3 -m64 -static -mavx2 -fomit-frame-pointer Nakamichi_Ryuugan-ditto-1TB_btree.c -o Nakamichi_Vanilla_LITE_Kaidanji_GCC_64bit.exe -D_N_YMM -
D_N_prefetch_4096 -D_N_alone -D_N_HIG
00,159 H_PRIORITY -DHashInBITS=24 -DHashChunkSizeInBITS=24 -DRAMpoolInKB=5120 -DBtreeHEURISTIC -D_WIN32_ENVIRONMENT_ -DLongestLineInclusive=128 -DSpeedUpBuilding=1000 -DLITE
-DKaidanji
00,160 x86_64-w64-mingw32-gcc -O3 -m64 -static -mavx2 -fomit-frame-pointer Nakamichi_Ryuugan-ditto-1TB_btree.c -o Nakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_GCC_64bit.exe -
D_N_YMM -D_N_prefetch_4096 -D_N_al
00,161 one -D_N_HIGH_PRIORITY -DHashInBITS=24 -DHashChunkSizeInBITS=24 -DRAMpoolInKB=5120 -DBtreeHEURISTIC -D_WIN32_ENVIRONMENT_ -DLongestLineInclusive=128 -
DSpeedUpBuilding=1000 -DLITE -DKaidanji -DKanshike
00,162 tsu
00,163 x86_64-w64-mingw32-gcc -O3 -m64 -static -mavx2 -maes -fomit-frame-pointer Nakamichi_Ryuugan-ditto-1TB_btree.c -o Nakamichi_Vanilla_LITE_Kaidanji_DD-
128AES_GCC_64bit.exe -D_N_YMM -D_N_prefetch_4096 -D_
00,164 N_alone -D_N_HIGH_PRIORITY -DHashInBITS=24 -DHashChunkSizeInBITS=24 -DRAMpoolInKB=5120 -DBtreeHEURISTIC -D_WIN32_ENVIRONMENT_ -DLongestLineInclusive=128 -
DSpeedUpBuilding=1000 -DLITE -DKaidanji -D_Naq
00,165 uaHash
00,166 x86_64-w64-mingw32-gcc -O3 -m64 -static -mavx2 -maes -fomit-frame-pointer Nakamichi_Ryuugan-ditto-1TB_btree.c -o Nakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_DD-
128AES_GCC_64bit.exe -D_N_YMM -D_N_prefe
00,167 tch_4096 -D_N_alone -D_N_HIGH_PRIORITY -DHashInBITS=24 -DHashChunkSizeInBITS=24 -DRAMpoolInKB=5120 -DBtreeHEURISTIC -D_WIN32_ENVIRONMENT_ -DLongestLineInclusive=128 -
DSpeedUpBuilding=1000 -DLITE -DKai
00,168 danji -D_NaquaHash -DKanshiketsu
00,169
00,170 strip -s Nakamichi_DD-192.elf
00,171 strip -s Nakamichi_PRV-192.elf
00,172 strip -s Nakamichi_SHA3-192.elf
00,173
00,174 strip -s Nakamichi_DD-192.exe
00,175 strip -s Nakamichi_PRV-192.exe
00,176 strip -s Nakamichi_SHA3-192.exe
00,177
00,178 strip -s Nakamichi_Vanilla_LITE_Kaidanji_GCC_64bit.elf
00,179 strip -s Nakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_GCC_64bit.elf
00,180 strip -s Nakamichi_Vanilla_LITE_Kaidanji_DD-128AES_GCC_64bit.elf
00,181 strip -s Nakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_DD-128AES_GCC_64bit.elf
00,182
00,183 strip -s Nakamichi_Vanilla_LITE_Kaidanji_GCC_64bit.exe
00,184 strip -s Nakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_GCC_64bit.exe
00,185 strip -s Nakamichi_Vanilla_LITE_Kaidanji_DD-128AES_GCC_64bit.exe
00,186 strip -s Nakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_DD-128AES_GCC_64bit.exe
00,187
00,188 upx -9 Nakamichi_DD-192.elf
00,189 upx -9 Nakamichi_PRV-192.elf
00,190 upx -9 Nakamichi_SHA3-192.elf
00,191
00,192 upx -9 Nakamichi_DD-192.exe
00,193 upx -9 Nakamichi_PRV-192.exe
00,194 upx -9 Nakamichi_SHA3-192.exe
00,195
00,196 upx -9 Nakamichi_Vanilla_LITE_Kaidanji_GCC_64bit.elf
```

```
00,197 upx -9 Nakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_GCC_64bit.elf
00,198 upx -9 Nakamichi_Vanilla_LITE_Kaidanji_DD-128AES_GCC_64bit.elf
00,199 upx -9 Nakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_DD-128AES_GCC_64bit.elf
00,200
00,201 upx -9 Nakamichi_Vanilla_LITE_Kaidanji_GCC_64bit.exe
00,202 upx -9 Nakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_GCC_64bit.exe
00,203 upx -9 Nakamichi_Vanilla_LITE_Kaidanji_DD-128AES_GCC_64bit.exe
00,204 upx -9 Nakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_DD-128AES_GCC_64bit.exe
00,205
00,206 E:\Nakamichi_Kanshiketsu_source_binaries_booklet_2021-Aug-28\Kanshiketsu_AES_sourcecode>
00,207 */
00,208
00,209 /*
00,210 [kaze@kaze Kanshiketsu_AES_sourcecode]$ sh _MakeELF_Nakamichi_GCC.sh
00,211 strip: Nakamichi_DD-192.elf[.gnu.build.attributes_libc_freeres_fn]: Warning: version note missing - assuming version 3
00,212 strip: Nakamichi_PRV-192.elf[.gnu.build.attributes_libc_freeres_fn]: Warning: version note missing - assuming version 3
00,213 strip: Nakamichi_SHA3-192.elf[.gnu.build.attributes_libc_freeres_fn]: Warning: version note missing - assuming version 3
00,214 strip: Nakamichi_Vanilla_LITE_Kaidanji_GCC_64bit.elf[.gnu.build.attributes_libc_freeres_fn]: Warning: version note missing - assuming version 3
00,215 strip: Nakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_GCC_64bit.elf[.gnu.build.attributes_libc_freeres_fn]: Warning: version note missing - assuming version 3
00,216 strip: Nakamichi_Vanilla_LITE_Kaidanji_DD-128AES_GCC_64bit.elf[.gnu.build.attributes_libc_freeres_fn]: Warning: version note missing - assuming version 3
00,217 strip: Nakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_DD-128AES_GCC_64bit.elf[.gnu.build.attributes_libc_freeres_fn]: Warning: version note missing - assuming version 3
00,218 Ultimate Packer for eXecutables
00,219 Copyright (C) 1996 - 2020
00,220 UPX 3.96 Markus Oberhumer, Laszlo Molnar & John Reiser Jan 23rd 2020
00,221
00,222 File size Ratio Format Name
00,223 -----
00,224 862048 -> 339908 39.43% linux/amd64 Nakamichi_DD-192.elf
00,225
00,226 Packed 1 file.
00,227 Ultimate Packer for eXecutables
00,228 Copyright (C) 1996 - 2020
00,229 UPX 3.96 Markus Oberhumer, Laszlo Molnar & John Reiser Jan 23rd 2020
00,230
00,231 File size Ratio Format Name
00,232 -----
00,233 862048 -> 340392 39.49% linux/amd64 Nakamichi_PRV-192.elf
00,234
00,235 Packed 1 file.
00,236 Ultimate Packer for eXecutables
00,237 Copyright (C) 1996 - 2020
00,238 UPX 3.96 Markus Oberhumer, Laszlo Molnar & John Reiser Jan 23rd 2020
00,239
00,240 File size Ratio Format Name
00,241 -----
00,242 862048 -> 340168 39.46% linux/amd64 Nakamichi_SHA3-192.elf
00,243
00,244 Packed 1 file.
00,245 Ultimate Packer for eXecutables
00,246 Copyright (C) 1996 - 2020
00,247 UPX 3.96 Markus Oberhumer, Laszlo Molnar & John Reiser Jan 23rd 2020
00,248
00,249 File size Ratio Format Name
00,250 -----
00,251 143872 -> 58880 40.93% win64/pe Nakamichi_DD-192.exe
00,252
00,253 Packed 1 file.
00,254 Ultimate Packer for eXecutables
```

```
00,255 Copyright (C) 1996 - 2020
00,256 UPX 3.96 Markus Oberhumer, Laszlo Molnar & John Reiser Jan 23rd 2020
00,257
00,258 File size Ratio Format Name
00,259 -----
00,260 144896 -> 59392 40.99% win64/pe Nakamichi_PRIV-192.exe
00,261
00,262 Packed 1 file.
00,263 Ultimate Packer for eXecutables
00,264 Copyright (C) 1996 - 2020
00,265 UPX 3.96 Markus Oberhumer, Laszlo Molnar & John Reiser Jan 23rd 2020
00,266
00,267 File size Ratio Format Name
00,268 -----
00,269 144384 -> 58880 40.78% win64/pe Nakamichi_SHA3-192.exe
00,270
00,271 Packed 1 file.
00,272 Ultimate Packer for eXecutables
00,273 Copyright (C) 1996 - 2020
00,274 UPX 3.96 Markus Oberhumer, Laszlo Molnar & John Reiser Jan 23rd 2020
00,275
00,276 File size Ratio Format Name
00,277 -----
00,278 853856 -> 336236 39.38% linux/amd64 Nakamichi_Vanilla_LITE_Kaidanji_GCC_64bit.elf
00,279
00,280 Packed 1 file.
00,281 Ultimate Packer for eXecutables
00,282 Copyright (C) 1996 - 2020
00,283 UPX 3.96 Markus Oberhumer, Laszlo Molnar & John Reiser Jan 23rd 2020
00,284
00,285 File size Ratio Format Name
00,286 -----
00,287 857952 -> 336916 39.27% linux/amd64 Nakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_GCC_64bit.elf
00,288
00,289 Packed 1 file.
00,290 Ultimate Packer for eXecutables
00,291 Copyright (C) 1996 - 2020
00,292 UPX 3.96 Markus Oberhumer, Laszlo Molnar & John Reiser Jan 23rd 2020
00,293
00,294 File size Ratio Format Name
00,295 -----
00,296 862048 -> 341028 39.56% linux/amd64 Nakamichi_Vanilla_LITE_Kaidanji_DD-128AES_GCC_64bit.elf
00,297
00,298 Packed 1 file.
00,299 Ultimate Packer for eXecutables
00,300 Copyright (C) 1996 - 2020
00,301 UPX 3.96 Markus Oberhumer, Laszlo Molnar & John Reiser Jan 23rd 2020
00,302
00,303 File size Ratio Format Name
00,304 -----
00,305 866144 -> 342528 39.55% linux/amd64 Nakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_DD-128AES_GCC_64bit.elf
00,306
00,307 Packed 1 file.
00,308 Ultimate Packer for eXecutables
00,309 Copyright (C) 1996 - 2020
00,310 UPX 3.96 Markus Oberhumer, Laszlo Molnar & John Reiser Jan 23rd 2020
00,311
00,312 File size Ratio Format Name
```

```

00,313 -----
00,314 138240 -> 55808 40.37% win64/pe Nakamichi_Vanilla_LITE_Kaidanji_GCC_64bit.exe
00,315
00,316 Packed 1 file.
00,317 Ultimate Packer for eXecutables
00,318 Copyright (C) 1996 - 2020
00,319 UPX 3.96 Markus Oberhumer, Laszlo Molnar & John Reiser Jan 23rd 2020
00,320
00,321 File size Ratio Format Name
00,322 -----
00,323 140288 -> 56832 40.51% win64/pe Nakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_GCC_64bit.exe
00,324
00,325 Packed 1 file.
00,326 Ultimate Packer for eXecutables
00,327 Copyright (C) 1996 - 2020
00,328 UPX 3.96 Markus Oberhumer, Laszlo Molnar & John Reiser Jan 23rd 2020
00,329
00,330 File size Ratio Format Name
00,331 -----
00,332 146432 -> 59904 40.91% win64/pe Nakamichi_Vanilla_LITE_Kaidanji_DD-128AES_GCC_64bit.exe
00,333
00,334 Packed 1 file.
00,335 Ultimate Packer for eXecutables
00,336 Copyright (C) 1996 - 2020
00,337 UPX 3.96 Markus Oberhumer, Laszlo Molnar & John Reiser Jan 23rd 2020
00,338
00,339 File size Ratio Format Name
00,340 -----
00,341 149504 -> 61440 41.10% win64/pe Nakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_DD-128AES_GCC_64bit.exe
00,342
00,343 Packed 1 file.
00,344 [kaze@kaze Kanshiketsu_AES_sourcecode]$ ls -l
00,345 -rwxrwxrwx. 2 kaze kaze 6435 Aug 29 00:21 _MakeELF_Nakamichi_GCC.sh
00,346 -rwxrwxrwx. 1 kaze kaze 339908 Aug 29 00:22 Nakamichi_DD-192.elf
00,347 -rwxrwxrwx. 1 kaze kaze 58880 Aug 29 00:22 Nakamichi_DD-192.exe
00,348 -rwxrwxrwx. 1 kaze kaze 340392 Aug 29 00:22 Nakamichi_PRV-192.elf
00,349 -rwxrwxrwx. 1 kaze kaze 59392 Aug 29 00:22 Nakamichi_PRV-192.exe
00,350 -rwxrwxrwx. 1 kaze kaze 340168 Aug 29 00:22 Nakamichi_SHA3-192.elf
00,351 -rwxrwxrwx. 1 kaze kaze 58880 Aug 29 00:22 Nakamichi_SHA3-192.exe
00,352 -rwxrwxrwx. 1 kaze kaze 341028 Aug 29 00:22 Nakamichi_Vanilla_LITE_Kaidanji_DD-128AES_GCC_64bit.elf
00,353 -rwxrwxrwx. 1 kaze kaze 59904 Aug 29 00:22 Nakamichi_Vanilla_LITE_Kaidanji_DD-128AES_GCC_64bit.exe
00,354 -rwxrwxrwx. 1 kaze kaze 336236 Aug 29 00:22 Nakamichi_Vanilla_LITE_Kaidanji_GCC_64bit.elf
00,355 -rwxrwxrwx. 1 kaze kaze 55808 Aug 29 00:22 Nakamichi_Vanilla_LITE_Kaidanji_GCC_64bit.exe
00,356 -rwxrwxrwx. 1 kaze kaze 342528 Aug 29 00:22 Nakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_DD-128AES_GCC_64bit.elf
00,357 -rwxrwxrwx. 1 kaze kaze 61440 Aug 29 00:22 Nakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_DD-128AES_GCC_64bit.exe
00,358 -rwxrwxrwx. 1 kaze kaze 336916 Aug 29 00:22 Nakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_GCC_64bit.elf
00,359 -rwxrwxrwx. 1 kaze kaze 56832 Aug 29 00:22 Nakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_GCC_64bit.exe
00,360 -rwxrwxrwx. 2 kaze kaze 3485 Aug 5 17:23 prvhash42.h
00,361
00,362 [kaze@kaze Kanshiketsu_AES_sourcecode]$ ./Nakamichi_Vanilla_LITE_Kaidanji-Kanshiketsu_DD-128AES_GCC_64bit.elf TERAPlG_Silesia_compression_corpus
TERAPIG_Silesia_compression_corpus.Nakamichi 20 26000 i
00,363
00,364 SMMi :MM2
00,365 0MMMM: rMMMMMa
00,366 ZMMM. 0Z :MMM7 7B
00,367 rMM0 MMM
00,368 MMM. MMM
00,369 BMMMa XZ :MM: MMM XMMX

```

```

00,370 XMSANMAYCE; rMMM rMM; MM@
00,371 WM0 ZMMMM ZMMM MMW BMM
00,372 : MMB aMMMM ZMM :MM8 MM7
00,373 : MMZ XMM MMa .WMMM0 @MM 8MMi
00,374 MMBa MM, aMM XMMB MM: rMMMi
00,375 @XMMM7 0MM MMS WMM MM2
00,376 MM MM, MM7 MM:
00,377 MMX ;MM imm MM8 MM5 MMZ iMBi :MMX @MMa WB: XMMZ2aX ZMM MM2 :M2
00,378 :MM ZMMM BM8 ;MMMMaaMM ,MM XMM0MM rMMMM2aMM XMMM SMMSMM. MMXMMM MMM0 aMMMa@MMr MMS ;MM;MM8 XMMM.
00,379 MMr MM0 MM 2MMW MM 0MM ZMS 2MM. aMMB MM BM aMM 8Ma MM imm BMM M7 MMZ ;MMX MMi ;MM ZMW MM0 rM 7MM
00,380 ;MM MM7 imm ZMM2 0MMa MMX 8M: MM ZMM5 BMM2 7 MM8 ZMr :MM rMB MM@ imm MMZ imm WMM BM2 MMX @MM
00,381 MMi ;MM 0MS 2MM7 aMM ;MM .8Mi MM aMM7 ZMM MM ;Mr XMM iM8 MMa 0MM BMM MM MM: BMX .MM MM7
00,382 aMM BMM MM ;MMX ZMMZ BM@ MM rMM7 ZMMZ :MM:MX 0MZ M8 XMM, MMX MM. ;W: rMM aMX ZMM 7MM
00,383 MM MMi MM MMZ 8M:MM MM; MM, MMa 0M:MM 8MM8 MMZM0 MM rMM 0MM WMa,M2 MMS MM8
00,384 @M0 :MMXMa MM WM XMB ; :MM;M8MM MM @M XM0 ; MM@ MMW MMrZi MMS MMi MMZM0 rMM Mi MM :
00,385 ;MM BMMM: rMM MM MMSMa ZMM@ immr 7MM MM MMXM2 :MM ZMM 8MM@ MMZ0 7MM :MMM MEMa 0MWBr
00,386 ,MMM MMM MMS MM MMZ MMM MM MMS .MM MMZ @MM. MM WMM0 0MM 2M ZMM :MMa MM7
00,387 immMMM@ XMM .MM XMM aMM MM: MMW .MM XM@ ZMM MM0 aMMX 8MM .MM WMM ZMr MMX MM8 0MM5
00,388 SMMMi MM, rMM,MM2 MM: SMM ;MM2 ;MM,MMS MM, MM MM MM. MM .MM MMZ MM XMM rMB
00,389 :MM7 MM@ BMM aMM ;MMX aMMX 8MM ZMM MM8 .MMX MM2 iMMX ,MMMMM2 .@; MM WMM
00,390 . : : 0MMZ .2Z :7r. ;i
00,391 0MMaSMi
00,392 aMMM7
00,393 .WW
00,394

```

00,395 Nakamichi 'Kaidanji', written by Kaze, inspired by Haruhiko Okumura sharing, based on Nobuo Ito's LZSS source, muffinesque tips by m^2, Jim Dempsey and Kirill Kryukov enforced.

00,396 This latest (2021-Aug-28) compile has B-trees only matchfinder - called 'SUPRALITE'; Downloadable at: <https://twitter.com/Sanmayce>

00,397

00,398 Nakamichi 'Kaidanji' SUPRALITE highlights:

00,399

00,400 - Back to supersimplistic - 'Kaidanji' returns refined. Simply, ripping only orders 4,6,8,10,12,14,16,18,36,64 and then quickly trying to expand the match in the found position.

00,401

00,402 Nakamichi 'Ryuugan-ditto-1TB'/'Dragoneye' LITE highlights:

00,403

00,404 - The Zennish LZSS Microdeduplicator, a texttoy (file-to-file [de]compressor) written in plain C, 100% FREE - licenseless it is;

00,405 - Single-threaded Non-SIMD console tool written entirely by yours truly Sanmayce;

00,406 - The first Lempel-Ziv-Storer-Szymanski (LZSS) compressor using 1TB sliding window;

00,407 - It targets superfast decompression of huge/tera corpora of textual data;

00,408

00,409 - LITE means no memmem() (Railgun) invocations are to be made, nah, back to minimalistic Railgun since 2021-Mar-12;

00,410 - Compressing at Linear Rate due to Matchfinding based on B-trees only;

00,411 - The Leprechaunesque (Internal/External) B-trees order 3 (2 keys MAX) are brutally-optimized;

00,412

00,413 - Ability to deduplicate (as little as) 64 bytes long chunks 1TB backwards;

00,414 - Ability to deduplicate 256[+] bytes long chunks 1TB backwards - when SHA3 enabled;

00,415

00,416 - CHUNKABLE! Since there are no headers/metadata in .Nakamichi files, arbitrary chunks/blocks can be compressed-n-concatenated;

00,417

00,418 - SCALABLE! Gets faster when more Physical or/and External RAM is available;

00,419 - Dynamical (as command line parameters) selection of HASH & B-trees memory blocks;

00,420 - AUTO-RESETTING PASSES (when TargetBuffer is filled then automatically restart building by doubling the passes);

00,421 - DEFRAGMENTED B-trees are in use during compression stage, their leaves contain only BBs appearing 2[+] times;

00,422

00,423 - Compileable under Windows and Linux, most suitable to run on 3TB machines;

00,424 - Many thanks go to: Haruhiko Okumura, Nobuo Ito, Hamid Buzidi, Landon Noll, J. Andrew Rogers, Aleksey Vaneev (<https://github.com/avaneev/prvhash>).

00,425

00,426 Sanmayce, Nakamichi's author, here, some more notes:

00,427

00,428 - The B-trees form the second layer, the first being HASH table handled by FNV1A-Pippip - the fastest lookup-hash;

00,429 - Hashpot a.k.a. hashpools (residing in Physical RAM) could be tuned via command line parameter, thus lessening the B-trees heights;

00,430 - The building of B-trees is done dynamically-n-automatically in PASSES, thus LOCALITY/LOCALIZATION leads to cache-friendliness, for example, instead of confusing/blinding

00,431 the SSD controller with building $2^{27} \approx 128\text{M}$ B-trees at a time, 'PASSES' revision lowers the "noise/mayhem" 128 times by processing e.g. (20bit) 1M B-trees at a time;

00,432

00,433 - For so long, the main goal was to throw it in battle, thus to forge one battle-ready *nix/windows tool with superb hash/search/decompress capabilities.

00,434 - Straight up, a wish of mine is to have the latest English Wikipedia XML dump decompressed by Nakamichi into physical RAM and then followed by Kazahana's superfast exact/wildcard/fuzzy etudes.

00,435 - Through the years, such a transparent decompressor (load booster) has been losing momentum and kinda became unnecessary, but not for me, when HDDs had 40MB/s sequential read speeds it was ...

00,436 boosty, now with SSDs reading at 1700+MB/s, not so smacky, yet uploading (on 128GB machines, 20GB+72GB needed) the whole Wikipedia ~20GB (instead of 72GB) is better, sizewise.

00,437

00,438 - Currently, having 16 (4,6,8,10,12,14,16,18,36,64,24,28,48,56,128,256) MatchLens being hashed (plus 1 being phantom/temporary), 29bit hashtable results in:

$(16+1)*8*(2^{29}) = (16+1)*8*(1<<29) =$

00,439 69632MB hashpot (the cluster of all hashpools), each slot houses a QWORD (being an offset/address of a B-tree root).

00,440 - In near future, wanna see a machine (256GB RAM) crunching the 77,265,813,300 bytes file with this command line (it requires 2x72GB + 68GB + 2930GB = 212GB physical RAM + 2.9TB SSD RAM):

00,441 D:\>timer64 Nakamichi.exe enwiki-20200420-pages-articles.xml enwiki-20200420-pages-articles.xml.Nakamichi 29 3000000 e

00,442 - In not so far future, wanna see a machine (4096GB RAM) crunching the 72GB file with this command line (it requires 2x72GB + 272GB + 2930GB = 3.3TB physical RAM):

00,443 D:\>timer64 Nakamichi.exe enwiki-20200420-pages-articles.xml enwiki-20200420-pages-articles.xml.Nakamichi 31 3000000 i

00,444

00,445 Q: - HOW TO BOOST [BUILDING/COMPRESSION] SPEED?

00,446 A: - Now, the name of the game is "RAM in spades", if we have a lot then speed is going to jump a lot. How?

00,447 - First, by increasing the hash-pool - 20 (number of bits) was used in 'Taishukan' resulting in $2^{20} \approx 1\text{M}$ hash-slots a.k.a. B-trees' roots per keysize, naturally, the bigger the faster since B-trees will

00,448 have less height and will be more - aiming to reach the ideal case - having only hash-slots with no actual B-trees! Back to 'Taishukan', the testfile is ~77MB, it suggests using 128M slots to lower

00,449 height/branching of B-trees - 2^{27} leads to using 27. Feartime, going from 20 to 27: $8*(15+1)*2^{(20)} = 134,217,793$ bytes; with the speed-up hash needs

$8*(15+1)*2^{(27)} = 17,179,869,184$ bytes, yes 16GB.

00,450 - Second, by lowering passes in building the B-trees - the trick is to increase the number in compile-time parameter -DSpeedUpBuilding=32 - (didn't make it as an execute-time parameter, lazy).

00,451 If we look at the 'Taishukan' console log it tells us that (Target-Buffer 106 MB) x 64 passes = 6784MB are needed for building the B-trees in ONE PASS ONLY, so compile with -DSpeedUpBuilding=6784 instead.

00,452

00,453 Enfun!

00,454

00,455 This compile uses B-trees only, no memmem() invocations - it compresses worse but much faster.

00,456 Allocating Source-Buffer 202 MB ...

00,457 Allocating Target-Buffer 1,202 MB ...

00,458 Source-file-Hash(FNV1A_YoshimitsuTRIAD) = 0x604d,d915

00,459 Leprechaun: Memory pool for B-trees is 26,000 MB.

00,460 Leprechaun: In this revision 8MB 10-way hash is used which results in 10 x 1,048,576 internal B-Trees of order 3.

00,461 Leprechaun: Allocating HASH memory 92,274,753 bytes = $8*(10+1)*2^{(20)}$... OK

00,462 Leprechaun: Allocating memory for B-trees 26001 MB ... OK

00,463 Kanshiketsu: Allocating Speed-up Array Source-Buffer_Skip-Vector 211,938,580 Bytes ...

00,464 Leprechaun: Size of input file: 211,938,580

00,465

00,466 Leprechaun: Using (first 16 bytes of) DoubleDeuceAES_Gumbotron_YMM for Matches 16+ long, in order to reduce memory footprint.

00,467 Leprechaun: Inserting keys/BBs of order 004 into B-trees, free RAM in B-tree pool is 00,026,000 MB; Pass #000,001 of 000,001 ... DONE; 00,001,034,062 DEFRAGMENTED B-trees have been rooted so far.

00,468 4,486,929 TotalNonUnique (of length 004) keys have been inserted into DEFRAGMENTED B-trees.

00,469 Kanshiketsu: HowManyPositionsAreNonUnique (for current order) = 197,146,478

00,470 Leprechaun: Inserting keys/BBs of order 006 into B-trees, free RAM in B-tree pool is 00,025,832 MB; Pass #000,001 of 000,001 ... DONE; 00,002,082,060 DEFRAGMENTED B-

```

trees have been rooted so far.
00,471      7,794,309 TotalNonUnique (of length 006) keys have been inserted into DEFRAGMENTED B-trees.
00,472 Kanshiketsu: HowManyPositionsAreNonUnique (for current order) = 178,071,522
00,473 Leprechaun: Inserting keys/BBs of order 008 into B-trees, free RAM in B-tree pool is 00,025,518 MB; Pass #000,001 of 000,001 ... DONE; 00,003,130,570 DEFRAGMENTED B-
trees have been rooted so far.
00,474      10,247,335 TotalNonUnique (of length 008) keys have been inserted into DEFRAGMENTED B-trees.
00,475 Kanshiketsu: HowManyPositionsAreNonUnique (for current order) = 160,211,062
00,476 Leprechaun: Inserting keys/BBs of order 010 into B-trees, free RAM in B-tree pool is 00,025,078 MB; Pass #000,001 of 000,001 ...
00,477 Leprechaun: Failure! Increment 'Memory for B-trees'!
00,478 Automatically increasing number of passes in order to fit B-trees into Target Buffer.
00,479 Leprechaun: Inserting keys/BBs of order 010 into B-trees, free RAM in B-tree pool is 00,024,799 MB; Pass #000,002 of 000,002 ... DONE; 00,004,179,134 DEFRAGMENTED B-
trees have been rooted so far.
00,480      12,114,174 TotalNonUnique (of length 010) keys have been inserted into DEFRAGMENTED B-trees.
00,481 Kanshiketsu: HowManyPositionsAreNonUnique (for current order) = 142,993,093
00,482 Leprechaun: Inserting keys/BBs of order 012 into B-trees, free RAM in B-tree pool is 00,024,225 MB; Pass #000,002 of 000,002 ... DONE; 00,005,227,690 DEFRAGMENTED B-
trees have been rooted so far.
00,483      12,081,422 TotalNonUnique (of length 012) keys have been inserted into DEFRAGMENTED B-trees.
00,484 Kanshiketsu: HowManyPositionsAreNonUnique (for current order) = 126,377,916
00,485 Leprechaun: Inserting keys/BBs of order 014 into B-trees, free RAM in B-tree pool is 00,023,641 MB; Pass #000,002 of 000,002 ... DONE; 00,006,276,244 DEFRAGMENTED B-
trees have been rooted so far.
00,486      11,112,573 TotalNonUnique (of length 014) keys have been inserted into DEFRAGMENTED B-trees.
00,487 Kanshiketsu: HowManyPositionsAreNonUnique (for current order) = 112,894,509
00,488 Leprechaun: Inserting keys/BBs of order 016 into B-trees, free RAM in B-tree pool is 00,023,077 MB; Pass #000,002 of 000,002 ... DONE; 00,007,324,748 DEFRAGMENTED B-
trees have been rooted so far.
00,489      10,103,874 TotalNonUnique (of length 016) keys have been inserted into DEFRAGMENTED B-trees.
00,490 Kanshiketsu: HowManyPositionsAreNonUnique (for current order) = 102,318,403
00,491 Leprechaun: Inserting keys/BBs of order 018 into B-trees, free RAM in B-tree pool is 00,022,546 MB; Pass #000,002 of 000,002 ... DONE; 00,008,373,194 DEFRAGMENTED B-
trees have been rooted so far.
00,492      9,404,275 TotalNonUnique (of length 018) keys have been inserted into DEFRAGMENTED B-trees.
00,493 Kanshiketsu: HowManyPositionsAreNonUnique (for current order) = 94,981,918
00,494 Leprechaun: Inserting keys/BBs of order 036 into B-trees, free RAM in B-tree pool is 00,022,098 MB; Pass #000,002 of 000,002 ... DONE; 00,009,420,139 DEFRAGMENTED B-
trees have been rooted so far.
00,495      6,766,031 TotalNonUnique (of length 036) keys have been inserted into DEFRAGMENTED B-trees.
00,496 Kanshiketsu: HowManyPositionsAreNonUnique (for current order) = 60,506,490
00,497 Leprechaun: Inserting keys/BBs of order 064 into B-trees, free RAM in B-tree pool is 00,021,732 MB; Pass #000,002 of 000,002 ... DONE; 00,010,466,093 DEFRAGMENTED B-
trees have been rooted so far.
00,498      6,277,787 TotalNonUnique (of length 064) keys have been inserted into DEFRAGMENTED B-trees.
00,499 Kanshiketsu: HowManyPositionsAreNonUnique (for current order) = 42,010,124
00,500
00,501 Histogram (rotate it 90 degrees clockwise) of Unique-Building-Blocks appearing 2[+] times (e.g. 'alfalfa-alfa' stream has 1 BB with length 4 appearing 3 times thus
adding 1 to the count for {004}):
00,502 6,277,787{064}
00,503 6,766,031{036}
00,504 9,404,275{018}
00,505 10,103,874{016}
00,506 11,112,573{014}
00,507 12,081,422{012}
00,508 12,114,174{010}
00,509 10,247,335{008}
00,510 7,794,309{006}
00,511 4,486,929{004}
00,512
00,513 Leprechaun: Total Searches-n-Inserts Per Second: 2,096,388 SNIPS = 2,589,040,181 keys in 1,235 seconds
00,514 Leprechaun: RAM needed to house B-trees (relative to the file being ripped): 21N = 4,443MB
00,515 Leprechaun: RAM needed to build B-trees IN ONE PASS: (Target-Buffer = 1,202 MB) x 2 passes = 2,404MB
00,516 Note: In case of RAM in spades then may use compile option: -DSpeedUpBuilding=2404
00,517
00,518 Compressing 211,938,580 bytes ...

```

[illegible]

```
00,666 */
00,667
00,668 // This is 'Non-Unique_Only_Keys_Are_Inserted_in_Btrees' revision, 2019-Sep-28:
00,669 // TO-DO Benefits:
00,670 // - Smaller footprint, much smaller!
00,671 // - Faster key searches due to smaller height of B-trees, thus compression bec
00,672 // - Wearing of the SSD is not as atrocious as before.
00,673
00,674 // This is 'PASSES' revision of main 'btree' revision, command-line-wise it dif
00,675 // Single-pass revision:
00,676 // D:\>Nakamichi %1 %1.Nakamichi 25 420000 e
00,677 // Multi-pass revision a.k.a. 'PASSES':
```

```
00,678 // D:\>Nakamichi %1 %1.Nakamichi 25 420000 E
00,679 // The upper case means 2^25 =~ 32M slots are to be divided in 128 passes, it creates B-trees into cache-friendly manner, DONE!
00,680 // TO-DO! Its purpose is to build all the Btrees in RAM without trashing the SSD with random reads/writes, when a pass is complete then writing in a single 'fwrite'
happens, thus concatenating Btrees sub-pools to the master pool. TO-DO!
00,681 // In order to avoid changing command line options, a few workarounds are to be enforced:
00,682 // - using the decompression buffer for building a given pass of the Btrees
00,683 // - Number of passes is automatically set to 2^7=128 (there is strong reason for that - to fit into N size i.e. filesize: 128N > 98N, because if all slots point to a
unique leaf (98 bytes-per-leaf)):
00,684 // HashChunkSizeInBITS_GLOBAL = HashInBITS_GLOBAL - 7;
00,685
00,686 //TO-DO1: Reduce (return actually) the keysize to 1:1, not as present where HEX doubles the keysize... DONE! 2019-Feb-07
00,687 //TO-DO2: Replace all 'if (BStorBtree == 2) {' with '#if defined(ExternalRAM)' '#else' '#endif'
00,688
00,689 // In 2019-Feb-10 revision, SHA3-224 is in use.
00,690
00,691 // In 2019-Feb-07 revision, the size of keys in the LEAF is back to 1:1 - instead of ending with ASCII 0, the first (i.e. zero) byte of the key houses the length -
1..255, pretty enough since 28 bytes will be the MAX - the SHA3-224.
00,692
00,693 // In 2019-Jan-20 revision, the allocation bug in SSD (z/Z) mode was fixed by adding Internal/External if-else check.
00,694 // Also, for benchmarking SSDs via USB, there is a tool reporting the mode:
00,695 /*
00,696 If the device is connected in USB 3.0 SuperSpeed mode, it will show:
00,697 Device Bus Speed      : 0x03 (SuperSpeed)
00,698
00,699 For USB 2.0, it will show:
00,700 Device Bus Speed      : 0x02 (High-Speed)
00,701
00,702 For USB 1.1, it will show:
00,703 Device Bus Speed      : 0x01 (Full-Speed)
00,704
00,705 https://www.uwe-sieber.de/usbtreetview_e.html
00,706 */
00,707
00,708 // Ryuugan-ditto-1TB features 1TB as well.
00,709 // Ryuugan+ features 128GB as well.
00,710 // Washigan+ has been tuned for 2/3 bytes windows...
00,711 // Nakamichi_Washigan is strongest so far and by far - it is just as Okamigan but uses the reserved value 9.
00,712 // Okamigan is strongest so far and by far.
00,713 // Okami is stronger than Okamiko thanks to the additional 6:3 (64KB window), plus, 14:4 (16MB window) instead of 48:3/24:3/12:3/6:3 (64KB window).
00,714 // Okamiko is stronger than Zato thanks to the additional 48:3/24:3/12:3/6:3 (64KB window).
00,715
00,716 // 狼 Okami, the next...
00,717 // Nakamichi 'Zato' is successor to 'Tsubame' which is to 'Tengu-Tsuyo' which is to 'Tengu'.
00,718 // 座頭 - Zato
00,719 // The character's name is actually Ichi. Zatō is a title, the lowest of the four official ranks within the Tōdōza, the historical guild for blind men. (Thus zato also
designates a blind person in Japanese slang.) Ichi is therefore properly called Zatō-no-Ichi ("Low-Ranking Blind Person Ichi", approximately), or Zatōichi for short. Massage
was a traditional occupation for the blind (as their lack of sight removed the issue of gender), as was playing the biwa or, for blind women (goze), the shamisen. Being
lesser Hinin (lit. "non-people"), blind people and masseurs were regarded as among the very lowest of the low in social class, other than Eta or outright criminals; they were
generally considered wretches, beneath notice, no better than beggars or even the insane - especially during the Edo period - and it was also commonly thought that the blind
were accursed, despicable, severely mentally disabled, deaf and sexually dangerous.
00,720 // Source: https://en.wikipedia.org/wiki/Zatoichi
00,721
00,722 // Compilation:
00,723 /*
00,724 Intel(R) Parallel Studio XE 2015
00,725 Copyright (C) 1985-2014 Intel Corporation. All rights reserved.
00,726
```



```

00,727 Intel(R) MPI Library 5.0 Update 1 Build Environment for Intel(R) 64 applications
00,728 Copyright (C) 2007-2014 Intel Corporation. All rights reserved.
00,729
00,730
00,731 Intel(R) Trace Analyzer and Collector 9.0 Update 1 for Windows* OS for Intel(R) 64 applications
00,732 Copyright (C) 1996-2014 Intel Corporation. All rights reserved.
00,733
00,734 Setting environment for using Microsoft Visual Studio 2010 x64 cross tools.
00,735
00,736 C:\Program Files (x86)\Intel>cd "Composer XE 2015"
00,737
00,738 C:\Program Files (x86)\Intel\Composer XE 2015>cd bin
00,739
00,740 C:\Program Files (x86)\Intel\Composer XE 2015\bin>iclvars.bat
00,741 Syntax:
00,742 iclvars.bat <arch> [vs]
00,743
00,744 <arch> must be is one of the following
00,745 ia32 : Set up for IA-32 host and target
00,746 ia32_intel64 : Set up for IA-32 host and Intel(R)64 target
00,747 intel64 : Set up for Intel(R) 64 host and target
00,748 If specified, <vs> must be one of the following
00,749 vs2010 : Set to use Microsoft Visual Studio 2010
00,750 vs2010shell : Set to use Microsoft Visual Studio Shell 2010
00,751 vs2012 : Set to use Microsoft Visual Studio 2012
00,752 vs2012shell : Set to use Microsoft Visual Studio Shell 2012
00,753 vs2013 : Set to use Microsoft Visual Studio 2013
00,754 If <vs> is not specified, the version of Visual Studio detected at install
00,755 time is used.
00,756
00,757
00,758 C:\Program Files (x86)\Intel\Composer XE 2015\bin>iclvars.bat intel64
00,759 Intel(R) Parallel Studio XE 2015
00,760 Copyright (C) 1985-2014 Intel Corporation. All rights reserved.
00,761 Intel(R) Parallel Studio XE 2015 Composer Edition (package 108)
00,762 Setting environment for using Microsoft Visual Studio 2010 x64 tools.
00,763
00,764
00,765 C:\Program Files (x86)\Intel\Composer XE 2015\bin>d:
00,766
00,767 D:\>cd D:\TEXTUAL_MADNESS_old\Nakamichi_Okamigan\Nakamichi_Okamigan_(source-executable-booklet)
00,768
00,769 D:\TEXTUAL_MADNESS_old\Nakamichi_Okamigan\Nakamichi_Okamigan_(source-executable-booklet)>dir
00,770 Volume in drive D is S640_Vol5
00,771 Volume Serial Number is 5861-9E6C
00,772
00,773 Directory of D:\TEXTUAL_MADNESS_old\Nakamichi_Okamigan\Nakamichi_Okamigan_(source-executable-booklet)
00,774
00,775 12/02/2016 01:25 PM <DIR> .
00,776 12/02/2016 01:25 PM <DIR> ..
00,777 11/28/2016 01:34 PM 4,353,536 Booklet_Okamigan.doc
00,778 11/28/2016 01:35 PM 2,766,041 Booklet_Okamigan.pdf
00,779 10/25/2016 01:08 AM 55,397 lzsse2.cpp
00,780 10/25/2016 01:08 AM 4,858 lzsse2.h
00,781 10/25/2016 01:08 AM 2,481 lzsse2_platform.h
00,782 11/14/2016 08:52 PM 1,357 MakeEXEs_Okamigan.bat
00,783 10/25/2016 01:08 AM 1,632 MokujiN BLUE 224 prompt.lnk
00,784 10/25/2016 01:08 AM 1,632 MokujiN CYAN 224 prompt.lnk

```

```
00,785 10/25/2016 01:08 AM          1,632 MokuJIN DARK 224 prompt.lnk
00,786 10/25/2016 01:08 AM          1,632 MokuJIN GREEN 224 prompt.lnk
00,787 10/25/2016 01:08 AM          1,632 MokuJIN ORANGE 224 prompt.lnk
00,788 12/02/2016 01:23 PM          417,892 Nakamichi_Okamigan.c
00,789 11/28/2016 06:55 PM        10,245,120 Okamigan_TurboBench.doc
00,790 11/28/2016 06:56 PM          5,119,192 Okamigan_TurboBench.pdf
00,791          14 File(s)        22,974,034 bytes
00,792          2 Dir(s)    30,903,410,688 bytes free
00,793
00,794 D:\TEXTUAL_MADNESS_old\Nakamichi_Okamigan\Nakamichi_Okamigan_(source-executable-booklet)>MakeEXEs_Okamigan.bat
00,795
00,796 D:\TEXTUAL_MADNESS_old\Nakamichi_Okamigan\Nakamichi_Okamigan_(source-executable-booklet)>icl /TP /O3 /QxSSE4.1 Nakamichi_Okamigan.c lzsse2.cpp -D_N_XMM -
D_N_prefetch_4096 -D_N_HIGH_PRIORITY /FAcs
00,797 Intel(R) C++ Intel(R) 64 Compiler XE for applications running on Intel(R) 64, Version 15.0.0.108 Build 20140726
00,798 Copyright (C) 1985-2014 Intel Corporation. All rights reserved.
00,799
00,800 Nakamichi_Okamigan.c
00,801 lzsse2.cpp
00,802 Microsoft (R) Incremental Linker Version 10.00.30319.01
00,803 Copyright (C) Microsoft Corporation. All rights reserved.
00,804
00,805 -out:Nakamichi_Okamigan.exe
00,806 Nakamichi_Okamigan.obj
00,807 lzsse2.obj
00,808
00,809 D:\TEXTUAL_MADNESS_old\Nakamichi_Okamigan\Nakamichi_Okamigan_(source-executable-booklet)>icl /TP /O3 /QxAVX Nakamichi_Okamigan.c lzsse2.cpp -D_N_XMM -D_N_prefetch_4096
-D_N_HIGH_PRIORITY /FAcs
00,810 Intel(R) C++ Intel(R) 64 Compiler XE for applications running on Intel(R) 64, Version 15.0.0.108 Build 20140726
00,811 Copyright (C) 1985-2014 Intel Corporation. All rights reserved.
00,812
00,813 Nakamichi_Okamigan.c
00,814 lzsse2.cpp
00,815 Microsoft (R) Incremental Linker Version 10.00.30319.01
00,816 Copyright (C) Microsoft Corporation. All rights reserved.
00,817
00,818 -out:Nakamichi_Okamigan.exe
00,819 Nakamichi_Okamigan.obj
00,820 lzsse2.obj
00,821 */
00,822
00,823 // 2016-Apr-04: Grrr... Stupid bug (due to overlooking) was crushed in 'TT', namely:
00,824 /*
00,825 //define Min_Match_Length (32)
00,826 #define Min_Match_Length (4)
00,827 */
00,828 // becomes:
00,829 /*
00,830 //define Min_Match_Length (32)
00,831 //define Min_Match_Length (4)
00,832 #define Min_Match_Length (16) // Maximum MatchLength is 16, it decides the size of Look-ahead buffer - to avoid search beyound end. This needs more attention in the
future - to clarify it fully. Overlapping is also yet to come.
00,833 */
00,834
00,835 // 2016-Apr-03: Finished the unfinished 2016-Mar-31 draft. Stupid stats bug was fixed TargetSize -> VerifySize in line: k = (((float)1000*VerifySize/(clocks2 - clocks1
+ 1))); k=k>>20; k=k*Trials;
00,836 // 2016-Mar-31: added the FASTEST decompressor - LZSSE2 - to juxtapose two XMM vs XMM etudes, below the compile lines:
00,837 /*
00,838 D:\The_Usual_Suspects\Nakamichi_(Tengu-Tsuyo)_1MB-Sliding-Window_vs_LZSSE2>MakeEXEs_Tengu-Tsuyo.bat
```

```
00,839
00,840 D:\_The_Usual_Suspects\Nakamichi_(Tengu-Tsuyo)_1MB-Sliding-Window_vs_LZSSE2>icl /TP /O3 /QxSSE4.1 Nakamichi_Tengu-Tsuyo.c lzsse2.cpp -D_N_XMM -D_N_prefetch_4096 -
D_N_HIGH_PRIORITY /FAcs
00,841 Intel(R) C++ Intel(R) 64 Compiler XE for applications running on Intel(R) 64, Version 15.0.0.108 Build 20140726
00,842 Copyright (C) 1985-2014 Intel Corporation. All rights reserved.
00,843
00,844 Nakamichi_Tengu-Tsuyo.c
00,845 lzsse2.cpp
00,846 Microsoft (R) Incremental Linker Version 10.00.30319.01
00,847 Copyright (C) Microsoft Corporation. All rights reserved.
00,848
00,849 -out:Nakamichi_Tengu-Tsuyo.exe
00,850 Nakamichi_Tengu-Tsuyo.obj
00,851 lzsse2.obj
00,852
00,853 D:\_The_Usual_Suspects\Nakamichi_(Tengu-Tsuyo)_1MB-Sliding-Window_vs_LZSSE2>icl /TP /O3 /QxAVX Nakamichi_Tengu-Tsuyo.c lzsse2.cpp -D_N_XMM -D_N_prefetch_4096 -
D_N_HIGH_PRIORITY /FAcs
00,854 Intel(R) C++ Intel(R) 64 Compiler XE for applications running on Intel(R) 64, Version 15.0.0.108 Build 20140726
00,855 Copyright (C) 1985-2014 Intel Corporation. All rights reserved.
00,856
00,857 Nakamichi_Tengu-Tsuyo.c
00,858 lzsse2.cpp
00,859 Microsoft (R) Incremental Linker Version 10.00.30319.01
00,860 Copyright (C) Microsoft Corporation. All rights reserved.
00,861
00,862 -out:Nakamichi_Tengu-Tsuyo.exe
00,863 Nakamichi_Tengu-Tsuyo.obj
00,864 lzsse2.obj
00,865
00,866 D:\_The_Usual_Suspects\Nakamichi_(Tengu-Tsuyo)_1MB-Sliding-Window_vs_LZSSE2>dir Nakamichi_Tengu-Tsuyo*
00,867 Volume in drive D is S640_Vol5
00,868 Volume Serial Number is 5861-9E6C
00,869
00,870 Directory of D:\_The_Usual_Suspects\Nakamichi_(Tengu-Tsuyo)_1MB-Sliding-Window_vs_LZSSE2
00,871
00,872 03/31/2016 12:50 AM 107,458 Nakamichi_Tengu-Tsuyo.c
00,873 03/27/2016 11:04 PM 1,234,944 Nakamichi_Tengu-Tsuyo.doc
00,874 03/31/2016 12:57 AM 948,131 Nakamichi_Tengu-Tsuyo_XMM_PREFETCH_4096_Intel_15.0_64bit_AVX.cod
00,875 03/31/2016 12:57 AM 139,264 Nakamichi_Tengu-Tsuyo_XMM_PREFETCH_4096_Intel_15.0_64bit_AVX.exe
00,876 03/31/2016 12:57 AM 943,556 Nakamichi_Tengu-Tsuyo_XMM_PREFETCH_4096_Intel_15.0_64bit_SSE41.cod
00,877 03/31/2016 12:57 AM 146,432 Nakamichi_Tengu-Tsuyo_XMM_PREFETCH_4096_Intel_15.0_64bit_SSE41.exe
00,878 6 File(s) 3,519,785 bytes
00,879 0 Dir(s) 8,169,893,888 bytes free
00,880
00,881 D:\_The_Usual_Suspects\Nakamichi_(Tengu-Tsuyo)_1MB-Sliding-Window_vs_LZSSE2>dir lzs*
00,882 Volume in drive D is S640_Vol5
00,883 Volume Serial Number is 5861-9E6C
00,884
00,885 Directory of D:\_The_Usual_Suspects\Nakamichi_(Tengu-Tsuyo)_1MB-Sliding-Window_vs_LZSSE2
00,886
00,887 03/30/2016 10:05 PM 55,397 lzsse2.cpp
00,888 03/30/2016 10:17 PM 4,858 lzsse2.h
00,889 03/31/2016 12:57 AM 690,697 lzsse2_64bit_AVX.cod
00,890 03/31/2016 12:57 AM 734,033 lzsse2_64bit_SSE41.cod
00,891 03/27/2016 07:49 AM 2,481 lzsse2_platform.h
00,892 5 File(s) 1,487,466 bytes
00,893 0 Dir(s) 8,169,893,888 bytes free
00,894
```

```
00,895 D:\_The_Usual_Suspects\Nakamichi_(Tengu-Tsuyo)_1MB-Sliding-Window_vs_LZSSE2>
00,896 */
00,897
00,898 // 2016-Mar-19: Tiny improvement in encoding (just forgot to refresh two define's), namely:
00,899 /*
00,900 // #define Min_Match_Length (32)
00,901 // #define Min_Match_Length (4)
00,902 // #define OffsetBITS (32-3)
00,903 // #define OffsetBITS (29)
00,904 */
00,905
00,906 // How to compile:
00,907 // icl /O3 /QxSSE2 Nakamichi_Tengu-Tsuyo.c -D_N_XMM -D_N_prefetch_4096 -D_N_HIGH_PRIORITY /FAcs
00,908
00,909 // Nakamichi_Tengu-Tsuyo.c, using 16B/4KB/1MB or (8-4)bit/(16-4)bit/(24-4)bit windows with 1/2/3 bytes long offsets.
00,910 // The variant 'Tsuyo' tries next position and if it gives higher ratio it is used.
00,911
00,912 // 中道 nakamichi [noun]
00,913 // English Meaning(s) for 中道
00,914 // 1. road through the middle; middle road
00,915 // Meanings for each kanji in 中道
00,916 // 中 in; inside; middle; mean; center
00,917 // 道 road-way; street; district; journey; course; moral; teachings
00,918
00,919 // 天狗 tengu
00,920 // heavenly dog
00,921
00,922 // 強 - Tsuyo
00,923 // https://glosbe.com/ja/en/%E5%BC%B7
00,924 // Translations into English:
00,925 // JMdict:
00,926 // a little more than
00,927 // JMdict:
00,928 // a little over
00,929 // JMdict:
00,930 // military build-up
00,931 // JMdict:
00,932 // one of the biggest
00,933 // JMdict:
00,934 // one of the most powerful
00,935 // JMdict:
00,936 // powerhouse
00,937
00,938 // The first Nakamichi was 'Kaidanji', based on Nobuo Ito's source, thanks Ito.
00,939 // The main goal of Nakamichi is to allow supersimple and superfast decoding for English x-grams (mainly) in pure C, or not, heh-heh.
00,940 // Natively Nakamichi is targeted as 64bit tool with 16 threads, helping Kazahana to traverse faster when I/O is not superior.
00,941 // In short, Nakamichi is intended as x-gram decompressor.
00,942
00,943 // Eightfold Path ~ the Buddhist path to nirvana, comprising eight aspects in which an aspirant must become practised;
00,944 // eightfold way ~ (Physics), the grouping of hadrons into supermultiplets by means of SU(3)); (b) adverb to eight times the number or quantity: OE.
00,945
00,946 void x64toaKAZE ( /* stdcall is faster and smaller... Might as well use it for the helper. */
00,947 unsigned long long val,
00,948 char *buf,
00,949 unsigned radix,
00,950 int is_neg
```

```
00,951     )
00,952 {
00,953     char *p;                /* pointer to traverse string */
00,954     char *firstdig;         /* pointer to first digit */
00,955     char temp;              /* temp char */
00,956     unsigned digval;        /* value of digit */
00,957
00,958     p = buf;
00,959
00,960     if ( is_neg )
00,961     {
00,962         *p++ = '-';         /* negative, so output '-' and negate */
00,963         val = (unsigned long long)(-(long long)val);
00,964     }
00,965
00,966     firstdig = p;           /* save pointer to first digit */
00,967
00,968     do {
00,969         digval = (unsigned) (val % radix);
00,970         val /= radix;       /* get next digit */
00,971
00,972         /* convert to ascii and store */
00,973         if (digval > 9)
00,974             *p++ = (char) (digval - 10 + 'a'); /* a letter */
00,975         else
00,976             *p++ = (char) (digval + '0');      /* a digit */
00,977     } while (val > 0);
00,978
00,979     /* We now have the digit of the number in the buffer, but in reverse
00,980     order. Thus we reverse them now. */
00,981
00,982     *p-- = '\0';           /* terminate string; p points to last digit */
00,983
00,984     do {
00,985         temp = *p;
00,986         *p = *firstdig;
00,987         *firstdig = temp;  /* swap *p and *firstdig */
00,988         --p;
00,989         ++firstdig;        /* advance to next two digits */
00,990     } while (firstdig < p); /* repeat until halfway */
00,991 }
00,992
00,993 /* Actual functions just call conversion helper with neg flag set correctly,
00,994 and return pointer to buffer. */
00,995
00,996 char * _i64toaKAZE (
00,997     long long val,
00,998     char *buf,
00,999     int radix
01,000 )
01,001 {
01,002     x64toaKAZE((unsigned long long)val, buf, radix, (radix == 10 && val < 0));
01,003     return buf;
01,004 }
01,005
01,006 char * _ui64toaKAZE (
01,007     unsigned long long val,
01,008     char *buf,
```



```
01,009         int radix
01,010         )
01,011 {
01,012         x64toaKAZE(val, buf, radix, 0);
01,013         return buf;
01,014 }
01,015
01,016 char * _ui64toaKAZEzerocomma (
01,017     unsigned long long val,
01,018     char *buf,
01,019     int radix
01,020 )
01,021 {
01,022     char *p;
01,023     char temp;
01,024     int txpman;
01,025     int pxnman;
01,026     x64toaKAZE(val, buf, radix, 0);
01,027     p = buf;
01,028     do {
01,029     } while (*++p != '\0');
01,030     p--; // p points to last digit
01,031     // buf points to first digit
01,032     buf[26] = 0;
01,033     txpman = 1;
01,034     pxnman = 0;
01,035     do
01,036     { if (buf <= p)
01,037     { temp = *p;
01,038       buf[26-txpman] = temp; pxnman++;
01,039       p--;
01,040       if (pxnman % 3 == 0)
01,041       { txpman++;
01,042         buf[26-txpman] = (char) (',');
01,043       }
01,044     }
01,045     else
01,046     { buf[26-txpman] = (char) ('\0'); pxnman++;
01,047       if (pxnman % 3 == 0)
01,048       { txpman++;
01,049         buf[26-txpman] = (char) (',');
01,050       }
01,051     }
01,052     txpman++;
01,053     } while (txpman <= 26);
01,054     return buf;
01,055 }
01,056
01,057 char * _ui64toaKAZEcomma (
01,058     unsigned long long val,
01,059     char *buf,
01,060     int radix
01,061 )
01,062 {
01,063     char *p;
01,064     char temp;
01,065     int txpman;
01,066     int pxnman;
```

```

01,067         x64toaKAZE(val, buf, radix, 0);
01,068             p = buf;
01,069             do {
01,070                 } while (*++p != '\0');
01,071             p--; // p points to last digit
01,072             // buf points to first digit
01,073             buf[26] = 0;
01,074             txpman = 1;
01,075             pxnman = 0;
01,076             while (buf <= p)
01,077             { temp = *p;
01,078               buf[26-txpman] = temp; pxnman++;
01,079               p--;
01,080               if (pxnman % 3 == 0 && buf <= p)
01,081               { txpman++;
01,082                 buf[26-txpman] = (char) (' ','');
01,083               }
01,084               txpman++;
01,085             }
01,086             return buf+26-(txpman-1);
01,087 }
01,088
01,089 char * _ui64toaKAZEzerocomma4 (
01,090     unsigned long long val,
01,091     char *buf,
01,092     int radix
01,093 )
01,094 {
01,095     char *p;
01,096     char temp;
01,097     int txpman;
01,098     int pxnman;
01,099     x64toaKAZE(val, buf, radix, 0);
01,100     p = buf;
01,101     do {
01,102         } while (*++p != '\0');
01,103     p--; // p points to last digit
01,104     // buf points to first digit
01,105     buf[26] = 0;
01,106     txpman = 1;
01,107     pxnman = 0;
01,108     do
01,109     { if (buf <= p)
01,110       { temp = *p;
01,111         buf[26-txpman] = temp; pxnman++;
01,112         p--;
01,113         if (pxnman % 4 == 0)
01,114         { txpman++;
01,115           buf[26-txpman] = (char) (' ','');
01,116         }
01,117       }
01,118     else
01,119     { buf[26-txpman] = (char) ('\0'); pxnman++;
01,120       if (pxnman % 4 == 0)
01,121       { txpman++;
01,122         buf[26-txpman] = (char) (' ','');
01,123       }
01,124     }

```

```

01,125         txpman++;
01,126         } while (txpman <= 26);
01,127     return buf;
01,128 }
01,129
01,130 /* minimum signed 64 bit value */
01,131 #define _I64_MIN    (-9223372036854775807i64 - 1)
01,132 /* maximum signed 64 bit value */
01,133 #define _I64_MAX    9223372036854775807i64
01,134 /* maximum unsigned 64 bit value */
01,135 #define _UI64_MAX    0xffffffffffffffffui64
01,136
01,137 /* minimum signed 128 bit value */
01,138 #define _I128_MIN    (-170141183460469231731687303715884105727i128 - 1)
01,139 /* maximum signed 128 bit value */
01,140 #define _I128_MAX    170141183460469231731687303715884105727i128
01,141 /* maximum unsigned 128 bit value */
01,142 #define _UI128_MAX    0xffffffffffffffffffffffffffffffffui128
01,143
01,144     char llToaDigits[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
01,145     // below duplicates are needed because of one_line_invoking need different buffers.
01,146     char llToaDigits2[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
01,147     char llToaDigits3[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
01,148     char llToaDigits4[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
01,149     char llToaDigits5[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
01,150
01,151 // During compilation use one of these, the granularity of the padded 'memcpy', 4x2x8/2x2x16/1x2x32/1x1x64 respectively as GP/XMM/YMM/ZMM, the maximum literal length
reduced from 127 to 63:
01,152 // #define _N_GP
01,153 // #define _N_XMM
01,154 // #define _N_YMM
01,155 // #define _N_ZMM
01,156
01,157 // #define _N_prefetch_64
01,158 // #define _N_prefetch_128
01,159 // #define _N_prefetch_4096
01,160
01,161 // Only one must be uncommented:
01,162 // #define _WIN32_ENVIRONMENT_
01,163 // #define _POSIX_ENVIRONMENT_
01,164
01,165 #include <stdio.h>
01,166 #include <stdlib.h>
01,167 #include <stdint.h> // uint64_t needed
01,168 #include <time.h>
01,169 #include <string.h>
01,170
01,171     clock_t clocks0, clocks1, clocks2;
01,172     time_t time1, time2;
01,173
01,174 #if defined(_WIN32_ENVIRONMENT_)
01,175 #include <io.h> // needed for Windows' 'lseeki64' and 'telli64'
01,176 // Above line must be commented in order to compile with Intel C compiler: an error "can't find io.h" occurs.
01,177 #include <fcntl.h> // 2020-Jan-13
01,178 #else
01,179 #endif /* defined(_WIN32_ENVIRONMENT_) */
01,180
01,181 #ifdef _N_XMM

```

```
01,182 #include <emmintrin.h> // SSE2 intrinsics
01,183 #include <smmmintrin.h> // SSE4.1 intrinsics
01,184 #endif
01,185 #ifdef _N_YMM
01,186 #include <emmintrin.h> // SSE2 intrinsics
01,187 #include <smmmintrin.h> // SSE4.1 intrinsics
01,188 #include <immintrin.h> // AVX intrinsics
01,189 #endif
01,190 #ifdef _N_ZMM
01,191 #include <emmintrin.h> // SSE2 intrinsics
01,192 #include <smmmintrin.h> // SSE4.1 intrinsics
01,193 #include <immintrin.h> // AVX intrinsics
01,194 #include <zmmmintrin.h> // AVX2 intrinsics, definitions and declarations for use with 512-bit compiler intrinsics.
01,195 #endif
01,196
01,197 #ifdef _N_XMM
01,198 void SlowCopy128bit (const char *SOURCE, char *TARGET) { _mm_storeu_si128((__m128i *) (TARGET), _mm_loadu_si128((const __m128i *) (SOURCE))); }
01,199 void NotSoSlowCopy128bit (const char *SOURCE, char *TARGET) { _mm_storeu_si128((__m128i *) (TARGET), _mm_lddqu_si128((const __m128i *) (SOURCE))); }
01,200 #endif
01,201 #ifdef _N_YMM
01,202 void SlowCopy128bit (const char *SOURCE, char *TARGET) { _mm_storeu_si128((__m128i *) (TARGET), _mm_loadu_si128((const __m128i *) (SOURCE))); }
01,203 void NotSoSlowCopy128bit (const char *SOURCE, char *TARGET) { _mm_storeu_si128((__m128i *) (TARGET), _mm_lddqu_si128((const __m128i *) (SOURCE))); }
01,204 #endif
01,205 #ifdef _N_ZMM
01,206 void SlowCopy128bit (const char *SOURCE, char *TARGET) { _mm_storeu_si128((__m128i *) (TARGET), _mm_loadu_si128((const __m128i *) (SOURCE))); }
01,207 #endif
01,208 /*
01,209  * Move Unaligned Packed Integer Values
01,210  * **** VMOVDQU ymm1, m256
01,211  * **** VMOVDQU m256, ymm1
01,212  * Moves 256 bits of packed integer values from the source operand to the
01,213  * destination
01,214  */
01,215 //extern __m256i __ICL_INTRINCC _mm256_loadu_si256(__m256i const *);
01,216 //extern void __ICL_INTRINCC _mm256_storeu_si256(__m256i *, __m256i);
01,217 #ifdef _N_YMM
01,218 void SlowCopy256bit (const char *SOURCE, char *TARGET) { _mm256_storeu_si256((__m256i *) (TARGET), _mm256_loadu_si256((const __m256i *) (SOURCE))); }
01,219 #endif
01,220 //extern __m512i __ICL_INTRINCC _mm512_loadu_si512(void const*);
01,221 //extern void __ICL_INTRINCC _mm512_storeu_si512(void*, __m512i);
01,222 #ifdef _N_ZMM
01,223 void SlowCopy512bit (const char *SOURCE, char *TARGET) { _mm512_storeu_si512((__m512i *) (TARGET), _mm512_loadu_si512((const __m512i *) (SOURCE))); }
01,224 #endif
01,225
01,226 // SearchIn... uses it:
01,227 void FIPS202_SHA3_224(const unsigned char *input, unsigned int inputByteLen, unsigned char *output);
01,228
01,229 #ifndef NULL
01,230 #define NULL ((void*)0)
01,231 #endif
01,232
01,233 // LZSSE2 [
01,234 #ifdef _N_alone
01,235 #else
01,236 #include "lzsse2.h"
01,237 #endif
01,238 // LZSSE2 ]
01,239
```

```
01,240 #ifdef _NAquaHash
01,241 // https://github.com/jandrewrogers/AquaHash/blob/master/aquahash.h
01,242 #include <wmmintrin.h>
01,243 // or may be just use #include <x86intrin.h> for all
01,244 #endif
01,245 #define MatchLenAboveWhichHASHkicksinAQUA (16) // 2021-Jul-31
01,246
01,247 #ifdef _NPRV
01,248 #include "prvhash42.h" //2020-Nov-12
01,249 #endif
01,250 // #define PRVhashlenInBYTES (28) //2020-Nov-12, it has to be multiples of 4, 4 is 32bit, 16 is 128bit, 20 is 160bit; ?!With 96bit/128bit there were collisions in
'Silesia'?!
01,251 // #define PRVhashlenInBYTES (12) // 2020-Nov-17, hacked for speed up
01,252 #define PRVhashlenInBYTES (24) // 2021-Jan-07, hacked for speed up, with 16 there were problems with SUPRPIG_The_Prague_Stringology_Club_(302-PostScript-files).tar
~500MB file?!
01,253 // #define MatchLenFromWhichPRVkicksin (256) //2020-Nov-12, if PRVhashlenInBYTES is 16 then MatchLenFromWhichPRVkicksin could be 18,24,36...
01,254 // #define MatchLenAboveWhichHASHkicksin (12) //2020-Nov-12, the [first X] bytes taken from the 224bit/28B hashes
01,255 #define MatchLenAboveWhichHASHkicksin (24) //2021-Jan-07, the [first X] bytes taken from the 224bit/28B hashes
01,256
01,257 // Comment it to see how slower 'BruteForce' is, for Wikipedia 100MB the ratio is 41KB/s versus 197KB/s.
01,258 #define ReplaceBruteForceWithRailgunSwampshineBailOut
01,259
01,260 void SearchIntoSlidingWindow(unsigned int* HowManyDittoTagsToEmit, unsigned int* ShortMediumLongOFFSET, uint64_t* retIndex, unsigned int* retMatch, char*
refStart, char* refEnd, char* encStart, char* encEnd, char* src, char* srcR, uint64_t srcSize);
01,261 unsigned int SlidingWindowVsLookAheadBuffer(char* refStart, char* refEnd, char* encStart, char* encEnd);
01,262 uint64_t Compress(char* ret, char* src, char* srcR, uint64_t srcSize);
01,263 //unsigned int Decompress(char* ret, char* src, unsigned int srcSize);
01,264 uint64_t Decompress(char* ret, char* src, uint64_t srcSize);
01,265 //char * Railgun_Trolldom(char * pbTarget, char * pbPattern, uint32_t cbTarget, uint32_t cbPattern);
01,266 char * Railgun_Trolldom_64(char * pbTarget, char * pbPattern, uint64_t cbTarget, uint32_t cbPattern); // 2020-Jan-10
01,267 char * Railgun_Doublent(char * pbTarget, char * pbPattern, uint32_t cbTarget, uint32_t cbPattern);
01,268 char * Railgun_BawBaw_reverse(char * pbTarget, char * pbPattern, uint32_t cbTarget, uint32_t cbPattern);
01,269 char * Railgun_Baw_reverse(char * pbTarget, char * pbPattern, uint32_t cbTarget, uint32_t cbPattern);
01,270
01,271 // should return how much bytes match, the count is the max chars compared:
01,272 int memcmp_CommonPrefixLength (
01,273     const void * buf1,
01,274     const void * buf2,
01,275     size_t count
01,276 )
01,277 {
01,278 int CPL=0;
01,279     if (!count)
01,280         return(0);
01,281
01,282     while ( count-- && *(char *)buf1 == *(char *)buf2 ) {
01,283         buf1 = (char *)buf1 + 1;
01,284         buf2 = (char *)buf2 + 1;
01,285 CPL++;
01,286     }
01,287
01,288     //return( *((unsigned char *)buf1) - *((unsigned char *)buf2) );
01,289     return( CPL );
01,290 }
01,291
01,292 /*
01,293 void memcpy_AVX_4K_prefetched (void *dst, const void *src, size_t nbytes) {
01,294 // F3 0F 6F /r RM V/V SSE2 Move unaligned packed integer values from xmm2/m128 to xmm1.
```



```

01,295 // MOVDQU xmm1, xmm2/m128
01,296 // F3 0F 7F /r MR V/V SSE2 Move unaligned packed integer values from xmm1 to xmm2/m128.
01,297 // MOVDQU xmm2/m128, xmm1
01,298 // VEX.128.F3.0F.WIG.6F /r RM V/V AVX Move unaligned packed integer values from xmm2/m128 to xmm1.
01,299 // VMOVDQU xmm1, xmm2/m128
01,300 // VEX.128.F3.0F.WIG.7F /r MR V/V AVX Move unaligned packed integer values from xmm1 to xmm2/m128.
01,301 // VMOVDQU xmm2/m128, xmm1
01,302 // VEX.256.F3.0F.WIG.6F /r RM V/V AVX Move unaligned packed integer values from ymm2/m256 to ymm1.
01,303 // VMOVDQU ymm1, ymm2/m256
01,304 // VEX.256.F3.0F.WIG.7F /r MR V/V AVX Move unaligned packed integer values from ymm1 to ymm2/m256.
01,305 // VMOVDQU ymm2/m256, ymm1
01,306 if ( (nbytes&0x3f) == 0 ) { // 64bytes per cycle
01,307     __asm{
01,308         mov         rsi, src
01,309         mov         rdi, dst
01,310         mov         rcx, nbytes
01,311         shr         rcx, 6
01,312     main_loop:
01,313         test        rcx, rcx ; 'nbytes' may be 0
01,314         jz          main_loop_end
01,315         prefetcht0  [rsi+64*64]
01,316         vmovdqu     xmm0, [rsi]
01,317         vmovdqu     xmm1, [rsi+16]
01,318         vmovdqu     xmm2, [rsi+32]
01,319         vmovdqu     xmm3, [rsi+48]
01,320         vmovdqu     [rdi], xmm0
01,321         vmovdqu     [rdi+16], xmm1
01,322         vmovdqu     [rdi+32], xmm2
01,323         vmovdqu     [rdi+48], xmm3
01,324         add         rsi, 64
01,325         add         rdi, 64
01,326         dec         rcx
01,327         jmp         main_loop
01,328     main_loop_end:
01,329         sfence
01,330     }
01,331 } else memcpy(dst, src, nbytes);
01,332 }
01,333
01,334 void memcpy_SSE2_4K_prefetched (void *dst, const void *src, size_t nbytes) {
01,335 // F3 0F 6F /r RM V/V SSE2 Move unaligned packed integer values from xmm2/m128 to xmm1.
01,336 // MOVDQU xmm1, xmm2/m128
01,337 // F3 0F 7F /r MR V/V SSE2 Move unaligned packed integer values from xmm1 to xmm2/m128.
01,338 // MOVDQU xmm2/m128, xmm1
01,339 // VEX.128.F3.0F.WIG.6F /r RM V/V AVX Move unaligned packed integer values from xmm2/m128 to xmm1.
01,340 // VMOVDQU xmm1, xmm2/m128
01,341 // VEX.128.F3.0F.WIG.7F /r MR V/V AVX Move unaligned packed integer values from xmm1 to xmm2/m128.
01,342 // VMOVDQU xmm2/m128, xmm1
01,343 // VEX.256.F3.0F.WIG.6F /r RM V/V AVX Move unaligned packed integer values from ymm2/m256 to ymm1.
01,344 // VMOVDQU ymm1, ymm2/m256
01,345 // VEX.256.F3.0F.WIG.7F /r MR V/V AVX Move unaligned packed integer values from ymm1 to ymm2/m256.
01,346 // VMOVDQU ymm2/m256, ymm1
01,347 if ( (nbytes&0x3f) == 0 ) { // 64bytes per cycle
01,348     __asm{
01,349         mov         rsi, src
01,350         mov         rdi, dst
01,351         mov         rcx, nbytes
01,352         shr         rcx, 6

```

```
01,353 main_loop:
01,354     test        rcx, rcx ; 'nbytes' may be 0
01,355     jz          main_loop_end
01,356     prefetcht0  [rsi+64*64]
01,357     movdqu      xmm0, [rsi]
01,358     movdqu      xmm1, [rsi+16]
01,359     movdqu      xmm2, [rsi+32]
01,360     movdqu      xmm3, [rsi+48]
01,361     movdqu      [rdi], xmm0
01,362     movdqu      [rdi+16], xmm1
01,363     movdqu      [rdi+32], xmm2
01,364     movdqu      [rdi+48], xmm3
01,365     add        rsi, 64
01,366     add        rdi, 64
01,367     dec        rcx
01,368     jmp        main_loop
01,369 main_loop_end:
01,370     sfence
01,371 }
01,372 } else memcpy(dst, src, nbytes);
01,373 }
01,374 */
01,375
01,376 #ifdef _N_HIGH_PRIORITY
01,377 // https://msdn.microsoft.com/en-us/library/windows/desktop/ms686219.aspx
01,378 #include <stdio.h>
01,379 #include <windows.h>
01,380 #include <tchar.h>
01,381 #endif
01,382
01,383 /*
01,384 int main( void )
01,385 {
01,386     DWORD dwError, dwPriClass;
01,387
01,388     if(!SetPriorityClass(GetCurrentProcess(), REALTIME_PRIORITY_CLASS))
01,389     {
01,390         _tprintf(TEXT("Already REALTIME_PRIORITY\n"));
01,391         goto Cleanup;
01,392     }
01,393
01,394     // Display priority class
01,395
01,396     dwPriClass = GetPriorityClass(GetCurrentProcess());
01,397
01,398     _tprintf(TEXT("Current priority class is 0x%x\n"), dwPriClass);
01,399
01,400     if (dwPriClass==0x00000100) printf("Current priority class is REALTIME_PRIORITY_CLASS.\n");
01,401
01,402
01,403 Cleanup:
01,404     // Clean up
01,405     ;
01,406     return 0;
01,407 }
01,408
01,409 // IDLE_PRIORITY_CLASS
01,410 // 0x00000040
```

```
01,411 // Process whose threads run only when the system is idle. The threads of the process are preempted by the threads of any process running in a higher priority class.
An example is a screen saver. The idle-priority class is inherited by child processes.
01,412
01,413 // NORMAL_PRIORITY_CLASS
01,414 // 0x00000020
01,415 // Process with no special scheduling needs.
01,416
01,417 // HIGH_PRIORITY_CLASS
01,418 // 0x00000080
01,419 // Process that performs time-critical tasks that must be executed immediately. The threads of the process preempt the threads of normal or idle priority class
processes. An example is the Task List, which must respond quickly when called by the user, regardless of the load on the operating system. Use extreme care when using the
high-priority class, because a high-priority class application can use nearly all available CPU time.
01,420
01,421 // REALTIME_PRIORITY_CLASS
01,422 // 0x00000100
01,423 // Process that has the highest possible priority. The threads of the process preempt the threads of all other processes, including operating system processes
performing important tasks. For example, a real-time process that executes for more than a very brief interval can cause disk caches not to flush or cause the mouse to be
unresponsive.
01,424 */
01,425
01,426 // Min_Match_Length=THRESHOLD=4 means 4 and bigger are to be encoded:
01,427 #define Min_Match_BAILOUT_Length (8)
01,428 #define Min_Match_Length (704)
01,429 // #define Min_Match_Length (4)
01,430 // #define Min_Match_Length (16) // Maximum MatchLength is 16, it decides the size of Look-ahead buffer - to avoid search beyound end. This needs more attention in the
future - to clarify it fully. Overlapping is also yet to come.
01,431 #define Min_Match_Length_SHORT (5)
01,432 // #define OffsetBITS (32-3*8) // 37bit=128GB
01,433 #define OffsetBITS (32*8) // 40bit=1TB
01,434 // #define OffsetBITS (20)
01,435 #define LengthBITS (1)
01,436
01,437 //12bit
01,438 // #define REF_SIZE (4095+Min_Match_Length)
01,439 // #define REF_SIZE ( ((1<<OffsetBITS)-1) + Min_Match_Length )
01,440 #define REF_SIZE ( ((1LL)<<OffsetBITS)-1) )
01,441 //3bit
01,442 // #define ENC_SIZE (7+Min_Match_Length)
01,443 #define ENC_SIZE ( ((1<<LengthBITS)-1) + Min_Match_Length )
01,444
01,445 #define _rotl_KAZE(x, n) (((x) << (n)) | ((x) >> (32-(n))))
01,446 #define _rotl_KAZE64(x, n) (((x) << (n)) | ((x) >> (64-(n))))
01,447
01,448 uint32_t FNV1A_Hash_YoshimitsuTRIADii(const char *str, uint64_t wrdlen)
01,449 {
01,450     const uint32_t PRIME = 709607;
01,451     uint32_t hash32 = 2166136261;
01,452     uint32_t hash32B = 2166136261;
01,453     uint32_t hash32C = 2166136261;
01,454     const char *p = str;
01,455     uint32_t Loop_Counter;
01,456     uint32_t Second_Line_Offset;
01,457
01,458     if (wrdlen >= 24) {
01,459         Loop_Counter = (wrdlen/24);
01,460         Loop_Counter++;
01,461         Second_Line_Offset = wrdlen-(Loop_Counter)*(3*4);
01,462         for(; Loop_Counter; Loop_Counter--, p += 3*sizeof(uint32_t)) {
```

```

01,463     hash32 = (hash32 ^ (_rotl_KAZE(*(uint32_t *) (p+0),5) ^ *(uint32_t *) (p+0+Second_Line_Offset))) * PRIME;
01,464     hash32B = (hash32B ^ (_rotl_KAZE(*(uint32_t *) (p+4+Second_Line_Offset),5) ^ *(uint32_t *) (p+4))) * PRIME;
01,465     hash32C = (hash32C ^ (_rotl_KAZE(*(uint32_t *) (p+8),5) ^ *(uint32_t *) (p+8+Second_Line_Offset))) * PRIME;
01,466 }
01,467     hash32 = (hash32 ^ _rotl_KAZE(hash32C,5) ) * PRIME;
01,468 } else {
01,469     // 1111=15; 10111=23
01,470     if (wrdlen & 4*sizeof(uint32_t)) {
01,471         hash32 = (hash32 ^ (_rotl_KAZE(*(uint32_t *) (p+0),5) ^ *(uint32_t *) (p+4))) * PRIME;
01,472         hash32B = (hash32B ^ (_rotl_KAZE(*(uint32_t *) (p+8),5) ^ *(uint32_t *) (p+12))) * PRIME;
01,473         p += 8*sizeof(uint16_t);
01,474     }
01,475     // Cases: 0,1,2,3,4,5,6,7,...,15
01,476     if (wrdlen & 2*sizeof(uint32_t)) {
01,477         hash32 = (hash32 ^ *(uint32_t *) (p+0)) * PRIME;
01,478         hash32B = (hash32B ^ *(uint32_t *) (p+4)) * PRIME;
01,479         p += 4*sizeof(uint16_t);
01,480     }
01,481     // Cases: 0,1,2,3,4,5,6,7
01,482     if (wrdlen & sizeof(uint32_t)) {
01,483         hash32 = (hash32 ^ *(uint16_t *) (p+0)) * PRIME;
01,484         hash32B = (hash32B ^ *(uint16_t *) (p+2)) * PRIME;
01,485         p += 2*sizeof(uint16_t);
01,486     }
01,487     if (wrdlen & sizeof(uint16_t)) {
01,488         hash32 = (hash32 ^ *(uint16_t *) p) * PRIME;
01,489         p += sizeof(uint16_t);
01,490     }
01,491     if (wrdlen & 1)
01,492         hash32 = (hash32 ^ *p) * PRIME;
01,493 }
01,494     hash32 = (hash32 ^ _rotl_KAZE(hash32B,5) ) * PRIME;
01,495     return hash32 ^ (hash32 >> 16);
01,496 }
01,497
01,498 uint32_t FNV1A_Hash_YoshimitsuTRIAD(const char *str, uint64_t wrdlen)
01,499 {
01,500     const uint32_t PRIME = 709607;
01,501     uint32_t hash32 = 2166136261;
01,502     uint32_t hash32B = 2166136261;
01,503     uint32_t hash32C = 2166136261;
01,504     //uint32_t hash32D = 2166136261;
01,505     const char *p = str;
01,506
01,507     for(; wrdlen >= 3*2*sizeof(uint32_t); wrdlen -= 3*2*sizeof(uint32_t), p += 3*2*sizeof(uint32_t)) {
01,508         hash32 = (hash32 ^ (_rotl_KAZE(*(uint32_t *) (p+0),5) ^ *(uint32_t *) (p+4))) * PRIME;
01,509         hash32B = (hash32B ^ (_rotl_KAZE(*(uint32_t *) (p+8),5) ^ *(uint32_t *) (p+12))) * PRIME;
01,510         hash32C = (hash32C ^ (_rotl_KAZE(*(uint32_t *) (p+16),5) ^ *(uint32_t *) (p+20))) * PRIME;
01,511         //hash32D = (hash32D ^ (_rotl_KAZE(*(uint32_t *) (p+24),5) ^ *(uint32_t *) (p+28))) * PRIME;
01,512     }
01,513 /*
01,514 // Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 16.00.30319.01 for 80x86 gave this:
01,515 // 12d-0f4+2= 59 bytes, No CARAMBA anymore.
01,516 $LL9@FNV1A_Hash@2:
01,517
01,518 ; 160 :             hash32 = (hash32 ^ (_rotl_KAZE(*(uint32_t *) (p+0),5) ^ *(uint32_t *) (p+4))) * PRIME;
01,519
01,520 000f4 8b 01             mov     eax, DWORD PTR [ecx]

```

```

01,521 000f6 c1 c0 05      rol     eax, 5
01,522 000f9 33 41 04      xor     eax, DWORD PTR [ecx+4]
01,523 000fc 83 eb 18      sub     ebx, 24                      ; 00000018H
01,524 000ff 33 f0      xor     esi, eax
01,525
01,526 ; 161 :          hash32B = (hash32B ^ (_rotl_KAZE(*(uint32_t *) (p+8),5) ^ *(uint32_t *) (p+12))) * PRIME;
01,527
01,528 00101 8b 41 08      mov     eax, DWORD PTR [ecx+8]
01,529 00104 69 f6 e7 d3 0a    imul    esi, 709607                  ; 000ad3e7H
01,530 00      imul
01,531 0010a c1 c0 05      rol     eax, 5
01,532 0010d 33 41 0c      xor     eax, DWORD PTR [ecx+12]
01,533 00110 83 c1 18      add     ecx, 24                      ; 00000018H
01,534 00113 33 f8      xor     edi, eax
01,535
01,536 ; 162 :          hash32C = (hash32C ^ (_rotl_KAZE(*(uint32_t *) (p+16),5) ^ *(uint32_t *) (p+20))) * PRIME;
01,537
01,538 00115 8b 41 f8      mov     eax, DWORD PTR [ecx-8]
01,539 00118 69 ff e7 d3 0a    imul    edi, 709607                  ; 000ad3e7H
01,540 00      imul
01,541 0011e c1 c0 05      rol     eax, 5
01,542 00121 33 41 fc      xor     eax, DWORD PTR [ecx-4]
01,543 00124 33 e8      xor     ebp, eax
01,544 00126 69 ed e7 d3 0a    imul    ebp, 709607                  ; 000ad3e7H
01,545 00      imul
01,546 0012c 4a      dec     edx
01,547 0012d 75 c5      jne     SHORT $LL9@FNV1A_Hash@2
01,548 */
01,549
01,550 /*
01,551 // Intel(R) C++ Compiler XE for applications running on IA-32, Version 12.1.1.258 Build 20111011 gave this:
01,552 // 216a-212f+2= 61 bytes, No CARAMBA anymore.
01,553 ;;      for(; wrdlen >= 3*2*sizeof(uint32_t); wrdlen -= 3*2*sizeof(uint32_t), p += 3*2*sizeof(uint32_t)) {
01,554
01,555 02127 83 fa 18      cmp     edx, 24
01,556 0212a 72 43      jb     .B4.5 ; Prob 10%
01,557                      ; LOE eax edx ecx ebx ebp esi edi
01,558 .B4.2:          ; Preds .B4.1
01,559 0212c 89 34 24      mov     DWORD PTR [esp], esi
01,560                      ; LOE eax edx ecx ebx ebp edi
01,561 .B4.3:          ; Preds .B4.2 .B4.3
01,562
01,563 ;;          hash32 = (hash32 ^ (_rotl_KAZE(*(uint32_t *) (p+0),5) ^ *(uint32_t *) (p+4))) * PRIME;
01,564
01,565 0212f 8b 31      mov     esi, DWORD PTR [ecx]
01,566 02131 83 c2 e8      add     edx, -24
01,567 02134 c1 c6 05      rol     esi, 5
01,568 02137 33 71 04      xor     esi, DWORD PTR [4+ecx]
01,569 0213a 33 de      xor     ebx, esi
01,570
01,571 ;;          hash32B = (hash32B ^ (_rotl_KAZE(*(uint32_t *) (p+8),5) ^ *(uint32_t *) (p+12))) * PRIME;
01,572
01,573 0213c 8b 71 08      mov     esi, DWORD PTR [8+ecx]
01,574 0213f c1 c6 05      rol     esi, 5
01,575 02142 33 71 0c      xor     esi, DWORD PTR [12+ecx]
01,576 02145 33 fe      xor     edi, esi
01,577
01,578 ;;          hash32C = (hash32C ^ (_rotl_KAZE(*(uint32_t *) (p+16),5) ^ *(uint32_t *) (p+20))) * PRIME;

```

```

01,579
01,580 02147 8b 71 10      mov esi, DWORD PTR [16+ecx]
01,581 0214a c1 c6 05      rol esi, 5
01,582 0214d 33 71 14      xor esi, DWORD PTR [20+ecx]
01,583 02150 83 c1 18      add ecx, 24
01,584 02153 33 ee        xor ebp, esi
01,585 02155 69 db e7 d3 0a
01,586      00          imul ebx, ebx, 709607
01,587 0215b 69 ff e7 d3 0a
01,588      00          imul edi, edi, 709607
01,589 02161 69 ed e7 d3 0a
01,590      00          imul ebp, ebp, 709607
01,591 02167 83 fa 18      cmp edx, 24
01,592 0216a 73 c3        jae .B4.3 ; Prob 82%
01,593      ; LOE eax edx ecx ebx ebp edi
01,594 .B4.4:          ; Preds .B4.3
01,595 0216c 8b 34 24      mov esi, DWORD PTR [esp]
01,596      ; LOE eax edx ecx ebx ebp esi edi
01,597 .B4.5:          ; Preds .B4.1 .B4.4
01,598 */
01,599
01,600 }
01,601 if (p != str) {
01,602     hash32 = (hash32 ^ _rotl_KAZE(hash32C,5) ) * PRIME;
01,603     //hash32B = (hash32B ^ _rotl_KAZE(hash32D,5) ) * PRIME;
01,604 }
01,605
01,606 // 1111=15; 10111=23
01,607 if (wrdlen & 4*sizeof(uint32_t)) {
01,608     hash32 = (hash32 ^ (_rotl_KAZE(*(uint32_t*)(p+0),5) ^ *(uint32_t*)(p+4))) * PRIME;
01,609     hash32B = (hash32B ^ (_rotl_KAZE(*(uint32_t*)(p+8),5) ^ *(uint32_t*)(p+12))) * PRIME;
01,610     p += 8*sizeof(uint16_t);
01,611 }
01,612 // Cases: 0,1,2,3,4,5,6,7,...,15
01,613 if (wrdlen & 2*sizeof(uint32_t)) {
01,614     hash32 = (hash32 ^ *(uint32_t*)(p+0)) * PRIME;
01,615     hash32B = (hash32B ^ *(uint32_t*)(p+4)) * PRIME;
01,616     p += 4*sizeof(uint16_t);
01,617 }
01,618 // Cases: 0,1,2,3,4,5,6,7
01,619 if (wrdlen & sizeof(uint32_t)) {
01,620     hash32 = (hash32 ^ *(uint16_t*)(p+0)) * PRIME;
01,621     hash32B = (hash32B ^ *(uint16_t*)(p+2)) * PRIME;
01,622     p += 2*sizeof(uint16_t);
01,623 }
01,624 if (wrdlen & sizeof(uint16_t)) {
01,625     hash32 = (hash32 ^ *(uint16_t*)p) * PRIME;
01,626     p += sizeof(uint16_t);
01,627 }
01,628 if (wrdlen & 1)
01,629     hash32 = (hash32 ^ *p) * PRIME;
01,630
01,631 hash32 = (hash32 ^ _rotl_KAZE(hash32B,5) ) * PRIME;
01,632 return hash32 ^ (hash32 >> 16);
01,633 }
01,634
01,635
01,636 //unsigned char DD[12]; //2020-Nov-20

```

```
01,637 unsigned char DD[MatchLenAboveWhichHASHkicksin]; //2021-Jan-02, it could be raised to 16B or 20B
01,638 unsigned char DDAES[MatchLenAboveWhichHASHkicksinAQUA];
01,639
01,640 // CRC64 & CRC32 [
01,641
01,642 uint64_t poly64 = 0x42F0E1EBA9EA3693;
01,643 //uint64_t poly64 = 0xC96C5795D7870F42;
01,644 //uint64_t poly64 = 0x92D8AF2BAF0E1E85;
01,645 //uint64_t poly64 = 0xA17870F5D4F51B49;
01,646
01,647 uint32_t poly32 = 0x82F63B78; //CRC32C polynomial: 0x82F63B78(LE) // CRC-32C (Castagnoli) Polynomial representations: Reversed
01,648
01,649 // 2020-Dec-31 [
01,650
01,651 // CRC-32C (Castagnoli) iSCSI, SCTP, G.hn payload, SSE4.2, Btrfs, ext4, Ceph: 0x1EDC6F41
01,652 // x^{32}+x^{28}+x^{27}+x^{26}+x^{25}+x^{23}+x^{22}+x^{20}+x^{19}+x^{18}+x^{14}+x^{13}+x^{11}+x^{10}+x^9+x^8+x^6+1
01,653 //uint32_t poly32CN = 0x1EDC6F41; // CRC-32C (Castagnoli) Polynomial representations: Normal
01,654 uint32_t poly32CN = 0x82F63B78; // CRC-32C (Castagnoli) Polynomial representations: Reversed
01,655
01,656 //Definition: Ethernet frame
01,657 //When transmitting data over Ethernet, the Ethernet frame is primarily responsible for the correct rulemaking and successful transmission of data packets.
01,658 //Essentially, data sent over Ethernet is carried by the frame. An Ethernet frame is between 64 bytes and 1,518 bytes big,
01,659 //depending on the size of the data to be transported.
01,660
01,661 // CRC-32K (Koopman {1,3,28}) Excellent at Ethernet frame length, poor performance with long files: 0x741B8CD7
01,662 // x^{32}+x^{30}+x^{29}+x^{28}+x^{26}+x^{20}+x^{19}+x^{17}+x^{16}+x^{15}+x^{11}+x^{10}+x^7+x^6+x^4+x^2+x+1
01,663 //uint32_t poly32KN = 0x741B8CD7; // CRC-32K (Koopman) Polynomial representations: Normal
01,664 uint32_t poly32KN = 0xEB31D82E; // CRC-32K (Koopman) Polynomial representations: Reversed
01,665
01,666 // CRC-32K2 (Koopman {1,1,30}) Excellent at Ethernet frame length, poor performance with long files: 0x32583499
01,667 //uint32_t poly32K2N = 0x32583499; // CRC-32K2 (Koopman) Polynomial representations: Normal
01,668 uint32_t poly32K2N = 0x992C1A4C; // CRC-32K2 (Koopman) Polynomial representations: Reversed
01,669
01,670 // 2020-Dec-31 ]
01,671
01,672 uint64_t crc64_tableGEN[256];
01,673 uint32_t crc32c_tableGEN[256];
01,674
01,675 // 2020-Dec-31 [
01,676
01,677 uint32_t crc32cn_tableGEN[256];
01,678 uint32_t crc32kn_tableGEN[256];
01,679 uint32_t crc32k2n_tableGEN[256];
01,680
01,681 // 2020-Dec-31 ]
01,682
01,683 void crc64_generate_table()
01,684 {
01,685     int i, j;
01,686     uint64_t crc;
01,687     for(i=0; i<256; ++i) {
01,688         crc = i;
01,689         for(j=0; j<8; ++j) {
01,690             if(crc & 1)
01,691                 crc = (crc >> 1) ^ poly64;
01,692             else
01,693                 crc >>= 1;
01,694         }
01,695     }
```

```

01,695     crc64_tableGEN[i] = crc;
01,696 //     printf("%016I64X\n", crc);
01,697 }
01,698 }
01,699
01,700 void crc32c_generate_table()
01,701 {
01,702     int i, j;
01,703     uint32_t crc;
01,704     for(i=0; i<256; ++i) {
01,705         crc = i;
01,706         for(j=0; j<8; ++j) {
01,707             // GCC complains, ICL not:
01,708             //Nakamichi_Ryuugan-ditto-1TB_btree.c: In function 'crc32c_generate_table':
01,709             //Nakamichi_Ryuugan-ditto-1TB_btree.c:1166:24: error: expected expression before 'int'
01,710             //     crc = (crc >> 1) ^ (-int(crc & 1) & poly32);
01,711             //
01,712             //crc = (crc >> 1) ^ (-int(crc & 1) & poly32);
01,713             if(crc & 1)
01,714                 crc = (crc >> 1) ^ poly32;
01,715             else
01,716                 crc >>= 1;
01,717         }
01,718         crc32c_tableGEN[i] = crc;
01,719 //     printf("%08I32X\n", crc); // This gives the pre-computed table further below...
01,720 }
01,721 }
01,722
01,723 uint64_t crc64(const unsigned char *buffer, unsigned long length)
01,724 {
01,725     uint64_t crc = ~0;
01,726     while (length--) {
01,727         crc = crc64_tableGEN[(crc ^ *(buffer++)) & 0xff] ^ (crc >> 8);
01,728     }
01,729
01,730     return crc;
01,731 }
01,732
01,733 uint32_t crc32c_sw(const unsigned char *data, unsigned long length)
01,734 {
01,735     uint32_t crc = ~0;
01,736
01,737     while (length--)
01,738         crc = crc32c_tableGEN[(crc ^ *data++) & 0xFF] ^ (crc >> 8);
01,739
01,740     return ~crc; // Had to add '~' in order to hash "Sanmayce" to DFA9D91A
01,741 }
01,742
01,743 // 2020-Dec-31 [
01,744
01,745 void crc32cn_generate_table()
01,746 {
01,747     int i, j;
01,748     uint32_t crc;
01,749     for(i=0; i<256; ++i) {
01,750         crc = i;
01,751         for(j=0; j<8; ++j) {
01,752             // GCC complains, ICL not:

```



```
01,753 //Nakamichi_Ryuugan-ditto-1TB_btree.c: In function 'crc32c_generate_table':
01,754 //Nakamichi_Ryuugan-ditto-1TB_btree.c:1166:24: error: expected expression before 'int'
01,755 //   crc = (crc >> 1) ^ (~int(crc & 1) & poly32CN);
01,756 //
01,757 //   //crc = (crc >> 1) ^ (-int(crc & 1) & poly32CN);
01,758 //   if(crc & 1)
01,759 //       crc = (crc >> 1) ^ poly32CN;
01,760 //   else
01,761 //       crc >>= 1;
01,762 //   }
01,763 //   crc32cn_tableGEN[i] = crc;
01,764 //   printf("%08I32X\n", crc);
01,765 // }
01,766 // }
01,767 //
01,768 void crc32kn_generate_table()
01,769 {
01,770 int i, j;
01,771 uint32_t crc;
01,772 for(i=0; i<256; ++i) {
01,773     crc = i;
01,774     for(j=0; j<8; ++j) {
01,775         // GCC complains, ICL not:
01,776 //Nakamichi_Ryuugan-ditto-1TB_btree.c: In function 'crc32c_generate_table':
01,777 //Nakamichi_Ryuugan-ditto-1TB_btree.c:1166:24: error: expected expression before 'int'
01,778 //   crc = (crc >> 1) ^ (~int(crc & 1) & poly32KN);
01,779 //
01,780 //   //crc = (crc >> 1) ^ (-int(crc & 1) & poly32KN);
01,781 //   if(crc & 1)
01,782 //       crc = (crc >> 1) ^ poly32KN;
01,783 //   else
01,784 //       crc >>= 1;
01,785 //   }
01,786 //   crc32kn_tableGEN[i] = crc;
01,787 //   printf("%08I32X\n", crc);
01,788 // }
01,789 // }
01,790 //
01,791 void crc32k2n_generate_table()
01,792 {
01,793 int i, j;
01,794 uint32_t crc;
01,795 for(i=0; i<256; ++i) {
01,796     crc = i;
01,797     for(j=0; j<8; ++j) {
01,798         // GCC complains, ICL not:
01,799 //Nakamichi_Ryuugan-ditto-1TB_btree.c: In function 'crc32c_generate_table':
01,800 //Nakamichi_Ryuugan-ditto-1TB_btree.c:1166:24: error: expected expression before 'int'
01,801 //   crc = (crc >> 1) ^ (~int(crc & 1) & poly32K2N);
01,802 //
01,803 //   //crc = (crc >> 1) ^ (-int(crc & 1) & poly32K2N);
01,804 //   if(crc & 1)
01,805 //       crc = (crc >> 1) ^ poly32K2N;
01,806 //   else
01,807 //       crc >>= 1;
01,808 //   }
01,809 //   crc32k2n_tableGEN[i] = crc;
01,810 //   printf("%08I32X\n", crc);
01,811 // }
```

```
01,811     }
01,812 }
01,813
01,814 uint32_t crc32cn_sw(const unsigned char *data, unsigned long length)
01,815 {
01,816     uint32_t crc = ~0;
01,817
01,818     while (length--)
01,819         crc = crc32cn_tableGEN[(crc ^ *data++) & 0xFF] ^ (crc >> 8);
01,820
01,821     return ~crc;
01,822 }
01,823
01,824 uint32_t crc32kn_sw(const unsigned char *data, unsigned long length)
01,825 {
01,826     uint32_t crc = ~0;
01,827
01,828     while (length--)
01,829         crc = crc32kn_tableGEN[(crc ^ *data++) & 0xFF] ^ (crc >> 8);
01,830
01,831     return ~crc;
01,832 }
01,833
01,834 uint32_t crc32k2n_sw(const unsigned char *data, unsigned long length)
01,835 {
01,836     uint32_t crc = ~0;
01,837
01,838     while (length--)
01,839         crc = crc32k2n_tableGEN[(crc ^ *data++) & 0xFF] ^ (crc >> 8);
01,840
01,841     return ~crc;
01,842 }
01,843
01,844 // 2020-Dec-31 ]
01,845
01,846 /*
01,847  * CRC32C
01,848  * @Article{castagnoli-crc,
01,849  *   author = { Guy Castagnoli and Stefan Braeuer and Martin Herrman},
01,850  *   title = {{Optimization of Cyclic Redundancy-Check Codes with 24
01,851  *             and 32 Parity Bits}},
01,852  *   journal = IEEE Transactions on Communication,
01,853  *   year = {1993},
01,854  *   volume = {41},
01,855  *   number = {6},
01,856  *   pages = {},
01,857  *   month = {June},
01,858  *}
01,859  * Used by the iSCSI driver, possibly others, and derived from the
01,860  * the iscsi-crc.c module of the linux-iscsi driver at
01,861  * http://linux-iscsi.sourceforge.net.
01,862  *
01,863  * Following the example of lib/crc32, this function is intended to be
01,864  * flexible and useful for all users. Modules that currently have their
01,865  * own crc32c, but hopefully may be able to use this one are:
01,866  * net/sctp (please add all your doco to here if you change to
01,867  *           use this one!)
01,868  * <endoflist>
```

```
01,869 *
01,870 * Copyright (c) 2004 Cisco Systems, Inc.
01,871 *
01,872 * This program is free software; you can redistribute it and/or modify it
01,873 * under the terms of the GNU General Public License as published by the Free
01,874 * Software Foundation; either version 2 of the License, or (at your option)
01,875 * any later version.
01,876 *
01,877 */
01,878
01,879 /*
01,880 * This is the CRC-32C table Big Endian
01,881 * Generated with:
01,882 * width = 32 bits
01,883 * poly = 0x1EDC6F41
01,884 * reflect input bytes = true
01,885 * reflect output bytes = true
01,886 */
01,887
01,888 /*
01,889 static const uint32_t crc32c_table[256] = {
01,890 0x00000000L, 0xF26B8303L, 0xE13B70F7L, 0x1350F3F4L,
01,891 0xC79A971FL, 0x35F1141CL, 0x26A1E7E8L, 0xD4CA64EBL,
01,892 0x8AD958CFL, 0x78B2DBCCL, 0x6BE22838L, 0x9989AB3BL,
01,893 0x4D43CFD0L, 0xBF284CD3L, 0xAC78BF27L, 0x5E133C24L,
01,894 0x105EC76FL, 0xE235446CL, 0xF165B798L, 0x030E349BL,
01,895 0xD7C45070L, 0x25AFD373L, 0x36FF2087L, 0xC494A384L,
01,896 0x9A879FA0L, 0x68EC1CA3L, 0x7BBCE57L, 0x89D76C54L,
01,897 0x5D1D08BFL, 0xAF768BBCL, 0xBC267848L, 0x4E4DFB4BL,
01,898 0x20BD8EDEL, 0xD2D60DDDL, 0xC186FE29L, 0x33ED7D2AL,
01,899 0xE72719C1L, 0x154C9AC2L, 0x061C6936L, 0xF477EA35L,
01,900 0xAA64D611L, 0x580F5512L, 0x4B5FA6E6L, 0xB93425E5L,
01,901 0x6DFFE410EL, 0x9F95C20DL, 0x8CC531F9L, 0x7EAE2FAL,
01,902 0x30E349B1L, 0xC288CAB2L, 0xD1D83946L, 0x23B3BA45L,
01,903 0xF779DEAEL, 0x05125DADL, 0x1642AE59L, 0xE4292D5AL,
01,904 0xBA3A117EL, 0x4851927DL, 0x5B016189L, 0xA96AE28AL,
01,905 0x7DA08661L, 0x8FCB0562L, 0x9C9BF696L, 0x6EF07595L,
01,906 0x417B1DBCL, 0xB3109EBFL, 0xA0406D4BL, 0x522BEE48L,
01,907 0x86E18AA3L, 0x748A09A0L, 0x67DAFA54L, 0x95B17957L,
01,908 0xCB8A24573L, 0x39C9C670L, 0x2A993584L, 0xD8F2B687L,
01,909 0x0C38D26CL, 0xFE53516FL, 0xED03A29BL, 0x1F682198L,
01,910 0x5125DAD3L, 0xA34E59D0L, 0xB01EAA24L, 0x42752927L,
01,911 0x96BF4DCCL, 0x64D4CECFL, 0x77843DBL, 0x85EFBE38L,
01,912 0xDBFC821CL, 0x2997011FL, 0x3AC7F2EBL, 0xC8AC71E8L,
01,913 0x1C661503L, 0xEE0D9600L, 0xFD5D65F4L, 0x0F36E6F7L,
01,914 0x61C69362L, 0x93AD1061L, 0x80FDE395L, 0x72966096L,
01,915 0xA65C047DL, 0x5437877EL, 0x4767748AL, 0xB50CF789L,
01,916 0xEB1FCBADL, 0x197448AEL, 0x0A24BB5AL, 0xF84F3859L,
01,917 0x2C855CB2L, 0xDEEDFB1L, 0xCDBE2C45L, 0x3FD5AF46L,
01,918 0x7198540DL, 0x83F3D70EL, 0x90A324FAL, 0x62C8A7F9L,
01,919 0xB602C312L, 0x44694011L, 0x5739B3E5L, 0xA55230E6L,
01,920 0xFB410CC2L, 0x092A8FC1L, 0x1A7A7C35L, 0xE811FF36L,
01,921 0x3CDB9BDDL, 0xCEB018DEL, 0xDDE0EB2AL, 0x2F8B6829L,
01,922 0x82F63B78L, 0x709DB87BL, 0x63CD4B8FL, 0x91A6C88CL,
01,923 0x456CAC67L, 0xB7072F64L, 0xA457DC90L, 0x563C5F93L,
01,924 0x082F63B7L, 0xFA44E0B4L, 0xE9141340L, 0x1B7F9043L,
01,925 0xCFB5F4A8L, 0x3DDE77ABL, 0x2E8E845FL, 0xDCE5075CL,
01,926 0x92A8FC17L, 0x60C37F14L, 0x73938CE0L, 0x81F80FE3L,
```

```

01,927 0x55326B08L, 0xA759E80BL, 0xB4091BFFL, 0x466298FCL,
01,928 0x1871A4D8L, 0xEA1A27DBL, 0xF94AD42FL, 0xB21572CL,
01,929 0xDFEB33C7L, 0x2D80B0C4L, 0x3ED04330L, 0xCCB8C033L,
01,930 0xA24BB5A6L, 0x502036A5L, 0x4370C551L, 0xB11B4652L,
01,931 0x65D122B9L, 0x97BAA1BAL, 0x84EA524EL, 0x7681D14DL,
01,932 0x2892ED69L, 0xD4F96E6AL, 0xC9A99D9EL, 0x3BC21E9DL,
01,933 0xEF087A76L, 0x1D63F975L, 0xE330A81L, 0xFC588982L,
01,934 0xB21572C9L, 0x407EF1CAL, 0x532E023EL, 0xA145813DL,
01,935 0x758FE5D6L, 0x87E466D5L, 0x94B49521L, 0x66DF1622L,
01,936 0x38CC2A06L, 0xCA7A905L, 0xD9F75AF1L, 0x2B9CD9F2L,
01,937 0xFF56BD19L, 0x0D3D3E1AL, 0x1E6DCDEEL, 0xEC064EEDL,
01,938 0xC38D26C4L, 0x31E6A5C7L, 0x22B65633L, 0xD0DD530L,
01,939 0x0417B1DBL, 0xF67C32D8L, 0xE52CC12CL, 0x1747422FL,
01,940 0x49547E0BL, 0xBB3FFD08L, 0xA86F0EFCL, 0x5A048DFFL,
01,941 0x8ECE914L, 0x7CA56A17L, 0x6FF599E3L, 0x9D9E1AE0L,
01,942 0xD3D3E1ABL, 0x21B862A8L, 0x32E8915CL, 0xC083125FL,
01,943 0x144976B4L, 0xE622F5B7L, 0xF5720643L, 0x07198540L,
01,944 0x590AB964L, 0xAB613A67L, 0xB831C993L, 0x4A5A4A90L,
01,945 0x9E902E7BL, 0x6CFBAD78L, 0x7FAB5E8CL, 0x8DC0DD8FL,
01,946 0xE330A81AL, 0x115B2B19L, 0x02BD8EDL, 0xF0605BEEL,
01,947 0x24AA3F05L, 0xD6C1BC06L, 0xC5914FF2L, 0x37FACCF1L,
01,948 0x69E9F0D5L, 0x9B8273D6L, 0x88D28022L, 0x7AB90321L,
01,949 0xAE7367CAL, 0x5C18E4C9L, 0x4F48173DL, 0xBD23943EL,
01,950 0xF36E6F75L, 0x0105EC76L, 0x12551F82L, 0xE03E9C81L,
01,951 0x34F4F86AL, 0xC69F7B69L, 0xD5CF889DL, 0x27A40B9EL,
01,952 0x79B737BAL, 0x8BDCB4B9L, 0x988C474DL, 0x6AE7C44EL,
01,953 0xBE2DA0A5L, 0x4C4623A6L, 0x5F16D052L, 0xAD7D5351L
01,954 };
01,955 */
01,956
01,957 /*
01,958  * Steps through buffer one byte at a time, calculates reflected
01,959  * crc using table.
01,960  */
01,961
01,962 // Roadhouse, you know, instead of returning 96bit directly put them into the global 'unsigned char DD[12]' 96bit definition.
01,963 /*
01,964 void DoubleDeuce(const unsigned char *buffer, unsigned long length)
01,965 {
01,966     uint64_t crc64 = ~0;
01,967     uint32_t crc32 = ~0;
01,968
01,969     while (length--) {
01,970         crc64 = crc64_tableGEN[(crc64 ^ *buffer) & 0xff] ^ (crc64 >> 8);
01,971         crc32 = crc32c_tableGEN[(crc32 ^ *buffer) & 0xFF] ^ (crc32 >> 8);
01,972         buffer++;
01,973     }
01,974     //return crc;
01,975     *(uint64_t*)&DD[0] = crc64;
01,976     *(uint32_t*)&DD[8] = crc32;
01,977 }
01,978 // Next one 16B was problematic with ~500MB stringology testfile?!
01,979 void DoubleDeuce(const unsigned char *buffer, unsigned long length)
01,980 {
01,981     uint32_t crc32cn = ~0; // OnWard: [ABCD] is traversed as ACBD
01,982     uint32_t crc32kn = ~0; // InWard: [ABCD] is traversed as ADCB
01,983     uint32_t crc32k2n = ~0; // OffWard: [ABCD] is traversed as DCBA
01,984     uint32_t crc32cnHALVED = ~0; // HalfWard: [ABCD] is traversed as ACBD

```

```

01,985 int HALVED= (length>>1); // They all are EVEN, if in the future ODD are involved then use: int HALVED= ((length+1)>>1); // 3:2, 4:2, 5:3
01,986 const unsigned char *bufferREVERSED=buffer;
01,987 const unsigned char *bufferInWardL=buffer;
01,988 const unsigned char *bufferInWardR=buffer+(length-1);
01,989 int InWard=0;
01,990 int HALFward=0;
01,991
01,992 while (length-->0) {
01,993     crc32cn = crc32cn_tableGEN[(crc32cn ^ *buffer) & 0xFF] ^ (crc32cn >> 8); // Forward i.e. OnWard
01,994     // HALFward traversing is 0,0;1,1; for length=4 or Half1+0, Half2+0, Half1+1, Half2+1,...
01,995     if (InWard & 1) { // Odd i.e. Take Half2
01,996         crc32cnHALVED = crc32cn_tableGEN[(crc32cnHALVED ^ *(bufferREVERSED+HALFward+HALVED)) & 0xFF] ^ (crc32cnHALVED >> 8);
01,997         HALFward++;
01,998     } else { // Even i.e. Take Half1
01,999         crc32cnHALVED = crc32cn_tableGEN[(crc32cnHALVED ^ *(bufferREVERSED+HALFward)) & 0xFF] ^ (crc32cnHALVED >> 8);
02,000     }
02,001     //crc32kn = crc32kn_tableGEN[(crc32kn ^ *buffer) & 0xFF] ^ (crc32kn >> 8);
02,002     // InWard traversing is Left, Right, Left, Right,...
02,003     if (InWard++ & 1) { // Odd i.e. Take Right
02,004         crc32kn = crc32kn_tableGEN[(crc32kn ^ *(bufferInWardR--)) & 0xFF] ^ (crc32kn >> 8);
02,005     } else { // Even i.e. Take Left
02,006         crc32kn = crc32kn_tableGEN[(crc32kn ^ *(bufferInWardL++)) & 0xFF] ^ (crc32kn >> 8);
02,007     }
02,008     //crc32k2n = crc32k2n_tableGEN[(crc32k2n ^ *buffer) & 0xFF] ^ (crc32k2n >> 8);
02,009     crc32k2n = crc32k2n_tableGEN[(crc32k2n ^ *(bufferREVERSED+length)) & 0xFF] ^ (crc32k2n >> 8); // Backward i.e. OffWard
02,010     buffer++;
02,011 }
02,012 //return crc;
02,013 *(uint32_t*)&DD[0] = crc32cn; //~
02,014 *(uint32_t*)&DD[4] = crc32kn; //~
02,015 *(uint32_t*)&DD[8] = crc32k2n; //~
02,016 *(uint32_t*)&DD[12] = crc32cnHALVED; //~
02,017 }
02,018 */
02,019 void DoubleDeuce(const unsigned char *buffer, unsigned long length) // 3x2x4B=24B: 2xCastagnoli, 2xKoopman, 2xKoopman2
02,020 {
02,021     uint32_t crc32cn = ~0; // OnWard: [ABCD] is traversed as ACBD, ugh 2021-Jul-31, ABCD
02,022     uint32_t crc32k2n = ~0; // OffWard: [ABCD] is traversed as DCBA
02,023
02,024     uint32_t crc32kn = ~0; // InWard: [ABCD] is traversed as ADCB
02,025     uint32_t crc32k2nOUT = ~0; // OutWard: [ABCD] is traversed as BCAD // 2020-Jan-07
02,026
02,027     uint32_t crc32cnHALVED = ~0; // HalfWard: [ABCD] is traversed as ACBD
02,028     uint32_t crc32knBHW = ~0; // BackHalfWard: [ABCD] is traversed as DCBA // 2020-Jan-07
02,029
02,030     int HALVED= (length>>1); // They all are EVEN, if in the future ODD are involved then use: int HALVED= ((length+1)>>1); // 3:2, 4:2, 5:3
02,031     const unsigned char *bufferREVERSED=buffer;
02,032     const unsigned char *bufferInWardL=buffer;
02,033     const unsigned char *bufferInWardR=buffer+(length-1);
02,034     const unsigned char *bufferOutWardL=buffer+(HALVED-1);
02,035     const unsigned char *bufferOutWardR=buffer+HALVED;
02,036     int InWard=0;
02,037     int HALFward=0;
02,038     int lengthKey=length;
02,039
02,040     while (length-->0) {
02,041         crc32cn = crc32cn_tableGEN[(crc32cn ^ *buffer) & 0xFF] ^ (crc32cn >> 8); // Forward i.e. OnWard
02,042         crc32k2n = crc32k2n_tableGEN[(crc32k2n ^ *(bufferREVERSED+length)) & 0xFF] ^ (crc32k2n >> 8); // Backward i.e. OffWard

```

Listing: Nakamichi_Ryuugan-ditto-1TB_btree.c; Last version: 2021-Aug-30; Font: MxPlus ToshibaTxL2 8x16.ttf; Downloadable at: www.sanmayce.com/Nakamichi/Kaidanji.zip

```

02,101 //    }
02,102
02,103 // https://software.intel.com/sites/landingpage/IntrinsicsGuide/#expand=233,272&text=_mm_aesenc_si128
02,104 /*
02,105 __m128i _mm_aesenc_si128 (__m128i a, __m128i RoundKey)
02,106 Synopsis
02,107 __m128i _mm_aesenc_si128 (__m128i a, __m128i RoundKey)
02,108 #include <wmmintrin.h>
02,109 Instruction: aesenc xmm, xmm
02,110 CPUID Flags: AES
02,111 Description
02,112 Perform one round of an AES encryption flow on data (state) in a using the round key in RoundKey, and store the result in dst."
02,113 Operation
02,114 a[127:0] := ShiftRows(a[127:0])
02,115 a[127:0] := SubBytes(a[127:0])
02,116 a[127:0] := MixColumns(a[127:0])
02,117 dst[127:0] := a[127:0] XOR RoundKey[127:0]
02,118
02,119 Performance
02,120 Architecture      Latency Throughput (CPI)
02,121 Skylake           4          1
02,122 Broadwell        7          1
02,123 Haswell          7          1
02,124 Ivy Bridge     8          1
02,125 */
02,126
02,127 /*
02,128 #include <stdlib.h>
02,129 #include <stdint.h> // uint64_t needed
02,130 #include <string.h> // memset
02,131 #include <smmmintrin.h> // SSE4.1 intrinsics
02,132 #include <wmmintrin.h>
02,133 void SlowCopy128bit (const char *SOURCE, char *TARGET) { __mm_storeu_si128((__m128i *) (TARGET), __mm_loadu_si128((const __m128i *) (SOURCE))); }
02,134 unsigned char DDAES[16];
02,135 void DoubleDeuceAES(const uint8_t *buffer, const size_t length) {
02,136     uint32_t i;
02,137     char MaxTo64a[64], MaxTo64b[64], MaxTo64c[64], MaxTo64d[64];
02,138     __m128i hashA = __mm_set_epi64x(0x6c62272e07bb0142, 0x62b821756295c58d); // 0x6c62272e07bb014262b821756295c58d // __mm_setzero_si128();
02,139     __m128i hashB = __mm_set_epi64x(0xdd268dbcaac55036, 0x2d98c384c4e576cc); // 0xdd268dbcaac550362d98c384c4e576cccc8b1536847b6bbb31023b4c8caee0535 // FNV offset basis
02,140     __m128i hashC = __mm_set_epi64x(0xc8b1536847b6bbb3, 0x1023b4c8caee0535); // 0xdd268dbcaac550362d98c384c4e576cccc8b1536847b6bbb31023b4c8caee0535 // FNV offset basis
02,141     __m128i hashD = __mm_setzero_si128();
02,142     const __m128i *ptr128a, *ptr128b, *ptr128c, *ptr128d;
02,143     memset(MaxTo64a, 0x33, 4*(128/8)); // padding the keys to be multiples of 128, up to 64 bytes
02,144     memset(MaxTo64b, 0x77, 4*(128/8)); // padding the keys to be multiples of 128, up to 64 bytes
02,145     for (i = 0; i < length; i++) {
02,146         MaxTo64a[i]=buffer[i];
02,147         MaxTo64b[63-i]=buffer[i]; // MaxTo64b[63-i]=MaxTo64a[i];
02,148     }
02,149     for (i = 0; i < (64>>1)/1; i++) { // 64/2/BYTE=31 i.e 0..31
02,150         MaxTo64c[(i<<1)+0]=MaxTo64a[i+0]; // a: 00,32 / 01,33 / ...31,63
02,151         MaxTo64c[(i<<1)+1]=MaxTo64a[i+32]; // c: 0*2+0,0*2+1 / 1*2+0,1*2+1 / 2*2+0,2*2+1 which is 0,1 / 2,3 / 4,5
02,152         MaxTo64d[(i<<1)+0]=MaxTo64b[i+0];
02,153         MaxTo64d[(i<<1)+1]=MaxTo64b[i+32];
02,154     }
02,155     ptr128a=(__m128i *)MaxTo64a;
02,156     ptr128b=(__m128i *)MaxTo64b;
02,157     ptr128c=(__m128i *)MaxTo64c;
02,158     ptr128d=(__m128i *)MaxTo64d;

```

```
02,159         for (i = 0; i < 64 / 16; i++) {
02,160             __m128i a = _mm_loadu_si128(ptr128a++);
02,161             __m128i b = _mm_loadu_si128(ptr128b++);
02,162             __m128i c = _mm_loadu_si128(ptr128c++);
02,163             __m128i d = _mm_loadu_si128(ptr128d++);
02,164             hashA = _mm_aesenc_si128(hashA, a);
02,165             hashB = _mm_aesenc_si128(hashB, b);
02,166             hashC = _mm_aesenc_si128(hashC, c);
02,167             hashD = _mm_aesenc_si128(hashD, d);
02,168         }
02,169         hashA = _mm_aesenc_si128(hashA, hashB);
02,170         hashA = _mm_aesenc_si128(hashA, hashC);
02,171         hashA = _mm_aesenc_si128(hashA, hashD);
02,172         SlowCopy128bit( (const char *)(&hashA), (char *)&DDAES[0]);
02,173     }
02,174     */
02,175
02,176     //static const uint8_t VectorsNeedNonVariable1[256] __attribute__((aligned(16))) =
02,177     static const uint8_t VectorsNeedNonVariable1[256] =
02,178     {
02,179         0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
02,180         0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
02,181         0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
02,182         0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
02,183         0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
02,184         0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
02,185         0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
02,186         0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
02,187         0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
02,188         0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
02,189         0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
02,190         0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00,
02,191         0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00,
02,192         0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00,
02,193         0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00,
02,194         0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x00,
02,195     };
02,196     static const __m128i *Mumbotron = (__m128i *) VectorsNeedNonVariable1;
02,197     //static const uint8_t VectorsNeedNonVariable2[256] __attribute__((aligned(16))) =
02,198     static const uint8_t VectorsNeedNonVariable2[256] =
02,199     {
02,200         0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
02,201         0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
02,202         0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
02,203         0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
02,204         0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
02,205         0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
02,206         0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
02,207         0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
02,208         0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
02,209         0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
02,210         0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
02,211         0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
02,212         0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
02,213         0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
02,214         0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
02,215         0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF,
02,216     };
```



```

02,217 static const __m128i *Jumbotron = (__m128i *) VectorsNeedNonVariable2;
02,218
02,219 // https://github.com/Cyan4973/xxHash/issues/568
02,220 // https://github.com/google/highwayhash/issues/102
02,221 // Revision 2021-Aug-28 [
02,222 // Written by Sarnmayce, inspired by J. Andrew Rogers's https://github.com/jandrewrogers/AquaHash/blob/master/aquahash.h
02,223 // This hash function serves two ... functions - useful for table lookups and to shrink keys (usually 64...256 bytes in length) down to 16 bytes:
02,224 // Its linear speed is quite good - 8+GB/s on Zen 2 Renoir 4.3GHz, DDR4 3200MHz.
02,225 // Some non-synthetic speed measurements:
02,226 /*
02,227 Testfile: KAZE(Dictionary_Specification_Language(ABBY Software_House))Hanyu_Cihai_new_Sea-of-Words(Zho-Zho).dsl (42,920,232 bytes)
02,228 Testmachine: Testmachine: laptop 'Brutalitto' AMD 4800H max turbo 4.3GHz, 64GB DDR4 3200MHz, Windows 10
02,229 Hashtable: 26bit, i.e. 67,108,864 slots, greater than (42,920,232 bytes), since in case of perfect hasher - slots should be more than the keys (could be all unique) at
each position
02,230
02,231 +-----+-----+-----+-----+
02,232 | Hasher,          | Number Of Hash Collisions = | RAW Hashing Speed (in one pass, | Linear Hashing Speed, |
02,233 | GCC-10.1 compiler | Distinct Keys -           | at each position) for keys      | the whole file as one key |
02,234 | -O3 -mavx        | Number Of Trees           | 4,6,8,10,12,14,16,18,36,64 bytes |
02,235 +-----+-----+-----+-----+
02,236 | XXH3_64bits v0.8.0 | 41,108,202 | 295,187,276 KEYS-PER-SECOND | 21,786,919,796 BYTES-PER-SECOND |
02,237 | HighwayHash128 (generic) | 41,109,295 | 5,986,502 KEYS-PER-SECOND | 1,644,642,372 BYTES-PER-SECOND |
02,238 | CRC32C (__mm_crc32_u32) | 41,109,478 | 274,426,023 KEYS-PER-SECOND | 5,241,205,519 BYTES-PER-SECOND |
02,239 | XXH3_128bits v0.8.0 | 41,111,196 | 214,493,903 KEYS-PER-SECOND | 20,331,706,300 BYTES-PER-SECOND |
02,240 | SHA3-224           | 41,111,291 | 153,854 KEYS-PER-SECOND | 22,319,413 BYTES-PER-SECOND |
02,241 | wyhash final       | 41,112,870 | 449,897,589 KEYS-PER-SECOND | 15,086,197,539 BYTES-PER-SECOND |
02,242 | DoubleDeuceAES_Gumbotron | 41,117,352 | 204,869,832 KEYS-PER-SECOND | 8,690,065,195 BYTES-PER-SECOND |
02,243 | FNV1A_Pippip       | 41,488,327 | 449,897,589 KEYS-PER-SECOND | 8,101,214,043 BYTES-PER-SECOND |
02,244 +-----+-----+-----+-----+
02,245 Note1: The second column houses the cumulative value for all collisions, the collisions for all orders 4..64 were summed, that is.
02,246 Note2: Folding of those 128bits should lessen the collisions.
02,247
02,248 Testfile: TERAPIG_Encyclopaedia_Judaica (in_22_volumes)_TXT.tar (107,784,192 bytes)
02,249 Testmachine: Testmachine: laptop 'Brutalitto' AMD 4800H max turbo 4.3GHz, 64GB DDR4 3200MHz, Windows 10
02,250 Hashtable: 27bit, i.e. 134,217,728 slots, greater than (107,784,192 bytes), since in case of perfect hasher - slots should be more than the keys (could be all unique)
at each position
02,251
02,252 +-----+-----+-----+-----+
02,253 | Hasher,          | Number Of Hash Collisions = | RAW Hashing Speed (in one pass, | Linear Hashing Speed, |
02,254 | GCC-10.1 compiler | Distinct Keys -           | at each position) for keys      | the whole file as one key |
02,255 | -O3 -mavx        | Number Of Trees           | 4,6,8,10,12,14,16,18,36,64 bytes |
02,256 +-----+-----+-----+-----+
02,257 | DoubleDeuceAES_Gumbotron | 135,752,271 | 204,640,573 KEYS-PER-SECOND | 8,742,330,440 BYTES-PER-SECOND |
02,258 | HighwayHash128 (generic) | 135,754,873 | 6,336,146 KEYS-PER-SECOND | 1,435,801,622 BYTES-PER-SECOND |
02,259 | XXH3_128bits v0.8.0 | 135,756,978 | 212,843,977 KEYS-PER-SECOND | 22,539,563,362 BYTES-PER-SECOND |
02,260 | wyhash final       | 135,762,454 | 442,100,861 KEYS-PER-SECOND | 14,959,638,029 BYTES-PER-SECOND |
02,261 | XXH3_64bits v0.8.0 | 135,763,366 | 290,994,033 KEYS-PER-SECOND | 22,464,400,166 BYTES-PER-SECOND |
02,262 | CRC32C (__mm_crc32_u32) | 135,764,628 | 252,599,460 KEYS-PER-SECOND | 5,241,402,061 BYTES-PER-SECOND |
02,263 | FNV1A_Pippip       | 135,768,302 | 450,602,801 KEYS-PER-SECOND | 8,048,401,433 BYTES-PER-SECOND |
02,264 | SHA3-224           | 135,771,905 | 153,841 KEYS-PER-SECOND | 22,246,479 BYTES-PER-SECOND |
02,265 +-----+-----+-----+-----+
02,266 The part I use is from https://github.com/google/highwayhash/tree/master/c
02,267 */
02,268 // // https://godbolt.org/ [[[
02,269 // #include <stdlib.h>
02,270 // #include <stdint.h> // uint64_t needed
02,271 // #include <string.h> // memset
02,272 // #include <smmintrin.h> // SSE4.1 intrinsics

```

```

02,273 // #include <mmmintrin.h>
02,274 // void SlowCopy128bit (const char *SOURCE, char *TARGET) { _mm_storeu_si128((__m128i *) (TARGET), _mm_loadu_si128((const __m128i *) (SOURCE))); }
02,275 // unsigned char DDAES[16];
02,276 // // https://godbolt.org/ ]]]
02,277 // Collision Benchmark - DoubleDeuceAES_128bits versus XXH3_64bits v0.8.0
02,278 //
02,279 // Testset: "A billion Knight-Tours variants (each KT with 256 variants, the KT itself omitted) - each 128 bytes long"
02,280 // Testfile: 1000000000.KnightTours.txt (130,000,000,000 bytes)
02,281 //
02,282 // The name of the game - hashing all lines and taking either 5 bytes or 6,7,8 bytes from the hash.
02,283 //
02,284 // ```
02,285 // +-----+-----+
02,286 // | Hasher                | Collisions within first 5 bytes |
02,287 // +-----+-----+
02,288 // | XXH3_64bits v0.8.0    | 1,000,000,000 - 999,545,727 distinct lines =    454,273 |
02,289 // +-----+-----+
02,290 // | DoubleDeuceAES_128bits | 1,000,000,000 - 999,545,796 distinct lines =    454,204 |
02,291 // +-----+-----+
02,292 //
02,293 // +-----+-----+
02,294 // | Hasher                | Collisions within first 6 bytes |
02,295 // +-----+-----+
02,296 // | XXH3_64bits v0.8.0    | 1,000,000,000 - 999,998,214 distinct lines =      1,786 |
02,297 // +-----+-----+
02,298 // | DoubleDeuceAES_128bits | 1,000,000,000 - 999,998,213 distinct lines =      1,787 |
02,299 // +-----+-----+
02,300 //
02,301 // +-----+-----+
02,302 // | Hasher                | Collisions within first 7 bytes |
02,303 // +-----+-----+
02,304 // | XXH3_64bits v0.8.0    | 1,000,000,000 - 999,999,989 distinct lines =         11 |
02,305 // +-----+-----+
02,306 // | DoubleDeuceAES_128bits | 1,000,000,000 - 999,999,994 distinct lines =          6 |
02,307 // +-----+-----+
02,308 //
02,309 // +-----+-----+
02,310 // | Hasher                | Collisions within first 8 bytes |
02,311 // +-----+-----+
02,312 // | XXH3_64bits v0.8.0    | 1,000,000,000 - 1,000,000,000 distinct lines =          0 |
02,313 // +-----+-----+
02,314 // | DoubleDeuceAES_128bits | 1,000,000,000 - 1,000,000,000 distinct lines =          0 |
02,315 // +-----+-----+
02,316 // ```
02,317 //
02,318 // The benchmark package (allowing to reproduce all the stuff):
02,319 // www.sarnmayce.com/COLLISION_Hashliner.zip
02,320 //
02,321 // This is how the console looks like:
02,322 //
02,323 // ```
02,324 // C:\test\COLLISION_Hashliner>GENERATE_Xmillion_Knight-Tours.bat 1000000000
02,325 // Generating 1000000000 Knight-Tours and dumping them into file ...
02,326 //
02,327 // C:\test\COLLISION_Hashliner>Knight-Tour_FNV1A_YoshimitsuTRIADii_vs_CRC32_TRISMUS.exe a8 1000000000 1>1000000000.KnightTours.txt
02,328 //
02,329 // C:\test\COLLISION_Hashliner>bench7.bat 1000000000.KnightTours.txt
02,330 //

```

```

02,331 // C:\test\COLLISION_Hashliner>Hashliner_XXH3_dump7byteshash.exe 1000000000.KnightTours.txt 1>1000000000.KnightTours.txt.xhx3.txt
02,332 //
02,333 // C:\test\COLLISION_Hashliner>Hashliner_DDAES_dump7byteshash.exe 1000000000.KnightTours.txt 1>1000000000.KnightTours.txt.DDAES.txt
02,334 //
02,335 // C:\test\COLLISION_Hashliner>Sandokan_QuickSortExternal_Deduplicated_4+GB_64bit_Intel.exe 1000000000.KnightTours.txt.xhx3.txt /fast /descend 3000
02,336 // Sandokan_QuickSortExternal_4+GB r.3+, written by Kaze, using Bill Durango's Quicksort source.
02,337 // Size of input file: 16,000,000,000
02,338 // Counting lines ...
02,339 // Lines encountered: 1,000,000,000
02,340 // Longest line (including CR if present): 15
02,341 // Allocated memory for pointers-to-lines in MB: 7629
02,342 // Assigning pointers ...
02,343 // sizeof(int), sizeof(void*): 4, 8
02,344 // Trying to allocate memory for the file itself in MB: 15258 ... OK! Get on with fast internal accesses.
02,345 // Uploading ...
02,346 // Sorting 1,000,000,000 Pointers ...
02,347 // Quicksort (Insertionsort for small blocks) commenced ...
02,348 // / RightEnd: 000,328,304,267; NumberOfSplittings: 0,114,284,204; Done: 100% ...
02,349 // NumberOfComparisons: 34,310,536,510
02,350 // The time to sort 1,000,000,000 items via Quicksort+Insertionsort was 2,848,402 clocks.
02,351 // Performance: 12,045,534 Comparisons_128B_long-Per-Second i.e 24,091,068 RandomReads_128B_long-Per-Second.
02,352 // Dumping the sorted data (Regime=2)...
02,353 // \ Done 100% ...
02,354 // Dumped 1,000,000,000 lines.
02,355 // OK! Incoming and resultant file's sizes match.
02,356 // Dumping the sorted data [deduplicated] ...
02,357 // Dumped 999,999,989 distinct lines.
02,358 // Dump time: 460,940 clocks.
02,359 // Total time: 3,347,265 clocks.
02,360 // Performance: 4,780 bytes/clock.
02,361 // Done successfully.
02,362 //
02,363 // C:\test\COLLISION_Hashliner>sort /R QuickSortExternal_4+GB.distinct.txt 1>1000000000.KnightTours.txt.xhx3.7bytes.2orABOVE.txt
02,364 //
02,365 // C:\test\COLLISION_Hashliner>Sandokan_QuickSortExternal_Deduplicated_4+GB_64bit_Intel.exe 1000000000.KnightTours.txt.DDAES.txt /fast /descend 3000
02,366 // Sandokan_QuickSortExternal_4+GB r.3+, written by Kaze, using Bill Durango's Quicksort source.
02,367 // Size of input file: 16,000,000,000
02,368 // Counting lines ...
02,369 // Lines encountered: 1,000,000,000
02,370 // Longest line (including CR if present): 15
02,371 // Allocated memory for pointers-to-lines in MB: 7629
02,372 // Assigning pointers ...
02,373 // sizeof(int), sizeof(void*): 4, 8
02,374 // Trying to allocate memory for the file itself in MB: 15258 ... OK! Get on with fast internal accesses.
02,375 // Uploading ...
02,376 // Sorting 1,000,000,000 Pointers ...
02,377 // Quicksort (Insertionsort for small blocks) commenced ...
02,378 // - RightEnd: 000,759,555,061; NumberOfSplittings: 0,114,282,509; Done: 100% ...
02,379 // NumberOfComparisons: 34,551,039,764
02,380 // The time to sort 1,000,000,000 items via Quicksort+Insertionsort was 2,896,271 clocks.
02,381 // Performance: 11,929,487 Comparisons_128B_long-Per-Second i.e 23,858,974 RandomReads_128B_long-Per-Second.
02,382 // Dumping the sorted data (Regime=2)...
02,383 // \ Done 100% ...
02,384 // Dumped 1,000,000,000 lines.
02,385 // OK! Incoming and resultant file's sizes match.
02,386 // Dumping the sorted data [deduplicated] ...
02,387 // Dumped 999,999,994 distinct lines.
02,388 // Dump time: 458,406 clocks.

```

```

02,389 // Total time: 3,393,196 clocks.
02,390 // Performance: 4,715 bytes/clock.
02,391 // Done successfully.
02,392 //
02,393 // C:\test\COLLISION_Hashliner>sort /R QuickSortExternal_4+GB.distinct.txt 1>1000000000.KnightTours.txt.DDAES.7bytes.2orABOVE.txt
02,394 //
02,395 // C:\test\COLLISION_Hashliner>dir *7b*
02,396 //
02,397 // 15-Aug-21 12:28          156 1000000000.KnightTours.txt.DDAES.7bytes.2orABOVE.txt
02,398 // 15-Aug-21 11:32          286 1000000000.KnightTours.txt.xzh3.7bytes.2orABOVE.txt
02,399 //
02,400 // C:\test\COLLISION_Hashliner>type 1000000000.KnightTours.txt.xzh3.7bytes.2orABOVE.txt
02,401 // 0,000,002      f84627e722e85e
02,402 // 0,000,002      f0039d0c4e4fce
02,403 // 0,000,002      c87c64d97df0e7
02,404 // 0,000,002      bb4344a5546572
02,405 // 0,000,002      af2f628f4b3ffb
02,406 // 0,000,002      a8cb8675c94610
02,407 // 0,000,002      a742cf83948622
02,408 // 0,000,002      657cb9dff2d962
02,409 // 0,000,002      436aef7ab54ce7
02,410 // 0,000,002      270fcde0563670
02,411 // 0,000,002      0b533d70915c51
02,412 //
02,413 // C:\test\COLLISION_Hashliner>
02,414 // ``
02,415 void DoubleDeuceAES_Gumbotron(const uint8_t *buffer, size_t length) {
02,416     size_t i, Cycles;
02,417     __m128i hashA = _mm_set_epi64x(0x6c62272e07bb0142, 0x62b821756295c58d); // 0x6c62272e07bb014262b821756295c58d // _mm_setzero_si128();
02,418     __m128i hashB = _mm_set_epi64x(0xdd268dbcaac55036, 0x2d98c384c4e576cc); // 0xdd268dbcaac550362d98c384c4e576ccc8b1536847b6bbb31023b4c8caee0535 // FNV offset basis
02,419     __m128i hashC = _mm_set_epi64x(0xc8b1536847b6bbb3, 0x1023b4c8caee0535); // 0xdd268dbcaac550362d98c384c4e576ccc8b1536847b6bbb31023b4c8caee0535 // FNV offset basis
02,420     __m128i hashD = _mm_setzero_si128();
02,421     __m128i a0,a1,a2,a3; // Instead of this chunkenization, ZMM houses the 4 XMMs, if there is shuffle across all the 512bits, use it. There is, but __m256i
    __mm256_shuffle_epi8(__m256i a, __m256i b) is more handy.
02,422     __m128i b0,b1,b2,b3;
02,423     __m128i c0,c1,c2,c3;
02,424     __m128i d0,d1,d2,d3;
02,425     __m128i tmp0,tmp1,tmp2,tmp3;
02,426     __m128i ReverseMask = _mm_set_epi8(0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15);
02,427     __m128i PartialInterleavingMask1 = _mm_set_epi8(0x80,7,0x80,6,0x80,5,0x80,4,0x80,3,0x80,2,0x80,1,0x80,0);
02,428     __m128i PartialInterleavingMask2 = _mm_set_epi8(0x80,0xf,0x80,0xe,0x80,0xd,0x80,0xc,0x80,0xb,0x80,0xa,0x80,9,0x80,8);
02,429     __m128i PartialInterleavingMask3 = _mm_set_epi8(7,0x80,6,0x80,5,0x80,4,0x80,3,0x80,2,0x80,1,0x80,0,0x80);
02,430     __m128i PartialInterleavingMask4 = _mm_set_epi8(0xf,0x80,0xe,0x80,0xd,0x80,0xc,0x80,0xb,0x80,0xa,0x80,9,0x80,8,0x80);
02,431     const __m128i *ptr128a, *ptr128b, *ptr128c, *ptr128d;
02,432
02,433     __m128i AgainstRules, GumbotronREVER, GumbotronINTER, Gumbotron, GumbotronREVERINTER;
02,434     const __m128i *ptr128;
02,435     __m128i InterleaveMask = _mm_set_epi8(15,7,14,6,13,5,12,4,11,3,10,2,9,1,8,0);
02,436
02,437     if (length >= 64) {
02,438         Cycles = length/64;
02,439         for(;; Cycles--; buffer += 64) {
02,440             a0 = _mm_loadu_si128((__m128i *) (buffer+0*16));
02,441             a1 = _mm_loadu_si128((__m128i *) (buffer+1*16));
02,442             a2 = _mm_loadu_si128((__m128i *) (buffer+2*16));
02,443             a3 = _mm_loadu_si128((__m128i *) (buffer+3*16));
02,444             b0 = _mm_shuffle_epi8 (a3, ReverseMask);
02,445             b1 = _mm_shuffle_epi8 (a2, ReverseMask);

```

```
02,446     b2 = _mm_shuffle_epi8 (a1, ReverseMask);
02,447     b3 = _mm_shuffle_epi8 (a0, ReverseMask);
02,448     tmp0 = _mm_shuffle_epi8 (a0, PartialInterleavingMask1);
02,449     tmp1 = _mm_shuffle_epi8 (a0, PartialInterleavingMask2);
02,450     tmp2 = _mm_shuffle_epi8 (a2, PartialInterleavingMask3);
02,451     tmp3 = _mm_shuffle_epi8 (a2, PartialInterleavingMask4);
02,452     c0 = _mm_or_si128 (tmp0, tmp2);
02,453     c1 = _mm_or_si128 (tmp1, tmp3);
02,454     tmp0 = _mm_shuffle_epi8 (a1, PartialInterleavingMask1);
02,455     tmp1 = _mm_shuffle_epi8 (a1, PartialInterleavingMask2);
02,456     tmp2 = _mm_shuffle_epi8 (a3, PartialInterleavingMask3);
02,457     tmp3 = _mm_shuffle_epi8 (a3, PartialInterleavingMask4);
02,458     c2 = _mm_or_si128 (tmp0, tmp2);
02,459     c3 = _mm_or_si128 (tmp1, tmp3);
02,460     tmp0 = _mm_shuffle_epi8 (b0, PartialInterleavingMask1);
02,461     tmp1 = _mm_shuffle_epi8 (b0, PartialInterleavingMask2);
02,462     tmp2 = _mm_shuffle_epi8 (b2, PartialInterleavingMask3);
02,463     tmp3 = _mm_shuffle_epi8 (b2, PartialInterleavingMask4);
02,464     d0 = _mm_or_si128 (tmp0, tmp2);
02,465     d1 = _mm_or_si128 (tmp1, tmp3);
02,466     tmp0 = _mm_shuffle_epi8 (b1, PartialInterleavingMask1);
02,467     tmp1 = _mm_shuffle_epi8 (b1, PartialInterleavingMask2);
02,468     tmp2 = _mm_shuffle_epi8 (b3, PartialInterleavingMask3);
02,469     tmp3 = _mm_shuffle_epi8 (b3, PartialInterleavingMask4);
02,470     d2 = _mm_or_si128 (tmp0, tmp2);
02,471     d3 = _mm_or_si128 (tmp1, tmp3);
02,472
02,473     hashA = _mm_aesenc_si128(hashA, a0);
02,474     hashB = _mm_aesenc_si128(hashB, b0);
02,475     hashC = _mm_aesenc_si128(hashC, c0);
02,476     hashD = _mm_aesenc_si128(hashD, d0);
02,477
02,478     hashA = _mm_aesenc_si128(hashA, a1);
02,479     hashB = _mm_aesenc_si128(hashB, b1);
02,480     hashC = _mm_aesenc_si128(hashC, c1);
02,481     hashD = _mm_aesenc_si128(hashD, d1);
02,482
02,483     hashA = _mm_aesenc_si128(hashA, a2);
02,484     hashB = _mm_aesenc_si128(hashB, b2);
02,485     hashC = _mm_aesenc_si128(hashC, c2);
02,486     hashD = _mm_aesenc_si128(hashD, d2);
02,487
02,488     hashA = _mm_aesenc_si128(hashA, a3);
02,489     hashB = _mm_aesenc_si128(hashB, b3);
02,490     hashC = _mm_aesenc_si128(hashC, c3);
02,491     hashD = _mm_aesenc_si128(hashD, d3);
02,492
02,493     hashA = _mm_aesenc_si128(hashA, hashB);
02,494     hashA = _mm_aesenc_si128(hashA, hashC);
02,495     hashA = _mm_aesenc_si128(hashA, hashD);
02,496     length = length - 64;
02,497 }
02,498 }
02,499
02,500 ptr128 = (__m128i *)buffer;
02,501 if (length >= 16) {
02,502     Cycles = length/16;
02,503     for(;; Cycles--; buffer += 16) {
```

```

02,504         AgainstRules = _mm_loadu_si128(ptr128++);
02,505         GumbotronREVER = _mm_shuffle_epi8 (AgainstRules, ReverseMask);
02,506         GumbotronINTER = _mm_shuffle_epi8 (AgainstRules, InterleaveMask);
02,507         GumbotronREVERINTER = _mm_shuffle_epi8 (GumbotronREVER, InterleaveMask);
02,508         hashA = _mm_aesenc_si128(hashA, AgainstRules);
02,509         hashB = _mm_aesenc_si128(hashB, GumbotronREVER);
02,510         hashC = _mm_aesenc_si128(hashC, GumbotronINTER);
02,511         hashD = _mm_aesenc_si128(hashD, GumbotronREVERINTER);
02,512         hashA = _mm_aesenc_si128(hashA, hashB);
02,513         hashA = _mm_aesenc_si128(hashA, hashC);
02,514         hashA = _mm_aesenc_si128(hashA, hashD);
02,515         length = length - 16;
02,516     }
02,517 }
02,518 // Inhere using Pippip's approach to read past the end ("the dirty" sentinel like style, or more like padding):
02,519 if (length&(16-1)) {
02,520     AgainstRules = _mm_loadu_si128(ptr128);
02,521     //AgainstRules = _mm_srli_si128 (AgainstRules, 16-length); // catastrophic error: Intrinsic parameter must be an immediate value
02,522     AgainstRules = _mm_and_si128 (AgainstRules, Gumbotron[length]);
02,523     //Gumbotron = _mm_slli_si128 (Gumbotron, 16-length); // catastrophic error: Intrinsic parameter must be an immediate value
02,524     Gumbotron = _mm_and_si128 (hashB, Gumbotron[length]);
02,525     AgainstRules = _mm_or_si128 (AgainstRules, Gumbotron);
02,526
02,527     GumbotronREVER = _mm_shuffle_epi8 (AgainstRules, ReverseMask);
02,528     GumbotronINTER = _mm_shuffle_epi8 (AgainstRules, InterleaveMask);
02,529     GumbotronREVERINTER = _mm_shuffle_epi8 (GumbotronREVER, InterleaveMask);
02,530     hashA = _mm_aesenc_si128(hashA, AgainstRules);
02,531     hashB = _mm_aesenc_si128(hashB, GumbotronREVER);
02,532     hashC = _mm_aesenc_si128(hashC, GumbotronINTER);
02,533     hashD = _mm_aesenc_si128(hashD, GumbotronREVERINTER);
02,534     hashA = _mm_aesenc_si128(hashA, hashB);
02,535     hashA = _mm_aesenc_si128(hashA, hashC);
02,536     hashA = _mm_aesenc_si128(hashA, hashD);
02,537 }
02,538 SlowCopy128bit( (const char *)(&hashA), (char *)&DDAES[0]);
02,539 }
02,540 /*
02,541 ; mark_description "Intel(R) C++ Compiler XE for applications running on Intel(R) 64, Version 15.0.0.108 Build 20140726";
02,542 ; mark_description "-O3 -arch:avx -FA -D_WIN32_ENVIRONMENT -D_N_DDAES";
02,543
02,544 DoubleDeuceAES_Gumbotron PROC
02,545     sub     rsp, 216
02,546     mov     r8, rdx
02,547     vmovups xmm3, XMMWORD PTR [_2i10floatpacket.0]
02,548     lea     rdx, QWORD PTR [__ImageBase]
02,549     vmovups xmm4, XMMWORD PTR [_2i10floatpacket.1]
02,550     vpxor   xmm2, xmm2, xmm2
02,551     vmovups xmm5, XMMWORD PTR [_2i10floatpacket.2]
02,552     vmovdqu xmm1, XMMWORD PTR [_2i10floatpacket.4]
02,553     vmovdqu xmm0, XMMWORD PTR [_2i10floatpacket.5]
02,554     cmp     r8, 64
02,555     jbe     .B6.6
02,556     mov     rax, r8
02,557     shr     rax, 6
02,558     dec     rax
02,559     cmp     rax, -1
02,560     je      .B6.6
02,561     vmovups XMMWORD PTR [64+rsp], xmm6

```

```
02,562      vmovups   XMMWORD PTR [48+rsp], xmm7
02,563      vmovups   XMMWORD PTR [32+rsp], xmm8
02,564      vmovups   XMMWORD PTR [80+rsp], xmm9
02,565      vmovups   XMMWORD PTR [96+rsp], xmm10
02,566      vmovups   XMMWORD PTR [112+rsp], xmm11
02,567      vmovups   XMMWORD PTR [128+rsp], xmm12
02,568      vmovups   XMMWORD PTR [144+rsp], xmm13
02,569      vmovups   XMMWORD PTR [160+rsp], xmm14
02,570      vmovups   XMMWORD PTR [176+rsp], xmm15
02,571      .B6.4::
02,572      vmovdqu   xmm10, XMMWORD PTR [48+rcx]
02,573      vmovdqu   xmm6, XMMWORD PTR [32+rcx]
02,574      vmovdqu   xmm11, XMMWORD PTR [16+rcx]
02,575      vmovdqu   xmm12, XMMWORD PTR [rcx]
02,576      vmovdqu   xmm14, XMMWORD PTR [_2il0floatpacket.3]
02,577      dec       rax
02,578      vpshufb   xmm8, xmm10, xmm14
02,579      vpshufb   xmm13, xmm6, xmm14
02,580      vpshufb   xmm9, xmm11, xmm14
02,581      vpshufb   xmm14, xmm12, xmm14
02,582      vpshufb   xmm15, xmm12, xmm1
02,583      vaesenc   xmm4, xmm4, xmm8
02,584      add       r8, -64
02,585      vaesenc   xmm7, xmm4, xmm13
02,586      vaesenc   xmm4, xmm7, xmm9
02,587      vaesenc   xmm7, xmm4, xmm14
02,588      vmovdqu   xmm4, XMMWORD PTR [_2il0floatpacket.6]
02,589      vmovups   XMMWORD PTR [192+rsp], xmm7
02,590      vpshufb   xmm7, xmm6, xmm4
02,591      vpor       xmm15, xmm15, xmm7
02,592      vmovdqu   xmm7, XMMWORD PTR [_2il0floatpacket.7]
02,593      vaesenc   xmm15, xmm5, xmm15
02,594      vpshufb   xmm5, xmm12, xmm0
02,595      vpshufb   xmm6, xmm6, xmm7
02,596      vpor       xmm5, xmm5, xmm6
02,597      vaesenc   xmm15, xmm15, xmm5
02,598      vpshufb   xmm6, xmm11, xmm1
02,599      vpshufb   xmm5, xmm10, xmm4
02,600      vpor       xmm6, xmm6, xmm5
02,601      vaesenc   xmm5, xmm15, xmm6
02,602      vpshufb   xmm15, xmm11, xmm0
02,603      vpshufb   xmm6, xmm10, xmm7
02,604      vpor       xmm15, xmm15, xmm6
02,605      vaesenc   xmm5, xmm5, xmm15
02,606      vpshufb   xmm6, xmm8, xmm1
02,607      vpshufb   xmm15, xmm9, xmm4
02,608      vpshufb   xmm8, xmm8, xmm0
02,609      vpshufb   xmm9, xmm9, xmm7
02,610      vpshufb   xmm4, xmm14, xmm4
02,611      vpor       xmm6, xmm6, xmm15
02,612      vaesenc   xmm2, xmm2, xmm6
02,613      vpor       xmm6, xmm8, xmm9
02,614      vaesenc   xmm6, xmm2, xmm6
02,615      vpshufb   xmm2, xmm13, xmm1
02,616      vpshufb   xmm13, xmm13, xmm0
02,617      vpor       xmm2, xmm2, xmm4
02,618      vaesenc   xmm8, xmm6, xmm2
02,619      vpshufb   xmm2, xmm14, xmm7
```

```
02,620      vaesenc    xmm3, xmm3, xmm12
02,621      vpor      xmm4, xmm13, xmm2
02,622      vaesenc    xmm2, xmm8, xmm4
02,623      vaesenc    xmm4, xmm3, xmm11
02,624      vaesenc    xmm3, xmm4, XMMWORD PTR [32+rcx]
02,625      add       rcx, 64
02,626      vaesenc    xmm6, xmm3, xmm10
02,627      vmovups    xmm4, XMMWORD PTR [192+rsp]
02,628      vaesenc    xmm7, xmm6, xmm4
02,629      vaesenc    xmm8, xmm7, xmm5
02,630      vaesenc    xmm3, xmm8, xmm2
02,631      cmp       rax, -1
02,632      jne       .B6.4
02,633      vmovups    xmm6, XMMWORD PTR [64+rsp]
02,634      vmovups    xmm7, XMMWORD PTR [48+rsp]
02,635      vmovups    xmm8, XMMWORD PTR [32+rsp]
02,636      vmovups    xmm9, XMMWORD PTR [80+rsp]
02,637      vmovups    xmm10, XMMWORD PTR [96+rsp]
02,638      vmovups    xmm11, XMMWORD PTR [112+rsp]
02,639      vmovups    xmm12, XMMWORD PTR [128+rsp]
02,640      vmovups    xmm13, XMMWORD PTR [144+rsp]
02,641      vmovups    xmm14, XMMWORD PTR [160+rsp]
02,642      vmovups    xmm15, XMMWORD PTR [176+rsp]
02,643 .B6.6::
02,644      cmp       r8, 16
02,645      jb        .B6.11
02,646      mov       rax, r8
02,647      shr       rax, 4
02,648      dec       rax
02,649      cmp       rax, -1
02,650      je        .B6.11
02,651      vmovups    XMMWORD PTR [64+rsp], xmm6
02,652      vmovups    XMMWORD PTR [48+rsp], xmm7
02,653      vmovups    XMMWORD PTR [32+rsp], xmm8
02,654      vmovdqu    xmm0, XMMWORD PTR [_2il0floatpacket.8]
02,655      vmovdqu    xmm6, XMMWORD PTR [_2il0floatpacket.3]
02,656      ALIGN     16
02,657 .B6.9::
02,658      vmovdqu    xmm1, XMMWORD PTR [rcx]
02,659      vpshufb    xmm8, xmm1, xmm6
02,660      vpshufb    xmm7, xmm1, xmm0
02,661      vaesenc    xmm5, xmm5, xmm7
02,662      dec       rax
02,663      vpshufb    xmm7, xmm8, xmm0
02,664      vaesenc    xmm4, xmm4, xmm8
02,665      add       rcx, 16
02,666      vaesenc    xmm3, xmm3, xmm1
02,667      add       r8, -16
02,668      vaesenc    xmm1, xmm3, xmm4
02,669      vaesenc    xmm2, xmm2, xmm7
02,670      vaesenc    xmm3, xmm1, xmm5
02,671      vaesenc    xmm3, xmm3, xmm2
02,672      cmp       rax, -1
02,673      jne       .B6.9
02,674      vmovups    xmm6, XMMWORD PTR [64+rsp]
02,675      vmovups    xmm7, XMMWORD PTR [48+rsp]
02,676      vmovups    xmm8, XMMWORD PTR [32+rsp]
02,677 .B6.11::
```



```
02,678      test      r8, 15
02,679      je       .B6.13
02,680      shl      r8, 4
02,681      vmovdqu   xmm1, XMMWORD PTR [rcx]
02,682      vpand     xmm0, xmm1, XMMWORD PTR [imagerel(VectorsNeedNonVariable1)+rdx+r8]
02,683      vpand     xmm1, xmm4, XMMWORD PTR [imagerel(VectorsNeedNonVariable2)+rdx+r8]
02,684      vpor      xmm0, xmm0, xmm1
02,685      vpshufb   xmm1, xmm0, XMMWORD PTR [_2il0floatpacket.3]
02,686      vaesenc   xmm3, xmm3, xmm0
02,687      vaesenc   xmm4, xmm4, xmm1
02,688      vaesenc   xmm4, xmm3, xmm4
02,689      vmovdqu   xmm3, XMMWORD PTR [_2il0floatpacket.8]
02,690      vpshufb   xmm0, xmm0, xmm3
02,691      vpshufb   xmm1, xmm1, xmm3
02,692      vaesenc   xmm5, xmm5, xmm0
02,693      vaesenc   xmm0, xmm4, xmm5
02,694      vaesenc   xmm2, xmm2, xmm1
02,695      vaesenc   xmm3, xmm0, xmm2
02,696 .B6.13::
02,697      vmovups    XMMWORD PTR [DDAES], xmm3
02,698      add       rsp, 216
02,699      ret
02,700      ALIGN     16
02,701 DoubleDeuceAES_Gumbotron ENDP
02,702 */
02,703
02,704 // Revision 2021-Aug-22 [
02,705 //VPShufB: __m256i __mm256_shuffle_epi8(__m256i a, __m256i b)
02,706 //VPShufB __m512i __mm512_shuffle_epi8(__m512i a, __m512i b);
02,707 void DoubleDeuceAES_Gumbotron_YMM(const uint8_t *buffer, size_t length) {
02,708     size_t i, Cycles;
02,709     __m128i hashA = __mm_set_epi64x(0x6c62272e07bb0142, 0x62b821756295c58d); // 0x6c62272e07bb014262b821756295c58d // __mm_setzero_si128();
02,710     __m128i hashB = __mm_set_epi64x(0xdd268dbcaac550362d98c384c4e576ccc8b1536847b6bbb31023b4c8caee0535, 0xdd268dbcaac550362d98c384c4e576ccc8b1536847b6bbb31023b4c8caee0535 // FNV offset basis
02,711     __m128i hashC = __mm_set_epi64x(0xc8b1536847b6bbb3, 0x1023b4c8caee0535); // 0xdd268dbcaac550362d98c384c4e576ccc8b1536847b6bbb31023b4c8caee0535 // FNV offset basis
02,712     __m128i hashD = __mm_setzero_si128();
02,713     __m128i a0,a1,a2,a3; // Instead of this chunkenization, ZMM houses the 4 XMMs, if there is shuffle across all the 512bits, use it. There is, but __m256i
02,714     __mm256_shuffle_epi8(__m256i a, __m256i b) is more handy.
02,715     __m256i a0YMM,a2YMM;
02,716     __m128i b0,b1,b2,b3;
02,717     __m256i b0YMM,b2YMM;
02,718     __m128i c0,c1,c2,c3;
02,719     __m256i c0YMM,c2YMM;
02,720     __m128i d0,d1,d2,d3;
02,721     __m256i d0YMM,d2YMM;
02,722     __m128i tmp0,tmp1,tmp2,tmp3;
02,723     __m256i tmp0YMM,tmp2YMM;
02,724     __m128i ReverseMask = __mm_set_epi8(0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15);
02,725     __m256i ReverseMaskYMM = __mm256_set_epi8(0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31);
02,726     __m128i PartialInterleavingMask1 = __mm_set_epi8(0x80,7,0x80,6,0x80,5,0x80,4,0x80,3,0x80,2,0x80,1,0x80,0);
02,727     __m128i PartialInterleavingMask2 = __mm_set_epi8(0x80,0xf,0x80,0xe,0x80,0xd,0x80,0xc,0x80,0xb,0x80,0xa,0x80,9,0x80,8);
02,728     __m128i PartialInterleavingMask3 = __mm_set_epi8(7,0x80,6,0x80,5,0x80,4,0x80,3,0x80,2,0x80,1,0x80,0,0x80);
02,729     __m128i PartialInterleavingMask4 = __mm_set_epi8(0xf,0x80,0xe,0x80,0xd,0x80,0xc,0x80,0xb,0x80,0xa,0x80,9,0x80,8,0x80);
02,730     __mm256_set_epi8(0x80,0xf,0x80,0xe,0x80,0xd,0x80,0xc,0x80,0xb,0x80,0xa,0x80,9,0x80,8,0x80,7,0x80,6,0x80,5,0x80,4,0x80,3,0x80,2,0x80,1,0x80,0);
02,731     __m256i PartialInterleavingMask2YMM =
__mm256_set_epi8(0x80,0xf+16,0x80,0xe+16,0x80,0xd+16,0x80,0xc+16,0x80,0xb+16,0x80,0xa+16,0x80,9+16,0x80,8+16,0x80,7+16,0x80,6+16,0x80,5+16,0x80,4+16,0x80,3+16,0x80,2+16,0x80,1+16,0x80,0+16);
02,732     __m256i PartialInterleavingMask3YMM =
```

```

_mm256_set_epi8(0xf,0x80,0xe,0x80,0xd,0x80,0xc,0x80,0xb,0x80,0xa,0x80,9,0x80,8,0x80,7,0x80,6,0x80,5,0x80,4,0x80,3,0x80,2,0x80,1,0x80,0,0x80);
02,732 // __m256i PartialInterleavingMask4YMM =
_mm256_set_epi8(0xf+16,0x80,0xe+16,0x80,0xd+16,0x80,0xc+16,0x80,0xb+16,0x80,0xa+16,0x80,9+16,0x80,8+16,0x80,7+16,0x80,6+16,0x80,5+16,0x80,4+16,0x80,3+16,0x80,2+16,0x80,1+16,0
x80,0+16,0x80);
02,733 const __m128i *ptr128a, *ptr128b, *ptr128c, *ptr128d;
02,734
02,735 __m128i AgainstRules, GumbotronREVER, GumbotronINTER, Gumbotron, GumbotronREVERINTER;
02,736 const __m128i *ptr128;
02,737 __m128i InterleaveMask = _mm_set_epi8(15,7,14,6,13,5,12,4,11,3,10,2,9,1,8,0);
02,738 uint8_t vector[32];
02,739
02,740 if (length >= 64) {
02,741     Cycles = length/64;
02,742     for(; Cycles--; buffer += 64) {
02,743         //a0 = _mm_loadu_si128((__m128i *) (buffer+0*16));
02,744         //a1 = _mm_loadu_si128((__m128i *) (buffer+1*16));
02,745         //a2 = _mm_loadu_si128((__m128i *) (buffer+2*16));
02,746         //a3 = _mm_loadu_si128((__m128i *) (buffer+3*16));
02,747         a0YMM = _mm256_loadu_si256((__m256i *) (buffer+0*16));
02,748         a2YMM = _mm256_loadu_si256((__m256i *) (buffer+2*16));
02,749         //b0 = _mm_shuffle_epi8 (a3, ReverseMask);
02,750         //b1 = _mm_shuffle_epi8 (a2, ReverseMask);
02,751         //b2 = _mm_shuffle_epi8 (a1, ReverseMask);
02,752         //b3 = _mm_shuffle_epi8 (a0, ReverseMask);
02,753         b0YMM = _mm256_shuffle_epi8 (a2YMM, ReverseMaskYMM); // Caution: the stupid intrinsic works on 128bit not on 256bit! b0YMM = b1+b0 not b0+b1
02,754         b2YMM = _mm256_shuffle_epi8 (a0YMM, ReverseMaskYMM);
02,755 // Should swap;
02,756 // https://godbolt.org/z/dY74zv1Ph
02,757 b0YMM = _mm256_permute4x64_epi64(b0YMM, 0b01001110); // # ymm0 = ymm0[2,3,0,1]
02,758 b2YMM = _mm256_permute4x64_epi64(b2YMM, 0b01001110); // # ymm0 = ymm0[2,3,0,1]
02,759
02,760 /*
02,761 __m256i _mm256_permute4x64_epi64 (__m256i a, const int imm8)
02,762 #include <immintrin.h>
02,763 Instruction: vpermq ymm, ymm, imm8
02,764 CUID Flags: AVX2
02,765 Description
02,766 Shuffle 64-bit integers in a across lanes using the control in imm8, and store the results in dst.
02,767 Operation
02,768 DEFINE SELECT4(src, control) {
02,769     CASE(control[1:0]) OF
02,770     0:     tmp[63:0] := src[63:0]
02,771     1:     tmp[63:0] := src[127:64]
02,772     2:     tmp[63:0] := src[191:128]
02,773     3:     tmp[63:0] := src[255:192]
02,774     ESAC
02,775     RETURN tmp[63:0]
02,776 }
02,777 dst[63:0] := SELECT4(a[255:0], imm8[1:0])
02,778 dst[127:64] := SELECT4(a[255:0], imm8[3:2])
02,779 dst[191:128] := SELECT4(a[255:0], imm8[5:4])
02,780 dst[255:192] := SELECT4(a[255:0], imm8[7:6])
02,781 dst[MAX:256] := 0
02,782 */
02,783
02,784 //a8C7E8G7H5G3H1F2H3G1E2C1A2B4A6B8D7F8H7G5F7H8G6H4G2E1C2A1B3A5B7D8C6A7C8E7G8H6G4H2F1D2B1A3B5D6F5D4F3E5C4B2D3F4E6C5A4B6D5F6E4C3D1E3
02,785 //      a0]          a2]          a0]          a2]
02,786 //a8C7E8G7H5G3H1F2H3G1E2C1A2B4A6B8 D7F8H7G5F7H8G6H4G2E1C2A1B3A5B7D8 C6A7C8E7G8H6G4H2F1D2B1A3B5D6F5D4 F3E5C4B2D3F4E6C5A4B6D5F6E4C3D1E3

```

```

02,787
02,788 //a0: 61 38 43 37 | 45 38 47 37 | 48 35 47 33 | 48 31 46 32 ! 48 33 47 31 | 45 32 43 31 | 41 32 42 34 | 41 36 42 38
02,789 //a2: 44 37 46 38 | 48 37 47 35 | 46 37 48 38 | 47 36 48 34 ! 47 32 45 31 | 43 32 41 31 | 42 33 41 35 | 42 37 44 38
02,790 //b0: 34 48 36 47 | 38 48 37 46 | 35 47 37 48 | 38 46 37 44 ! 38 44 37 42 | 35 41 33 42 | 31 41 32 43 | 31 45 32 47
02,791 //b2: 32 46 31 48 | 33 47 35 48 | 37 47 38 45 | 37 43 38 61 ! 38 42 36 41 | 34 42 32 41 | 31 43 32 45 | 31 47 33 48
02,792
02,793 //tmp0 = _mm_shuffle_epi8 (a0, PartialInterleavingMask1);
02,794 //tmp1 = _mm_shuffle_epi8 (a0, PartialInterleavingMask2);
02,795 //tmp2 = _mm_shuffle_epi8 (a2, PartialInterleavingMask3);
02,796 //tmp3 = _mm_shuffle_epi8 (a2, PartialInterleavingMask4);
02,797 //c0 = _mm_or_si128 (tmp0, tmp2);
02,798 //c1 = _mm_or_si128 (tmp1, tmp3);
02,799 //tmp0 = _mm_shuffle_epi8 (a1, PartialInterleavingMask1);
02,800 //tmp1 = _mm_shuffle_epi8 (a1, PartialInterleavingMask2);
02,801 //tmp2 = _mm_shuffle_epi8 (a3, PartialInterleavingMask3);
02,802 //tmp3 = _mm_shuffle_epi8 (a3, PartialInterleavingMask4);
02,803 //c2 = _mm_or_si128 (tmp0, tmp2);
02,804 //c3 = _mm_or_si128 (tmp1, tmp3);
02,805 // c0: 00 20 01 21 | 02 22 03 23 | 04 24 05 25 | 06 26 07 27
02,806 // c1: 08 28 09 29 | 0a 2a 0b 2b | 0c 2c 0d 2d | 0e 2e 0f 2f
02,807 // c2: 10 30 11 31 | 12 32 13 33 | 14 34 15 35 | 16 36 17 37
02,808 // c3: 18 38 19 39 | 1a 3a 1b 3b | 1c 3c 1d 3d | 1e 3e 1f 3f
02,809 /*
02,810 __m256i _mm256_unpacklo_epi8 (__m256i a, __m256i b)
02,811 Synopsis
02,812 __m256i _mm256_unpacklo_epi8 (__m256i a, __m256i b)
02,813 #include <immintrin.h>
02,814 Instruction: vpunpcklbw ymm, ymm, ymm
02,815 CPUID Flags: AVX2
02,816 Description
02,817 Unpack and interleave 8-bit integers from the low half of each 128-bit lane in a and b, and store the results in dst.
02,818 Operation
02,819 #define INTERLEAVE_BYTES(src1[127:0], src2[127:0]) {
02,820     dst[7:0] := src1[7:0]
02,821     dst[15:8] := src2[7:0]
02,822     dst[23:16] := src1[15:8]
02,823     dst[31:24] := src2[15:8]
02,824     dst[39:32] := src1[23:16]
02,825     dst[47:40] := src2[23:16]
02,826     dst[55:48] := src1[31:24]
02,827     dst[63:56] := src2[31:24]
02,828     dst[71:64] := src1[39:32]
02,829     dst[79:72] := src2[39:32]
02,830     dst[87:80] := src1[47:40]
02,831     dst[95:88] := src2[47:40]
02,832     dst[103:96] := src1[55:48]
02,833     dst[111:104] := src2[55:48]
02,834     dst[119:112] := src1[63:56]
02,835     dst[127:120] := src2[63:56]
02,836     RETURN dst[127:0]
02,837 }
02,838 dst[127:0] := INTERLEAVE_BYTES(a[127:0], b[127:0])
02,839 dst[255:128] := INTERLEAVE_BYTES(a[255:128], b[255:128])
02,840 dst[MAX:256] := 0
02,841 */
02,842 //__m128i _mm_unpacklo_epi8 (__m128i a, __m128i b)
02,843
02,844     c0YMM = _mm256_unpacklo_epi8 (a0YMM, a2YMM);

```

```

02,845             c2YMM = _mm256_unpackhi_epi8 (a0YMM, a2YMM);
02,846 // Above two lines gave:
02,847 /*
02,848             [           0] [           1] [           2] [           3]
02,849 a0: 61 38 43 37 | 45 38 47 37 | 48 35 47 33 | 48 31 46 32 | 48 33 47 31 | 45 32 43 31 | 41 32 42 34 | 41 36 42 38
02,850             [           4] [           5] [           6] [           7]
02,851 a2: 44 37 46 38 | 48 37 47 35 | 46 37 48 38 | 47 36 48 34 | 47 32 45 31 | 43 32 41 31 | 42 33 41 35 | 42 37 44 38
02,852
02,853             [           0+4] [           2+6]
02,854 c0: 61 44 38 37 | 43 46 37 38 | 45 48 38 37 | 47 47 37 35 | 48 47 33 32 | 47 45 31 31 | 45 43 32 32 | 43 41 31 31
02,855             [           1+5] [           3+7]
02,856 c2: 48 46 35 37 | 47 48 33 38 | 48 47 31 36 | 46 48 32 34 | 41 42 32 33 | 42 41 34 35 | 41 42 36 37 | 42 44 38 38
02,857 */
02,858 // But I need:
02,859 // c0,c1,c2,c3      not c0,c2,c1,c3
02,860 // 0+4,1+5,2+6,3+7 not 0+4,2+6,1+5,3+7
02,861 // as in XMM:
02,862 // c0: 00 20 01 21 | 02 22 03 23 | 04 24 05 25 | 06 26 07 27
02,863 // c1: 08 28 09 29 | 0a 2a 0b 2b | 0c 2c 0d 2d | 0e 2e 0f 2f
02,864 // c2: 10 30 11 31 | 12 32 13 33 | 14 34 15 35 | 16 36 17 37
02,865 // c3: 18 38 19 39 | 1a 3a 1b 3b | 1c 3c 1d 3d | 1e 3e 1f 3f
02,866
02,867 //a0: 61 38 43 37 | 45 38 47 37 | 48 35 47 33 | 48 31 46 32 | 48 33 47 31 | 45 32 43 31 | 41 32 42 34 | 41 36 42 38
02,868 //a2: 44 37 46 38 | 48 37 47 35 | 46 37 48 38 | 47 36 48 34 | 47 32 45 31 | 43 32 41 31 | 42 33 41 35 | 42 37 44 38
02,869
02,870 //tmp0YMM: 61 00 38 00 | 43 00 37 00 | 45 00 38 00 | 47 00 37 00 | 41 00 32 00 | 42 00 34 00 | 41 00 36 00 | 42 00 38 00
02,871 //tmp2YMM: 00 44 00 37 | 00 46 00 38 | 00 48 00 37 | 00 47 00 35 | 00 42 00 33 | 00 41 00 35 | 00 42 00 37 | 00 44 00 38
02,872
02,873 //             tmp0YMM = _mm256_shuffle_epi8 (a0YMM, PartialInterleavingMask1YMM);
02,874 //             tmp2YMM = _mm256_shuffle_epi8 (a2YMM, PartialInterleavingMask3YMM);
02,875 //             c0YMM = _mm256_or_si256 (tmp0YMM, tmp2YMM);
02,876 //             tmp0YMM = _mm256_shuffle_epi8 (a0YMM, PartialInterleavingMask1YMM);
02,877 //             tmp2YMM = _mm256_shuffle_epi8 (a2YMM, PartialInterleavingMask3YMM);
02,878 //             c2YMM = _mm256_or_si256 (tmp0YMM, tmp2YMM);
02,879
02,880 //tmp0 = _mm_shuffle_epi8 (b0, PartialInterleavingMask1);
02,881 //tmp1 = _mm_shuffle_epi8 (b0, PartialInterleavingMask2);
02,882 //tmp2 = _mm_shuffle_epi8 (b2, PartialInterleavingMask3);
02,883 //tmp3 = _mm_shuffle_epi8 (b2, PartialInterleavingMask4);
02,884 //d0 = _mm_or_si128 (tmp0, tmp2);
02,885 //d1 = _mm_or_si128 (tmp1, tmp3);
02,886 //tmp0 = _mm_shuffle_epi8 (b1, PartialInterleavingMask1);
02,887 //tmp1 = _mm_shuffle_epi8 (b1, PartialInterleavingMask2);
02,888 //tmp2 = _mm_shuffle_epi8 (b3, PartialInterleavingMask3);
02,889 //tmp3 = _mm_shuffle_epi8 (b3, PartialInterleavingMask4);
02,890 //d2 = _mm_or_si128 (tmp0, tmp2);
02,891 //d3 = _mm_or_si128 (tmp1, tmp3);
02,892 // d0: 3f 1f 3e 1e | 3d 1d 3c 1c | 3b 1b 3a 1a | 39 19 38 18
02,893 // d1: 37 17 36 16 | 35 15 34 14 | 33 13 32 12 | 31 11 30 10
02,894 // d2: 2f 0f 2e 0e | 2d 0d 2c 0c | 2b 0b 2a 0a | 29 09 28 08
02,895 // d3: 27 07 26 06 | 25 05 24 04 | 23 03 22 02 | 21 01 20 00
02,896 // [[[ Next 6 lines are identical to simply REVERSE C vector - which is in 2 lines
02,897 //
02,898 //             tmp0YMM = _mm256_shuffle_epi8 (b0YMM, PartialInterleavingMask1YMM);
02,899 //             tmp2YMM = _mm256_shuffle_epi8 (b2YMM, PartialInterleavingMask3YMM);
02,900 //             d0YMM = _mm256_or_si256 (tmp0YMM, tmp2YMM);
02,901 //             tmp0YMM = _mm256_shuffle_epi8 (b0YMM, PartialInterleavingMask2YMM);
02,902 //             tmp2YMM = _mm256_shuffle_epi8 (b2YMM, PartialInterleavingMask4YMM);

```

```
02,903         d2YMM = _mm256_or_si256 (tmp0YMM, tmp2YMM);
02,904         */
02,905         // ]]] Next 6 lines are identical to simply REVERSE C vector - which is in 2 lines, but I don't have it, so {{{
02,906         d0YMM = _mm256_unpacklo_epi8 (b0YMM, b2YMM);
02,907         d2YMM = _mm256_unpackhi_epi8 (b0YMM, b2YMM);
02,908         // }}}
02,909
02,910 // For above code (the last thing to fix: b should be REVERSED a, not b1,b0,b3,b2):
02,911 //a0: 61 38 43 37 | 45 38 47 37 | 48 35 47 33 | 48 31 46 32 ! 48 33 47 31 | 45 32 43 31 | 41 32 42 34 | 41 36 42 38
02,912 //a2: 44 37 46 38 | 48 37 47 35 | 46 37 48 38 | 47 36 48 34 ! 47 32 45 31 | 43 32 41 31 | 42 33 41 35 | 42 37 44 38
02,913 //
02,914 //b0: 34 48 36 47 | 38 48 37 46 | 35 47 37 48 | 38 46 37 44 ! 38 44 37 42 | 35 41 33 42 | 31 41 32 43 | 31 45 32 47
02,915 //b2: 32 46 31 48 | 33 47 35 48 | 37 47 38 45 | 37 43 38 61 ! 38 42 36 41 | 34 42 32 41 | 31 43 32 45 | 31 47 33 48
02,916 //
02,917 //c0: 61 44 38 37 | 43 46 37 38 | 45 48 38 37 | 47 47 37 35 ! 48 47 33 32 | 47 45 31 31 | 45 43 32 32 | 43 41 31 31
02,918 //c2: 48 46 35 37 | 47 48 33 38 | 48 47 31 36 | 46 48 32 34 ! 41 42 32 33 | 42 41 34 35 | 41 42 36 37 | 42 44 38 38
02,919 //
02,920 //d0: 34 32 48 46 | 36 31 47 48 | 38 33 48 47 | 37 35 46 48 ! 38 38 44 42 | 37 36 42 41 | 35 34 41 42 | 33 32 42 41
02,921 //d2: 35 37 47 47 | 37 38 48 45 | 38 37 46 43 | 37 38 44 61 ! 31 31 41 43 | 32 32 43 45 | 31 31 45 47 | 32 33 47 48
02,922
02,923 /*
02,924     _mm256_storeu_si256((__m256i*)vector, a0YMM);
02,925     printf("a0: %02x %02x %02x %02x | %02x %02x %02x %02x | %02x %02x %02x %02x | %02x %02x %02x %02x ! %02x %02x %02x %02x | %02x %02x %02x %02x | %02x %02x %02x %02x\n",
02,926           vector[0], vector[1], vector[2], vector[3], vector[4], vector[5], vector[6], vector[7],
02,927           vector[8], vector[9], vector[10], vector[11], vector[12], vector[13], vector[14], vector[15],
02,928           vector[0+16], vector[1+16], vector[2+16], vector[3+16], vector[4+16], vector[5+16], vector[6+16], vector[7+16],
02,929           vector[8+16], vector[9+16], vector[10+16], vector[11+16], vector[12+16], vector[13+16], vector[14+16], vector[15+16]);
02,930
02,931     _mm256_storeu_si256((__m256i*)vector, a2YMM);
02,932     printf("a2: %02x %02x %02x %02x | %02x %02x %02x %02x | %02x %02x %02x %02x ! %02x %02x %02x %02x | %02x %02x %02x %02x | %02x %02x %02x %02x\n",
02,933           vector[0], vector[1], vector[2], vector[3], vector[4], vector[5], vector[6], vector[7],
02,934           vector[8], vector[9], vector[10], vector[11], vector[12], vector[13], vector[14], vector[15],
02,935           vector[0+16], vector[1+16], vector[2+16], vector[3+16], vector[4+16], vector[5+16], vector[6+16], vector[7+16],
02,936           vector[8+16], vector[9+16], vector[10+16], vector[11+16], vector[12+16], vector[13+16], vector[14+16], vector[15+16]);
02,937
02,938     _mm256_storeu_si256((__m256i*)vector, b0YMM);
02,939     printf("b0: %02x %02x %02x %02x | %02x %02x %02x %02x | %02x %02x %02x %02x ! %02x %02x %02x %02x | %02x %02x %02x %02x | %02x %02x %02x %02x\n",
02,940           vector[0], vector[1], vector[2], vector[3], vector[4], vector[5], vector[6], vector[7],
02,941           vector[8], vector[9], vector[10], vector[11], vector[12], vector[13], vector[14], vector[15],
02,942           vector[0+16], vector[1+16], vector[2+16], vector[3+16], vector[4+16], vector[5+16], vector[6+16], vector[7+16],
02,943           vector[8+16], vector[9+16], vector[10+16], vector[11+16], vector[12+16], vector[13+16], vector[14+16], vector[15+16]);
02,944
02,945     _mm256_storeu_si256((__m256i*)vector, b2YMM);
02,946     printf("b2: %02x %02x %02x %02x | %02x %02x %02x %02x | %02x %02x %02x %02x ! %02x %02x %02x %02x | %02x %02x %02x %02x | %02x %02x %02x %02x\n",
02,947           vector[0], vector[1], vector[2], vector[3], vector[4], vector[5], vector[6], vector[7],
02,948           vector[8], vector[9], vector[10], vector[11], vector[12], vector[13], vector[14], vector[15],
02,949           vector[0+16], vector[1+16], vector[2+16], vector[3+16], vector[4+16], vector[5+16], vector[6+16], vector[7+16],
02,950           vector[8+16], vector[9+16], vector[10+16], vector[11+16], vector[12+16], vector[13+16], vector[14+16], vector[15+16]);
02,951
02,952     _mm256_storeu_si256((__m256i*)vector, c0YMM);
02,953     printf("c0: %02x %02x %02x %02x | %02x %02x %02x %02x | %02x %02x %02x %02x ! %02x %02x %02x %02x | %02x %02x %02x %02x | %02x %02x %02x %02x\n",
02,954           vector[0], vector[1], vector[2], vector[3], vector[4], vector[5], vector[6], vector[7],
02,955           vector[8], vector[9], vector[10], vector[11], vector[12], vector[13], vector[14], vector[15],
```

```
02,956     vector[0+16], vector[1+16], vector[2+16], vector[3+16], vector[4+16], vector[5+16], vector[6+16], vector[7+16],
02,957     vector[8+16], vector[9+16], vector[10+16], vector[11+16], vector[12+16], vector[13+16], vector[14+16], vector[15+16]);
02,958     _mm256_storeu_si256((__m256i*)vector, c2YMM);
02,959     printf("c2: %02x %02x %02x %02x ! %02x %02x %02x %02x ! %02x %02x %02x %02x ! %02x %02x %02x %02x ! %02x %02x %02x %02x\n",
! %02x %02x %02x %02x\n",
02,960     vector[0], vector[1], vector[2], vector[3], vector[4], vector[5], vector[6], vector[7],
02,961     vector[8], vector[9], vector[10], vector[11], vector[12], vector[13], vector[14], vector[15],
02,962     vector[0+16], vector[1+16], vector[2+16], vector[3+16], vector[4+16], vector[5+16], vector[6+16], vector[7+16],
02,963     vector[8+16], vector[9+16], vector[10+16], vector[11+16], vector[12+16], vector[13+16], vector[14+16], vector[15+16]);
02,964
02,965     _mm256_storeu_si256((__m256i*)vector, d0YMM);
02,966     printf("d0: %02x %02x %02x %02x ! %02x %02x %02x %02x ! %02x %02x %02x %02x ! %02x %02x %02x %02x ! %02x %02x %02x %02x\n",
! %02x %02x %02x %02x\n",
02,967     vector[0], vector[1], vector[2], vector[3], vector[4], vector[5], vector[6], vector[7],
02,968     vector[8], vector[9], vector[10], vector[11], vector[12], vector[13], vector[14], vector[15],
02,969     vector[0+16], vector[1+16], vector[2+16], vector[3+16], vector[4+16], vector[5+16], vector[6+16], vector[7+16],
02,970     vector[8+16], vector[9+16], vector[10+16], vector[11+16], vector[12+16], vector[13+16], vector[14+16], vector[15+16]);
02,971     _mm256_storeu_si256((__m256i*)vector, d2YMM);
02,972     printf("d2: %02x %02x %02x %02x ! %02x %02x %02x %02x ! %02x %02x %02x %02x ! %02x %02x %02x %02x ! %02x %02x %02x %02x\n",
! %02x %02x %02x %02x\n",
02,973     vector[0], vector[1], vector[2], vector[3], vector[4], vector[5], vector[6], vector[7],
02,974     vector[8], vector[9], vector[10], vector[11], vector[12], vector[13], vector[14], vector[15],
02,975     vector[0+16], vector[1+16], vector[2+16], vector[3+16], vector[4+16], vector[5+16], vector[6+16], vector[7+16],
02,976     vector[8+16], vector[9+16], vector[10+16], vector[11+16], vector[12+16], vector[13+16], vector[14+16], vector[15+16]);
02,977
02,978     printf("\n");
02,979 */
02,980
02,981 /*
02,982 G:\Lookupperorama_r13\COLLISION_Hashliner>Hashliner_DDAES_dump5byteshash.exe 1.KnightTours.txt
02,983 394e907d43
02,984
02,985 G:\Lookupperorama_r13\COLLISION_Hashliner>Hashliner_DDAES-XMM_dump5byteshash.exe 1.KnightTours.txt
02,986 f4b027e3ab
02,987
02,988 G:\Lookupperorama_r13\COLLISION_Hashliner>d:
02,989
02,990 D:\>g:
02,991
02,992 G:\Lookupperorama_r13\COLLISION_Hashliner>Hashliner_DDAES_dump5byteshash.exe 1.KnightTours.txt
02,993 a0: 61 38 43 37 ! 45 38 47 37 ! 48 35 47 33 ! 48 31 46 32 ! 48 33 47 31 ! 45 32 43 31 ! 41 32 42 34 ! 41 36 42 38
02,994 a2: 44 37 46 38 ! 48 37 47 35 ! 46 37 48 38 ! 47 36 48 34 ! 47 32 45 31 ! 43 32 41 31 ! 42 33 41 35 ! 42 37 44 38
02,995 c0: 61 44 38 37 ! 43 46 37 38 ! 45 48 38 37 ! 47 47 37 35 ! 41 42 32 33 ! 42 41 34 35 ! 41 42 36 37 ! 42 44 38 38
02,996 c2: 61 44 38 37 ! 43 46 37 38 ! 45 48 38 37 ! 47 47 37 35 ! 41 42 32 33 ! 42 41 34 35 ! 41 42 36 37 ! 42 44 38 38
02,997
02,998 a0: 43 36 41 37 ! 43 38 45 37 ! 47 38 48 36 ! 47 34 48 32 ! 46 31 44 32 ! 42 31 41 33 ! 42 35 44 36 ! 46 35 44 34
02,999 a2: 46 33 45 35 ! 43 34 42 32 ! 44 33 46 34 ! 45 36 43 35 ! 41 34 42 36 ! 44 35 46 36 ! 45 34 43 33 ! 44 31 45 33
03,000 c0: 43 46 36 33 ! 41 45 37 35 ! 43 43 38 34 ! 45 42 37 32 ! 42 45 35 34 ! 44 43 36 33 ! 46 44 35 31 ! 44 45 34 33
03,001 c2: 43 46 36 33 ! 41 45 37 35 ! 43 43 38 34 ! 45 42 37 32 ! 42 45 35 34 ! 44 43 36 33 ! 46 44 35 31 ! 44 45 34 33
03,002
03,003 394e907d43
03,004
03,005 G:\Lookupperorama_r13\COLLISION_Hashliner>type 1.KnightTours.txt
03,006 a8C7E8G7H5G3H1F2H3G1E2C1A2B4A6B8D7F8H7G5F7H8G6H4G2E1C2A1B3A5B7D8C6A7C8E7G8H6G4H2F1D2B1A3B5D6F5D4F3E5C4B2D3F4E6C5A4B6D5F6E4C3D1E3
03,007
03,008 G:\Lookupperorama_r13\COLLISION_Hashliner>
03,009         a0]                               a2]                               a0]                               a2]
03,010 a8C7E8G7H5G3H1F2H3G1E2C1A2B4A6B8 D7F8H7G5F7H8G6H4G2E1C2A1B3A5B7D8 C6A7C8E7G8H6G4H2F1D2B1A3B5D6F5D4 F3E5C4B2D3F4E6C5A4B6D5F6E4C3D1E3
```

```
03,011
03,012 */
03,013
03,014 /*
03,015     _mm_storeu_si128((__m128i*)vector, *((_m128i *)(&a0YMM));
03,016     printf("a0lo: %02x %02x %02x %02x | %02x %02x %02x %02x | %02x %02x %02x %02x\n",
03,017         vector[0], vector[1], vector[2], vector[3], vector[4], vector[5], vector[6], vector[7],
03,018         vector[8], vector[9], vector[10], vector[11], vector[12], vector[13], vector[14], vector[15]);
03,019     _mm_storeu_si128((__m128i*)vector, *((_m128i *)(&a0YMM)+1));
03,020     printf("a0hi: %02x %02x %02x %02x | %02x %02x %02x %02x | %02x %02x %02x %02x\n",
03,021         vector[0], vector[1], vector[2], vector[3], vector[4], vector[5], vector[6], vector[7],
03,022         vector[8], vector[9], vector[10], vector[11], vector[12], vector[13], vector[14], vector[15]);
03,023
03,024     printf("Above two should equal a0YMM\n");
03,025 */
03,026 //void SlowCopy128bit (const char *SOURCE, char *TARGET) { _mm_storeu_si128((__m128i *) (TARGET), _mm_loadu_si128((const __m128i *) (SOURCE))); }
03,027
03,028 // Okay, the final dump (shows XMM and YMM variants produce the same hash):
03,029 /*
03,030
03,031 D:\Lookupperorama_r13\COLLISION_Hashliner>Hashliner_DD AES_dump5byteshash.exe 1.KnightTours.txt
03,032 a0: 61 38 43 37 | 45 38 47 37 | 48 35 47 33 | 48 31 46 32
03,033 a1: 48 33 47 31 | 45 32 43 31 | 41 32 42 34 | 41 36 42 38
03,034 a2: 44 37 46 38 | 48 37 47 35 | 46 37 48 38 | 47 36 48 34
03,035 a3: 47 32 45 31 | 43 32 41 31 | 42 33 41 35 | 42 37 44 38
03,036
03,037 b0: 38 44 37 42 | 35 41 33 42 | 31 41 32 43 | 31 45 32 47
03,038 b1: 34 48 36 47 | 38 48 37 46 | 35 47 37 48 | 38 46 37 44
03,039 b2: 38 42 36 41 | 34 42 32 41 | 31 43 32 45 | 31 47 33 48
03,040 b3: 32 46 31 48 | 33 47 35 48 | 37 47 38 45 | 37 43 38 61
03,041
03,042 c0: 61 44 38 37 | 43 46 37 38 | 45 48 38 37 | 47 47 37 35
03,043 c1: 48 46 35 37 | 47 48 33 38 | 48 47 31 36 | 46 48 32 34
03,044 c2: 48 47 33 32 | 47 45 31 31 | 45 43 32 32 | 43 41 31 31
03,045 c3: 41 42 32 33 | 42 41 34 35 | 41 42 36 37 | 42 44 38 38
03,046
03,047 d0: 38 38 44 42 | 37 36 42 41 | 35 34 41 42 | 33 32 42 41
03,048 d1: 31 31 41 43 | 32 32 43 45 | 31 31 45 47 | 32 33 47 48
03,049 d2: 34 32 48 46 | 36 31 47 48 | 38 33 48 47 | 37 35 46 48
03,050 d3: 35 37 47 47 | 37 38 48 45 | 38 37 46 43 | 37 38 44 61
03,051
03,052 a0: 43 36 41 37 | 43 38 45 37 | 47 38 48 36 | 47 34 48 32
03,053 a1: 46 31 44 32 | 42 31 41 33 | 42 35 44 36 | 46 35 44 34
03,054 a2: 46 33 45 35 | 43 34 42 32 | 44 33 46 34 | 45 36 43 35
03,055 a3: 41 34 42 36 | 44 35 46 36 | 45 34 43 33 | 44 31 45 33
03,056
03,057 b0: 33 45 31 44 | 33 43 34 45 | 36 46 35 44 | 36 42 34 41
03,058 b1: 35 43 36 45 | 34 46 33 44 | 32 42 34 43 | 35 45 33 46
03,059 b2: 34 44 35 46 | 36 44 35 42 | 33 41 31 42 | 32 44 31 46
03,060 b3: 32 48 34 47 | 36 48 38 47 | 37 45 38 43 | 37 41 36 43
03,061
03,062 c0: 43 46 36 33 | 41 45 37 35 | 43 43 38 34 | 45 42 37 32
03,063 c1: 47 44 38 33 | 48 46 36 34 | 47 45 34 36 | 48 43 32 35
03,064 c2: 46 41 31 34 | 44 42 32 36 | 42 44 31 35 | 41 46 33 36
03,065 c3: 42 45 35 34 | 44 43 36 33 | 46 44 35 31 | 44 45 34 33
03,066
03,067 d0: 33 34 45 44 | 31 35 44 46 | 33 36 43 44 | 34 35 45 42
03,068 d1: 36 33 46 41 | 35 31 44 42 | 36 32 42 44 | 34 31 41 46
```

```
03,069 d2: 35 32 43 48 | 36 34 45 47 | 34 36 46 48 | 33 38 44 47
03,070 d3: 32 37 42 45 | 34 38 43 43 | 35 37 45 41 | 33 36 46 43
03,071
03,072 f4b027e3ab
03,073 a0: 61 38 43 37 | 45 38 47 37 | 48 35 47 33 | 48 31 46 32 | 48 33 47 31 | 45 32 43 31 | 41 32 42 34 | 41 36 42 38
03,074 a2: 44 37 46 38 | 48 37 47 35 | 46 37 48 38 | 47 36 48 34 | 47 32 45 31 | 43 32 41 31 | 42 33 41 35 | 42 37 44 38
03,075 b0: 38 44 37 42 | 35 41 33 42 | 31 41 32 43 | 31 45 32 47 | 34 48 36 47 | 38 48 37 46 | 35 47 37 48 | 38 46 37 44
03,076 b2: 38 42 36 41 | 34 42 32 41 | 31 43 32 45 | 31 47 33 48 | 32 46 31 48 | 33 47 35 48 | 37 47 38 45 | 37 43 38 61
03,077 c0: 61 44 38 37 | 43 46 37 38 | 45 48 38 37 | 47 47 37 35 | 48 47 33 32 | 47 45 31 31 | 45 43 32 32 | 43 41 31 31
03,078 c2: 48 46 35 37 | 47 48 33 38 | 48 47 31 36 | 46 48 32 34 | 41 42 32 33 | 42 41 34 35 | 41 42 36 37 | 42 44 38 38
03,079 d0: 38 38 44 42 | 37 36 42 41 | 35 34 41 42 | 33 32 42 41 | 34 32 48 46 | 36 31 47 48 | 38 33 48 47 | 37 35 46 48
03,080 d2: 31 31 41 43 | 32 32 43 45 | 31 31 45 47 | 32 33 47 48 | 35 37 47 47 | 37 38 48 45 | 38 37 46 43 | 37 38 44 61
03,081
03,082 a0lo: 61 38 43 37 | 45 38 47 37 | 48 35 47 33 | 48 31 46 32
03,083 a0hi: 48 33 47 31 | 45 32 43 31 | 41 32 42 34 | 41 36 42 38
03,084 Above two should equal a0YMM
03,085 a0: 43 36 41 37 | 43 38 45 37 | 47 38 48 36 | 47 34 48 32 | 46 31 44 32 | 42 31 41 33 | 42 35 44 36 | 46 35 44 34
03,086 a2: 46 33 45 35 | 43 34 42 32 | 44 33 46 34 | 45 36 43 35 | 41 34 42 36 | 44 35 46 36 | 45 34 43 33 | 44 31 45 33
03,087 b0: 33 45 31 44 | 33 43 34 45 | 36 46 35 44 | 36 42 34 41 | 35 43 36 45 | 34 46 33 44 | 32 42 34 43 | 35 45 33 46
03,088 b2: 34 44 35 46 | 36 44 35 42 | 33 41 31 42 | 32 44 31 46 | 32 48 34 47 | 36 48 38 47 | 37 45 38 43 | 37 41 36 43
03,089 c0: 43 46 36 33 | 41 45 37 35 | 43 43 38 34 | 45 42 37 32 | 46 41 31 34 | 44 42 32 36 | 42 44 31 35 | 41 46 33 36
03,090 c2: 47 44 38 33 | 48 46 36 34 | 47 45 34 36 | 48 43 32 35 | 42 45 35 34 | 44 43 36 33 | 46 44 35 31 | 44 45 34 33
03,091 d0: 33 34 45 44 | 31 35 44 46 | 33 36 43 44 | 34 35 45 42 | 35 32 43 48 | 36 34 45 47 | 34 36 46 48 | 33 38 44 47
03,092 d2: 36 33 46 41 | 35 31 44 42 | 36 32 42 44 | 34 31 41 46 | 32 37 42 45 | 34 38 43 43 | 35 37 45 41 | 33 36 46 43
03,093
03,094 a0lo: 43 36 41 37 | 43 38 45 37 | 47 38 48 36 | 47 34 48 32
03,095 a0hi: 46 31 44 32 | 42 31 41 33 | 42 35 44 36 | 46 35 44 34
03,096 Above two should equal a0YMM
03,097 f4b027e3ab
03,098
03,099 D:\Lookupperorama_r13\COLLISION_Hashliner>
03,100 */
03,101
03,102 //In YMM should swap c1 and c2, also d1 and d2
03,103
03,104 //hashA = _mm_aesenc_si128(hashA, a0);
03,105 hashA = _mm_aesenc_si128(hashA, *((_m128i *)(&a0YMM)));
03,106 //hashB = _mm_aesenc_si128(hashB, b0);
03,107 hashB = _mm_aesenc_si128(hashB, *((_m128i *)(&b0YMM)));
03,108 //hashB = _mm_aesenc_si128(hashB, *((_m128i *)(&b0YMM)+1));
03,109 //hashC = _mm_aesenc_si128(hashC, c0);
03,110 hashC = _mm_aesenc_si128(hashC, *((_m128i *)(&c0YMM)));
03,111 //hashD = _mm_aesenc_si128(hashD, d0);
03,112 hashD = _mm_aesenc_si128(hashD, *((_m128i *)(&d0YMM)));
03,113
03,114 //hashA = _mm_aesenc_si128(hashA, a1);
03,115 hashA = _mm_aesenc_si128(hashA, *((_m128i *)(&a0YMM)+1));
03,116 //hashB = _mm_aesenc_si128(hashB, b1);
03,117 hashB = _mm_aesenc_si128(hashB, *((_m128i *)(&b0YMM)+1));
03,118 //hashB = _mm_aesenc_si128(hashB, *((_m128i *)(&b0YMM)));
03,119 //hashC = _mm_aesenc_si128(hashC, c1);
03,120 hashC = _mm_aesenc_si128(hashC, *((_m128i *)(&c2YMM)));
03,121 //hashD = _mm_aesenc_si128(hashD, d1);
03,122 hashD = _mm_aesenc_si128(hashD, *((_m128i *)(&d2YMM)));
03,123
03,124 //hashA = _mm_aesenc_si128(hashA, a2);
03,125 hashA = _mm_aesenc_si128(hashA, *((_m128i *)(&a2YMM)));
03,126 //hashB = _mm_aesenc_si128(hashB, b2);
```

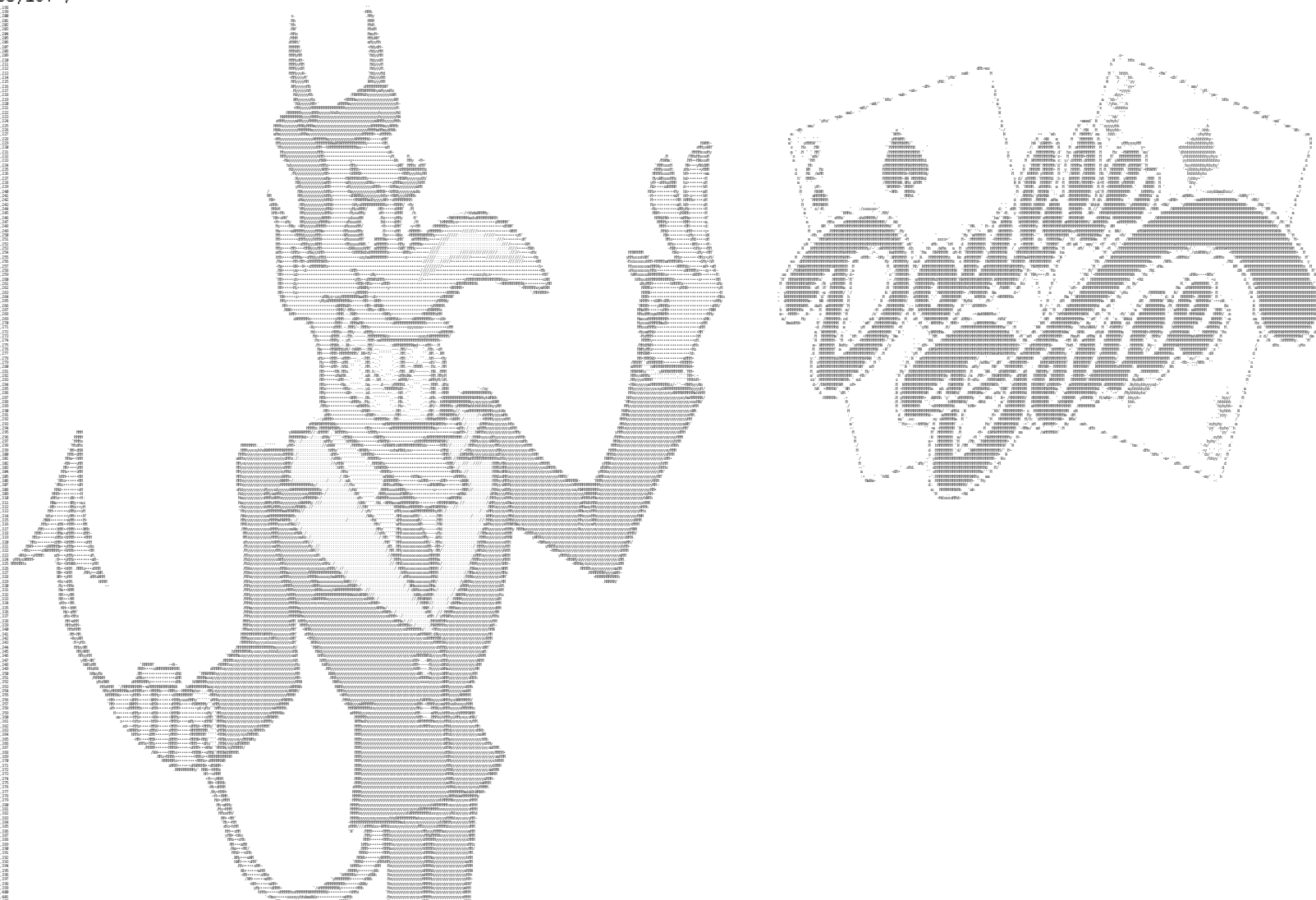


```

03,127         hashB = _mm_aesenc_si128(hashB, *((__m128i *)(&b2YMM));
03,128         //hashB = _mm_aesenc_si128(hashB, *((__m128i *)(&b2YMM)+1));
03,129         //hashC = _mm_aesenc_si128(hashC, c2);
03,130         hashC = _mm_aesenc_si128(hashC, *((__m128i *)(&c0YMM)+1));
03,131         //hashD = _mm_aesenc_si128(hashD, d2);
03,132         hashD = _mm_aesenc_si128(hashD, *((__m128i *)(&d0YMM)+1));
03,133
03,134         //hashA = _mm_aesenc_si128(hashA, a3);
03,135         hashA = _mm_aesenc_si128(hashA, *((__m128i *)(&a2YMM)+1));
03,136         //hashB = _mm_aesenc_si128(hashB, b3);
03,137         hashB = _mm_aesenc_si128(hashB, *((__m128i *)(&b2YMM)+1));
03,138         //hashB = _mm_aesenc_si128(hashB, *((__m128i *)(&b2YMM));
03,139         //hashC = _mm_aesenc_si128(hashC, c3);
03,140         hashC = _mm_aesenc_si128(hashC, *((__m128i *)(&c2YMM)+1));
03,141         //hashD = _mm_aesenc_si128(hashD, d3);
03,142         hashD = _mm_aesenc_si128(hashD, *((__m128i *)(&d2YMM)+1));
03,143
03,144         hashA = _mm_aesenc_si128(hashA, hashB);
03,145         hashA = _mm_aesenc_si128(hashA, hashC);
03,146         hashA = _mm_aesenc_si128(hashA, hashD);
03,147         length = length - 64;
03,148     }
03,149 }
03,150
03,151 ptr128 = (__m128i *)buffer;
03,152 if (length >= 16) {
03,153     Cycles = length/16;
03,154     for(; Cycles--; buffer += 16) {
03,155         AgainstRules = _mm_loadu_si128(ptr128++);
03,156         GumbottronREVER = _mm_shuffle_epi8 (AgainstRules, ReverseMask);
03,157         GumbottronINTER = _mm_shuffle_epi8 (AgainstRules, InterleaveMask);
03,158         GumbottronREVERINTER = _mm_shuffle_epi8 (GumbottronREVER, InterleaveMask);
03,159         hashA = _mm_aesenc_si128(hashA, AgainstRules);
03,160         hashB = _mm_aesenc_si128(hashB, GumbottronREVER);
03,161         hashC = _mm_aesenc_si128(hashC, GumbottronINTER);
03,162         hashD = _mm_aesenc_si128(hashD, GumbottronREVERINTER);
03,163         hashA = _mm_aesenc_si128(hashA, hashB);
03,164         hashA = _mm_aesenc_si128(hashA, hashC);
03,165         hashA = _mm_aesenc_si128(hashA, hashD);
03,166         length = length - 16;
03,167     }
03,168 }
03,169 // Inhere using Pippip's approach to read past the end ("the dirty" sentinel like style, or more like padding):
03,170 if (length & (16-1)) {
03,171     AgainstRules = _mm_loadu_si128(ptr128);
03,172     //AgainstRules = _mm_srli_si128 (AgainstRules, 16-length); // catastrophic error: Intrinsic parameter must be an immediate value
03,173     AgainstRules = _mm_and_si128 (AgainstRules, Gumbottron[length]);
03,174     //Gumbottron = _mm_slli_si128 (Gumbottron, 16-length); // catastrophic error: Intrinsic parameter must be an immediate value
03,175     Gumbottron = _mm_and_si128 (hashB, Gumbottron[length]);
03,176     AgainstRules = _mm_or_si128 (AgainstRules, Gumbottron);
03,177
03,178     GumbottronREVER = _mm_shuffle_epi8 (AgainstRules, ReverseMask);
03,179     GumbottronINTER = _mm_shuffle_epi8 (AgainstRules, InterleaveMask);
03,180     GumbottronREVERINTER = _mm_shuffle_epi8 (GumbottronREVER, InterleaveMask);
03,181     hashA = _mm_aesenc_si128(hashA, AgainstRules);
03,182     hashB = _mm_aesenc_si128(hashB, GumbottronREVER);
03,183     hashC = _mm_aesenc_si128(hashC, GumbottronINTER);
03,184     hashD = _mm_aesenc_si128(hashD, GumbottronREVERINTER);

```

```
03,185         hashA = _mm_aesenc_si128(hashA, hashB);
03,186         hashA = _mm_aesenc_si128(hashA, hashC);
03,187         hashA = _mm_aesenc_si128(hashA, hashD);
03,188     }
03,189     SlowCopy128bit( (const char *)(&hashA), (char *)&DDAES[0]);
03,190 }
03,191 // Revision 2021-Aug-22 ]
03,192
03,193 // Revision 2021-Aug-28 ]
03,194
03,195 #endif
03,196
03,197 /*
```



```

03,462 */
03,463
03,464 #if defined(_icl_mumbo_jumbo_)
03,465 // GetRDTSC() taken from strchr.com
03,466 #if defined(_M_IX86)
03,467 unsigned long long __forceinline GetRDTSC(void) {
03,468     __asm {
03,469         ; Flush the pipeline
03,470         XOR eax, eax
03,471         CPUID
03,472         ; Get RDTSC counter in edx:eax
03,473         RDTSC
03,474     }
03,475 }
03,476 #elif defined(_M_X64)
03,477 unsigned long long __forceinline GetRDTSC(void) {
03,478     return __rdtsc();
03,479 }
03,480 #else
03,481 unsigned long long __forceinline GetRDTSC(void) {
03,482     return GetTickCount();
03,483 }
03,484 #endif
03,485 #endif
03,486
03,487 // Leprechaun BHex [
03,488 // How to Compile?
03,489 // For *nix replace -D_WIN32_ENVIRONMENT_ with -D_POSIX_ENVIRONMENT_
03,490
03,491 // The command line should have one of these, if not the default is set to Windows:
03,492 // #define _WIN32_ENVIRONMENT_
03,493 // #define _POSIX_ENVIRONMENT_
03,494
03,495 #if defined(_WIN32_ENVIRONMENT_) || defined(_POSIX_ENVIRONMENT_)
03,496 #else
03,497 #define _WIN32_ENVIRONMENT_ // Default
03,498 #endif
03,499
03,500 // Compile for BB=24 with:
03,501 // icl /Ox /TcLeprechaun_x-leton.c /FaLeprechaun_x-leton /w /Facs -Dsingleton -D1p -D_WIN32_ENVIRONMENT_ -DLongestLineInclusive=48 -DExternalRAM -
03,502 // DRAMpoolInKB=12000123
03,503 #ifndef LongestLineInclusive
03,504 // #define LongestLineInclusive 36 // Default is 18*2, for HEXed i.e. 2:1 - printable keys, that is.
03,505 #define LongestLineInclusive 28 // Actually for not HEXed keys the MAX should be 28*1=28 - the 1:1 length of SHA3-224.

```

```
03,506 #else
03,507 #endif
03,508
03,509 #ifndef SpeedUpBuilding // This is how many megabytes you want to add up to the 'TargetBuffer' - if more then less passes - if huge (like 92000) then probably 1 pass
only
03,510 #define SpeedUpBuilding 32
03,511 #else
03,512 #endif
03,513
03,514 #if defined(InternalRAM) || defined(ExternalRAM)
03,515 #else
03,516 #define InternalRAM // Default
03,517 #endif
03,518
03,519 #ifndef RAMpoolInKB
03,520 #define RAMpoolInKB 5123456 // Default ~5GB
03,521 #else
03,522 #endif
03,523
03,524 #define MAXsizeDecompressionRAMpoolInMB 12288 // Default 12GB, 2021-Jun-23
03,525
03,526 #ifndef pseudoLAZY
03,527 #define GREEDY
03,528 #else
03,529 #endif
03,530
03,531 #define Sandokan (512) // Sandokan should be multiple of 64! Exploiting the reserved 0x93 tag by assigning this matchlen and this 3 bytes offset, it could be 4 for 4GB
but the compression rate will go down
03,532
03,533 /*
03,534 #if defined(_WIN32_ENVIRONMENT_)
03,535 __declspec(aligned(64))
03,536 #else
03,537 //__attribute__((aligned(64)));
03,538 #endif
03,539 */
03,540
03,541 typedef unsigned short USHORT; // As for 'With *(DWORD*)', a buffer overrun is possible at the end of a memory page.' I knew about it but was fooled by assembly code
generated by VS2010 which translates it to a word access:
03,542 typedef unsigned int UINT;
03,543 //typedef unsigned int DWORD;
03,544
03,545 #ifndef NULL
03,546 #ifdef __cplusplus
03,547 #define NULL 0
03,548 #else
03,549 #define NULL ((void*)0)
03,550 #endif
03,551 #endif
03,552
03,553 #ifndef HashInBITS
03,554 #define HashInBITS 27 // default 26 i.e. 2^26 i.e. 64MS(Mega Slots); slots contain 8bytes pointers or 512MB, because many netbooks have 512MB free (1GB in total)!
03,555 #else
03,556 #endif
03,557
03,558 #ifndef HashChunkSizeInBITS
03,559 #define HashChunkSizeInBITS 27 // Defines the number of passes. Should be smaller or equal to HashInBITS. If HashInBITS == HashChunkSizeInBITS then 2^(HashInBITS-
HashChunkSizeInBITS)=2^0=1 pass(es).
```

```
03,560 #else
03,561 #endif
03,562
03,563 int RAMpoolInKB_GLOBAL= RAMpoolInKB;
03,564 int HashInBITS_GLOBAL= HashInBITS;
03,565 int HashChunkSizeInBITS_GLOBAL= HashChunkSizeInBITS;
03,566
03,567 #if defined(InternalRAM)
03,568 int BStorBtree=3;
03,569 #endif
03,570
03,571 #if defined(ExternalRAM)
03,572 int BStorBtree=2;
03,573 #endif
03,574
03,575 #include <stdio.h>
03,576 #include <ctype.h>
03,577 #include <time.h>
03,578 #if defined(_WIN32_ENVIRONMENT_)
03,579 #include <io.h> // needed for Windows' 'lseeki64' and 'telli64'
03,580 //Above line must be commented in order to compile with Intel C compiler: an error "can't find io.h" occurs.
03,581 #else
03,582 #endif /* defined(_WIN32_ENVIRONMENT_) */
03,583
03,584 #define ROL(x, n) (((x) << (n)) | ((x) >> (32-(n))))
03,585 UINT FNV1A_Hash_Jesteress_27bit(const char *str, unsigned int wrdlen)
03,586 {
03,587     const UINT PRIME = 709607;
03,588     UINT hash32 = 2166136261;
03,589     const char *p = str;
03,590
03,591     // Idea comes from Igor Pavlov's 7zCRC, thanks.
03,592     /*
03,593     for(; wrdlen && ((unsigned)(ptrdiff_t)p&3); wrdlen -= 1, p++) {
03,594         hash32 = (hash32 ^ *p) * PRIME;
03,595     }
03,596     */
03,597     for(; wrdlen >= 2*sizeof(UINT); wrdlen -= 2*sizeof(UINT), p += 2*sizeof(UINT)) {
03,598         hash32 = (hash32 ^ (ROL(*(UINT *)p,5)^*(UINT *)p+4))) * PRIME;
03,599     }
03,600     // Cases: 0,1,2,3,4,5,6,7
03,601     if (wrdlen & sizeof(UINT)) {
03,602         hash32 = (hash32 ^ *(UINT*)p) * PRIME;
03,603         p += sizeof(UINT);
03,604     }
03,605     if (wrdlen & sizeof(USHORT)) {
03,606         hash32 = (hash32 ^ *(USHORT*)p) * PRIME;
03,607         p += sizeof(USHORT);
03,608     }
03,609     if (wrdlen & 1)
03,610         hash32 = (hash32 ^ *p) * PRIME;
03,611
03,612     return (hash32 ^ (hash32 >> 16)) & (( (1LL)<<HashInBITS_GLOBAL )-1);
03,613 }
03,614
03,615 // "There it now stands for ever. Black on white.
03,616 // I can't get away from it. Ahoy, Yorikke, ahoy, hoy, ho!
03,617 // Go to hell now if you wish. What do I care? It's all the same now to me.
```

```

03,618 // I am part of you now. Where you go I go, where you leave I leave, when you go to the devil I go. Married.
03,619 // Vanished from the living. Damned and doomed. Of me there is not left a breath in all the vast world.
03,620 // Ahoy, Yorikke! Ahoy, hoy, ho!
03,621 // I am not buried in the sea,
03,622 // The death ship is now part of me
03,623 // So far from sunny New Orleans
03,624 // So far from lovely Louisiana."
03,625 // /An excerpt from 'THE DEATH SHIP - THE STORY OF AN AMERICAN SAILOR' by B.TRAVEN/
03,626 //
03,627 // "Walking home to our good old Yorikke, I could not help thinking of this beautiful ship, with a crew on board that had faces as if they were seeing ghosts by day
and by night.
03,628 // Compared to that gilded Empress, the Yorikke was an honorable old lady with lavender sachets in her drawers.
03,629 // Yorikke did not pretend to anything she was not. She lived up to her looks. Honest to her lowest ribs and to the leaks in her bilge.
03,630 // Now, what is this? I find myself falling in love with that old jane.
03,631 // All right, I cannot pass by you, Yorikke; I have to tell you I love you. Honest, baby, I love you.
03,632 // I have six black finger-nails, and four black and green-blue nails on my toes, which you, honey, gave me when necking you.
03,633 // Grate-bars have crushed some of my toes. And each finger-nail has its own painful story to tell.
03,634 // My chest, my back, my arms, my legs are covered with scars of burns and scorchings.
03,635 // Each scar, when it was being created, caused me pains which I shall surely never forget.
03,636 // But every outcry of pain was a love-cry for you, honey.
03,637 // You are no hypocrite. Your heart does not bleed tears when you do not feel heart-aches deeply and truly.
03,638 // You do not dance on the water if you do not feel like being jolly and kicking chasers in the pants.
03,639 // Your heart never lies. It is fine and clean like polished gold. Never mind the rags, honey dear.
03,640 // When you laugh, your whole soul and all your body is laughing.
03,641 // And when you weep, sweetly, then you weep so that even the reefs you pass feel like weeping with you.
03,642 // I never want to leave you again, honey. I mean it. Not for all the rich and elegant buckets in the world.
03,643 // I love you, my gypsy of the sea!"
03,644 // /An excerpt from 'THE DEATH SHIP - THE STORY OF AN AMERICAN SAILOR' by B.TRAVEN/
03,645 //
03,646 // Dedicated to Pippip, the main character in the 'Das Totenschiff' roman, actually the B.Traven himself, his real name was Hermann Albert Otto Maksymilian Feige.
03,647 // CAUTION: Add 8 more bytes to the buffer being hashed, usually malloc(...+8) - to prevent out of boundary reads!
03,648 // Many thanks go to Yurii 'Hordi' Hordiienko, he lessened with 3 instructions the original 'Pippip', thus:
03,649 // #include <stdlib.h>
03,650 // #include <stdint.h>
03,651 #define _PADr_KAZE(x, n) ( ((x) << (n)) >> (n) )
03,652 uint32_t FNV1A_Pippip_Yurii(const char *str, size_t wrdlen) {
03,653     const uint32_t PRIME = 591798841; uint32_t hash32; uint64_t hash64 = 14695981039346656037ULL;
03,654     size_t Cycles, NDhead;
03,655     if (wrdlen > 8) {
03,656         Cycles = ((wrdlen - 1) >> 4) + 1; NDhead = wrdlen - (Cycles << 3);
03,657         #pragma nounroll
03,658         for(; Cycles--; str += 8) {
03,659             hash64 = ( hash64 ^ (*(uint64_t *) (str)) ) * PRIME;
03,660             hash64 = ( hash64 ^ (*(uint64_t *) (str+NDhead)) ) * PRIME;
03,661         }
03,662     } else
03,663     hash64 = ( hash64 ^ _PADr_KAZE(*(uint64_t *) (str+0), (8-wrdlen) << 3) ) * PRIME;
03,664     hash32 = (uint32_t) (hash64 ^ (hash64 >> 32));
03,665     //return hash32 ^ (hash32 >> 16);
03,666     return (hash32 ^ (hash32 >> 16)) & (( (1LL) << HashInBITS_GLOBAL ) - 1);
03,667 } // Last update: 2019-Oct-30, 14 C lines strong, Kaze.
03,668
03,669 // https://godbolt.org/z/i40ipj x86-64 gcc 9.2 -O3
03,670 /*
03,671 FNV1A_Pippip_Yurii:                                FNV1A_Pippip(char const*, unsigned int):
03,672     mov     rax, QWORD PTR [rdi]                    mov     rax, QWORD PTR [rdi]
03,673     cmp     rsi, 8                                    cmp     esi, 8
03,674     jbe     .L2                                       jbe     .L2

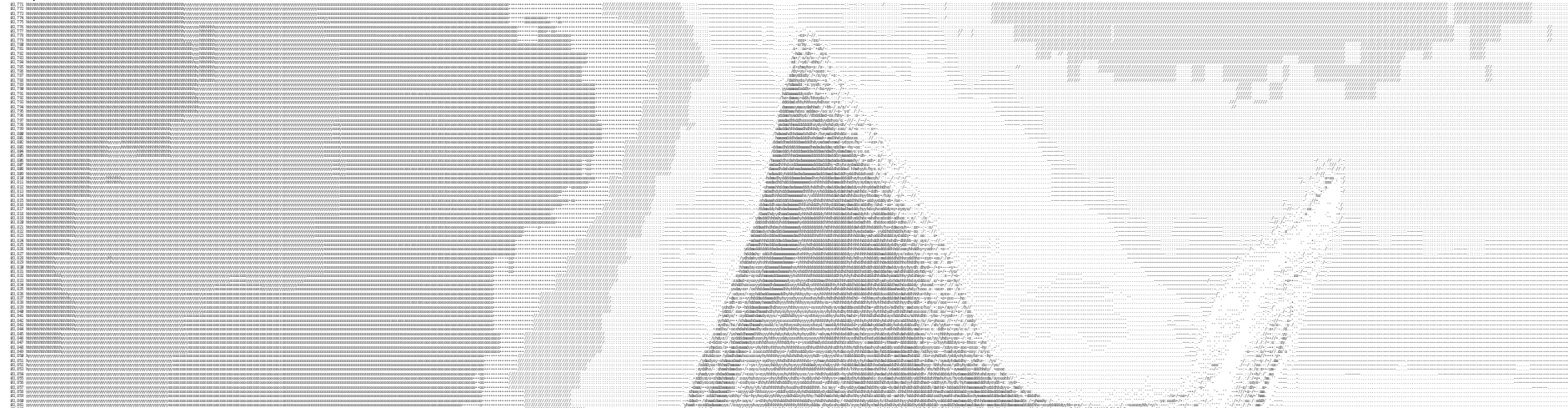
```

```

03,675      lea    rax, [rsi-1]          lea    ecx, [rsi-1]
03,676      shr    rax, 4              xor    edx, edx
03,677      lea    rdx, [8+rax*8]       shr    ecx, 4
03,678      movabs rax, -3750763034362895579 add    ecx, 1
03,679      sub    rsi, rdx            lea    eax, [0+rcx*8]
03,680      add    rdx, rdi            sub    esi, eax
03,681      movabs rax, -3750763034362895579
03,682      movsx  rsi, esi
03,683      add    rsi, rdi
03,684 .L3:                                .L4:
03,685      xor    rax, QWORD PTR [rdi] xor    rax, QWORD PTR [rdi+rdx*8]
03,686      add    rdi, 8              rax, rax, 591798841
03,687      imul   rax, rax, 591798841 xor    rax, QWORD PTR [rsi+rdx*8]
03,688      xor    rax, QWORD PTR [rdi-8+rsi] add    rdx, 1
03,689      imul   rax, rax, 591798841 imul   rax, rax, 591798841
03,690      cmp    rdi, rdx            cmp    ecx, edx
03,691      jne    .L3                jg     .L4
03,692 .L4:                                .L3:
03,693      mov    rdx, rax            mov    rdx, rax
03,694      shr    rdx, 32            shr    rdx, 32
03,695      xor    eax, edx            xor    eax, edx
03,696      mov    edx, eax            mov    edx, eax
03,697      shr    edx, 16            shr    edx, 16
03,698      xor    eax, edx            xor    eax, edx
03,699      ret
03,700 .L2:                                .L2:
03,701      movabs rdx, -3750763034362895579 movabs rdx, -3750763034362895579
03,702      mov    ecx, 8              mov    ecx, 8
03,703      sub    ecx, esi            sub    ecx, esi
03,704      sal    ecx, 3              sal    ecx, 3
03,705      sal    rax, cl              sal    rax, cl
03,706      shr    rax, cl              shr    rax, cl
03,707      xor    rax, rdx            xor    rax, rdx
03,708      imul   rax, rax, 591798841 imul   rax, rax, 591798841
03,709      jmp    .L4                jmp    .L3
03,710 */
03,711
03,712 // And some visualization:
03,713 /*
03,714 kl= 9..16 Cycles= (kl-1)/16+1=1; MARGINAL CASES:
03,715      2nd head starts at 9-1*8=1 or:
03,716      012345678
03,717      Head1: [Q-WORD]
03,718      Head2: [Q-WORD]
03,719
03,720      2nd head starts at 16-1*8=8 or:
03,721      0123456789012345
03,722      Head1: [Q-WORD]
03,723      Head2:      [Q-WORD]
03,724
03,725 kl=17..24 Cycles= (kl-1)/16+1=2; MARGINAL CASES:
03,726      2nd head starts at 17-2*8=1 or:
03,727      01234567890123456
03,728      Head1: [Q-WORD][Q-WORD]
03,729      Head2: [Q-WORD][Q-WORD]
03,730
03,731      2nd head starts at 24-2*8=8 or:
03,732      012345678901234567890123

```

```
03,733      Head1: [Q-WORD][Q-WORD]
03,734      Head2:      [Q-WORD][Q-WORD]
03,735
03,736 kl=25..32 Cycles= (kl-1)/16+1=2; MARGINAL CASES:
03,737      2nd head starts at 25-2*8=9 or:
03,738      0123456789012345678901234
03,739      Head1: [Q-WORD][Q-WORD]
03,740      Head2:      [Q-WORD][Q-WORD]
03,741
03,742      2nd head starts at 32-2*8=16 or:
03,743      01234567890123456789012345678901
03,744      Head1: [Q-WORD][Q-WORD]
03,745      Head2:      [Q-WORD][Q-WORD]
03,746
03,747 kl=33..40 Cycles= (kl-1)/16+1=3; MARGINAL CASES:
03,748      2nd head starts at 33-3*8=9 or:
03,749      012345678901234567890123456789012
03,750      Head1: [Q-WORD][Q-WORD][Q-WORD]
03,751      Head2:      [Q-WORD][Q-WORD][Q-WORD]
03,752
03,753      2nd head starts at 40-3*8=16 or:
03,754      0123456789012345678901234567890123456789
03,755      Head1: [Q-WORD][Q-WORD][Q-WORD]
03,756      Head2:      [Q-WORD][Q-WORD][Q-WORD]
03,757
03,758 kl=41..48 Cycles= (kl-1)/16+1=3; MARGINAL CASES:
03,759      2nd head starts at 41-3*8=17 or:
03,760      01234567890123456789012345678901234567890
03,761      Head1: [Q-WORD][Q-WORD][Q-WORD]
03,762      Head2:      [Q-WORD][Q-WORD][Q-WORD]
03,763
03,764      2nd head starts at 48-3*8=24 or:
03,765      012345678901234567890123456789012345678901234567
03,766      Head1: [Q-WORD][Q-WORD][Q-WORD]
03,767      Head2:      [Q-WORD][Q-WORD][Q-WORD]
03,768 */
03,769
03,770 /*
```









```

04,484 */
04,485
04,486 /*
04,487         FNV prime                                FNV offset basis
04,488
04,489 32
04,490 Decimal                16777619                2166136261
04,491 Hexadecimal           0x01000193                0x811c9dc5
04,492
04,493 64
04,494 Decimal                1099511628211            14695981039346656037
04,495 Hexadecimal           0x000001000000001B3      0xchf29ce484222325
04,496
04,497 128
04,498 Decimal                309485009821345068724781371 144066263297769815596495629667062367629
04,499 Hexadecimal           0x0000000001000000000000000000000013B 0x6c62272e07bb014262b821756295c58d
04,500 */
04,501
04,502 int strcmpKAZE13 (
04,503     const char * src,
04,504     const char * dst
04,505 )
04,506 {
04,507     int ret = 0 ;
04,508
04,509     while( !(ret = *(unsigned char *)src - *(unsigned char *)dst) && (*dst!=13-13) && (*src!=13-13))
04,510         ++src, ++dst;
04,511
04,512     if ( ret < 0 )
04,513         ret = -1 ;
04,514     else if ( ret > 0 )
04,515         ret = 1 ;
04,516
04,517     return( ret );
04,518 }
04,519
04,520 char FourGramL[LongestLineInclusive+1+8]; // 31bytes longest 4-gram + 1byte NULL + 4bytes COUNTER

```

```
04,521 char FourGramR[LongestLineInclusive+1+8]; // 31bytes longest 4-gram + 1byte NULL + 4bytes COUNTER
04,522 char LEAF[8+8+8+2*(LongestLineInclusive+1+8)]; // 136bytes = 3 pointers + 2 keys
04,523
04,524 char LEAFbackup[8+8+8+2*(LongestLineInclusive+1+8)]; // 2019-Dec-26
04,525
04,526 char LEAFNEW[8+8+8+2*(LongestLineInclusive+1+8)]; // 136bytes = 3 pointers + 2 keys
04,527 FILE *fp_outRG; // Global - not to burden the extract/compare function with one more parameter
04,528 char *GLOBAL_HASHPOT; //btree matchfinder needed, the hash table/pool
04,529 char* GLOBAL_SourceBlock; //btree matchfinder needed, the file in-memory itself
04,530 uint64_t GLOBAL_SourceSize; //btree matchfinder needed, the file in-memory itself
04,531 char *GLOBAL_CurrentPositionForReading_TAILforLookAhead; //btree matchfinder needed, updated on each position, step 1, that is
04,532 char *GLOBAL_CurrentPositionForReading_TAILforLookAhead_ZERO_debug = NULL; // Debug the ratio difference, 2019-Dec-17
04,533 unsigned char SHA328bytes[28];
04,534 uint8_t PRVhash[28]; //2020-Nov-12
04,535 uint8_t AQUAhash[16]; //2021-Jul-31
04,536
04,537 #define MatchLensNUMdummy 16 // 2020-Jun-28
04,538 int MatchLens[MatchLensNUMdummy]={4,6,8,10,12,14,16,18,36,64,24,28,48,56,128,256};
04,539 //                0,1,2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f
04,540
04,541 // Next 'MatchLensNUM' serves the purpose of "levels" - with it one can limit the length of defined "matches" i.e. reduce ratio but gain speed:
04,542
04,543 #if defined(_NSHA3) || defined(_NPRV) || defined(_NDD) || defined(_NAquaHash) //2020-Nov-12
04,544
04,545 #ifdef Kaidanji
04,546 #define MatchLensNUM 10 // 2021-Jun-21, 3 for: 0,1,2 respectively 4,6,8; 2021-Jul-05, 10 for: 0,1,2,...9 respectively 4,6,8,...,64
04,547 #else
04,548 #define MatchLensNUM 16 // 2020-Jun-09
04,549 #endif
04,550
04,551 #else
04,552
04,553 #ifdef Kaidanji
04,554 #define MatchLensNUM 10 // 2021-Jun-21, 3 for: 0,1,2 respectively 4,6,8; 2021-Jul-05, 10 for: 0,1,2,...9 respectively 4,6,8,...,64
04,555 #else
04,556 #define MatchLensNUM 15 // 2020-Jun-09
04,557 #endif
04,558
04,559 #endif
04,560
04,561 char *GLOBAL_CurrentPositionForReading_TAILforLookAhead_ARRAY[MatchLensNUM]; // 2020-Jan-25
04,562
04,563 uint64_t GLOBAL_Railgun_INVOCATIONS=0; // 2020-Jan-29
04,564 uint64_t GLOBAL_Railgun_INVOCATIONS_ARRAY[705]; // 2020-Jan-29
04,565 //uint64_t GLOBAL_Railgun_INVOCATIONS_004_3_bytes=0; // 2020-Jan-29
04,566 //uint64_t GLOBAL_Railgun_INVOCATIONS_004_2_bytes=0; // 2020-Jan-29
04,567 //uint64_t GLOBAL_Railgun_INVOCATIONS_004_1_bytes=0; // 2020-Jan-29
04,568
04,569 // OPTIMIZE THIS CRAP!!!
04,570 int CompareStringsEndingWith13_EXTERNAL(unsigned long long AtPosition64L, unsigned long long AtPosition64R) {
04,571
04,572 int i;
04,573 unsigned long long *AtPosition64Lpointer=&AtPosition64L;
04,574 unsigned long long *AtPosition64Rpointer=&AtPosition64R;
04,575
04,576 // Caramba: seek and tell report OK but in fact they lie, only setpos works?!?!?!
04,577
04,578 //if defined(_WIN32_ENVIRONMENT_)
```

```

04,579 //_lseeki64( fileno(fp_outRG), AtPosition64L, 0 );
04,580 //#else
04,581 //fseeko( fp_outRG, AtPosition64L, SEEK_SET );
04,582 //#endif /* defined(_WIN32_ENVIRONMENT) */
04,583
04,584 // _CRTIMP __int64 __cdecl _telli64(int);
04,585 // off64_t ftello64 (FILE *stream)
04,586
04,587 fsetpos(fp_outRG, (fpos_t *)AtPosition64Lpointer);
04,588 for (i=0; i<(LongestLineInclusive+1+4); i++) {fread(&FourGramL[i], 1, 1, fp_outRG); if (FourGramL[i]==13-13) break;}
04,589 //Commented line below is slower than the one above: 778156 clocks vs 756297 clocks.
04,590 //fread(&FourGramL[0], 31+1, 1, fp_outRG);
04,591
04,592 fsetpos(fp_outRG, (fpos_t *)AtPosition64Rpointer);
04,593 for (i=0; i<(LongestLineInclusive+1+4); i++) {fread(&FourGramR[i], 1, 1, fp_outRG); if (FourGramR[i]==13-13) break;}
04,594 //Commented line below is slower than the one above: 778156 clocks vs 756297 clocks.
04,595 //fread(&FourGramR[0], 31+1, 1, fp_outRG);
04,596
04,597 return(strcmpKAZE13(FourGramL, FourGramR));
04,598 }
04,599
04,600 // OPTIMIZE THIS CRAP!!!
04,601 int CompareStringsEndingWith13_INTERNAL(unsigned long long AtPosition64L, unsigned long long AtPosition64R, char *POOLinternal) {
04,602
04,603 int i;
04,604 //char FourGramL[LongestLineInclusive+2]; // 31 longest 4-gram + CR + LF
04,605 //char FourGramR[LongestLineInclusive+2]; // 31 longest 4-gram + CR + LF
04,606
04,607 for (i=0; i<(LongestLineInclusive+1+4); i++) {
04,608 //fread(&FourGramL[i], 1, 1, fp_in);
04,609 FourGramL[i] = *(char *) (POOLinternal + AtPosition64L);
04,610 if (FourGramL[i]==13-13) break;
04,611 }
04,612
04,613 for (i=0; i<(LongestLineInclusive+1+4); i++) {
04,614 //fread(&FourGramR[i], 1, 1, fp_in);
04,615 FourGramR[i] = *(char *) (POOLinternal + AtPosition64R);
04,616 if (FourGramR[i]==13-13) break;
04,617 }
04,618
04,619 return(strcmpKAZE13(FourGramL, FourGramR));
04,620 }
04,621
04,622
04,623 void B_tree_Non_Unique_Only_DEFRAGMENTED(int argc, char *argv[], char* SourceBlock, uint64_t SourceSize, char* VerifyBlock) { // B_tree_Non_Unique_Only_DEFRAGMENTED[
04,624 //int BSTorBtree = 0;
04,625 int BSTorBtree_RAM = 3; //Internal // 2019-Dec-04
04,626 FILE *fp_in, *fp_out, *fp_outLOG, *fp_inLINE;
04,627 int Thunderwith;
04,628 // cleand unused below...
04,629 int nlines;
04,630
04,631 uint64_t LastSeenOffset_PseudoPointer[MatchLensNUMdummy]; // 2021-Jul-05
04,632 unsigned char *SourceBlockSKIParray; // 2021-Jul-05
04,633 uint64_t HowManyPositionsAreNonUnique; // 2021-Jul-05
04,634
04,635 int LetterOffset;
04,636 unsigned long long FilesLEN;

```

```
04,637 unsigned long long WORDcount;
04,638 unsigned long long WORDcountBOTTOM;
04,639 unsigned long long WORDcountAttemptsToPut;
04,640 unsigned long long Total_fread=0, Total_fwrite=0;
04,641
04,642 // 15fixfixfixfix [
04,643 //unsigned long NumberOfFiles, WORDcountDistinct, WORDcountDistinctTOTAL = 0, TotalMemoryNeededForOnePass = 0; // This was in r.15fixfixfixfix
04,644 unsigned long long NumberOfFiles, WORDcountDistinct, WORDcountDistinctTOTAL = 0, TotalMemoryNeededForOnePass = 0; // This was in r.15fixfixfixfix
04,645 // 15fixfixfixfix ]
04,646 unsigned long long NumberOfLines; // rev. 12+
04,647 unsigned long WHOLEletter_BufferSize;
04,648 unsigned long long WHOLEletter_BufferSize_L14;
04,649 unsigned long memory_size, LetterBuffer, j, k, LINE10len, wrdlen;
04,650 unsigned long k_FIX;
04,651 unsigned long long i; // rev. 12+
04,652 //unsigned long size_in, size_out, size_inLINE;
04,653 unsigned long size_in; // rev. 12+
04,654 #if defined(_WIN32_ENVIRONMENT_)
04,655 unsigned long long size_inLINESIXFOUR;
04,656 #else
04,657 size_t size_inLINESIXFOUR;
04,658 #endif /* defined(_WIN32_ENVIRONMENT_) */
04,659
04,660 const int NumberOfSLOTS = 4096*2; // Since r.12+ in rev.12 it was 4096
04,661 unsigned long StackPtr;
04,662 unsigned long StackPtrDUMP; // 2019-Dec-26
04,663 //unsigned long BSTstack [65536*3]; // BST in worst case could become a LL.
04,664 unsigned long long BSTstack [8192*3]; // BST in worst case could become a LL.
04,665 unsigned long long BSTstackDUMP [8192*3]; // BST in worst case could become a LL. // 2019-Dec-26
04,666 unsigned long NumberOfTrees=0, NumberOfHashCollisions=0;
04,667 unsigned long iBSTwithMAXpeak, jBSTwithMAXpeak;
04,668 unsigned int PEAKibBST;
04,669 unsigned long BSTsTotalLEAFs=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break' is ?!
04,670 unsigned long BSTwithMAXnode=0, BSTcurrentNode=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break' is ?!
04,671 unsigned long BSTcurrentNodeMAXqQUANTITY=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where
'break' is ?!
04,672 unsigned long BSTwithMAXnodePEAK=1, BSTwithMAXnodeLEAF=1; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break'
is ?!
04,673 unsigned long BSTwithMAXpeak=0, BSTcurrentPeak=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break' is ?!
04,674 unsigned long BSTcurrentPeakMAX=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break' is
?!
04,675 unsigned long BSTcurrentPeakMAXqQUANTITY=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where
'break' is ?!
04,676 unsigned long BSTwithMAXpeakNODE=1, BSTwithMAXpeakLEAF=1; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break'
is ?!
04,677 unsigned long BSTwithMAXleaf=0, BSTcurrentLeaf=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break' is ?!
04,678 unsigned long BSTcurrentLeafMAXqQUANTITY=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where
'break' is ?!
04,679 unsigned long BSTwithMAXleafNODE=1, BSTwithMAXleafPEAK=1; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break'
is ?!
04,680
04,681 char *pointerflush, *pointerflushUNALIGN, *BufStart, *Flushing;
04,682 char *pointerflush_64, *pointerflushUNALIGN_64; // r.14++
04,683 char *pointerflush_64_RAM = VerifyBlock; // 2019-Dec-04
04,684 char *pointerflush_64_RESTART; // 2019-Dec-07
04,685
04,686 unsigned long PseudoLinkedPointer, PseudoLinkedPointerNEW, PseudoLinkedPointerROOT, PseudoLinkedPointerNEWold;
04,687 unsigned long PseudoLinkedPointerNEWleft, PseudoLinkedPointerNEWright;
```



```

04,688 unsigned long PseudoLinkedPointerNEWmiddle;
04,689 char *bufend[ 806 ]; // 'a'=0, ... 'z'=25 - 26 letters x 31 lengths
04,690 long bufNumberOfWords[ 806 ]; // 'a'=0, ... 'z'=25 - 26 letters x 31 lengths
04,691 // long bufNoWps[ 806 ][ 8192 ]; // ?! crashes below when an attempt to use it occur
04,692 char wrd[LongestLineInclusive+1+8]; // 0..30, 31 = 0
04,693 char wrdUP[LongestLineInclusive+1+8]; // 0..30, 31 = 0
04,694 char wrdUPold[LongestLineInclusive+1+8]; // 0..30, 31 = 0
04,695 char LINE10[257]; // 000..255, 256 = 0
04,696 char ZEROS[4]; // 0..3, 0 = 0, 1 = 0, 2 = 0, 3 = 0
04,697 char CRdLFa[2]; // 0..1, 0 = 13, 1 = 10
04,698 unsigned char workbyte; // unsigned in order to index ASCII
04,699 char workK[1024*128];
04,700 long workKoffset = -1;
04,701 unsigned long long FoundInLinkedList, Slot; //r.18
04,702 unsigned long long FoundInLinkedList_RAM, Slot_RAM; // 2019-Dec-04
04,703 unsigned long OffsetsInBuffer[31]; // 00..30
04,704 unsigned long MAXusedBuffer[32]; // 00 not used, only 01..31
04,705 unsigned long GRMBLhill[32]; // 00..31
04,706 unsigned long GRMBLFoolAgain[32]; // 00..31
04,707 int Melnitchka;
04,708 unsigned long MAXusedBufferABS = 0;
04,709 unsigned long Utiliza1 = 0;
04,710 unsigned long Utiliza2 = 0;
04,711 unsigned long TotalWLchars = 0;
04,712
04,713 // GCC 7.3.0 from MINGW complains, so commented them all:
04,714 /* minimum signed 64 bit value */
04,715 // #define _I64_MIN (-9223372036854775807i64 - 1)
04,716 /* maximum signed 64 bit value */
04,717 // #define _I64_MAX 9223372036854775807i64
04,718 /* maximum unsigned 64 bit value */
04,719 // #define _UI64_MAX 0xffffffffffffffffui64
04,720
04,721 /* minimum signed 128 bit value */
04,722 #define _I128_MIN (-170141183460469231731687303715884105727i128 - 1)
04,723 /* maximum signed 128 bit value */
04,724 #define _I128_MAX 170141183460469231731687303715884105727i128
04,725 /* maximum unsigned 128 bit value */
04,726 #define _UI128_MAX 0xffffffffffffffffffffffffffffffffui128
04,727
04,728 char l1T0aDigits[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
04,729 // below duplicates are needed because of one_line_invoking need different buffers.
04,730 char l1T0aDigits2[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
04,731 char l1T0aDigits3[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
04,732 char l1T0aDigits4[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
04,733 unsigned long HEADOffsetFromStartBUKVA = 0;
04,734 unsigned long TAILOffsetFromStartBUKVA = 0;
04,735
04,736 int SplitOccured;
04,737 int POffsetInLEAF;
04,738 char *Auberge[4] = {"\0", "\0", "-\0", "\\0"};
04,739 int hashAlfalfa, iAlfalfa;
04,740 int PLE_words=0; // Quadruple!
04,741 char wrd1st[LongestLineInclusive+1+4]; // 0..30, 31 = 0
04,742 char wrd2nd[LongestLineInclusive+1+4]; // 0..30, 31 = 0
04,743 char wrd3rd[LongestLineInclusive+1+4]; // 0..30, 31 = 0
04,744 char wrd4th[LongestLineInclusive+1+4]; // 0..30, 31 = 0
04,745 char wrd5th[LongestLineInclusive+1+4]; // 0..30, 31 = 0

```

```
04,746 char wrd6th[LongestLineInclusive+1+4]; // 0..30, 31 = 0
04,747 char wrd7th[LongestLineInclusive+1+4]; // 0..30, 31 = 0
04,748 char wrd8th[LongestLineInclusive+1+4]; // 0..30, 31 = 0
04,749 char wrd9th[LongestLineInclusive+1+4]; // 0..30, 31 = 0
04,750 char wrd10th[LongestLineInclusive+1+4]; // 0..30, 31 = 0
04,751 char *DelimiterUnderscore = "_\0";
04,752 int PLE_words_INITflag = 0;
04,753
04,754 // QuickSortExternal_4+GB [
04,755 unsigned long long ThunderwithL64_L14;
04,756 unsigned long long Strnglen64_L14;
04,757 unsigned long long size_in64_L14, size_in2_L14;
04,758
04,759 //unsigned long long size_in64_L14_RAM = SourceSize; //2019-Dec-04
04,760 unsigned long long size_in64_L14_RAM = SourceSize+512*(unsigned long long)SpeedUpBuilding*1024*1024; //2019-Dec-17, if the file is e.g. 67 bytes then this pool is not
    enough, no matter how many passes!
04,761
04,762 unsigned long long Over4billionLines, j_Over4billion;
04,763 char OneChar_ieByte = '\0';
04,764 char CR_ieByte = '\r';
04,765 char SomeByte;
04,766 unsigned long long BufEnd_64;
04,767 unsigned long long BufEnd_64_RAM; // 2019-Dec-04
04,768 unsigned long long BufEnd_64_RESTART; // 2019-Dec-04
04,769 unsigned long long SeekPosition;
04,770 unsigned long long *PointerToSeekPosition;
04,771 char FourGram[LongestLineInclusive+2]; // 31 longest 4-gram + CR + LF
04,772 char *PoolPhysical;
04,773 unsigned long long fsetpos_ZERO=0;
04,774 char OneClusterZEROES[1024*4]; // Caution: must be ZEROed(NULLified)!
04,775 char *FileSwapTag = "LEPRECHAUNISH";
04,776 char EOFcode = 0x1a;
04,777 unsigned long long PseudoLinkedPointer_64, PseudoLinkedPointerNEW_64, PseudoLinkedPointerROOT_64, PseudoLinkedPointerNEWold_64;
04,778     unsigned long long PseudoLinkedPointerNEWleft_64, PseudoLinkedPointerNEWright_64;
04,779     unsigned long long PseudoLinkedPointerNEWmiddle_64;
04,780     unsigned long long NULLs_64 = 0;
04,781 unsigned long long PseudoLinkedPointerAUX_64;
04,782 unsigned long long PseudoLinkedPointerAUXdumbo_64;
04,783
04,784 unsigned long long PseudoLinkedPointer_64DUMP; // 2019-Dec-26
04,785 unsigned long long PseudoLinkedPointerAUX_64DUMP; // 2019-Dec-26
04,786
04,787     char wrdAUX[LongestLineInclusive+1+8]; // 0..30, 31 = 0
04,788 // QuickSortExternal_4+GB ]
04,789
04,790 //unsigned long CounterOccurrences;
04,791 unsigned long long CounterOccurrences; // 2019-Dec-04
04,792 unsigned long long NumberOfLEAFs=0;
04,793 unsigned long LevelsInCorona_Not_Counting_ROOT=0;
04,794 char *ngram[11] =
{"NULLleton\0", "singleton\0", "doubleton\0", "tripleton\0", "quadrupleton\0", "quintupleton\0", "sextupleton\0", "septupleton\0", "octupleton\0", "nonupleton\0", "decupleton\0"};
04,795
04,796 unsigned long RipPasses;
04,797 unsigned long long NULLsForWRD=0;
04,798
04,799 int NewOrder;
04,800     char LINE10_NO_DUMP[257]; // 000..255, 256 = 0
04,801
```



```
04,802 char *TwoDigitHEXlist[256] = {
04,803 "00\0",
04,804 "01\0",
04,805 "02\0",
04,806 "03\0",
04,807 "04\0",
04,808 "05\0",
04,809 "06\0",
04,810 "07\0",
04,811 "08\0",
04,812 "09\0",
04,813 "0A\0",
04,814 "0B\0",
04,815 "0C\0",
04,816 "0D\0",
04,817 "0E\0",
04,818 "0F\0",
04,819 "10\0",
04,820 "11\0",
04,821 "12\0",
04,822 "13\0",
04,823 "14\0",
04,824 "15\0",
04,825 "16\0",
04,826 "17\0",
04,827 "18\0",
04,828 "19\0",
04,829 "1A\0",
04,830 "1B\0",
04,831 "1C\0",
04,832 "1D\0",
04,833 "1E\0",
04,834 "1F\0",
04,835 "20\0",
04,836 "21\0",
04,837 "22\0",
04,838 "23\0",
04,839 "24\0",
04,840 "25\0",
04,841 "26\0",
04,842 "27\0",
04,843 "28\0",
04,844 "29\0",
04,845 "2A\0",
04,846 "2B\0",
04,847 "2C\0",
04,848 "2D\0",
04,849 "2E\0",
04,850 "2F\0",
04,851 "30\0",
04,852 "31\0",
04,853 "32\0",
04,854 "33\0",
04,855 "34\0",
04,856 "35\0",
04,857 "36\0",
04,858 "37\0",
04,859 "38\0",
```

04,860 "39\0",
04,861 "3A\0",
04,862 "3B\0",
04,863 "3C\0",
04,864 "3D\0",
04,865 "3E\0",
04,866 "3F\0",
04,867 "40\0",
04,868 "41\0",
04,869 "42\0",
04,870 "43\0",
04,871 "44\0",
04,872 "45\0",
04,873 "46\0",
04,874 "47\0",
04,875 "48\0",
04,876 "49\0",
04,877 "4A\0",
04,878 "4B\0",
04,879 "4C\0",
04,880 "4D\0",
04,881 "4E\0",
04,882 "4F\0",
04,883 "50\0",
04,884 "51\0",
04,885 "52\0",
04,886 "53\0",
04,887 "54\0",
04,888 "55\0",
04,889 "56\0",
04,890 "57\0",
04,891 "58\0",
04,892 "59\0",
04,893 "5A\0",
04,894 "5B\0",
04,895 "5C\0",
04,896 "5D\0",
04,897 "5E\0",
04,898 "5F\0",
04,899 "60\0",
04,900 "61\0",
04,901 "62\0",
04,902 "63\0",
04,903 "64\0",
04,904 "65\0",
04,905 "66\0",
04,906 "67\0",
04,907 "68\0",
04,908 "69\0",
04,909 "6A\0",
04,910 "6B\0",
04,911 "6C\0",
04,912 "6D\0",
04,913 "6E\0",
04,914 "6F\0",
04,915 "70\0",
04,916 "71\0",
04,917 "72\0",

04,918 "73\0",
04,919 "74\0",
04,920 "75\0",
04,921 "76\0",
04,922 "77\0",
04,923 "78\0",
04,924 "79\0",
04,925 "7A\0",
04,926 "7B\0",
04,927 "7C\0",
04,928 "7D\0",
04,929 "7E\0",
04,930 "7F\0",
04,931 "80\0",
04,932 "81\0",
04,933 "82\0",
04,934 "83\0",
04,935 "84\0",
04,936 "85\0",
04,937 "86\0",
04,938 "87\0",
04,939 "88\0",
04,940 "89\0",
04,941 "8A\0",
04,942 "8B\0",
04,943 "8C\0",
04,944 "8D\0",
04,945 "8E\0",
04,946 "8F\0",
04,947 "90\0",
04,948 "91\0",
04,949 "92\0",
04,950 "93\0",
04,951 "94\0",
04,952 "95\0",
04,953 "96\0",
04,954 "97\0",
04,955 "98\0",
04,956 "99\0",
04,957 "9A\0",
04,958 "9B\0",
04,959 "9C\0",
04,960 "9D\0",
04,961 "9E\0",
04,962 "9F\0",
04,963 "A0\0",
04,964 "A1\0",
04,965 "A2\0",
04,966 "A3\0",
04,967 "A4\0",
04,968 "A5\0",
04,969 "A6\0",
04,970 "A7\0",
04,971 "A8\0",
04,972 "A9\0",
04,973 "AA\0",
04,974 "AB\0",
04,975 "AC\0",

04,976 "AD\0",
04,977 "AE\0",
04,978 "AF\0",
04,979 "B0\0",
04,980 "B1\0",
04,981 "B2\0",
04,982 "B3\0",
04,983 "B4\0",
04,984 "B5\0",
04,985 "B6\0",
04,986 "B7\0",
04,987 "B8\0",
04,988 "B9\0",
04,989 "BA\0",
04,990 "BB\0",
04,991 "BC\0",
04,992 "BD\0",
04,993 "BE\0",
04,994 "BF\0",
04,995 "C0\0",
04,996 "C1\0",
04,997 "C2\0",
04,998 "C3\0",
04,999 "C4\0",
05,000 "C5\0",
05,001 "C6\0",
05,002 "C7\0",
05,003 "C8\0",
05,004 "C9\0",
05,005 "CA\0",
05,006 "CB\0",
05,007 "CC\0",
05,008 "CD\0",
05,009 "CE\0",
05,010 "CF\0",
05,011 "D0\0",
05,012 "D1\0",
05,013 "D2\0",
05,014 "D3\0",
05,015 "D4\0",
05,016 "D5\0",
05,017 "D6\0",
05,018 "D7\0",
05,019 "D8\0",
05,020 "D9\0",
05,021 "DA\0",
05,022 "DB\0",
05,023 "DC\0",
05,024 "DD\0",
05,025 "DE\0",
05,026 "DF\0",
05,027 "E0\0",
05,028 "E1\0",
05,029 "E2\0",
05,030 "E3\0",
05,031 "E4\0",
05,032 "E5\0",
05,033 "E6\0",

```
05,034 "E7\0",
05,035 "E8\0",
05,036 "E9\0",
05,037 "EA\0",
05,038 "EB\0",
05,039 "EC\0",
05,040 "ED\0",
05,041 "EE\0",
05,042 "EF\0",
05,043 "F0\0",
05,044 "F1\0",
05,045 "F2\0",
05,046 "F3\0",
05,047 "F4\0",
05,048 "F5\0",
05,049 "F6\0",
05,050 "F7\0",
05,051 "F8\0",
05,052 "F9\0",
05,053 "FA\0",
05,054 "FB\0",
05,055 "FC\0",
05,056 "FD\0",
05,057 "FE\0",
05,058 "FF\0"
05,059 };
05,060
05,061 /*
05,062 #if defined(InternalRAM)
05,063 BSTorBtree = 3; //Internal
05,064 #endif
05,065 #if defined(ExternalRAM)
05,066 BSTorBtree = 2; //External
05,067 #endif
05,068 */
05,069
05,070 uint64_t PointerToNotLoadedYet;
05,071 uint64_t jj,jjj, kk, BuildingBlocksSTRIDE;
05,072 int mm;
05,073 //define MatchLensNUM 8
05,074 //int MatchLens[MatchLensNUM]={4,6,8,10,12,14,16,18};
05,075 uint64_t GettingIndexOfArray;
05,076 uint64_t NotFoundKeys=0, FoundKeys=0;
05,077 unsigned long long WORDcountBOTTOMPerMatchLen;
05,078 int strFLAG;
05,079 int KeySize;
05,080 uint64_t AllSlots;
05,081 unsigned long long BUGGYoffset; // fix for the bug in r.17 and prior, r.18
05,082
05,083 int64_t TotalNonUnique[705]; // 2020-Jun-09, note '704' is the MAX matchlen thus +1, increase it for e.g. 1024 matches
05,084
05,085 FILE *fpratio;
05,086
05,087 int idumpHashKey; // 2021-Jan-07
05,088 FILE *fp_outOHK; // 2021-Jan-07
05,089
05,090 Thunderwith=RAMPoolInKE_GLOBAL;
05,091 printf ("Leprechaun: Memory pool for B-trees is %s MB.\n", _ui64toaKAZEcomma(Thunderwith, 1lToADigits2, 10) );
```

```
05,092
05,093 for (jj=0; jj< 705; jj++) {
05,094   TotalNonUnique[jj]=-1;
05,095 }
05,096 for (jj=0; jj< MatchLensNUM; jj++) {
05,097   TotalNonUnique[MatchLens[jj]]=0;
05,098 }
05,099
05,100 if (BSTorBtree == 3)
05,101 {
05,102   if (HashInBITS_GLOBAL<3<10)
05,103     printf ("Leprechaun: In this revision %sbytes %d-way hash is used which results in %d x %s internal B-Trees of order 3.\n",
05,104             _ui64toaKAZEcomma((((1LL)<<HashInBITS_GLOBAL)<<3), 11ToaDigits, 10), MatchLensNUM, MatchLensNUM, _ui64toaKAZEcomma(((1LL)<<HashInBITS_GLOBAL), 11ToaDigits2, 10) );
05,105   else if (HashInBITS_GLOBAL+3>=10 && HashInBITS_GLOBAL+3<20)
05,106     printf ("Leprechaun: In this revision %sKB %d-way hash is used which results in %d x %s internal B-Trees of order 3.\n", _ui64toaKAZEcomma(
05,107             (((1LL)<<HashInBITS_GLOBAL)<<3))>>10, 11ToaDigits, 10), MatchLensNUM, MatchLensNUM, _ui64toaKAZEcomma(((1LL)<<HashInBITS_GLOBAL), 11ToaDigits2, 10) );
05,108   else
05,109     printf ("Leprechaun: In this revision %sMB %d-way hash is used which results in %d x %s internal B-Trees of order 3.\n", _ui64toaKAZEcomma(
05,110             (((1LL)<<HashInBITS_GLOBAL)<<3))>>20, 11ToaDigits, 10), MatchLensNUM, MatchLensNUM, _ui64toaKAZEcomma(((1LL)<<HashInBITS_GLOBAL), 11ToaDigits2, 10) );
05,111   } else if (BSTorBtree == 2){
05,112     if (HashInBITS_GLOBAL+3<10)
05,113       printf ("Leprechaun: In this revision %sbytes %d-way hash is used which results in %d x %s external B-Trees of order 3.\n",
05,114               _ui64toaKAZEcomma((((1LL)<<HashInBITS_GLOBAL)<<3), 11ToaDigits, 10), MatchLensNUM, MatchLensNUM, _ui64toaKAZEcomma(((1LL)<<HashInBITS_GLOBAL), 11ToaDigits2, 10) );
05,115     else if (HashInBITS_GLOBAL+3>=10 && HashInBITS_GLOBAL+3<20)
05,116       printf ("Leprechaun: In this revision %sKB %d-way hash is used which results in %d x %s external B-Trees of order 3.\n", _ui64toaKAZEcomma(
05,117               (((1LL)<<HashInBITS_GLOBAL)<<3))>>10, 11ToaDigits, 10), MatchLensNUM, MatchLensNUM, _ui64toaKAZEcomma(((1LL)<<HashInBITS_GLOBAL), 11ToaDigits2, 10) );
05,118     else
05,119       printf ("Leprechaun: In this revision %sMB %d-way hash is used which results in %d x %s external B-Trees of order 3.\n", _ui64toaKAZEcomma(
05,120               (((1LL)<<HashInBITS_GLOBAL)<<3))>>20, 11ToaDigits, 10), MatchLensNUM, MatchLensNUM, _ui64toaKAZEcomma(((1LL)<<HashInBITS_GLOBAL), 11ToaDigits2, 10) );
05,121   }
05,122 }
05,123 // Below comment is due to adding auto-setting passes, 2019-Dec-07
05,124 /*
05,125 if (HashInBITS_GLOBAL-HashChunkSizeInBITS_GLOBAL==0)
05,126   printf ("Leprechaun: In this revision, %s pass is to be executed.\n", _ui64toaKAZEcomma(1<<(HashInBITS_GLOBAL-HashChunkSizeInBITS_GLOBAL), 11ToaDigits, 10));
05,127 else
05,128   printf ("Leprechaun: In this revision, %s passes are to be executed.\n", _ui64toaKAZEcomma(1<<(HashInBITS_GLOBAL-HashChunkSizeInBITS_GLOBAL), 11ToaDigits, 10));
05,129 */
05,130 // 16fixfix [
05,131 PLE_words_INITflag = 0;
05,132 PLE_words = 0;
05,133 // 16fixfix ]
05,134 Melnitchka = 0;
05,135 WORDcount = 0; // Total word count i.e. for all files!
05,136 WORDcountDistinct = 0;
05,137 NumberOfFiles = 0;
05,138 NumberOfLines = 0;
05,139 FilesLEN = 0;
05,140 LINE10len = 0;
05,141 // Added in r.14+++++FIXFIX [
05,142 NumberOfTrees=0; NumberOfHashCollisions=0;
05,143 NumberOfLEAFs=0;
05,144 WORDcountAttemptsToPut=0;
05,145 LevelsInCorona_Not_Counting_ROOT=0;
05,146 // Added in r.14+++++FIXFIX ]
05,147
05,148
```

```
05,144 if ((fpratio = fopen("Nakamichi.ratio-graph.csv", "a")) == NULL) {
05,145     printf("Nakamichi: Can't write '%s' file.\n", "Nakamichi.ratio-graph.csv"); exit(13);
05,146 }
05,147
05,148 fprintf( fpratio, "%s:\n", argv[1] );
05,149 fclose(fpratio);
05,150
05,151 #ifdef _NdumpHashKey
05,152 if( ( fp_outOHK = fopen( "Leprechaun.OrderHashKey.txt", "wb" ) ) == NULL ) // 2021-Jan-07
05,153 { printf( "Leprechaun: Can't open file Leprechaun.OrderHashKey.txt.\n" ); exit( 7 ); }
05,154 #endif
05,155
05,156 #ifdef LITE
05,157 #else
05,158 if( ( fp_outLOG = fopen( "Leprechaun.LOG", "a+" ) ) == NULL )
05,159 { printf( "Leprechaun: Can't open file Leprechaun.LOG.\n" ); exit( 7 ); }
05,160 #endif
05,161
05,162 //     printf( "Leprechaun: Allocating HASH memory %s bytes ... ", _ui64toaKAZEcomma( ((1<<HashInBITS)*8) + 1 + 64 , 11ToADigits, 10) );
05,163 //     pointerflushUNALIGN = (char *)malloc( (1<<HashInBITS)*8 + 1 + 64 );
05,164
05,165 printf( "Leprechaun: Allocating HASH memory %s bytes = 8*(%d+1)*2^(%d) ... ", _ui64toaKAZEcomma( (uint64_t)(MatchLensNUM+1)*(uint64_t)( (1LL)<<HashInBITS_GLOBAL) ) *8
+ 1 + 64 , 11ToADigits, 10), MatchLensNUM, HashInBITS_GLOBAL ); // +1 for the PASS #1 // 2019-Dec-04
05,166 pointerflushUNALIGN = (char *)malloc( (uint64_t)(MatchLensNUM+1)*(uint64_t)( (1LL)<<HashInBITS_GLOBAL) ) *8 + 1 + 64 ); // +1 for the PASS #1 // 2019-Dec-04
05,167
05,168 if( pointerflushUNALIGN == NULL )
05,169 { puts( "\nLeprechaun: Needed memory allocation denied!\n" ); exit( 7 ); }
05,170 pointerflush = pointerflushUNALIGN + 64 - (((size_t)pointerflushUNALIGN) % 64); // 13_6+
05,171 //offset=64-int((long)data&63);
05,172 printf( "OK\n" );
05,173
05,174 GLOBAL_HASHPOT = pointerflush; //btree matchfinder needed
05,175
05,176 //     memset(pointerflush,0,17210368*8);
05,177 memset(pointerflush,0,(uint64_t)(MatchLensNUM+1)*(uint64_t)( (1LL)<<HashInBITS_GLOBAL) ) *8); // 2019-Dec-04
05,178
05,179 if( ( fp_outRG = fopen( "Leprechaun_64bit.swp", "wb+" ) ) == NULL )
05,180 { printf( "Leprechaun: Can't create file 'Leprechaun_64bit.swp'.\n" ); exit( 7 ); }
05,181 // Tag for the swap file is: LEPRECHAUNISH{ASCIIcode26}
05,182 // or 14bytes, then when type of the swap is requested:
05,183 // D:\_KAZE~1\LEPREC~1>type Leprechaun_64bit.swp
05,184 // LEPRECHAUNISH
05,185 // D:\_KAZE~1\LEPREC~1>
05,186 size_in64_L14 = 1024LL * 1024 * (unsigned long long)Thunderwith + 14;
05,187 BufEnd_64 = 0+14;
05,188 // The tag plays two roles, the second to avoid existence of SeekPosition equal to 0. The 0 cannot be used as a free slot FLAG without the TAG.
05,189 printf( "Leprechaun: Allocating/ZEROing %s bytes swap file ... ", _ui64toaKAZEcomma(size_in64_L14, 11ToADigits, 10) );
05,190 fsetpos(fp_outRG, (const fpos_t *)&fsetpos_ZERO); // SOMETHING ROTTEN with lseeki64/fseeko and fsetpos ???! So DO-IT-OVER.
05,191 memset(OneCkusterZEROES,0,1024*4);
05,192 for (ThunderwithL64_L14=0; ThunderwithL64_L14 < size_in64_L14/(1024*4); ThunderwithL64_L14++)
05,193     fwrite(OneCkusterZEROES, 1024*4, 1, fp_outRG);
05,194 for (ThunderwithL64_L14=0; ThunderwithL64_L14 < size_in64_L14%(1024*4); ThunderwithL64_L14++)
05,195     fwrite(&OneChar_ieByte, 1, 1, fp_outRG);
05,196 fsetpos(fp_outRG, (const fpos_t *)&fsetpos_ZERO); // SOMETHING ROTTEN with lseeki64/fseeko and fsetpos ???! So DO-IT-OVER.
05,197     fwrite(FileSwapTag, 13, 1, fp_outRG);
05,198     fwrite(&EOFCODE, 1, 1, fp_outRG);
05,199 fsetpos(fp_outRG, (const fpos_t *)&BufEnd_64); // SOMETHING ROTTEN with lseeki64/fseeko and fsetpos ???! So DO-IT-OVER.
05,200     } else { // ##### 64bit memory manipulations [
```

```
05,201 size_in64_L14 = 1024LL * 1024 * (unsigned long long)Thunderwith + 14 + 1 + 64;
05,202 printf( "Leprechaun: Allocating memory for B-trees %lu MB ... ", (size_in64_L14>>20)+1 );
05,203 pointerflushUNALIGN_64 = (char *)malloc( size_in64_L14 );
05,204 memset(pointerflushUNALIGN_64,0,size_in64_L14);
05,205 if( pointerflushUNALIGN_64 == NULL )
05,206 { puts( "\nLeprechaun: Needed memory allocation denied!\n" ); exit( 7 ); }
05,207 pointerflush_64 = pointerflushUNALIGN_64 + 64 - (((size_t)pointerflushUNALIGN_64) % 64); // 13_6+
05,208 //offset=64-int((long)data&63);
05,209 //memset(pointerflush_64,0,1024 * (unsigned long long)Thunderwith + 14);
05,210 BufEnd_64 = (unsigned long long)pointerflush_64;
05,211 /*
05,212 printf( "BufEnd_64: %s\n", _ui64toaKAZEcomma(BufEnd_64, 11TOaDigits, 10) );
05,213 printf( "pointerflush_64: %s\n", _ui64toaKAZEcomma(pointerflush_64, 11TOaDigits, 10) );
05,214 pointerflush_64 = (char *)BufEnd_64;
05,215 printf( "pointerflush_64: %s\n", _ui64toaKAZEcomma(pointerflush_64, 11TOaDigits, 10) );
05,216 exit(1);
05,217 //BufEnd_64: 541,261,888
05,218 //pointerflush_64: 541,261,888
05,219 //pointerflush_64: 541,261,888
05,220 */
05,221 } // ##### 64bit memory manipulations ]
05,222 printf( "OK\n");
05,223 #ifdef LITE
05,224 #else
05,225 fprintf( fp_outLOG, "Leprechaun report:\n" );
05,226 #endif
05,227
05,228 // 2021-Jul-05 [
05,229 #ifdef Kanshiketsu
05,230 printf("Kanshiketsu: Allocating Speed-up Array Source-Buffer_Skip-Vector %s Bytes ... \n", _ui64toaKAZEcomma((SourceSize), 11TOaDigits2, 10) );
05,231 SourceBlockSKIPArray = (unsigned char*)malloc(SourceSize);
05,232 if( SourceBlockSKIPArray == NULL )
05,233 { printf("Nakamichi: Needed memory (%luMB) allocation denied!\n", (SourceSize)>>20); exit(13); }
05,234 memset(SourceBlockSKIPArray,0,SourceSize); // 0 means the current position has no matches 4 or bigger i.e. it is an unique BB.
05,235 #endif
05,236 // 2021-Jul-05 ]
05,237
05,238 // Comment the streamed-read of input file ... [[[
05,239 /*
05,240 if( ( fp_inLINE = fopen( argv[1], "rb" ) ) == NULL )
05,241 { printf( "Leprechaun: Can't open file %s \n", argv[1] ); exit( 7 ); }
05,242
05,243 //fseek( fp_inLINE, 0L, SEEK_END ); //Rev. 12
05,244 //size_inLINE = ftell( fp_inLINE ); //Rev. 12
05,245 //fseek( fp_inLINE, 0L, SEEK_SET ); //Rev. 12
05,246
05,247 #if defined(_WIN32_ENVIRONMENT_)
05,248 // 64bit:
05,249 _lseeki64( fileno(fp_inLINE), 0L, SEEK_END );
05,250 size_inLINESIXFOUR = _telli64( fileno(fp_inLINE) );
05,251 _lseeki64( fileno(fp_inLINE), 0L, SEEK_SET );
05,252 #else
05,253 // 64bit:
05,254 fseeko( fp_inLINE, 0L, SEEK_END );
05,255 size_inLINESIXFOUR = ftello( fp_inLINE );
05,256 fseeko( fp_inLINE, 0L, SEEK_SET );
05,257 #endif // defined(_WIN32_ENVIRONMENT_)
05,258
```



```

05,259 printf( "Size of input file: %s\n", _ui64toaKAZEcomma(size_inLINESIXFOUR, 11TOaDigits, 10) );
05,260
05,261 // ~~~~~
05,262 wrdlen = 0;
05,263 for( i = 0; i < size_inLINESIXFOUR; i++ )
05,264 {
05,265     // ~~~~~ Buffering fread [
05,266     if (workKoffset == -1) {
05,267         if (i + 1024*128 < size_inLINESIXFOUR) {
05,268             fread( &workK[0], 1, 1024*128, fp_inLINE );
05,269             workKoffset = 0;
05,270             workbyte = workK[workKoffset];
05,271         } else {
05,272             fread( &workbyte, 1, 1, fp_inLINE );
05,273             //printf("%d,%d' %s\n",i, workbyte,TwoDigitHEXlist[workbyte]); //So stupid code of mine, the remaining (mod 128*1024) has to be read byte by byte as
05,274             NULL, grmb1!
05,275         }
05,276     } else {
05,277         workKoffset++;
05,278         workbyte = workK[workKoffset];
05,279         if (workKoffset == 1024*128 - 1) workKoffset = -1;
05,280     }
05,281     // ~~~~~ Buffering fread ]
05,282     if( isalpha( workbyte ) )
05,283     {
05,284         if( wrdlen < 31 )
05,285         { wrd[ wrdlen ] = tolower( workbyte ); }
05,286         wrdlen++;
05,287     }
05,288     memcpy( &wrd[ wrdlen ], TwoDigitHEXlist[workbyte], 2 );
05,289     wrdlen++;
05,290     wrdlen++;
05,291     wrdlen++;
05,292 } // i 'for'
05,293 // ~~~~~
05,294 */
05,295 // Comment the streamed-read of input file ... ]]]
05,296 size_inLINESIXFOUR=SourceSize;
05,297 printf( "Leprechaun: Size of input file: %s\n", _ui64toaKAZEcomma(size_inLINESIXFOUR, 11TOaDigits, 10) );
05,298 printf( "\n");
05,299
05,300 time1=time(NULL); //fix of bigtime
05,301
05,302 HashChunkSizeInBITS_GLOBAL = HashInBITS_GLOBAL; // 2019-Dec-07
05,303
05,304 #ifdef _NSHA3
05,305 printf( "Leprechaun: Using (first %d bytes of) SHA3-224 for Matches %d+ long, in order to reduce memory footprint.\n", MatchLenAboveWhichHASHkicksin,
05,306 MatchLenAboveWhichHASHkicksin ); // 2020-Nov-12, since 2020-Jun-14 only 256 were SHA3fied, now extended
05,307 #endif
05,308 #ifdef _NPRV
05,309 printf( "Leprechaun: Using (first %d bytes of) PRV-224 for Matches %d+ long, in order to reduce memory footprint.\n", MatchLenAboveWhichHASHkicksin,
05,310 MatchLenAboveWhichHASHkicksin ); // 2020-Nov-12, since 2020-Jun-14 only 256 were SHA3fied, now extended
05,311 #endif
05,312 #ifdef _NDD
05,312 //printf( "Leprechaun: Using (first %d bytes of) DoubleDeuce (a.k.a. CRC64+CRC32C) for Matches %d+ long, in order to reduce memory footprint.\n",
05,312 MatchLenAboveWhichHASHkicksin, MatchLenAboveWhichHASHkicksin ); // 2020-Nov-12, since 2020-Jun-14 only 256 were SHA3fied, now extended

```

```
05,313 //printf( "Leprechaun: Using (first %d bytes of) DoubleDeuce (a.k.a. CRC32C+CRC32K+CRC32K2) for Matches %d+ long, in order to reduce memory footprint.\n",
MatchLenAboveWhichHASHkicksin, MatchLenAboveWhichHASHkicksin ); // 2020-Nov-12, since 2020-Jun-14 only 256 were SHA3fied, now extended
05,314 printf( "Leprechaun: Using (first %d bytes of) DoubleDeuce (a.k.a. 3x2x4B=24B: 2xCastagnoli, 2xKoopman, 2xKoopman2) for Matches %d+ long, in order to reduce memory
footprint.\n", MatchLenAboveWhichHASHkicksin, MatchLenAboveWhichHASHkicksin ); // 2021-Jan-02, since 2020-Jun-14 only 256 were SHA3fied, now extended
05,315 #endif
05,316 #ifdef _NAquaHash
05,317 //printf( "Leprechaun: Using (first %d bytes of) DoubleDeuce (a.k.a. CRC64+CRC32C) for Matches %d+ long, in order to reduce memory footprint.\n",
MatchLenAboveWhichHASHkicksin, MatchLenAboveWhichHASHkicksin ); // 2020-Nov-12, since 2020-Jun-14 only 256 were SHA3fied, now extended
05,318 //printf( "Leprechaun: Using (first %d bytes of) DoubleDeuce (a.k.a. CRC32C+CRC32K+CRC32K2) for Matches %d+ long, in order to reduce memory footprint.\n",
MatchLenAboveWhichHASHkicksin, MatchLenAboveWhichHASHkicksin ); // 2020-Nov-12, since 2020-Jun-14 only 256 were SHA3fied, now extended
05,319 printf( "Leprechaun: Using (first %d bytes of) DoubleDeuceAES_Gumbotron_YMM for Matches %d+ long, in order to reduce memory footprint.\n",
MatchLenAboveWhichHASHkicksinAQUA, MatchLenAboveWhichHASHkicksinAQUA ); // 2021-Jul-31
05,320 #endif
05,321
05,322 for (jj=0; jj< MatchLensNUM; jj++) {
05,323
05,324 BufEnd_64_RESTART = BufEnd_64; // 2019-Dec-07
05,325 pointerflush_64_RESTART = pointerflush_64; // 2019-Dec-07
05,326
05,327 NewAttemptRaisePasses: // 2019-Dec-07
05,328
05,329 BufEnd_64 = BufEnd_64_RESTART; // 2019-Dec-07
05,330 pointerflush_64 = pointerflush_64_RESTART; // 2019-Dec-07
05,331 memset(pointerflush + (uint64_t)(jj)*(uint64_t)((1LL)<<HashInBITS_GLOBAL)*8, 0, (uint64_t)(1)*(uint64_t)((1LL)<<HashInBITS_GLOBAL)*8); // 2019-Dec-04
05,332
05,333 // BUG IN R.17 and prior - the next 'if' is only for RAM (y/Y) not for SSD (z/Z):
05,334 if (BSTorBtree == 2) { //r.18
05,335 BUGGYoffset = (BufEnd_64-(0+14));
05,336 } else { // ##### 64bit memory manipulations [
05,337 BUGGYoffset = (BufEnd_64-(unsigned long long)pointerflush_64);
05,338 } // ##### 64bit memory manipulations ]
05,339
05,340 //for( RipPasses = 1-1; RipPasses <= (1<<<(HashInBITS-HashChunkSizeInBITS))-1; RipPasses++ )
05,341 //{
05,342 RipPasses = 1-1;
05,343 WhyTheHellForIsNotWorking:
05,344
05,345 // 2020-Jun-17 [ Fixing the reporting stats bug by refreshing the master/real BUGGYoffset, since we have two B-trees pools (that use BUGGYoffset) we only report the
one from pass #2 i.e. the final one, not the temporary one which uses '_RAM'!
05,346 if (BSTorBtree == 2) { //r.18
05,347 BUGGYoffset = (BufEnd_64-(0+14));
05,348 } else { // ##### 64bit memory manipulations [
05,349 BUGGYoffset = (BufEnd_64-(unsigned long long)pointerflush_64);
05,350 } // ##### 64bit memory manipulations ]
05,351 // 2020-Jun-17 ] It could be fixed simply by moving that fragment which is now doubled - the one above 'WhyTheHellForIsNotWorking:'
05,352
05,353 if (RipPasses < (1<<<(HashInBITS_GLOBAL-HashChunkSizeInBITS_GLOBAL))-1)
05,354 printf( "Leprechaun: Inserting keys/BBs of order %s into B-trees, free RAM in B-tree pool is %s MB; Pass #s of %s ... \r", _ui64toaKAZEzerocomma(MatchLens[jj],
11ToaDigits2, 10)+(26-3), _ui64toaKAZEzerocomma((size_in64_L14 - BUGGYoffset)>>20, 11ToaDigits, 10)+(26-10), _ui64toaKAZEzerocomma(RipPasses+1, 11ToaDigits3, 10)+(26-7),
_ui64toaKAZEzerocomma((1<<<(HashInBITS_GLOBAL-HashChunkSizeInBITS_GLOBAL)), 11ToaDigits4, 10)+(26-7) );
05,355 else
05,356 printf( "Leprechaun: Inserting keys/BBs of order %s into B-trees, free RAM in B-tree pool is %s MB; Pass #s of %s ... ", _ui64toaKAZEzerocomma(MatchLens[jj],
11ToaDigits2, 10)+(26-3), _ui64toaKAZEzerocomma((size_in64_L14 - BUGGYoffset)>>20, 11ToaDigits, 10)+(26-10), _ui64toaKAZEzerocomma(RipPasses+1, 11ToaDigits3, 10)+(26-7),
_ui64toaKAZEzerocomma((1<<<(HashInBITS_GLOBAL-HashChunkSizeInBITS_GLOBAL)), 11ToaDigits4, 10)+(26-7) );
05,357
05,358 fflush(stdout);
05,359
05,360 // In-here the new lowering MEMORY FOOTPRINT improvement begins:
```

```
05,361 // It is from two passes:
05,362 // - the first builds B-trees by putting all keys (unique and non-unique) into leaves;
05,363 // - the second builds B-trees by putting ONLY non-unique keys into leaves;
05,364 // Each pass has its own two POOLs:
05,365 // Pass #1: HASH POOL size: ((1LL)<<HashInBITS_GLOBAL)*8
05,366 // Pass #1: HASH POOL address: pointerflush //BufStart + Slot *(MatchLensNUM*(1LL<<HashInBITS_GLOBAL)*8) // MatchLensNUM=10
05,367 // Pass #1: Btree POOL size: size_in64_L14_RAM = SourceSize
05,368 // Pass #1: Btree POOL address: pointerflush_64_RAM = VerifyBlock
05,369
05,370 // Pass #2: HASH POOL size: MatchLensNUM*((1LL)<<HashInBITS_GLOBAL)*8
05,371 // Pass #2: HASH POOL address: pointerflush //BufStart + Slot *(jj*(1LL<<HashInBITS_GLOBAL)*8) // jj=0..9
05,372 // Pass #2: Btree POOL size: size_in64_L14 = 1024LL * 1024 * (unsigned long long)Thunderwith + 14
05,373 // Pass #2: Btree POOL address: pointerflush_64
05,374
05,375 // Variable-Differences between two passes:
05,376 // Pass #1: Slot_RAM, BufEnd_64_RAM, size_in64_L14_RAM, pointerflush_64_RAM, BSTorBtree_RAM enforced to be 3 i.e. RAM; Pass #1 has 'counter' uncommented!
05,377 // Pass #2: Slot, BufEnd_64, size_in64_L14, pointerflush_64, BSTorBtree
05,378 // 'FoundInLinkedList_RAM' decides whether to execute B-tree fragment AT ALL in PASS #2! If the key's occurrences are 1+ then FoundInLinkedList_RAM = 1
05,379
05,380 // The ASCII block-scheme of major mumbo-jumbo:
05,381
05,382 // 2019-Dec-28
05,383 /*
05,384 https://software.intel.com/en-us/forums/intel-moderncode-for-parallel-architectures/topic/520602#comment-1950202
05,385 https://community.centminmod.com/threads/a-lzss-microdeduplicator-tagetting-huge-texts-with-c-source.16427/#post-80053
05,386
05,387 Gladden I am to share the most refined revision of my texttoy.
05,388
05,389 Nakamichi already entered Dr. Mahoney's LTCB (Large Text Compression Benchmark) roster as Pareto frontier setter i.e. no other (open-source) decompressor is faster
with this compression ratio.
05,390 http://mattmahoney.net/dc/text.html#2774
05,391
05,392 Big news! Now, the matchfinder is able to find a match with ONE only random (external) access, one random fetch done within a hashtable (based on physical RAM), and
one random fetch done within a B-tree (based on physical/external RAM).
05,393
05,394 The revision achieving above results is from 2019-Aug-06, today gladly I share the latest much superior revision featuring:
05,395 - Needs 3N physical RAM (+ adjustable physical RAM for HASH pools), where N is the size of file being compressed;
05,396 - Automatic setting of PASSES (allows to fit building B-trees into RAM (the third N used to house output file) and just then to rebuild the current PASS either on RAM
or SSD) - it lowers drastically the wearing of external RAM;
05,397 - Significantly Lowered Memory Footprint - now, only the NON-UNIQUE keys are put into B-trees;
05,398 - All B-trees are DEFRAGMENTED i.e. all their leaves are concatenated i.e. each B-tree is housed in one SOLID block - thus PREFETCH-FRIENDLY they became.
05,399
05,400 The beauty of the Nakamichi C source comes from it being a practical etude/benchmark of superfast implementations of:
05,401 - FASTEST known to me memmem(), namely the function char * Railgun_Trolldom(char * pbTarget, char * pbPattern, uint32_t cbTarget, uint32_t cbPattern);
05,402 - FASTEST known to me LOOKUP hasher, namely the uint32_t FNV1A_Pippip_Yurii(const char *str, size_t wrdlen);
05,403 - FASTEST known to me B-tree, namely the void B_tree_Non_Unique_Only_DEFRAGMENTED(int argc, char *argv[], char* SourceBlock, uint64_t SourceSize, char* VerifyBlock),
it is capable transparently to work with internal/external RAM.
05,404
05,405 The "normal" case (which is advocated even in Prof. Knuth's bible) is to use a LEAF with many many keys - to avoid TAPE/HDD revolvments/repositions.
05,406 Footprintwise, I always considered this advice out of fashion/style, even today's NAND and Xpoint SSDs cannot deliver magical speeds unless the B-tree is both compact
(my order 3 is more size effective) and prefetch-friendly.
05,407 My two benchmarking laptops are now busy, but in the next year will share how in practice this revision works with some big corpora as:
05,408 - SILVA DNA corpus ~1.1 GB strong;
05,409 - Human Genome DNA corpus ~3.3 GB strong.
05,410
05,411 On 'SILVA' Nakamichi sets Pareto, but actually Skibinski's LIZARD is better:
05,412 https://community.centminmod.com/threads/a-lzss-microdeduplicator-tagetting-huge-texts-with-c-source.16427/#post-80053
05,413
```

05,414 The needed memory for latter is 20N i.e. 66GB, the FRAGMENTED revision is already on the 4th percent, after finishing will run the DEFRAGMENTED one...

05,415

05,416 ```

05,417 Parsing all the BBs (Building-Blocks) of length 4,6,8,10,12,14,16,18,36,64 at each position:

05,418

05,419 PASS #1 - TEMPORARY STUFF:

05,420 One POOL based/located always on physical (internal RAM) - the last/hidden one of all HASH SUB-POOLS:

05,421

05,422 | Hash BLOCK (TEMPORARY HASH Sub-Pool), size = command line parameter |-----

05,423

05,424

05,425 | All slots point to B-tree roots in the TEMPORARY B-tree POOL

05,426

05,427 | Three POOLs based/located always on physical (internal RAM): \ /

05,428

05,429 | Source Block, size = N | | Source REVERSED Block, size = N | | Target Block, size = N, or, B-trees BLOCK (TEMPORARY) |

05,431

05,432 PASS #2 - TEMPORARY-TO-NONTEMPORARY STUFF (takes into account what above two temporary blocks house):

05,434

05,435 | Source Pool: Houses the file being compressed |

05,436

05,437

05,438

05,439 | Hash 6 bytes

05,440

05,441

05,442 | Hash 4 bytes

05,443

05,444

05,445 | Hash Sub-Pool for BBs 4 bytes long

05,446

05,447 | Hash Sub-Pool for BBs 6 bytes long

05,448

05,449

05,450

05,451

05,452

05,453

05,454

05,455

05,456

05,457

05,458

05,459

05,460

05,461

05,462

05,463

05,464

05,465

05,466 B-tree order 3 (MAX 3 pointers, MAX 2 keys) LEAF #1 (ROOT is also a LEAF - if no successors) structure:

05,467

05,468

05,469

05,470

05,471

05,472

05,473

05,474

05,475

05,476

05,477

05,478

05,479

05,480 Note1: Hash Pool: Houses (10+1) consecutive Sub-Pools, the 11th (the last one actually) is used in PASS #1 only (as temporary, its slots point to physical RAM only addresses located in TargetBlock of size N=SourceSize).

05,481 Note2: Usually 'LeftQWORD' and 'RightQWORD' are used either for OCCURRENCES or LastSeenOffset of that key.

05,482 Note3: If 'LeftKeySize' or 'RightKeySize' is 0 then the field is not used - there is no KEY.

05,483 ```

05,484

05,485 In the near future I intend to reduce more memmem() invocations which are the main culprit for speed drops, it is a matter of few hours to write it down, and ... more

```
SSD memory.
05,486 */
05,487
05,488 // PASS #1: [
05,489
05,490 // INITIALIZE:
05,491 // Below line is zeroing the LAST HASH SUB-POOL for RAM RIP:
05,492 memset(pointerflush + (uint64_t)(MatchLensNUM)*(uint64_t)((1LL)<<HashInBITS_GLOBAL)*8,0, (uint64_t)(1)*(uint64_t)((1LL)<<HashInBITS_GLOBAL)*8); // 2019-Dec-04
05,493 // Below line is resetting Btree POOL for RAM RIP:
05,494 BufEnd_64_RAM = (unsigned long long)pointerflush_64_RAM;
05,495 memset(pointerflush_64_RAM,0, size_in64_L14_RAM); // Probably no need, have to check whether it is necessary... YES, if commented then it crashes!? Maybe all the B-
tree building/init causes the worse ratio compared to older versions?!
05,496
05,497 for (BuildingBlocksSTRIDE=0; (signed long long)BuildingBlocksSTRIDE < (signed long long)size_inLINESIXFOUR-MatchLens[jj]+1; BuildingBlocksSTRIDE++) { // 2020-Jan-01
05,498
05,499 #ifdef Kanshiketsu
05,500 if ( (jj!=0 && jj<=9) && (SourceBlockSKIParray[BuildingBlocksSTRIDE]==0) ) goto SKIPaPositionSinceItIsUnique; // Couldn't be 0 since order 4 is not yet ripped. Enforce
it to be up to 64 inclusive i.e. in range 0..9 due to being sorted until [9] - have to sort the matchlens within the array...
05,501 #endif
05,502
05,503 // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed [
05,504 // wrdlen=0;
05,505 // memset( &wrd[0], 0, (LongestLineInclusive+1+8) ); //r.18, see below, the old nullifier is commented.
05,506 // for (mm=0; mm< MatchLens[jj]; mm++) {
05,507 //     memcpy( &wrd[ wrdlen ], TwoDigitHEXlist[(unsigned char *) (SourceBlock+mm+BuildingBlocksSTRIDE)], 2 );
05,508 //     wrdlen++;
05,509 //     wrdlen++;
05,510 // }
05,511 // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed ]
05,512
05,513 wrdlen=MatchLens[jj];
05,514 memset( &wrd[0], 0, (LongestLineInclusive+1+8) ); //r.18, see below, the old nullifier is commented.
05,515
05,516 #if defined(_NSHA3) || defined(_NPRV) || defined(_NDD) || defined(_NAquaHash) //2020-Nov-20
05,517
05,518 #ifdef _NPRV //2020-Nov-12 [ Consider not only 256[+] matchlens but 24[+] as well - in order to save memory!
05,519     if (wrdlen > MatchLenAboveWhichHASHkicksin) {
05,520         prvhash42( (unsigned char *) (SourceBlock+BuildingBlocksSTRIDE), wrdlen, (unsigned char *) PRVhash, PRVhashlenInBYTES, 0, 0, 0 ); //2020-Nov-17, for
v2.0 ", 0" was added
05,521 #ifdef _NdumphashKey
05,522 //printf("%03d: ", wrdlen);
05,523 fprintf( fp_outOHK, "%03d: ", wrdlen );
05,524 for( idumphashKey = 0; idumphashKey < MatchLenAboveWhichHASHkicksin; idumphashKey++ ) {
05,525     //printf("%02x", *(uint8_t *) (PRVhash+idumphashKey));
05,526     fprintf( fp_outOHK, "%02x", *(uint8_t *) (PRVhash+idumphashKey) );
05,527 }
05,528 //printf(" ");
05,529 fprintf( fp_outOHK, " " );
05,530 for( idumphashKey = 0; idumphashKey < wrdlen; idumphashKey++ ) {
05,531     //printf("%02x", *(uint8_t *) ((unsigned char *) (SourceBlock+BuildingBlocksSTRIDE)+idumphashKey));
05,532     fprintf( fp_outOHK, "%02x", *(uint8_t *) ((unsigned char *) (SourceBlock+BuildingBlocksSTRIDE)+idumphashKey) );
05,533 }
05,534 //printf("\n");
05,535 fprintf( fp_outOHK, "\n" );
05,536 #endif
05,537 wrdlen=MatchLenAboveWhichHASHkicksin; // very aggressive is 16B or 128b, if collisions happen then increase to 20B or 160b
05,538 memcpy( &wrd[ 0+1 ], (unsigned char *) PRVhash, wrdlen );
05,539 } else
```

```
05,540         memcpy( &wrd[ 0+1 ], (unsigned char *) (SourceBlock+BuildingBlocksSTRIDE), wrdlen );
05,541 #endif
05,542 #ifdef _NSHA3
05,543     //if (wrdlen >= 256) { // 2020-Jun-13
05,544         if (wrdlen > MatchLenAboveWhichHASHkicksin) {
05,545             FIPS202_SHA3_224((unsigned char *) (SourceBlock+BuildingBlocksSTRIDE), wrdlen, (unsigned char *) SHA328bytes);
05,546             wrdlen=MatchLenAboveWhichHASHkicksin; //2020-Nov-12
05,547             memcpy( &wrd[ 0+1 ], (unsigned char *) SHA328bytes, wrdlen );
05,548         } else
05,549             memcpy( &wrd[ 0+1 ], (unsigned char *) (SourceBlock+BuildingBlocksSTRIDE), wrdlen );
05,550 #endif
05,551 #ifdef _NDD
05,552     //if (wrdlen >= 256) { // 2020-Jun-13
05,553         if (wrdlen > MatchLenAboveWhichHASHkicksin) {
05,554             //*(uint64_t*)&DD[0] = crc64((unsigned char *) (SourceBlock+BuildingBlocksSTRIDE), wrdlen);
05,555             //*(uint32_t*)&DD[8] = crc32c_sw((unsigned char *) (SourceBlock+BuildingBlocksSTRIDE), wrdlen);
05,556             // Above 2 lines are glued into next one:
05,557             DoubleDeuce( (unsigned char *) (SourceBlock+BuildingBlocksSTRIDE), wrdlen );
05,558 #ifdef _NdumpHashKey
05,559             //printf("%03d: ", wrdlen);
05,560             fprintf( fp_outOHK, "%03d: ", wrdlen );
05,561             for( idumpHashKey = 0; idumpHashKey < MatchLenAboveWhichHASHkicksin; idumpHashKey++ ) {
05,562                 if (idumpHashKey%4==0 && idumpHashKey!=0) fprintf( fp_outOHK, "+" );
05,563                 //printf("%02x", *(uint8_t *) (DD+idumpHashKey));
05,564                 fprintf( fp_outOHK, "%02x", *(uint8_t *) (DD+idumpHashKey) );
05,565             }
05,566             //printf(" ");
05,567             fprintf( fp_outOHK, " " );
05,568             for( idumpHashKey = 0; idumpHashKey < wrdlen; idumpHashKey++ ) {
05,569                 //printf("%02x", *(uint8_t *) ((unsigned char *) (SourceBlock+BuildingBlocksSTRIDE)+idumpHashKey));
05,570                 fprintf( fp_outOHK, "%02x", *(uint8_t *) ((unsigned char *) (SourceBlock+BuildingBlocksSTRIDE)+idumpHashKey) );
05,571             }
05,572             //printf("\n");
05,573             fprintf( fp_outOHK, "\n" );
05,574 #endif
05,575             wrdlen=MatchLenAboveWhichHASHkicksin; //2020-Nov-12
05,576             memcpy( &wrd[ 0+1 ], (unsigned char *) DD, wrdlen );
05,577         } else
05,578             memcpy( &wrd[ 0+1 ], (unsigned char *) (SourceBlock+BuildingBlocksSTRIDE), wrdlen );
05,579 #endif
05,580
05,581 #ifdef _NAquaHash
05,582     //if (wrdlen >= 256) { // 2020-Jun-13
05,583         if (wrdlen > MatchLenAboveWhichHASHkicksinAQUA) {
05,584             DoubleDeuceAES_Gumbotron_YMM( (unsigned char *) (SourceBlock+BuildingBlocksSTRIDE), wrdlen );
05,585             wrdlen=MatchLenAboveWhichHASHkicksinAQUA; //2020-Nov-12
05,586             memcpy( &wrd[ 0+1 ], (unsigned char *) DD_AES, wrdlen );
05,587         } else
05,588             memcpy( &wrd[ 0+1 ], (unsigned char *) (SourceBlock+BuildingBlocksSTRIDE), wrdlen );
05,589 #endif
05,590
05,591 #else
05,592         memcpy( &wrd[ 0+1 ], (unsigned char *) (SourceBlock+BuildingBlocksSTRIDE), wrdlen );
05,593 #endif
05,594
05,595     memcpy( &wrd[ 0 ], &wrdlen, 1 ); // The keysize is 1..255, pretty enough since my needs are 1..28 (up to SHA3-224).
05,596 /*
05,597 #ifdef _NSHA3
```

```

05,598 //          if (wrdlen < 28)
05,599         if (wrdlen != 28) // 2020-Jun-13
05,600             memcpy( &wrd[ 0+1 ], (unsigned char *) (SourceBlock+BuildingBlocksSTRIDE), wrdlen );
05,601         else
05,602             memcpy( &wrd[ 0+1 ], (unsigned char *) SHA328bytes, wrdlen );
05,603 #else
05,604 //          if (wrdlen < 28)
05,605             memcpy( &wrd[ 0+1 ], (unsigned char *) (SourceBlock+BuildingBlocksSTRIDE), wrdlen );
05,606 //          else
05,607 //              memcpy( &wrd[ 0+1 ], (unsigned char *) SHA328bytes, wrdlen );
05,608 #endif
05,609 */
05,610
05,611 BufStart = pointerflush;
05,612 //          Slot = ((wrd[0]-'_')*28*28*28*28 + (wrd[1]-'_')*28*28*28 + (wrd[2]-'_')*28*28 + (wrd[3]-'_')*28 + (wrd[4]-'_'))<<3;
05,613 //          Slot = FNV1A_Hash_Jesteress_27bit(wrd, wrdlen)<<3; // Commented since r.14+++ because of passes.
05,614 //Slot = FNV1A_Hash_Jesteress_27bit(wrd +1, wrdlen); WORDcount++;
05,615 Slot = FNV1A_Pippip_Yurii(wrd +1, wrdlen); WORDcount++; // Changed 2019-Nov-28
05,616
05,617 // Bug fix for all r.14+++ and below! [
05,618 //memcpy( &wrd[(LongestLineInclusive+1+4)-4], &NULLsForWRD, 4 );
05,619 //memcpy( &wrd[(LongestLineInclusive+1+4)-4-1], &NULLsForWRD, 1 );
05,620 // Bug fix for all r.14+++ and below! ]
05,621
05,622 // Example: HashInBITS-HashChunkSizeInBITS=2
05,623 //          HashInBITS = 5
05,624 //          HashChunkSizeInBITS = 3
05,625 //          RipPasses = 1<<((HashInBITS-HashChunkSizeInBITS) i.e. 1<<2 which is 4 i.e. 32 slots with 4 passes 8 slots each.
05,626 //          00??? 5bits 0-7
05,627 //          01??? 5bits 8-15
05,628 //          10??? 5bits 16-23
05,629 //          11??? 5bits 24-31
05,630 if ( (Slot>>HashChunkSizeInBITS_GLOBAL) == RipPasses ) {
05,631     Slot = Slot<<3;
05,632 // NEW NEW NEW [ //r.18
05,633 // In here Slot is not within a single pool but in MatchLensNUM sub-pools i.e. MatchLensNUM-way i.e. MatchLensNUM hashpots:
05,634 /*
05,635 for (GettingIndexOfArray=0; GettingIndexOfArray<MatchLensNUM; GettingIndexOfArray++) {
05,636     // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed [
05,637     //          if ( (wrdlen>>1)==MatchLens[GettingIndexOfArray] ) break;
05,638     if ( (wrdlen)==MatchLens[GettingIndexOfArray] ) break;
05,639     // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed ]
05,640 }
05,641 */
05,642 GettingIndexOfArray=jj; // same as above atrocity
05,643 Slot = Slot +(MatchLensNUM*(1LL<<HashInBITS_GLOBAL)*8); // CAUTION: 'wrdlen' is halved here, when a new revision comes with 1:1 keysize then change it.
05,644 // The line above uses the upper/last part of HASH POOL i.e. for MatchLensNUM=10 11th i.e. 0..9 for pass #2, 10 for pass #1
05,645 // NEW NEW NEW ] //r.18
05,646
05,647 //KeySize = LongestLineInclusive; // 'LongestLineInclusive' is for 1-way hashing where different KeySizes can occupy same HASH pool.
05,648 KeySize = wrdlen; //r.18, In case multi-way hashing: either the MAX allowed or the 'wrdlen'.
05,649
05,650 //Slot = 0; // One Tree only!
05,651 memcpy( &PseudoLinkedPointer_64, BufStart+Slot, 8 );
05,652
05,653 // ##### B-tree order 3 fragment 64bit [
05,654 //
05,655 // LEAF structure: [LeftPointer][MiddlePointer][RightPointer][LeftWord][RightWord]

```

```

05,656 //          4bytes      4bytes      4bytes      wrdlen      wrdlen
05,657 //          *          *          *          *          *          <- if *(char *)==0 means the word cell is empty
05,658 // ALL B-tree order 3 fragment consists of 3 sub-fragments:
05,659 // 1] Search 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search 3] Insert Iterative
05,660
05,661 // LEAF_64 structure: [LeftPointer][MiddlePointer][RightPointer][LeftWord] [RightWord]
05,662 //          8bytes      8bytes      8bytes      LongestLineInclusive+1+4 LongestLineInclusive+1+4
05,663 //          *          *          *          *          *          <- if *(char *)==0 means the word cell is empty
05,664 // Note: In order to use one fread(and strcmp) a NULL postfix for LeftWord, RightWord i.e. LeftWord_Length=len(LeftWord)+1 a kinda stupid choice ...
05,665 // Note: BufEnd_64_RAM in fact is the first free position after the BUFFER END!
05,666
05,667 // 1] Search [
05,668 //          if (PseudoLinkedPointer_64 == 0) // means EMPTY-SLOT
05,669 //          {
05,670 //if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrdlen + 4 + 4 + 4 < GRMBL FoolAgain((int)wrdlen) ) // +4 more for BST instead of LL; + more(see
LEAF)
05,671 //          {
05,672 //          memcpy( BufStart+Slot, &bufend[LetterOffset], 4 );
05,673 //          bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
05,674 //          memcpy( bufend[LetterOffset], wrd, wrdlen ); WORDcountDistinct++; bufNumberOfWords[LetterOffset]++;
05,675 //          bufend[LetterOffset] = bufend[LetterOffset] + 2*wrdlen;
05,676 //          if (MAXusedBuffer[wrdlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrdlen] = (unsigned long)(bufend[LetterOffset] -
BufStart);}
05,677 //          }
05,678 //          // BUG IN R.17 and prior - the next 'if' is only for RAM (y/Y) not for SSD (z/Z):
05,679 //          if (BSTorBtree_RAM == 2) { //r.18
05,680 BUGGYoffset = (BufEnd_64_RAM-(0+14));
05,681 //          } else { // ##### 64bit memory manipulations [
05,682 BUGGYoffset = (BufEnd_64_RAM-(unsigned long long)pointerflush_64_RAM);
05,683 //          } // ##### 64bit memory manipulations ]
05,684 //          if( 8 + 8 + 8 + 2*(KeySize+1+8) < size_in64_L14_RAM - BUGGYoffset ) // the longest wrdlen is LongestLineInclusive but actual is LongestLineInclusive(+ CR
char)
05,685 //          {
05,686 //          memcpy( BufStart+Slot, &BufEnd_64_RAM, 8 );
05,687 BufEnd_64_RAM = BufEnd_64_RAM + 8 + 8 + 8;
05,688 //          if (BSTorBtree_RAM == 2) {
05,689 //          fsetpos(fp_outRG, (const fpos_t *)&BufEnd_64_RAM);
05,690 //          fwrite(wrd, wrdlen, 1, fp_outRG); //GRMBL! r.18
05,691 //          fwrite(wrd, (KeySize+1+8), 1, fp_outRG); //GRMBL! r.18
05,692 WORDcountDistinct++; Total_fwrite++;
05,693 //          fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
05,694 //          // r.14++ The above line was commented because the pool is already ZEROed.
05,695 //          } else { // ##### 64bit memory manipulations [
05,696 //          memcpy( (char *)BufEnd_64_RAM, wrd, wrdlen ); //GRMBL! r.18
05,697 //          memcpy( (char *)BufEnd_64_RAM, wrd, (KeySize+1+8) ); //GRMBL! r.18
05,698 WORDcountDistinct++;
05,699 //          } // ##### 64bit memory manipulations ]
05,700 BufEnd_64_RAM = BufEnd_64_RAM + 2*(KeySize+1+8);
05,701 //          fsetpos(fp_outRG, &BufEnd_64_RAM);
05,702 //          }
05,703 //          else
05,704 //          { printf( "\nLeprechaun: Failure! Increment 'Memory for B-trees'!\n" );
05,705
05,706 #ifdef LITE
05,707 #else
05,708 fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, llTOaDigits, 10), _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct,
llTOaDigits2, 10) );
05,709 fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14_RAM, llTOaDigits, 10) );

```



```

05,710 fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11ToaDigits, 10) );
05,711 fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
05,712         fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for B-trees'!\n\n");
05,713 #endif
05,714
05,715 if ( HashChunkSizeInBITS_GLOBAL > 0 ) { // 2019-Dec-07
05,716     HashChunkSizeInBITS_GLOBAL--;
05,717     printf( "Automatically increasing number of passes in order to fit B-trees into Target Buffer.\n" );
05,718     goto NewAttemptRaisePasses;
05,719 } else exit( 7 );
05,720     }
05,721         FoundInLinkedList = 1+1;
05,722     }
05,723     else // means USED-SLOT
05,724     { FoundInLinkedList = 0;
05,725     StackPtr = 0;
05,726     //         while (PseudoLinkedPointer != 0 && FoundInLinkedList == 0)
05,727         while (PseudoLinkedPointer_64 != 0 && FoundInLinkedList == 0)
05,728         {
05,729     // ***** 'P W P' section [
05,730     // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
05,731     // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
05,732     // here ALWAYS LW exists: no need for existence check - line below
05,733     // if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
05,734     //         if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wordlen) > 0) // go LP
05,735     // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
05,736     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
05,737     //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
05,738     //fread(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
05,739     // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
05,740     // [ //r.14+
05,741     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
05,742     if (BSTorBtree_RAM == 2) {
05,743     fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
05,744     fread(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fread++;
05,745     } else { // ##### 64bit memory manipulations [
05,746     memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64, 8+8+2*(KeySize+1+8) );
05,747     } // ##### 64bit memory manipulations ]
05,748     memcpy( &FourGramL[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
05,749     // ] //r.14+
05,750     //strFLAG=strcmpKAZE13(FourGramL, wrd);
05,751     strFLAG=memcmp(FourGramL+1, wrd+1, wordlen); // Since 2019-Feb-07, it is not ASCIIZ string but zero byte is the length of the 1..
05,752     if (strFLAG > 0) // go LP
05,753     { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 0, 4 ); //LP
05,754     //         PseudoLinkedPointer = PseudoLinkedPointerNEW;
05,755     //     }
05,756     {
05,757     // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
05,758     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 0; //LP
05,759     // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
05,760     if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
05,761     BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
05,762     BSTstack[StackPtr] = 0; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
05,763     // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
05,764     //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
05,765     //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
05,766     // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
05,767     // [ //r.14+

```

```

05,768         memcpy( &PseudoLinkedPointer_64, &LEAF[0], 8 );
05,769         // ] //r.14+
05,770         }
05,771 //         else if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wrdlen) < 0) // go RP or MP
05,772         else if (strFLAG < 0) // go RP or MP
05,773         { // RW existence check - line below:
05,774 //         if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 ) // RW exists
05,775         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
05,776         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
05,777         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
05,778         //fread(&SomeByte, 1, 1, fp_outRG);
05,779         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
05,780         // [ //r.14+
05,781         memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (KeySize+1+8)], 1 );
05,782         // ] //r.14+
05,783         if (SomeByte != 0) // RW exists
05,784         { // Here all 'P W P' section is repeated; the way of handling case when dynamic number of words in leaf
05,785 // *****
05,786 // ***** 'P W P' section 2 [
05,787 // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
05,788 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
05,789         // here ALWAYS RW exists: no need for existence check - line below
05,790         // if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
05,791 //         if (memcmp(PseudoLinkedPointer+4+4+4+wrdlen, wrd, wrdlen) > 0) // go MP
05,792         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
05,793         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
05,794         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
05,795         //fread(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
05,796         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
05,797         // [ //r.14+
05,798         memcpy( &FourGramL[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
05,799         // ] //r.14+
05,800         //strFLAG=strcmpKAZE13(FourGramL, wrd);
05,801         strFLAG=memcmp(FourGramL+1, wrd+1, wrdlen); // Since 2019-Feb-07, it is not ASCIIZ string but zero byte is the length of the 1..
05,802         if (strFLAG > 0) // go MP
05,803         { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
05,804 //         PseudoLinkedPointer = PseudoLinkedPointerNEW;
05,805 //         }
05,806 //         {
05,807         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
05,808         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8; //MP
05,809         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
05,810         if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
05,811         BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
05,812         BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
05,813         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
05,814         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
05,815         //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
05,816         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
05,817         // [ //r.14+
05,818         memcpy( &PseudoLinkedPointer_64, &LEAF[8], 8 );
05,819         // ] //r.14+
05,820         }
05,821 //         else if (memcmp(PseudoLinkedPointer+4+4+4+wrdlen, wrd, wrdlen) < 0) // go RP
05,822         else if (strFLAG < 0) // go RP
05,823         { // No ?W after RW - go RP
05,824 //         memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4 + 4, 4 ); //RP
05,825 //         PseudoLinkedPointer = PseudoLinkedPointerNEW;

```

```

05,826 //      }
05,827 //      {
05,828 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
05,829 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8; //BP
05,830 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
05,831 if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
05,832 BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
05,833 BSTstack[StackPtr] = 16; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
05,834 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
05,835 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
05,836 //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
05,837 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
05,838 // [ //r.14+
05,839 memcpy( &PseudoLinkedPointer_64, &LEAF[8 + 8], 8 );
05,840 // ] //r.14+
05,841 //      }
05,842 //      else { FoundInLinkedList = 1; // wrd is RW
05,843 // Counter [
05,844 //      if (BSTorBtree_RAM == 2) {
05,845 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
05,846 //      }
05,847 //      memcpy( &CounterOccurrences, &FourGramL[(KeySize+1+8)-8], 8 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
05,848 //      if (CounterOccurrences==0) CounterOccurrences++; // Fixing an old bug, counting starts from 0 since this is 'search' not 'insert' - it
was already inserted i.e. with 1 occurrence but the field is ZERO when put initially. Fix: 2019-Dec-19
05,849 //      if (CounterOccurrences<999999999) CounterOccurrences++; // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
05,850 //      memcpy( &FourGramL[(KeySize+1+8)-8], &CounterOccurrences, 8 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
05,851 //      }
05,852 // Debug the ratio difference [
05,853 //      if (CounterOccurrences>1) {
05,854 //          printf("\nPASS #1: BuildingBlocksSTRIDE, jj = %s, %s\n", _ui64toaKAZEzerocomma4(BuildingBlocksSTRIDE, llTOaDigits3, 16)+(26-8-1),
05,855 //          _ui64toaKAZEzerocomma4(jj, llTOaDigits2, 10)+(26-2));
05,856 //      }
05,857 // Debug the ratio difference ]
05,858 //      }
05,859 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
05,860 //fwrite(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
05,861 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
05,862 // [ //r.14+
05,863 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &FourGramL[0], (KeySize+1+8) );
05,864 //      if (BSTorBtree_RAM == 2) {
05,865 //fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
05,866 //      } else { // ##### 64bit memory manipulations [
05,867 //      memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
05,868 //      } // ##### 64bit memory manipulations ]
05,869 // ] //r.14+
05,870 // Counter ]
05,871 //      }
05,872 // WORDcountAttemptsToPut++;
05,873 // ***** 'P W P' section 2 ]
05,874 // *****
05,875 //      }
05,876 //      else // RW empty - go MP
05,877 //      { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
05,878 //      PseudoLinkedPointer = PseudoLinkedPointerNEW;
05,879 //      }
05,880 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
05,881 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8; //MP

```

```

05,882 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
05,883 if (StackPtr > 8192*3-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
05,884 BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
05,885 BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
05,886 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
05,887 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
05,888 //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
05,889 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
05,890 // [ //r.14+
05,891 memcpy( &PseudoLinkedPointer_64, &LEAF[8], 8 );
05,892 // ] //r.14+
05,893 }
05,894 }
05,895 else { FoundInLinkedList = 1; // wrd is LW
05,896 // Counter [
05,897 if (BSTorBtree_RAM == 2) {
05,898 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
05,899 }
05,900 memcpy( &CounterOccurrences, &FourGramL[(KeySize+1*8)-8], 8 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
05,901 if (CounterOccurrences==0) CounterOccurrences++; // Fixing an old bug, counting starts from 0 since this is 'search' not 'insert' - it
was already inserted i.e. with 1 occurrence but the field is ZERO when put initially. Fix: 2019-Dec-19
05,902 if (CounterOccurrences<999999999) CounterOccurrences++; // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
05,903 memcpy( &FourGramL[(KeySize+1*8)-8], &CounterOccurrences, 8 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
05,904
05,905 // Debug the ratio difference [
05,906 // if (CounterOccurrences>1) {
05,907 // printf("\nPASS #1: BuildingBlocksSTRIDE, jj = %s, %s\n", _ui64toaKAZEzerocomma4(BuildingBlocksSTRIDE, 11TOaDigits3, 16)+(26-8-1),
05,908 // _ui64toaKAZEzerocomma4(jj, 11TOaDigits2, 10)+(26-2));
05,909 // }
05,909 // Debug the ratio difference ]
05,910
05,911 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
05,912 //fwrite(&FourGramL[0], (LongestLineInclusive+1*4), 1, fp_outRG);
05,913 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
05,914 // [ //r.14+
05,915 memcpy( &LEAF[8 + 8 + 8], &FourGramL[0], (KeySize+1*8) );
05,916 if (BSTorBtree_RAM == 2) {
05,917 fwrite(&LEAF[0], 8*8+2*(KeySize+1*8), 1, fp_outRG); Total_fwrite++;
05,918 } else { // ##### 64bit memory manipulations [
05,919 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8*8+2*(KeySize+1*8) );
05,920 } // ##### 64bit memory manipulations ]
05,921 // ] //r.14+
05,922 // Counter ]
05,923 }
05,924 WORDcountAttemptsToPut++;
05,925 // ***** 'P W P' section ]
05,926 } // while
05,927 WORDcountAttemptsToPut--; // - 1 due to BST way of counting i.e. direct hash hit is not counted only successors
05,928 }
05,929 // 1] Search ]
05,930 if (FoundInLinkedList == 0) NotFoundKeys++; //r.18
05,931 if (FoundInLinkedList == 1) FoundKeys++; //r.18
05,932 if (FoundInLinkedList == 2) NotFoundKeys++; //r.18
05,933
05,934 if (FoundInLinkedList == 0)
05,935 {
05,936 /*
05,937 // ===== [ The whole section/sub-fragment 2 is commented due to great time differences for Internal_vs_External memory accesses - it is far more

```

```

cheap to have the STACK overhead (moved to sub-fragment 1) ] ===== [
05,938 // 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search [
05,939 // 'TracingSearch' is the same as 'Search' except that adds the trail in my simulated stack,
05,940 // the goal is not to waste time in 'Search' by dealing with no needed trail in case of not 'Insert'.
05,941 // Simulated stack contains pairs of 'Address of ParentLEAF' + 'Offset of ParentPointer in ParentLEAF i.e. 0 for LP, 4 for MP, 8 for RP'.
05,942 // 'Offset ...' saves unnecessary comparisons of NEWword which after splitting goes up.
05,943     memcpy( &PseudoLinkedPointer, BufStart+Slot, 4 );
05,944     StackPtr = 0;
05,945     while (PseudoLinkedPointer != 0)
05,946     {
05,947 // ***** 'P W P' section [
05,948 // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
05,949 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
05,950 // here ALWAYS LW exists: no need for existence check - line below
05,951 // if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
05,952 // if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wordlen) > 0) // go LP
05,953 // { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 0, 4 ); //LP
05,954 // if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
05,955 // BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
05,956 // BSTstack[StackPtr] = 0; ++StackPtr; //LPoffset=0;MPoffset=4;RPOffset=8;
05,957 //     PseudoLinkedPointer = PseudoLinkedPointerNEW;
05,958 // }
05,959 // else if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wordlen) < 0) // go RP or MP
05,960 // { // RW existence check - line below:
05,961 //     if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 ) // RW exists
05,962 //     { // Here all 'P W P' section is repeated; the way of handling case when dynamic number of words in leaf
05,963 // *****
05,964 // ***** 'P W P' section 2 [
05,965 // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
05,966 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
05,967 // here ALWAYS RW exists: no need for existence check - line below
05,968 // if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
05,969 // if (memcmp(PseudoLinkedPointer+4+4+4+wordlen, wrd, wordlen) > 0) // go MP
05,970 // { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
05,971 // if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
05,972 // BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
05,973 // BSTstack[StackPtr] = 4; ++StackPtr; //LPoffset=0;MPoffset=4;RPOffset=8;
05,974 //     PseudoLinkedPointer = PseudoLinkedPointerNEW;
05,975 // }
05,976 // else if (memcmp(PseudoLinkedPointer+4+4+4+wordlen, wrd, wordlen) < 0) // go RP
05,977 // { // No ?W after RW - go RP
05,978 //     memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4 + 4, 4 ); //RP
05,979 // if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
05,980 // BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
05,981 // BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=4;RPOffset=8;
05,982 //     PseudoLinkedPointer = PseudoLinkedPointerNEW;
05,983 // }
05,984 // else FoundInLinkedList = 1; // wrd is RW
05,985 // ***** 'P W P' section 2 ]
05,986 // *****
05,987 // }
05,988 // else // RW empty - go MP
05,989 // { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
05,990 // if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
05,991 // BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
05,992 // BSTstack[StackPtr] = 4; ++StackPtr; //LPoffset=0;MPoffset=4;RPOffset=8;
05,993 //     PseudoLinkedPointer = PseudoLinkedPointerNEW;
05,994 // }

```

```

05,995         }
05,996         else FoundInLinkedList = 1; // wrd is LW
05,997 // ***** 'P W P' section ]
05,998         } // while
05,999 // 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search ]
06,000 // ===== [ The whole section/sub-fragment 2 is commented due to great time differences for Internal_vs_External memory accesses - it is far more
cheap to have the STACK overhead (moved to sub-fragment 1) ] =====
06,001 */
06,002
06,003 // 3] Insert Iterative [
06,004 //   There are total 4 situations:
06,005 //   Case #1: Outer NODE(including ROOT) [ ][ ][ ][LW][ ]
06,006 //   Case #2: Outer NODE(including ROOT) [ ][ ][ ][LW][RW] Split Occurs -----
06,007 //   Case #3: ROOT [LP][MP][ ][LW][ ] | 'wrdUP' (wrklen bytes)
06,008 //   Case #4: Inner NODE(including ROOT) [LP][MP][RP][LW][RW] Split Occurs --- | &
06,009 //   | | 'PseudoLinkedPointerNEW' (ptr to NEW LEAF)
06,010 //   There are total 2 situations for PARENT LEAF: <----- ARE GOING UP
06,011 //   Case #3: [LP][MP][ ][LW][ ]
06,012 //   Case #4: [LP][MP][RP][LW][RW] Split Occurs
06,013
06,014 // ~ First deal alonely with the OUTER NODE(LEAF) where Search stopped i.e Case #1 & Case #2:
06,015     POffsetInLEAF = BSTstack[--StackPtr];
06,016     PseudoLinkedPointer_64 = BSTstack[--StackPtr];
06,017 // NOTE: ONE LEAF IS FULL ONLY WHEN LAST CELL FOR KEY(here RW) EXISTS!
06,018 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4*4+4*wrklen) != 0 )
06,019 //if ( *(char *) (PseudoLinkedPointer+4*4+4*wrklen) != 0 ) // If LEAF is full: Case #2
06,020     // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,021     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
06,022     //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,023     //fread(&SomeByte, 1, 1, fp_outRG);
06,024     // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,025     // [ //r.14+
06,026     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
06,027     if (BSTorBtree_RAM == 2) {
06,028         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,029         fread(&LEAF[0], 8*8+2*(KeySize+1+8), 1, fp_outRG); Total_fread++;
06,030     } else { // ##### 64bit memory manipulations [
06,031         memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64, 8*8+2*(KeySize+1+8) );
06,032     } // ##### 64bit memory manipulations ]
06,033     memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (KeySize+1+8)], 1 );
06,034     // ] //r.14+
06,035     if (SomeByte != 0 ) // RW exists
06,036     { SplitOccured = 1; WORDcountDistinct++;
06,037         // ALlocate NEW LEAF:
06,038     // if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrklen + 4 + 4 + 4 < GRMBLFoolAgain[(int)wrklen] ) // +4 more for BST instead of LL; + more(see
LEAF)
06,039     // {
06,040     //     memcpy( &PseudoLinkedPointerNEW, &bufend[LetterOffset], 4 );
06,041     //     bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
06,042     //     bufend[LetterOffset] = bufend[LetterOffset] + 2*wrklen;
06,043     //     if (MAXusedBuffer[wrklen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrklen] = (unsigned long)(bufend[LetterOffset] -
BufStart);}
06,044     // }
06,045     // BUG IN R.17 and prior - the next 'if' is only for RAM (y/Y) not for SSD (z/Z):
06,046     if (BSTorBtree_RAM == 2) { //r.18
06,047         BUGGYoffset = (BufEnd_64_RAM-(0+14));
06,048     } else { // ##### 64bit memory manipulations [
06,049         BUGGYoffset = (BufEnd_64_RAM-(unsigned long long)pointerflush_64_RAM);

```

```

06,050          } // ##### 64bit memory manipulations ]
06,051      if( 8 + 8 + 8 + 2*(KeySize+1+8) < size_in64_L14_RAM - BUGGYoffset ) // the longest wrdlen is LongestLineInclusive but actual is LongestLineInclusive(+ CR
char)
06,052          {
06,053              PseudoLinkedPointerNEW_64 = BufEnd_64_RAM;
06,054              BufEnd_64_RAM = BufEnd_64_RAM + 8 + 8 + 8;
06,055              BufEnd_64_RAM = BufEnd_64_RAM + 2*(KeySize+1+8);
06,056          }
06,057      else
06,058      { printf( "\nLeprechaun: Failure! Increment 'Memory for B-trees'!\n" );
06,059      #ifdef LITE
06,060      #else
06,061          fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, 11TOaDigits, 10), _ui64toaKAZEcomma((unsigned long
long)WORDcountDistinct, 11TOaDigits2, 10) );
06,062          fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14_RAM, 11TOaDigits, 10) );
06,063          fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11TOaDigits, 10) );
06,064          fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
06,065          fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for B-trees'!\n\n");
06,066      #endif
06,067
06,068      if ( HashChunkSizeInBITS_GLOBAL > 0 ) { // 2019-Dec-07
06,069          HashChunkSizeInBITS_GLOBAL--;
06,070          printf( "Automatically increasing number of passes in order to fit B-trees into Target Buffer.\n" );
06,071          fflush(stdout);
06,072          goto NewAttemptRaisePasses;
06,073      } else exit( 7 );
06,074      }
06,075      if (POffsetInLEAF == 0) // wrd < LW
06,076      {
06,077          //          memcpy( wrdUP, PseudoLinkedPointer+4+4+4, wrdlen ); // LW up
06,078          //          memcpy( PseudoLinkedPointer+4+4+4, wrd, wrdlen ); // wrd go to OLD LEAF
06,079          //          memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
06,080          //          *(char *)(&PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
06,081          // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,082          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
06,083          //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,084          //fread(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
06,085          //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,086          //fwrite(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
06,087          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
06,088          //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,089          //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
06,090          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
06,091          //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,092          //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
06,093          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
06,094          //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,095          //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
06,096          // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,097          // [ //r.14+
06,098          memcpy( &wrdUP[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
06,099          memcpy( &LEAF[8 + 8 + 8], &wrd[0], (KeySize+1+8) );
06,100          memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
06,101          // Here reordering (of writing wrdAUX) is needed to avoid seek the position NEW and stupidly to seek again OLD/current position!
06,102          memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
06,103          if (BSTorBtree_RAM == 2) {
06,104              fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,105              fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;

```

```

06,106         } else { // ##### 64bit memory manipulations [
06,107     memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8*8+2*(KeySize+1+8) );
06,108         } // ##### 64bit memory manipulations ]
06,109     PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
06,110     if (BSTorBtree_RAM == 2) {
06,111         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,112         fwrite(&wrdAUX[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
06,113     } else { // ##### 64bit memory manipulations [
06,114     memcpy( (char *)PseudoLinkedPointerAUX_64, &wrdAUX[0], (KeySize+1+8) );
06,115         } // ##### 64bit memory manipulations ]
06,116     // ] //r.14+
06,117 }
06,118 if (PoffsetInLEAF == 8) // LW < wrd < RW
06,119 {
06,120     // memcpy( wrdUP, wrd, wrdlen ); // wrd up
06,121     // memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
06,122     // *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
06,123     // memcpy( wrdUP, wrd, (KeySize+1+8) ); // wrd up
06,124     // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,125     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
06,126     //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,127     //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
06,128     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
06,129     //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,130     //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
06,131     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
06,132     //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,133     //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
06,134     // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,135     // [ //r.14+
06,136     memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
06,137     // Here reordering (of writing wrdAUX) is needed to avoid seek the position NEW and stupidly to seek again OLD/current position!
06,138     memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
06,139     if (BSTorBtree_RAM == 2) {
06,140         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,141         fwrite(&LEAF[0], 8*8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
06,142     } else { // ##### 64bit memory manipulations [
06,143     memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8*8+2*(KeySize+1+8) );
06,144         } // ##### 64bit memory manipulations ]
06,145     PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
06,146     if (BSTorBtree_RAM == 2) {
06,147         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,148         fwrite(&wrdAUX[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
06,149     } else { // ##### 64bit memory manipulations [
06,150     memcpy( (char *)PseudoLinkedPointerAUX_64, &wrdAUX[0], (KeySize+1+8) );
06,151         } // ##### 64bit memory manipulations ]
06,152     // ] //r.14+
06,153 }
06,154 if (PoffsetInLEAF == 16) // wrd > RW
06,155 {
06,156     // memcpy( wrdUP, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW up
06,157     // *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
06,158     // memcpy( PseudoLinkedPointerNEW+4+4+4, wrd, wrdlen ); // wrd go to NEW LEAF
06,159     // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,160     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
06,161     //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,162     //fread(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
06,163     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);

```



```

06,164         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,165         //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
06,166         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
06,167         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,168         //fwrite(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
06,169         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,170         // [ //r.14+
06,171         memcpy( &wrdUP[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
06,172         memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
06,173         if (BSTorBtree_RAM == 2) {
06,174             fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,175             fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
06,176             } else { // ##### 64bit memory manipulations [
06,177             memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
06,178             } // ##### 64bit memory manipulations ]
06,179         // ] //r.14+
06,180         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one! Here NO need!
06,181         PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
06,182         if (BSTorBtree_RAM == 2) {
06,183             fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,184             fwrite(&wrd[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
06,185             } else { // ##### 64bit memory manipulations [
06,186             memcpy( (char *)PseudoLinkedPointerAUX_64, &wrd[0], (KeySize+1+8) );
06,187             } // ##### 64bit memory manipulations ]
06,188         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one! Here NO need!
06,189     }
06,190 }
06,191 else // If LEAF is not full: Case #1
06,192 { SplitOccured = 0; WORDcountDistinct++;
06,193   if (POffsetInLEAF == 0) // wrd < [LW][ ] so [LW][ ] -> [ ][LW] -> [wrd][LW]
06,194   {
06,195       //         memcpy( PseudoLinkedPointer+4+4+4+wrdlen, PseudoLinkedPointer+4+4+4, wrdlen );
06,196       //         memcpy( PseudoLinkedPointer+4+4+4, wrd, wrdlen );
06,197       // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,198       //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
06,199       //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,200       //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
06,201       //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
06,202       //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,203       //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
06,204       //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
06,205       //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,206       //fwrite(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
06,207       // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,208       // [ //r.14+
06,209       memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
06,210       memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &wrdAUX[0], (KeySize+1+8) );
06,211       memcpy( &LEAF[8 + 8 + 8], &wrd[0], (KeySize+1+8) );
06,212       if (BSTorBtree_RAM == 2) {
06,213           fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,214           fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
06,215           } else { // ##### 64bit memory manipulations [
06,216           memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
06,217           } // ##### 64bit memory manipulations ]
06,218       // ] //r.14+
06,219   }
06,220 }
06,221 if (POffsetInLEAF == 8) // wrd > [LW][ ] so [LW][ ] -> [LW][wrd]

```

```

06,222     {
06,223 //         memcpy( PseudoLinkedPointer+4+4+4*wordlen, wrd, wordlen );
06,224 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one! Here NO need!
06,225     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (KeySize+1+8);
06,226         if (BSTorBtree_RAM == 2) {
06,227             fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,228             fwrite(&wrd[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
06,229         } else { // ##### 64bit memory manipulations [
06,230             memcpy( (char *)PseudoLinkedPointerAUX_64, &wrd[0], (KeySize+1+8) );
06,231         } // ##### 64bit memory manipulations ]
06,232     // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one! Here NO need!
06,233 }
06,234 }
06,235
06,236 if (SplitOccured != 0)
06,237 {
06,238 // ~ Second deal with the INNER NODE(S) i.e Case #3 & Case #4:
06,239     while (StackPtr != 0 || SplitOccured != 0)
06,240     {
06,241         // 'PseudoLinkedPointerNEW' is new LEAF to be inserted
06,242         // 'wrdUP' is NEW word to be inserted
06,243         if (StackPtr != 0)
06,244         {
06,245             POffsetInLEAF = BSTstack[--StackPtr];
06,246             PseudoLinkedPointer_64 = BSTstack[--StackPtr];
06,247 //if ( *(char *) (PseudoLinkedPointer+4+4+4*wordlen) != 0 ) // If LEAF is full: Case #4
06,248             // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,249             //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
06,250             //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,251             //fread(&SomeByte, 1, 1, fp_outRG);
06,252             // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,253             // [ //r.14+
06,254             PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
06,255             if (BSTorBtree_RAM == 2) {
06,256                 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,257                 fread(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fread++;
06,258             } else { // ##### 64bit memory manipulations [
06,259                 memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64, 8+8+8+2*(KeySize+1+8) );
06,260             } // ##### 64bit memory manipulations ]
06,261             memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (KeySize+1+8)], 1 );
06,262             // ] //r.14+
06,263             if (SomeByte != 0 ) // RW exists
06,264         { SplitOccured = 1;
06,265             memcpy( wrdUPold, wrdUP, wordlen ); // LW up
06,266             PseudoLinkedPointerNEWold = PseudoLinkedPointerNEW;
06,267             memcpy( wrdUPold, wrdUP, (KeySize+1+8) );
06,268             PseudoLinkedPointerNEWold_64 = PseudoLinkedPointerNEW_64;
06,269             // ALlocate NEW LEAF:
06,270             if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wordlen + 4 + 4 + 4 < GRMBLFoolAgain((int)wordlen) ) // +4 more for BST instead of LL; + more(see
06,271             // LEAF)
06,272             {
06,273                 memcpy( &PseudoLinkedPointerNEW, &bufend[LetterOffset], 4 );
06,274                 bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
06,275                 bufend[LetterOffset] = bufend[LetterOffset] + 2*wordlen;
06,276                 if (MAXusedBuffer[wordlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wordlen] = (unsigned long)(bufend[LetterOffset] -
06,277                 BufStart);}
06,278             }
06,279             // BUG IN R.17 and prior - the next 'if' is only for RAM (y/Y) not for SSD (z/Z):

```

```

06,278         if (BSTorBtree_RAM == 2) { //r.18
06,279 BUGGYoffset = (BufEnd_64_RAM-(0+14));
06,280         } else { // ##### 64bit memory manipulations [
06,281 BUGGYoffset = (BufEnd_64_RAM-(unsigned long long)pointerflush_64_RAM);
06,282         } // ##### 64bit memory manipulations ]
06,283         if( 8 + 8 + 8 + 2*(KeySize+1+8) < size_in64_L14_RAM - BUGGYoffset ) // the longest wrdlen is LongestLineInclusive but actual is LongestLineInclusive(+ CR
char)
06,284         {
06,285             PseudoLinkedPointerNEW_64 = BufEnd_64_RAM;
06,286             BufEnd_64_RAM = BufEnd_64_RAM + 8 + 8 + 8;
06,287             BufEnd_64_RAM = BufEnd_64_RAM + 2*(KeySize+1+8);
06,288             // [ //r.14+
06,289             //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64;
06,290             //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,291             //fread(&LEAFNEW[0], 8+8+2*(LongestLineInclusive+1+4), 1, fp_outRG);
06,292             // In fact above three lines are slow, the only need is ZEROed LEAFNEW.
06,293             memset(&LEAFNEW[0],0,8+8+2*(KeySize+1+8));
06,294             // ] //r.14+
06,295         }
06,296     else
06,297     { printf( "\nLeprechaun: Failure! Increment 'Memory for B-trees'!\n" );
06,298 #ifdef LITE
06,299 #else
06,300 //             fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, 11TOaDigits, 10), _ui64toaKAZEcomma((unsigned long
long)WORDcountDistinct, 11TOaDigits2, 10) );
06,301             fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14_RAM, 11TOaDigits, 10) );
06,302             fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11TOaDigits, 10) );
06,303             fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
06,304             fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for B-trees'!\n\n");
06,305 #endif
06,306
06,307 if ( HashChunkSizeInBITS_GLOBAL > 0 ) { // 2019-Dec-07
06,308     HashChunkSizeInBITS_GLOBAL--;
06,309     printf( "Automatically increasing number of passes in order to fit B-trees into Target Buffer.\n" );
06,310     fflush(stdout);
06,311     goto NewAttemptRaisePasses;
06,312 } else exit( 7 );
06,313     }
06,314     if (PoffsetInLEAF == 0) // wrdUPold < LW
06,315     {
06,316 //         memcpy( wrdUP, PseudoLinkedPointer+4+4+4, wrdlen ); // LW up
06,317 //         memcpy( PseudoLinkedPointer+4+4+4, wrdUPold, wrdlen ); // wrdUPold go to OLD LEAF
06,318 //         memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
06,319 //         *(char *)(&PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
06,320 //         // [LP](&PseudoLinkedPointerNEWold)[MP][RP][LW][RW] -----
06,321 //         // pair [LW] PseudoLinkedPointerNEW goes up !
06,322 //         // PseudoLinkedPointer: PseudoLinkedPointerNEW: !
06,323 //         // [LP](&PseudoLinkedPointerNEWold)[wrdUPold] [MP][RP][LW][RW] <----
06,324 //         // no need to put zero in RP because logic is based on words existence:
06,325 //         memcpy( PseudoLinkedPointerNEW+0, PseudoLinkedPointer+4, 4 );
06,326 //         memcpy( PseudoLinkedPointerNEW+4, PseudoLinkedPointer+8, 4 );
06,327 //         memcpy( PseudoLinkedPointer+4, &PseudoLinkedPointerNEWold, 4 );
06,328 //         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,329 //         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
06,330 //         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,331 //         //fread(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
06,332 //         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,333 //         //fwrite(&wrdUPold[0], (LongestLineInclusive+1+4), 1, fp_outRG);

```

```

06,334 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
06,335 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,336 //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
06,337 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
06,338 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,339 //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
06,340 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
06,341 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,342 //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
06,343 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
06,344 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,345 //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
06,346 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 0;
06,347 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,348 //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
06,349 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
06,350 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,351 //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
06,352 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8;
06,353 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,354 //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
06,355 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
06,356 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,357 //fwrite(&PseudoLinkedPointerNEWold_64, 8, 1, fp_outRG);
06,358 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,359 // [ //r.14+
06,360 memcpy( &wrdUP[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
06,361 memcpy( &LEAF[8 + 8 + 8], &wrdUPold[0], (KeySize+1+8) );
06,362 memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
06,363 memcpy( &LEAFNEW[8 + 8 + 8], &wrdAUX[0], (KeySize+1+8) );
06,364 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
06,365 memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[8], 8 );
06,366 memcpy( &LEAFNEW[0], &PseudoLinkedPointerAUXdumbo_64, 8 );
06,367 memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[16], 8 );
06,368 memcpy( &LEAFNEW[8], &PseudoLinkedPointerAUXdumbo_64, 8 );
06,369 memcpy( &LEAF[8], &PseudoLinkedPointerNEWold_64, 8 );
06,370 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
06,371 if (BSTorBtree_RAM == 2) {
06,372 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,373 fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
06,374 } else { // ##### 64bit memory manipulations [
06,375 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
06,376 } // ##### 64bit memory manipulations ]
06,377 PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64;
06,378 if (BSTorBtree_RAM == 2) {
06,379 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,380 fwrite(&LEAFNEW[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
06,381 } else { // ##### 64bit memory manipulations [
06,382 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAFNEW[0], 8+8+8+2*(KeySize+1+8) );
06,383 } // ##### 64bit memory manipulations ]
06,384 // ] //r.14+
06,385 }
06,386 if (PoffsetInLEAF == 8) // LW < wrdUPold < RW
06,387 {
06,388 // memcpy( wrdUP, wrdUPold, wrdlen ); // wrdUPold up
06,389 // memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
06,390 // *(char *)PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
06,391 // [LP][MP](PseudoLinkedPointerNEWold)[RP][LW](wrdUPold)[RW] -----

```

```

06,392 //                //                pair [wrdUPold] PseudoLinkedPointerNEW goes up                |
06,393 //                // PseudoLinkedPointer: PseudoLinkedPointerNEW:                |
06,394 //                // [LP][MP][][LW] (PseudoLinkedPointerNEWold)[RP][][RW]                <----
06,395 //                // no need to put zero in RP because logic is based on words existence:
06,396 //                memcpy( PseudoLinkedPointerNEW+0, &PseudoLinkedPointerNEWold, 4 );
06,397 //                memcpy( PseudoLinkedPointerNEW+4, PseudoLinkedPointer+8, 4 );
06,398 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,399 //     memcpy( wrdUP, wrdUPold, (KeySize+1+8) );
06,400 //     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
06,401 //     //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,402 //     //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
06,403 //     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
06,404 //     //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,405 //     //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
06,406 //     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
06,407 //     //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,408 //     //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
06,409 //     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 0;
06,410 //     //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,411 //     //fwrite(&PseudoLinkedPointerNEWold_64, 8, 1, fp_outRG);
06,412 //     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
06,413 //     //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,414 //     //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
06,415 //     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8;
06,416 //     //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,417 //     //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
06,418 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,419 // [ //r.14+
06,420 //     memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
06,421 //     memcpy( &LEAFNEW[8 + 8 + 8], &wrdAUX[0], (KeySize+1+8) );
06,422 //     memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
06,423 //     memcpy( &LEAFNEW[0], &PseudoLinkedPointerNEWold_64, 8 );
06,424 //     memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[16], 8 );
06,425 //     memcpy( &LEAFNEW[8], &PseudoLinkedPointerAUXdumbo_64, 8 );
06,426 //     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
06,427 //     if (BSTorBtree_RAM == 2) {
06,428 //         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,429 //         fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
06,430 //     } else { // ##### 64bit memory manipulations [
06,431 //         memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
06,432 //     } // ##### 64bit memory manipulations ]
06,433 //     PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64;
06,434 //     if (BSTorBtree_RAM == 2) {
06,435 //         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,436 //         fwrite(&LEAFNEW[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
06,437 //     } else { // ##### 64bit memory manipulations [
06,438 //         memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAFNEW[0], 8+8+8+2*(KeySize+1+8) );
06,439 //     } // ##### 64bit memory manipulations ]
06,440 // ] //r.14+
06,441 // }
06,442 // if (POffsetInLEAF == 16) // wrdUPold > RW
06,443 // {
06,444 //     memcpy( wrdUP, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW up
06,445 //     *(char *)PseudoLinkedPointer+4+4+4+wrdlen = 0; // RW mark unused in OLD LEAF
06,446 //     memcpy( PseudoLinkedPointerNEW+4+4+4, wrdUPold, wrdlen ); // wrdUPold go to NEW LEAF
06,447 //     // [LP][MP][RP](PseudoLinkedPointerNEWold)[LW][RW](wrdUPold) -----
06,448 //     //                pair [RW] PseudoLinkedPointerNEW goes up                |
06,449 //     // PseudoLinkedPointer: PseudoLinkedPointerNEW:                |

```

```

06,450 //          // [LP][MP][][LW]          [RP](PseudoLinkedPointerNEWold)[](wrUPold) <---
06,451 //          // no need to put zero in RP because logic is based on words existence:
06,452 //          memcpy( PseudoLinkedPointerNEW+0, PseudoLinkedPointer+8, 4 );
06,453 //          memcpy( PseudoLinkedPointerNEW+4, &PseudoLinkedPointerNEWold, 4 );
06,454 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,455 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
06,456 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,457 //fread(&wrUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
06,458 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
06,459 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,460 //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
06,461 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
06,462 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,463 //fwrite(&wrUPold[0], (LongestLineInclusive+1+4), 1, fp_outRG);
06,464 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
06,465 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,466 //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
06,467 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 0;
06,468 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,469 //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
06,470 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8;
06,471 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,472 //fwrite(&PseudoLinkedPointerNEWold_64, 8, 1, fp_outRG);
06,473 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,474 // [ //r.14+
06,475 memcpy( &wrUP[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
06,476 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
06,477 memcpy( &LEAFNEW[8 + 8 + 8], &wrUPold[0], (KeySize+1+8) );
06,478 memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[16], 8 );
06,479 memcpy( &LEAFNEW[0], &PseudoLinkedPointerAUXdumbo_64, 8 );
06,480 memcpy( &LEAFNEW[8], &PseudoLinkedPointerNEWold_64, 8 );
06,481 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
06,482 if (BSTorBtree_RAM == 2) {
06,483 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,484 fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
06,485 } else { // ##### 64bit memory manipulations [
06,486 memcpy( (char *)&PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
06,487 } // ##### 64bit memory manipulations ]
06,488 PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64;
06,489 if (BSTorBtree_RAM == 2) {
06,490 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,491 fwrite(&LEAFNEW[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
06,492 } else { // ##### 64bit memory manipulations [
06,493 memcpy( (char *)&PseudoLinkedPointerAUX_64, &LEAFNEW[0], 8+8+8+2*(KeySize+1+8) );
06,494 } // ##### 64bit memory manipulations ]
06,495 // ] //r.14+
06,496 }
06,497 }
06,498 else // If LEAF is not full: Case #3
06,499 { SplitOccured = 0;
06,500 if (POffsetInLEAF == 0) // wrUP < [LW][ ] so [LW][ ] -> [ ][LW] -> [wrUP][LW]
06,501 {
06,502 //          memcpy( PseudoLinkedPointer+4+4+4+wrUPlen, PseudoLinkedPointer+4+4+4, wrUPlen );
06,503 //          memcpy( PseudoLinkedPointer+4+4+4, wrUP, wrUPlen );
06,504 //          // [LP][MP][ ] -> [LP][ ][MP] -> [LP][np][MP]
06,505 //          memcpy( PseudoLinkedPointer+8, PseudoLinkedPointer+4, 4 );
06,506 //          memcpy( PseudoLinkedPointer+4, &PseudoLinkedPointerNEW, 4 );
06,507 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!

```

```

06,508 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
06,509 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,510 //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
06,511 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
06,512 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,513 //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
06,514 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
06,515 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,516 //fwrite(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
06,517 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
06,518 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,519 //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
06,520 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
06,521 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,522 //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
06,523 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
06,524 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,525 //fwrite(&PseudoLinkedPointerNEW_64, 8, 1, fp_outRG);
06,526 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,527 // [ //r.14+
06,528 memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
06,529 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &wrdAUX[0], (KeySize+1+8) );
06,530 memcpy( &LEAF[8 + 8 + 8], &wrdUP[0], (KeySize+1+8) );
06,531 memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[8], 8 );
06,532 memcpy( &LEAF[8 + 8], &PseudoLinkedPointerAUXdumbo_64, 8 );
06,533 memcpy( &LEAF[8], &PseudoLinkedPointerNEW_64, 8 );
06,534 if (BSTorBtree_RAM == 2) {
06,535 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,536 fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
06,537 } else { // ##### 64bit memory manipulations [
06,538 memcpy( (char *)&PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
06,539 } // ##### 64bit memory manipulations ]
06,540 // ] //r.14+
06,541 }
06,542 if (PoffsetInLEAF == 8) // wrdUP > [LW][ ] so [LW][ ] -> [LW][wrdUP]
06,543 {
06,544 //     memcpy( PseudoLinkedPointer+4+4+wrdlen, wrdUP, wrdlen );
06,545 //     // [LP][MP][ ] -> [LP][MP][np]
06,546 //     memcpy( PseudoLinkedPointer+8, &PseudoLinkedPointerNEW, 4 );
06,547 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,548 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
06,549 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,550 //fwrite(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
06,551 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
06,552 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,553 //fwrite(&PseudoLinkedPointerNEW_64, 8, 1, fp_outRG);
06,554 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,555 // [ //r.14+
06,556 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &wrdUP[0], (KeySize+1+8) );
06,557 memcpy( &LEAF[8 + 8], &PseudoLinkedPointerNEW_64, 8 );
06,558 if (BSTorBtree_RAM == 2) {
06,559 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,560 fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
06,561 } else { // ##### 64bit memory manipulations [
06,562 memcpy( (char *)&PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
06,563 } // ##### 64bit memory manipulations ]
06,564 // ] //r.14+
06,565 }

```

```

06,566         break;
06,567     }
06,568     }
06,569     else // Empty stack means ROOT and more over ROOT is already splitted(Case #4 is off)
06,570     {
06,571         // If LEAF is not full: Case #3
06,572         // THIS IS WHERE A NEW(SECOND) LEAF 'PseudoLinkedPointerROOT' must be allocated:
06,573         // if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wordlen + 4 + 4 + 4 < GRMBLFoolAgain((int)wordlen) ) // +4 more for BST instead of LL; + more(see
LEAF)
06,574         // {
06,575         //     memcpy( &PseudoLinkedPointerROOT, &bufend[LetterOffset], 4 );
06,576         //     bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
06,577         //     memcpy( bufend[LetterOffset], wrdUP, wordlen );
06,578         //     bufend[LetterOffset] = bufend[LetterOffset] + 2*wordlen;
06,579         //     if (MAXusedBuffer[wordlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wordlen] = (unsigned long)(bufend[LetterOffset] -
BufStart);}
06,580         // }
06,581         // BUG IN R.17 and prior - the next 'if' is only for RAM (y/Y) not for SSD (z/Z):
06,582         if (BSTorBtree_RAM == 2) { //r.18
06,583             BUGGYoffset = (BufEnd_64_RAM-(0+14));
06,584             } else { // ##### 64bit memory manipulations [
06,585             BUGGYoffset = (BufEnd_64_RAM-(unsigned long long)pointerflush_64_RAM);
06,586             } // ##### 64bit memory manipulations ]
06,587         if( 8 + 8 + 8 + 2*(KeySize+1+8) < size_in64_L14_RAM - BUGGYoffset ) // the longest wordlen is LongestLineInclusive but actual is LongestLineInclusive(+ CR
char)
06,588         {
06,589             PseudoLinkedPointerROOT_64 = BufEnd_64_RAM;
06,590             BufEnd_64_RAM = BufEnd_64_RAM + 8 + 8 + 8;
06,591             BufEnd_64_RAM = BufEnd_64_RAM + 2*(KeySize+1+8);
06,592             PseudoLinkedPointerAUX_64 = PseudoLinkedPointerROOT_64 + 8 + 8 + 8;
06,593             if (BSTorBtree_RAM == 2) {
06,594                 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,595                 fwrite(&wrdUP[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
06,596                 } else { // ##### 64bit memory manipulations [
06,597                 memcpy( (char *)PseudoLinkedPointerAUX_64, &wrdUP[0], (KeySize+1+8) );
06,598                 } // ##### 64bit memory manipulations ]
06,599             }
06,600         else
06,601         { printf( "\nLeprechaun: Failure! Increment 'Memory for B-trees'!\n" );
06,602         #ifdef LITE
06,603         #else
06,604             fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, 11ToaDigits, 10), _ui64toaKAZEcomma((unsigned long
long)WORDcountDistinct, 11ToaDigits2, 10) );
06,605             fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14_RAM, 11ToaDigits, 10) );
06,606             fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11ToaDigits, 10) );
06,607             fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
06,608             fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for B-trees'!\n\n");
06,609         #endif
06,610         if ( HashChunkSizeInBITS_GLOBAL > 0 ) { // 2019-Dec-07
06,611             HashChunkSizeInBITS_GLOBAL--;
06,612             printf( "Automatically increasing number of passes in order to fit B-trees into Target Buffer.\n" );
06,613             goto NewAttemptRaisePasses;
06,614         } else exit( 7 );
06,615         }
06,616         // Here -- 'PseudoLinkedPointerROOT' --
06,617         // | (wrdUP) |
06,618         // 'PseudoLinkedPointer' <-- --> 'PseudoLinkedPointerNEW'
06,619         // (LW) (RW)

```



```

06,620 //      memcpy( PseudoLinkedPointerROOT, &PseudoLinkedPointer, 4 );      // LP
06,621 //      memcpy( PseudoLinkedPointerROOT+4, &PseudoLinkedPointerNEW, 4 ); // MP
06,622 //      // Here must NEW ROOT be updated i.e. HASH table(SLOT) must point it:
06,623 //      memcpy( BufStart+Slot, &PseudoLinkedPointerROOT, 4 );
06,624      PseudoLinkedPointerAUX_64 = PseudoLinkedPointerROOT_64;
06,625      if (BSTorBtree_RAM == 2) {
06,626      fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,627      fwrite(&PseudoLinkedPointer_64, 8, 1, fp_outRG); Total_fwrite++;
06,628      } else { // ##### 64bit memory manipulations [
06,629      memcpy( (char *)&PseudoLinkedPointerAUX_64, &PseudoLinkedPointer_64, 8 );
06,630      } // ##### 64bit memory manipulations ]
06,631      PseudoLinkedPointerAUX_64 = PseudoLinkedPointerROOT_64 + 8;
06,632      if (BSTorBtree_RAM == 2) {
06,633      fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,634      fwrite(&PseudoLinkedPointerNEW_64, 8, 1, fp_outRG); Total_fwrite++;
06,635      } else { // ##### 64bit memory manipulations [
06,636      memcpy( (char *)&PseudoLinkedPointerAUX_64, &PseudoLinkedPointerNEW_64, 8 );
06,637      } // ##### 64bit memory manipulations ]
06,638      memcpy( BufStart+Slot, &PseudoLinkedPointerROOT_64, 8 );
06,639      break; //because it is ROOT without split
06,640  }
06,641  } // while
06,642 } //if (SplitOccured != 0)
06,643 // 3] Insert Iterative ]
06,644 } //if (FoundInLinkedList == 0)
06,645 // ##### B-tree order 3 fragment 64bit ]
06,646
06,647 } //if ( (Slot>>HashChunkSizeInBITS) == RipPasses ) {
06,648
06,649 SKIPaPositionSinceItIsUnique;; // 2021-Jul-05
06,650
06,651 } //for (BuildingBlocksSTRIDE=0;
06,652
06,653 // PASS #1: ]
06,654
06,655 // PASS #2: [
06,656
06,657 //KeySize = MatchLens[jj]; // UNNECESSARY for non-SHA3, and buggy for SHA3 - it should be 'wrklen', // 2020-Feb-02
06,658 for( jjj = 0; jjj < ( (1LL)<<HashInBITS_GLOBAL ); jjj++ ) //r.18
06,659 {
06,660
06,661      Slot_RAM = (jjj<<3) +(MatchLensNUM*(1LL<<HashInBITS_GLOBAL)*8); // Check out all slots of TEMPORARY
06,662      memcpy( &PseudoLinkedPointer_64DUMP, BufStart+Slot_RAM, 8 );
06,663
06,664      if (PseudoLinkedPointer_64DUMP != 0)
06,665      {
06,666
06,667 // CAUTION: Did forget that dumping/traversing the B-trees - DESTROYS them! On purpose, pointers are zeroed.
06,668 // $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ B-tree order 3 traverse 64bit [
06,669 // DONE JOB:
06,670 // Must be written B-tree traverse ! with simulated stack i.e. non-recursive.
06,671 // ...
06,672      StackPtrDUMP = 0;
06,673      while ( 2==2 ) {
06,674      while (PseudoLinkedPointer_64DUMP != 0)
06,675      {
06,676      if (StackPtrDUMP > 8192*3-1) { printf( "\nLeprechaun: Failure! B-tree simulated stack overflow, too high B-tree!\n" ); exit( 13 );}
06,677      BSTstackDUMP[StackPtrDUMP] = PseudoLinkedPointer_64DUMP; ++StackPtrDUMP; //ptr to Rwrdr

```

```

06,678 //if ( *(char *) (PseudoLinkedPointer + 4 + 4 + 4 + i%31+1) == 0 ) {memcpy( PseudoLinkedPointer + 4 + 4, &BufStart[NumberOfSLOTS*4], 4 );}
06,679 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,680 //PseudoLinkedPointerAUX_64DUMP = PseudoLinkedPointer_64DUMP + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
06,681 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64DUMP);
06,682 //fread(&SomeByte, 1, 1, fp_outRG);
06,683 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,684 // [ //r.14+
06,685 PseudoLinkedPointerAUX_64DUMP = PseudoLinkedPointer_64DUMP;
06,686 if (BSTorBtree_RAM == 2) {
06,687 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64DUMP);
06,688 fread(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fread++;
06,689 } else { // ##### 64bit memory manipulations [
06,690 memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64DUMP, 8+8+2*(KeySize+1+8) );
06,691 } // ##### 64bit memory manipulations ]
06,692 memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (KeySize+1+8)], 1 );
06,693 // ] //r.14+
06,694 if (SomeByte == 0 ) // RW exists not
06,695 {
06,696 PseudoLinkedPointerAUX_64DUMP = PseudoLinkedPointer_64DUMP + 8 + 8;
06,697 if (BSTorBtree_RAM == 2) {
06,698 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64DUMP);
06,699 fwrite(&NULLs_64, 8, 1, fp_outRG); Total_fwrite++;
06,700 } else { // ##### 64bit memory manipulations [
06,701 memcpy( (char *)PseudoLinkedPointerAUX_64DUMP, &NULLs_64, 8 );
06,702 } // ##### 64bit memory manipulations ]
06,703 // [ //r.14+
06,704 memcpy( &LEAF[8 + 8], &NULLs_64, 8 );
06,705 // ] //r.14+
06,706 }
06,707 // memcpy( &PseudoLinkedPointerNEWleft, PseudoLinkedPointer, 4 );
06,708 // memcpy( &PseudoLinkedPointerNEWMiddle, PseudoLinkedPointer + 4, 4 );
06,709 // memcpy( &PseudoLinkedPointerNEWright, PseudoLinkedPointer + 4 + 4, 4 );
06,710 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,711 //PseudoLinkedPointerAUX_64DUMP = PseudoLinkedPointer_64DUMP;
06,712 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64DUMP);
06,713 //fread(&PseudoLinkedPointerNEWleft_64, 8, 1, fp_outRG);
06,714 //PseudoLinkedPointerAUX_64DUMP = PseudoLinkedPointer_64DUMP + 8;
06,715 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64DUMP);
06,716 //fread(&PseudoLinkedPointerNEWMiddle_64, 8, 1, fp_outRG);
06,717 //PseudoLinkedPointerAUX_64DUMP = PseudoLinkedPointer_64DUMP + 8 + 8;
06,718 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64DUMP);
06,719 //fread(&PseudoLinkedPointerNEWright_64, 8, 1, fp_outRG);
06,720 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,721 // [ //r.14+
06,722 memcpy( &PseudoLinkedPointerNEWleft_64, &LEAF[0], 8 );
06,723 memcpy( &PseudoLinkedPointerNEWMiddle_64, &LEAF[8], 8 );
06,724 memcpy( &PseudoLinkedPointerNEWright_64, &LEAF[8+8], 8 );
06,725 // ] //r.14+
06,726 // Give first from right to left non-zero PTR
06,727 if (PseudoLinkedPointerNEWright_64 != 0 )
06,728 { memcpy( PseudoLinkedPointer + 4 + 4, &BufStart[NumberOfSLOTS*4], 4 );
06,729 PseudoLinkedPointer = PseudoLinkedPointerNEWright;
06,730 }
06,731 {
06,732 PseudoLinkedPointerAUX_64DUMP = PseudoLinkedPointer_64DUMP + 8 + 8;
06,733 if (BSTorBtree_RAM == 2) {
06,734 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64DUMP);
06,735 fwrite(&NULLs_64, 8, 1, fp_outRG); Total_fwrite++;

```

```

06,736         } else { // ##### 64bit memory manipulations [
06,737         memcpy( (char *)PseudoLinkedPointerAUX_64DUMP, &NULLs_64, 8 );
06,738         } // ##### 64bit memory manipulations ]
06,739     PseudoLinkedPointer_64DUMP = PseudoLinkedPointerNEWright_64;
06,740     }
06,741     else if (PseudoLinkedPointerNEWmiddle_64 != 0 )
06,742     { memcpy( PseudoLinkedPointer + 4, &BufStart[NumberOfSLOTS*4], 4 );
06,743       PseudoLinkedPointer = PseudoLinkedPointerNEWmiddle;
06,744     }
06,745     {
06,746     PseudoLinkedPointerAUX_64DUMP = PseudoLinkedPointer_64DUMP + 8;
06,747     if (BSTorBtree_RAM == 2) {
06,748     fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64DUMP);
06,749     fwrite(&NULLs_64, 8, 1, fp_outRG); Total_fwrite++;
06,750     } else { // ##### 64bit memory manipulations [
06,751     memcpy( (char *)PseudoLinkedPointerAUX_64DUMP, &NULLs_64, 8 );
06,752     } // ##### 64bit memory manipulations ]
06,753     PseudoLinkedPointer_64DUMP = PseudoLinkedPointerNEWmiddle_64;
06,754     }
06,755     else if (PseudoLinkedPointerNEWleft_64 != 0 )
06,756     { memcpy( PseudoLinkedPointer, &BufStart[NumberOfSLOTS*4], 4 );
06,757       PseudoLinkedPointer = PseudoLinkedPointerNEWleft;
06,758     }
06,759     {
06,760     PseudoLinkedPointerAUX_64DUMP = PseudoLinkedPointer_64DUMP;
06,761     if (BSTorBtree_RAM == 2) {
06,762     fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64DUMP);
06,763     fwrite(&NULLs_64, 8, 1, fp_outRG); Total_fwrite++;
06,764     } else { // ##### 64bit memory manipulations [
06,765     memcpy( (char *)PseudoLinkedPointerAUX_64DUMP, &NULLs_64, 8 );
06,766     } // ##### 64bit memory manipulations ]
06,767     PseudoLinkedPointer_64DUMP = PseudoLinkedPointerNEWleft_64;
06,768     }
06,769     else
06,770     {
06,771     PseudoLinkedPointer_64DUMP = 0;
06,772     }
06,773     }
06,774     if (LevelsInCorona_Not_Counting_ROOT < StackPtrDUMP) LevelsInCorona_Not_Counting_ROOT = StackPtrDUMP; //r.14
06,775     if (StackPtrDUMP == 0) break;
06,776     PseudoLinkedPointer_64DUMP = BSTstackDUMP[--StackPtrDUMP];
06,777     // memcpy( &PseudoLinkedPointerNEWleft, PseudoLinkedPointer, 4 );
06,778     // memcpy( &PseudoLinkedPointerNEWmiddle, PseudoLinkedPointer + 4, 4 );
06,779     // memcpy( &PseudoLinkedPointerNEWright, PseudoLinkedPointer + 4 + 4, 4 );
06,780     // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,781     //PseudoLinkedPointerAUX_64DUMP = PseudoLinkedPointer_64DUMP;
06,782     //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64DUMP);
06,783     //fread(&PseudoLinkedPointerNEWleft_64, 8, 1, fp_outRG);
06,784     //PseudoLinkedPointerAUX_64DUMP = PseudoLinkedPointer_64DUMP + 8;
06,785     //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64DUMP);
06,786     //fread(&PseudoLinkedPointerNEWmiddle_64, 8, 1, fp_outRG);
06,787     //PseudoLinkedPointerAUX_64DUMP = PseudoLinkedPointer_64DUMP + 8 + 8;
06,788     //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64DUMP);
06,789     //fread(&PseudoLinkedPointerNEWright_64, 8, 1, fp_outRG);
06,790     // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,791     // [ //r.14+
06,792     PseudoLinkedPointerAUX_64DUMP = PseudoLinkedPointer_64DUMP;
06,793     if (BSTorBtree_RAM == 2) {

```

```

06,794         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64DUMP);
06,795         fread(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fread++;
06,796     } else { // ##### 64bit memory manipulations [
06,797         memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64DUMP, 8+8+2*(KeySize+1+8) );
06,798     } // ##### 64bit memory manipulations ]
06,799     memcpy( &PseudoLinkedPointerNEWleft_64, &LEAF[0], 8 );
06,800     memcpy( &PseudoLinkedPointerNEWMiddle_64, &LEAF[8], 8 );
06,801     memcpy( &PseudoLinkedPointerNEWright_64, &LEAF[8+8], 8 );
06,802     // ] //r.14+
06,803     if (PseudoLinkedPointerNEWleft_64 + PseudoLinkedPointerNEWMiddle_64 + PseudoLinkedPointerNEWright_64 == 0) // One LEAF is PRINTED when LP=0 MP=0 RP=0
06,804     {
06,805         // memcpy( wrd, PseudoLinkedPointer + 4 + 4 + 4, i%31+1 );
06,806         // fwrite(wrd, i%31+1, 1, fp_out);
06,807         // fwrite(CRdLFa, 2, 1, fp_out);
06,808         // if ( *(char *)PseudoLinkedPointer + 4 + 4 + 4 + i%31+1 != 0 )
06,809         // { memcpy( wrd, PseudoLinkedPointer + 4 + 4 + 4 + i%31+1, i%31+1 );
06,810         //     fwrite(wrd, i%31+1, 1, fp_out);
06,811         //     fwrite(CRdLFa, 2, 1, fp_out);
06,812         // }
06,813         // PseudoLinkedPointer = 0;
06,814         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,815         //PseudoLinkedPointerAUX_64DUMP = PseudoLinkedPointer_64DUMP + 8 + 8 + 8;
06,816         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64DUMP);
06,817         //fread(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
06,818         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,819         // [ //r.14+
06,820         memcpy( &wrd[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
06,821         // ] //r.14+
06,822         // Counter [
06,823         //memcpy( &CounterOccurrences, &wrd[(KeySize+1+4)-4], 4 );
06,824         //if (CounterOccurrences<999999999) CounterOccurrences++; // Starting from ZERO! Because when insertion happened there was no setting counter to 1.
06,825         memcpy( &CounterOccurrences, &wrd[(KeySize+1+8)-8], 8 ); // 2019-Dec-26
06,826         FoundInLinkedList_RAM = 0; // Assume not found // 2019-Dec-26
06,827         if (CounterOccurrences>1) FoundInLinkedList_RAM = 1; // 2019-Dec-26
06,828         memcpy( &CounterOccurrences, &wrd[(KeySize+1+8)-8], 8 ); // 2019-Dec-26
06,829         memset(&wrd[(KeySize+1+8)-8], 0, 8); // 2021-Jan-13, NastyBugFixed, for a long time this nasty bug remained uncrushed, namely not nullifying the
3bytes for 'CounterOccurrences' now serving also as LastSeenOffset, simply forgot to zeroing it :(
06,830
06,831 // Have to back up the LEAF since it is changed in 1of2 and 2of2 sections:
06,832     memcpy( &LEAFbackup[0], &LEAF[0], 8+8+2*(KeySize+1+8) );
06,833
06,834 if (FoundInLinkedList_RAM == 1) TotalNonUnique[MatchLens[jj]]++; // 2020-Jun-09
06,835 // Building nonunique 1 of 2 [
06,836
06,837 #if defined(_NSHA3) || defined(_NPRV) || defined(_NDD) || defined(_NAquaHash) //2020-Nov-12
06,838     memcpy( &wrdlen, &wrd[ 0 ], 1 ); // The keysize is 1..255, pretty enough since my needs are 1..28 (up to SHA3-224). // 2020-Feb-02
06,839 #else
06,840     wrdlen=MatchLens[jj];
06,841 #endif
06,842
06,843 //memset( &wrd[0], 0, (LongestLineInclusive+1+8) ); //r.18, see below, the old nullifier is commented. // 2019-Dec-26
06,844
06,845 //         if (wrdlen > 28) {
06,846 //             FIPS202_SHA3_224((unsigned char *) (SourceBlock+BuildingBlocksSTRIDE), wrdlen, (unsigned char *)SHA328bytes);
06,847 //             wrdlen=28;
06,848 //         }
06,849
06,850     memcpy( &wrd[ 0 ], &wrdlen, 1 ); // The keysize is 1..255, pretty enough since my needs are 1..28 (up to SHA3-224).

```

```

06,851 //if (memcmp( &wrd[ 0 ], &wrdlen, 1 )) printf ("\nwrld != wrdlen\n");
06,852
06,853 //          if (wrdlen < 28)
06,854 //memcpy( &wrd[ 0+1 ], (unsigned char *)(<SourceBlock+BuildingBlocksSTRIDE), wrdlen ); // 2019-Dec-26
06,855 //          else
06,856 //              memcpy( &wrd[ 0+1 ], (unsigned char *)SHA328bytes, wrdlen );
06,857
06,858 BufStart = pointerflush;
06,859 //      Slot = ((wrd[0]-'_')*28*28*28*28 + (wrd[1]-'_')*28*28*28 + (wrd[2]-'_')*28*28 + (wrd[3]-'_')*28 + (wrd[4]-'_'))<<3;
06,860 //      Slot = FNV1A_Hash_Jesteress_27bit(wrd, wrdlen)<<3; // Commented since r.14+++ because of passes.
06,861 //Slot = FNV1A_Hash_Jesteress_27bit(wrd +1, wrdlen); WORDcount++;
06,862 Slot = FNV1A_Pippip_Yurii(wrd +1, wrdlen); WORDcount++; // Changed 2019-Nov-28
06,863 Slot = Slot<<3;
06,864 GettingIndexOfArray=jj; // same as above atrocity
06,865 Slot = Slot +(GettingIndexOfArray*(1LL<<HashInBITS_GLOBAL)*8); // CAUTION: 'wrdlen' is halved here, when a new revision comes with 1:1 keysize then change it.
06,866
06,867 //KeySize = LongestLineInclusive; // 'LongestLineInclusive' is for 1-way hashing where different KeySizes can occupy same HASH pool.
06,868 KeySize = wrdlen; //r.18, In case multi-way hashing: either the MAX allowed or the 'wrdlen'.
06,869
06,870 //Slot = 0; // One Tree only!
06,871 memcpy( &PseudoLinkedPointer_64, BufStart+Slot, 8 );
06,872
06,873 if (FoundInLinkedList_RAM == 1) { // 2019-Dec-04
06,874 // ##### B-tree order 3 fragment 64bit [
06,875 //
06,876 // LEAF structure: [LeftPointer][MiddlePointer][RightPointer][LeftWord][RightWord]
06,877 //                4bytes      4bytes      4bytes      wrdlen      wrdlen
06,878 //                *          *          *          *          *          <- if *(char *)==0 means the word cell is empty
06,879 // ALL B-tree order 3 fragment consists of 3 sub-fragments:
06,880 // 1] Search 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search 3] Insert Iterative
06,881
06,882 // LEAF_64 structure: [LeftPointer][MiddlePointer][RightPointer][LeftWord] [RightWord]
06,883 //                   8bytes      8bytes      8bytes      LongestLineInclusive+1+4 LongestLineInclusive+1+4
06,884 //                   *          *          *          *          *          <- if *(char *)==0 means the word cell is empty
06,885 // Note: In order to use one fread(and strcmp) a NULL postfix for LeftWord, RightWord i.e. LeftWord_Length=len(LeftWord)+1 a kinda stupid choice ...
06,886 // Note: BufEnd_64 in fact is the first free position after the BUFFER END!
06,887
06,888 // 1] Search [
06,889 //          if (PseudoLinkedPointer_64 == 0) // means EMPTY-SLOT
06,890 //          {
06,891 //              //if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrdlen + 4 + 4 + 4 < GRMBLFoolAgain[(int)wrdlen] ) // +4 more for BST instead of LL; + more(see
06,892 //              {
06,893 //                  memcpy( BufStart+Slot, &bufend[LetterOffset], 4 );
06,894 //                  bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
06,895 //                  memcpy( bufend[LetterOffset], wrd, wrdlen ); WORDcountDistinct++; bufNumberOfWords[LetterOffset]++;
06,896 //                  bufend[LetterOffset] = bufend[LetterOffset] + 2*wrdlen;
06,897 //                  if (MAXusedBuffer[wrdlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrdlen] = (unsigned long)(bufend[LetterOffset] -
06,898 //                  BufStart);}
06,899 //              }
06,900 //              // BUG IN R.17 and prior - the next 'if' is only for RAM (y/Y) not for SSD (z/Z):
06,901 //              if (BSTorBtree == 2) { //r.18
06,902 //                  BUGGYoffset = (BufEnd_64-(0+14));
06,903 //              } else { // ##### 64bit memory manipulations [
06,904 //                  BUGGYoffset = (BufEnd_64-(unsigned long long)pointerflush_64);
06,905 //              } // ##### 64bit memory manipulations ]
06,906 //              if( 8 + 8 + 8 + 2*(KeySize+1+8) < size_in64_L14 - BUGGYoffset ) // the longest wrdlen is LongestLineInclusive but actual is LongestLineInclusive(+ CR char)
06,907 //              {

```

```

06,907         memcpy( BufStart+Slot, &BufEnd_64, 8 );
06,908         BufEnd_64 = BufEnd_64 + 8 + 8 + 8;
06,909         if (BSTorBtree == 2) {
06,910             fsetpos(fp_outRG, (const fpos_t *)&BufEnd_64);
06,911             //fwrite(wrd, wrdlen, 1, fp_outRG); //GRMBL! r.18
06,912             fwrite(wrd, (KeySize+1+8), 1, fp_outRG); //GRMBL! r.18
06,913             WORDcountDistinct++; Total_fwrite++;
06,914             //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
06,915             // r.14++ The above line was commented because the pool is already ZEROed.
06,916             } else { // ##### 64bit memory manipulations [
06,917             //memcpy( (char *)BufEnd_64, wrd, wrdlen ); //GRMBL! r.18
06,918             memcpy( (char *)BufEnd_64, wrd, (KeySize+1+8) ); //GRMBL! r.18
06,919             WORDcountDistinct++;
06,920             } // ##### 64bit memory manipulations ]
06,921             BufEnd_64 = BufEnd_64 + 2*(KeySize+1+8);
06,922             //fsetpos(fp_outRG, &BufEnd_64);
06,923         }
06,924     else
06,925     { printf( "\nLeprechaun: Failure! Increment 'Memory for B-trees'!\n" );
06,926 #ifdef LITE
06,927 #else
06,928 fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, llToaDigits, 10), _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct,
llToaDigits2, 10) );
06,929 fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14, llToaDigits, 10) );
06,930 fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, llToaDigits, 10) );
06,931 fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
06,932         fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for B-trees'!\n\n");
06,933 #endif
06,934         exit( 7 );
06,935     }
06,936         FoundInLinkedList = 1+1;
06,937     }
06,938     else // means USED-SLOT
06,939     { FoundInLinkedList = 0;
06,940     StackPtr = 0;
06,941     //         while (PseudoLinkedPointer != 0 && FoundInLinkedList == 0)
06,942     //         while (PseudoLinkedPointer_64 != 0 && FoundInLinkedList == 0)
06,943     {
06,944     // ***** 'P W P' section [
06,945     // LW: existence check if ( *(char *)(&PseudoLinkedPointer+4+4+4) != 0 )
06,946     // RW: existence check if ( *(char *)(&PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
06,947     // here ALWAYS LW exists: no need for existence check - line below
06,948     // if ( *(char *)(&PseudoLinkedPointer+4+4+4) != 0 )
06,949     //         if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wrdlen) > 0) // go LP
06,950     // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,951     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
06,952     //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,953     //fread(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
06,954     // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,955     // [ //r.14+
06,956     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
06,957     if (BSTorBtree == 2) {
06,958         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,959         fread(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fread++;
06,960         } else { // ##### 64bit memory manipulations [
06,961         memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64, 8+8+2*(KeySize+1+8) );
06,962         } // ##### 64bit memory manipulations ]
06,963         memcpy( &FourGramL[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );

```

```

06,964 // ] //r.14+
06,965 //strFLAG=strcmpKAZE13(FourGramL, wrd);
06,966 strFLAG=memcmp(FourGramL +1, wrd +1, wrdlen); // Since 2019-Feb-07, it is not ASCIIZ string but zero byte is the length of the 1..
06,967 if (strFLAG > 0) // go LP
06,968 { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 0, 4 ); //LP
06,969 PseudoLinkedPointer = PseudoLinkedPointerNEW;
06,970 }
06,971 {
06,972 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,973 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 0; //LP
06,974 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,975 if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
06,976 BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
06,977 BSTstack[StackPtr] = 0; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
06,978 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,979 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,980 //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
06,981 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,982 // [ //r.14+
06,983 memcpy( &PseudoLinkedPointer_64, &LEAF[0], 8 );
06,984 // ] //r.14+
06,985 }
06,986 else if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wrdlen) < 0) // go RP or MP
06,987 else if (strFLAG < 0) // go RP or MP
06,988 { // RW existence check - line below:
06,989 if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 ) // RW exists
06,990 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,991 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
06,992 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
06,993 //fread(&SomeByte, 1, 1, fp_outRG);
06,994 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
06,995 // [ //r.14+
06,996 memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (KeySize+1+8)], 1 );
06,997 // ] //r.14+
06,998 if (SomeByte != 0 ) // RW exists
06,999 { // Here all 'P W P' section is repeated; the way of handling case when dynamic number of words in leaf
07,000 // *****
07,001 // ***** 'P W P' section 2 [
07,002 // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
07,003 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
07,004 // here ALWAYS RW exists: no need for existence check - line below
07,005 // if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
07,006 if (memcmp(PseudoLinkedPointer+4+4+4+wrdlen, wrd, wrdlen) > 0) // go MP
07,007 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,008 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
07,009 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,010 //fread(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
07,011 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,012 // [ //r.14+
07,013 memcpy( &FourGramL[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
07,014 // ] //r.14+
07,015 //strFLAG=strcmpKAZE13(FourGramL, wrd);
07,016 strFLAG=memcmp(FourGramL +1, wrd +1, wrdlen); // Since 2019-Feb-07, it is not ASCIIZ string but zero byte is the length of the 1..
07,017 if (strFLAG > 0) // go MP
07,018 { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
07,019 PseudoLinkedPointer = PseudoLinkedPointerNEW;
07,020 }
07,021 {

```

```

07,022 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,023 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8; //MP
07,024 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,025 if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
07,026 BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
07,027 BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
07,028 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,029 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,030 //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
07,031 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,032 // [ //r.14+
07,033 memcpy( &PseudoLinkedPointer_64, &LEAF[8], 8 );
07,034 // ] //r.14+
07,035 }
07,036 // else if (memcmp(PseudoLinkedPointer+4+4+4*wordlen, wrd, wordlen) < 0) // go RP
07,037 // else if (strFLAG < 0) // go RP
07,038 // { // No ?W after RW - go RP
07,039 //     memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4 + 4, 4 ); //RP
07,040 //     PseudoLinkedPointer = PseudoLinkedPointerNEW;
07,041 // }
07,042 {
07,043 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,044 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8; //RP
07,045 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,046 if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
07,047 BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
07,048 BSTstack[StackPtr] = 16; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
07,049 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,050 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,051 //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
07,052 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,053 // [ //r.14+
07,054 memcpy( &PseudoLinkedPointer_64, &LEAF[8 + 8], 8 );
07,055 // ] //r.14+
07,056 }
07,057 // else { FoundInLinkedList = 1; // wrd is RW
07,058 // Counter [
07,059 //     if (BSTorBtree == 2) {
07,060 // fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,061 //     }
07,062 // memcpy( &CounterOccurrences, &FourGramL[(KeySize+1+4)-4], 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
07,063 // if (CounterOccurrences<999999999) CounterOccurrences++; // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
07,064 // memcpy( &FourGramL[(KeySize+1+4)-4], &CounterOccurrences, 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
07,065 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,066 //fwrite(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
07,067 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,068 // [ //r.14+
07,069 //     memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &FourGramL[0], (KeySize+1+8) );
07,070 //     if (BSTorBtree == 2) {
07,071 //         fwrite(&LEAF[0], 8+8+8*2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
07,072 //     } else { // ##### 64bit memory manipulations [
07,073 //         memcpy( (char *)&PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8*2*(KeySize+1+8) );
07,074 //     } // ##### 64bit memory manipulations ]
07,075 // ] //r.14+
07,076 // Counter ]
07,077 // }
07,078 // WORDcountAttemptsToPut++;
07,079 // ***** 'P W P' section 2 ]

```



```

07,080 // *****
07,081     }
07,082     else // RW empty - go MP
07,083     {
07,084         memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
07,085         PseudoLinkedPointer = PseudoLinkedPointerNEW;
07,086     }
07,087     // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,088     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8; //MP
07,089     // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,090     if (StackPtr > 8192*3-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
07,091     BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
07,092     BSTstack[StackPtr] = 8; ++StackPtr; //LPOffset=0;MPOffset=8;RPOffset=16;
07,093     // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,094     //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,095     //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
07,096     // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,097     // [ //r.14+
07,098     memcpy( &PseudoLinkedPointer_64, &LEAF[8], 8 );
07,099     // ] //r.14+
07,100     }
07,101     }
07,102     else { FoundInLinkedList = 1; // wrd is LW
07,103     // Counter [
07,104         if (BSTorBtree == 2) {
07,105             fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,106         }
07,107         //memcpy( &CounterOccurrences, &FourGramL[(KeySize+1+4)-4], 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
07,108         //if (CounterOccurrences<999999999) CounterOccurrences++; // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
07,109         //memcpy( &FourGramL[(KeySize+1+4)-4], &CounterOccurrences, 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
07,110         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,111         //fwrite(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
07,112         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,113         // [ //r.14+
07,114         memcpy( &LEAF[8 + 8 + 8], &FourGramL[0], (KeySize+1+8) );
07,115         if (BSTorBtree == 2) {
07,116             fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
07,117         } else { // ##### 64bit memory manipulations [
07,118             memcpy( (char *)&PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
07,119             // ##### 64bit memory manipulations ]
07,120         }
07,121         // ] //r.14+
07,122         // Counter ]
07,123         WORDcountAttemptsToPut++;
07,124     // ***** 'P W P' section ]
07,125         } // while
07,126         WORDcountAttemptsToPut--; // - 1 due to BST way of counting i.e. direct hash hit is not counted only successors
07,127     }
07,128 // 1] Search ]
07,129 if (FoundInLinkedList == 0) NotFoundKeys++; //r.18
07,130 if (FoundInLinkedList == 1) FoundKeys++; //r.18
07,131 if (FoundInLinkedList == 2) NotFoundKeys++; //r.18
07,132
07,133 if (FoundInLinkedList == 0)
07,134 {
07,135 /*
07,136 // ===== [ The whole section/sub-fragment 2 is commented due to great time differences for Internal_vs_External memory accesses - it is far more
cheap to have the STACK overhead (moved to sub-fragment 1) ] ===== [

```

```

07,137 // 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search [
07,138 //   'TracingSearch' is the same as 'Search' except that adds the trail in my simulated stack,
07,139 //   the goal is not to waste time in 'Search' by dealing with no needed trail in case of not 'Insert'.
07,140 //   Simulated stack contains pairs of 'Address of ParentLEAF' + 'Offset of ParentPointer in ParentLEAF i.e. 0 for LP, 4 for MP, 8 for RP'.
07,141 //   'Offset ...' saves unnecessary comparisons of NEWword which after splitting goes up.
07,142 //   memcpy( &PseudoLinkedPointer, BufStart+Slot, 4 );
07,143 //   StackPtr = 0;
07,144 //   while (PseudoLinkedPointer != 0)
07,145 //   {
07,146 // ***** 'P W P' section [
07,147 // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4) != 0 )
07,148 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+wordlen) != 0 )
07,149 // here ALWAYS LW exists: no need for existence check - line below
07,150 // if ( *(char *) (PseudoLinkedPointer+4+4) != 0 )
07,151 // if (memcmp(PseudoLinkedPointer+4+4,word,wordlen) > 0) // go LP
07,152 // { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 0, 4 ); //LP
07,153 // if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
07,154 // BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
07,155 // BSTstack[StackPtr] = 0; ++StackPtr; //LPoffset=0;MPoffset=4;RPOffset=8;
07,156 //   PseudoLinkedPointer = PseudoLinkedPointerNEW;
07,157 // }
07,158 // else if (memcmp(PseudoLinkedPointer+4+4,word,wordlen) < 0) // go RP or MP
07,159 // { // RW existence check - line below:
07,160 //   if ( *(char *) (PseudoLinkedPointer+4+4+wordlen) != 0 ) // RW exists
07,161 //   { // Here all 'P W P' section is repeated; the way of handling case when dynamic number of words in leaf
07,162 // *****
07,163 // ***** 'P W P' section 2 [
07,164 // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4) != 0 )
07,165 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+wordlen) != 0 )
07,166 // here ALWAYS RW exists: no need for existence check - line below
07,167 // if ( *(char *) (PseudoLinkedPointer+4+4+wordlen) != 0 )
07,168 // if (memcmp(PseudoLinkedPointer+4+4+wordlen,word,wordlen) > 0) // go MP
07,169 // { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
07,170 // if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
07,171 // BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
07,172 // BSTstack[StackPtr] = 4; ++StackPtr; //LPoffset=0;MPoffset=4;RPOffset=8;
07,173 //   PseudoLinkedPointer = PseudoLinkedPointerNEW;
07,174 // }
07,175 // else if (memcmp(PseudoLinkedPointer+4+4+wordlen,word,wordlen) < 0) // go RP
07,176 // { // No ?W after RW - go RP
07,177 //   memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4 + 4, 4 ); //BP
07,178 // if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
07,179 // BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
07,180 // BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=4;RPOffset=8;
07,181 //   PseudoLinkedPointer = PseudoLinkedPointerNEW;
07,182 // }
07,183 // else FoundInLinkedList = 1; // wrd is RW
07,184 // ***** 'P W P' section 2 ]
07,185 // *****
07,186 //   }
07,187 //   else // RW empty - go MP
07,188 //   { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
07,189 //   if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
07,190 //   BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
07,191 //   BSTstack[StackPtr] = 4; ++StackPtr; //LPoffset=0;MPoffset=4;RPOffset=8;
07,192 //   PseudoLinkedPointer = PseudoLinkedPointerNEW;
07,193 //   }
07,194 // }

```

```

07,195         else FoundInLinkedList = 1; // wrd is LW
07,196 // ***** 'P W P' section ]
07,197         } // while
07,198 // 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search ]
07,199 // ===== [ The whole section/sub-fragment 2 is commented due to great time differences for Internal_vs_External memory accesses - it is far more
cheap to have the STACK overhead (moved to sub-fragment 1) ] ===== ]
07,200 */
07,201
07,202 // 3] Insert Iterative [
07,203 //     There are total 4 situations:
07,204 //     Case #1: Outer NODE(including ROOT) [ ][ ][ ][LW][ ]
07,205 //     Case #2: Outer NODE(including ROOT) [ ][ ][ ][LW][RW] Split Occurs -----
07,206 //     Case #3: ROOT [LP][MP][ ][LW][ ] | 'wrUP' (wrklen bytes)
07,207 //     Case #4: Inner NODE(including ROOT) [LP][MP][RP][LW][RW] Split Occurs --- | &
07,208 // | | 'PseudoLinkedPointerNEW' (ptr to NEW LEAF)
07,209 // There are total 2 situations for PARENT LEAF: <----- ARE GOING UP
07,210 // Case #3: [LP][MP][ ][LW][ ]
07,211 // Case #4: [LP][MP][RP][LW][RW] Split Occurs
07,212
07,213 // ~ First deal alonely with the OUTER NODE(LEAF) where Search stopped i.e Case #1 & Case #2:
07,214     POffsetInLEAF = BSTstack[--StackPtr];
07,215     PseudoLinkedPointer_64 = BSTstack[--StackPtr];
07,216 // NOTE: ONE LEAF IS FULL ONLY WHEN LAST CELL FOR KEY(here RW) EXISTS!
07,217 // RW: existence check if ( *(char *)(&PseudoLinkedPointer+4*4+4*wrklen) != 0 )
07,218 //if ( *(char *)(&PseudoLinkedPointer+4*4+4*wrklen) != 0 ) // If LEAF is full: Case #2
07,219     // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,220     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1*4); //RW
07,221     //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,222     //fread(&SomeByte, 1, 1, fp_outRG);
07,223     // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,224     // [ //r.14+
07,225     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
07,226     if (BSTorBtree == 2) {
07,227         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,228         fread(&LEAF[0], 8*8+2*(KeySize+1*8), 1, fp_outRG); Total_fread++;
07,229     } else { // ##### 64bit memory manipulations [
07,230         memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64, 8*8+2*(KeySize+1*8) );
07,231     } // ##### 64bit memory manipulations ]
07,232     memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (KeySize+1*8)], 1 );
07,233     // ] //r.14+
07,234     if (SomeByte != 0) // RW exists
07,235     { SplitOccured = 1; WORDcountDistinct++;
07,236       // Allocate NEW LEAF:
07,237       // if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrklen + 4 + 4 + 4 < GRMBL FoolAgain((int)wrklen) ) // +4 more for BST instead of LL; + more(see
LEAF)
07,238       // {
07,239       //     memcpy( &PseudoLinkedPointerNEW, &bufend[LetterOffset], 4 );
07,240       //     bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
07,241       //     bufend[LetterOffset] = bufend[LetterOffset] + 2*wrklen;
07,242       //     if (MAXusedBuffer[wrklen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrklen] = (unsigned long)(bufend[LetterOffset] -
BufStart);}
07,243       // }
07,244       // BUG IN R.17 and prior - the next 'if' is only for RAM (y/Y) not for SSD (z/Z):
07,245       if (BSTorBtree == 2) { //r.18
07,246         BUGGYoffset = (BufEnd_64-(0*14));
07,247     } else { // ##### 64bit memory manipulations [
07,248         BUGGYoffset = (BufEnd_64-(unsigned long long)pointerflush_64);
07,249     } // ##### 64bit memory manipulations ]

```

```

07,250         if( 8 + 8 + 8 + 2*(KeySize+1+8) < size_in64_L14 - BUGGYoffset ) // the longest wrdlen is LongestLineInclusive but actual is LongestLineInclusive(+ CR char)
07,251             {
07,252                 PseudoLinkedPointerNEW_64 = BufEnd_64;
07,253                 BufEnd_64 = BufEnd_64 + 8 + 8 + 8;
07,254                 BufEnd_64 = BufEnd_64 + 2*(KeySize+1+8);
07,255             }
07,256         else
07,257             { printf( "\nLeprechaun: Failure! Increment 'Memory for B-trees'!\n" );
07,258 #ifdef LITE
07,259 #else
07,260             fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, 11TOaDigits, 10), _ui64toaKAZEcomma((unsigned long
long)WORDcountDistinct, 11TOaDigits2, 10) );
07,261             fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14, 11TOaDigits, 10) );
07,262             fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11TOaDigits, 10) );
07,263             fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
07,264             fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for B-trees'!\n\n");
07,265 #endif
07,266             exit( 7 );
07,267         }
07,268         if (POffsetInLEAF == 0) // wrd < LW
07,269         {
07,270             memcpy( wrdUP, PseudoLinkedPointer+4+4+4, wrdlen ); // LW up
07,271             memcpy( PseudoLinkedPointer+4+4+4, wrd, wrdlen ); // wrd go to OLD LEAF
07,272             memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
07,273             *(char *)(&PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
07,274             // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,275             //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
07,276             //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,277             //fread(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
07,278             //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,279             //fwrite(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
07,280             //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
07,281             //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,282             //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
07,283             //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
07,284             //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,285             //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
07,286             //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
07,287             //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,288             //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
07,289             // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,290             // [ //r.14+
07,291             memcpy( &wrdUP[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
07,292             memcpy( &LEAF[8 + 8 + 8], &wrd[0], (KeySize+1+8) );
07,293             memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
07,294             // Here reordering (of writing wrdAUX) is needed to avoid seek the position NEW and stupidly to seek again OLD/current position!
07,295             memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
07,296             if (BSTorBtree == 2) {
07,297                 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,298                 fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
07,299             } else { // ##### 64bit memory manipulations [
07,300             memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
07,301             // ##### 64bit memory manipulations ]
07,302             PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
07,303             if (BSTorBtree == 2) {
07,304                 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,305                 fwrite(&wrdAUX[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
07,306             } else { // ##### 64bit memory manipulations [

```

```

07,307         memcpy( (char *)PseudoLinkedPointerAUX_64, &wrdAUX[0], (KeySize+1+8) );
07,308             } // ##### 64bit memory manipulations ]
07,309         // ] //r.14+
07,310     }
07,311     if (POffsetInLEAF == 8) // LW < wrd < RW
07,312     {
07,313         memcpy( wrdUP, wrd, wrdlen ); // wrd up
07,314         memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
07,315         *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
07,316         memcpy( wrdUP, wrd, (KeySize+1+8) ); // wrd up
07,317         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,318         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
07,319         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,320         //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
07,321         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
07,322         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,323         //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
07,324         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
07,325         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,326         //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
07,327         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,328         // [ //r.14+
07,329         memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
07,330         // Here reordering (of writing wrdAUX) is needed to avoid seek the position NEW and stupidly to seek again OLD/current position!
07,331         memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
07,332         if (BSTorBtree == 2) {
07,333             fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,334             fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
07,335         } else { // ##### 64bit memory manipulations [
07,336             memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
07,337             // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,338             PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
07,339             if (BSTorBtree == 2) {
07,340                 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,341                 fwrite(&wrdAUX[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
07,342             } else { // ##### 64bit memory manipulations [
07,343                 memcpy( (char *)PseudoLinkedPointerAUX_64, &wrdAUX[0], (KeySize+1+8) );
07,344                 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,345                 // [ //r.14+
07,346             }
07,347         if (POffsetInLEAF == 16) // wrd > RW
07,348         {
07,349             memcpy( wrdUP, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW up
07,350             *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
07,351             memcpy( PseudoLinkedPointerNEW+4+4+4, wrd, wrdlen ); // wrd go to NEW LEAF
07,352             // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,353             //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
07,354             //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,355             //fread(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
07,356             //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
07,357             //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,358             //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
07,359             //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
07,360             //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,361             //fwrite(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
07,362             // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,363             // [ //r.14+
07,364             memcpy( &wrdUP[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );

```

```

07,365         memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
07,366             if (BSTorBtree == 2) {
07,367                 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,368                 fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
07,369             } else { // ##### 64bit memory manipulations [
07,370                 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
07,371             } // ##### 64bit memory manipulations ]
07,372         // ] //r.14+
07,373         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one! Here NO need!
07,374         PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
07,375             if (BSTorBtree == 2) {
07,376                 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,377                 fwrite(&wrd[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
07,378             } else { // ##### 64bit memory manipulations [
07,379                 memcpy( (char *)PseudoLinkedPointerAUX_64, &wrd[0], (KeySize+1+8) );
07,380             } // ##### 64bit memory manipulations ]
07,381         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one! Here NO need!
07,382     }
07,383 }
07,384 else // If LEAF is not full: Case #1
07,385 { SplitOccured = 0; WORDcountDistinct++;
07,386     if (POffsetInLEAF == 0) // wrd < [LW][ ] so [LW][ ] -> [ ][LW] -> [wrd][LW]
07,387     {
07,388         //         memcpy( PseudoLinkedPointer+4+4+4+wrdlen, PseudoLinkedPointer+4+4+4, wrdlen );
07,389         //         memcpy( PseudoLinkedPointer+4+4+4, wrd, wrdlen );
07,390         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,391         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
07,392         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,393         //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
07,394         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
07,395         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,396         //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
07,397         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
07,398         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,399         //fwrite(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
07,400         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,401         // [ //r.14+
07,402         memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
07,403         memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &wrdAUX[0], (KeySize+1+8) );
07,404         memcpy( &LEAF[8 + 8 + 8], &wrd[0], (KeySize+1+8) );
07,405             if (BSTorBtree == 2) {
07,406                 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,407                 fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
07,408             } else { // ##### 64bit memory manipulations [
07,409                 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
07,410             } // ##### 64bit memory manipulations ]
07,411         // ] //r.14+
07,412     }
07,413 }
07,414 if (POffsetInLEAF == 8) // wrd > [LW][ ] so [LW][ ] -> [LW][wrd]
07,415 {
07,416     //         memcpy( PseudoLinkedPointer+4+4+4+wrdlen, wrd, wrdlen );
07,417     // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one! Here NO need!
07,418     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (KeySize+1+8);
07,419     if (BSTorBtree == 2) {
07,420         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,421         fwrite(&wrd[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
07,422     } else { // ##### 64bit memory manipulations [

```

```

07,423         memcpy( (char *)PseudoLinkedPointerAUX_64, &wrd[0], (KeySize+1+8) );
07,424         } // ##### 64bit memory manipulations ]
07,425         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one! Here NO need!
07,426     }
07,427 }
07,428
07,429 if (SplitOccured != 0)
07,430 {
07,431 // ~ Second deal with the INNER NODE(S) i.e Case #3 & Case #4:
07,432     while (StackPtr != 0 || SplitOccured != 0)
07,433     {
07,434         // 'PseudoLinkedPointerNEW' is new LEAF to be inserted
07,435         // 'wrdUP' is NEW word to be inserted
07,436         if (StackPtr != 0)
07,437         {
07,438             POffsetInLEAF = BSTstack[--StackPtr];
07,439             PseudoLinkedPointer_64 = BSTstack[--StackPtr];
07,440 //if ( *(char *) (PseudoLinkedPointer+4+4+4+4+4+4) != 0 ) // If LEAF is full: Case #4
07,441             // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,442             //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
07,443             //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,444             //fread(&SomeByte, 1, 1, fp_outRG);
07,445             // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,446             // [ //r.14+
07,447             PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
07,448             if (BSTorBtree == 2) {
07,449                 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,450                 fread(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fread++;
07,451             } else { // ##### 64bit memory manipulations [
07,452                 memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64, 8+8+8+2*(KeySize+1+8) );
07,453             } // ##### 64bit memory manipulations ]
07,454             memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (KeySize+1+8)], 1 );
07,455             // ] //r.14+
07,456             if (SomeByte != 0 ) // RW exists
07,457         { SplitOccured = 1;
07,458             //         memcpy( wrdUPold, wrdUP, wrdlen ); // LW up
07,459             //         PseudoLinkedPointerNEWold = PseudoLinkedPointerNEW;
07,460             //         memcpy( wrdUPold, wrdUP, (KeySize+1+8) );
07,461             //         PseudoLinkedPointerNEWold_64 = PseudoLinkedPointerNEW_64;
07,462             // ALlocate NEW LEAF:
07,463             //         if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrdlen + 4 + 4 + 4 < GRMBLFoolAgain[(int)wrdlen] ) // +4 more for BST instead of LL; + more(see
LEAF)
07,464             //         {
07,465             //             memcpy( &PseudoLinkedPointerNEW, &bufend[LetterOffset], 4 );
07,466             //             bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
07,467             //             bufend[LetterOffset] = bufend[LetterOffset] + 2*wrdlen;
07,468             //             if (MAXusedBuffer[wrdlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrdlen] = (unsigned long)(bufend[LetterOffset] -
BufStart);}
07,469             //         }
07,470             // BUG IN R.17 and prior - the next 'if' is only for RAM (y/Y) not for SSD (z/Z):
07,471             if (BSTorBtree == 2) { //r.18
07,472                 BUGGYoffset = (BufEnd_64-(0+14));
07,473             } else { // ##### 64bit memory manipulations [
07,474                 BUGGYoffset = (BufEnd_64-(unsigned long long)pointerflush_64);
07,475             } // ##### 64bit memory manipulations ]
07,476             if( 8 + 8 + 8 + 2*(KeySize+1+8) < size_in64_L14 - BUGGYoffset ) // the longest wrdlen is LongestLineInclusive but actual is LongestLineInclusive(+ CR char)
07,477             {
07,478                 PseudoLinkedPointerNEW_64 = BufEnd_64;

```

```

07,479         BufEnd_64 = BufEnd_64 + 8 + 8 + 8;
07,480         BufEnd_64 = BufEnd_64 + 2*(KeySize+1+8);
07,481         // [ //r.14+
07,482         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64;
07,483         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,484         //fread(&LEAFNEW[0], 8+8+2*(LongestLineInclusive+1+4), 1, fp_outRG);
07,485         // In fact above three lines are slow, the only need is ZEROed LEAFNEW.
07,486         memset(&LEAFNEW[0], 0, 8+8+2*(KeySize+1+8));
07,487         // ] //r.14+
07,488     }
07,489     else
07,490     { printf( "\nLeprechaun: Failure! Increment 'Memory for B-trees'!\n" );
07,491 #ifdef LITE
07,492 #else
07,493         fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, 11TOaDigits, 10), _ui64toaKAZEcomma((unsigned long)WORDcountDistinct, 11TOaDigits2, 10) );
07,494         fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14, 11TOaDigits, 10) );
07,495         fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11TOaDigits, 10) );
07,496         fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
07,497         fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for B-trees'!\n\n" );
07,498 #endif
07,499         exit( 7 );
07,500     }
07,501     if (POffsetInLEAF == 0) // wrdUPold < LW
07,502     {
07,503         // memcpy( wrdUP, PseudoLinkedPointer+4+4+4, wrdlen ); // LW up
07,504         // memcpy( PseudoLinkedPointer+4+4+4, wrdUPold, wrdlen ); // wrdUPold go to OLD LEAF
07,505         // memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
07,506         // *(char *)(&PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
07,507         // // [LP](PseudoLinkedPointerNEWold)[MP][RP](wrdUPold)[LW][RW] -----
07,508         // // pair [LW] PseudoLinkedPointerNEW goes up !
07,509         // // PseudoLinkedPointer: PseudoLinkedPointerNEW: !
07,510         // // [LP](PseudoLinkedPointerNEWold)[](wrdUPold) [MP][RP][][RW] <----
07,511         // // no need to put zero in RP because logic is based on words existence:
07,512         // memcpy( PseudoLinkedPointerNEW+0, PseudoLinkedPointer+4, 4 );
07,513         // memcpy( PseudoLinkedPointerNEW+4, PseudoLinkedPointer+8, 4 );
07,514         // memcpy( PseudoLinkedPointer+4, &PseudoLinkedPointerNEWold, 4 );
07,515         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,516         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
07,517         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,518         //fread(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
07,519         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,520         //fwrite(&wrdUPold[0], (LongestLineInclusive+1+4), 1, fp_outRG);
07,521         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
07,522         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,523         //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
07,524         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
07,525         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,526         //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
07,527         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
07,528         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,529         //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
07,530         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
07,531         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,532         //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
07,533         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 0;
07,534         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,535         //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);

```



```

07,536 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
07,537 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,538 //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
07,539 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8;
07,540 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,541 //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
07,542 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
07,543 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,544 //fwrite(&PseudoLinkedPointerNEWold_64, 8, 1, fp_outRG);
07,545 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,546 // [ //r.14+
07,547 memcpy( &wrdUP[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
07,548 memcpy( &LEAF[8 + 8 + 8], &wrdUPold[0], (KeySize+1+8) );
07,549 memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
07,550 memcpy( &LEAFNEW[8 + 8 + 8], &wrdAUX[0], (KeySize+1+8) );
07,551 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
07,552 memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[8], 8 );
07,553 memcpy( &LEAFNEW[0], &PseudoLinkedPointerAUXdumbo_64, 8 );
07,554 memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[16], 8 );
07,555 memcpy( &LEAFNEW[8], &PseudoLinkedPointerAUXdumbo_64, 8 );
07,556 memcpy( &LEAF[8], &PseudoLinkedPointerNEWold_64, 8 );
07,557 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
07,558 if (BSTorBtree == 2) {
07,559 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,560 fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
07,561 } else { // ##### 64bit memory manipulations [
07,562 memcpy( (char *)&PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
07,563 } // ##### 64bit memory manipulations ]
07,564 PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64;
07,565 if (BSTorBtree == 2) {
07,566 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,567 fwrite(&LEAFNEW[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
07,568 } else { // ##### 64bit memory manipulations [
07,569 memcpy( (char *)&PseudoLinkedPointerAUX_64, &LEAFNEW[0], 8+8+8+2*(KeySize+1+8) );
07,570 } // ##### 64bit memory manipulations ]
07,571 // ] //r.14+
07,572 }
07,573 if (POffsetInLEAF == 8) // LW < wrdUPold < RW
07,574 {
07,575 // memcpy( wrdUP, wrdUPold, wrdlen ); // wrdUPold up
07,576 // memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
07,577 // *(char *)&(PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
07,578 // [LP][MP](PseudoLinkedPointerNEWold)[RP][LW](wrdUPold)[RW] -----
07,579 // // pair [wrdUPold] PseudoLinkedPointerNEW goes up !
07,580 // // PseudoLinkedPointer: PseudoLinkedPointerNEW: !
07,581 // // [LP][MP][][LW] (PseudoLinkedPointerNEWold)[RP][][RW] <----
07,582 // // no need to put zero in RP because logic is based on words existence:
07,583 // memcpy( PseudoLinkedPointerNEW+0, &PseudoLinkedPointerNEWold, 4 );
07,584 // memcpy( PseudoLinkedPointerNEW+4, PseudoLinkedPointer+8, 4 );
07,585 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,586 // memcpy( wrdUP, wrdUPold, (KeySize+1+8) );
07,587 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
07,588 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,589 //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
07,590 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
07,591 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,592 //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
07,593 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);

```

```

07,594         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,595         //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
07,596         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 0;
07,597         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,598         //fwrite(&PseudoLinkedPointerNEWold_64, 8, 1, fp_outRG);
07,599         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
07,600         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,601         //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
07,602         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8;
07,603         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,604         //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
07,605         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,606         // [ //r.14+
07,607         memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
07,608         memcpy( &LEAFNEW[8 + 8 + 8], &wrdAUX[0], (KeySize+1+8) );
07,609         memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
07,610         memcpy( &LEAFNEW[0], &PseudoLinkedPointerNEWold_64, 8 );
07,611         memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[16], 8 );
07,612         memcpy( &LEAFNEW[8], &PseudoLinkedPointerAUXdumbo_64, 8 );
07,613         PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
07,614         if (BSTorBtree == 2) {
07,615             fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,616             fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
07,617             } else { // ##### 64bit memory manipulations [
07,618             memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
07,619             } // ##### 64bit memory manipulations ]
07,620         PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64;
07,621         if (BSTorBtree == 2) {
07,622             fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,623             fwrite(&LEAFNEW[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
07,624             } else { // ##### 64bit memory manipulations [
07,625             memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAFNEW[0], 8+8+8+2*(KeySize+1+8) );
07,626             } // ##### 64bit memory manipulations ]
07,627         // ] //r.14+
07,628     }
07,629     if (PoffsetInLEAF == 16) // wrdUPold > RW
07,630     {
07,631         //         memcpy( wrdUP, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW up
07,632         //         *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
07,633         //         memcpy( PseudoLinkedPointerNEW+4+4+4, wrdUPold, wrdlen ); // wrdUPold go to NEW LEAF
07,634         //         // [LP][MP][RP] (PseudoLinkedPointerNEWold)[LW][RW] (wrdUPold) -----
07,635         //         // pair [RW] PseudoLinkedPointerNEW goes up !
07,636         //         // PseudoLinkedPointer: PseudoLinkedPointerNEW: !
07,637         //         // [LP][MP][LW] [RP] (PseudoLinkedPointerNEWold)[] (wrdUPold) <---
07,638         //         // no need to put zero in RP because logic is based on words existence:
07,639         //         memcpy( PseudoLinkedPointerNEW+0, PseudoLinkedPointer+8, 4 );
07,640         //         memcpy( PseudoLinkedPointerNEW+4, &PseudoLinkedPointerNEWold, 4 );
07,641         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,642         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
07,643         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,644         //fread(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
07,645         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
07,646         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,647         //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
07,648         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
07,649         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,650         //fwrite(&wrdUPold[0], (LongestLineInclusive+1+4), 1, fp_outRG);
07,651         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;

```

```

07,652 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,653 //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
07,654 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 0;
07,655 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,656 //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
07,657 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8;
07,658 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,659 //fwrite(&PseudoLinkedPointerNEWold_64, 8, 1, fp_outRG);
07,660 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,661 // [ //r.14+
07,662 memcpy( &wrdUP[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
07,663 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
07,664 memcpy( &LEAFNEW[8 + 8 + 8], &wrdUPold[0], (KeySize+1+8) );
07,665 memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[16], 8 );
07,666 memcpy( &LEAFNEW[0], &PseudoLinkedPointerAUXdumbo_64, 8 );
07,667 memcpy( &LEAFNEW[8], &PseudoLinkedPointerNEWold_64, 8 );
07,668 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
07,669 if (BSTorBtree == 2) {
07,670 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,671 fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
07,672 } else { // ##### 64bit memory manipulations [
07,673 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
07,674 } // ##### 64bit memory manipulations ]
07,675 PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64;
07,676 if (BSTorBtree == 2) {
07,677 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,678 fwrite(&LEAFNEW[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
07,679 } else { // ##### 64bit memory manipulations [
07,680 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAFNEW[0], 8+8+2*(KeySize+1+8) );
07,681 } // ##### 64bit memory manipulations ]
07,682 // ] //r.14+
07,683 }
07,684 }
07,685 else // If LEAF is not full: Case #3
07,686 { SplitOccured = 0;
07,687 if (POffsetInLEAF == 0) // wrdUP < [LW][ ] so [LW][ ] -> [ ][LW] -> [wrdUP][LW]
07,688 {
07,689 // memcpy( PseudoLinkedPointer+4+4+4*wrdlen, PseudoLinkedPointer+4+4+4, wrdlen );
07,690 // memcpy( PseudoLinkedPointer+4+4+4, wrdUP, wrdlen );
07,691 // // [LP][MP][ ] -> [LP][ ][MP] -> [LP][np][MP]
07,692 // memcpy( PseudoLinkedPointer+8, PseudoLinkedPointer+4, 4 );
07,693 // memcpy( PseudoLinkedPointer+4, &PseudoLinkedPointerNEW, 4 );
07,694 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,695 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
07,696 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,697 //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
07,698 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
07,699 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,700 //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
07,701 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
07,702 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,703 //fwrite(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
07,704 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
07,705 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,706 //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
07,707 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
07,708 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,709 //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);

```

```

07,710 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
07,711 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,712 //fwrite(&PseudoLinkedPointerNEW_64, 8, 1, fp_outRG);
07,713 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,714 // [ //r.14+
07,715 memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
07,716 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &wrdAUX[0], (KeySize+1+8) );
07,717 memcpy( &LEAF[8 + 8 + 8], &wrdUP[0], (KeySize+1+8) );
07,718 memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[8], 8 );
07,719 memcpy( &LEAF[8 + 8], &PseudoLinkedPointerAUXdumbo_64, 8 );
07,720 memcpy( &LEAF[8], &PseudoLinkedPointerNEW_64, 8 );
07,721 if (BSTorBtree == 2) {
07,722 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,723 fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
07,724 } else { // ##### 64bit memory manipulations [
07,725 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
07,726 } // ##### 64bit memory manipulations ]
07,727 // ] //r.14+
07,728 }
07,729 if (PoffsetInLEAF == 8) // wrdUP > [LW][ ] so [LW][ ] -> [LW][wrdUP]
07,730 {
07,731 // memcpy( PseudoLinkedPointer+4+4+4*wrdlen, wrdUP, wrdlen );
07,732 // // [LP][MP][ ] -> [LP][MP][np]
07,733 // memcpy( PseudoLinkedPointer+8, &PseudoLinkedPointerNEW, 4 );
07,734 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,735 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
07,736 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,737 //fwrite(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
07,738 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
07,739 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,740 //fwrite(&PseudoLinkedPointerNEW_64, 8, 1, fp_outRG);
07,741 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,742 // [ //r.14+
07,743 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &wrdUP[0], (KeySize+1+8) );
07,744 memcpy( &LEAF[8 + 8], &PseudoLinkedPointerNEW_64, 8 );
07,745 if (BSTorBtree == 2) {
07,746 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,747 fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
07,748 } else { // ##### 64bit memory manipulations [
07,749 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
07,750 } // ##### 64bit memory manipulations ]
07,751 // ] //r.14+
07,752 }
07,753 break;
07,754 }
07,755 }
07,756 else // Empty stack means ROOT and more over ROOT is already splitted(Case #4 is off)
07,757 {
07,758 // If LEAF is not full: Case #3
07,759 // THIS IS WHERE A NEW(SECOND) LEAF 'PseudoLinkedPointerROOT' must be allocated:
07,760 // if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrdlen + 4 + 4 + 4 < GRMBLFoolAgain((int)wrdlen) ) // +4 more for BST instead of LL; + more(see
LEAF)
07,761 // {
07,762 // memcpy( &PseudoLinkedPointerROOT, &bufend[LetterOffset], 4 );
07,763 // bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
07,764 // memcpy( bufend[LetterOffset], wrdUP, wrdlen );
07,765 // bufend[LetterOffset] = bufend[LetterOffset] + 2*wrdlen;
07,766 // if (MAXusedBuffer[wrdlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrdlen] = (unsigned long)(bufend[LetterOffset] -

```

```

BufStart);}
07,767 //          }
07,768          // BUG IN R.17 and prior - the next 'if' is only for RAM (y/Y) not for SSD (z/Z):
07,769          if (BSTorBtree == 2) { //r.18
07,770      BUGGYoffset = (BufEnd_64-(0+14));
07,771          } else { // ##### 64bit memory manipulations [
07,772      BUGGYoffset = (BufEnd_64-(unsigned long long)pointerflush_64);
07,773          } // ##### 64bit memory manipulations ]
07,774      if( 8 + 8 + 8 + 2*(KeySize+1+8) < size_in64_L14 - BUGGYoffset ) // the longest wrdlen is LongestLineInclusive but actual is LongestLineInclusive(+ CR char)
07,775      {
07,776          PseudoLinkedPointerROOT_64 = BufEnd_64;
07,777          BufEnd_64 = BufEnd_64 + 8 + 8 + 8;
07,778          BufEnd_64 = BufEnd_64 + 2*(KeySize+1+8);
07,779          PseudoLinkedPointerAUX_64 = PseudoLinkedPointerROOT_64 + 8 + 8 + 8;
07,780          if (BSTorBtree == 2) {
07,781              fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,782              fwrite(&wrdUP[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
07,783          } else { // ##### 64bit memory manipulations [
07,784              memcpy( (char *)PseudoLinkedPointerAUX_64, &wrdUP[0], (KeySize+1+8) );
07,785          } // ##### 64bit memory manipulations ]
07,786      }
07,787      else
07,788      { printf( "\nLeprechaun: Failure! Increment 'Memory for B-trees'!\n" );
07,789      #ifdef LITE
07,790      #else
07,791          fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, llTOaDigits, 10), _ui64toaKAZEcomma((unsigned long
long)WORDcountDistinct, llTOaDigits2, 10) );
07,792          fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14, llTOaDigits, 10) );
07,793          fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, llTOaDigits, 10) );
07,794          fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
07,795          fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for B-trees'!\n\n");
07,796      #endif
07,797          exit( 7 );
07,798      }
07,799      // Here          -- 'PseudoLinkedPointerROOT' --
07,800      //          |          (wrdUP)          |
07,801      // 'PseudoLinkedPointer' <--          --> 'PseudoLinkedPointerNEW'
07,802      //          (LW)          (RW)
07,803      //      memcpy( PseudoLinkedPointerROOT, &PseudoLinkedPointer, 4 ); // LP
07,804      //      memcpy( PseudoLinkedPointerROOT+4, &PseudoLinkedPointerNEW, 4 ); // MP
07,805      //      // Here must NEW ROOT be updated i.e. HASH table(SLOT) must point it:
07,806      //      memcpy( BufStart+Slot, &PseudoLinkedPointerROOT, 4 );
07,807      PseudoLinkedPointerAUX_64 = PseudoLinkedPointerROOT_64;
07,808      if (BSTorBtree == 2) {
07,809          fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,810          fwrite(&PseudoLinkedPointer_64, 8, 1, fp_outRG); Total_fwrite++;
07,811          } else { // ##### 64bit memory manipulations [
07,812              memcpy( (char *)PseudoLinkedPointerAUX_64, &PseudoLinkedPointer_64, 8 );
07,813          } // ##### 64bit memory manipulations ]
07,814      PseudoLinkedPointerAUX_64 = PseudoLinkedPointerROOT_64 + 8;
07,815      if (BSTorBtree == 2) {
07,816          fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,817          fwrite(&PseudoLinkedPointerNEW_64, 8, 1, fp_outRG); Total_fwrite++;
07,818          } else { // ##### 64bit memory manipulations [
07,819              memcpy( (char *)PseudoLinkedPointerAUX_64, &PseudoLinkedPointerNEW_64, 8 );
07,820          } // ##### 64bit memory manipulations ]
07,821      memcpy( BufStart+Slot, &PseudoLinkedPointerROOT_64, 8 );
07,822      break; //because it is ROOT without split

```

```

07,823     }
07,824     } // while
07,825 } //if (SplitOccured != 0)
07,826 // 3] Insert Iterative ]
07,827 } //if (FoundInLinkedList == 0)
07,828 // ##### B-tree order 3 fragment 64bit ]
07,829 } // if (FoundInLinkedList_RAM == 1) { // 2019-Dec-04
07,830
07,831 // Building nonunique 1 of 2 ]
07,832     // Counter ]
07,833
07,834 // Have to restore the LEAF since it is changed in 1of2 and 2of2 sections:
07,835     //memcpy( &LEAFbackup[0], &LEAF[0], 8+8+2*(KeySize+1+8) );
07,836     memcpy( &LEAF[0], &LEAFbackup[0], 8+8+2*(KeySize+1+8) );
07,837
07,838     WORDcountBOTTOM++; //Nakamichi
07,839     WORDcountBOTTOMPerMatchLen++;
07,840 if (*argv[k_FIX] == 'Y' || *argv[k_FIX] == 'Z')
07,841     //if (CounterOccurrences>1) // For Nakamichi
07,842     //     fprintf(fp_out, "%s\t%s\r\n", _ui64toaKAZEzerocomma(CounterOccurrences, 11ToaDigits2, 10)+(26-11), wrd); WORDcountBOTTOM++;
07,843 if (*argv[k_FIX] == 'y' || *argv[k_FIX] == 'z')
07,844 // NO-DUMP: [
07,845     WORDcountBOTTOM++;
07,846 //     fprintf(fp_out, "%s\r\n", wrd); WORDcountBOTTOM++;
07,847 // NO-DUMP: ]
07,848 //fwrite(CRDLa, 2, 1, fp_out);
07,849 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,850 //PseudoLinkedPointerAUX_64DUMP = PseudoLinkedPointer_64DUMP + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
07,851 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64DUMP);
07,852 //fread(&SomeByte, 1, 1, fp_outRG);
07,853 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,854 // [ //r.14+
07,855     memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (KeySize+1+8)], 1 );
07,856 // ] //r.14+
07,857 if (SomeByte != 0 ) // RW exists
07,858 {
07,859     // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,860 //PseudoLinkedPointerAUX_64DUMP = PseudoLinkedPointer_64DUMP + 8 + 8 + 8 + (LongestLineInclusive+1+4);
07,861 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64DUMP);
07,862 //fread(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
07,863 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,864 // [ //r.14+
07,865     memcpy( &wrd[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
07,866 // ] //r.14+
07,867 // Counter [
07,868 //memcpy( &CounterOccurrences, &wrd[(KeySize+1+4)-4], 4 );
07,869 //if (CounterOccurrences<999999999) CounterOccurrences++; // Starting from ZERO! Because when insertion happened there was no setting counter to 1.
07,870 memcpy( &CounterOccurrences, &wrd[(KeySize+1+8)-8], 8 ); // 2019-Dec-26
07,871 FoundInLinkedList_RAM = 0; // Assume not found // 2019-Dec-26
07,872 if (CounterOccurrences>1) FoundInLinkedList_RAM = 1; // 2019-Dec-26
07,873 memset(&wrd[(KeySize+1+8)-8], 0, 8); // 2021-Jan-13, NastyBugFixed, for a long time this nasty bug remained uncrushed, namely not nullifying the
8bytes for 'CounterOccurrences' now serving also as LastSeenOffset, simply forgot to zeroing it :(
07,874
07,875 if (FoundInLinkedList_RAM == 1) TotalNonUnique[MatchLens[jj]]++; // 2020-Jun-09
07,876 // Building nonunique 2 of 2 [
07,877
07,878 #if defined(_NSHA3) || defined(_NPRV) || defined(_NDD) || defined(_NAquaHash) //2020-Nov-12
07,879     memcpy( &wrdlen, &wrd[ 0 ], 1 ); // The keysize is 1..255, pretty enough since my needs are 1..28 (up to SHA3-224). // 2020-Feb-02

```

```

07,880 #else
07,881 wrdlen=MatchLens[jj];
07,882 #endif
07,883
07,884 //memset( &wrd[0], 0, (LongestLineInclusive+1+8) ); //r.18, see below, the old nullifier is commented. // 2019-Dec-26
07,885
07,886 // if (wrdlen > 28) {
07,887 //     FIPS202_SHA3_224((unsigned char *) (SourceBlock+BuildingBlocksSTRIDE), wrdlen, (unsigned char *) SHA328bytes);
07,888 //     wrdlen=28;
07,889 // }
07,890
07,891 memcpy( &wrd[ 0 ], &wrdlen, 1 ); // The keysize is 1..255, pretty enough since my needs are 1..28 (up to SHA3-224).
07,892 //if (memcmp( &wrd[ 0 ], &wrdlen, 1 )) printf ("nwrld != wrdlen\n");
07,893
07,894 // if (wrdlen < 28)
07,895 //     memcpy( &wrd[ 0+1 ], (unsigned char *) (SourceBlock+BuildingBlocksSTRIDE), wrdlen ); // 2019-Dec-26
07,896 // else
07,897 //     memcpy( &wrd[ 0+1 ], (unsigned char *) SHA328bytes, wrdlen );
07,898
07,899 BufStart = pointerflush;
07,900 // Slot = ((wrd[0]-'_')*28*28*28*28 + (wrd[1]-'_')*28*28*28 + (wrd[2]-'_')*28*28 + (wrd[3]-'_')*28 + (wrd[4]-'_'))<<3;
07,901 // Slot = FNV1A_Hash_Jesteress_27bit(wrd, wrdlen)<<3; // Commented since r.14+++ because of passes.
07,902 //Slot = FNV1A_Hash_Jesteress_27bit(wrd +1, wrdlen); WORDcount++;
07,903 Slot = FNV1A_Pippip_Yurii(wrd +1, wrdlen); WORDcount++; // Changed 2019-Nov-28
07,904 Slot = Slot<<3;
07,905 GettingIndexOfArray=jj; // same as above atrocity
07,906 Slot = Slot + (GettingIndexOfArray*(1LL<<HashInBITS_GLOBAL)*8); // CAUTION: 'wrdlen' is halved here, when a new revision comes with 1:1 keysize then change it.
07,907
07,908 //KeySize = LongestLineInclusive; // 'LongestLineInclusive' is for 1-way hashing where different KeySizes can occupy same HASH pool.
07,909 KeySize = wrdlen; //r.18, In case multi-way hashing: either the MAX allowed or the 'wrdlen'.
07,910
07,911 //Slot = 0; // One Tree only!
07,912 memcpy( &PseudoLinkedPointer_64, BufStart+Slot, 8 );
07,913
07,914 if (FoundInLinkedList_RAM == 1) { // 2019-Dec-04
07,915 // ##### B-tree order 3 fragment 64bit [
07,916 //
07,917 // LEAF structure: [LeftPointer][MiddlePointer][RightPointer][LeftWord][RightWord]
07,918 //                  4bytes      4bytes      4bytes      wrdlen      wrdlen
07,919 //                  *          *          *          *          *          <- if *(char *)==0 means the word cell is empty
07,920 // ALL B-tree order 3 fragment consists of 3 sub-fragments:
07,921 // 1] Search 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search 3] Insert Iterative
07,922
07,923 // LEAF_64 structure: [LeftPointer][MiddlePointer][RightPointer][LeftWord] [RightWord]
07,924 //                   8bytes      8bytes      8bytes      LongestLineInclusive+1+4 LongestLineInclusive+1+4
07,925 //                   *          *          *          *          *          <- if *(char *)==0 means the word cell is empty
07,926 // Note: In order to use one fread(and strcmp) a NULL postfix for LeftWord, RightWord i.e. LeftWord_Length=len(LeftWord)+1 a kinda stupid choice ...
07,927 // Note: BufEnd_64 in fact is the first free position after the BUFFER END!
07,928
07,929 // 1] Search [
07,930 //             if (PseudoLinkedPointer_64 == 0) // means EMPTY-SLOT
07,931 //             {
07,932 //                 //if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrdlen + 4 + 4 + 4 < GRMBL FoolAgain[(int)wrdlen] ) // +4 more for BST instead of LL; + more(see
07,933 //                 // {
07,934 //                 memcpy( BufStart+Slot, &bufend[LetterOffset], 4 );
07,935 //                 bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
07,936 //                 memcpy( bufend[LetterOffset], wrd, wrdlen ); WORDcountDistinct++; bufNumberOfWords[LetterOffset]++;

```

```

07,937          //          bufend[LetterOffset] = bufend[LetterOffset] + 2*wrklen;
07,938          //          if (MAXusedBuffer[wrklen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrklen] = (unsigned long)(bufend[LetterOffset] -
BufStart);}
07,939          //          }
07,940          // BUG IN R.17 and prior - the next 'if' is only for RAM (y/Y) not for SSD (z/Z):
07,941          if (BSTorBtree == 2) { //r.18
07,942      BUGGYoffset = (BufEnd_64-(0+14));
07,943          } else { // ##### 64bit memory manipulations [
07,944      BUGGYoffset = (BufEnd_64-(unsigned long long)pointerflush_64);
07,945          } // ##### 64bit memory manipulations ]
07,946      if( 8 + 8 + 8 + 2*(KeySize+1+8) < size_in64_L14 - BUGGYoffset ) // the longest wrklen is LongestLineInclusive but actual is LongestLineInclusive(+ CR char)
07,947          {
07,948              memcpy( BufStart+Slot, &BufEnd_64, 8 );
07,949              BufEnd_64 = BufEnd_64 + 8 + 8 + 8;
07,950              if (BSTorBtree == 2) {
07,951                  fsetpos(fp_outRG, (const fpos_t *)&BufEnd_64);
07,952                  //fwrite(wrd, wrklen, 1, fp_outRG); //GRMBL! r.18
07,953                  fwrite(wrd, (KeySize+1+8), 1, fp_outRG); //GRMBL! r.18
07,954                  WORDcountDistinct++; Total_fwrite++;
07,955                  //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
07,956                  // r.14++ The above line was commented because the pool is already ZEROed.
07,957                  } else { // ##### 64bit memory manipulations [
07,958                      //memcpy( (char *)BufEnd_64, wrd, wrklen ); //GRMBL! r.18
07,959                      memcpy( (char *)BufEnd_64, wrd, (KeySize+1+8) ); //GRMBL! r.18
07,960                      WORDcountDistinct++;
07,961                      } // ##### 64bit memory manipulations ]
07,962                      BufEnd_64 = BufEnd_64 + 2*(KeySize+1+8);
07,963                      //fsetpos(fp_outRG, &BufEnd_64);
07,964                      }
07,965      else
07,966          { printf( "\nLeprechaun: Failure! Increment 'Memory for B-trees'!\n" );
07,967      #ifdef LITE
07,968      #else
07,969      fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, llToaDigits, 10), _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct,
llToaDigits2, 10) );
07,970      fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14, llToaDigits, 10) );
07,971      fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, llToaDigits, 10) );
07,972      fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
07,973          fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for B-trees'!\n\n");
07,974      #endif
07,975          exit( 7 );
07,976      }
07,977          FoundInLinkedList = 1+1;
07,978          }
07,979          else // means USED-SLOT
07,980          { FoundInLinkedList = 0;
07,981      StackPtr = 0;
07,982      //          while (PseudoLinkedPointer != 0 && FoundInLinkedList == 0)
07,983          while (PseudoLinkedPointer_64 != 0 && FoundInLinkedList == 0)
07,984          {
07,985      // ***** 'P W P' section [
07,986      // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
07,987      // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wrklen) != 0 )
07,988          // here ALWAYS LW exists: no need for existence check - line below
07,989          // if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
07,990      //          if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wrklen) > 0) // go LP
07,991          // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,992          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;

```



```

07,993 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
07,994 //fread(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
07,995 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
07,996 // [ //r.14+
07,997 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
07,998 if (BSTorBtree == 2) {
07,999 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,000 fread(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fread++;
08,001 } else { // ##### 64bit memory manipulations [
08,002 memcpy( &LEAF[0], (char *)&PseudoLinkedPointerAUX_64, 8+8+8+2*(KeySize+1+8) );
08,003 } // ##### 64bit memory manipulations ]
08,004 memcpy( &FourGramL[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
08,005 // ] //r.14+
08,006 //strFLAG=strcmpKAZE13(FourGramL, wrd);
08,007 strFLAG=memcmp(FourGramL+1, wrd+1, wrdlen); // Since 2019-Feb-07, it is not ASCIIZ string but zero byte is the length of the 1..
08,008 if (strFLAG > 0) // go LP
08,009 // { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 0, 4 ); //LP
08,010 // PseudoLinkedPointer = PseudoLinkedPointerNEW;
08,011 // }
08,012 {
08,013 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,014 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 0; //LP
08,015 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,016 if (StackPtr > 8192*3-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
08,017 BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
08,018 BSTstack[StackPtr] = 0; ++StackPtr; //LPOffset=0;MPOffset=8;RPOffset=16;
08,019 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,020 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,021 //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
08,022 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,023 // [ //r.14+
08,024 memcpy( &PseudoLinkedPointer_64, &LEAF[0], 8 );
08,025 // ] //r.14+
08,026 }
08,027 // else if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wrdlen) < 0) // go RP or MP
08,028 else if (strFLAG < 0) // go RP or MP
08,029 { // RW existence check - line below:
08,030 // if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 ) // RW exists
08,031 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,032 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
08,033 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,034 //fread(&SomeByte, 1, 1, fp_outRG);
08,035 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,036 // [ //r.14+
08,037 memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (KeySize+1+8)], 1 );
08,038 // ] //r.14+
08,039 if (SomeByte != 0 ) // RW exists
08,040 { // Here all 'P W P' section is repeated; the way of handling case when dynamic number of words in leaf
08,041 // *****
08,042 // ***** 'P W P' section 2 [
08,043 // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
08,044 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
08,045 // here ALWAYS RW exists: no need for existence check - line below
08,046 // if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
08,047 // if (memcmp(PseudoLinkedPointer+4+4+4+wrdlen, wrd, wrdlen) > 0) // go MP
08,048 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,049 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
08,050 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);

```

Listing: Nakamichi Ryuugan-ditto-1TB btree.c; Last version: 2021-Aug-30; Font: MxPlus ToshibaTxL2 8x16.ttf; Downloadable at: www.sanmayce.com/Nakamichi/Kaidanji.zip

```

08,109          // [ //r.14+
08,110 //          memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &FourGramL[0], (KeySize+1+8) );
08,111          if (BSTorBtree == 2) {
08,112 //          fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
08,113          } else { // ##### 64bit memory manipulations [
08,114 //          memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
08,115          } // ##### 64bit memory manipulations ]
08,116          // ] //r.14+
08,117          // Counter ]
08,118          }
08,119          WORDcountAttemptsToPut++;
08,120 // ***** 'P W P' section 2 ]
08,121 // *****
08,122          }
08,123          else // RW empty - go MP
08,124 //          { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
08,125 //          PseudoLinkedPointer = PseudoLinkedPointerNEW;
08,126 //          }
08,127          {
08,128          // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,129          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8; //MP
08,130          // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,131          if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
08,132          BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
08,133          BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
08,134          // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,135          //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,136          //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
08,137          // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,138          // [ //r.14+
08,139          memcpy( &PseudoLinkedPointer_64, &LEAF[8], 8 );
08,140          // ] //r.14+
08,141          }
08,142          }
08,143          else { FoundInLinkedList = 1; // wrd is LW
08,144          // Counter [
08,145          if (BSTorBtree == 2) {
08,146          fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,147          }
08,148          //memcpy( &CounterOccurencies, &FourGramL[(KeySize+1+4)-4], 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
08,149          //if (CounterOccurencies<999999999) CounterOccurencies++; // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
08,150          //memcpy( &FourGramL[(KeySize+1+4)-4], &CounterOccurencies, 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
08,151          // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,152          //fwrite(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
08,153          // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,154          // [ //r.14+
08,155 //          memcpy( &LEAF[8 + 8 + 8], &FourGramL[0], (KeySize+1+8) );
08,156          if (BSTorBtree == 2) {
08,157 //          fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
08,158          } else { // ##### 64bit memory manipulations [
08,159 //          memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
08,160          } // ##### 64bit memory manipulations ]
08,161          // ] //r.14+
08,162          // Counter ]
08,163          }
08,164          WORDcountAttemptsToPut++;
08,165 // ***** 'P W P' section ]
08,166          } // while

```

```

08,167 WORDcountAttemptsToPut--; // - 1 due to BST way of counting i.e. direct hash hit is not counted only successors
08,168 }
08,169 // 1] Search ]
08,170 if (FoundInLinkedList == 0) NotFoundKeys++; //r.18
08,171 if (FoundInLinkedList == 1) FoundKeys++; //r.18
08,172 if (FoundInLinkedList == 2) NotFoundKeys++; //r.18
08,173
08,174 if (FoundInLinkedList == 0)
08,175 {
08,176 /*
08,177 // ===== [ The whole section/sub-fragment 2 is commented due to great time differences for Internal_vs_External memory accesses - it is far more
cheap to have the STACK overhead (moved to sub-fragment 1) ] ===== [
08,178 // 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search [
08,179 // 'TracingSearch' is the same as 'Search' except that adds the trail in my simulated stack,
08,180 // the goal is not to waste time in 'Search' by dealing with no needed trail in case of not 'Insert'.
08,181 // Simulated stack contains pairs of 'Address of ParentLEAF' + 'Offset of ParentPointer in ParentLEAF i.e. 0 for LP, 4 for MP, 8 for RP'.
08,182 // 'Offset ...' saves unnecessary comparisons of NEWword which after splitting goes up.
08,183 memcpv( &PseudoLinkedPointer, BufStart+Slot, 4 );
08,184 StackPtr = 0;
08,185 while (PseudoLinkedPointer != 0)
08,186 {
08,187 // ***** 'P W P' section [
08,188 // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
08,189 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
08,190 // here ALWAYS LW exists: no need for existence check - line below
08,191 // if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
08,192 if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wordlen) > 0) // go LP
08,193 { memcpv( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 0, 4 ); //LP
08,194 if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
08,195 BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
08,196 BSTstack[StackPtr] = 0; ++StackPtr; //LPoffset=0;MPoffset=4;RPOffset=8;
08,197 PseudoLinkedPointer = PseudoLinkedPointerNEW;
08,198 }
08,199 else if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wordlen) < 0) // go RP or MP
08,200 { // RW existence check - line below:
08,201 if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 ) // RW exists
08,202 { // Here all 'P W P' section is repeated; the way of handling case when dynamic number of words in leaf
08,203 // *****
08,204 // ***** 'P W P' section 2 [
08,205 // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
08,206 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
08,207 // here ALWAYS RW exists: no need for existence check - line below
08,208 // if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
08,209 if (memcmp(PseudoLinkedPointer+4+4+4+wordlen, wrd, wordlen) > 0) // go MP
08,210 { memcpv( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
08,211 if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
08,212 BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
08,213 BSTstack[StackPtr] = 4; ++StackPtr; //LPoffset=0;MPoffset=4;RPOffset=8;
08,214 PseudoLinkedPointer = PseudoLinkedPointerNEW;
08,215 }
08,216 else if (memcmp(PseudoLinkedPointer+4+4+4+wordlen, wrd, wordlen) < 0) // go RP
08,217 { // No ?W after RW - go RP
08,218 memcpv( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4 + 4, 4 ); //RP
08,219 if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
08,220 BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
08,221 BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=4;RPOffset=8;
08,222 PseudoLinkedPointer = PseudoLinkedPointerNEW;
08,223 }

```

Listing: Nakamichi Ryuugan-ditto-1TB_btree.c; Last version: 2021-Aug-30; Font: MxPlus ToshibaTxL2 8x16.ttf; Downloadable at: www.sanmayce.com/Nakamichi/Kaidanji.zip

```

08,280 //      memcpy( &PseudoLinkedPointerNEW, &bufend[LetterOffset], 4 );
08,281 //      bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
08,282 //      bufend[LetterOffset] = bufend[LetterOffset] + 2*wrklen;
08,283 //      if (MAXusedBuffer[wrklen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrklen] = (unsigned long)(bufend[LetterOffset] -
BufStart);}
08,284 //      }
08,285 //      // BUG IN R.17 and prior - the next 'if' is only for RAM (y/Y) not for SSD (z/Z):
08,286 //      if (BSTorBtree == 2) { //r.18
08,287 BUGGYoffset = (BufEnd_64-(0+14));
08,288 //      } else { // ##### 64bit memory manipulations [
08,289 BUGGYoffset = (BufEnd_64-(unsigned long long)pointerflush_64);
08,290 //      } // ##### 64bit memory manipulations ]
08,291 //      if( 8 + 8 + 8 + 2*(KeySize+1+8) < size_in64_L14 - BUGGYoffset ) // the longest wrklen is LongestLineInclusive but actual is LongestLineInclusive(+ CR char)
08,292 //      {
08,293 PseudoLinkedPointerNEW_64 = BufEnd_64;
08,294 BufEnd_64 = BufEnd_64 + 8 + 8 + 8;
08,295 BufEnd_64 = BufEnd_64 + 2*(KeySize+1+8);
08,296 //      }
08,297 //      else
08,298 //      { printf( "\nLeprechaun: Failure! Increment 'Memory for B-trees'!\n" );
08,299 #ifdef LITE
08,300 #else
08,301 fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, llTOaDigits, 10), _ui64toaKAZEcomma((unsigned long
long)WORDcountDistinct, llTOaDigits2, 10) );
08,302 fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14, llTOaDigits, 10) );
08,303 fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, llTOaDigits, 10) );
08,304 fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
08,305 fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for B-trees'!\n\n");
08,306 #endif
08,307 //      exit( 7 );
08,308 //      }
08,309 //      if (POffsetInLEAF == 0) // wrd < LW
08,310 //      {
08,311 //      memcpy( wrdUP, PseudoLinkedPointer+4+4+4, wrklen ); // LW up
08,312 //      memcpy( PseudoLinkedPointer+4+4+4, wrd, wrklen ); // wrd go to OLD LEAF
08,313 //      memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrklen, wrklen ); // RW go to NEW LEAF
08,314 //      *(char *)(&PseudoLinkedPointer+4+4+4+wrklen) = 0; // RW mark unused in OLD LEAF
08,315 //      [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,316 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
08,317 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,318 //fread(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
08,319 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,320 //fwrite(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
08,321 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
08,322 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,323 //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
08,324 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
08,325 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,326 //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
08,327 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
08,328 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,329 //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
08,330 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,331 // [ //r.14+
08,332 memcpy( &wrdUP[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
08,333 memcpy( &LEAF[8 + 8 + 8], &wrd[0], (KeySize+1+8) );
08,334 memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
08,335 // Here reordering (of writing wrdAUX) is needed to avoid seek the position NEW and stupidly to seek again OLD/current position!

```

```

08,336         memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
08,337             if (BSTorBtree == 2) {
08,338                 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,339                 fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
08,340             } else { // ##### 64bit memory manipulations [
08,341                 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
08,342             } // ##### 64bit memory manipulations ]
08,343             PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
08,344             if (BSTorBtree == 2) {
08,345                 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,346                 fwrite(&wrdAUX[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
08,347             } else { // ##### 64bit memory manipulations [
08,348                 memcpy( (char *)PseudoLinkedPointerAUX_64, &wrdAUX[0], (KeySize+1+8) );
08,349             } // ##### 64bit memory manipulations ]
08,350             // ] //r.14+
08,351         }
08,352     if (PoffsetInLEAF == 8) // LW < wrd < RW
08,353     {
08,354         //         memcpy( wrdUP, wrd, wrdlen ); // wrd up
08,355         //         memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
08,356         //         *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
08,357         //         memcpy( wrdUP, wrd, (KeySize+1+8) ); // wrd up
08,358         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,359         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
08,360         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,361         //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
08,362         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
08,363         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,364         //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
08,365         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
08,366         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,367         //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
08,368         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,369         // [ //r.14+
08,370         memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
08,371         // Here reordering (of writing wrdAUX) is needed to avoid seek the position NEW and stupidly to seek again OLD/current position!
08,372         memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
08,373             if (BSTorBtree == 2) {
08,374                 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,375                 fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
08,376             } else { // ##### 64bit memory manipulations [
08,377                 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
08,378             } // ##### 64bit memory manipulations ]
08,379             PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
08,380             if (BSTorBtree == 2) {
08,381                 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,382                 fwrite(&wrdAUX[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
08,383             } else { // ##### 64bit memory manipulations [
08,384                 memcpy( (char *)PseudoLinkedPointerAUX_64, &wrdAUX[0], (KeySize+1+8) );
08,385             } // ##### 64bit memory manipulations ]
08,386             // ] //r.14+
08,387         }
08,388     if (PoffsetInLEAF == 16) // wrd > RW
08,389     {
08,390         //         memcpy( wrdUP, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW up
08,391         //         *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
08,392         //         memcpy( PseudoLinkedPointerNEW+4+4+4, wrd, wrdlen ); // wrd go to NEW LEAF
08,393         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!

```

```

08,394 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
08,395 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,396 //fread(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
08,397 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
08,398 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,399 //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
08,400 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
08,401 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,402 //fwrite(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
08,403 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,404 // [ //r.14+
08,405 memcpy( &wrdUP[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
08,406 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
08,407 if (BSTorBtree == 2) {
08,408 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,409 fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
08,410 } else { // ##### 64bit memory manipulations [
08,411 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
08,412 } // ##### 64bit memory manipulations ]
08,413 // ] //r.14+
08,414 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one! Here NO need!
08,415 PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
08,416 if (BSTorBtree == 2) {
08,417 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,418 fwrite(&wrd[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
08,419 } else { // ##### 64bit memory manipulations [
08,420 memcpy( (char *)PseudoLinkedPointerAUX_64, &wrd[0], (KeySize+1+8) );
08,421 } // ##### 64bit memory manipulations ]
08,422 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one! Here NO need!
08,423 }
08,424 }
08,425 else // If LEAF is not full: Case #1
08,426 { SplitOccured = 0; WORDcountDistinct++;
08,427 if (POffsetInLEAF == 0) // wrd < [LW][ ] so [LW][ ] -> [ ][LW] -> [wrd][LW]
08,428 {
08,429 // memcpy( PseudoLinkedPointer+4+4+4*wrdlen, PseudoLinkedPointer+4+4+4, wrdlen );
08,430 // memcpy( PseudoLinkedPointer+4+4+4, wrd, wrdlen );
08,431 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,432 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
08,433 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,434 //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
08,435 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
08,436 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,437 //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
08,438 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
08,439 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,440 //fwrite(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
08,441 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,442 // [ //r.14+
08,443 memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
08,444 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &wrdAUX[0], (KeySize+1+8) );
08,445 memcpy( &LEAF[8 + 8 + 8], &wrd[0], (KeySize+1+8) );
08,446 if (BSTorBtree == 2) {
08,447 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,448 fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
08,449 } else { // ##### 64bit memory manipulations [
08,450 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
08,451 } // ##### 64bit memory manipulations ]

```



```

08,452         // ] //r.14+
08,453     }
08,454 }
08,455 if (PoffsetInLEAF == 8) // wrd > [LW][] so [LW][] -> [LW][wrd]
08,456 {
08,457     // memcpy( PseudoLinkedPointer+4+4+4*wrklen, wrd, wrklen );
08,458     // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one! Here NO need!
08,459     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (KeySize+1+8);
08,460     if (BSTorBtree == 2) {
08,461         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,462         fwrite(&wrd[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
08,463     } else { // ##### 64bit memory manipulations [
08,464         memcpy( (char *)PseudoLinkedPointerAUX_64, &wrd[0], (KeySize+1+8) );
08,465     } // ##### 64bit memory manipulations ]
08,466     // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one! Here NO need!
08,467 }
08,468 }
08,469
08,470 if (SplitOccured != 0)
08,471 {
08,472     // ~ Second deal with the INNER NODE(S) i.e Case #3 & Case #4:
08,473     while (StackPtr != 0 || SplitOccured != 0)
08,474     {
08,475         // 'PseudoLinkedPointerNEW' is new LEAF to be inserted
08,476         // 'wrdUP' is NEW word to be inserted
08,477         if (StackPtr != 0)
08,478         {
08,479             PoffsetInLEAF = BSTstack[--StackPtr];
08,480             PseudoLinkedPointer_64 = BSTstack[--StackPtr];
08,481             //if ( *(char *) (PseudoLinkedPointer+4+4+4*wrklen) != 0 ) // If LEAF is full: Case #4
08,482             // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,483             //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
08,484             //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,485             //fread(&SomeByte, 1, 1, fp_outRG);
08,486             // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,487             // [ //r.14+
08,488             PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
08,489             if (BSTorBtree == 2) {
08,490                 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,491                 fread(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fread++;
08,492             } else { // ##### 64bit memory manipulations [
08,493                 memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64, 8+8+2*(KeySize+1+8) );
08,494             } // ##### 64bit memory manipulations ]
08,495             memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (KeySize+1+8)], 1 );
08,496             // ] //r.14+
08,497             if (SomeByte != 0) // RW exists
08,498         { SplitOccured = 1;
08,499             // memcpy( wrdUPold, wrdUP, wrklen ); // LW up
08,500             // PseudoLinkedPointerNEWold = PseudoLinkedPointerNEW;
08,501             // memcpy( wrdUPold, wrdUP, (KeySize+1+8) );
08,502             // PseudoLinkedPointerNEWold_64 = PseudoLinkedPointerNEW_64;
08,503             // Allocate NEW LEAF:
08,504             // if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrklen + 4 + 4 + 4 < GRMBL FoolAgain[(int)wrklen] ) // +4 more for BST instead of LL; + more(see
08,505             // {
08,506             //     memcpy( &PseudoLinkedPointerNEW, &bufend[LetterOffset], 4 );
08,507             //     bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
08,508             //     bufend[LetterOffset] = bufend[LetterOffset] + 2*wrklen;

```

```

08,509 //          if (MAXusedBuffer[wrklen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrklen] = (unsigned long)(bufend[LetterOffset] -
BufStart);}
08,510 //          }
08,511          // BUG IN R.17 and prior - the next 'if' is only for RAM (y/Y) not for SSD (z/Z):
08,512          if (BSTorBtree == 2) { //r.18
08,513      BUGGYoffset = (BufEnd_64-(0+14));
08,514          } else { // ##### 64bit memory manipulations [
08,515      BUGGYoffset = (BufEnd_64-(unsigned long long)pointerflush_64);
08,516          } // ##### 64bit memory manipulations ]
08,517      if( 8 + 8 + 8 + 2*(KeySize+1+8) < size_in64_L14 - BUGGYoffset ) // the longest wrklen is LongestLineInclusive but actual is LongestLineInclusive(+ CR char)
08,518      {
08,519          PseudoLinkedPointerNEW_64 = BufEnd_64;
08,520          BufEnd_64 = BufEnd_64 + 8 + 8 + 8;
08,521          BufEnd_64 = BufEnd_64 + 2*(KeySize+1+8);
08,522          // [ //r.14+
08,523          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64;
08,524          //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,525          //fread(&LEAFNEW[0], 8+8+8+2*(LongestLineInclusive+1+4), 1, fp_outRG);
08,526          // In fact above three lines are slow, the only need is ZEROed LEAFNEW.
08,527          memset(&LEAFNEW[0], 0, 8+8+8+2*(KeySize+1+8));
08,528          // ] //r.14+
08,529      }
08,530      else
08,531      { printf( "\nLeprechaun: Failure! Increment 'Memory for B-trees'!\n" );
08,532      #ifdef LITE
08,533      #else
08,534      //          fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, 11ToaDigits, 10), _ui64toaKAZEcomma((unsigned long
long)WORDcountDistinct, 11ToaDigits, 10) );
08,535      fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14, 11ToaDigits, 10) );
08,536      fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11ToaDigits, 10) );
08,537      fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
08,538      fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for B-trees'!\n\n");
08,539      #endif
08,540      exit( 7 );
08,541      }
08,542      if (POffsetInLEAF == 0) // wrdUPold < LW
08,543      {
08,544      //          memcpy( wrdUP, PseudoLinkedPointer+4+4+4, wrklen ); // LW up
08,545      //          memcpy( PseudoLinkedPointer+4+4+4, wrdUPold, wrklen ); // wrdUPold go to OLD LEAF
08,546      //          memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrklen, wrklen ); // RW go to NEW LEAF
08,547      //          *(char *)(&PseudoLinkedPointer+4+4+4+wrklen) = 0; // RW mark unused in OLD LEAF
08,548      //          // [LP](PseudoLinkedPointerNEWold)[MP][RP](wrdUPold)[LW][RW] -----
08,549      //          // pair [LW] PseudoLinkedPointerNEW goes up !
08,550      //          // PseudoLinkedPointer: PseudoLinkedPointerNEW: !
08,551      //          // [LP](PseudoLinkedPointerNEWold)[](wrdUPold) [MP][RP][][RW] <----
08,552      //          // no need to put zero in RP because logic is based on words existence:
08,553      //          memcpy( PseudoLinkedPointerNEW+0, PseudoLinkedPointer+4, 4 );
08,554      //          memcpy( PseudoLinkedPointerNEW+4, PseudoLinkedPointer+8, 4 );
08,555      //          memcpy( PseudoLinkedPointer+4, &PseudoLinkedPointerNEWold, 4 );
08,556      // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,557      //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
08,558      //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,559      //fread(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
08,560      //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,561      //fwrite(&wrdUPold[0], (LongestLineInclusive+1+4), 1, fp_outRG);
08,562      //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
08,563      //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,564      //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);

```

```

08,565 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
08,566 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,567 //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
08,568 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
08,569 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,570 //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
08,571 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
08,572 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,573 //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
08,574 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 0;
08,575 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,576 //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
08,577 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
08,578 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,579 //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
08,580 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8;
08,581 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,582 //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
08,583 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
08,584 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,585 //fwrite(&PseudoLinkedPointerNEWold_64, 8, 1, fp_outRG);
08,586 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,587 // [ //r.14+
08,588 memcpy( &wrdUP[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
08,589 memcpy( &LEAF[8 + 8 + 8], &wrdUPold[0], (KeySize+1+8) );
08,590 memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
08,591 memcpy( &LEAFNEW[8 + 8 + 8], &wrdAUX[0], (KeySize+1+8) );
08,592 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
08,593 memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[8], 8 );
08,594 memcpy( &LEAFNEW[0], &PseudoLinkedPointerAUXdumbo_64, 8 );
08,595 memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[16], 8 );
08,596 memcpy( &LEAFNEW[8], &PseudoLinkedPointerAUXdumbo_64, 8 );
08,597 memcpy( &LEAF[8], &PseudoLinkedPointerNEWold_64, 8 );
08,598 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
08,599 if (BSTorBtree == 2) {
08,600 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,601 fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
08,602 } else { // ##### 64bit memory manipulations [
08,603 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
08,604 } // ##### 64bit memory manipulations ]
08,605 PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64;
08,606 if (BSTorBtree == 2) {
08,607 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,608 fwrite(&LEAFNEW[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
08,609 } else { // ##### 64bit memory manipulations [
08,610 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAFNEW[0], 8+8+2*(KeySize+1+8) );
08,611 } // ##### 64bit memory manipulations ]
08,612 // ] //r.14+
08,613 }
08,614 if (PoffsetInLEAF == 8) // LW < wrdUPold < RW
08,615 {
08,616 // memcpy( wrdUP, wrdUPold, wrdlen ); // wrdUPold up
08,617 // memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
08,618 // *(char *)PseudoLinkedPointer+4+4+4+wrdlen = 0; // RW mark unused in OLD LEAF
08,619 // // [LP][MP](PseudoLinkedPointerNEWold)[RP][LW](wrdUPold)[RW] -----
08,620 // // pair [wrdUPold] PseudoLinkedPointerNEW goes up !
08,621 // // PseudoLinkedPointer: PseudoLinkedPointerNEW: !
08,622 // // [LP][MP][][LW] (PseudoLinkedPointerNEWold)[RP][][RW] <----

```

```

08,623 //          // no need to put zero in RP because logic is based on words existence:
08,624 //          memcpy( PseudoLinkedPointerNEW+0, &PseudoLinkedPointerNEWold, 4 );
08,625 //          memcpy( PseudoLinkedPointerNEW+4, PseudoLinkedPointer+8, 4 );
08,626 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,627 //          memcpy( wrdUP, wrdUPold, (KeySize+1+8) );
08,628 //          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
08,629 //          //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,630 //          //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
08,631 //          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
08,632 //          //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,633 //          //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
08,634 //          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
08,635 //          //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,636 //          //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
08,637 //          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 0;
08,638 //          //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,639 //          //fwrite(&PseudoLinkedPointerNEWold_64, 8, 1, fp_outRG);
08,640 //          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
08,641 //          //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,642 //          //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
08,643 //          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8;
08,644 //          //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,645 //          //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
08,646 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,647 // [ //r.14+
08,648 //          memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
08,649 //          memcpy( &LEAFNEW[8 + 8 + 8], &wrdAUX[0], (KeySize+1+8) );
08,650 //          memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
08,651 //          memcpy( &LEAFNEW[0], &PseudoLinkedPointerNEWold_64, 8 );
08,652 //          memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[16], 8 );
08,653 //          memcpy( &LEAFNEW[8], &PseudoLinkedPointerAUXdumbo_64, 8 );
08,654 //          PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
08,655 //          if (BSTorBtree == 2) {
08,656 //              fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,657 //              fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
08,658 //          } else { // ##### 64bit memory manipulations [
08,659 //              memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
08,660 //          } // ##### 64bit memory manipulations ]
08,661 //          PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64;
08,662 //          if (BSTorBtree == 2) {
08,663 //              fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,664 //              fwrite(&LEAFNEW[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
08,665 //          } else { // ##### 64bit memory manipulations [
08,666 //              memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAFNEW[0], 8+8+8+2*(KeySize+1+8) );
08,667 //          } // ##### 64bit memory manipulations ]
08,668 //          // ] //r.14+
08,669 //      }
08,670 //      if (PoffsetInLEAF == 16) // wrdUPold > RW
08,671 //      {
08,672 //          memcpy( wrdUP, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW up
08,673 //          *(char *)PseudoLinkedPointer+4+4+4+wrdlen = 0; // RW mark unused in OLD LEAF
08,674 //          memcpy( PseudoLinkedPointerNEW+4+4+4, wrdUPold, wrdlen ); // wrdUPold go to NEW LEAF
08,675 //          // [LP][MP][RP](PseudoLinkedPointerNEWold)[LW][RW](wrdUPold) -----
08,676 //          // pair [RW] PseudoLinkedPointerNEW goes up |
08,677 //          // PseudoLinkedPointer: PseudoLinkedPointerNEW: |
08,678 //          // [LP][MP][LW] [RP](PseudoLinkedPointerNEWold)[wrdUPold] <---
08,679 //          // no need to put zero in RP because logic is based on words existence:
08,680 //          memcpy( PseudoLinkedPointerNEW+0, PseudoLinkedPointer+8, 4 );

```

```

08,681 //      memcpy( PseudoLinkedPointerNEW+4, &PseudoLinkedPointerNEWold, 4 );
08,682 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,683 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
08,684 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,685 //fread(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
08,686 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
08,687 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,688 //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
08,689 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
08,690 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,691 //fwrite(&wrdUPold[0], (LongestLineInclusive+1+4), 1, fp_outRG);
08,692 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
08,693 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,694 //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
08,695 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 0;
08,696 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,697 //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
08,698 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8;
08,699 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,700 //fwrite(&PseudoLinkedPointerNEWold_64, 8, 1, fp_outRG);
08,701 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,702 // [ //r.14+
08,703 memcpy( &wrdUP[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
08,704 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
08,705 memcpy( &LEAFNEW[8 + 8 + 8], &wrdUPold[0], (KeySize+1+8) );
08,706 memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[16], 8 );
08,707 memcpy( &LEAFNEW[0], &PseudoLinkedPointerAUXdumbo_64, 8 );
08,708 memcpy( &LEAFNEW[8], &PseudoLinkedPointerNEWold_64, 8 );
08,709 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
08,710 if (BSTorBtree == 2) {
08,711 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,712 fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
08,713 } else { // ##### 64bit memory manipulations [
08,714 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
08,715 } // ##### 64bit memory manipulations ]
08,716 PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64;
08,717 if (BSTorBtree == 2) {
08,718 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,719 fwrite(&LEAFNEW[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
08,720 } else { // ##### 64bit memory manipulations [
08,721 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAFNEW[0], 8+8+2*(KeySize+1+8) );
08,722 } // ##### 64bit memory manipulations ]
08,723 // ] //r.14+
08,724 }
08,725 }
08,726 else // If LEAF is not full: Case #3
08,727 { SplitOccured = 0;
08,728 if (POffsetInLEAF == 0) // wrdUP < [LW][ ] so [LW][ ] -> [ ][LW] -> [wrdUP][LW]
08,729 {
08,730 //      memcpy( PseudoLinkedPointer+4+4+4*wrdlen, PseudoLinkedPointer+4+4+4, wrdlen );
08,731 //      memcpy( PseudoLinkedPointer+4+4+4, wrdUP, wrdlen );
08,732 //      // [LP][MP][ ] -> [LP][ ][MP] -> [LP][np][MP]
08,733 //      memcpy( PseudoLinkedPointer+8, PseudoLinkedPointer+4, 4 );
08,734 //      memcpy( PseudoLinkedPointer+4, &PseudoLinkedPointerNEW, 4 );
08,735 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,736 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
08,737 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,738 //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);

```

```

08,739 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
08,740 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,741 //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
08,742 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
08,743 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,744 //fwrite(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
08,745 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
08,746 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,747 //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
08,748 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
08,749 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,750 //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
08,751 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
08,752 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,753 //fwrite(&PseudoLinkedPointerNEW_64, 8, 1, fp_outRG);
08,754 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,755 // [ //r.14+
08,756 memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
08,757 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &wrdAUX[0], (KeySize+1+8) );
08,758 memcpy( &LEAF[8 + 8 + 8], &wrdUP[0], (KeySize+1+8) );
08,759 memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[8], 8 );
08,760 memcpy( &LEAF[8 + 8], &PseudoLinkedPointerAUXdumbo_64, 8 );
08,761 memcpy( &LEAF[8], &PseudoLinkedPointerNEW_64, 8 );
08,762 if (BSTorBtree == 2) {
08,763 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,764 fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
08,765 } else { // ##### 64bit memory manipulations [
08,766 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
08,767 } // ##### 64bit memory manipulations ]
08,768 // ] //r.14+
08,769 }
08,770 if (POffsetInLEAF == 8) // wrdUP > [LW][] so [LW][] -> [LW][wrdUP]
08,771 {
08,772 // memcpy( PseudoLinkedPointer+4+4+4+wrdlen, wrdUP, wrdlen );
08,773 // // [LP][MP][] -> [LP][MP][np]
08,774 // memcpy( PseudoLinkedPointer+8, &PseudoLinkedPointerNEW, 4 );
08,775 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,776 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
08,777 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,778 //fwrite(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
08,779 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
08,780 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,781 //fwrite(&PseudoLinkedPointerNEW_64, 8, 1, fp_outRG);
08,782 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
08,783 // [ //r.14+
08,784 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &wrdUP[0], (KeySize+1+8) );
08,785 memcpy( &LEAF[8 + 8], &PseudoLinkedPointerNEW_64, 8 );
08,786 if (BSTorBtree == 2) {
08,787 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,788 fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
08,789 } else { // ##### 64bit memory manipulations [
08,790 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
08,791 } // ##### 64bit memory manipulations ]
08,792 // ] //r.14+
08,793 }
08,794 break;
08,795 }
08,796 }

```

```

08,797     else // Empty stack means ROOT and more over ROOT is already splitted(Case #4 is off)
08,798     {
08,799         // If LEAF is not full: Case #3
08,800         // THIS IS WHERE A NEW(SECOND) LEAF 'PseudoLinkedPointerROOT' must be allocated:
08,801         //         if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrklen + 4 + 4 + 4 < GRMBLFoolAgain[(int)wrklen] ) // +4 more for BST instead of LL; + more(see
LEAF)
08,802         //         {
08,803         //             memcpy( &PseudoLinkedPointerROOT, &bufend[LetterOffset], 4 );
08,804         //             bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
08,805         //             memcpy( bufend[LetterOffset], wrdUP, wrklen );
08,806         //             bufend[LetterOffset] = bufend[LetterOffset] + 2*wrklen;
08,807         //             if (MAXusedBuffer[wrklen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrklen] = (unsigned long)(bufend[LetterOffset] -
BufStart);}
08,808         //         }
08,809         //         // BUG IN R.17 and prior - the next 'if' is only for RAM (y/Y) not for SSD (z/Z):
08,810         //         if (BSTorBtree == 2) { //r.18
08,811         BUGGYoffset = (BufEnd_64-(0+14));
08,812         //         } else { // ##### 64bit memory manipulations [
08,813         BUGGYoffset = (BufEnd_64-(unsigned long long)pointerflush_64);
08,814         //         } // ##### 64bit memory manipulations ]
08,815         if( 8 + 8 + 8 + 2*(KeySize+1+8) < size_in64_L14 - BUGGYoffset ) // the longest wrklen is LongestLineInclusive but actual is LongestLineInclusive(+ CR char)
08,816         {
08,817             PseudoLinkedPointerROOT_64 = BufEnd_64;
08,818             BufEnd_64 = BufEnd_64 + 8 + 8 + 8;
08,819             BufEnd_64 = BufEnd_64 + 2*(KeySize+1+8);
08,820             PseudoLinkedPointerAUX_64 = PseudoLinkedPointerROOT_64 + 8 + 8 + 8;
08,821             if (BSTorBtree == 2) {
08,822                 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,823                 fwrite(&wrdUP[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
08,824                 } else { // ##### 64bit memory manipulations [
08,825                 memcpy( (char *)&PseudoLinkedPointerAUX_64, &wrdUP[0], (KeySize+1+8) );
08,826                 //         } // ##### 64bit memory manipulations ]
08,827             }
08,828         else
08,829         { printf( "\nLeprechaun: Failure! Increment 'Memory for B-trees'!\n" );
08,830 #ifdef LITE
08,831 #else
08,832         fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, 11ToaDigits, 10), _ui64toaKAZEcomma((unsigned long
long)WORDcountDistinct, 11ToaDigits2, 10) );
08,833         fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14, 11ToaDigits, 10) );
08,834         fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11ToaDigits, 10) );
08,835         fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
08,836         fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for B-trees'!\n\n");
08,837 #endif
08,838         exit( 7 );
08,839     }
08,840     // Here -- 'PseudoLinkedPointerROOT' --
08,841     //         | (wrdUP) |
08,842     // 'PseudoLinkedPointer' <-- --> 'PseudoLinkedPointerNEW'
08,843     // (LW) (RW)
08,844     //     memcpy( PseudoLinkedPointerROOT, &PseudoLinkedPointer, 4 ); // LP
08,845     //     memcpy( PseudoLinkedPointerROOT+4, &PseudoLinkedPointerNEW, 4 ); // MP
08,846     //     // Here must NEW ROOT be updated i.e. HASH table(SLOT) must point it:
08,847     //     memcpy( BufStart+Slot, &PseudoLinkedPointerROOT, 4 );
08,848     PseudoLinkedPointerAUX_64 = PseudoLinkedPointerROOT_64;
08,849     if (BSTorBtree == 2) {
08,850         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,851         fwrite(&PseudoLinkedPointer_64, 8, 1, fp_outRG); Total_fwrite++;

```

```

08,852             } else { // ##### 64bit memory manipulations [
08,853             memcpy( (char *)PseudoLinkedPointerAUX_64, &PseudoLinkedPointer_64, 8 );
08,854             } // ##### 64bit memory manipulations ]
08,855     PseudoLinkedPointerAUX_64 = PseudoLinkedPointerROOT_64 + 8;
08,856     if (BSTorBtree == 2) {
08,857         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
08,858         fwrite(&PseudoLinkedPointerNEW_64, 8, 1, fp_outRG); Total_fwrite++;
08,859         } else { // ##### 64bit memory manipulations [
08,860         memcpy( (char *)PseudoLinkedPointerAUX_64, &PseudoLinkedPointerNEW_64, 8 );
08,861         } // ##### 64bit memory manipulations ]
08,862         memcpy( BufStart+Slot, &PseudoLinkedPointerROOT_64, 8 );
08,863         break; //because it is ROOT without split
08,864     }
08,865     } // while
08,866 } //if (SplitOccured != 0)
08,867 // 3] Insert Iterative ]
08,868 } //if (FoundInLinkedList == 0)
08,869 // ##### B-tree order 3 fragment 64bit ]
08,870 } // if (FoundInLinkedList_RAM == 1) { // 2019-Dec-04
08,871
08,872 // Building nonunique 2 of 2 ]
08,873     // Counter ]
08,874     WORDcountBOTTOM++; //Nakamichi
08,875     WORDcountBOTTOMPerMatchLen++;
08,876 if (*argv[k_FIX] == 'Y' || *argv[k_FIX] == 'Z')
08,877 //if (CounterOccurrences>1) // For Nakamichi
08,878 //     fprintf(fp_out, "%s\t%s\r\n", _ui64toaKAZEzerocomma(CounterOccurrences, 11ToaDigits2, 10)+(26-11), wrd); WORDcountBOTTOM++;
08,879 if (*argv[k_FIX] == 'y' || *argv[k_FIX] == 'z')
08,880 // NO-DUMP: [
08,881     WORDcountBOTTOM++;
08,882 //     fprintf(fp_out, "%s\r\n", wrd); WORDcountBOTTOM++;
08,883 // NO-DUMP: ]
08,884 //fwrite(CRdLFa, 2, 1, fp_out);
08,885 }
08,886     PseudoLinkedPointer_64DUMP = 0;
08,887     NumberOfLEAFs++; //r.14
08,888 }
08,889 }
08,890 // $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ B-tree order 3 traverse 64bit ]
08,891
08,892     } //if (PseudoLinkedPointer_64 != 0)
08,893
08,894 } // for( jjj = 0; jjj < ( ((1LL)<<HashInBITS_GLOBAL) ); jjj++ ) //r.18
08,895
08,896 // PASS #2: ]
08,897
08,898 RipPasses++;
08,899 if (RipPasses <= (1<<<(HashInBITS_GLOBAL-HashChunkSizeInBITS_GLOBAL))-1) goto WhyTheHellForIsNotWorking;
08,900 //} // for( RipPasses = 1-1; RipPasses <= (1<<<(HashInBITS-HashChunkSizeInBITS))-1; RipPasses++ )
08,901
08,902 // Counting trees [
08,903 NumberOfTrees=0;
08,904 AllSlots=0;
08,905
08,906 for( j = 0; j < MatchLensNUM; j++ ) //r.18
08,907 {
08,908     for( jjj = 0; jjj < ( ((1LL)<<HashInBITS_GLOBAL) ); jjj++ ) //r.18
08,909 {

```



```

08,910         Slot = (AllSlots)<<3; AllSlots++;
08,911         memcpy( &PseudoLinkedPointer_64, BufStart+Slot, 8 );
08,912         if (PseudoLinkedPointer_64 != 0) NumberOfTrees++;
08,913     }
08,914 }
08,915 // Counting trees ]
08,916
08,917 printf( "DONE; %s DEFRAGMENTED B-trees have been rooted so far.\n", _ui64toaKAZEzerocomma(NumberOfTrees, 11ToaDigits, 10)+(26-14));
08,918 printf( "          %s TotalNonUnique (of length %s) keys have been inserted into DEFRAGMENTED B-trees.\n", _ui64toaKAZEzerocomma(TotalNonUnique[MatchLens[jj]],
11ToaDigits, 10), _ui64toaKAZEzerocomma(MatchLens[jj], 11ToaDigits2, 10)+(26-3) );
08,919 fflush(stdout);
08,920
08,921
08,922 // Kinda pass #3 (only speedup functionality):
08,923 #ifdef Kanshiketsu
08,924 if (jj<=9) { // Enforce it to be up to 64 inclusive i.e. in range 0..9 due to being sorted until [9] - have to sort the matchlens within the array...
08,925
08,926 memset(SourceBlockSKIParray,0,SourceSize); // 0 means the current position has no matches 4 or bigger i.e. it is an unique BB.
08,927 //int MatchLens[MatchLensNUMdummy]={4,6,8,10,12,14,16,18,36,64,24,28,48,56,128,256};
08,928 //          0,1,2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f
08,929 HowManyPositionsAreNonUnique=0;
08,930 for (BuildingBlocksSTRIDE=0; (signed long long)BuildingBlocksSTRIDE < (signed long long)size_inLINESIXFOUR-MatchLens[jj]+1; BuildingBlocksSTRIDE++) {
08,931 GLOBAL_CurrentPositionForReading_TAILforLookAhead = GLOBAL_SourceBlock + BuildingBlocksSTRIDE;
08,932
08,933 // Below segment is identical to 'step #2 of 2: Loading into LastSeenOffset_PseudoPointer[] the actual LookAheadBuffer i.e. EncStart' segment appearing in
SlidingWindow where just checking is done...
08,934 // step #2 of 2: Loading into LastSeenOffset_PseudoPointer[] the actual LookAheadBuffer i.e. EncStart
08,935 //GLOBAL_CurrentPositionForReading_TAILforLookAhead = encStart;
08,936 //while ( GLOBAL_CurrentPositionForReading_TAILforLookAhead <= encStart ) { // TAIL should become LookAhead (i.e. encStart)
08,937 //for (jj=0; jj< MatchLensNUM; jj++) {
08,938     LastSeenOffset_PseudoPointer[jj] = 0; // By Default - Not Seen
08,939     if ( GLOBAL_CurrentPositionForReading_TAILforLookAhead + MatchLens[jj] -1 <= GLOBAL_SourceBlock + GLOBAL_SourceSize -1 ) {
08,940
08,941         // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed [
08,942         //         wrdlen=0;
08,943         //         memset( &wrd[0], 0, (LongestLineInclusive+1*8) ); //r.18, see below, the old nullifier is commented.
08,944         //         for (mm=0; mm< MatchLens[jj]; mm++) {
08,945         //             memcpy( &wrd[ wrdlen ], TwoDigitHEXlist[(unsigned char *)](GLOBAL_CurrentPositionForReading_TAILforLookAhead+mm)], 2 );
08,946         //             wrdlen++;
08,947         //             wrdlen++;
08,948         //         }
08,949         // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed ]
08,950
08,951         wrdlen=MatchLens[jj];
08,952
08,953 #if defined(_NSHA3) || defined(_NPRV) || defined(_NDD) || defined(_NAquaHash) //2020-Nov-20
08,954
08,955 #ifdef _NPRV //2020-Nov-12 [ Consider not only 256[+] matchlens but 24[+] as well - in order to save memory!
08,956     if (wrdlen > MatchLenAboveWhichHASHkicksin) {
08,957         prvhash42( (unsigned char *))(GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen, (unsigned char *)PRVhash, PRVhashlenInBYTES, 0, 0, 0 );
//2020-Nov-17, for v2.0 ", 0" was added
08,958         wrdlen=MatchLenAboveWhichHASHkicksin; // very aggressive is 16B or 128b, if collisions happen then increase to 20B or 160b
08,959         memcpy( &wrd[ 0+1 ], (unsigned char *)PRVhash, wrdlen );
08,960     } else
08,961         memcpy( &wrd[ 0+1 ], (unsigned char *))(GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen );
08,962 #endif
08,963 #ifdef _NSHA3
08,964     //if (wrdlen >= 256) {

```

```
08,965         if (wrdlen > MatchLenAboveWhichHASHkicksin) {
08,966             FIP5202_SHA3_224((unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen, (unsigned char *) SHA328bytes);
08,967             wrdlen=MatchLenAboveWhichHASHkicksin; //2020-Nov-12
08,968             memcpy( &wrd[ 0+1 ], (unsigned char *) SHA328bytes, wrdlen );
08,969         } else
08,970             memcpy( &wrd[ 0+1 ], (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen );
08,971 #endif
08,972 #ifdef _NDD
08,973     //if (wrdlen >= 256) { // 2020-Jun-13
08,974     if (wrdlen > MatchLenAboveWhichHASHkicksin) {
08,975         /*(uint64_t*)&DD[0] = crc64((unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen);
08,976         /*(uint32_t*)&DD[8] = crc32c_sw((unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen);
08,977         // Above 2 lines are glued into next one:
08,978         DoubleDeuce( (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen );
08,979         wrdlen=MatchLenAboveWhichHASHkicksin; //2020-Nov-12
08,980         memcpy( &wrd[ 0+1 ], (unsigned char *) DD, wrdlen );
08,981     } else
08,982         memcpy( &wrd[ 0+1 ], (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen );
08,983 #endif
08,984 #ifdef _NAquaHash
08,985     //if (wrdlen >= 256) { // 2020-Jun-13
08,986     if (wrdlen > MatchLenAboveWhichHASHkicksinAQUA) {
08,987         /*(uint64_t*)&DD[0] = crc64((unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen);
08,988         /*(uint32_t*)&DD[8] = crc32c_sw((unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen);
08,989         // Above 2 lines are glued into next one:
08,990         DoubleDeuceAES_Gumbotron_YMM( (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen );
08,991         wrdlen=MatchLenAboveWhichHASHkicksinAQUA; //2020-Nov-12
08,992         memcpy( &wrd[ 0+1 ], (unsigned char *) DDAES, wrdlen );
08,993     } else
08,994         memcpy( &wrd[ 0+1 ], (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen );
08,995 #endif
08,996 #endif
08,997 #endif
08,998 #else
08,999     memcpy( &wrd[ 0+1 ], (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen );
09,000 #endif
09,001 #endif
09,002     memcpy( &wrd[ 0 ], &wrdlen, 1 ); // The keysize is 1..255, pretty enough since my needs are 1..28 (up to SHA3-224).
09,003 /*
09,004 #ifdef _NSHA3
09,005     //
09,006     if (wrdlen != 28) // 2020-Jun-13
09,007         memcpy( &wrd[ 0+1 ], (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen );
09,008     else
09,009         memcpy( &wrd[ 0+1 ], (unsigned char *) SHA328bytes, wrdlen );
09,010 #else
09,011     //
09,012     if (wrdlen < 28)
09,013         memcpy( &wrd[ 0+1 ], (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen );
09,014     else
09,015         memcpy( &wrd[ 0+1 ], (unsigned char *) SHA328bytes, wrdlen );
09,016 #endif
09,017 /*
09,018 //Slot = FNV1A_Hash_Jesteress_27bit(wrd +1, wrdlen);
09,019 Slot = FNV1A_Pippip_Yurii(wrd +1, wrdlen); // Changed 2019-Nov-28
09,020
09,021 Slot = Slot<<3;
09,022 // NEW NEW NEW [ //r.18
```

```

09,023 // In here Slot is not within a single pool but in MatchLensNUM sub-pools i.e. MatchLensNUM-way i.e. MatchLensNUM hashpots:
09,024 /*
09,025 for (GettingIndexOfArray=0; GettingIndexOfArray<MatchLensNUM; GettingIndexOfArray++) {
09,026     // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed [
09,027     //     if ( (wrdlen>>1)==MatchLens[GettingIndexOfArray] ) break;
09,028     if ( (wrdlen)==MatchLens[GettingIndexOfArray] ) break;
09,029     // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed ]
09,030 }
09,031 */
09,032 GettingIndexOfArray=jj; // same as above atrocity
09,033 Slot = Slot +(GettingIndexOfArray*(1LL<<HashInBITS_GLOBAL)*8); // CAUTION: 'wrdlen' is halved here, when a new revision comes with 1:1 keysize then change it.
09,034 // NEW NEW NEW ] //r.18
09,035
09,036 //KeySize = LongestLineInclusive; // 'LongestLineInclusive' is for 1-way hashing where different KeySizes can occupy same HASH pool.
09,037 KeySize = wrdlen; //r.18, In case multi-way hashing: either the MAX allowed or the 'wrdlen'.
09,038
09,039 //Slot = 0; // One Tree only!
09,040 memcpy( &PseudoLinkedPointer_64, GLOBAL_HASHPOT+Slot, 8 );
09,041
09,042 // 1] Search [ Read the 8 bytes field into LastSeenOffset_PseudoPointer[jj]
09,043
09,044     if (PseudoLinkedPointer_64 == 0) // means EMPTY-SLOT
09,045     {
09,046         // Impossible, they all have already been inserted...
09,047     }
09,048     else // means USED-SLOT
09,049     { FoundInLinkedList = 0;
09,050       StackPtr = 0;
09,051       // while (PseudoLinkedPointer != 0 && FoundInLinkedList == 0)
09,052       while (PseudoLinkedPointer_64 != 0 && FoundInLinkedList == 0)
09,053       {
09,054       // ***** 'P W P' section [
09,055       // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
09,056       // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
09,057       // here ALWAYS LW exists: no need for existence check - line below
09,058       // if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
09,059       //     if (memcmp(PseudoLinkedPointer+4+4+4,wrd,wrdlen) > 0) // go LP
09,060       // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
09,061       //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
09,062       //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
09,063       //fread(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
09,064       // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
09,065       // [ //r.14+
09,066       PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
09,067       if (BSTorBtree == 2) {
09,068       fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
09,069       fread(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG);
09,070       } else { // ##### 64bit memory manipulations [
09,071       memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64, 8+8+2*(KeySize+1+8) );
09,072       } // ##### 64bit memory manipulations ]
09,073       memcpy( &FourGramL[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
09,074       // ] //r.14+
09,075       //strFLAG=strcmpKAZE13(FourGramL, wrd);
09,076       strFLAG=memcmp(FourGramL+1, wrd +1, wrdlen); // Since 2019-Feb-07, it is not ASCIIZ string but zero byte is the length of the 1..
09,077       if (strFLAG > 0) // go LP
09,078       { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 0, 4 ); //LP
09,079       //     PseudoLinkedPointer = PseudoLinkedPointerNEW;
09,080       // }

```

```

09,081      {
09,082      // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
09,083      //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 0; //LP
09,084      // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
09,085      if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
09,086      BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
09,087      BSTstack[StackPtr] = 0; ++StackPtr; //LPoffset=0;MPoffset=8;RPOffset=16;
09,088      // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
09,089      //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
09,090      //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
09,091      // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
09,092      // [ //r.14+
09,093      memcpy( &PseudoLinkedPointer_64, &LEAF[0], 8 );
09,094      // ] //r.14+
09,095      }
09,096      //      else if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wrdlen) < 0) // go RP or MP
09,097      //      else if (strFLAG < 0) // go RP or MP
09,098      //      { // RW existence check - line below:
09,099      //      if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 ) // RW exists
09,100      //      [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
09,101      //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
09,102      //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
09,103      //fread(&SomeByte, 1, 1, fp_outRG);
09,104      // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
09,105      // [ //r.14+
09,106      memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (KeySize+1+8)], 1 );
09,107      // ] //r.14+
09,108      if (SomeByte != 0 ) // RW exists
09,109      { // Here all 'P W P' section is repeated; the way of handling case when dynamic number of words in leaf
09,110      // *****
09,111      // ***** 'P W P' section 2 [
09,112      // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
09,113      // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
09,114      // here ALWAYS RW exists: no need for existence check - line below
09,115      // if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
09,116      // if (memcmp(PseudoLinkedPointer+4+4+4+wrdlen, wrd, wrdlen) > 0) // go MP
09,117      // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
09,118      //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
09,119      //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
09,120      //fread(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
09,121      // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
09,122      // [ //r.14+
09,123      memcpy( &FourGramL[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
09,124      // ] //r.14+
09,125      //strFLAG=strcmpKAZE13(FourGramL, wrd);
09,126      strFLAG=memcmp(FourGramL+1, wrd+1, wrdlen); // Since 2019-Feb-07, it is not ASCIIZ string but zero byte is the length of the 1..
09,127      if (strFLAG > 0) // go MP
09,128      { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
09,129      //      PseudoLinkedPointer = PseudoLinkedPointerNEW;
09,130      //      }
09,131      //      {
09,132      // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
09,133      //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8; //MP
09,134      // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
09,135      if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
09,136      BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
09,137      BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=8;RPOffset=16;
09,138      // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!

```

```

09,139 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
09,140 //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
09,141 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
09,142 // [ //r.14+
09,143 memcpy( &PseudoLinkedPointer_64, &LEAF[8], 8 );
09,144 // ] //r.14+
09,145 }
09,146 // else if (memcmp(PseudoLinkedPointer+4+4+4*wordlen, wrd, wordlen) < 0) // go RP
09,147 else if (strFLAG < 0) // go RP
09,148 { // No ?W after RW - go RP
09,149 // memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4 + 4, 4 ); //RP
09,150 // PseudoLinkedPointer = PseudoLinkedPointerNEW;
09,151 // }
09,152 {
09,153 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
09,154 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8; //RP
09,155 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
09,156 if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
09,157 BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
09,158 BSTstack[StackPtr] = 16; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
09,159 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
09,160 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
09,161 //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
09,162 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
09,163 // [ //r.14+
09,164 memcpy( &PseudoLinkedPointer_64, &LEAF[8 + 8], 8 );
09,165 // ] //r.14+
09,166 }
09,167 else { FoundInLinkedList = 1; // wrd is RW
09,168 // Counter [
09,169 // if (BSTorBtree == 2) {
09,170 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
09,171 }
09,172 //memcpy( &CounterOccurrences, &FourGramL[(KeySize+1+4)-4], 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
09,173 memcpy( &LastSeenOffset_PseudoPointer[jj], &FourGramL[(KeySize+1+8)-8], 8 );
09,174 //if (CounterOccurrences<999999999) CounterOccurrences++; // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
09,175 //memcpy( &FourGramL[(KeySize+1+4)-4], &CounterOccurrences, 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
09,176 // memcpy( &FourGramL[(KeySize+1+8)-8], &GLOBAL_CurrentPositionForReading_TAILforLookAhead, 8 ); // 2020-Feb-02, commented - NO UPDATING! //
2020-Jun-30: In 32bit code it is 4 not 8: printf("%d", sizeof(char*));
09,177 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
09,178 //fwrite(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
09,179 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
09,180 // [ //r.14+
09,181 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &FourGramL[0], (KeySize+1+8) );
09,182 // if (BSTorBtree == 2) {
09,183 fwrite(&LEAF[0], 8+8+8*2*(KeySize+1+8), 1, fp_outRG);
09,184 } else { // ##### 64bit memory manipulations [
09,185 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8*2*(KeySize+1+8) );
09,186 } // ##### 64bit memory manipulations ]
09,187 // ] //r.14+
09,188 // Counter ]
09,189 }
09,190 // ***** 'P W P' section 2 ]
09,191 // *****
09,192 }
09,193 else // RW empty - go MP
09,194 { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
09,195 // PseudoLinkedPointer = PseudoLinkedPointerNEW;

```

```

09,196 //      }
09,197 //      {
09,198 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
09,199 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8; //MP
09,200 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
09,201 if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
09,202 BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
09,203 BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
09,204 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
09,205 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
09,206 //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
09,207 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
09,208 // [ //r.14+
09,209 memcpy( &PseudoLinkedPointer_64, &LEAF[8], 8 );
09,210 // ] //r.14+
09,211 //      }
09,212 //    }
09,213 //    else { FoundInLinkedList = 1; // wrd is LW
09,214 // Counter [
09,215 //    if (BSTorBtree == 2) {
09,216 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
09,217 //    }
09,218 //memcpy( &CounterOccurrences, &FourGramL[(KeySize+1+4)-4], 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
09,219 memcpy( &LastSeenOffset_PseudoPointer[jj], &FourGramL[(KeySize+1+8)-8], 8 );
09,220 //if (CounterOccurrences<999999999) CounterOccurrences++; // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
09,221 //memcpy( &FourGramL[(KeySize+1+4)-4], &CounterOccurrences, 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
09,222 //    memcpy( &FourGramL[(KeySize+1+8)-8], &GLOBAL_CurrentPositionForReading_TAILforLookAhead, 8 ); // 2020-Feb-02, commented - NO UPDATING! //
2020-Jun-30: In 32bit code it is 4 not 8: printf("%d",sizeof(char*));
09,223 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
09,224 //fwrite(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
09,225 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
09,226 // [ //r.14+
09,227 memcpy( &LEAF[8 + 8 + 8], &FourGramL[0], (KeySize+1+8) );
09,228 //    if (BSTorBtree == 2) {
09,229 fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG);
09,230 //    } else { // ##### 64bit memory manipulations [
09,231 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
09,232 //    } // ##### 64bit memory manipulations ]
09,233 // ] //r.14+
09,234 // Counter ]
09,235 //    }
09,236 // ***** 'P W P' section ]
09,237 //    } // while
09,238 //    }
09,239 //  ]
09,240 // 1] Search ] Write (uint64_t)(GLOBAL_CurrentPositionForReading_TAILforLookAhead) into the 8 bytes field
09,241 //  ]
09,242 // } //for (jj=0;
09,243 //
09,244 // GLOBAL_CurrentPositionForReading_TAILforLookAhead++;
09,245 // } // At the end of this loop we should have (in LastSeenOffset_PseudoPointer[]) all matches found for position 'encStart'.
09,246 if (FoundInLinkedList) {/(LastSeenOffset_PseudoPointer[jj]) {
09,247 SourceBlockSKIParray[BuildingBlocksSTRIDE] = MatchLens[jj]; // could be (in the future) >127 i.e. 0..255
09,248 HowManyPositionsAreNonUnique++;
09,249 }
09,250 }// for (BuildingBlocksSTRIDE=0; (signed long long)BuildingBlocksSTRIDE < (signed long long)size_inLINESIXFOUR-MatchLens[jj]+1; BuildingBlocksSTRIDE++) {
09,251
09,252 printf( "Kanshiketsu: HowManyPositionsAreNonUnique (for current order) = %s\n", _ui64toaKAZEcomma(HowManyPositionsAreNonUnique, 11ToADigits5, 10));

```

```

09,253
09,254 }//if (jj<=9) { // Enforce it to be up to 64 inclusive due to being sorted until [9] - have to sort the matchlens within the array...
09,255 #endif
09,256
09,257
09,258 } //for (jj=0;
09,259 printf( "\n");
09,260
09,261 goto SkipDestroyingTheTrees;
09,262 //...
09,263 SkipDestroyingTheTrees:
09,264
09,265 printf( "Histogram (rotate it 90 degrees clockwise) of Unique-Building-Blocks appearing 2[+] times (e.g. 'alfalfa-alfa' stream has 1 BB with length 4 appearing 3
times thus adding 1 to the count for {004}): \n" );
09,266 j=1;
09,267 for (jj=0; jj< MatchLensNUM; jj++) {
09,268 // padding 123,123,123,123 4x3+3=15
09,269 if ( j<strlen(_ui64toaKAZEcomma(TotalNonUnique[MatchLens[jj]], 11TOaDigits, 10)) ) j=strlen(_ui64toaKAZEcomma(TotalNonUnique[MatchLens[jj]], 11TOaDigits, 10));
09,270 }
09,271 /*
09,272 for (jj=MatchLensNUM-1; (signed int)jj>= 0; (signed int)jj--) {
09,273 // padding 123,123,123,123 4x3+3=15
09,274 for (jjj=0; jjj< j-strlen(_ui64toaKAZEcomma(TotalNonUnique[MatchLens[jj]], 11TOaDigits, 10)); jjj++) {
09,275 printf( " ");
09,276 }
09,277 printf( "%s%s\n", _ui64toaKAZEcomma(TotalNonUnique[MatchLens[jj]], 11TOaDigits, 10), _ui64toaKAZEzerocomma(MatchLens[jj], 11TOaDigits2, 10)+(26-3) );
09,278 }
09,279 */
09,280 for (jj=704; jj>=1; jj--) {
09,281 if (TotalNonUnique[jj] != -1) {
09,282 for (jjj=0; jjj< j-strlen(_ui64toaKAZEcomma(TotalNonUnique[jj], 11TOaDigits, 10)); jjj++) {
09,283 printf( " ");
09,284 }
09,285 printf( "%s%s\n", _ui64toaKAZEcomma(TotalNonUnique[jj], 11TOaDigits, 10), _ui64toaKAZEzerocomma(jj, 11TOaDigits2, 10)+(26-3) );
09,286 }
09,287 }
09,288
09,289 printf( "\n");
09,290 printf( "Leprechaun: Total Searches-n-Inserts Per Second: %s SNIPS = %s keys in %s seconds\n", _ui64toaKAZEcomma((double)(WORDcount)/((double)(time(NULL) - time1 + 1),
11TOaDigits3, 10), _ui64toaKAZEcomma(WORDcount, 11TOaDigits4, 10), _ui64toaKAZEcomma(time(NULL) - time1 + 1, 11TOaDigits5, 10));
09,291 printf( "Leprechaun: RAM needed to house B-trees (relative to the file being ripped): %sN = %sMB\n", _ui64toaKAZEcomma((BufEnd_64-(unsigned long
long)pointerflush_64+1)/size_inLINESIXFOUR, 11TOaDigits3, 10), _ui64toaKAZEcomma(1+((BufEnd_64-(unsigned long long)pointerflush_64+1)>>20), 11TOaDigits2, 10));
09,292 printf( "Leprechaun: RAM needed to build B-trees IN ONE PASS: (Target-Buffer = %s MB) x %s passes = %sMB\n", _ui64toaKAZEcomma((SourceSize+512+(unsigned long
long)SpeedUpBuilding*1024*1024)>>20, 11TOaDigits2, 10), _ui64toaKAZEcomma(1<<<(HashInBITS_GLOBAL-HashChunkSizeInBITS_GLOBAL), 11TOaDigits3, 10), _ui64toaKAZEcomma(
((SourceSize+512+(unsigned long long)SpeedUpBuilding*1024*1024)>>20) * (1<<<(HashInBITS_GLOBAL-HashChunkSizeInBITS_GLOBAL)), 11TOaDigits4, 10) );
09,293 printf( "Note: In case of RAM in spades then may use compile option: -DSpeedUpBuilding=%s\n", _ui64toaKAZE( ((SourceSize+512+(unsigned long
long)SpeedUpBuilding*1024*1024)>>20) * (1<<<(HashInBITS_GLOBAL-HashChunkSizeInBITS_GLOBAL)), 11TOaDigits4, 10) );
09,294
09,295 //if defined(ExternalRAM)
09,296 if (BSTorBtree == 2) {
09,297 printf( "Leprechaun: Total IOPS for %s 'freads' and %s 'fwrites' (of packets %d bytes long) during loading traversing all orders: %s IOPS\n",
_ui64toaKAZEcomma(Total_fread, 11TOaDigits, 10), _ui64toaKAZEcomma(Total_fwrite, 11TOaDigits2, 10), 8+8+2*(LongestLineInclusive+1+8),
_ui64toaKAZEcomma((double)(Total_fwrite+Total_fread)/((double)(time(NULL) - time1 + 1), 11TOaDigits3, 10));
09,298 }
09,299 //endif
09,300 printf( "\n");
09,301
09,302 //exit(0);

```

```

09,303
09,304 } // B_tree_Non_Unique_Only_DEFRAGMENTED]
09,305
09,306
09,307 void B_tree_Non_Unique_Only(int argc, char *argv[], char* SourceBlock, uint64_t SourceSize, char* VerifyBlock) { // B_tree_Non_Unique_Only[
09,308     //int BSTorBtree = 0;
09,309     int BSTorBtree_RAM = 3; //Internal // 2019-Dec-04
09,310     FILE *fp_in, *fp_out, *fp_outLOG, *fp_inLINE;
09,311     int Thunderwith;
09,312 // cleand unused below...
09,313     int nlines;
09,314
09,315     int LetterOffset;
09,316     unsigned long long FilesLEN;
09,317     unsigned long long WORDcount;
09,318     unsigned long long WORDcountBOTTOM;
09,319     unsigned long long WORDcountAttemptsToPut;
09,320     unsigned long long Total_fread=0, Total_fwrite=0;
09,321
09,322 // 15fixfixfixfix [
09,323     //unsigned long NumberOfFiles, WORDcountDistinct, WORDcountDistinctTOTAL = 0, TotalMemoryNeededForOnePass = 0; // This was in r.15fixfixfix
09,324     unsigned long long NumberOfFiles, WORDcountDistinct, WORDcountDistinctTOTAL = 0, TotalMemoryNeededForOnePass = 0; // This was in r.15fixfixfixfix
09,325 // 15fixfixfixfix ]
09,326     unsigned long long NumberOfLines; // rev. 12+
09,327     unsigned long WHOLEletter_BufferSize;
09,328     unsigned long long WHOLEletter_BufferSize_L14;
09,329     unsigned long memory_size, LetterBuffer, j, k, LINE10len, wrdlen;
09,330     unsigned long k_FIX;
09,331     unsigned long long i; // rev. 12+
09,332     //unsigned long size_in, size_out, size_inLINE;
09,333     unsigned long size_in; // rev. 12+
09,334 #if defined(_WIN32_ENVIRONMENT_)
09,335     unsigned long long size_inLINESIXFOUR;
09,336 #else
09,337     size_t size_inLINESIXFOUR;
09,338 #endif /* defined(_WIN32_ENVIRONMENT_) */
09,339
09,340     const int NumberOfSLOTs = 4096*2; // Since r.12+ in rev.12 it was 4096
09,341     unsigned long StackPtr;
09,342     //unsigned long BSTstack [65536*3]; // BST in worst case could become a LL.
09,343     unsigned long long BSTstack [8192*3]; // BST in worst case could become a LL.
09,344     unsigned long NumberOfTrees=0, NumberOfHashCollisions=0;
09,345     unsigned long iBSTwithMAXpeak, jBSTwithMAXpeak;
09,346     unsigned int PEAKibBST;
09,347     unsigned long BSTsTotalLEAFs=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break' is ?!
09,348     unsigned long BSTwithMAXnode=0, BSTcurrentNode=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break' is ?!
09,349     unsigned long BSTcurrentNodeMAXqQUANTITY=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where
    'break' is ?!
09,350     unsigned long BSTwithMAXnodePEAK=1, BSTwithMAXnodeLEAF=1; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break'
    is ?!
09,351     unsigned long BSTwithMAXpeak=0, BSTcurrentPeak=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break' is ?!
09,352     unsigned long BSTcurrentPeakMAX=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break' is
    ?!
09,353     unsigned long BSTcurrentPeakMAXqQUANTITY=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where
    'break' is ?!
09,354     unsigned long BSTwithMAXpeakNODE=1, BSTwithMAXpeakLEAF=1; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break'
    is ?!
09,355     unsigned long BSTwithMAXleaf=0, BSTcurrentLeaf=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break' is ?!

```



```

09,356 unsigned long          BSTcurrentLeafMAXqUANTITY=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where
'break' is ?!
09,357 unsigned long BSTwithMAXleafNODE=1, BSTwithMAXleafPEAK=1; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break'
is ?!
09,358
09,359 char *pointerflush, *pointerflushUNALIGN, *BufStart, *Flushing;
09,360 char *pointerflush_64, *pointerflushUNALIGN_64; // r.14++
09,361 char *pointerflush_64_RAM = VerifyBlock; // 2019-Dec-04
09,362 char *pointerflush_64_RESTART; // 2019-Dec-07
09,363
09,364 unsigned long PseudoLinkedPointer, PseudoLinkedPointerNEW, PseudoLinkedPointerROOT, PseudoLinkedPointerNEWold;
09,365 unsigned long PseudoLinkedPointerNEWleft, PseudoLinkedPointerNEWright;
09,366 unsigned long PseudoLinkedPointerNEWmiddle;
09,367 char *bufend[ 806 ]; // 'a'=0, ... 'z'=25 - 26 letters x 31 lengths
09,368 long bufNumberOfWords[ 806 ]; // 'a'=0, ... 'z'=25 - 26 letters x 31 lengths
09,369 // long bufNoWpS[ 806 ] [ 8192 ]; // ?! crashes below when an attempt to use it occur
09,370 char wrd[LongestLineInclusive+1+8]; // 0..30, 31 = 0
09,371 char wrdUP[LongestLineInclusive+1+8]; // 0..30, 31 = 0
09,372 char wrdUPold[LongestLineInclusive+1+8]; // 0..30, 31 = 0
09,373 char LINE10[257]; // 000..255, 256 = 0
09,374 char ZEROS[4]; // 0..3, 0 = 0, 1 = 0, 2 = 0, 3 = 0
09,375 char CRdLFa[2]; // 0..1, 0 = 13, 1 = 10
09,376 unsigned char workbyte; // unsigned in order to index ASCII
09,377 char workK[1024*128];
09,378 long workKoffset = -1;
09,379 unsigned long long FoundInLinkedList, Slot; //r.18
09,380 unsigned long long FoundInLinkedList_RAM, Slot_RAM; // 2019-Dec-04
09,381 unsigned long OffsetsInBuffer[31]; // 00..30
09,382 unsigned long MAXusedBuffer[32]; // 00 not used, only 01..31
09,383 unsigned long GRMBLhill[32]; // 00..31
09,384 unsigned long GRMBLPoolAgain[32]; // 00..31
09,385 int Melnitchka;
09,386 unsigned long MAXusedBufferABS = 0;
09,387 unsigned long Utiliza1 = 0;
09,388 unsigned long Utiliza2 = 0;
09,389 unsigned long TotalWLchars = 0;
09,390
09,391 // GCC 7.3.0 from MINGW complains, so commented them all:
09,392 /* minimum signed 64 bit value */
09,393 // #define _I64_MIN (-9223372036854775807i64 - 1)
09,394 /* maximum signed 64 bit value */
09,395 // #define _I64_MAX 9223372036854775807i64
09,396 /* maximum unsigned 64 bit value */
09,397 // #define _UI64_MAX 0xffffffffffffffffui64
09,398
09,399 /* minimum signed 128 bit value */
09,400 #define _I128_MIN (-170141183460469231731687303715884105727i128 - 1)
09,401 /* maximum signed 128 bit value */
09,402 #define _I128_MAX 170141183460469231731687303715884105727i128
09,403 /* maximum unsigned 128 bit value */
09,404 #define _UI128_MAX 0xffffffffffffffffffffffffffffffffui128
09,405
09,406 char l1TOaDigits[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
09,407 // below duplicates are needed because of one_line_invoking need different buffers.
09,408 char l1TOaDigits2[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
09,409 char l1TOaDigits3[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
09,410 char l1TOaDigits4[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
09,411 unsigned long HEADOffsetFromStartBUKVA = 0;

```

```
09,412 unsigned long TAILOffsetFromStartBUKVA = 0;
09,413
09,414 int SplitOccured;
09,415 int POffsetInLEAF;
09,416 char *Auberge[4] = {"!\\0", "\\0", "-\\0", "\\0"};
09,417 int hashAlfalfa, iAlfalfa;
09,418 int PLE_words=0; // Quadruple!
09,419 char wrd1st[LongestLineInclusive+1+4]; // 0..30, 31 = 0
09,420 char wrd2nd[LongestLineInclusive+1+4]; // 0..30, 31 = 0
09,421 char wrd3rd[LongestLineInclusive+1+4]; // 0..30, 31 = 0
09,422 char wrd4th[LongestLineInclusive+1+4]; // 0..30, 31 = 0
09,423 char wrd5th[LongestLineInclusive+1+4]; // 0..30, 31 = 0
09,424 char wrd6th[LongestLineInclusive+1+4]; // 0..30, 31 = 0
09,425 char wrd7th[LongestLineInclusive+1+4]; // 0..30, 31 = 0
09,426 char wrd8th[LongestLineInclusive+1+4]; // 0..30, 31 = 0
09,427 char wrd9th[LongestLineInclusive+1+4]; // 0..30, 31 = 0
09,428 char wrd10th[LongestLineInclusive+1+4]; // 0..30, 31 = 0
09,429 char *DelimiterUnderscore = "_\\0";
09,430 int PLE_words_INITflag = 0;
09,431
09,432 // QuickSortExternal_4+GB [
09,433 unsigned long long ThunderwithL64_L14;
09,434 unsigned long long Strnglen64_L14;
09,435 unsigned long long size_in64_L14, size_in2_L14;
09,436
09,437 //unsigned long long size_in64_L14_RAM = SourceSize; //2019-Dec-04
09,438 unsigned long long size_in64_L14_RAM = SourceSize+512+32*1024*1024; //2019-Dec-17, if the file is e.g. 67 bytes then this pool is not enough, no matter how many
passes!
09,439
09,440 unsigned long long Over4billionLines, j_Over4billion;
09,441 char OneChar_ieByte = '\\0';
09,442 char CR_ieByte = '\\r';
09,443 char SomeByte;
09,444 unsigned long long BufEnd_64;
09,445 unsigned long long BufEnd_64_RAM; // 2019-Dec-04
09,446 unsigned long long BufEnd_64_RESTART; // 2019-Dec-04
09,447 unsigned long long SeekPosition;
09,448 unsigned long long *PointerToSeekPosition;
09,449 char FourGram[LongestLineInclusive+2]; // 31 longest 4-gram + CR + LF
09,450 char *PoolPhysical;
09,451 unsigned long long fsetpos_ZERO=0;
09,452 char OneCkusterZEROES[1024*4]; // Caution: must be ZEROed(NULLified)!
09,453 char *FileSwapTag = "LEPRECHAUNISH";
09,454 char EOFcode = 0x1A;
09,455 unsigned long long PseudoLinkedPointer_64, PseudoLinkedPointerNEW_64, PseudoLinkedPointerROOT_64, PseudoLinkedPointerNEWold_64;
09,456 unsigned long long PseudoLinkedPointerNEWleft_64, PseudoLinkedPointerNEWright_64;
09,457 unsigned long long PseudoLinkedPointerNEWmiddle_64;
09,458 unsigned long long NULLs_64 = 0;
09,459 unsigned long long PseudoLinkedPointerAUX_64;
09,460 unsigned long long PseudoLinkedPointerAUXdumbo_64;
09,461 char wrdAUX[LongestLineInclusive+1+8]; // 0..30, 31 = 0
09,462 // QuickSortExternal_4+GB ]
09,463
09,464 //unsigned long CounterOccurrences;
09,465 unsigned long long CounterOccurrences; // 2019-Dec-04
09,466 unsigned long long NumberOfLEAFs=0;
09,467 unsigned long LevelsInCorona_Not_Counting_ROOT=0;
09,468 char *ngram[11] =
```

```
{ "NULLleton\0", "singleton\0", "doubleton\0", "tripleton\0", "quadrupleton\0", "quintupleton\0", "sextupleton\0", "septupleton\0", "octupleton\0", "nonupleton\0", "decupleton\0"};
09,469
09,470 unsigned long RipPasses;
09,471 unsigned long long NULLsForWRD=0;
09,472
09,473 int NewOrder;
09,474     char LINE10_NO_DUMP[257]; // 000..255, 256 = 0
09,475
09,476 char *TwoDigitHEXlist[256] = {
09,477 "00\0",
09,478 "01\0",
09,479 "02\0",
09,480 "03\0",
09,481 "04\0",
09,482 "05\0",
09,483 "06\0",
09,484 "07\0",
09,485 "08\0",
09,486 "09\0",
09,487 "0A\0",
09,488 "0B\0",
09,489 "0C\0",
09,490 "0D\0",
09,491 "0E\0",
09,492 "0F\0",
09,493 "10\0",
09,494 "11\0",
09,495 "12\0",
09,496 "13\0",
09,497 "14\0",
09,498 "15\0",
09,499 "16\0",
09,500 "17\0",
09,501 "18\0",
09,502 "19\0",
09,503 "1A\0",
09,504 "1B\0",
09,505 "1C\0",
09,506 "1D\0",
09,507 "1E\0",
09,508 "1F\0",
09,509 "20\0",
09,510 "21\0",
09,511 "22\0",
09,512 "23\0",
09,513 "24\0",
09,514 "25\0",
09,515 "26\0",
09,516 "27\0",
09,517 "28\0",
09,518 "29\0",
09,519 "2A\0",
09,520 "2B\0",
09,521 "2C\0",
09,522 "2D\0",
09,523 "2E\0",
09,524 "2F\0",
09,525 "30\0",
```

09,526 "31\0",
09,527 "32\0",
09,528 "33\0",
09,529 "34\0",
09,530 "35\0",
09,531 "36\0",
09,532 "37\0",
09,533 "38\0",
09,534 "39\0",
09,535 "3A\0",
09,536 "3B\0",
09,537 "3C\0",
09,538 "3D\0",
09,539 "3E\0",
09,540 "3F\0",
09,541 "40\0",
09,542 "41\0",
09,543 "42\0",
09,544 "43\0",
09,545 "44\0",
09,546 "45\0",
09,547 "46\0",
09,548 "47\0",
09,549 "48\0",
09,550 "49\0",
09,551 "4A\0",
09,552 "4B\0",
09,553 "4C\0",
09,554 "4D\0",
09,555 "4E\0",
09,556 "4F\0",
09,557 "50\0",
09,558 "51\0",
09,559 "52\0",
09,560 "53\0",
09,561 "54\0",
09,562 "55\0",
09,563 "56\0",
09,564 "57\0",
09,565 "58\0",
09,566 "59\0",
09,567 "5A\0",
09,568 "5B\0",
09,569 "5C\0",
09,570 "5D\0",
09,571 "5E\0",
09,572 "5F\0",
09,573 "60\0",
09,574 "61\0",
09,575 "62\0",
09,576 "63\0",
09,577 "64\0",
09,578 "65\0",
09,579 "66\0",
09,580 "67\0",
09,581 "68\0",
09,582 "69\0",
09,583 "6A\0",

09,584 "6B\0",
09,585 "6C\0",
09,586 "6D\0",
09,587 "6E\0",
09,588 "6F\0",
09,589 "70\0",
09,590 "71\0",
09,591 "72\0",
09,592 "73\0",
09,593 "74\0",
09,594 "75\0",
09,595 "76\0",
09,596 "77\0",
09,597 "78\0",
09,598 "79\0",
09,599 "7A\0",
09,600 "7B\0",
09,601 "7C\0",
09,602 "7D\0",
09,603 "7E\0",
09,604 "7F\0",
09,605 "80\0",
09,606 "81\0",
09,607 "82\0",
09,608 "83\0",
09,609 "84\0",
09,610 "85\0",
09,611 "86\0",
09,612 "87\0",
09,613 "88\0",
09,614 "89\0",
09,615 "8A\0",
09,616 "8B\0",
09,617 "8C\0",
09,618 "8D\0",
09,619 "8E\0",
09,620 "8F\0",
09,621 "90\0",
09,622 "91\0",
09,623 "92\0",
09,624 "93\0",
09,625 "94\0",
09,626 "95\0",
09,627 "96\0",
09,628 "97\0",
09,629 "98\0",
09,630 "99\0",
09,631 "9A\0",
09,632 "9B\0",
09,633 "9C\0",
09,634 "9D\0",
09,635 "9E\0",
09,636 "9F\0",
09,637 "A0\0",
09,638 "A1\0",
09,639 "A2\0",
09,640 "A3\0",
09,641 "A4\0",

09,642 "A5\0",
09,643 "A6\0",
09,644 "A7\0",
09,645 "A8\0",
09,646 "A9\0",
09,647 "AA\0",
09,648 "AB\0",
09,649 "AC\0",
09,650 "AD\0",
09,651 "AE\0",
09,652 "AF\0",
09,653 "B0\0",
09,654 "B1\0",
09,655 "B2\0",
09,656 "B3\0",
09,657 "B4\0",
09,658 "B5\0",
09,659 "B6\0",
09,660 "B7\0",
09,661 "B8\0",
09,662 "B9\0",
09,663 "BA\0",
09,664 "BB\0",
09,665 "BC\0",
09,666 "BD\0",
09,667 "BE\0",
09,668 "BF\0",
09,669 "C0\0",
09,670 "C1\0",
09,671 "C2\0",
09,672 "C3\0",
09,673 "C4\0",
09,674 "C5\0",
09,675 "C6\0",
09,676 "C7\0",
09,677 "C8\0",
09,678 "C9\0",
09,679 "CA\0",
09,680 "CB\0",
09,681 "CC\0",
09,682 "CD\0",
09,683 "CE\0",
09,684 "CF\0",
09,685 "D0\0",
09,686 "D1\0",
09,687 "D2\0",
09,688 "D3\0",
09,689 "D4\0",
09,690 "D5\0",
09,691 "D6\0",
09,692 "D7\0",
09,693 "D8\0",
09,694 "D9\0",
09,695 "DA\0",
09,696 "DE\0",
09,697 "DC\0",
09,698 "DD\0",
09,699 "DE\0",

```
09,700 "DF\0",
09,701 "E0\0",
09,702 "E1\0",
09,703 "E2\0",
09,704 "E3\0",
09,705 "E4\0",
09,706 "E5\0",
09,707 "E6\0",
09,708 "E7\0",
09,709 "E8\0",
09,710 "E9\0",
09,711 "EA\0",
09,712 "EB\0",
09,713 "EC\0",
09,714 "ED\0",
09,715 "EE\0",
09,716 "EF\0",
09,717 "F0\0",
09,718 "F1\0",
09,719 "F2\0",
09,720 "F3\0",
09,721 "F4\0",
09,722 "F5\0",
09,723 "F6\0",
09,724 "F7\0",
09,725 "F8\0",
09,726 "F9\0",
09,727 "FA\0",
09,728 "FB\0",
09,729 "FC\0",
09,730 "FD\0",
09,731 "FE\0",
09,732 "FF\0"
09,733 };
09,734
09,735 /*
09,736 #if defined(InternalRAM)
09,737 BSTorBtree = 3; //Internal
09,738 #endif
09,739 #if defined(ExternalRAM)
09,740 BSTorBtree = 2; //External
09,741 #endif
09,742 */
09,743
09,744 uint64_t PointerToNotLoadedYet;
09,745 uint64_t jj,jjj, kk, BuildingBlocksSTRIDE;
09,746 int mm;
09,747 //define MatchLensNUM 8
09,748 //int MatchLens[MatchLensNUM]={4,6,8,10,12,14,16,18};
09,749 uint64_t GettingIndexFromArray;
09,750 uint64_t NotFoundKeys=0, FoundKeys=0;
09,751 unsigned long long WORDcountBOTTOMPerMatchLen;
09,752 int strFLAG;
09,753 int KeySize;
09,754 uint64_t AllSlots;
09,755 unsigned long long BUGGYoffset; // fix for the bug in r.17 and prior, r.18
09,756
09,757 Thunderwith=RAMpoolInKB_GLOBAL;
```

```

09,758 printf ("Leprechaun: Memory pool for B-trees is %s MB.\n", _ui64toaKAZEcomma(Thunderwith, 11TOaDigits2, 10) );
09,759
09,760 if (BSTorBtree == 3)
09,761 {
09,762 if (HashInBITS_GLOBAL<3<10)
09,763 printf ("Leprechaun: In this revision %sbytes %d-way hash is used which results in %d x %s internal B-Trees of order 3.\n",
    _ui64toaKAZEcomma((((1LL)<<HashInBITS_GLOBAL)<<3), 11TOaDigits, 10), MatchLensNUM, MatchLensNUM, _ui64toaKAZEcomma(((1LL)<<HashInBITS_GLOBAL) ), 11TOaDigits2, 10) );
09,764 else if (HashInBITS_GLOBAL+3>=10 && HashInBITS_GLOBAL+3<20)
09,765 printf ("Leprechaun: In this revision %sKB %d-way hash is used which results in %d x %s internal B-Trees of order 3.\n", _ui64toaKAZEcomma(
    (((1LL)<<HashInBITS_GLOBAL)<<3))>>10, 11TOaDigits, 10), MatchLensNUM, MatchLensNUM, _ui64toaKAZEcomma(((1LL)<<HashInBITS_GLOBAL) ), 11TOaDigits2, 10) );
09,766 else
09,767 printf ("Leprechaun: In this revision %sMB %d-way hash is used which results in %d x %s internal B-Trees of order 3.\n", _ui64toaKAZEcomma(
    (((1LL)<<HashInBITS_GLOBAL)<<3))>>20, 11TOaDigits, 10), MatchLensNUM, MatchLensNUM, _ui64toaKAZEcomma(((1LL)<<HashInBITS_GLOBAL) ), 11TOaDigits2, 10) );
09,768 } else if (BSTorBtree == 2){
09,769 if (HashInBITS_GLOBAL+3<10)
09,770 printf ("Leprechaun: In this revision %sbytes %d-way hash is used which results in %d x %s external B-Trees of order 3.\n",
    _ui64toaKAZEcomma((((1LL)<<HashInBITS_GLOBAL)<<3), 11TOaDigits, 10), MatchLensNUM, MatchLensNUM, _ui64toaKAZEcomma(((1LL)<<HashInBITS_GLOBAL) ), 11TOaDigits2, 10) );
09,771 else if (HashInBITS_GLOBAL+3>=10 && HashInBITS_GLOBAL+3<20)
09,772 printf ("Leprechaun: In this revision %sKB %d-way hash is used which results in %d x %s external B-Trees of order 3.\n", _ui64toaKAZEcomma(
    (((1LL)<<HashInBITS_GLOBAL)<<3))>>10, 11TOaDigits, 10), MatchLensNUM, MatchLensNUM, _ui64toaKAZEcomma(((1LL)<<HashInBITS_GLOBAL) ), 11TOaDigits2, 10) );
09,773 else
09,774 printf ("Leprechaun: In this revision %sMB %d-way hash is used which results in %d x %s external B-Trees of order 3.\n", _ui64toaKAZEcomma(
    (((1LL)<<HashInBITS_GLOBAL)<<3))>>20, 11TOaDigits, 10), MatchLensNUM, MatchLensNUM, _ui64toaKAZEcomma(((1LL)<<HashInBITS_GLOBAL) ), 11TOaDigits2, 10) );
09,775 }
09,776
09,777 // Below comment is due to adding auto-setting passes, 2019-Dec-07
09,778 /*
09,779 if (HashInBITS_GLOBAL-HashChunkSizeInBITS_GLOBAL==0)
09,780 printf ("Leprechaun: In this revision, %s pass is to be executed.\n", _ui64toaKAZEcomma(1<<<(HashInBITS_GLOBAL-HashChunkSizeInBITS_GLOBAL), 11TOaDigits, 10));
09,781 else
09,782 printf ("Leprechaun: In this revision, %s passes are to be executed.\n", _ui64toaKAZEcomma(1<<<(HashInBITS_GLOBAL-HashChunkSizeInBITS_GLOBAL), 11TOaDigits, 10));
09,783 */
09,784
09,785 // 16fixfix [
09,786 PLE_words_INITflag = 0;
09,787 PLE_words = 0;
09,788 // 16fixfix ]
09,789     Melnitchka = 0;
09,790     WORDcount = 0; // Total word count i.e. for all files!
09,791     WORDcountDistinct = 0;
09,792     NumberOfFiles = 0;
09,793     NumberOfLines = 0;
09,794     FilesLEN = 0;
09,795     LINE10len = 0;
09,796 // Added in r.14+++++FIXFIX [
09,797     NumberOfTrees=0; NumberOfHashCollisions=0;
09,798     NumberOfLEAFs=0;
09,799     WORDcountAttemptsToPut=0;
09,800     LevelsInCorona_Not_Counting_ROOT=0;
09,801 // Added in r.14+++++FIXFIX ]
09,802
09,803 if( ( fp_outLOG = fopen( "Leprechaun.LOG", "a+" ) ) == NULL )
09,804 { printf( "Leprechaun: Can't open file Leprechaun.LOG.\n" ); exit( 7 ); }
09,805
09,806 //     printf( "Leprechaun: Allocating HASH memory %s bytes ... ", _ui64toaKAZEcomma( ((1<<<HashInBITS)*8) + 1 + 64 , 11TOaDigits, 10) );
09,807 //     pointerflushUNALIGN = (char *)malloc( (1<<<HashInBITS)*8 + 1 + 64 );
09,808
09,809 printf( "Leprechaun: Allocating HASH memory %s bytes ... ", _ui64toaKAZEcomma( (uint64_t)(MatchLensNUM+1)*(uint64_t)((1LL)<<HashInBITS_GLOBAL) )*8 + 1 + 64 ,

```



```

11TOaDigits, 10) ); // +1 for the PASS #1 // 2019-Dec-04
09,810 pointerflushUNALIGN = (char *)malloc( (uint64_t)(MatchLensNUM+1)*(uint64_t)((1LL)<<HashInBITS_GLOBAL) *8 + 1 + 64 ); // +1 for the PASS #1 // 2019-Dec-04
09,811
09,812 if( pointerflushUNALIGN == NULL )
09,813 { puts( "\nLeprechaun: Needed memory allocation denied!\n" ); exit( 7 ); }
09,814 pointerflush = pointerflushUNALIGN + 64 - (((size_t)pointerflushUNALIGN) % 64); // 13_6+
09,815 //offset=64-int((long)data&63);
09,816 printf( "OK\n");
09,817
09,818 GLOBAL_HASHPOT = pointerflush; //btree matchfinder needed
09,819
09,820 //      memset(pointerflush,0,17210368*8);
09,821 memset(pointerflush,0,(uint64_t)(MatchLensNUM+1)*(uint64_t)((1LL)<<HashInBITS_GLOBAL) *8); // 2019-Dec-04
09,822 if (BSTorBtree == 2) {
09,823 if( ( fp_outRG = fopen( "Leprechaun_64bit.swp", "wb+" ) ) == NULL )
09,824 { printf( "Leprechaun: Can't create file 'Leprechaun_64bit.swp'.\n" ); exit( 7 ); }
09,825 // Tag for the swap file is: LEPRECHAUNISH{ASCIIcode26}
09,826 // or 14bytes, then when type of the swap is requested:
09,827 // D:\KAZE~1\LEPREC~1>type Leprechaun_64bit.swp
09,828 // LEPRECHAUNISH
09,829 // D:\KAZE~1\LEPREC~1>
09,830 size_in64_L14 = 1024LL * 1024 * (unsigned long long)Thunderwith + 14;
09,831 BufEnd_64 = 0+14;
09,832 // The tag plays two roles, the second to avoid existence of SeekPosition equal to 0. The 0 cannot be used as a free slot FLAG without the TAG.
09,833 printf( "Leprechaun: Allocating/ZEROing %s bytes swap file ... ", _ui64toaKAZEcomma(size_in64_L14, 11TOaDigits, 10) );
09,834 fsetpos(fp_outRG, (const fpos_t *)&fsetpos_ZERO); // SOMETHING ROTTEN with lseeki64/fseeko and fsetpos ???! So DO-IT-OVER.
09,835 memset(OneCkusterZEROES,0,1024*4);
09,836 for (ThunderwithL64_L14=0; ThunderwithL64_L14 < size_in64_L14/(1024*4); ThunderwithL64_L14++)
09,837     fwrite(OneCkusterZEROES, 1024*4, 1, fp_outRG);
09,838 for (ThunderwithL64_L14=0; ThunderwithL64_L14 < size_in64_L14%(1024*4); ThunderwithL64_L14++)
09,839     fwrite(&OneChar_ieByte, 1, 1, fp_outRG);
09,840 fsetpos(fp_outRG, (const fpos_t *)&fsetpos_ZERO); // SOMETHING ROTTEN with lseeki64/fseeko and fsetpos ???! So DO-IT-OVER.
09,841 fwrite(FileSwapTag, 13, 1, fp_outRG);
09,842 fwrite(&EOFCODE, 1, 1, fp_outRG);
09,843 fsetpos(fp_outRG, (const fpos_t *)&BufEnd_64); // SOMETHING ROTTEN with lseeki64/fseeko and fsetpos ???! So DO-IT-OVER.
09,844 } else { // ##### 64bit memory manipulations [
09,845 size_in64_L14 = 1024LL * 1024 * (unsigned long long)Thunderwith + 14 + 1 + 64;
09,846 printf( "Leprechaun: Allocating memory for B-trees %lu MB ... ", (size_in64_L14>>20)+1 );
09,847 pointerflushUNALIGN_64 = (char *)malloc( size_in64_L14 );
09,848 memset(pointerflushUNALIGN_64,0,size_in64_L14);
09,849 if( pointerflushUNALIGN_64 == NULL )
09,850 { puts( "\nLeprechaun: Needed memory allocation denied!\n" ); exit( 7 ); }
09,851 pointerflush_64 = pointerflushUNALIGN_64 + 64 - (((size_t)pointerflushUNALIGN_64) % 64); // 13_6+
09,852 //offset=64-int((long)data&63);
09,853 //memset(pointerflush_64,0,1024 * (unsigned long long)Thunderwith + 14);
09,854 BufEnd_64 = (unsigned long long)pointerflush_64;
09,855 /*
09,856 printf( "BufEnd_64: %s\n", _ui64toaKAZEcomma(BufEnd_64, 11TOaDigits, 10) );
09,857 printf( "pointerflush_64: %s\n", _ui64toaKAZEcomma(pointerflush_64, 11TOaDigits, 10) );
09,858 pointerflush_64 = (char *)BufEnd_64;
09,859 printf( "pointerflush_64: %s\n", _ui64toaKAZEcomma(pointerflush_64, 11TOaDigits, 10) );
09,860 exit (1);
09,861 //BufEnd_64: 541,261,888
09,862 //pointerflush_64: 541,261,888
09,863 //pointerflush_64: 541,261,888
09,864 */
09,865 } // ##### 64bit memory manipulations ]
09,866 printf( "OK\n");

```

```

09,867 fprintf( fp_outLOG, "Leprechaun report:\n" );
09,868
09,869 // Comment the streamed-read of input file ... [[[
09,870 /*
09,871 if( ( fp_inLINE = fopen( argv[1], "rb" ) ) == NULL )
09,872 { printf( "Leprechaun: Can't open file %s \n", argv[1] ); exit( 7 ); }
09,873
09,874 //fseek( fp_inLINE, 0L, SEEK_END ); //Rev. 12
09,875 //size_inLINE = ftell( fp_inLINE ); //Rev. 12
09,876 //fseek( fp_inLINE, 0L, SEEK_SET ); //Rev. 12
09,877
09,878 #if defined(_WIN32_ENVIRONMENT_)
09,879 // 64bit:
09,880 _lseeki64( fileno(fp_inLINE), 0L, SEEK_END );
09,881 size_inLINESIXFOUR = _telli64( fileno(fp_inLINE) );
09,882 _lseeki64( fileno(fp_inLINE), 0L, SEEK_SET );
09,883 #else
09,884 // 64bit:
09,885 fseeko( fp_inLINE, 0L, SEEK_END );
09,886 size_inLINESIXFOUR = ftello( fp_inLINE );
09,887 fseeko( fp_inLINE, 0L, SEEK_SET );
09,888 #endif // defined(_WIN32_ENVIRONMENT_)
09,889
09,890 printf( "Size of input file: %s\n", _ui64toaKAZEcomma(size_inLINESIXFOUR, 11ToaDigits, 10) );
09,891
09,892 //~~~~~
09,893 wrdlen = 0;
09,894 for( i = 0; i < size_inLINESIXFOUR; i++ )
09,895 {
09,896
09,897 // ~~~~~ Buffering fread [
09,898 if (workKoffset == -1) {
09,899     if (i + 1024*128 < size_inLINESIXFOUR) {
09,900         fread( &workK[0], 1, 1024*128, fp_inLINE );
09,901         workKoffset = 0;
09,902         workbyte = workK[workKoffset];
09,903     } else {
09,904         fread( &workbyte, 1, 1, fp_inLINE );
09,905         //printf("%d, '%d' %s\n", i, workbyte, TwoDigitHEXlist[workbyte]); //So stupid code of mine, the remaining (mod 128*1024) has to be read byte by byte as
09,906         NULL, grmb1!
09,907     }
09,908 } else {
09,909     workKoffset++;
09,910     workbyte = workK[workKoffset];
09,911     if (workKoffset == 1024*128 - 1) workKoffset = -1;
09,912 }
09,913 // ~~~~~ Buffering fread ]
09,914
09,915 //     if( isalpha( workbyte ) )
09,916 //     {
09,917 //         if( wrdlen < 31 )
09,918 //         { wrd[ wrdlen ] = tolower( workbyte ); }
09,919 //         wrdlen++;
09,920 //     }
09,921 //         memcpy( &wrd[ wrdlen ], TwoDigitHEXlist[workbyte], 2 );
09,922 //         wrdlen++;
09,923 //         wrdlen++;

```

```

09,924     } // i 'for'
09,925     //~~~~~
09,926 */
09,927 // Comment the streamed-read of input file ... ]]]
09,928 size_inLINESIXFOUR=SourceSize;
09,929 printf( "Leprechaun: Size of input file: %s\n", _ui64toaKAZEcomma(size_inLINESIXFOUR, 11TOaDigits, 10) );
09,930 printf( "\n");
09,931
09,932     time1=time(NULL); //fix of bigtime
09,933
09,934 HashChunkSizeInBITS_GLOBAL = HashInBITS_GLOBAL; // 2019-Dec-07
09,935
09,936 for (jj=0; jj< MatchLensNUM; jj++) {
09,937
09,938         // BUG IN R.17 and prior - the next 'if' is only for RAM (y/Y) not for SSD (z/Z):
09,939         if (BSTorBtree == 2) { //r.18
09,940             BUGGYoffset = (BufEnd_64-(0+14));
09,941         } else { // ##### 64bit memory manipulations [
09,942             BUGGYoffset = (BufEnd_64-(unsigned long long)pointerflush_64);
09,943         } // ##### 64bit memory manipulations ]
09,944
09,945 BufEnd_64_RESTART = BufEnd_64; // 2019-Dec-07
09,946 pointerflush_64_RESTART = pointerflush_64; // 2019-Dec-07
09,947
09,948 NewAttemptRaisePasses: // 2019-Dec-07
09,949
09,950 BufEnd_64 = BufEnd_64_RESTART; // 2019-Dec-07
09,951 pointerflush_64 = pointerflush_64_RESTART; // 2019-Dec-07
09,952 memset(pointerflush + (uint64_t)(jj)*(uint64_t)((1LL)<<HashInBITS_GLOBAL)*8, 0, (uint64_t)(1)*(uint64_t)((1LL)<<HashInBITS_GLOBAL)*8); // 2019-Dec-04
09,953
09,954 //for( RipPasses = 1-1; RipPasses <= (1<<(HashInBITS-HashChunkSizeInBITS))-1; RipPasses++ )
09,955 //{
09,956 RipPasses = 1-1;
09,957 WhyTheHellForIsNotWorking:
09,958
09,959 if (RipPasses < (1<<(HashInBITS_GLOBAL-HashChunkSizeInBITS_GLOBAL))-1)
09,960     printf( "Leprechaun: Inserting keys/BEs of order %s into B-trees, free RAM in B-tree pool is %s MB; Pass #%s of %s ... \r", _ui64toaKAZEzerocomma(MatchLens[jj],
11TOaDigits2, 10)+(26-3), _ui64toaKAZEzerocomma((size_in64_L14 - BUGGYoffset)>>20, 11TOaDigits, 10)+(26-10), _ui64toaKAZEzerocomma(RipPasses+1, 11TOaDigits3, 10)+(26-7),
_ui64toaKAZEzerocomma((1<<(HashInBITS_GLOBAL-HashChunkSizeInBITS_GLOBAL)), 11TOaDigits4, 10)+(26-7) );
09,961 else
09,962     printf( "Leprechaun: Inserting keys/BEs of order %s into B-trees, free RAM in B-tree pool is %s MB; Pass #%s of %s ... ", _ui64toaKAZEzerocomma(MatchLens[jj],
11TOaDigits2, 10)+(26-3), _ui64toaKAZEzerocomma((size_in64_L14 - BUGGYoffset)>>20, 11TOaDigits, 10)+(26-10), _ui64toaKAZEzerocomma(RipPasses+1, 11TOaDigits3, 10)+(26-7),
_ui64toaKAZEzerocomma((1<<(HashInBITS_GLOBAL-HashChunkSizeInBITS_GLOBAL)), 11TOaDigits4, 10)+(26-7) );
09,963
09,964 // In-here the new lowering MEMORY FOOTPRINT improvement begins:
09,965 // It is from two passes:
09,966 // - the first builds B-trees by putting all keys (unique and non-unique) into leaves;
09,967 // - the second builds B-trees by putting ONLY non-unique keys into leaves;
09,968 // Each pass has its own two POOLs:
09,969 // Pass #1: HASH POOL size:      ((1LL)<<HashInBITS_GLOBAL)*8
09,970 // Pass #1: HASH POOL address:  pointerflush //BufStart + Slot +(MatchLensNUM*(1LL<<HashInBITS_GLOBAL)*8) // MatchLensNUM=10
09,971 // Pass #1: Btree POOL size:    size_in64_L14_RAM = SourceSize
09,972 // Pass #1: Btree POOL address: pointerflush_64_RAM = VerifyBlock
09,973
09,974 // Pass #2: HASH POOL size:      MatchLensNUM*((1LL)<<HashInBITS_GLOBAL)*8
09,975 // Pass #2: HASH POOL address:  pointerflush //BufStart + Slot +(jj*(1LL<<HashInBITS_GLOBAL)*8) // jj=0..9
09,976 // Pass #2: Btree POOL size:    size_in64_L14 = 1024LL * 1024 * (unsigned long long)Thunderwith + 14
09,977 // Pass #2: Btree POOL address: pointerflush_64

```

```

09,978
09,979 // Variable-Differences between two passes:
09,980 // Pass #1: Slot_RAM, BufEnd_64_RAM, size_in64_L14_RAM, pointerflush_64_RAM, BSTorBtree_RAM enforced to be 3 i.e. RAM; Pass #1 has 'counter' uncommented!
09,981 // Pass #2: Slot, BufEnd_64, size_in64_L14, pointerflush_64, BSTorBtree
09,982 // 'FoundInLinkedList_RAM' decides whether to execute B-tree fragment AT ALL in PASS #2! If the key's occurrences are 1+ then FoundInLinkedList_RAM = 1
09,983
09,984 // The ASCII block-scheme of major mumbo-jumbo:
09,985
09,986 /*
09,987 Parsing all the BBs (Building-Blocks) of length 4,6,8,10,12,14,16,18,36,64 at each position:
09,988
09,989 PASS #1 - TEMPORARY STUFF:
09,990 One POOL based/located always on physical (internal RAM) - the last/hidden one of all HASH SUB-POOLS:
09,991 -----
09,992 | Hash BLOCK (TEMPORARY HASH Sub-Pool), size = command line parameter |-----
09,993 |
09,994 |
09,995 | All slots point to B-tree roots in the TEMPORARY B-tree POOL
09,996 |-----
09,997 |
09,998 Three POOLs based/located always on physical (internal RAM): \ /
09,999 |
10,000 | Source Block, size = N | | Source REVERSED Block, size = N | | Target Block, size = N, or, B-trees BLOCK (TEMPORARY) |
10,001 |-----
10,002 PASS #2 - TEMPORARY-TO-NONTEMPORARY STUFF (takes into account what above two temporary blocks house):
10,003 -----
10,004 | Source Pool: Houses the file being compressed |
10,005 |
10,006 | \-----
10,007 | | Hash 64 bytes
10,008 | | Hash 6 bytes
10,009 | | \-----
10,010 | | Hash 4 bytes
10,011 | | \-----
10,012 | | \-----
10,013 | | \-----
10,014 | Hash Sub-Pool for BBs 4 bytes long | Hash Sub-Pool for BBs 6 bytes long | ... | Hash Sub-Pool for BBs 64 bytes long | TEMPORARY HASH Sub-Pool |
10,015 |-----
10,016 | Slot # 0 | ... | Slot # (1<<HashInBITS_GLOBAL)-1 | Slot # 0 | ... | Slot # (1<<HashInBITS_GLOBAL)-1 |
10,017 | Address 8 bytes | ... | Address 8 bytes | Address 8 bytes | ... | Address 8 bytes |
10,018 | to a B-tree root | ... | to a B-tree root | to a B-tree root | ... | to a B-tree root |
10,019 |-----
10,020 |
10,021 |
10,022 | \-----
10,023 | \-----
10,024 | \-----
10,025 | \-----
10,026 | \-----
10,027 | Btree Pool: Houses the leaves of all B-trees | ROOT #1 | ... some leaves/roots ... | ROOT #2 | LEAF #2 | ... some leaves/roots ... | LEAF #3 | LEAF #4 | ... | LEAF #5 | ! Located either on
10,028 | \-----
10,029 | \-----
10,030 | \-----
10,031 | \-----
10,032 | \-----
10,033 | \-----
10,034 B-tree order 3 (MAX 3 pointers, MAX 2 keys) LEAF #1 (ROOT is also a LEAF - if no successors) structure:
10,035 -----
10,036 | LeftPointer, | MiddlePointer, | RightPointer, | DynamicLeftKey (LeftKeySize + LeftKey + LeftQWORD) | DynamicRightKey (RightKeySize + RightKey + RightQWORD) |
10,037 | Address 8 bytes | Address 8 bytes | Address 8 bytes | 1 byte + 1..255 bytes + 8 bytes | 1 byte + 1..255 bytes + 8 bytes |
10,038 |-----
10,039 |
10,040 |
10,041 |
10,042 | \-----
10,043 | \-----
10,044 | \-----
10,045 | LEAF #2 (housing keys smaller than LeftKey) | | LEAF #3 (housing keys between LeftKey and RightKey) | | LEAF #4 (housing keys bigger than RightKey) |
10,046 |-----
10,047
10,048 Note1: Hash Pool: Houses (10+1) consecutive Sub-Pools, the 11th (the last one actually) is used in PASS #1 only (as temporary, its slots point to physical RAM only addresses located in TargetBlock of size
N=SourceSize).
10,049 Note2: Usually 'LeftQWORD' and 'RightQWORD' are used either for OCCURRENCES or LastSeenOffset of that key.

```

10,050 Note3: If 'LeftKeySize' or 'RightKeySize' is 0 then the field is not used - there is no KEY.

10,051

10,052 // Source of 2019-Dec-07 revision is downloadable at: <https://community.centmimod.com/threads/a-lzss-microduplicator-tagetting-huge-texts-with-c-source.16427/#post-80053>

10,053 // And glad to share the latest revision of Nakamichi, featuring:

10,054 // - Needs 3N physical RAM (+ adjustable physical RAM for HASH pools), where N is the size of file being compressed;

10,055 // - Automatic setting of PASSES (allows to fit building B-trees into RAM (the third N used to house output file) and just then to rebuild the current PASS either on RAM or SSD) - it lowers drastically the wearing of external RAM;

10,056 // - Significantly Lowered Memory Footprint - now, only the NON-UNIQUE keys are put into B-trees.

10,057 //

10,058

10,059 // PASS #1: [

10,060

10,061 // INITIALIZE:

10,062 // Below line is zeroing the LAST HASH SUB-POOL for RAM RIP:

10,063 memset(pointerflush + (uint64_t)(MatchLensNUM)*(uint64_t)((1LL)<<HashInBITS_GLOBAL)*8, 0, (uint64_t)(1)*(uint64_t)((1LL)<<HashInBITS_GLOBAL)*8); // 2019-Dec-04

10,064 // Below line is resetting Btree POOL for RAM RIP:

10,065 BufEnd_64_RAM = (unsigned long long)pointerflush_64_RAM;

10,066 memset(pointerflush_64_RAM, 0, size_in64_L14_RAM); // Probably no need, have to check whether it is necessary... YES, if commented then it crashes!? Maybe all the B-tree building/init causes the worse ratio compared to older versions?!

10,067

10,068 for (BuildingBlocksSTRIDE=0; BuildingBlocksSTRIDE < size_inLINESIXFOUR-MatchLens[jj]+1; BuildingBlocksSTRIDE++) {

10,069 // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed [

10,070 // wrdlen=0;

10,071 // memset(&wrd[0], 0, (LongestLineInclusive+1+8)); //r.18, see below, the old nullifier is commented.

10,072 // for (mm=0; mm< MatchLens[jj]; mm++) {

10,073 // memcpy(&wrd[wrdlen], TwoDigitHEXlist[(unsigned char *) (SourceBlock+mm+BuildingBlocksSTRIDE)], 2);

10,074 // wrdlen++;

10,075 // wrdlen++;

10,076 // }

10,077 // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed]

10,078

10,079 wrdlen=MatchLens[jj];

10,080 memset(&wrd[0], 0, (LongestLineInclusive+1+8)); //r.18, see below, the old nullifier is commented.

10,081

10,082 // if (wrdlen > 28) {

10,083 // FIPS202_SHA3_224((unsigned char *) (SourceBlock+BuildingBlocksSTRIDE), wrdlen, (unsigned char *) SHA328bytes);

10,084 // wrdlen=28;

10,085 // }

10,086

10,087 memcpy(&wrd[0], &wrdlen, 1); // The keysize is 1..255, pretty enough since my needs are 1..28 (up to SHA3-224).

10,088 // if (wrdlen < 28)

10,089 memcpy(&wrd[0+1], (unsigned char *) (SourceBlock+BuildingBlocksSTRIDE), wrdlen);

10,090 // else

10,091 // memcpy(&wrd[0+1], (unsigned char *) SHA328bytes, wrdlen);

10,092

10,093 BufStart = pointerflush;

10,094 // Slot = ((wrd[0]-' ')*28*28*28*28 + (wrd[1]-' ')*28*28*28 + (wrd[2]-' ')*28*28 + (wrd[3]-' ')*28 + (wrd[4]-' '))<<3;

10,095 // Slot = FNV1A_Hash_Jesteress_27bit(wrd, wrdlen)<<3; // Commented since r.14+++ because of passes.

10,096 // Slot = FNV1A_Hash_Jesteress_27bit(wrd +1, wrdlen); WORDcount++;

10,097 Slot = FNV1A_Pippip_Yurii(wrd +1, wrdlen); WORDcount++; // Changed 2019-Nov-28

10,098

10,099 // Bug fix for all r.14+++ and below! [

10,100 //memcpy(&wrd[(LongestLineInclusive+1+4)-4], &NULLsForWRD, 4);

10,101 //memcpy(&wrd[(LongestLineInclusive+1+4)-4-1], &NULLsForWRD, 1);

10,102 // Bug fix for all r.14+++ and below!]

10,103

10,104 // Example: HashInBITS-HashChunkSizeInBITS=2

10,105 // HashInBITS = 5

10,106 // HashChunkSizeInBITS = 3

10,107 // RipPasses = 1<<((HashInBITS-HashChunkSizeInBITS) i.e. 1<<2 which is 4 i.e. 32 slots with 4 passes 8 slots each.

```

10,108 //          00??? 5bits 0-7
10,109 //          01??? 5bits 8-15
10,110 //          10??? 5bits 16-23
10,111 //          11??? 5bits 24-31
10,112 if ( (Slot>>HashChunkSizeInBITS_GLOBAL) == RipPasses ) {
10,113     Slot = Slot<<3;
10,114 // NEW NEW NEW [ //r.18
10,115 // In here Slot is not within a single pool but in MatchLensNUM sub-pools i.e. MatchLensNUM-way i.e. MatchLensNUM hashpots:
10,116 /*
10,117 for (GettingIndexOfArray=0; GettingIndexOfArray<MatchLensNUM; GettingIndexOfArray++) {
10,118     // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed [
10,119     // if ( (wrklen>>1)==MatchLens[GettingIndexOfArray] ) break;
10,120     if ( (wrklen)==MatchLens[GettingIndexOfArray] ) break;
10,121     // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed ]
10,122 }
10,123 */
10,124 GettingIndexOfArray=jj; // same as above atrocity
10,125 Slot = Slot * (MatchLensNUM*(1LL<<HashInBITS_GLOBAL)*8); // CAUTION: 'wrklen' is halved here, when a new revision comes with 1:1 keysize then change it.
10,126 // The line above uses the upper/last part of HASH POOL i.e. for MatchLensNUM=10 11th i.e. 0..9 for pass #2, 10 for pass #1
10,127 // NEW NEW NEW ] //r.18
10,128
10,129 //KeySize = LongestLineInclusive; // 'LongestLineInclusive' is for 1-way hashing where different KeySizes can occupy same HASH pool.
10,130 KeySize = wrklen; //r.18, In case multi-way hashing: either the MAX allowed or the 'wrklen'.
10,131
10,132 //Slot = 0; // One Tree only!
10,133 memcpy( &PseudoLinkedPointer_64, BufStart+Slot, 8 );
10,134
10,135 // ##### B-tree order 3 fragment 64bit [
10,136 //
10,137 // LEAF structure: [LeftPointer][MiddlePointer][RightPointer][LeftWord][RightWord]
10,138 //                  4bytes      4bytes      4bytes      wrklen      wrklen
10,139 //                  *          *                                <- if *(char *)==0 means the word cell is empty
10,140 // ALL B-tree order 3 fragment consists of 3 sub-fragments:
10,141 // 1] Search 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search 3] Insert Iterative
10,142
10,143 // LEAF_64 structure: [LeftPointer][MiddlePointer][RightPointer][LeftWord] [RightWord]
10,144 //                   8bytes      8bytes      8bytes      LongestLineInclusive+1+4 LongestLineInclusive+1+4
10,145 //                   *          *                                <- if *(char *)==0 means the word cell is empty
10,146 // Note: In order to use one fread(and strcmp) a NULL postfix for LeftWord, RightWord i.e. LeftWord_Length=len(LeftWord)+1 a kinda stupid choice ...
10,147 // Note: BufEnd_64_RAM in fact is the first free position after the BUFFER END!
10,148
10,149 // 1] Search [
10,150         if (PseudoLinkedPointer_64 == 0) // means EMPTY-SLOT
10,151         {
10,152             //if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrklen + 4 + 4 + 4 < GRMBLFoolAgain((int)wrklen) ) // +4 more for BST instead of LL; + more(see
10,153             // {
10,154             //     memcpy( BufStart+Slot, &bufend[LetterOffset], 4 );
10,155             //     bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
10,156             //     memcpy( bufend[LetterOffset], wrd, wrklen ); WORDcountDistinct++; bufNumberOfWords[LetterOffset]++;
10,157             //     bufend[LetterOffset] = bufend[LetterOffset] + 2*wrklen;
10,158             //     if (MAXusedBuffer[wrklen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrklen] = (unsigned long)(bufend[LetterOffset] -
BufStart);}
10,159             // }
10,160             // BUG IN R.17 and prior - the next 'if' is only for RAM (y/Y) not for SSD (z/Z):
10,161             if (BSTorBtree_RAM == 2) { //r.18
10,162                 BUGGYoffset = (BufEnd_64_RAM-(0+14));
10,163                 } else { // ##### 64bit memory manipulations [

```

```

10,164 BUGGYoffset = (BufEnd_64_RAM-(unsigned long long)pointerflush_64_RAM);
10,165         } // ##### 64bit memory manipulations ]
10,166         if( 8 + 8 + 8 + 2*(KeySize+1+8) < size_in64_L14_RAM - BUGGYoffset ) // the longest wrdlen is LongestLineInclusive but actual is LongestLineInclusive(+ CR
char)
10,167         {
10,168             memcpy( BufStart+Slot, &BufEnd_64_RAM, 8 );
10,169             BufEnd_64_RAM = BufEnd_64_RAM + 8 + 8 + 8;
10,170             if (BSTorBtree_RAM == 2) {
10,171                 fsetpos(fp_outRG, (const fpos_t *)&BufEnd_64_RAM);
10,172                 //fwrite(wrd, wrdlen, 1, fp_outRG); //GRMEL! r.18
10,173                 fwrite(wrd, (KeySize+1+8), 1, fp_outRG); //GRMEL! r.18
10,174                 WORDcountDistinct++; Total_fwrite++;
10,175                 //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
10,176                 // r.14++ The above line was commented because the pool is already ZEROed.
10,177             } else { // ##### 64bit memory manipulations [
10,178                 //memcpy( (char *)BufEnd_64_RAM, wrd, wrdlen ); //GRMEL! r.18
10,179                 memcpy( (char *)BufEnd_64_RAM, wrd, (KeySize+1+8) ); //GRMEL! r.18
10,180                 WORDcountDistinct++;
10,181             } // ##### 64bit memory manipulations ]
10,182             BufEnd_64_RAM = BufEnd_64_RAM + 2*(KeySize+1+8);
10,183             //fsetpos(fp_outRG, &BufEnd_64_RAM);
10,184         }
10,185     else
10,186     { printf( "\nLeprechaun: Failure! Increment 'Memory for B-trees'!\n" );
10,187 fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, llToaDigits, 10), _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct,
llToaDigits2, 10) );
10,188 fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14_RAM, llToaDigits, 10) );
10,189 fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, llToaDigits, 10) );
10,190 fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
10,191 fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for B-trees'!\n\n");
10,192 if ( HashChunkSizeInBITS_GLOBAL > 0 ) { // 2019-Dec-07
10,193     HashChunkSizeInBITS_GLOBAL--;
10,194     printf( "Automatically increasing number of passes in order to fit B-trees into Target Buffer.\n" );
10,195     goto NewAttemptRaisePasses;
10,196 } else exit( 7 );
10,197     }
10,198     FoundInLinkedList = 1+1;
10,199     }
10,200     else // means USED-SLOT
10,201     { FoundInLinkedList = 0;
10,202     StackPtr = 0;
10,203 // while (PseudoLinkedPointer != 0 && FoundInLinkedList == 0)
10,204     while (PseudoLinkedPointer_64 != 0 && FoundInLinkedList == 0)
10,205     {
10,206 // ***** 'P W P' section [
10,207 // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
10,208 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
10,209 // here ALWAYS LW exists: no need for existence check - line below
10,210 // if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
10,211 // if (memcmp(PseudoLinkedPointer+4+4+4,wrd,wrdlen) > 0) // go LP
10,212 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,213 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
10,214 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,215 //fread(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
10,216 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,217 // [ //r.14+
10,218     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
10,219     if (BSTorBtree_RAM == 2) {

```

```

10,220         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,221         fread(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fread++;
10,222     } else { // ##### 64bit memory manipulations [
10,223         memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64, 8+8+2*(KeySize+1+8) );
10,224     } // ##### 64bit memory manipulations ]
10,225     memcpy( &FourGramL[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
10,226     // ] //r.14+
10,227     //strFLAG=strcmpKAZE13(FourGramL, wrd);
10,228     strFLAG=memcmp(FourGramL+1, wrd +1, wrdlen); // Since 2019-Feb-07, it is not ASCIIZ string but zero byte is the length of the 1..
10,229     if (strFLAG > 0) // go LP
10,230     { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 0, 4 ); //LP
10,231       PseudoLinkedPointer = PseudoLinkedPointerNEW;
10,232     }
10,233     {
10,234         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,235         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 0; //LP
10,236         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,237         if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
10,238         BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
10,239         BSTstack[StackPtr] = 0; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
10,240         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,241         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,242         //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
10,243         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,244         // [ //r.14+
10,245         memcpy( &PseudoLinkedPointer_64, &LEAF[0], 8 );
10,246         // ] //r.14+
10,247         }
10,248     //     else if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wrdlen) < 0) // go RP or MP
10,249     //     else if (strFLAG < 0) // go RP or MP
10,250     //     { // RW existence check - line below:
10,251     //         if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 ) // RW exists
10,252     //         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,253     //         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
10,254     //         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,255     //         //fread(&SomeByte, 1, 1, fp_outRG);
10,256     //         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,257     //         // [ //r.14+
10,258     //         memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (KeySize+1+8)], 1 );
10,259     //         // ] //r.14+
10,260     //         if (SomeByte != 0) // RW exists
10,261     //             { // Here all 'P W P' section is repeated; the way of handling case when dynamic number of words in leaf
10,262     // *****
10,263     // ***** 'P W P' section 2 [
10,264     // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
10,265     // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
10,266     //     // here ALWAYS RW exists: no need for existence check - line below
10,267     //     // if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
10,268     //     if (memcmp(PseudoLinkedPointer+4+4+4+wrdlen, wrd, wrdlen) > 0) // go MP
10,269     //     // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,270     //     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
10,271     //     //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,272     //     //fread(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
10,273     //     // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,274     //     // [ //r.14+
10,275     //     memcpy( &FourGramL[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
10,276     //     // ] //r.14+
10,277     //     //strFLAG=strcmpKAZE13(FourGramL, wrd);

```



```

10,278         strFLAG=memcmp(FourGramL+1, wrd +1, wrdlen); // Since 2019-Feb-07, it is not ASCIIZ string but zero byte is the length of the 1..
10,279         if (strFLAG > 0) // go MP
10,280         {
10,281             memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
10,282             PseudoLinkedPointer = PseudoLinkedPointerNEW;
10,283         }
10,284         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,285         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8; //MP
10,286         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,287         if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
10,288         BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
10,289         BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
10,290         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,291         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,292         //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
10,293         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,294         // [ //r.14+
10,295         memcpy( &PseudoLinkedPointer_64, &LEAF[8], 8 );
10,296         // ] //r.14+
10,297         }
10,298         else if (memcmp(PseudoLinkedPointer+4+4+4+wrdlen, wrd, wrdlen) < 0) // go RP
10,299         else if (strFLAG < 0) // go RP
10,300         { // No ?W after RW - go RP
10,301             memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4 + 4, 4 ); //RP
10,302             PseudoLinkedPointer = PseudoLinkedPointerNEW;
10,303         }
10,304         {
10,305             // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,306             //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8; //RP
10,307             // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,308             if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
10,309             BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
10,310             BSTstack[StackPtr] = 16; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
10,311             // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,312             //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,313             //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
10,314             // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,315             // [ //r.14+
10,316             memcpy( &PseudoLinkedPointer_64, &LEAF[8 + 8], 8 );
10,317             // ] //r.14+
10,318             }
10,319         else { FoundInLinkedList = 1; // wrd is RW
10,320         // Counter [
10,321             if (BSTorBtree_RAM == 2) {
10,322                 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,323             }
10,324             memcpy( &CounterOccurrences, &FourGramL[(KeySize+1+8)-8], 8 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
10,325             if (CounterOccurrences==0) CounterOccurrences++; // Fixing an old bug, counting starts from 0 since this is 'search' not 'insert' - it
was already inserted i.e. with 1 occurrence but the field is ZERO when put initially. Fix: 2019-Dec-19
10,326             if (CounterOccurrences<999999999) CounterOccurrences++; // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
10,327             memcpy( &FourGramL[(KeySize+1+8)-8], &CounterOccurrences, 8 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
10,328
10,329         // Debug the ratio difference [
10,330             if (CounterOccurrences>1) {
10,331                 printf("\nPASS #1: BuildingBlocksSTRIDE, jj = %s, %s\n", _ui64toaKAZEzerocomma4(BuildingBlocksSTRIDE, 11TOaDigits3, 16)+(26-8-1),
_ui64toaKAZEzerocomma4(jj, 11TOaDigits2, 10)+(26-2));
10,332             }
10,333         // Debug the ratio difference ]

```

```

10,334
10,335 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,336 //fwrite(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
10,337 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,338 // [ //r.14+
10,339 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &FourGramL[0], (KeySize+1+8) );
10,340 // if (BSTorBtree_RAM == 2) {
10,341 fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
10,342 } else { // ##### 64bit memory manipulations [
10,343 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
10,344 } // ##### 64bit memory manipulations ]
10,345 // ] //r.14+
10,346 // Counter ]
10,347 }
10,348 WORDcountAttemptsToPut++;
10,349 // ***** 'P W P' section 2 ]
10,350 // *****
10,351 }
10,352 else // RW empty - go MP
10,353 { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
10,354 PseudoLinkedPointer = PseudoLinkedPointerNEW;
10,355 }
10,356 {
10,357 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,358 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8; //MP
10,359 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,360 if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
10,361 BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
10,362 BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
10,363 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,364 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,365 //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
10,366 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,367 // [ //r.14+
10,368 memcpy( &PseudoLinkedPointer_64, &LEAF[8], 8 );
10,369 // ] //r.14+
10,370 }
10,371 }
10,372 else { FoundInLinkedList = 1; // wrd is LW
10,373 // Counter [
10,374 if (BSTorBtree_RAM == 2) {
10,375 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,376 }
10,377 memcpy( &CounterOccurrences, &FourGramL[(KeySize+1+8)-8], 8 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
10,378 if (CounterOccurrences==0) CounterOccurrences++; // Fixing an old bug, counting starts from 0 since this is 'search' not 'insert' - it
was already inserted i.e. with 1 occurrence but the field is ZERO when put initially. Fix: 2019-Dec-19
10,379 if (CounterOccurrences<999999999) CounterOccurrences++; // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
10,380 memcpy( &FourGramL[(KeySize+1+8)-8], &CounterOccurrences, 8 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
10,381
10,382 // Debug the ratio difference [
10,383 // if (CounterOccurrences>1) {
10,384 // printf("\nPASS #1: BuildingBlocksSTRIDE, jj = %s, %s\n", _ui64toaKAZEzerocomma4(BuildingBlocksSTRIDE, 11TOaDigits3, 16)+(26-8-1),
_ui64toaKAZEzerocomma4(jj, 11TOaDigits2, 10)+(26-2));
10,385 // }
10,386 // Debug the ratio difference ]
10,387
10,388 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,389 //fwrite(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);

```

```

10,390 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,391 // [ //r.14+
10,392 memcpy( &LEAF[8 + 8 + 8], &FourGramL[0], (KeySize+1+8) );
10,393 if (BSTorBtree_RAM == 2) {
10,394 fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
10,395 } else { // ##### 64bit memory manipulations [
10,396 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
10,397 } // ##### 64bit memory manipulations ]
10,398 // ] //r.14+
10,399 // Counter ]
10,400 }
10,401 WORDcountAttemptsToPut++;
10,402 // ***** 'P W P' section ]
10,403 } // while
10,404 WORDcountAttemptsToPut--; // - 1 due to BST way of counting i.e. direct hash hit is not counted only successors
10,405 }
10,406 // 1] Search ]
10,407 if (FoundInLinkedList == 0) NotFoundKeys++; //r.18
10,408 if (FoundInLinkedList == 1) FoundKeys++; //r.18
10,409 if (FoundInLinkedList == 2) NotFoundKeys++; //r.18
10,410
10,411 if (FoundInLinkedList == 0)
10,412 {
10,413 /*
10,414 // ===== [ The whole section/sub-fragment 2 is commented due to great time differences for Internal_vs_External memory accesses - it is far more
cheap to have the STACK overhead (moved to sub-fragment 1) ] ===== [
10,415 // 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search [
10,416 // 'TracingSearch' is the same as 'Search' except that adds the trail in my simulated stack,
10,417 // the goal is not to waste time in 'Search' by dealing with no needed trail in case of not 'Insert'.
10,418 // Simulated stack contains pairs of 'Address of ParentLEAF' + 'Offset of ParentPointer in ParentLEAF i.e. 0 for LP, 4 for MP, 8 for RP'.
10,419 // 'Offset ...' saves unnecessary comparisons of NEWword which after splitting goes up.
10,420 memcpy( &PseudoLinkedPointer, BufStart+Slot, 4 );
10,421 StackPtr = 0;
10,422 while (PseudoLinkedPointer != 0)
10,423 {
10,424 // ***** 'P W P' section [
10,425 // LW: existence check if ( *(char *)(&PseudoLinkedPointer+4+4+4) != 0 )
10,426 // RW: existence check if ( *(char *)(&PseudoLinkedPointer+4+4+4+wordlen) != 0 )
10,427 // here ALWAYS LW exists: no need for existence check - line below
10,428 // if ( *(char *)(&PseudoLinkedPointer+4+4+4) != 0 )
10,429 if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wordlen) > 0) // go LP
10,430 { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 0, 4 ); //LP
10,431 if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
10,432 BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
10,433 BSTstack[StackPtr] = 0; ++StackPtr; //LPoffset=0;MPoffset=4;RPoffset=8;
10,434 PseudoLinkedPointer = PseudoLinkedPointerNEW;
10,435 }
10,436 else if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wordlen) < 0) // go RP or MP
10,437 { // RW existence check - line below:
10,438 if ( *(char *)(&PseudoLinkedPointer+4+4+4+wordlen) != 0 ) // RW exists
10,439 { // Here all 'P W P' section is repeated; the way of handling case when dynamic number of words in leaf
10,440 // =====
10,441 // ***** 'P W P' section 2 [
10,442 // LW: existence check if ( *(char *)(&PseudoLinkedPointer+4+4+4) != 0 )
10,443 // RW: existence check if ( *(char *)(&PseudoLinkedPointer+4+4+4+wordlen) != 0 )
10,444 // here ALWAYS RW exists: no need for existence check - line below
10,445 // if ( *(char *)(&PseudoLinkedPointer+4+4+4+wordlen) != 0 )
10,446 if (memcmp(PseudoLinkedPointer+4+4+4+wordlen, wrd, wordlen) > 0) // go MP

```

```

10,447         { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
10,448         if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
10,449         BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
10,450         BSTstack[StackPtr] = 4; ++StackPtr; //LPoffset=0;MPoffset=4;RPoffset=8;
10,451         PseudoLinkedPointer = PseudoLinkedPointerNEW;
10,452     }
10,453     else if (memcmp(PseudoLinkedPointer+4+4+4*wordlen, wrd, wordlen) < 0) // go RP
10,454     { // No ?W after RW - go RP
10,455         memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4 + 4, 4 ); //RP
10,456         if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
10,457         BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
10,458         BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=4;RPoffset=8;
10,459         PseudoLinkedPointer = PseudoLinkedPointerNEW;
10,460     }
10,461     else FoundInLinkedList = 1; // wrd is RW
10,462 // ***** 'P W P' section 2 ]
10,463 // *****
10,464     }
10,465     else // RW empty - go MP
10,466     { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
10,467         if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
10,468         BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
10,469         BSTstack[StackPtr] = 4; ++StackPtr; //LPoffset=0;MPoffset=4;RPoffset=8;
10,470         PseudoLinkedPointer = PseudoLinkedPointerNEW;
10,471     }
10,472     }
10,473     else FoundInLinkedList = 1; // wrd is LW
10,474 // ***** 'P W P' section ]
10,475     } // while
10,476 // 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search ]
10,477 // ===== [ The whole section/sub-fragment 2 is commented due to great time differences for Internal_vs_External memory accesses - it is far more
10,478 // cheap to have the STACK overhead (moved to sub-fragment 1) ] =====
10,479 */
10,480 // 3] Insert Iterative [
10,481 // There are total 4 situations:
10,482 // Case #1: Outer NODE(including ROOT) [ ][ ][ ][LW][ ]
10,483 // Case #2: Outer NODE(including ROOT) [ ][ ][ ][LW][RW] Split Occurs ----
10,484 // Case #3: ROOT [LP][MP][ ][LW][ ] | 'wrUP' (wordlen bytes)
10,485 // Case #4: Inner NODE(including ROOT) [LP][MP][RP][LW][RW] Split Occurs --- | &
10,486 // | | 'PseudoLinkedPointerNEW' (ptr to NEW LEAF)
10,487 // There are total 2 situations for PARENT LEAF: <----- ARE GOING UP
10,488 // Case #3: [LP][MP][ ][LW][ ]
10,489 // Case #4: [LP][MP][RP][LW][RW] Split Occurs
10,490
10,491 // ~ First deal alongly with the OUTER NODE(LEAF) where Search stopped i.e Case #1 & Case #2:
10,492     PoffsetInLEAF = BSTstack[--StackPtr];
10,493     PseudoLinkedPointer_64 = BSTstack[--StackPtr];
10,494 // NOTE: ONE LEAF IS FULL ONLY WHEN LAST CELL FOR KEY(here RW) EXISTS!
10,495 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4*wordlen) != 0 )
10,496 //if ( *(char *) (PseudoLinkedPointer+4+4+4*wordlen) != 0 ) // If LEAF is full: Case #2
10,497     // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,498     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
10,499     //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,500     //fread(&SomeByte, 1, 1, fp_outRG);
10,501     // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,502     // [ //r.14+
10,503     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;

```

```

10,504         if (BSTorBtree_RAM == 2) {
10,505             fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,506             fread(&LEAF[0], 8*8+8*2*(KeySize+1+8), 1, fp_outRG); Total_fread++;
10,507             } else { // ##### 64bit memory manipulations [
10,508             memcpy( &LEAF[0], (char *)&PseudoLinkedPointerAUX_64, 8*8+8*2*(KeySize+1+8) );
10,509             } // ##### 64bit memory manipulations ]
10,510             memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (KeySize+1+8)], 1 );
10,511             // ] //r.14+
10,512             if (SomeByte != 0 ) // RW exists
10,513             { SplitOccured = 1; WORDcountDistinct++;
10,514             // ALlocate NEW LEAF:
10,515             // if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrklen + 4 + 4 + 4 < GRMBLFoolAgain((int)wrklen) ) // +4 more for BST instead of LL; + more(see
LEAF)
10,516             // {
10,517             //     memcpy( &PseudoLinkedPointerNEW, &bufend[LetterOffset], 4 );
10,518             //     bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
10,519             //     bufend[LetterOffset] = bufend[LetterOffset] + 2*wrklen;
10,520             //     if (MAXusedBuffer[wrklen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrklen] = (unsigned long)(bufend[LetterOffset] -
BufStart);}
10,521             // }
10,522             // BUG IN R.17 and prior - the next 'if' is only for RAM (y/Y) not for SSD (z/Z):
10,523             if (BSTorBtree_RAM == 2) { //r.18
10,524             BUGGYoffset = (BufEnd_64_RAM-(0+14));
10,525             } else { // ##### 64bit memory manipulations [
10,526             BUGGYoffset = (BufEnd_64_RAM-(unsigned long long)pointerflush_64_RAM);
10,527             } // ##### 64bit memory manipulations ]
10,528             if( 8 + 8 + 8 + 2*(KeySize+1+8) < size_in64_L14_RAM - BUGGYoffset ) // the longest wrklen is LongestLineInclusive but actual is LongestLineInclusive(+ CR
char)
10,529             {
10,530             PseudoLinkedPointerNEW_64 = BufEnd_64_RAM;
10,531             BufEnd_64_RAM = BufEnd_64_RAM + 8 + 8 + 8;
10,532             BufEnd_64_RAM = BufEnd_64_RAM + 2*(KeySize+1+8);
10,533             }
10,534             else
10,535             { printf( "\nLeprechaun: Failure! Increment 'Memory for B-trees'!\n" );
10,536             fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, 11ToaDigits, 10), _ui64toaKAZEcomma((unsigned long
long)WORDcountDistinct, 11ToaDigits2, 10) );
10,537             fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14_RAM, 11ToaDigits, 10) );
10,538             fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11ToaDigits, 10) );
10,539             fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
10,540             fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for B-trees'!\n\n");
10,541             if ( HashChunkSizeInBITS_GLOBAL > 0 ) { // 2019-Dec-07
10,542             HashChunkSizeInBITS_GLOBAL--;
10,543             printf( "Automatically increasing number of passes in order to fit B-trees into Target Buffer.\n" );
10,544             goto NewAttemptRaisePasses;
10,545             } else exit( 7 );
10,546             }
10,547             if (PoffsetInLEAF == 0) // wrd < LW
10,548             {
10,549             //     memcpy( wrdUP, PseudoLinkedPointer+4+4+4, wrklen ); // LW up
10,550             //     memcpy( PseudoLinkedPointer+4+4+4, wrd, wrklen ); // wrd go to OLD LEAF
10,551             //     memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrklen, wrklen ); // RW go to NEW LEAF
10,552             //     *(char *)(&PseudoLinkedPointer+4+4+4+wrklen) = 0; // RW mark unused in OLD LEAF
10,553             // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,554             //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
10,555             //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,556             //fread(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
10,557             //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);

```

```

10,558 //fwrite(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
10,559 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
10,560 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,561 //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
10,562 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
10,563 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,564 //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
10,565 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
10,566 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,567 //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
10,568 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,569 // [ //r.14+
10,570 memcpy( &wrdUP[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
10,571 memcpy( &LEAF[8 + 8 + 8], &wrd[0], (KeySize+1+8) );
10,572 memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
10,573 // Here reordering (of writing wrdAUX) is needed to avoid seek the position NEW and stupidly to seek again OLD/current position!
10,574 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
10,575 if (BSTorBtree_RAM == 2) {
10,576 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,577 fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
10,578 } else { // ##### 64bit memory manipulations [
10,579 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
10,580 } // ##### 64bit memory manipulations ]
10,581 PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
10,582 if (BSTorBtree_RAM == 2) {
10,583 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,584 fwrite(&wrdAUX[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
10,585 } else { // ##### 64bit memory manipulations [
10,586 memcpy( (char *)PseudoLinkedPointerAUX_64, &wrdAUX[0], (KeySize+1+8) );
10,587 } // ##### 64bit memory manipulations ]
10,588 // ] //r.14+
10,589 }
10,590 if (POffsetInLEAF == 8) // LW < wrd < RW
10,591 {
10,592     memcpy( wrdUP, wrd, wrdlen ); // wrd up
10,593     memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
10,594     *(char *)(&PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
10,595     memcpy( wrdUP, wrd, (KeySize+1+8) ); // wrd up
10,596 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,597 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
10,598 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,599 //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
10,600 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
10,601 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,602 //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
10,603 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
10,604 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,605 //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
10,606 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,607 // [ //r.14+
10,608 memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
10,609 // Here reordering (of writing wrdAUX) is needed to avoid seek the position NEW and stupidly to seek again OLD/current position!
10,610 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
10,611 if (BSTorBtree_RAM == 2) {
10,612 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,613 fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
10,614 } else { // ##### 64bit memory manipulations [
10,615 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );

```

```

10,616         } // ##### 64bit memory manipulations ]
10,617         PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
10,618         if (BSTorBtree_RAM == 2) {
10,619             fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,620             fwrite(&wrdAUX[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
10,621         } else { // ##### 64bit memory manipulations [
10,622             memcpy( (char *)PseudoLinkedPointerAUX_64, &wrdAUX[0], (KeySize+1+8) );
10,623         } // ##### 64bit memory manipulations ]
10,624         // ] //r.14+
10,625     }
10,626     if (POffsetInLEAF == 16) // wrd > RW
10,627     {
10,628         //         memcpy( wrdUP, PseudoLinkedPointer+4+4+4+wrklen, wrklen ); // RW up
10,629         //         *(char *) (PseudoLinkedPointer+4+4+4+wrklen) = 0; // RW mark unused in OLD LEAF
10,630         //         memcpy( PseudoLinkedPointerNEW+4+4+4, wrd, wrklen ); // wrd go to NEW LEAF
10,631         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,632         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
10,633         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,634         //fread(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
10,635         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
10,636         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,637         //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
10,638         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
10,639         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,640         //fwrite(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
10,641         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,642         // [ //r.14+
10,643         memcpy( &wrdUP[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
10,644         memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
10,645         if (BSTorBtree_RAM == 2) {
10,646             fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,647             fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
10,648         } else { // ##### 64bit memory manipulations [
10,649             memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
10,650         } // ##### 64bit memory manipulations ]
10,651         // ] //r.14+
10,652         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one! Here NO need!
10,653         PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
10,654         if (BSTorBtree_RAM == 2) {
10,655             fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,656             fwrite(&wrd[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
10,657         } else { // ##### 64bit memory manipulations [
10,658             memcpy( (char *)PseudoLinkedPointerAUX_64, &wrd[0], (KeySize+1+8) );
10,659         } // ##### 64bit memory manipulations ]
10,660         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one! Here NO need!
10,661     }
10,662 }
10,663 else // If LEAF is not full: Case #1
10,664 { SplitOccured = 0; WORDcountDistinct++;
10,665     if (POffsetInLEAF == 0) // wrd < [LW][ ] so [LW][ ] -> [ ][LW] -> [wrd][LW]
10,666     {
10,667         //         memcpy( PseudoLinkedPointer+4+4+4+wrklen, PseudoLinkedPointer+4+4+4, wrklen );
10,668         //         memcpy( PseudoLinkedPointer+4+4+4, wrd, wrklen );
10,669         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,670         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
10,671         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,672         //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
10,673         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);

```

```

10,674 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,675 //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
10,676 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
10,677 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,678 //fwrite(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
10,679 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,680 // [ //r.14+
10,681 memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
10,682 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &wrdAUX[0], (KeySize+1+8) );
10,683 memcpy( &LEAF[8 + 8 + 8], &wrd[0], (KeySize+1+8) );
10,684 if (BSTorBtree_RAM == 2) {
10,685 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,686 fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
10,687 } else { // ##### 64bit memory manipulations [
10,688 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
10,689 } // ##### 64bit memory manipulations ]
10,690 // ] //r.14+
10,691 }
10,692 }
10,693 if (POffsetInLEAF == 8) // wrd > [LW][ ] so [LW][ ] -> [LW][wrd]
10,694 {
10,695 // memcpy( PseudoLinkedPointer+4+4+4+wrdlen, wrd, wrdlen );
10,696 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one! Here NO need!
10,697 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (KeySize+1+8);
10,698 if (BSTorBtree_RAM == 2) {
10,699 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,700 fwrite(&wrd[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
10,701 } else { // ##### 64bit memory manipulations [
10,702 memcpy( (char *)PseudoLinkedPointerAUX_64, &wrd[0], (KeySize+1+8) );
10,703 } // ##### 64bit memory manipulations ]
10,704 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one! Here NO need!
10,705 }
10,706 }
10,707
10,708 if (SplitOccured != 0)
10,709 {
10,710 // ~ Second deal with the INNER NODE(S) i.e Case #3 & Case #4:
10,711 while (StackPtr != 0 || SplitOccured != 0)
10,712 {
10,713 // 'PseudoLinkedPointerNEW' is new LEAF to be inserted
10,714 // 'wrdUP' is NEW word to be inserted
10,715 if (StackPtr != 0)
10,716 {
10,717 POffsetInLEAF = BSTstack[--StackPtr];
10,718 PseudoLinkedPointer_64 = BSTstack[--StackPtr];
10,719 //if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 ) // If LEAF is full: Case #4
10,720 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,721 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
10,722 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,723 //fread(&SomeByte, 1, 1, fp_outRG);
10,724 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,725 // [ //r.14+
10,726 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
10,727 if (BSTorBtree_RAM == 2) {
10,728 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,729 fread(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fread++;
10,730 } else { // ##### 64bit memory manipulations [
10,731 memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64, 8+8+8+2*(KeySize+1+8) );

```



```

10,732         } // ##### 64bit memory manipulations ]
10,733         memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (KeySize+1+8)], 1 );
10,734         // ] //r.14+
10,735         if (SomeByte != 0 ) // RW exists
10,736     { SplitOccured = 1;
10,737 //         memcpy( wrdUPold, wrdUP, wrdlen ); // LW up
10,738 //         PseudoLinkedPointerNEWold = PseudoLinkedPointerNEW;
10,739         memcpy( wrdUPold, wrdUP, (KeySize+1+8) );
10,740         PseudoLinkedPointerNEWold_64 = PseudoLinkedPointerNEW_64;
10,741 // Allocate NEW LEAF:
10,742 //         if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrdlen + 4 + 4 + 4 < GRMBLFoolAgain((int)wrdlen) ) // +4 more for BST instead of LL; + more(see
LEAF)
10,743 //         {
10,744 //             memcpy( &PseudoLinkedPointerNEW, &bufend[LetterOffset], 4 );
10,745 //             bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
10,746 //             bufend[LetterOffset] = bufend[LetterOffset] + 2*wrdlen;
10,747 //             if (MAXusedBuffer[wrdlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrdlen] = (unsigned long)(bufend[LetterOffset] -
BufStart);}
10,748 //         }
10,749 //         // BUG IN R.17 and prior - the next 'if' is only for RAM (y/Y) not for SSD (z/Z):
10,750 //         if (BSTorBtree_RAM == 2) { //r.18
10,751 BUGGYoffset = (BufEnd_64_RAM-(0+14));
10,752 //         } else { // ##### 64bit memory manipulations [
10,753 BUGGYoffset = (BufEnd_64_RAM-(unsigned long long)pointerflush_64_RAM);
10,754 //         } // ##### 64bit memory manipulations ]
10,755         if( 8 + 8 + 8 + 2*(KeySize+1+8) < size_in64_L14_RAM - BUGGYoffset ) // the longest wrdlen is LongestLineInclusive but actual is LongestLineInclusive(+ CR
char)
10,756         {
10,757             PseudoLinkedPointerNEW_64 = BufEnd_64_RAM;
10,758             BufEnd_64_RAM = BufEnd_64_RAM + 8 + 8 + 8;
10,759             BufEnd_64_RAM = BufEnd_64_RAM + 2*(KeySize+1+8);
10,760             // [ //r.14+
10,761             //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64;
10,762             //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,763             //fread(&LEAFNEW[0], 8+8+8+2*(LongestLineInclusive+1+4), 1, fp_outRG);
10,764             // In fact above three lines are slow, the only need is ZEROed LEAFNEW.
10,765             memset(&LEAFNEW[0], 0, 8+8+8+2*(KeySize+1+8));
10,766             // ] //r.14+
10,767         }
10,768     else
10,769     { printf( "\nLeprechaun: Failure! Increment 'Memory for B-trees'!\n" );
10,770 //         fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, 11TOaDigits, 10), _ui64toaKAZEcomma((unsigned long
long)WORDcountDistinct, 11TOaDigits2, 10) );
10,771         fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14_RAM, 11TOaDigits, 10) );
10,772         fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11TOaDigits, 10) );
10,773         fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
10,774         fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for B-trees'!\n\n" );
10,775 if ( HashChunkSizeInBITS_GLOBAL > 0 ) { // 2019-Dec-07
10,776     HashChunkSizeInBITS_GLOBAL--;
10,777     printf( "Automatically increasing number of passes in order to fit B-trees into Target Buffer.\n" );
10,778     goto NewAttemptRaisePasses;
10,779 } else exit( 7 );
10,780     }
10,781     if (POffsetInLEAF == 0) // wrdUPold < LW
10,782     {
10,783 //         memcpy( wrdUP, PseudoLinkedPointer+4+4+4, wrdlen ); // LW up
10,784 //         memcpy( PseudoLinkedPointer+4+4+4, wrdUPold, wrdlen ); // wrdUPold go to OLD LEAF
10,785 //         memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF

```

```

10,786 //      *(char *)(&PseudoLinkedPointer+4+4+wordlen) = 0; // RW mark unused in OLD LEAF
10,787 //      // [LP](PseudoLinkedPointerNEWold)[MP][RP](wordUPold)[LW][RW] -----
10,788 //      // pair [LW] PseudoLinkedPointerNEW goes up !
10,789 //      // PseudoLinkedPointer: PseudoLinkedPointerNEW: !
10,790 //      // [LP](PseudoLinkedPointerNEWold)[](wordUPold) [MP][RP][][RW] <----
10,791 //      // no need to put zero in RP because logic is based on words existence:
10,792 //      memcpy( PseudoLinkedPointerNEW+0, PseudoLinkedPointer+4, 4 );
10,793 //      memcpy( PseudoLinkedPointerNEW+4, PseudoLinkedPointer+8, 4 );
10,794 //      memcpy( PseudoLinkedPointer+4, &PseudoLinkedPointerNEWold, 4 );
10,795 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,796 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
10,797 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,798 //fread(&wordUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
10,799 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,800 //fwrite(&wordUPold[0], (LongestLineInclusive+1+4), 1, fp_outRG);
10,801 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
10,802 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,803 //fread(&wordAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
10,804 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
10,805 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,806 //fwrite(&wordAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
10,807 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
10,808 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,809 //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
10,810 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
10,811 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,812 //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
10,813 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 0;
10,814 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,815 //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
10,816 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
10,817 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,818 //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
10,819 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8;
10,820 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,821 //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
10,822 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
10,823 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,824 //fwrite(&PseudoLinkedPointerNEWold_64, 8, 1, fp_outRG);
10,825 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,826 // [ //r.14+
10,827 memcpy( &wordUP[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
10,828 memcpy( &LEAF[8 + 8 + 8], &wordUPold[0], (KeySize+1+8) );
10,829 memcpy( &wordAUX[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
10,830 memcpy( &LEAFNEW[8 + 8 + 8], &wordAUX[0], (KeySize+1+8) );
10,831 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
10,832 memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[8], 8 );
10,833 memcpy( &LEAFNEW[0], &PseudoLinkedPointerAUXdumbo_64, 8 );
10,834 memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[16], 8 );
10,835 memcpy( &LEAFNEW[8], &PseudoLinkedPointerAUXdumbo_64, 8 );
10,836 memcpy( &LEAF[8], &PseudoLinkedPointerNEWold_64, 8 );
10,837 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
10,838 if (BSTorBtree_RAM == 2) {
10,839 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,840 fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
10,841 } else { // ##### 64bit memory manipulations [
10,842 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
10,843 // ##### 64bit memory manipulations ]

```

```

10,844     PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64;
10,845         if (BSTorBtree_RAM == 2) {
10,846             fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,847             fwrite(&LEAFNEW[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
10,848             } else { // ##### 64bit memory manipulations [
10,849             memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAFNEW[0], 8+8+2*(KeySize+1+8) );
10,850             } // ##### 64bit memory manipulations ]
10,851         // ] //r.14+
10,852     }
10,853     if (POffsetInLEAF == 8) // LW < wrdUPold < RW
10,854     {
10,855         //         memcpy( wrdUP, wrdUPold, wrdlen ); // wrdUPold up
10,856         //         memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
10,857         //         *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
10,858         //         // [LP][MP](PseudoLinkedPointerNEWold)[RP][LW](wrdUPold)[RW] -----
10,859         //         // pair [wrdUPold] PseudoLinkedPointerNEW goes up |
10,860         //         // PseudoLinkedPointer: PseudoLinkedPointerNEW: |
10,861         //         // [LP][MP][ ][LW] (PseudoLinkedPointerNEWold)[RP][ ][RW] <----
10,862         //         // no need to put zero in RP because logic is based on words existence:
10,863         //         memcpy( PseudoLinkedPointerNEW+0, &PseudoLinkedPointerNEWold, 4 );
10,864         //         memcpy( PseudoLinkedPointerNEW+4, PseudoLinkedPointer+8, 4 );
10,865         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,866         //         memcpy( wrdUP, wrdUPold, (KeySize+1+8) );
10,867         //         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
10,868         //         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,869         //         //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
10,870         //         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
10,871         //         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,872         //         //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
10,873         //         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
10,874         //         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,875         //         //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
10,876         //         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 0;
10,877         //         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,878         //         //fwrite(&PseudoLinkedPointerNEWold_64, 8, 1, fp_outRG);
10,879         //         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
10,880         //         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,881         //         //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
10,882         //         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8;
10,883         //         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,884         //         //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
10,885         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,886         // [ //r.14+
10,887         //         memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
10,888         //         memcpy( &LEAFNEW[8 + 8 + 8], &wrdAUX[0], (KeySize+1+8) );
10,889         //         memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
10,890         //         memcpy( &LEAFNEW[0], &PseudoLinkedPointerNEWold_64, 8 );
10,891         //         memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[16], 8 );
10,892         //         memcpy( &LEAFNEW[8], &PseudoLinkedPointerAUXdumbo_64, 8 );
10,893         //         PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
10,894         //         if (BSTorBtree_RAM == 2) {
10,895         //             fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,896         //             fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
10,897         //             } else { // ##### 64bit memory manipulations [
10,898         //             memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
10,899         //             } // ##### 64bit memory manipulations ]
10,900         //         PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64;
10,901         //         if (BSTorBtree_RAM == 2) {

```

```

10,902         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,903         fwrite(&LEAFNEW[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
10,904         } else { // ##### 64bit memory manipulations [
10,905         memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAFNEW[0], 8+8+2*(KeySize+1+8) );
10,906         } // ##### 64bit memory manipulations ]
10,907         // ] //r.14+
10,908     }
10,909     if (POffsetInLEAF == 16) // wrdUPold > RW
10,910     {
10,911         //         memcpy( wrdUP, PseudoLinkedPointer+4+4+4*wrklen, wrklen ); // RW up
10,912         //         *(char *) (PseudoLinkedPointer+4+4+4*wrklen) = 0; // RW mark unused in OLD LEAF
10,913         //         memcpy( PseudoLinkedPointerNEW+4+4+4, wrdUPold, wrklen ); // wrdUPold go to NEW LEAF
10,914         //         // [LP][MP][RP](PseudoLinkedPointerNEWold)[LW][RW](wrdUPold) -----
10,915         //         // pair [RW] PseudoLinkedPointerNEW goes up |
10,916         //         // PseudoLinkedPointer: PseudoLinkedPointerNEW: |
10,917         //         // [LP][MP][LW] [RP](PseudoLinkedPointerNEWold)[LW](wrdUPold) <---
10,918         //         // no need to put zero in RP because logic is based on words existence:
10,919         //         memcpy( PseudoLinkedPointerNEW+0, PseudoLinkedPointer+8, 4 );
10,920         //         memcpy( PseudoLinkedPointerNEW+4, &PseudoLinkedPointerNEWold, 4 );
10,921         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,922         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
10,923         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,924         //fread(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
10,925         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
10,926         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,927         //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
10,928         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
10,929         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,930         //fwrite(&wrdUPold[0], (LongestLineInclusive+1+4), 1, fp_outRG);
10,931         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
10,932         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,933         //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
10,934         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 0;
10,935         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,936         //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
10,937         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8;
10,938         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,939         //fwrite(&PseudoLinkedPointerNEWold_64, 8, 1, fp_outRG);
10,940         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,941         // [ //r.14+
10,942         memcpy( &wrdUP[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
10,943         memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
10,944         memcpy( &LEAFNEW[8 + 8 + 8], &wrdUPold[0], (KeySize+1+8) );
10,945         memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[16], 8 );
10,946         memcpy( &LEAFNEW[0], &PseudoLinkedPointerAUXdumbo_64, 8 );
10,947         memcpy( &LEAFNEW[8], &PseudoLinkedPointerNEWold_64, 8 );
10,948         PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
10,949         if (BSTorBtree_RAM == 2) {
10,950             fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,951             fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
10,952             } else { // ##### 64bit memory manipulations [
10,953             memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
10,954             } // ##### 64bit memory manipulations ]
10,955         PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64;
10,956         if (BSTorBtree_RAM == 2) {
10,957             fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,958             fwrite(&LEAFNEW[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
10,959             } else { // ##### 64bit memory manipulations [

```

```

10,960         memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAFNEW[0], 8+8+2*(KeySize+1+8) );
10,961         } // ##### 64bit memory manipulations ]
10,962     // ] //r.14+
10,963 }
10,964 }
10,965 else // If LEAF is not full: Case #3
10,966 { SplitOccured = 0;
10,967     if (POffsetInLEAF == 0) // wrdUP < [LW][ ] so [LW][ ] -> [ ][LW] -> [wrdUP][LW]
10,968     {
10,969         //         memcpy( PseudoLinkedPointer+4+4+4*wrdlen, PseudoLinkedPointer+4+4+4, wrdlen );
10,970         //         memcpy( PseudoLinkedPointer+4+4+4, wrdUP, wrdlen );
10,971         //         // [LP][MP][ ] -> [LP][ ][MP] -> [LP][np][MP]
10,972         //         memcpy( PseudoLinkedPointer+8, PseudoLinkedPointer+4, 4 );
10,973         //         memcpy( PseudoLinkedPointer+4, &PseudoLinkedPointerNEW, 4 );
10,974         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,975         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
10,976         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,977         //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
10,978         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
10,979         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,980         //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
10,981         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
10,982         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,983         //fwrite(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
10,984         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
10,985         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,986         //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
10,987         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
10,988         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,989         //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
10,990         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
10,991         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
10,992         //fwrite(&PseudoLinkedPointerNEW_64, 8, 1, fp_outRG);
10,993         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
10,994         // [ //r.14+
10,995         memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
10,996         memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &wrdAUX[0], (KeySize+1+8) );
10,997         memcpy( &LEAF[8 + 8 + 8], &wrdUP[0], (KeySize+1+8) );
10,998         memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[8], 8 );
10,999         memcpy( &LEAF[8 + 8], &PseudoLinkedPointerAUXdumbo_64, 8 );
11,000         memcpy( &LEAF[8], &PseudoLinkedPointerNEW_64, 8 );
11,001         if (BSTorBtree_RAM == 2) {
11,002             fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,003             fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
11,004         } else { // ##### 64bit memory manipulations [
11,005             memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
11,006             } // ##### 64bit memory manipulations ]
11,007         // ] //r.14+
11,008     }
11,009     if (POffsetInLEAF == 8) // wrdUP > [LW][ ] so [LW][ ] -> [LW][wrdUP]
11,010     {
11,011         //         memcpy( PseudoLinkedPointer+4+4+4*wrdlen, wrdUP, wrdlen );
11,012         //         // [LP][MP][ ] -> [LP][MP][np]
11,013         //         memcpy( PseudoLinkedPointer+8, &PseudoLinkedPointerNEW, 4 );
11,014         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,015         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
11,016         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,017         //fwrite(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);

```

```

11,018 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
11,019 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,020 //fwrite(&PseudoLinkedPointerNEW_64, 8, 1, fp_outRG);
11,021 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,022 // [ //r.14+
11,023 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &wrdUP[0], (KeySize+1+8) );
11,024 memcpy( &LEAF[8 + 8], &PseudoLinkedPointerNEW_64, 8 );
11,025 if (BSTorBtree_RAM == 2) {
11,026 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,027 fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
11,028 } else { // ##### 64bit memory manipulations [
11,029 memcpy( (char *)&PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
11,030 } // ##### 64bit memory manipulations ]
11,031 // ] //r.14+
11,032 }
11,033 break;
11,034 }
11,035 }
11,036 else // Empty stack means ROOT and more over ROOT is already splitted(Case #4 is off)
11,037 {
11,038 // If LEAF is not full: Case #3
11,039 // THIS IS WHERE A NEW(SECOND) LEAF 'PseudoLinkedPointerROOT' must be allocated:
11,040 // if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrklen + 4 + 4 + 4 < GRMBLFoolAgain((int)wrklen) ) // +4 more for BST instead of LL; + more(see
LEAF)
11,041 // {
11,042 // memcpy( &PseudoLinkedPointerROOT, &bufend[LetterOffset], 4 );
11,043 // bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
11,044 // memcpy( bufend[LetterOffset], wrdUP, wrklen );
11,045 // bufend[LetterOffset] = bufend[LetterOffset] + 2*wrklen;
11,046 // if (MAXusedBuffer[wrklen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrklen] = (unsigned long)(bufend[LetterOffset] -
BufStart);}
11,047 // }
11,048 // BUG IN R.17 and prior - the next 'if' is only for RAM (y/Y) not for SSD (z/Z):
11,049 if (BSTorBtree_RAM == 2) { //r.18
11,050 BUGGYoffset = (BufEnd_64_RAM-(0+14));
11,051 } else { // ##### 64bit memory manipulations [
11,052 BUGGYoffset = (BufEnd_64_RAM-(unsigned long long)pointerflush_64_RAM);
11,053 } // ##### 64bit memory manipulations ]
11,054 if( 8 + 8 + 8 + 2*(KeySize+1+8) < size_in64_L14_RAM - BUGGYoffset ) // the longest wrklen is LongestLineInclusive but actual is LongestLineInclusive(+ CR
char)
11,055 {
11,056 PseudoLinkedPointerROOT_64 = BufEnd_64_RAM;
11,057 BufEnd_64_RAM = BufEnd_64_RAM + 8 + 8 + 8;
11,058 BufEnd_64_RAM = BufEnd_64_RAM + 2*(KeySize+1+8);
11,059 PseudoLinkedPointerAUX_64 = PseudoLinkedPointerROOT_64 + 8 + 8 + 8;
11,060 if (BSTorBtree_RAM == 2) {
11,061 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,062 fwrite(&wrdUP[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
11,063 } else { // ##### 64bit memory manipulations [
11,064 memcpy( (char *)&PseudoLinkedPointerAUX_64, &wrdUP[0], (KeySize+1+8) );
11,065 } // ##### 64bit memory manipulations ]
11,066 }
11,067 else
11,068 { printf( "\nLeprechaun: Failure! Increment 'Memory for B-trees'!\n" );
11,069 fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, 11ToaDigits, 10), _ui64toaKAZEcomma((unsigned long
long)WORDcountDistinct, 11ToaDigits2, 10) );
11,070 fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14_RAM, 11ToaDigits, 10) );
11,071 fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11ToaDigits, 10) );

```

```

11,072     fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
11,073     fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for B-trees'!\n\n");
11,074 if ( HashChunkSizeInBITS_GLOBAL > 0 ) { // 2019-Dec-07
11,075     HashChunkSizeInBITS_GLOBAL--;
11,076     printf( "Automatically increasing number of passes in order to fit B-trees into Target Buffer.\n" );
11,077     goto NewAttemptRaisePasses;
11,078 } else exit( 7 );
11,079     }
11,080     // Here -- 'PseudoLinkedPointerROOT' --
11,081     // ! (wrUP) !
11,082     // 'PseudoLinkedPointer' <-- --> 'PseudoLinkedPointerNEW'
11,083     // (LW) (RW)
11,084 //     memcpy( PseudoLinkedPointerROOT, &PseudoLinkedPointer, 4 ); // LP
11,085 //     memcpy( PseudoLinkedPointerROOT+4, &PseudoLinkedPointerNEW, 4 ); // MP
11,086 //     // Here must NEW ROOT be updated i.e. HASH table(SLOT) must point it:
11,087 //     memcpy( BufStart+Slot, &PseudoLinkedPointerROOT, 4 );
11,088     PseudoLinkedPointerAUX_64 = PseudoLinkedPointerROOT_64;
11,089     if (BSTorBtree_RAM == 2) {
11,090     fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,091     fwrite(&PseudoLinkedPointer_64, 8, 1, fp_outRG); Total_fwrite++;
11,092     } else { // ##### 64bit memory manipulations [
11,093     memcpy( (char *)&PseudoLinkedPointerAUX_64, &PseudoLinkedPointer_64, 8 );
11,094     } // ##### 64bit memory manipulations ]
11,095     PseudoLinkedPointerAUX_64 = PseudoLinkedPointerROOT_64 + 8;
11,096     if (BSTorBtree_RAM == 2) {
11,097     fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,098     fwrite(&PseudoLinkedPointerNEW_64, 8, 1, fp_outRG); Total_fwrite++;
11,099     } else { // ##### 64bit memory manipulations [
11,100     memcpy( (char *)&PseudoLinkedPointerAUX_64, &PseudoLinkedPointerNEW_64, 8 );
11,101     } // ##### 64bit memory manipulations ]
11,102     memcpy( BufStart+Slot, &PseudoLinkedPointerROOT_64, 8 );
11,103     break; //because it is ROOT without split
11,104 }
11,105     } // while
11,106 } //if (SplitOccured != 0)
11,107 // 3] Insert Iterative ]
11,108 } //if (FoundInLinkedList == 0)
11,109 // ##### B-tree order 3 fragment 64bit ]
11,110
11,111 } //if ( (Slot>>HashChunkSizeInBITS) == RipPasses ) {
11,112 } //for (BuildingBlocksSTRIDE=0;
11,113
11,114 // PASS #1: ]
11,115
11,116 // PASS #2: [
11,117
11,118 for (BuildingBlocksSTRIDE=0; BuildingBlocksSTRIDE < size_inLINESIXFOUR-MatchLens[jj]+1; BuildingBlocksSTRIDE++) {
11,119     // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed [
11,120     //     wrdlen=0;
11,121     //     memset( &wrd[0], 0, (LongestLineInclusive+1+8) ); //r.18, see below, the old nullifier is commented.
11,122     //     for (mm=0; mm< MatchLens[jj]; mm++) {
11,123     //         memcpy( &wrd[ wrdlen ], TwoDigitHEXlist[*(unsigned char *) (SourceBlock+mm+BuildingBlocksSTRIDE)], 2 );
11,124     //         wrdlen++;
11,125     //         wrdlen++;
11,126     //     }
11,127     // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed ]
11,128
11,129     wrdlen=MatchLens[jj];

```

```
11,130         memset( &wrd[0], 0, (LongestLineInclusive+1+8) ); //r.18, see below, the old nullifier is commented.
11,131
11,132 //         if ( wrdlen > 28 ) {
11,133 //             FIP5202_SHA3_224((unsigned char *) (SourceBlock+BuildingBlocksSTRIDE), wrdlen, (unsigned char *) SHA328bytes);
11,134 //             wrdlen=28;
11,135 //         }
11,136
11,137         memcpy( &wrd[ 0 ], &wrdlen, 1 ); // The keysize is 1..255, pretty enough since my needs are 1..28 (up to SHA3-224).
11,138 //         if ( wrdlen < 28 )
11,139 //             memcpy( &wrd[ 0+1 ], (unsigned char *) (SourceBlock+BuildingBlocksSTRIDE), wrdlen );
11,140 //         else
11,141 //             memcpy( &wrd[ 0+1 ], (unsigned char *) SHA328bytes, wrdlen );
11,142
11,143         BufStart = pointerflush;
11,144 //         Slot = ((wrd[0]-'_')*28*28*28*28 + (wrd[1]-'_')*28*28*28 + (wrd[2]-'_')*28*28 + (wrd[3]-'_')*28 + (wrd[4]-'_'))<<3;
11,145 //         Slot = FNV1A_Hash_Jesteress_27bit(wrd, wrdlen)<<3; // Commented since r.14+++ because of passes.
11,146 // Slot = FNV1A_Hash_Jesteress_27bit(wrd +1, wrdlen); WORDcount++;
11,147 Slot = FNV1A_Pippip_Yurii(wrd +1, wrdlen); WORDcount++; // Changed 2019-Nov-28
11,148
11,149 // Bug fix for all r.14+++ and below! [
11,150 //memcpy( &wrd[(LongestLineInclusive+1+4)-4], &NULLsForWRD, 4 );
11,151 //memcpy( &wrd[(LongestLineInclusive+1+4)-4-1], &NULLsForWRD, 1 );
11,152 // Bug fix for all r.14+++ and below! ]
11,153
11,154 // Example: HashInBITS-HashChunkSizeInBITS=2
11,155 //         HashInBITS = 5
11,156 //         HashChunkSizeInBITS = 3
11,157 //         RipPasses = 1<<((HashInBITS-HashChunkSizeInBITS) i.e. 1<<2 which is 4 i.e. 32 slots with 4 passes 8 slots each.
11,158 //         00??? 5bits 0-7
11,159 //         01??? 5bits 8-15
11,160 //         10??? 5bits 16-23
11,161 //         11??? 5bits 24-31
11,162 if ( (Slot>>HashChunkSizeInBITS_GLOBAL) == RipPasses ) {
11,163     Slot = Slot<<3;
11,164 // NEW NEW NEW [ //r.18
11,165 // In here Slot is not within a single pool but in MatchLensNUM sub-pools i.e. MatchLensNUM-way i.e. MatchLensNUM hashpots:
11,166 /*
11,167 for (GettingIndexOfArray=0; GettingIndexOfArray<MatchLensNUM; GettingIndexOfArray++) {
11,168     // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed [
11,169     //         if ( (wrdlen>>1)==MatchLens[GettingIndexOfArray] ) break;
11,170     if ( (wrdlen)==MatchLens[GettingIndexOfArray] ) break;
11,171     // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed ]
11,172 }
11,173 */
11,174 GettingIndexOfArray=jj; // same as above atrocity
11,175 Slot_RAM = Slot +(MatchLensNUM*(1LL<<(HashInBITS_GLOBAL)*8); // 2019-Dec-04
11,176 Slot = Slot *(GettingIndexOfArray*(1LL<<(HashInBITS_GLOBAL)*8); // CAUTION: 'wrdlen' is halved here, when a new revision comes with 1:1 keysize then change it.
11,177 // NEW NEW NEW ] //r.18
11,178
11,179 //KeySize = LongestLineInclusive; // 'LongestLineInclusive' is for 1-way hashing where different KeySizes can occupy same HASH pool.
11,180 KeySize = wrdlen; //r.18, In case multi-way hashing: either the MAX allowed or the 'wrdlen'.
11,181
11,182 //Slot = 0; // One Tree only!
11,183 memcpy( &PseudoLinkedPointer_64, BufStart+Slot_RAM, 8 );
11,184
11,185 // HERE THE 'DECIDER' is EXECUTED (whether 'wrd' is encountered 1+ times within ALL KEYS already put in PASS #1) [[[
11,186 FoundInLinkedList_RAM = 0; // Assume not found
11,187 // 1] Search [ Just check for 1+ occurrences
```



```

11,188
11,189         if (PseudoLinkedPointer_64 == 0) // means EMPTY-SLOT
11,190         {
11,191             // Impossible, they all have already been inserted...
11,192         }
11,193         else // means USED-SLOT
11,194         { FoundInLinkedList = 0;
11,195             StackPtr = 0;
11,196             // while (PseudoLinkedPointer != 0 && FoundInLinkedList == 0)
11,197             while (PseudoLinkedPointer_64 != 0 && FoundInLinkedList == 0)
11,198             {
11,199             // ***** 'P W P' section [
11,200             // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
11,201             // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
11,202             // here ALWAYS LW exists: no need for existence check - line below
11,203             // if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
11,204             // if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wordlen) > 0) // go LP
11,205             // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,206             //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
11,207             //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,208             //fread(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
11,209             // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,210             // [ //r.14+
11,211             PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
11,212             if (BSTorBtree_RAM == 2) {
11,213             fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,214             fread(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG);
11,215             } else { // ##### 64bit memory manipulations [
11,216             memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64, 8+8+8+2*(KeySize+1+8) );
11,217             } // ##### 64bit memory manipulations ]
11,218             memcpy( &FourGramL[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
11,219             // ] //r.14+
11,220             //strFLAG=strcmpKAZE13(FourGramL, wrd);
11,221             strFLAG=memcmp(FourGramL+1, wrd+1, wordlen); // Since 2019-Feb-07, it is not ASCIIZ string but zero byte is the length of the 1..
11,222             if (strFLAG > 0) // go LP
11,223             { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 0, 4 ); //LP
11,224             // PseudoLinkedPointer = PseudoLinkedPointerNEW;
11,225             // }
11,226             {
11,227             // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,228             //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 0; //LP
11,229             // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,230             if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
11,231             BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
11,232             BSTstack[StackPtr] = 0; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
11,233             // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,234             //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,235             //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
11,236             // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,237             // [ //r.14+
11,238             memcpy( &PseudoLinkedPointer_64, &LEAF[0], 8 );
11,239             // ] //r.14+
11,240             }
11,241             // else if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wordlen) < 0) // go RP or MP
11,242             else if (strFLAG < 0) // go RP or MP
11,243             { // RW existence check - line below:
11,244             // if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 ) // RW exists
11,245             // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!

```

```

11,246 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
11,247 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,248 //fread(&SomeByte, 1, 1, fp_outRG);
11,249 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,250 // [ //r.14+
11,251 memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (KeySize+1+8)], 1 );
11,252 // ] //r.14+
11,253 if (SomeByte != 0 ) // RW exists
11,254 { // Here all 'P W P' section is repeated; the way of handling case when dynamic number of words in leaf
11,255 // ***** 'P W P' section 2 [
11,256 // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
11,257 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
11,258 // here ALWAYS RW exists: no need for existence check - line below
11,259 // if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
11,260 // if (memcmp(PseudoLinkedPointer+4+4+4+wordlen, wrd, wordlen) > 0) // go MP
11,261 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,262 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
11,263 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,264 //fread(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
11,265 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,266 // [ //r.14+
11,267 memcpy( &FourGramL[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
11,268 // ] //r.14+
11,269 //strFLAG=strcmpKAZE13(FourGramL, wrd);
11,270 strFLAG=memcmp(FourGramL+1, wrd+1, wordlen); // Since 2019-Feb-07, it is not ASCIIZ string but zero byte is the length of the 1..
11,271 if (strFLAG > 0) // go MP
11,272 { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
11,273 // PseudoLinkedPointer = PseudoLinkedPointerNEW;
11,274 // }
11,275 // {
11,276 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,277 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8; //MP
11,278 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,279 if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
11,280 BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
11,281 BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
11,282 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,283 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,284 //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
11,285 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,286 // [ //r.14+
11,287 memcpy( &PseudoLinkedPointer_64, &LEAF[8], 8 );
11,288 // ] //r.14+
11,289 }
11,290 }
11,291 // else if (memcmp(PseudoLinkedPointer+4+4+4+wordlen, wrd, wordlen) < 0) // go RP
11,292 // else if (strFLAG < 0) // go RP
11,293 // { // No ?W after RW - go RP
11,294 // memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4 + 4, 4 ); //RP
11,295 // PseudoLinkedPointer = PseudoLinkedPointerNEW;
11,296 // }
11,297 // {
11,298 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,299 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8; //RP
11,300 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,301 if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
11,302 BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
11,303 BSTstack[StackPtr] = 16; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;

```

```

11,304 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,305 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,306 //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
11,307 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,308 // [ //r.14+
11,309 memcpy( &PseudoLinkedPointer_64, &LEAF[8 + 8], 8 );
11,310 // ] //r.14+
11,311 }
11,312 else { FoundInLinkedList = 1; // wrd is RW
11,313 // Counter [
11,314 if (BSTorBtree_RAM == 2) {
11,315 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,316 }
11,317 //memcpy( &CounterOccurrences, &FourGramL[(KeySize+1+4)-4], 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
11,318 memcpy( &CounterOccurrences, &FourGramL[(KeySize+1+8)-8], 8 ); // 2019-Dec-04
11,319 if (CounterOccurrences>1) FoundInLinkedList_RAM = 1; // 2019-Dec-04
11,320
11,321 // Debug the ratio difference [
11,322 // if (CounterOccurrences>1) {
11,323 // printf("\nPASS #2 (decider): BuildingBlocksSTRIDE, jj = %s, %s\n", _ui64toaKAZEzerocomma4(BuildingBlocksSTRIDE, llTOaDigits3, 16)+(26-
8-1), _ui64toaKAZEzerocomma4(jj, llTOaDigits2, 10)+(26-2));
11,324 // }
11,325 // Debug the ratio difference ]
11,326
11,327 //if (CounterOccurrences<999999999) CounterOccurrences++; // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
11,328 //memcpy( &FourGramL[(KeySize+1+4)-4], &CounterOccurrences, 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
11,329 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,330 //fwrite(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
11,331 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,332 // [ //r.14+
11,333 // memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &FourGramL[0], (KeySize+1+8) );
11,334 // if (BSTorBtree_RAM == 2) {
11,335 // fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG);
11,336 // } else { // ##### 64bit memory manipulations [
11,337 // memcpy( (char *)&PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
11,338 // } // ##### 64bit memory manipulations ]
11,339 // ] //r.14+
11,340 // Counter ]
11,341 }
11,342 // ***** 'P W P' section 2 ]
11,343 // *****
11,344 }
11,345 else // RW empty - go MP
11,346 { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
11,347 // PseudoLinkedPointer = PseudoLinkedPointerNEW;
11,348 // }
11,349 {
11,350 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,351 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8; //MP
11,352 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,353 if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
11,354 BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
11,355 BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
11,356 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,357 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,358 //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
11,359 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,360 // [ //r.14+

```

```

11,361         memcpy( &PseudoLinkedPointer_64, &LEAF[8], 8 );
11,362         // ] //r.14+
11,363         }
11,364     }
11,365     else { FoundInLinkedList = 1; // wrd is LW
11,366     // Counter [
11,367         if (BSTorBtree_RAM == 2) {
11,368             fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,369         }
11,370         //memcpy( &CounterOccurrences, &FourGramL[(KeySize+1+4)-4], 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
11,371         memcpy( &CounterOccurrences, &FourGramL[(KeySize+1+8)-8], 8 ); // 2019-Dec-04
11,372         if (CounterOccurrences>1) FoundInLinkedList_RAM = 1; // 2019-Dec-04
11,373
11,374     // Debug the ratio difference [
11,375         if (CounterOccurrences>1) {
11,376             // printf("\nPASS #2 (decider): BuildingBlocksSTRIDE, jj = %s, %s\n", _ui64toaKAZEzerocomma4(BuildingBlocksSTRIDE, llToaDigits3, 16)+(26-
8-1), _ui64toaKAZEzerocomma4(jj, llToaDigits2, 10)+(26-2));
11,377         }
11,378     // Debug the ratio difference ]
11,379
11,380     //if (CounterOccurrences<999999999) CounterOccurrences++; // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
11,381     //memcpy( &FourGramL[(KeySize+1+4)-4], &CounterOccurrences, 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
11,382     // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,383     //fwrite(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
11,384     // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,385     // [ //r.14+
11,386     //     memcpy( &LEAF[8 + 8 + 8], &FourGramL[0], (KeySize+1+8) );
11,387     //     if (BSTorBtree_RAM == 2) {
11,388     //         fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG);
11,389     //     } else { // ##### 64bit memory manipulations [
11,390     //         memcpy( (char *)&PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
11,391     //     } // ##### 64bit memory manipulations ]
11,392     // ] //r.14+
11,393     // Counter ]
11,394     }
11,395     // ***** 'P W P' section ]
11,396     // while
11,397     }
11,398
11,399     // 1] Search ] Just check for 1+ occurrences
11,400     // HERE THE 'DECIDER' is EXECUTED (whether 'word' is encountered 1+ times within ALL KEYS already put in PASS #1) ]]]
11,401
11,402     //Slot = 0; // One Tree only!
11,403     memcpy( &PseudoLinkedPointer_64, BufStart+Slot, 8 );
11,404
11,405     if (FoundInLinkedList_RAM == 1) { // 2019-Dec-04
11,406         // ##### B-tree order 3 fragment 64bit [
11,407         //
11,408         // LEAF structure: [LeftPointer][MiddlePointer][RightPointer][LeftWord][RightWord]
11,409         //                     4bytes      4bytes      4bytes      wrdlen      wrdlen
11,410         //                                     *          *          <- if *(char *)==0 means the word cell is empty
11,411         // ALL B-tree order 3 fragment consists of 3 sub-fragments:
11,412         // 1] Search 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search 3] Insert Iterative
11,413
11,414         // LEAF_64 structure: [LeftPointer][MiddlePointer][RightPointer][LeftWord] [RightWord]
11,415         //                     8bytes      8bytes      8bytes      LongestLineInclusive+1+4 LongestLineInclusive+1+4
11,416         //                                     *          *          <- if *(char *)==0 means the word cell is empty
11,417         // Note: In order to use one fread(and strcmp) a NULL postfix for LeftWord, RightWord i.e. LeftWord_Length=len(LeftWord)+1 a kinda stupid choice ...

```

```

11,418 // Note: BufEnd_64 in fact is the first free position after the BUFFER END!
11,419
11,420 // 1] Search [
11,421         if (PseudoLinkedPointer_64 == 0) // means EMPTY-SLOT
11,422         {
11,423             //if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrklen + 4 + 4 + 4 < GRMBLFoolAgain((int)wrklen) ) // +4 more for BST instead of LL; + more(see
LEAF)
11,424             //         {
11,425             //             memcpy( BufStart+Slot, &bufend[LetterOffset], 4 );
11,426             //             bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
11,427             //             memcpy( bufend[LetterOffset], wrd, wrklen ); WORDcountDistinct++; bufNumberOfWords[LetterOffset]++;
11,428             //             bufend[LetterOffset] = bufend[LetterOffset] + 2*wrklen;
11,429             //             if (MAXusedBuffer[wrklen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrklen] = (unsigned long)(bufend[LetterOffset] -
BufStart);}
11,430             //         }
11,431             // BUG IN R.17 and prior - the next 'if' is only for RAM (y/Y) not for SSD (z/Z):
11,432             if (BSTorBtree == 2) { //r.18
11,433                 BUGGYoffset = (BufEnd_64-(0*14));
11,434                 } else { // ##### 64bit memory manipulations [
11,435                 BUGGYoffset = (BufEnd_64-(unsigned long long)pointerflush_64);
11,436                 } // ##### 64bit memory manipulations ]
11,437             if( 8 + 8 + 8 + 2*(KeySize+1+8) < size_in64_L14 - BUGGYoffset ) // the longest wrklen is LongestLineInclusive but actual is LongestLineInclusive(+ CR char)
11,438             {
11,439                 memcpy( BufStart+Slot, &BufEnd_64, 8 );
11,440                 BufEnd_64 = BufEnd_64 + 8 + 8 + 8;
11,441                 if (BSTorBtree == 2) {
11,442                     fsetpos(fp_outRG, (const fpos_t *)&BufEnd_64);
11,443                     //fwrite(wrd, wrklen, 1, fp_outRG); //GRMBL! r.18
11,444                     fwrite(wrd, (KeySize+1+8), 1, fp_outRG); //GRMBL! r.18
11,445                     WORDcountDistinct++; Total_fwrite++;
11,446                     //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
11,447                     // r.14++ The above line was commented because the pool is already ZEROed.
11,448                     } else { // ##### 64bit memory manipulations [
11,449                     //memcpy( (char *)BufEnd_64, wrd, wrklen ); //GRMBL! r.18
11,450                     memcpy( (char *)BufEnd_64, wrd, (KeySize+1+8) ); //GRMBL! r.18
11,451                     WORDcountDistinct++;
11,452                     } // ##### 64bit memory manipulations ]
11,453                     BufEnd_64 = BufEnd_64 + 2*(KeySize+1+8);
11,454                     //fsetpos(fp_outRG, &BufEnd_64);
11,455                     }
11,456                 else
11,457                 { printf( "\nLeprechaun: Failure! Increment 'Memory for B-trees'\n" );
11,458 fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, llToaDigits, 10), _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct,
llToaDigits2, 10) );
11,459 fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14, llToaDigits, 10) );
11,460 fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, llToaDigits, 10) );
11,461 fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
11,462 fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for B-trees'\n\n");
11,463 exit( 7 );
11,464 }
11,465         FoundInLinkedList = 1+1;
11,466         }
11,467         else // means USED-SLOT
11,468         { FoundInLinkedList = 0;
11,469         StackPtr = 0;
11,470         //         while (PseudoLinkedPointer != 0 && FoundInLinkedList == 0)
11,471         //         while (PseudoLinkedPointer_64 != 0 && FoundInLinkedList == 0)
11,472         {

```

```

11,473 // ***** 'P W P' section [
11,474 // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
11,475 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
11,476 // here ALWAYS LW exists: no need for existence check - line below
11,477 // if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
11,478 // if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wordlen) > 0) // go LP
11,479 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,480 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
11,481 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,482 //fread(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
11,483 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,484 // [ //r.14+
11,485 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
11,486 if (BSTorBtree == 2) {
11,487 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,488 fread(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fread++;
11,489 } else { // ##### 64bit memory manipulations [
11,490 memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64, 8+8+8+2*(KeySize+1+8) );
11,491 } // ##### 64bit memory manipulations ]
11,492 memcpy( &FourGramL[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
11,493 // ] //r.14+
11,494 //strFLAG=strcmpKAZE13(FourGramL, wrd);
11,495 strFLAG=memcmp(FourGramL+1, wrd+1, wordlen); // Since 2019-Feb-07, it is not ASCIIZ string but zero byte is the length of the 1..
11,496 if (strFLAG > 0) // go LP
11,497 { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 0, 4 ); //LP
11,498 PseudoLinkedPointer = PseudoLinkedPointerNEW;
11,499 }
11,500 {
11,501 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,502 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 0; //LP
11,503 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,504 if (StackPtr > 8192*3-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
11,505 BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
11,506 BSTstack[StackPtr] = 0; ++StackPtr; //LPOffset=0;MPOffset=8;RPOffset=16;
11,507 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,508 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,509 //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
11,510 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,511 // [ //r.14+
11,512 memcpy( &PseudoLinkedPointer_64, &LEAF[0], 8 );
11,513 // ] //r.14+
11,514 }
11,515 // else if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wordlen) < 0) // go RP or MP
11,516 else if (strFLAG < 0) // go RP or MP
11,517 { // RW existence check - line below:
11,518 if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 ) // RW exists
11,519 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,520 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
11,521 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,522 //fread(&SomeByte, 1, 1, fp_outRG);
11,523 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,524 // [ //r.14+
11,525 memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (KeySize+1+8)], 1 );
11,526 // ] //r.14+
11,527 if (SomeByte != 0) // RW exists
11,528 { // Here all 'P W P' section is repeated; the way of handling case when dynamic number of words in leaf
11,529 // *****
11,530 // ***** 'P W P' section 2 [

```

```

11,531 // LW: existence check if ( *(char *)(&PseudoLinkedPointer+4+4+4) != 0 )
11,532 // RW: existence check if ( *(char *)(&PseudoLinkedPointer+4+4+4+wordlen) != 0 )
11,533 // here ALWAYS RW exists: no need for existence check - line below
11,534 // if ( *(char *)(&PseudoLinkedPointer+4+4+4+wordlen) != 0 )
11,535 // if (memcmp(&PseudoLinkedPointer+4+4+4+wordlen, wrd, wordlen) > 0) // go MP
11,536 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,537 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
11,538 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,539 //fread(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
11,540 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,541 // [ //r.14+
11,542 memcpy( &FourGramL[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
11,543 // ] //r.14+
11,544 //strFLAG=strcmpKAZE13(FourGramL, wrd);
11,545 strFLAG=memcmp(FourGramL+1, wrd+1, wordlen); // Since 2019-Feb-07, it is not ASCIIZ string but zero byte is the length of the 1..
11,546 if (strFLAG > 0) // go MP
11,547 { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
11,548 PseudoLinkedPointer = PseudoLinkedPointerNEW;
11,549 }
11,550 {
11,551 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,552 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8; //MP
11,553 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,554 if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
11,555 BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
11,556 BSTstack[StackPtr] = 8; ++StackPtr; //LPOffset=0;MPOffset=8;RPOffset=16;
11,557 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,558 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,559 //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
11,560 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,561 // [ //r.14+
11,562 memcpy( &PseudoLinkedPointer_64, &LEAF[8], 8 );
11,563 // ] //r.14+
11,564 }
11,565 // else if (memcmp(&PseudoLinkedPointer+4+4+4+wordlen, wrd, wordlen) < 0) // go RP
11,566 else if (strFLAG < 0) // go RP
11,567 { // No ?W after RW - go RP
11,568 memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4 + 4, 4 ); //RP
11,569 PseudoLinkedPointer = PseudoLinkedPointerNEW;
11,570 }
11,571 {
11,572 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,573 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8; //RP
11,574 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,575 if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
11,576 BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
11,577 BSTstack[StackPtr] = 16; ++StackPtr; //LPOffset=0;MPOffset=8;RPOffset=16;
11,578 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,579 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,580 //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
11,581 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,582 // [ //r.14+
11,583 memcpy( &PseudoLinkedPointer_64, &LEAF[8 + 8], 8 );
11,584 // ] //r.14+
11,585 }
11,586 else { FoundInLinkedList = 1; // wrd is RW
11,587 // Counter [
11,588 if (BSTorBtree == 2) {

```

```

11,589         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,590     }
11,591     //memcpy( &CounterOccurrences, &FourGramL[(KeySize+1+4)-4], 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
11,592     //if (CounterOccurrences<999999999) CounterOccurrences++; // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
11,593     //memcpy( &FourGramL[(KeySize+1+4)-4], &CounterOccurrences, 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
11,594     // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,595     //fwrite(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
11,596     // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,597     // [ //r.14+
11,598     //     memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &FourGramL[0], (KeySize+1+8) );
11,599     //     if (BSTorBtree == 2) {
11,600     //         fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
11,601     //     } else { // ##### 64bit memory manipulations [
11,602     //         memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
11,603     //     } // ##### 64bit memory manipulations ]
11,604     // ] //r.14+
11,605     // Counter ]
11,606     }
11,607     WORDcountAttemptsToPut++;
11,608     // ***** 'P W P' section 2 ]
11,609     // ++++++
11,610     }
11,611     else // RW empty - go MP
11,612     {
11,613         memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
11,614         PseudoLinkedPointer = PseudoLinkedPointerNEW;
11,615     }
11,616     // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,617     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8; //MP
11,618     // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,619     if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
11,620     BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
11,621     BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
11,622     // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,623     //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,624     //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
11,625     // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,626     // [ //r.14+
11,627     memcpy( &PseudoLinkedPointer_64, &LEAF[8], 8 );
11,628     // ] //r.14+
11,629     }
11,630     }
11,631     else { FoundInLinkedList = 1; // wrd is LW
11,632     // Counter [
11,633     //     if (BSTorBtree == 2) {
11,634     //         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,635     //     }
11,636     //     memcpy( &CounterOccurrences, &FourGramL[(KeySize+1+4)-4], 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
11,637     //     if (CounterOccurrences<999999999) CounterOccurrences++; // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
11,638     //     memcpy( &FourGramL[(KeySize+1+4)-4], &CounterOccurrences, 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
11,639     // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,640     //fwrite(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
11,641     // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,642     // [ //r.14+
11,643     //     memcpy( &LEAF[8 + 8 + 8], &FourGramL[0], (KeySize+1+8) );
11,644     //     if (BSTorBtree == 2) {
11,645     //         fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
11,646     //     } else { // ##### 64bit memory manipulations [

```



```

11,647 //          memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8*8+2*(KeySize+1*8) );
11,648 //          } // ##### 64bit memory manipulations ]
11,649 // ] //r.14+
11,650 // Counter ]
11,651 // }
11,652 // WORDcountAttemptsToPut++;
11,653 // ***** 'P W P' section ]
11,654 // } // while
11,655 // WORDcountAttemptsToPut--; // - 1 due to BST way of counting i.e. direct hash hit is not counted only successors
11,656 // }
11,657 // 1] Search ]
11,658 if (FoundInLinkedList == 0) NotFoundKeys++; //r.18
11,659 if (FoundInLinkedList == 1) FoundKeys++; //r.18
11,660 if (FoundInLinkedList == 2) NotFoundKeys++; //r.18
11,661
11,662 if (FoundInLinkedList == 0)
11,663 {
11,664 /*
11,665 // ===== [ The whole section/sub-fragment 2 is commented due to great time differences for Internal_vs_External memory accesses - it is far more
cheap to have the STACK overhead (moved to sub-fragment 1) ] ===== [
11,666 // 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search [
11,667 // 'TracingSearch' is the same as 'Search' except that adds the trail in my simulated stack,
11,668 // the goal is not to waste time in 'Search' by dealing with no needed trail in case of not 'Insert'.
11,669 // Simulated stack contains pairs of 'Address of ParentLEAF' + 'Offset of ParentPointer in ParentLEAF i.e. 0 for LP, 4 for MP, 8 for RP'.
11,670 // 'Offset ...' saves unnecessary comparisons of NEWword which after splitting goes up.
11,671 //          memcpy( &PseudoLinkedPointer, BufStart+Slot, 4 );
11,672 //          StackPtr = 0;
11,673 //          while (PseudoLinkedPointer != 0)
11,674 //          {
11,675 // ***** 'P W P' section [
11,676 // LW: existence check if ( *(char *) (PseudoLinkedPointer+4*4) != 0 )
11,677 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4*4+wordlen) != 0 )
11,678 // here ALWAYS LW exists: no need for existence check - line below
11,679 // if ( *(char *) (PseudoLinkedPointer+4*4) != 0 )
11,680 // if (memcmp(PseudoLinkedPointer+4*4, wrd, wrdlen) > 0) // go LP
11,681 // { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 0, 4 ); //LP
11,682 // if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
11,683 // BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
11,684 // BSTstack[StackPtr] = 0; ++StackPtr; //LPoffset=0;MPoffset=4;RPoffset=8;
11,685 // PseudoLinkedPointer = PseudoLinkedPointerNEW;
11,686 // }
11,687 // else if (memcmp(PseudoLinkedPointer+4*4, wrd, wrdlen) < 0) // go RP or MP
11,688 // { // RW existence check - line below:
11,689 // if ( *(char *) (PseudoLinkedPointer+4*4+wordlen) != 0 ) // RW exists
11,690 // { // Here all 'P W P' section is repeated; the way of handling case when dynamic number of words in leaf
11,691 // *****
11,692 // ***** 'P W P' section 2 [
11,693 // LW: existence check if ( *(char *) (PseudoLinkedPointer+4*4) != 0 )
11,694 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4*4+wordlen) != 0 )
11,695 // here ALWAYS RW exists: no need for existence check - line below
11,696 // if ( *(char *) (PseudoLinkedPointer+4*4+wordlen) != 0 )
11,697 // if (memcmp(PseudoLinkedPointer+4*4+wordlen, wrd, wrdlen) > 0) // go MP
11,698 // { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
11,699 // if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
11,700 // BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
11,701 // BSTstack[StackPtr] = 4; ++StackPtr; //LPoffset=0;MPoffset=4;RPoffset=8;
11,702 // PseudoLinkedPointer = PseudoLinkedPointerNEW;
11,703 // }

```

```

11,704         else if (memcmp(PseudoLinkedPointer+4*4+4*wrklen, wrd, wrklen) < 0) // go RP
11,705             { // No ?W after RW - go RP
11,706                 memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4 + 4, 4 ); //RP
11,707                 if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
11,708                 BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
11,709                 BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=4;RPoffset=8;
11,710                 PseudoLinkedPointer = PseudoLinkedPointerNEW;
11,711             }
11,712         else FoundInLinkedList = 1; // wrd is RW
11,713 // ***** 'P W P' section 2 ]
11,714 // *****
11,715             }
11,716         else // RW empty - go MP
11,717             { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
11,718                 if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
11,719                 BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
11,720                 BSTstack[StackPtr] = 4; ++StackPtr; //LPoffset=0;MPoffset=4;RPoffset=8;
11,721                 PseudoLinkedPointer = PseudoLinkedPointerNEW;
11,722             }
11,723         }
11,724         else FoundInLinkedList = 1; // wrd is LW
11,725 // ***** 'P W P' section ]
11,726         } // while
11,727 // 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search ]
11,728 // ===== [ The whole section/sub-fragment 2 is commented due to great time differences for Internal_vs_External memory accesses - it is far more
cheap to have the STACK overhead (moved to sub-fragment 1) ] ===== ]
11,729 */
11,730
11,731 // 3] Insert Iterative [
11,732 //     There are total 4 situations:
11,733 //     Case #1: Outer NODE(including ROOT) [ ][ ][ ][LW][ ]
11,734 //     Case #2: Outer NODE(including ROOT) [ ][ ][ ][LW][RW] Split Occurs -----
11,735 //     Case #3: ROOT [LP][MP][ ][LW][ ] | 'wrdUP' (wrklen bytes)
11,736 //     Case #4: Inner NODE(including ROOT) [LP][MP][RP][LW][RW] Split Occurs --- | &
11,737 // | 'PseudoLinkedPointerNEW' (ptr to NEW LEAF)
11,738 // There are total 2 situations for PARENT LEAF: <----- ARE GOING UP
11,739 // Case #3: [LP][MP][ ][LW][ ]
11,740 // Case #4: [LP][MP][RP][LW][RW] Split Occurs
11,741
11,742 // ~ First deal alonely with the OUTER NODE(LEAF) where Search stopped i.e Case #1 & Case #2:
11,743     PoffsetInLEAF = BSTstack[--StackPtr];
11,744     PseudoLinkedPointer_64 = BSTstack[--StackPtr];
11,745 // NOTE: ONE LEAF IS FULL ONLY WHEN LAST CELL FOR KEY(here RW) EXISTS!
11,746 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4*4+4*wrklen) != 0 )
11,747 //if ( *(char *) (PseudoLinkedPointer+4*4+4*wrklen) != 0 ) // If LEAF is full: Case #2
11,748 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,749 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
11,750 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,751 //fread(&SomeByte, 1, 1, fp_outRG);
11,752 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,753 // [ //r.14+
11,754     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
11,755     if (BSTorBtree == 2) {
11,756         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,757         fread(&LEAF[0], 8*8+8*2*(KeySize+1+8), 1, fp_outRG); Total_fread++;
11,758     } else { // ##### 64bit memory manipulations [
11,759         memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64, 8*8+8*2*(KeySize+1+8) );
11,760     } // ##### 64bit memory manipulations ]

```

```

11,761         memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (KeySize+1+8)], 1 );
11,762         // ] //r.14+
11,763         if (SomeByte != 0) // RW exists
11,764     { SplitOccured = 1; WORDcountDistinct++;
11,765         // ALlocate NEW LEAF:
11,766     //         if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrklen + 4 + 4 + 4 < GRMBLFoolAgain((int)wrklen) ) // +4 more for BST instead of LL; + more(see
LEAF)
11,767     //         {
11,768     //             memcpy( &PseudoLinkedPointerNEW, &bufend[LetterOffset], 4 );
11,769     //             bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
11,770     //             bufend[LetterOffset] = bufend[LetterOffset] + 2*wrklen;
11,771     //             if (MAXusedBuffer[wrklen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrklen] = (unsigned long)(bufend[LetterOffset] -
BufStart);}
11,772     //         }
11,773     //         // BUG IN R.17 and prior - the next 'if' is only for RAM (y/Y) not for SSD (z/Z):
11,774     //         if (BSTorBtree == 2) { //r.18
11,775     BUGGYoffset = (BufEnd_64-(0+14));
11,776     //         } else { // ##### 64bit memory manipulations [
11,777     BUGGYoffset = (BufEnd_64-(unsigned long long)pointerflush_64);
11,778     //         } // ##### 64bit memory manipulations ]
11,779     //         if( 8 + 8 + 8 + 2*(KeySize+1+8) < size_in64_L14 - BUGGYoffset ) // the longest wrklen is LongestLineInclusive but actual is LongestLineInclusive(+ CR char)
11,780     //         {
11,781     //             PseudoLinkedPointerNEW_64 = BufEnd_64;
11,782     //             BufEnd_64 = BufEnd_64 + 8 + 8 + 8;
11,783     //             BufEnd_64 = BufEnd_64 + 2*(KeySize+1+8);
11,784     //         }
11,785     //         else
11,786     //         { printf( "\nLeprechaun: Failure! Increment 'Memory for B-trees'!\n" );
11,787     //             fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, 11TOaDigits, 10), _ui64toaKAZEcomma((unsigned long
long)WORDcountDistinct, 11TOaDigits2, 10) );
11,788     //             fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14, 11TOaDigits, 10) );
11,789     //             fprintf( fp_outLOG, "Total Attempts to Find/Put WORDS into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11TOaDigits, 10) );
11,790     //             fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
11,791     //             fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for B-trees'!\n\n");
11,792     //             exit( 7 );
11,793     //         }
11,794     //         if (POffsetInLEAF == 0) // wrd < LW
11,795     //         {
11,796     //             memcpy( wrdUP, PseudoLinkedPointer+4+4+4, wrklen ); // LW up
11,797     //             memcpy( PseudoLinkedPointer+4+4+4, wrd, wrklen ); // wrd go to OLD LEAF
11,798     //             memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrklen, wrklen ); // RW go to NEW LEAF
11,799     //             *(char *) (PseudoLinkedPointer+4+4+4+wrklen) = 0; // RW mark unused in OLD LEAF
11,800     //         [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,801     //         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
11,802     //         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,803     //         //fread(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
11,804     //         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,805     //         //fwrite(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
11,806     //         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerAUX_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
11,807     //         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,808     //         //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
11,809     //         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
11,810     //         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,811     //         //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
11,812     //         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
11,813     //         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,814     //         //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
11,815     //         ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!

```

```

11,816 // [ //r.14+
11,817 memcpy( &wrdUP[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
11,818 memcpy( &LEAF[8 + 8 + 8], &wrd[0], (KeySize+1+8) );
11,819 memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
11,820 // Here reordering (of writing wrdAUX) is needed to avoid seek the position NEW and stupidly to seek again OLD/current position!
11,821 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
11,822 if (BSTorBtree == 2) {
11,823 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,824 fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
11,825 } else { // ##### 64bit memory manipulations [
11,826 memcpy( (char *)&PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
11,827 } // ##### 64bit memory manipulations ]
11,828 PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
11,829 if (BSTorBtree == 2) {
11,830 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,831 fwrite(&wrdAUX[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
11,832 } else { // ##### 64bit memory manipulations [
11,833 memcpy( (char *)&PseudoLinkedPointerAUX_64, &wrdAUX[0], (KeySize+1+8) );
11,834 } // ##### 64bit memory manipulations ]
11,835 // ] //r.14+
11,836 }
11,837 if (POffsetInLEAF == 8) // LW < wrd < RW
11,838 {
11,839 // memcpy( wrdUP, wrd, wrdlen ); // wrd up
11,840 // memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
11,841 // *(char *)&(PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
11,842 // memcpy( wrdUP, wrd, (KeySize+1+8) ); // wrd up
11,843 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,844 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
11,845 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,846 //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
11,847 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
11,848 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,849 //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
11,850 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
11,851 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,852 //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
11,853 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,854 // [ //r.14+
11,855 memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
11,856 // Here reordering (of writing wrdAUX) is needed to avoid seek the position NEW and stupidly to seek again OLD/current position!
11,857 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
11,858 if (BSTorBtree == 2) {
11,859 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,860 fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
11,861 } else { // ##### 64bit memory manipulations [
11,862 memcpy( (char *)&PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
11,863 } // ##### 64bit memory manipulations ]
11,864 PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
11,865 if (BSTorBtree == 2) {
11,866 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,867 fwrite(&wrdAUX[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
11,868 } else { // ##### 64bit memory manipulations [
11,869 memcpy( (char *)&PseudoLinkedPointerAUX_64, &wrdAUX[0], (KeySize+1+8) );
11,870 } // ##### 64bit memory manipulations ]
11,871 // ] //r.14+
11,872 }
11,873 if (POffsetInLEAF == 16) // wrd > RW

```

```

11,874     {
11,875         memcpy( wrdUP, PseudoLinkedPointer+4+4+4* wrdlen, wrdlen ); // RW up
11,876         *(char *) (PseudoLinkedPointer+4+4+4* wrdlen) = 0; // RW mark unused in OLD LEAF
11,877         memcpy( PseudoLinkedPointerNEW+4+4+4, wrd, wrdlen ); // wrd go to NEW LEAF
11,878         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,879         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
11,880         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,881         //fread(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
11,882         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
11,883         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,884         //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
11,885         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
11,886         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,887         //fwrite(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
11,888         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,889         // [ //r.14+
11,890         memcpy( &wrdUP[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
11,891         memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
11,892         if (BSTorBtree == 2) {
11,893             fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,894             fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
11,895             } else { // ##### 64bit memory manipulations [
11,896             memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
11,897             } // ##### 64bit memory manipulations ]
11,898         // ] //r.14+
11,899         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one! Here NO need!
11,900         PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
11,901         if (BSTorBtree == 2) {
11,902             fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,903             fwrite(&wrd[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
11,904             } else { // ##### 64bit memory manipulations [
11,905             memcpy( (char *)PseudoLinkedPointerAUX_64, &wrd[0], (KeySize+1+8) );
11,906             } // ##### 64bit memory manipulations ]
11,907         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one! Here NO need!
11,908     }
11,909 }
11,910 else // If LEAF is not full: Case #1
11,911 { SplitOccured = 0; WORDcountDistinct++;
11,912     if (PoffsetInLEAF == 0) // wrd < [LW][ ] so [LW][ ] -> [ ][LW] -> [wrd][LW]
11,913     {
11,914         //         memcpy( PseudoLinkedPointer+4+4+4* wrdlen, PseudoLinkedPointer+4+4+4, wrdlen );
11,915         //         memcpy( PseudoLinkedPointer+4+4+4, wrd, wrdlen );
11,916         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,917         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
11,918         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,919         //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
11,920         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
11,921         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,922         //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
11,923         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
11,924         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,925         //fwrite(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
11,926         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,927         // [ //r.14+
11,928         memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
11,929         memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &wrdAUX[0], (KeySize+1+8) );
11,930         memcpy( &LEAF[8 + 8 + 8], &wrd[0], (KeySize+1+8) );
11,931         if (BSTorBtree == 2) {

```

```

11,932         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,933         fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
11,934     } else { // ##### 64bit memory manipulations [
11,935         memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
11,936     } // ##### 64bit memory manipulations ]
11,937     // ] //r.14+
11,938
11,939 }
11,940 if (POffsetInLEAF == 8) // wrd > [LW][ ] so [LW][ ] -> [LW][wrd]
11,941 {
11,942     //         memcpy( PseudoLinkedPointer+4+4+4+wrdlen, wrd, wrdlen );
11,943     // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one! Here NO need!
11,944     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (KeySize+1+8);
11,945     if (BSTorBtree == 2) {
11,946         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,947         fwrite(&wrd[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
11,948     } else { // ##### 64bit memory manipulations [
11,949         memcpy( (char *)PseudoLinkedPointerAUX_64, &wrd[0], (KeySize+1+8) );
11,950     } // ##### 64bit memory manipulations ]
11,951     // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one! Here NO need!
11,952 }
11,953 }
11,954
11,955 if (SplitOccured != 0)
11,956 {
11,957     // ~ Second deal with the INNER NODE(S) i.e Case #3 & Case #4:
11,958     while (StackPtr != 0 || SplitOccured != 0)
11,959     {
11,960         // 'PseudoLinkedPointerNEW' is new LEAF to be inserted
11,961         // 'wrdUP' is NEW word to be inserted
11,962         if (StackPtr != 0)
11,963         {
11,964             POffsetInLEAF = BSTstack[--StackPtr];
11,965             PseudoLinkedPointer_64 = BSTstack[--StackPtr];
11,966             //if ( *(char *)PseudoLinkedPointer+4+4+4+wrdlen) != 0 ) // If LEAF is full: Case #4
11,967             // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,968             //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
11,969             //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,970             //fread(&SomeByte, 1, 1, fp_outRG);
11,971             // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
11,972             // [ //r.14+
11,973             PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
11,974             if (BSTorBtree == 2) {
11,975                 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
11,976                 fread(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fread++;
11,977             } else { // ##### 64bit memory manipulations [
11,978                 memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64, 8+8+8+2*(KeySize+1+8) );
11,979             } // ##### 64bit memory manipulations ]
11,980             memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (KeySize+1+8)], 1 );
11,981             // ] //r.14+
11,982             if (SomeByte != 0 ) // RW exists
11,983             { SplitOccured = 1;
11,984                 //         memcpy( wrdUPold, wrdUP, wrdlen ); // LW up
11,985                 //         PseudoLinkedPointerNEWold = PseudoLinkedPointerNEW;
11,986                 //         memcpy( wrdUPold, wrdUP, (KeySize+1+8) );
11,987                 //         PseudoLinkedPointerNEWold_64 = PseudoLinkedPointerNEW_64;
11,988                 // ALlocate NEW LEAF:
11,989                 //         if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrdlen + 4 + 4 + 4 < GRMBLFoolAgain[(int)wrdlen] ) // +4 more for BST instead of LL; + more(see

```

```

LEAF)
11,990 //      {
11,991 //      memcpy( &PseudoLinkedPointerNEW, &bufend[LetterOffset], 4 );
11,992 //      bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
11,993 //      bufend[LetterOffset] = bufend[LetterOffset] + 2*wrklen;
11,994 //      if (MAXusedBuffer[wrklen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrklen] = (unsigned long)(bufend[LetterOffset] -
BufStart);}
11,995 //      }
11,996 //      // BUG IN R.17 and prior - the next 'if' is only for RAM (y/Y) not for SSD (z/Z):
11,997 //      if (BSTorBtree == 2) { //r.18
11,998 BUGGYoffset = (BufEnd_64-(0+14));
11,999 //      } else { // ##### 64bit memory manipulations [
12,000 BUGGYoffset = (BufEnd_64-(unsigned long long)pointerflush_64);
12,001 //      } // ##### 64bit memory manipulations ]
12,002 if( 8 + 8 + 8 + 2*(KeySize+1+8) < size_in64_L14 - BUGGYoffset ) // the longest wrklen is LongestLineInclusive but actual is LongestLineInclusive(+ CR char)
12,003 {
12,004     PseudoLinkedPointerNEW_64 = BufEnd_64;
12,005     BufEnd_64 = BufEnd_64 + 8 + 8 + 8;
12,006     BufEnd_64 = BufEnd_64 + 2*(KeySize+1+8);
12,007     // [ //r.14+
12,008     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64;
12,009     //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,010     //fread(&LEAFNEW[0], 8+8+2*(LongestLineInclusive+1+4), 1, fp_outRG);
12,011     // In fact above three lines are slow, the only need is ZEROed LEAFNEW.
12,012     memset(&LEAFNEW[0], 0, 8+8+2*(KeySize+1+8));
12,013     // ] //r.14+
12,014 }
12,015 else
12,016 { printf( "\nLeprechaun: Failure! Increment 'Memory for B-trees'!\n" );
12,017 //     fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, 11TOaDigits, 10), _ui64toaKAZEcomma((unsigned long
long)WORDcountDistinct, 11TOaDigits2, 10) );
12,018     fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14, 11TOaDigits, 10) );
12,019     fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11TOaDigits, 10) );
12,020     fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
12,021     fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for B-trees'!\n\n" );
12,022     exit( 7 );
12,023 }
12,024 if (POffsetInLEAF == 0) // wrdUPold < LW
12,025 {
12,026 //     memcpy( wrdUP, PseudoLinkedPointer+4+4+4, wrklen ); // LW up
12,027 //     memcpy( PseudoLinkedPointer+4+4+4, wrdUPold, wrklen ); // wrdUPold go to OLD LEAF
12,028 //     memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrklen, wrklen ); // RW go to NEW LEAF
12,029 //     *(char *)(&PseudoLinkedPointer+4+4+4+wrklen) = 0; // RW mark unused in OLD LEAF
12,030 //     // [LP](PseudoLinkedPointerNEWold)[MP][RP](wrdUPold)[LW][RW] -----
12,031 //     // pair [LW] PseudoLinkedPointerNEW goes up !
12,032 //     // PseudoLinkedPointer: PseudoLinkedPointerNEW: !
12,033 //     // [LP](PseudoLinkedPointerNEWold)[](wrdUPold) [MP][RP][][RW] <----
12,034 //     // no need to put zero in RP because logic is based on words existence:
12,035 //     memcpy( PseudoLinkedPointerNEW+0, PseudoLinkedPointer+4, 4 );
12,036 //     memcpy( PseudoLinkedPointerNEW+4, PseudoLinkedPointer+8, 4 );
12,037 //     memcpy( PseudoLinkedPointer+4, &PseudoLinkedPointerNEWold, 4 );
12,038 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
12,039 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
12,040 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,041 //fread(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
12,042 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,043 //fwrite(&wrdUPold[0], (LongestLineInclusive+1+4), 1, fp_outRG);
12,044 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);

```

```

12,045 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,046 //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
12,047 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
12,048 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,049 //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
12,050 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
12,051 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,052 //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
12,053 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
12,054 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,055 //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
12,056 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 0;
12,057 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,058 //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
12,059 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
12,060 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,061 //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
12,062 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8;
12,063 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,064 //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
12,065 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
12,066 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,067 //fwrite(&PseudoLinkedPointerNEWold_64, 8, 1, fp_outRG);
12,068 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
12,069 // [ //r.14+
12,070 memcpy( &wrdUP[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
12,071 memcpy( &LEAF[8 + 8 + 8], &wrdUPold[0], (KeySize+1+8) );
12,072 memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
12,073 memcpy( &LEAFNEW[8 + 8 + 8], &wrdAUX[0], (KeySize+1+8) );
12,074 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
12,075 memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[8], 8 );
12,076 memcpy( &LEAFNEW[0], &PseudoLinkedPointerAUXdumbo_64, 8 );
12,077 memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[16], 8 );
12,078 memcpy( &LEAFNEW[8], &PseudoLinkedPointerAUXdumbo_64, 8 );
12,079 memcpy( &LEAF[8], &PseudoLinkedPointerNEWold_64, 8 );
12,080 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
12,081 if (BSTorBtree == 2) {
12,082 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,083 fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
12,084 } else { // ##### 64bit memory manipulations [
12,085 memcpy( (char *)&PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
12,086 } // ##### 64bit memory manipulations ]
12,087 PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64;
12,088 if (BSTorBtree == 2) {
12,089 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,090 fwrite(&LEAFNEW[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
12,091 } else { // ##### 64bit memory manipulations [
12,092 memcpy( (char *)&PseudoLinkedPointerAUX_64, &LEAFNEW[0], 8+8+8+2*(KeySize+1+8) );
12,093 } // ##### 64bit memory manipulations ]
12,094 // ] //r.14+
12,095 }
12,096 if (PoffsetInLEAF == 8) // LW < wrdUPold < RW
12,097 {
12,098 // memcpy( wrdUP, wrdUPold, wrdlen ); // wrdUPold up
12,099 // memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
12,100 // *(char *)&(PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
12,101 // // [LP][MP](PseudoLinkedPointerNEWold)[RP][LW](wrdUPold)[RW] -----
12,102 // // pair [wrdUPold] PseudoLinkedPointerNEW goes up !

```



```

12,103 //          // PseudoLinkedPointer: PseudoLinkedPointerNEW:
12,104 //          // [LP][MP][][LW]          (PseudoLinkedPointerNEWold)[RP][][RW]          <----
12,105 //          // no need to put zero in RP because logic is based on words existence:
12,106 //          memcpy( PseudoLinkedPointerNEW+0, &PseudoLinkedPointerNEWold, 4 );
12,107 //          memcpy( PseudoLinkedPointerNEW+4, PseudoLinkedPointer+8, 4 );
12,108 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
12,109 //          memcpy( wrdUP, wrdUPold, (KeySize+1+8) );
12,110 //          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
12,111 //          //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,112 //          //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
12,113 //          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
12,114 //          //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,115 //          //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
12,116 //          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
12,117 //          //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,118 //          //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
12,119 //          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 0;
12,120 //          //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,121 //          //fwrite(&PseudoLinkedPointerNEWold_64, 8, 1, fp_outRG);
12,122 //          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
12,123 //          //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,124 //          //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
12,125 //          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8;
12,126 //          //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,127 //          //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
12,128 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
12,129 // [ //r.14+
12,130 //          memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
12,131 //          memcpy( &LEAFNEW[8 + 8 + 8], &wrdAUX[0], (KeySize+1+8) );
12,132 //          memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
12,133 //          memcpy( &LEAFNEW[0], &PseudoLinkedPointerNEWold_64, 8 );
12,134 //          memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[16], 8 );
12,135 //          memcpy( &LEAFNEW[8], &PseudoLinkedPointerAUXdumbo_64, 8 );
12,136 //          PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
12,137 //          if (BSTorBtree == 2) {
12,138 //              fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,139 //              fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
12,140 //          } else { // ##### 64bit memory manipulations [
12,141 //              memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
12,142 //          } // ##### 64bit memory manipulations ]
12,143 //          PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64;
12,144 //          if (BSTorBtree == 2) {
12,145 //              fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,146 //              fwrite(&LEAFNEW[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
12,147 //          } else { // ##### 64bit memory manipulations [
12,148 //              memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAFNEW[0], 8+8+8+2*(KeySize+1+8) );
12,149 //          } // ##### 64bit memory manipulations ]
12,150 //          // ] //r.14+
12,151 //      }
12,152 //      if (PoffsetInLEAF == 16) // wrdUPold > RW
12,153 //      {
12,154 //          memcpy( wrdUP, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW up
12,155 //          *(char *)PseudoLinkedPointer+4+4+4+wrdlen = 0; // RW mark unused in OLD LEAF
12,156 //          memcpy( PseudoLinkedPointerNEW+4+4+4, wrdUPold, wrdlen ); // wrdUPold go to NEW LEAF
12,157 //          // [LP][MP][RP](PseudoLinkedPointerNEWold)[LW][RW](wrdUPold) -----
12,158 //          // pair [RW] PseudoLinkedPointerNEW goes up
12,159 //          // PseudoLinkedPointer: PseudoLinkedPointerNEW:
12,160 //          // [LP][MP][][LW]          [RP](PseudoLinkedPointerNEWold)[](wrdUPold) <---

```

```

12,161 //          // no need to put zero in RP because logic is based on words existence:
12,162 //          memcpy( PseudoLinkedPointerNEW+0, PseudoLinkedPointer+8, 4 );
12,163 //          memcpy( PseudoLinkedPointerNEW+4, &PseudoLinkedPointerNEWold, 4 );
12,164 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
12,165 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
12,166 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,167 //fread(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
12,168 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
12,169 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,170 //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
12,171 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
12,172 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,173 //fwrite(&wrdUPold[0], (LongestLineInclusive+1+4), 1, fp_outRG);
12,174 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
12,175 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,176 //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
12,177 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 0;
12,178 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,179 //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
12,180 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8;
12,181 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,182 //fwrite(&PseudoLinkedPointerNEWold_64, 8, 1, fp_outRG);
12,183 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
12,184 // [ //r.14+
12,185 memcpy( &wrdUP[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
12,186 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
12,187 memcpy( &LEAFNEW[8 + 8 + 8], &wrdUPold[0], (KeySize+1+8) );
12,188 memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[16], 8 );
12,189 memcpy( &LEAFNEW[0], &PseudoLinkedPointerAUXdumbo_64, 8 );
12,190 memcpy( &LEAFNEW[8], &PseudoLinkedPointerNEWold_64, 8 );
12,191 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
12,192 if (BSTorBtree == 2) {
12,193 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,194 fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
12,195 } else { // ##### 64bit memory manipulations [
12,196 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
12,197 } // ##### 64bit memory manipulations ]
12,198 PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64;
12,199 if (BSTorBtree == 2) {
12,200 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,201 fwrite(&LEAFNEW[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
12,202 } else { // ##### 64bit memory manipulations [
12,203 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAFNEW[0], 8+8+8+2*(KeySize+1+8) );
12,204 } // ##### 64bit memory manipulations ]
12,205 // ] //r.14+
12,206 }
12,207 }
12,208 else // If LEAF is not full: Case #3
12,209 { SplitOccured = 0;
12,210 if (POffsetInLEAF == 0) // wrdUP < [LW][ ] so [LW][ ] -> [ ][LW] -> [wrdUP][LW]
12,211 {
12,212 //          memcpy( PseudoLinkedPointer+4+4+4+wrdlen, PseudoLinkedPointer+4+4+4, wrdlen );
12,213 //          memcpy( PseudoLinkedPointer+4+4+4, wrdUP, wrdlen );
12,214 //          // [LP][MP][ ] -> [LP][ ][MP] -> [LP][np][MP]
12,215 //          memcpy( PseudoLinkedPointer+8, PseudoLinkedPointer+4, 4 );
12,216 //          memcpy( PseudoLinkedPointer+4, &PseudoLinkedPointerNEW, 4 );
12,217 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
12,218 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;

```

```

12,219 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,220 //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
12,221 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + (LongestLineInclusive+1+4);
12,222 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,223 //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
12,224 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
12,225 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,226 //fwrite(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
12,227 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
12,228 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,229 //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
12,230 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
12,231 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,232 //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
12,233 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
12,234 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,235 //fwrite(&PseudoLinkedPointerNEW_64, 8, 1, fp_outRG);
12,236 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
12,237 // [ //r.14+
12,238 memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
12,239 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &wrdAUX[0], (KeySize+1+8) );
12,240 memcpy( &LEAF[8 + 8 + 8], &wrdUP[0], (KeySize+1+8) );
12,241 memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[8], 8 );
12,242 memcpy( &LEAF[8 + 8], &PseudoLinkedPointerAUXdumbo_64, 8 );
12,243 memcpy( &LEAF[8], &PseudoLinkedPointerNEW_64, 8 );
12,244 if (BSTorBtree == 2) {
12,245 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,246 fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
12,247 } else { // ##### 64bit memory manipulations [
12,248 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
12,249 } // ##### 64bit memory manipulations ]
12,250 // ] //r.14+
12,251 }
12,252 if (PoffsetInLEAF == 8) // wrdUP > [LW][] so [LW][] -> [LW][wrdUP]
12,253 {
12,254 //     memcpy( PseudoLinkedPointer+4+4+4+wrdlen, wrdUP, wrdlen );
12,255 //     // [LP][MP][] -> [LP][MP][np]
12,256 //     memcpy( PseudoLinkedPointer+8, &PseudoLinkedPointerNEW, 4 );
12,257 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
12,258 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + (LongestLineInclusive+1+4);
12,259 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,260 //fwrite(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
12,261 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
12,262 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,263 //fwrite(&PseudoLinkedPointerNEW_64, 8, 1, fp_outRG);
12,264 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
12,265 // [ //r.14+
12,266 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &wrdUP[0], (KeySize+1+8) );
12,267 memcpy( &LEAF[8 + 8], &PseudoLinkedPointerNEW_64, 8 );
12,268 if (BSTorBtree == 2) {
12,269 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,270 fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
12,271 } else { // ##### 64bit memory manipulations [
12,272 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
12,273 } // ##### 64bit memory manipulations ]
12,274 // ] //r.14+
12,275 }
12,276 break;

```

```

12,277     }
12,278     }
12,279     else // Empty stack means ROOT and more over ROOT is already splitted(Case #4 is off)
12,280     {
12,281     // If LEAF is not full: Case #3
12,282     // THIS IS WHERE A NEW(SECOND) LEAF 'PseudoLinkedPointerROOT' must be allocated:
12,283     // if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrklen + 4 + 4 + 4 < GRMELFoolAgain((int)wrklen) ) // +4 more for BST instead of LL; + more(see
LEAF)
12,284     // {
12,285     //     memcpy( &PseudoLinkedPointerROOT, &bufend[LetterOffset], 4 );
12,286     //     bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
12,287     //     memcpy( bufend[LetterOffset], wrdUP, wrklen );
12,288     //     bufend[LetterOffset] = bufend[LetterOffset] + 2*wrklen;
12,289     //     if (MAXusedBuffer[wrklen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrklen] = (unsigned long)(bufend[LetterOffset] -
BufStart);}
12,290     // }
12,291     // BUG IN R.17 and prior - the next 'if' is only for RAM (y/Y) not for SSD (z/Z):
12,292     if (BSTorBtree == 2) { //r.18
12,293     BUGGYoffset = (BufEnd_64-(0+14));
12,294     } else { // ##### 64bit memory manipulations [
12,295     BUGGYoffset = (BufEnd_64-(unsigned long long)pointerflush_64);
12,296     } // ##### 64bit memory manipulations ]
12,297     if( 8 + 8 + 8 + 2*(KeySize+1+8) < size_in64_L14 - BUGGYoffset ) // the longest wrklen is LongestLineInclusive but actual is LongestLineInclusive(+ CR char)
12,298     {
12,299     PseudoLinkedPointerROOT_64 = BufEnd_64;
12,300     BufEnd_64 = BufEnd_64 + 8 + 8 + 8;
12,301     BufEnd_64 = BufEnd_64 + 2*(KeySize+1+8);
12,302     PseudoLinkedPointerAUX_64 = PseudoLinkedPointerROOT_64 + 8 + 8 + 8;
12,303     if (BSTorBtree == 2) {
12,304     fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,305     fwrite(&wrdUP[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
12,306     } else { // ##### 64bit memory manipulations [
12,307     memcpy( (char *)PseudoLinkedPointerAUX_64, &wrdUP[0], (KeySize+1+8) );
12,308     } // ##### 64bit memory manipulations ]
12,309     }
12,310     else
12,311     { printf( "\nLeprechaun: Failure! Increment 'Memory for B-trees'!\n" );
12,312     fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, 11TOaDigits, 10), _ui64toaKAZEcomma((unsigned long
long)WORDcountDistinct, 11TOaDigits2, 10) );
12,313     fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14, 11TOaDigits, 10) );
12,314     fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11TOaDigits, 10) );
12,315     fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
12,316     fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for B-trees'!\n\n");
12,317     exit( 7 );
12,318     }
12,319     // Here -- 'PseudoLinkedPointerROOT' --
12,320     // | (wrdUP) |
12,321     // 'PseudoLinkedPointer' <-- --> 'PseudoLinkedPointerNEW'
12,322     // (LW) (RW)
12,323     // memcpy( PseudoLinkedPointerROOT, &PseudoLinkedPointer, 4 ); // LP
12,324     // memcpy( PseudoLinkedPointerROOT+4, &PseudoLinkedPointerNEW, 4 ); // MP
12,325     // Here must NEW ROOT be updated i.e. HASH table(SLOT) must point it:
12,326     // memcpy( BufStart+Slot, &PseudoLinkedPointerROOT, 4 );
12,327     PseudoLinkedPointerAUX_64 = PseudoLinkedPointerROOT_64;
12,328     if (BSTorBtree == 2) {
12,329     fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,330     fwrite(&PseudoLinkedPointer_64, 8, 1, fp_outRG); Total_fwrite++;
12,331     } else { // ##### 64bit memory manipulations [

```

```
12,332         memcpy( (char *)PseudoLinkedPointerAUX_64, &PseudoLinkedPointer_64, 8 );
12,333         } // ##### 64bit memory manipulations ]
12,334     PseudoLinkedPointerAUX_64 = PseudoLinkedPointerROOT_64 + 8;
12,335     if (BSTorBtree == 2) {
12,336         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,337         fwrite(&PseudoLinkedPointerNEW_64, 8, 1, fp_outRG); Total_fwrite++;
12,338     } else { // ##### 64bit memory manipulations [
12,339         memcpy( (char *)PseudoLinkedPointerAUX_64, &PseudoLinkedPointerNEW_64, 8 );
12,340         } // ##### 64bit memory manipulations ]
12,341     memcpy( BufStart+Slot, &PseudoLinkedPointerROOT_64, 8 );
12,342     break; //because it is ROOT without split
12,343 }
12,344     } // while
12,345 } //if (SplitOccured != 0)
12,346 // 3] Insert Iterative ]
12,347 } //if (FoundInLinkedList == 0)
12,348 // ##### B-tree order 3 fragment 64bit ]
12,349 } // if (FoundInLinkedList_RAM == 0) { // 2019-Dec-04
12,350 } //if ( (Slot>>HashChunkSizeInBITS) == RipPasses ) {
12,351 } //for (BuildingBlocksSTRIDE=0;
12,352
12,353 // PASS #2: ]
12,354
12,355 RipPasses++;
12,356 if (RipPasses <= (1<<((HashInBITS_GLOBAL-HashChunkSizeInBITS_GLOBAL))-1) goto WhyTheHellForIsNotWorking;
12,357 //} // for( RipPasses = 1-1; RipPasses <= (1<<((HashInBITS-HashChunkSizeInBITS))-1; RipPasses++ )
12,358
12,359 // Counting trees [
12,360 NumberOfTrees=0;
12,361 AllSlots=0;
12,362
12,363 for( j = 0; j < MatchLensNUM; j++ ) //r.18
12,364 {
12,365     for( jjj = 0; jjj < ( ((1LL)<<HashInBITS_GLOBAL) ); jjj++ ) //r.18
12,366     {
12,367         Slot = (AllSlots)<<3; AllSlots++;
12,368         memcpy( &PseudoLinkedPointer_64, BufStart+Slot, 8 );
12,369         if (PseudoLinkedPointer_64 != 0) NumberOfTrees++;
12,370     }
12,371 }
12,372 // Counting trees ]
12,373
12,374 printf( "DONE; %s B-trees have been rooted so far.\n", _ui64toaKAZEzerocomma(NumberOfTrees, llTOaDigits, 10)+(26-14));
12,375 } //for (jj=0;
12,376 printf( "\n");
12,377
12,378 goto SkipDestroyingTheTrees;
12,379 // Stats... [
12,380
12,381 time2=time(NULL); //fix of bigtime
12,382 NumberOfTrees=0;
12,383 NumberOfLEAFs=0;
12,384 LevelsInCorona_Not_Counting_ROOT=0;
12,385 // DUMP and STATS [[
12,386 WORDcountBOTTOM = 0;
12,387 // The dump file with ripped data [
12,388 //if( ( fp_out = fopen( argv[2], "ab+" ) ) == NULL )
12,389 //{ printf( "Leprechaun: Can't create file %s \n", argv[2] ); exit( 7 ); }
```

```

12,390 // The dump file with ripped data ]
12,391 CRdLFa[0] = 13; CRdLFa[1] = 10;
12,392 AllSlots=0;
12,393
12,394 BufStart = pointerflush;
12,395 //      for( j = 0; j < 28*28*28*28*28; j++ )
12,396 //for( j = 0; j < (1<<HashInBITS); j++ )
12,397 for( j = 0; j < MatchLensNUM; j++ ) //r.18
12,398 {
12,399 //KeySize = LongestLineInclusive; // 'LongestLineInclusive' is for 1-way hashing where different KeySizes can occupy same HASH pool.
12,400 KeySize = (MatchLens[j]<<1); //r.18, In case multi-way hashing: either the MAX allowed or the 'wrklen'.
12,401
12,402 for( jjj = 0; jjj < ( ((1LL)<<HashInBITS_GLOBAL) ); jjj++ ) //r.18
12,403 {
12,404 WORDcountBOTTOMPerMatchLen=0; //r.18
12,405         Slot = (AllSlots)<<3; AllSlots++;
12,406         memcpy( &PseudoLinkedPointer_64, BufStart+Slot, 8 );
12,407         if (PseudoLinkedPointer_64 != 0)
12,408         {
12,409         NumberOfTrees++;
12,410
12,411 // CAUTION: Did forget that dumping/traversing the B-trees - DESTROYS them! On purpose, pointers are zeroed.
12,412 // $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ B-tree order 3 traverse 64bit [
12,413         // DONE JOB:
12,414         // Must be written B-tree traverse ! with simulated stack i.e. non-recursive.
12,415         // ...
12,416         StackPtr = 0;
12,417         while ( 2==2 ) {
12,418             while (PseudoLinkedPointer_64 != 0)
12,419             {
12,420                 if (StackPtr > 8192*3-1) { printf( "\nLeprechaun: Failure! B-tree simulated stack overflow, too high B-tree!\n" ); exit( 13 );}
12,421                 BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //ptr to Rword
12,422 //if ( *(char *) (PseudoLinkedPointer + 4 + 4 + 4 + i%31+1) == 0 ) {memcpy( PseudoLinkedPointer + 4 + 4, &BufStart[NumberOfSLOTS*4], 4 );}
12,423                 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
12,424                 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
12,425                 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,426                 //fread(&SomeByte, 1, 1, fp_outRG);
12,427                 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
12,428                 // [ //r.14+
12,429                 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
12,430                 if (BSTorBtree == 2) {
12,431                     fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,432                     fread(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fread++;
12,433                     } else { // ##### 64bit memory manipulations [
12,434                     memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64, 8+8+2*(KeySize+1+8) );
12,435                     } // ##### 64bit memory manipulations ]
12,436                     memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (KeySize+1+8)], 1 );
12,437                     // ] //r.14+
12,438                 if (SomeByte == 0) // RW exists not
12,439                 {
12,440                     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8;
12,441                     if (BSTorBtree == 2) {
12,442                         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,443                         fwrite(&NULLs_64, 8, 1, fp_outRG); Total_fwrite++;
12,444                         } else { // ##### 64bit memory manipulations [
12,445                         memcpy( (char *)PseudoLinkedPointerAUX_64, &NULLs_64, 8 );
12,446                         } // ##### 64bit memory manipulations ]
12,447                     // [ //r.14+

```

```

12,448         memcpy( &LEAF[8 + 8], &NULLs_64, 8 );
12,449         // ] //r.14+
12,450     }
12,451     //         memcpy( &PseudoLinkedPointerNEWleft, PseudoLinkedPointer, 4 );
12,452     //         memcpy( &PseudoLinkedPointerNEWmiddle, PseudoLinkedPointer + 4, 4 );
12,453     //         memcpy( &PseudoLinkedPointerNEWright, PseudoLinkedPointer + 4 + 4, 4 );
12,454     // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
12,455     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
12,456     //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,457     //fread(&PseudoLinkedPointerNEWleft_64, 8, 1, fp_outRG);
12,458     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
12,459     //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,460     //fread(&PseudoLinkedPointerNEWmiddle_64, 8, 1, fp_outRG);
12,461     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8;
12,462     //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,463     //fread(&PseudoLinkedPointerNEWright_64, 8, 1, fp_outRG);
12,464     // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
12,465     // [ //r.14+
12,466     memcpy( &PseudoLinkedPointerNEWleft_64, &LEAF[0], 8 );
12,467     memcpy( &PseudoLinkedPointerNEWmiddle_64, &LEAF[8], 8 );
12,468     memcpy( &PseudoLinkedPointerNEWright_64, &LEAF[8+8], 8 );
12,469     // ] //r.14+
12,470 // Give first from right to left non-zero PTR
12,471     if (PseudoLinkedPointerNEWright_64 !=0 )
12,472     {
12,473         memcpy( PseudoLinkedPointer + 4 + 4, &BufStart[NumberOfSLOTS*4], 4 );
12,474         PseudoLinkedPointer = PseudoLinkedPointerNEWright;
12,475     }
12,476     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8;
12,477     if (BSTorBtree == 2) {
12,478         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,479         fwrite(&NULLs_64, 8, 1, fp_outRG); Total_fwrite++;
12,480     } else { // ##### 64bit memory manipulations [
12,481         memcpy( (char *)PseudoLinkedPointerAUX_64, &NULLs_64, 8 );
12,482     } // ##### 64bit memory manipulations ]
12,483     PseudoLinkedPointer_64 = PseudoLinkedPointerNEWright_64;
12,484 }
12,485 else if (PseudoLinkedPointerNEWmiddle_64 !=0 )
12,486 {
12,487     memcpy( PseudoLinkedPointer + 4, &BufStart[NumberOfSLOTS*4], 4 );
12,488     PseudoLinkedPointer = PseudoLinkedPointerNEWmiddle;
12,489 }
12,490 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
12,491 if (BSTorBtree == 2) {
12,492     fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,493     fwrite(&NULLs_64, 8, 1, fp_outRG); Total_fwrite++;
12,494 } else { // ##### 64bit memory manipulations [
12,495     memcpy( (char *)PseudoLinkedPointerAUX_64, &NULLs_64, 8 );
12,496 } // ##### 64bit memory manipulations ]
12,497 PseudoLinkedPointer_64 = PseudoLinkedPointerNEWmiddle_64;
12,498 }
12,499 else if (PseudoLinkedPointerNEWleft_64 !=0 )
12,500 {
12,501     memcpy( PseudoLinkedPointer, &BufStart[NumberOfSLOTS*4], 4 );
12,502     PseudoLinkedPointer = PseudoLinkedPointerNEWleft;
12,503 }
12,504 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
12,505 if (BSTorBtree == 2) {

```

```

12,506         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,507         fwrite(&NULLs_64, 8, 1, fp_outRG); Total_fwrite++;
12,508         } else { // ##### 64bit memory manipulations [
12,509             memcpy( (char *)PseudoLinkedPointerAUX_64, &NULLs_64, 8 );
12,510         } // ##### 64bit memory manipulations ]
12,511         PseudoLinkedPointer_64 = PseudoLinkedPointerNEWleft_64;
12,512     }
12,513     else
12,514     {
12,515         PseudoLinkedPointer_64 = 0;
12,516     }
12,517 }
12,518 if (LevelsInCorona_Not_Counting_ROOT < StackPtr) LevelsInCorona_Not_Counting_ROOT = StackPtr; //r.14
12,519 if (StackPtr == 0) break;
12,520 PseudoLinkedPointer_64 = BSTstack[--StackPtr];
12,521 //         memcpy( &PseudoLinkedPointerNEWleft, PseudoLinkedPointer, 4 );
12,522 //         memcpy( &PseudoLinkedPointerNEWMiddle, PseudoLinkedPointer + 4, 4 );
12,523 //         memcpy( &PseudoLinkedPointerNEWright, PseudoLinkedPointer + 4 + 4, 4 );
12,524 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
12,525 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
12,526 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,527 //fread(&PseudoLinkedPointerNEWleft_64, 8, 1, fp_outRG);
12,528 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
12,529 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,530 //fread(&PseudoLinkedPointerNEWMiddle_64, 8, 1, fp_outRG);
12,531 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8;
12,532 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,533 //fread(&PseudoLinkedPointerNEWright_64, 8, 1, fp_outRG);
12,534 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
12,535 // [ //r.14+
12,536 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
12,537     if (BSTorBtree == 2) {
12,538         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,539         fread(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fread++;
12,540         } else { // ##### 64bit memory manipulations [
12,541             memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64, 8+8+2*(KeySize+1+8) );
12,542         } // ##### 64bit memory manipulations ]
12,543         memcpy( &PseudoLinkedPointerNEWleft_64, &LEAF[0], 8 );
12,544         memcpy( &PseudoLinkedPointerNEWMiddle_64, &LEAF[8], 8 );
12,545         memcpy( &PseudoLinkedPointerNEWright_64, &LEAF[8+8], 8 );
12,546         // ] //r.14+
12,547     if (PseudoLinkedPointerNEWleft_64 + PseudoLinkedPointerNEWMiddle_64 + PseudoLinkedPointerNEWright_64 == 0) // One LEAF is PRINTED when LP=0 MP=0 RP=0
12,548     {
12,549         memcpy( wrd, PseudoLinkedPointer + 4 + 4 + 4, i%31+1 );
12,550         fwrite(wrd, i%31+1, 1, fp_out);
12,551         fwrite(CRDLFa, 2, 1, fp_out);
12,552         if ( *(char *) (PseudoLinkedPointer + 4 + 4 + 4 + i%31+1) != 0 )
12,553         { memcpy( wrd, PseudoLinkedPointer + 4 + 4 + 4 + i%31+1, i%31+1 );
12,554             fwrite(wrd, i%31+1, 1, fp_out);
12,555             fwrite(CRDLFa, 2, 1, fp_out);
12,556         }
12,557         PseudoLinkedPointer = 0;
12,558         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
12,559         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
12,560         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,561         //fread(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
12,562         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
12,563         // [ //r.14+

```



```

12,564         memcpy( &wrd[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
12,565         // ] //r.14+
12,566         // Counter [
12,567         //memcpy( &CounterOccurrences, &wrd[(KeySize+1+4)-4], 4 );
12,568         //if (CounterOccurrences<999999999) CounterOccurrences++; // Starting from ZERO! Because when insertion happened there was no setting counter to 1.
12,569         // Counter ]
12,570
12,571         WORDcountBOTTOM++; //Nakamichi
12,572         WORDcountBOTTOMPerMatchLen++;
12,573 if (*argv[k_FIX] == 'Y' || *argv[k_FIX] == 'Z')
12,574 //if (CounterOccurrences>1) // For Nakamichi
12,575 //      fprintf(fp_out, "%s\t%s\r\n", _ui64toaKAZEzerocomma(CounterOccurrences, 11ToaDigits2, 10)+(26-11), wrd); WORDcountBOTTOM++;
12,576 if (*argv[k_FIX] == 'y' || *argv[k_FIX] == 'z')
12,577 // NO-DUMP: [
12,578 //      WORDcountBOTTOM++;
12,579 //      fprintf(fp_out, "%s\r\n", wrd); WORDcountBOTTOM++;
12,580 // NO-DUMP: ]
12,581 //fwrite(CRdLFa, 2, 1, fp_out);
12,582 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
12,583 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
12,584 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,585 //fread(&SomeByte, 1, 1, fp_outRG);
12,586 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
12,587 // [ //r.14+
12,588 //      memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (KeySize+1+8)], 1 );
12,589 // ] //r.14+
12,590 if (SomeByte != 0 ) // RW exists
12,591 {
12,592 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
12,593 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
12,594 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
12,595 //fread(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
12,596 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
12,597 // [ //r.14+
12,598 //      memcpy( &wrd[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
12,599 // ] //r.14+
12,600 // Counter [
12,601 //memcpy( &CounterOccurrences, &wrd[(KeySize+1+4)-4], 4 );
12,602 //if (CounterOccurrences<999999999) CounterOccurrences++; // Starting from ZERO! Because when insertion happened there was no setting counter to 1.
12,603 // Counter ]
12,604
12,605         WORDcountBOTTOM++; //Nakamichi
12,606         WORDcountBOTTOMPerMatchLen++;
12,607 if (*argv[k_FIX] == 'Y' || *argv[k_FIX] == 'Z')
12,608 //if (CounterOccurrences>1) // For Nakamichi
12,609 //      fprintf(fp_out, "%s\t%s\r\n", _ui64toaKAZEzerocomma(CounterOccurrences, 11ToaDigits2, 10)+(26-11), wrd); WORDcountBOTTOM++;
12,610 if (*argv[k_FIX] == 'y' || *argv[k_FIX] == 'z')
12,611 // NO-DUMP: [
12,612 //      WORDcountBOTTOM++;
12,613 //      fprintf(fp_out, "%s\r\n", wrd); WORDcountBOTTOM++;
12,614 // NO-DUMP: ]
12,615 //fwrite(CRdLFa, 2, 1, fp_out);
12,616 // [ //r.14+
12,617 //      PseudoLinkedPointer_64 = 0;
12,618 //      NumberOfLEAFs++;
12,619 //      }
12,620 // }
12,621 // } //if (PseudoLinkedPointer_64 != 0)

```

```
12,622 // Stats for one tree only:
12,623 //printf( "\nLeprechaun: Total keys (j=%d) into B-trees order 3 (during traversal/dump): %s\n", j, _ui64toaKAZEcomma(WORDcountBOTTOMPerMatchLen, 11TOaDigits, 10) );
12,624
12,625 } //for( jjj = 0;
12,626 } //for( j = 0;
12,627
12,628 // The dump file with ripped data [
12,629 //fclose(fp_out);
12,630 // The dump file with ripped data ]
12,631
12,632 fprintf( fp_outLOG, "Number Of Hash Collisions(Distinct WORDs - Number Of Trees): %s\n", _ui64toaKAZEcomma(WORDcountDistinct - NumberOfTrees, 11TOaDigits, 10) );
12,633 fprintf( fp_outLOG, "Number Of Trees(GREATER THE BETTER): %s\n", _ui64toaKAZEcomma(NumberOfTrees, 11TOaDigits, 10) );
12,634 fprintf( fp_outLOG, "Number Of LEAFs(littler THE BETTER) not counting ROOT LEAFs: %s\n", _ui64toaKAZEcomma(NumberOfLEAFs-NumberOfTrees, 11TOaDigits, 10) );
12,635 fprintf( fp_outLOG, "Highest Tree not counting ROOT Level i.e. CORONA levels(littler THE BETTER): %s\n", _ui64toaKAZEcomma(LevelsInCorona_Not_Counting_ROOT-1, 11TOaDigits, 10) );
12,636
12,637 fprintf( fp_outLOG, "Used value for third parameter in KB: %s\n", _ui64toaKAZEcomma(Thunderwith, 11TOaDigits, 10) );
12,638 fprintf( fp_outLOG, "Use next time as third parameter: %s\n", _ui64toaKAZEcomma((((BufEnd_64-(unsigned long long)pointerflush_64)+1)>>10)+1, 11TOaDigits, 10) );
12,639 fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11TOaDigits, 10) );
12,640
12,641 // DUMP and STATS ]]]
12,642
12,643 printf( "\n");
12,644 printf( "Leprechaun: Number Of Total/Distinct/Undistinct keys/BBs (all orders): %s/%s/%s\n", _ui64toaKAZEcomma(WORDcount, 11TOaDigits, 10),
12,645 _ui64toaKAZEcomma(WORDcountDistinct, 11TOaDigits2, 10), _ui64toaKAZEcomma(WORDcount-WORDcountDistinct, 11TOaDigits3, 10) );
12,646 printf( "Leprechaun: Number Of TREES(GREATER THE BETTER): %s\n", _ui64toaKAZEcomma(NumberOfTrees, 11TOaDigits, 10) );
12,647 printf( "Leprechaun: Number Of LEAFs(littler THE BETTER) not counting ROOT LEAFs: %s\n", _ui64toaKAZEcomma(NumberOfLEAFs-NumberOfTrees, 11TOaDigits, 10) );
12,648 printf( "Leprechaun: Highest Tree not counting ROOT Level i.e. CORONA levels(littler THE BETTER): %s\n", _ui64toaKAZEcomma(LevelsInCorona_Not_Counting_ROOT-1, 11TOaDigits, 10) );
12,649 printf( "Leprechaun: Used value for B-trees pool in KB: %s\n", _ui64toaKAZEcomma(Thunderwith, 11TOaDigits, 10) );
12,650 printf( "Leprechaun: Use next time for B-trees pool in KB: %s\n", _ui64toaKAZEcomma((((BufEnd_64-(unsigned long long)pointerflush_64)+1)>>10)+1, 11TOaDigits, 10) );
12,651 // printf( "Leprechaun: Total Attempts to Find/Put keys into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11TOaDigits, 10) );
12,652 printf( "Leprechaun: Total keys (all orders) into B-trees order 3 (during traversal/dump): %s\n", _ui64toaKAZEcomma(WORDcountBOTTOM, 11TOaDigits, 10) );
12,653 printf( "Leprechaun: Total keys MISSES/HITS (all orders) into B-trees order 3: %s/%s\n", _ui64toaKAZEcomma(NotFoundKeys, 11TOaDigits, 10),
12,654 _ui64toaKAZEcomma(FoundKeys, 11TOaDigits2, 10) );
12,655 printf( "\n");
12,656 // Stats... ]
12,657
12,658 printf( "Leprechaun: B-trees building speed (counting ROOT LEAFs): %s LEAFs/s\n", _ui64toaKAZEcomma((double)(NumberOfLEAFs)/(double)(time2 - time1 + 1), 11TOaDigits3, 10));
12,659 printf( "Leprechaun: B-trees traverse speed (counting ROOT LEAFs): %s LEAFs/s\n", _ui64toaKAZEcomma((double)(NumberOfLEAFs)/(double)(time(NULL) - time2 + 1), 11TOaDigits3, 10));
12,660 SkipDestroyingTheTrees:
12,661 printf( "Leprechaun: Total Searches-n-Inserts Per Second: %s SNIPS\n", _ui64toaKAZEcomma((double)(WORDcount)/(double)(time(NULL) - time1 + 1), 11TOaDigits3, 10));
12,662 printf( "Leprechaun: RAM needed to house B-trees (relative to the file being ripped): %sN = %sMB\n", _ui64toaKAZEcomma((BufEnd_64-(unsigned long long)pointerflush_64+1)/size_inLINESIXFOUR, 11TOaDigits3, 10), _ui64toaKAZEcomma(1+((BufEnd_64-(unsigned long long)pointerflush_64+1)>>20), 11TOaDigits2, 10));
12,663 //if defined(ExternalRAM)
12,664 if (BSTorBtree == 2) {
12,665 printf( "Leprechaun: Total IOPS for %s 'freads' and %s 'fwrites' (of packets %d bytes long) during loading traversing all orders: %s IOPS\n",
12,666 _ui64toaKAZEcomma(Total_fread, 11TOaDigits, 10), _ui64toaKAZEcomma(Total_fwrite, 11TOaDigits2, 10), 8*8*8*2*(LongestLineInclusive+1*8),
12,667 _ui64toaKAZEcomma((double)(Total_fwrite+Total_fread)/(double)(time(NULL) - time1 + 1), 11TOaDigits3, 10));
12,668 }
12,669 //endif
12,670 printf( "\n");
12,671 //exit(0);
12,672
12,673 } // B_tree_Non_Unique_Only]
```

```

12,671
12,672
12,673 void B_tree(int argc, char *argv[], char* SourceBlock, uint64_t SourceSize) { // b_tree[
12,674     //int BSTorBtree = 0;
12,675     FILE *fp_in, *fp_out, *fp_outLOG, *fp_inLINE;
12,676     int Thunderwith;
12,677 // cleand unused below...
12,678     int nlines;
12,679
12,680     int LetterOffset;
12,681     unsigned long long FilesLEN;
12,682     unsigned long long WORDcount;
12,683     unsigned long long WORDcountBOTTOM;
12,684     unsigned long long WORDcountAttemptsToPut;
12,685     unsigned long long Total_fread=0, Total_fwrite=0;
12,686
12,687 // 15fixfixfixfix [
12,688     //unsigned long NumberOfFiles, WORDcountDistinct, WORDcountDistinctTOTAL = 0, TotalMemoryNeededForOnePass = 0; // This was in r.15fixfixfix
12,689     unsigned long long NumberOfFiles, WORDcountDistinct, WORDcountDistinctTOTAL = 0, TotalMemoryNeededForOnePass = 0; // This was in r.15fixfixfixfix
12,690 // 15fixfixfixfix ]
12,691     unsigned long long NumberOfLines; // rev. 12+
12,692     unsigned long WHOLEletter_BufferSize;
12,693     unsigned long long WHOLEletter_BufferSize_L14;
12,694     unsigned long memory_size, LetterBuffer, j, k, LINE10len, wrdlen;
12,695     unsigned long k_FIX;
12,696     unsigned long long i; // rev. 12+
12,697     //unsigned long size_in, size_out, size_inLINE;
12,698     unsigned long size_in; // rev. 12+
12,699 #if defined(_WIN32_ENVIRONMENT_)
12,700     unsigned long long size_inLINESIXFOUR;
12,701 #else
12,702     size_t size_inLINESIXFOUR;
12,703 #endif /* defined(_WIN32_ENVIRONMENT_) */
12,704
12,705     const int NumberOfSLOTs = 4096*2; // Since r.12+ in rev.12 it was 4096
12,706     unsigned long StackPtr;
12,707     //unsigned long BSTstack [65536*3]; // BST in worst case could become a LL.
12,708     unsigned long long BSTstack [8192*3]; // BST in worst case could become a LL.
12,709     unsigned long NumberOfTrees=0, NumberOfHashCollisions=0;
12,710     unsigned long iBSTwithMAXpeak, jBSTwithMAXpeak;
12,711     unsigned int PEAKibBST;
12,712     unsigned long BSTsTotalLEAFs=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break' is ?!
12,713     unsigned long BSTwithMAXnode=0, BSTcurrentNode=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break' is ?!
12,714     unsigned long BSTcurrentNodeMAXqUANTITY=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where
12,715     'break' is ?!
12,716     unsigned long BSTwithMAXnodePEAK=1, BSTwithMAXnodeLEAF=1; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break'
12,717     is ?!
12,718     unsigned long BSTwithMAXpeak=0, BSTcurrentPeak=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break' is ?!
12,719     unsigned long BSTcurrentPeakMAX=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break' is
12,720     ?!
12,721     unsigned long BSTcurrentPeakMAXqUANTITY=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where
12,722     'break' is ?!
12,723     unsigned long BSTwithMAXpeakNODE=1, BSTwithMAXpeakLEAF=1; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break'
12,724     is ?!
12,725     unsigned long BSTwithMAXleaf=0, BSTcurrentLeaf=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break' is ?!
12,726     unsigned long BSTcurrentLeafMAXqUANTITY=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where
12,727     'break' is ?!
12,728     unsigned long BSTwithMAXleafNODE=1, BSTwithMAXleafPEAK=1; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break'

```

```

is ?!
12,723
12,724 char *pointerflush, *pointerflushUNALIGN, *BufStart, *Flushing;
12,725 char *pointerflush_64, *pointerflushUNALIGN_64; // r.14++
12,726 unsigned long PseudoLinkedPointer, PseudoLinkedPointerNEW, PseudoLinkedPointerROOT, PseudoLinkedPointerNEWold;
12,727 unsigned long PseudoLinkedPointerNEWleft, PseudoLinkedPointerNEWright;
12,728 unsigned long PseudoLinkedPointerNEWmiddle;
12,729 char *bufend[ 806 ]; // 'a'=0, ... 'z'=25 - 26 letters x 31 lengths
12,730 long bufNumberOfWords[ 806 ]; // 'a'=0, ... 'z'=25 - 26 letters x 31 lengths
12,731 // long bufNoWpS[ 806 ][ 8192 ]; // ?! crashes below when an attempt to use it occur
12,732 char wrd[LongestLineInclusive+1+8]; // 0..30, 31 = 0
12,733 char wrdUP[LongestLineInclusive+1+8]; // 0..30, 31 = 0
12,734 char wrdUPold[LongestLineInclusive+1+8]; // 0..30, 31 = 0
12,735 char LINE10[257]; // 000..255, 256 = 0
12,736 char ZEROS[4]; // 0..3, 0 = 0, 1 = 0, 2 = 0, 3 = 0
12,737 char CRdLFa[2]; // 0..1, 0 = 13, 1 = 10
12,738 unsigned char workbyte; // unsigned in order to index ASCII
12,739 char workK[1024*128];
12,740 long workKoffset = -1;
12,741 unsigned long long FoundInLinkedList, Slot; //r.18
12,742 unsigned long OffsetsInBuffer[31]; // 00..30
12,743 unsigned long MAXusedBuffer[32]; // 00 not used, only 01..31
12,744 unsigned long GRMELhill[32]; // 00..31
12,745 unsigned long GRMELFoolAgain[32]; // 00..31
12,746 int Melnitchka;
12,747 unsigned long MAXusedBufferABS = 0;
12,748 unsigned long Utiliza1 = 0;
12,749 unsigned long Utiliza2 = 0;
12,750 unsigned long TotalWLchars = 0;
12,751
12,752 // GCC 7.3.0 from MINGW complains, so commented them all:
12,753 /* minimum signed 64 bit value */
12,754 // #define _I64_MIN (-9223372036854775807i64 - 1)
12,755 /* maximum signed 64 bit value */
12,756 // #define _I64_MAX 9223372036854775807i64
12,757 /* maximum unsigned 64 bit value */
12,758 // #define _UI64_MAX 0xffffffffffffffffui64
12,759
12,760 /* minimum signed 128 bit value */
12,761 #define _I128_MIN (-170141183460469231731687303715884105727i128 - 1)
12,762 /* maximum signed 128 bit value */
12,763 #define _I128_MAX 170141183460469231731687303715884105727i128
12,764 /* maximum unsigned 128 bit value */
12,765 #define _UI128_MAX 0xffffffffffffffffffffffffffffffffui128
12,766
12,767 char l1T0aDigits[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
12,768 // below duplicates are needed because of one_line_invoking need different buffers.
12,769 char l1T0aDigits2[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
12,770 char l1T0aDigits3[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
12,771 char l1T0aDigits4[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
12,772 unsigned long HEADOffsetFromStartBUKVA = 0;
12,773 unsigned long TAILOffsetFromStartBUKVA = 0;
12,774
12,775 int SplitOccured;
12,776 int POffsetInLEAF;
12,777 char *Auberge[4] = {"!\0", "/\0", "-\0", "\\0"};
12,778 int hashAlfalfa, iAlfalfa;
12,779 int PLE_words=0; // Quadruple!

```

```

12,780 char wrd1st[LongestLineInclusive+1+4]; // 0..30, 31 = 0
12,781 char wrd2nd[LongestLineInclusive+1+4]; // 0..30, 31 = 0
12,782 char wrd3rd[LongestLineInclusive+1+4]; // 0..30, 31 = 0
12,783 char wrd4th[LongestLineInclusive+1+4]; // 0..30, 31 = 0
12,784 char wrd5th[LongestLineInclusive+1+4]; // 0..30, 31 = 0
12,785 char wrd6th[LongestLineInclusive+1+4]; // 0..30, 31 = 0
12,786 char wrd7th[LongestLineInclusive+1+4]; // 0..30, 31 = 0
12,787 char wrd8th[LongestLineInclusive+1+4]; // 0..30, 31 = 0
12,788 char wrd9th[LongestLineInclusive+1+4]; // 0..30, 31 = 0
12,789 char wrd10th[LongestLineInclusive+1+4]; // 0..30, 31 = 0
12,790 char *DelimiterUnderscore = "_\0";
12,791 int PLE_words_INITflag = 0;
12,792
12,793 // QuickSortExternal_4+GB [
12,794 unsigned long long ThunderwithL64_L14;
12,795 unsigned long long Strnglen64_L14;
12,796 unsigned long long size_in64_L14, size_in2_L14;
12,797 unsigned long long Over4billionLines, j_Over4billion;
12,798 char OneChar_ieByte = '\0';
12,799 char CR_ieByte = '\r';
12,800 char SomeByte;
12,801 unsigned long long BufEnd_64;
12,802 unsigned long long SeekPosition;
12,803 unsigned long long *PointerToSeekPosition;
12,804 char FourGram[LongestLineInclusive+2]; // 31 longest 4-gram + CR + LF
12,805 char *PoolPhysical;
12,806 unsigned long long fsetpos_ZERO=0;
12,807 char OneCkusterZEROES[1024*4]; // Caution: must be ZEROed(NULLified)!
12,808 char *FileSwapTag = "LEPRECHAUNISH";
12,809 char EOFcode = 0x1A;
12,810 unsigned long long PseudoLinkedPointer_64, PseudoLinkedPointerNEW_64, PseudoLinkedPointerROOT_64, PseudoLinkedPointerNEWold_64;
12,811     unsigned long long PseudoLinkedPointerNEWleft_64, PseudoLinkedPointerNEWright_64;
12,812     unsigned long long PseudoLinkedPointerNEWmiddle_64;
12,813     unsigned long long NULLs_64 = 0;
12,814 unsigned long long PseudoLinkedPointerAUX_64;
12,815 unsigned long long PseudoLinkedPointerAUXdumbo_64;
12,816     char wrdAUX[LongestLineInclusive+1+8]; // 0..30, 31 = 0
12,817 // QuickSortExternal_4+GB ]
12,818
12,819 unsigned long CounterOccurrences;
12,820 unsigned long long NumberOfLEAFs=0;
12,821 unsigned long LevelsInCorona_Not_Counting_ROOT=0;
12,822 char *ngram[11] =
{"NULLleton\0", "singleton\0", "doubleton\0", "tripleton\0", "quadruplet\0", "quintuplet\0", "sextuplet\0", "septuplet\0", "octuplet\0", "nonuplet\0", "decuplet\0"};
12,823
12,824 unsigned long RipPasses;
12,825 unsigned long long NULLsForWRD=0;
12,826
12,827 int NewOrder;
12,828     char LINE10_NO_DUMP[257]; // 000..255, 256 = 0
12,829
12,830 char *TwoDigitHEXlist[256] = {
12,831     "00\0",
12,832     "01\0",
12,833     "02\0",
12,834     "03\0",
12,835     "04\0",
12,836     "05\0",

```

12,837 "06\0",
12,838 "07\0",
12,839 "08\0",
12,840 "09\0",
12,841 "0A\0",
12,842 "0B\0",
12,843 "0C\0",
12,844 "0D\0",
12,845 "0E\0",
12,846 "0F\0",
12,847 "10\0",
12,848 "11\0",
12,849 "12\0",
12,850 "13\0",
12,851 "14\0",
12,852 "15\0",
12,853 "16\0",
12,854 "17\0",
12,855 "18\0",
12,856 "19\0",
12,857 "1A\0",
12,858 "1B\0",
12,859 "1C\0",
12,860 "1D\0",
12,861 "1E\0",
12,862 "1F\0",
12,863 "20\0",
12,864 "21\0",
12,865 "22\0",
12,866 "23\0",
12,867 "24\0",
12,868 "25\0",
12,869 "26\0",
12,870 "27\0",
12,871 "28\0",
12,872 "29\0",
12,873 "2A\0",
12,874 "2B\0",
12,875 "2C\0",
12,876 "2D\0",
12,877 "2E\0",
12,878 "2F\0",
12,879 "30\0",
12,880 "31\0",
12,881 "32\0",
12,882 "33\0",
12,883 "34\0",
12,884 "35\0",
12,885 "36\0",
12,886 "37\0",
12,887 "38\0",
12,888 "39\0",
12,889 "3A\0",
12,890 "3B\0",
12,891 "3C\0",
12,892 "3D\0",
12,893 "3E\0",
12,894 "3F\0",

12,895 "40\0",
12,896 "41\0",
12,897 "42\0",
12,898 "43\0",
12,899 "44\0",
12,900 "45\0",
12,901 "46\0",
12,902 "47\0",
12,903 "48\0",
12,904 "49\0",
12,905 "4A\0",
12,906 "4B\0",
12,907 "4C\0",
12,908 "4D\0",
12,909 "4E\0",
12,910 "4F\0",
12,911 "50\0",
12,912 "51\0",
12,913 "52\0",
12,914 "53\0",
12,915 "54\0",
12,916 "55\0",
12,917 "56\0",
12,918 "57\0",
12,919 "58\0",
12,920 "59\0",
12,921 "5A\0",
12,922 "5B\0",
12,923 "5C\0",
12,924 "5D\0",
12,925 "5E\0",
12,926 "5F\0",
12,927 "60\0",
12,928 "61\0",
12,929 "62\0",
12,930 "63\0",
12,931 "64\0",
12,932 "65\0",
12,933 "66\0",
12,934 "67\0",
12,935 "68\0",
12,936 "69\0",
12,937 "6A\0",
12,938 "6B\0",
12,939 "6C\0",
12,940 "6D\0",
12,941 "6E\0",
12,942 "6F\0",
12,943 "70\0",
12,944 "71\0",
12,945 "72\0",
12,946 "73\0",
12,947 "74\0",
12,948 "75\0",
12,949 "76\0",
12,950 "77\0",
12,951 "78\0",
12,952 "79\0",

12,953 "7A\0",
12,954 "7B\0",
12,955 "7C\0",
12,956 "7D\0",
12,957 "7E\0",
12,958 "7F\0",
12,959 "80\0",
12,960 "81\0",
12,961 "82\0",
12,962 "83\0",
12,963 "84\0",
12,964 "85\0",
12,965 "86\0",
12,966 "87\0",
12,967 "88\0",
12,968 "89\0",
12,969 "8A\0",
12,970 "8B\0",
12,971 "8C\0",
12,972 "8D\0",
12,973 "8E\0",
12,974 "8F\0",
12,975 "90\0",
12,976 "91\0",
12,977 "92\0",
12,978 "93\0",
12,979 "94\0",
12,980 "95\0",
12,981 "96\0",
12,982 "97\0",
12,983 "98\0",
12,984 "99\0",
12,985 "9A\0",
12,986 "9B\0",
12,987 "9C\0",
12,988 "9D\0",
12,989 "9E\0",
12,990 "9F\0",
12,991 "A0\0",
12,992 "A1\0",
12,993 "A2\0",
12,994 "A3\0",
12,995 "A4\0",
12,996 "A5\0",
12,997 "A6\0",
12,998 "A7\0",
12,999 "A8\0",
13,000 "A9\0",
13,001 "AA\0",
13,002 "AB\0",
13,003 "AC\0",
13,004 "AD\0",
13,005 "AE\0",
13,006 "AF\0",
13,007 "B0\0",
13,008 "B1\0",
13,009 "B2\0",
13,010 "B3\0",

13,011 "B4\0",
13,012 "B5\0",
13,013 "B6\0",
13,014 "B7\0",
13,015 "B8\0",
13,016 "B9\0",
13,017 "BA\0",
13,018 "BB\0",
13,019 "BC\0",
13,020 "BD\0",
13,021 "BE\0",
13,022 "BF\0",
13,023 "C0\0",
13,024 "C1\0",
13,025 "C2\0",
13,026 "C3\0",
13,027 "C4\0",
13,028 "C5\0",
13,029 "C6\0",
13,030 "C7\0",
13,031 "C8\0",
13,032 "C9\0",
13,033 "CA\0",
13,034 "CB\0",
13,035 "CC\0",
13,036 "CD\0",
13,037 "CE\0",
13,038 "CF\0",
13,039 "D0\0",
13,040 "D1\0",
13,041 "D2\0",
13,042 "D3\0",
13,043 "D4\0",
13,044 "D5\0",
13,045 "D6\0",
13,046 "D7\0",
13,047 "D8\0",
13,048 "D9\0",
13,049 "DA\0",
13,050 "DB\0",
13,051 "DC\0",
13,052 "DD\0",
13,053 "DE\0",
13,054 "DF\0",
13,055 "E0\0",
13,056 "E1\0",
13,057 "E2\0",
13,058 "E3\0",
13,059 "E4\0",
13,060 "E5\0",
13,061 "E6\0",
13,062 "E7\0",
13,063 "E8\0",
13,064 "E9\0",
13,065 "EA\0",
13,066 "EB\0",
13,067 "EC\0",
13,068 "ED\0",

```
13,069 "EE\0",
13,070 "EF\0",
13,071 "F0\0",
13,072 "F1\0",
13,073 "F2\0",
13,074 "F3\0",
13,075 "F4\0",
13,076 "F5\0",
13,077 "F6\0",
13,078 "F7\0",
13,079 "F8\0",
13,080 "F9\0",
13,081 "FA\0",
13,082 "FB\0",
13,083 "FC\0",
13,084 "FD\0",
13,085 "FE\0",
13,086 "FF\0"
13,087 };
13,088
13,089 /*
13,090 #if defined(InternalRAM)
13,091 BStorBtree = 3; //Internal
13,092 #endif
13,093 #if defined(ExternalRAM)
13,094 BStorBtree = 2; //External
13,095 #endif
13,096 */
13,097
13,098 uint64_t PointerToNotLoadedYet;
13,099 uint64_t jj,jjj, kk, BuildingBlocksSTRIDE;
13,100 int mm;
13,101 //define MatchLensNUM 8
13,102 //int MatchLens[MatchLensNUM]={4,6,8,10,12,14,16,18};
13,103 uint64_t GettingIndexOfArray;
13,104 uint64_t NotFoundKeys=0, FoundKeys=0;
13,105 unsigned long long WORDcountBOTTOMPerMatchLen;
13,106 int strFLAG;
13,107 int KeySize;
13,108 uint64_t AllSlots;
13,109 unsigned long long BUGGYoffset; // fix for the bug in r.17 and prior, r.18
13,110
13,111 Thunderwith=RAMpoolInKB_GLOBAL;
13,112 printf ("Leprechaun: Memory pool for B-trees is %s MB.\n", _ui64toaKAZEcomma(Thunderwith, 11TOaDigits2, 10) );
13,113
13,114 if (BStorBtree == 3)
13,115 {
13,116 if (HashInBITS_GLOBAL+3<10)
13,117 printf ("Leprechaun: In this revision %sbytes %d-way hash is used which results in %d x %s internal B-Trees of order 3.\n",
13,118 _ui64toaKAZEcomma((((1LL)<<HashInBITS_GLOBAL)<<3), 11TOaDigits, 10), MatchLensNUM, MatchLensNUM, _ui64toaKAZEcomma(((1LL)<<HashInBITS_GLOBAL) ), 11TOaDigits2, 10) );
13,119 else if (HashInBITS_GLOBAL+3>=10 && HashInBITS_GLOBAL+3<20)
13,120 printf ("Leprechaun: In this revision %sKB %d-way hash is used which results in %d x %s internal B-Trees of order 3.\n", _ui64toaKAZEcomma(
13,121 (((1LL)<<HashInBITS_GLOBAL)<<3))>10, 11TOaDigits, 10), MatchLensNUM, MatchLensNUM, _ui64toaKAZEcomma(((1LL)<<HashInBITS_GLOBAL) ), 11TOaDigits2, 10) );
13,122 else
13,123 printf ("Leprechaun: In this revision %sMB %d-way hash is used which results in %d x %s internal B-Trees of order 3.\n", _ui64toaKAZEcomma(
13,124 (((1LL)<<HashInBITS_GLOBAL)<<3))>20, 11TOaDigits, 10), MatchLensNUM, MatchLensNUM, _ui64toaKAZEcomma(((1LL)<<HashInBITS_GLOBAL) ), 11TOaDigits2, 10) );
13,125 } else if (BStorBtree == 2){
13,126 if (HashInBITS_GLOBAL+3<10)
```

```

13,124 printf ("Leprechaun: In this revision %sbytes %d-way hash is used which results in %d x %s external B-Trees of order 3.\n",
_ui64toaKAZEcomma((((1LL)<<HashInBITS_GLOBAL)<<3), 1lToaDigits, 10), MatchLensNUM, MatchLensNUM, _ui64toaKAZEcomma(((1LL)<<HashInBITS_GLOBAL)), 1lToaDigits2, 10) );
13,125 else if (HashInBITS_GLOBAL+3>=10 && HashInBITS_GLOBAL+3<20)
13,126 printf ("Leprechaun: In this revision %sKB %d-way hash is used which results in %d x %s external B-Trees of order 3.\n", _ui64toaKAZEcomma(
((((1LL)<<HashInBITS_GLOBAL)<<3))>>10, 1lToaDigits, 10), MatchLensNUM, MatchLensNUM, _ui64toaKAZEcomma(((1LL)<<HashInBITS_GLOBAL)), 1lToaDigits2, 10) );
13,127 else
13,128 printf ("Leprechaun: In this revision %sMB %d-way hash is used which results in %d x %s external B-Trees of order 3.\n", _ui64toaKAZEcomma(
((((1LL)<<HashInBITS_GLOBAL)<<3))>>20, 1lToaDigits, 10), MatchLensNUM, MatchLensNUM, _ui64toaKAZEcomma(((1LL)<<HashInBITS_GLOBAL)), 1lToaDigits2, 10) );
13,129 }
13,130
13,131 if (HashInBITS_GLOBAL-HashChunkSizeInBITS_GLOBAL==0)
13,132 printf ("Leprechaun: In this revision, %s pass is to be executed.\n", _ui64toaKAZEcomma(1<<<HashInBITS_GLOBAL-HashChunkSizeInBITS_GLOBAL), 1lToaDigits, 10));
13,133 else
13,134 printf ("Leprechaun: In this revision, %s passes are to be executed.\n", _ui64toaKAZEcomma(1<<<HashInBITS_GLOBAL-HashChunkSizeInBITS_GLOBAL), 1lToaDigits, 10));
13,135
13,136 // 16fixfix [
13,137 PLE_words_INITflag = 0;
13,138 PLE_words = 0;
13,139 // 16fixfix ]
13,140     Melnitchka = 0;
13,141     WORDcount = 0; // Total word count i.e. for all files!
13,142     WORDcountDistinct = 0;
13,143     NumberOfFiles = 0;
13,144     NumberOfLines = 0;
13,145     FilesLEN = 0;
13,146     LINE10len = 0;
13,147 // Added in r.14+++++FIXFIX [
13,148     NumberOfTrees=0; NumberOfHashCollisions=0;
13,149     NumberOfLEAFs=0;
13,150     WORDcountAttemptsToPut=0;
13,151     LevelsInCorona_Not_Counting_ROOT=0;
13,152 // Added in r.14+++++FIXFIX ]
13,153
13,154 if( ( fp_outLOG = fopen( "Leprechaun.LOG", "a+" ) ) == NULL )
13,155 { printf( "Leprechaun: Can't open file Leprechaun.LOG.\n" ); exit( 7 ); }
13,156
13,157 //     printf( "Leprechaun: Allocating HASH memory %s bytes ... ", _ui64toaKAZEcomma( ((1<<<HashInBITS)*8) + 1 + 64 , 1lToaDigits, 10) );
13,158 //     pointerflushUNALIGN = (char *)malloc( (1<<<HashInBITS)*8 + 1 + 64 );
13,159
13,160 printf( "Leprechaun: Allocating HASH memory %s bytes ... ", _ui64toaKAZEcomma( (uint64_t)MatchLensNUM*((uint64_t)( ((1LL)<<HashInBITS_GLOBAL) ) *8) + 1 + 64 ,
1lToaDigits, 10) );
13,161 pointerflushUNALIGN = (char *)malloc( (uint64_t)(MatchLensNUM)*(uint64_t)( ((1LL)<<HashInBITS_GLOBAL) ) *8 + 1 + 64 );
13,162
13,163 if( pointerflushUNALIGN == NULL )
13,164 { puts( "\nLeprechaun: Needed memory allocation denied!\n" ); exit( 7 ); }
13,165 pointerflush = pointerflushUNALIGN + 64 - (((size_t)pointerflushUNALIGN) % 64); // 13_6+
13,166 //offset=64-int((long)data&63);
13,167 printf( "OK\n" );
13,168
13,169 GLOBAL_HASHPOT = pointerflush; //btree matchfinder needed
13,170
13,171 //     memset(pointerflush,0,17210368*8);
13,172 memset(pointerflush,0,(uint64_t)(MatchLensNUM)*(uint64_t)( ((1LL)<<HashInBITS_GLOBAL) ) *8);
13,173     if (BSTorBtree == 2) {
13,174     if( ( fp_outRG = fopen( "Leprechaun_64bit.swp", "wb+" ) ) == NULL )
13,175     { printf( "Leprechaun: Can't create file 'Leprechaun_64bit.swp'.\n" ); exit( 7 ); }
13,176 // Tag for the swap file is: LEPRECHAUNISH(ASCIIcode26)
13,177 // or 14bytes, then when type of the swap is requested:

```

```

13,178 // D:\KAZE_~1\LEPREC~1>type Leprechaun_64bit.swp
13,179 // LEPRECHAUNISH
13,180 // D:\KAZE_~1\LEPREC~1>
13,181 size_in64_L14 = 1024LL * 1024 * (unsigned long long)Thunderwith + 14;
13,182 BufEnd_64 = 0+14;
13,183 // The tag plays two roles, the second to avoid existence of SeekPosition equal to 0. The 0 cannot be used as a free slot FLAG without the TAG.
13,184 printf( "Leprechaun: Allocating/ZEROing %s bytes swap file ... ", _ui64toaKAZEcomma(size_in64_L14, llTOaDigits, 10) );
13,185 fsetpos(fp_outRG, (const fpos_t *)&fsetpos_ZERO); // SOMETHING ROTTEN with lseeki64/fseeko and fsetpos ???! So DO-IT-OVER.
13,186 memset(OneCkusterZEROES,0,1024*4);
13,187 for (ThunderwithL64_L14=0; ThunderwithL64_L14 < size_in64_L14/(1024*4); ThunderwithL64_L14++)
13,188     fwrite(OneCkusterZEROES, 1024*4, 1, fp_outRG);
13,189 for (ThunderwithL64_L14=0; ThunderwithL64_L14 < size_in64_L14%(1024*4); ThunderwithL64_L14++)
13,190     fwrite(&OneChar_ieByte, 1, 1, fp_outRG);
13,191 fsetpos(fp_outRG, (const fpos_t *)&fsetpos_ZERO); // SOMETHING ROTTEN with lseeki64/fseeko and fsetpos ???! So DO-IT-OVER.
13,192     fwrite(FileSwapTag, 13, 1, fp_outRG);
13,193     fwrite(&EOFcode, 1, 1, fp_outRG);
13,194 fsetpos(fp_outRG, (const fpos_t *)&BufEnd_64); // SOMETHING ROTTEN with lseeki64/fseeko and fsetpos ???! So DO-IT-OVER.
13,195     } else { // ##### 64bit memory manipulations [
13,196     size_in64_L14 = 1024LL * 1024 * (unsigned long long)Thunderwith + 14 + 1 + 64;
13,197 printf( "Leprechaun: Allocating memory for B-trees %lu MB ... ", (size_in64_L14>>20)+1 );
13,198 pointerflushUNALIGN_64 = (char *)malloc( size_in64_L14 );
13,199 memset(pointerflushUNALIGN_64,0,size_in64_L14);
13,200 if( pointerflushUNALIGN_64 == NULL )
13,201 { puts( "\nLeprechaun: Needed memory allocation denied!\n" ); exit( 7 ); }
13,202 pointerflush_64 = pointerflushUNALIGN_64 + 64 - (((size_t)pointerflushUNALIGN_64) % 64); // 13_6+
13,203 //offset=64-int((long)data&63);
13,204 //memset(pointerflush_64,0,1024 * (unsigned long long)Thunderwith + 14);
13,205 BufEnd_64 = (unsigned long long)pointerflush_64;
13,206 /*
13,207 printf( "BufEnd_64: %s\n", _ui64toaKAZEcomma(BufEnd_64, llTOaDigits, 10) );
13,208 printf( "pointerflush_64: %s\n", _ui64toaKAZEcomma(pointerflush_64, llTOaDigits, 10) );
13,209 pointerflush_64 = (char *)BufEnd_64;
13,210 printf( "pointerflush_64: %s\n", _ui64toaKAZEcomma(pointerflush_64, llTOaDigits, 10) );
13,211 exit( 1);
13,212 //BufEnd_64: 541,261,888
13,213 //pointerflush_64: 541,261,888
13,214 //pointerflush_64: 541,261,888
13,215 */
13,216     } // ##### 64bit memory manipulations ]
13,217 printf( "OK\n" );
13,218 fprintf( fp_outLOG, "Leprechaun report:\n" );
13,219
13,220 // Comment the streamed-read of input file ... [[[
13,221 /*
13,222 if( ( fp_inLINE = fopen( argv[1], "rb" ) ) == NULL )
13,223 { printf( "Leprechaun: Can't open file %s\n", argv[1] ); exit( 7 ); }
13,224
13,225 //fseek( fp_inLINE, 0L, SEEK_END ); //Rev. 12
13,226 //size_inLINE = ftell( fp_inLINE ); //Rev. 12
13,227 //fseek( fp_inLINE, 0L, SEEK_SET ); //Rev. 12
13,228
13,229 #if defined(_WIN32_ENVIRONMENT_)
13,230     // 64bit:
13,231 _lseeki64( fileno(fp_inLINE), 0L, SEEK_END );
13,232 size_inLINESIXFOUR = _telli64( fileno(fp_inLINE) );
13,233 _lseeki64( fileno(fp_inLINE), 0L, SEEK_SET );
13,234 #else
13,235     // 64bit:

```

```

13,236 fseeko( fp_inLINE, 0L, SEEK_END );
13,237 size_inLINESIXFOUR = ftello( fp_inLINE );
13,238 fseeko( fp_inLINE, 0L, SEEK_SET );
13,239 #endif // defined( WIN32_ENVIRONMENT_ )
13,240
13,241 printf( "Size of input file: %s\n", _ui64toaKAZEcomma(size_inLINESIXFOUR, llToaDigits, 10) );
13,242
13,243 // ~~~~~
13,244 wrdlen = 0;
13,245 for( i = 0; i < size_inLINESIXFOUR; i++ )
13,246 {
13,247     // ~~~~~ Buffering fread [
13,248     if (workKoffset == -1) {
13,249         if (i + 1024*128 < size_inLINESIXFOUR) {
13,250             fread( &workK[0], 1, 1024*128, fp_inLINE );
13,251             workKoffset = 0;
13,252             workbyte = workK[workKoffset];
13,253         } else {
13,254             fread( &workbyte, 1, 1, fp_inLINE );
13,255             //printf("%d, '%d' %s\n", i, workbyte, TwoDigitHEXlist[workbyte]); //So stupid code of mine, the remaining (mod 128*1024) has to be read byte by byte as
13,256             NULL, grmb1!
13,257         }
13,258     } else {
13,259         workKoffset++;
13,260         workbyte = workK[workKoffset];
13,261         if (workKoffset == 1024*128 - 1) workKoffset = -1;
13,262     }
13,263     // ~~~~~ Buffering fread ]
13,264
13,265     if( isalpha( workbyte ) )
13,266     {
13,267         if( wrdlen < 31 )
13,268         { wrd[ wrdlen ] = tolower( workbyte ); }
13,269         wrdlen++;
13,270     }
13,271     memcpy( &wrd[ wrdlen ], TwoDigitHEXlist[workbyte], 2 );
13,272     wrdlen++;
13,273     wrdlen++;
13,274 } // i 'for'
13,275 // ~~~~~
13,276 */
13,277 // Comment the streamed-read of input file ... ]]]
13,278 size_inLINESIXFOUR=SourceSize;
13,279 printf( "Leprechaun: Size of input file: %s\n", _ui64toaKAZEcomma(size_inLINESIXFOUR, llToaDigits, 10) );
13,280 printf( "\n");
13,281
13,282     time1=time(NULL); //fix of bigtime
13,283
13,284 for (jj=0; jj< MatchLensNUM; jj++) {
13,285     // BUG IN R.17 and prior - the next 'if' is only for RAM (y/Y) not for SSD (z/Z):
13,286     if (BSTorBtree == 2) { //r.18
13,287         BUGGYoffset = (BufEnd_64-(0+14));
13,288     } else { // ##### 64bit memory manipulations [
13,289         BUGGYoffset = (BufEnd_64-(unsigned long long)pointerflush_64);
13,290     } // ##### 64bit memory manipulations ]
13,291
13,292

```

```

13,293 //for( RipPasses = 1-1; RipPasses <= (1<<<(HashInBITS-HashChunkSizeInBITS))-1; RipPasses++ )
13,294 //{
13,295 RipPasses = 1-1;
13,296 WhyTheHellForIsNotWorking:
13,297
13,298 if (RipPasses < (1<<<(HashInBITS_GLOBAL-HashChunkSizeInBITS_GLOBAL))-1)
13,299 printf( "Leprechaun: Inserting keys/BBs of order %s into B-trees, free RAM in B-tree pool is %s MB; Pass #s of %lu ... \r", _ui64toaKAZEzerocomma(MatchLens[jj],
11TOaDigits2, 10)+(26-3), _ui64toaKAZEzerocomma((size_in64_L14 - BUGGYoffset)>>20, 11TOaDigits, 10)+(26-10), _ui64toaKAZEzerocomma(RipPasses+1, 11TOaDigits3, 10)+(26-3),
(1<<<(HashInBITS_GLOBAL-HashChunkSizeInBITS_GLOBAL)) );
13,300 else
13,301 printf( "Leprechaun: Inserting keys/BBs of order %s into B-trees, free RAM in B-tree pool is %s MB; Pass #s of %lu ... ", _ui64toaKAZEzerocomma(MatchLens[jj],
11TOaDigits2, 10)+(26-3), _ui64toaKAZEzerocomma((size_in64_L14 - BUGGYoffset)>>20, 11TOaDigits, 10)+(26-10), _ui64toaKAZEzerocomma(RipPasses+1, 11TOaDigits3, 10)+(26-3),
(1<<<(HashInBITS_GLOBAL-HashChunkSizeInBITS_GLOBAL)) );
13,302 for (BuildingBlocksSTRIDE=0; BuildingBlocksSTRIDE < size_inLINESIXFOUR-MatchLens[jj]+1; BuildingBlocksSTRIDE++) {
13,303 // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed [
13,304 // wrdlen=0;
13,305 // memset( &wrd[0], 0, (LongestLineInclusive+1+8) ); //r.18, see below, the old nullifier is commented.
13,306 // for (mm=0; mm< MatchLens[jj]; mm++) {
13,307 // memcpy( &wrd[ wrdlen ], TwoDigitHEXlist[(unsigned char *) (SourceBlock+mm+BuildingBlocksSTRIDE)], 2 );
13,308 // wrdlen++;
13,309 // wrdlen++;
13,310 // }
13,311 // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed ]
13,312
13,313 wrdlen=MatchLens[jj];
13,314 memset( &wrd[0], 0, (LongestLineInclusive+1+8) ); //r.18, see below, the old nullifier is commented.
13,315
13,316 // if (wrdlen > 28) {
13,317 // FIPS202_SHA3_224((unsigned char *) (SourceBlock+BuildingBlocksSTRIDE), wrdlen, (unsigned char *)SHA328bytes);
13,318 // wrdlen=28;
13,319 // }
13,320
13,321 memcpy( &wrd[ 0 ], &wrdlen, 1 ); // The keysize is 1..255, pretty enough since my needs are 1..28 (up to SHA3-224).
13,322 // if (wrdlen < 28)
13,323 memcpy( &wrd[ 0+1 ], (unsigned char *) (SourceBlock+BuildingBlocksSTRIDE), wrdlen );
13,324 // else
13,325 // memcpy( &wrd[ 0+1 ], (unsigned char *)SHA328bytes, wrdlen );
13,326
13,327 BufStart = pointerflush;
13,328 // Slot = ((wrd[0]-'_')*28*28*28*28 + (wrd[1]-'_')*28*28*28 + (wrd[2]-'_')*28*28 + (wrd[3]-'_')*28 + (wrd[4]-'_'))<<3;
13,329 // Slot = FNV1A_Hash_Jesteress_27bit(wrd, wrdlen)<<3; // Commented since r.14+++ because of passes.
13,330 //Slot = FNV1A_Hash_Jesteress_27bit(wrd +1, wrdlen); WORDcount++;
13,331 Slot = FNV1A_Pippip_Yurii(wrd +1, wrdlen); WORDcount++; // Changed 2019-Nov-28
13,332
13,333 // Bug fix for all r.14+++ and below! [
13,334 //memcpy( &wrd[(LongestLineInclusive+1+4)-4], &NULLsForWRD, 4 );
13,335 //memcpy( &wrd[(LongestLineInclusive+1+4)-4-1], &NULLsForWRD, 1 );
13,336 // Bug fix for all r.14+++ and below! ]
13,337
13,338 // Example: HashInBITS-HashChunkSizeInBITS=2
13,339 // HashInBITS = 5
13,340 // HashChunkSizeInBITS = 3
13,341 // RipPasses = 1<<<(HashInBITS-HashChunkSizeInBITS) i.e. 1<<2 which is 4 i.e. 32 slots with 4 passes 8 slots each.
13,342 // 00??? 5bits 0-7
13,343 // 01??? 5bits 8-15
13,344 // 10??? 5bits 16-23
13,345 // 11??? 5bits 24-31
13,346 if ( (Slot>>HashChunkSizeInBITS_GLOBAL) == RipPasses ) {

```

```

13,347 Slot = Slot<<3;
13,348 // NEW NEW NEW [ //r.18
13,349 // In here Slot is not within a single pool but in MatchLensNUM sub-pools i.e. MatchLensNUM-way i.e. MatchLensNUM hashpots:
13,350 /*
13,351 for (GettingIndexOfArray=0; GettingIndexOfArray<MatchLensNUM; GettingIndexOfArray++) {
13,352     // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed [
13,353     //     if ( (wrklen>>1)==MatchLens[GettingIndexOfArray] ) break;
13,354     if ( (wrklen)==MatchLens[GettingIndexOfArray] ) break;
13,355     // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed ]
13,356 }
13,357 */
13,358 GettingIndexOfArray=jj; // same as above atrocity
13,359 Slot = Slot + (GettingIndexOfArray*(1LL<<HashInBITS_GLOBAL)*8); // CAUTION: 'wrklen' is halved here, when a new revision comes with 1:1 keysize then change it.
13,360 // NEW NEW NEW ] //r.18
13,361
13,362 //KeySize = LongestLineInclusive; // 'LongestLineInclusive' is for 1-way hashing where different KeySizes can occupy same HASH pool.
13,363 KeySize = wrklen; //r.18, In case multi-way hashing: either the MAX allowed or the 'wrklen'.
13,364
13,365 //Slot = 0; // One Tree only!
13,366 memcpy( &PseudoLinkedPointer_64, BufStart+Slot, 8 );
13,367
13,368 // ##### B-tree order 3 fragment 64bit [
13,369 //
13,370 // LEAF structure: [LeftPointer][MiddlePointer][RightPointer][LeftWord][RightWord]
13,371 //                4bytes      4bytes      4bytes      wrklen  wrklen
13,372 //                *          *                      <- if *(char *)==0 means the word cell is empty
13,373 // ALL B-tree order 3 fragment consists of 3 sub-fragments:
13,374 // 1] Search 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search 3] Insert Iterative
13,375
13,376 // LEAF_64 structure: [LeftPointer][MiddlePointer][RightPointer][LeftWord] [RightWord]
13,377 //                   8bytes      8bytes      8bytes      LongestLineInclusive+1+4 LongestLineInclusive+1+4
13,378 //                   *          *                      <- if *(char *)==0 means the word cell is empty
13,379 // Note: In order to use one fread(and strcmp) a NULL postfix for LeftWord, RightWord i.e. LeftWord_Length=len(LeftWord)+1 a kinda stupid choice ...
13,380 // Note: BufEnd_64 in fact is the first free position after the BUFFER END!
13,381
13,382 // 1] Search [
13,383         if (PseudoLinkedPointer_64 == 0) // means EMPTY-SLOT
13,384         {
13,385             //if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrklen + 4 + 4 + 4 < GRMELFoolAgain((int)wrklen) ) // +4 more for BST instead of LL; + more(see
13,386             //
13,387             //     {
13,388             //         memcpy( BufStart+Slot, &bufend[LetterOffset], 4 );
13,389             //         bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
13,390             //         memcpy( bufend[LetterOffset], wrd, wrklen ); WORDcountDistinct++; bufNumberOfWords[LetterOffset]++;
13,391             //         bufend[LetterOffset] = bufend[LetterOffset] + 2*wrklen;
13,392             //         if (MAXusedBuffer[wrklen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrklen] = (unsigned long)(bufend[LetterOffset] -
13,393             // BUG IN R.17 and prior - the next 'if' is only for RAM (y/Y) not for SSD (z/Z):
13,394             // if (BSTorBtree == 2) { //r.18
13,395             BUGGYoffset = (BufEnd_64-(0+14));
13,396             } else { // ##### 64bit memory manipulations [
13,397             BUGGYoffset = (BufEnd_64-(unsigned long long)pointerflush_64);
13,398             } // ##### 64bit memory manipulations ]
13,399             if( 8 + 8 + 8 + 2*(KeySize+1+8) < size_in64_L14 - BUGGYoffset ) // the longest wrklen is LongestLineInclusive but actual is LongestLineInclusive(+ CR char)
13,400             {
13,401                 memcpy( BufStart+Slot, &BufEnd_64, 8 );
13,402                 BufEnd_64 = BufEnd_64 + 8 + 8 + 8;

```

```

13,403         if (BSTorBtree == 2) {
13,404             fsetpos(fp_outRG, (const fpos_t *)&BufEnd_64);
13,405             //fwrite(wrd, wrdlen, 1, fp_outRG); //GRMBL! r.18
13,406             fwrite(wrd, (KeySize+1+8), 1, fp_outRG); //GRMBL! r.18
13,407             WORDcountDistinct++; Total_fwrite++;
13,408             //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
13,409             // r.14+ The above line was commented because the pool is already ZEROed.
13,410             } else { // ##### 64bit memory manipulations [
13,411             //memcpy( (char *)BufEnd_64, wrd, wrdlen ); //GRMBL! r.18
13,412             memcpy( (char *)BufEnd_64, wrd, (KeySize+1+8) ); //GRMBL! r.18
13,413             WORDcountDistinct++;
13,414             } // ##### 64bit memory manipulations ]
13,415             BufEnd_64 = BufEnd_64 + 2*(KeySize+1+8);
13,416             //fsetpos(fp_outRG, &BufEnd_64);
13,417             }
13,418         else
13,419             { printf( "\nLeprechaun: Failure! Increment 'Memory for B-trees'\n" );
13,420 fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, llToaDigits, 10), _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct,
llToaDigits2, 10) );
13,421 fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14, llToaDigits, 10) );
13,422 fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, llToaDigits, 10) );
13,423 fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
13,424             fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for B-trees'\n\n");
13,425             exit( 7 );
13,426         }
13,427             FoundInLinkedList = 1+1;
13,428             }
13,429             else // means USED-SLOT
13,430             { FoundInLinkedList = 0;
13,431             StackPtr = 0;
13,432             // while (PseudoLinkedPointer != 0 && FoundInLinkedList == 0)
13,433             while (PseudoLinkedPointer_64 != 0 && FoundInLinkedList == 0)
13,434             {
13,435             // ***** 'P W P' section [
13,436             // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
13,437             // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
13,438             // here ALWAYS LW exists: no need for existence check - line below
13,439             // if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
13,440             // if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wrdlen) > 0) // go LP
13,441             // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,442             //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
13,443             //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,444             //fread(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
13,445             // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,446             // [ //r.14+
13,447             PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
13,448             if (BSTorBtree == 2) {
13,449             fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,450             fread(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fread++;
13,451             } else { // ##### 64bit memory manipulations [
13,452             memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64, 8+8+8+2*(KeySize+1+8) );
13,453             } // ##### 64bit memory manipulations ]
13,454             memcpy( &FourGramL[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
13,455             // ] //r.14+
13,456             //strFLAG=strcmpKAZE13(FourGramL, wrd);
13,457             strFLAG=memcmp(FourGramL+1, wrd+1, wrdlen); // Since 2019-Feb-07, it is not ASCIIZ string but zero byte is the length of the 1..
13,458             if (strFLAG > 0) // go LP
13,459             { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 0, 4 ); //LP

```



```

13,460 //          PseudoLinkedPointer = PseudoLinkedPointerNEW;
13,461 //      }
13,462 //      {
13,463 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,464 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 0; //LP
13,465 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,466 if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
13,467 BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
13,468 BSTstack[StackPtr] = 0; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
13,469 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,470 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,471 //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
13,472 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,473 // [ //r.14+
13,474 memcpy( &PseudoLinkedPointer_64, &LEAF[0], 8 );
13,475 // ] //r.14+
13,476 }
13,477 //      else if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wrdlen) < 0) // go RP or MP
13,478 //      else if (strFLAG < 0) // go RP or MP
13,479 //      { // RW existence check - line below:
13,480 //      if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 ) // RW exists
13,481 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,482 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
13,483 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,484 //fread(&SomeByte, 1, 1, fp_outRG);
13,485 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,486 // [ //r.14+
13,487 memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (KeySize+1+8)], 1 );
13,488 // ] //r.14+
13,489 if (SomeByte != 0 ) // RW exists
13,490 { // Here all 'P W P' section is repeated; the way of handling case when dynamic number of words in leaf
13,491 // *****
13,492 // ***** 'P W P' section 2 [
13,493 // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
13,494 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
13,495 // here ALWAYS RW exists: no need for existence check - line below
13,496 // if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
13,497 // if (memcmp(PseudoLinkedPointer+4+4+4+wrdlen, wrd, wrdlen) > 0) // go MP
13,498 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,499 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
13,500 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,501 //fread(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
13,502 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,503 // [ //r.14+
13,504 memcpy( &FourGramL[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
13,505 // ] //r.14+
13,506 //strFLAG=strcmpKAZE13(FourGramL, wrd);
13,507 strFLAG=memcmp(FourGramL+1, wrd+1, wrdlen); // Since 2019-Feb-07, it is not ASCIIZ string but zero byte is the length of the 1..
13,508 if (strFLAG > 0) // go MP
13,509 { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
13,510 //      PseudoLinkedPointer = PseudoLinkedPointerNEW;
13,511 //      }
13,512 //      {
13,513 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,514 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8; //MP
13,515 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,516 if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
13,517 BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf

```

```

13,518 BSTstack[StackPtr] = 8; ++StackPtr; //LPOffset=0;MPOffset=8;RPOffset=16;
13,519 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,520 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,521 //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
13,522 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,523 // [ //r.14+
13,524 memcpy( &PseudoLinkedPointer_64, &LEAF[8], 8 );
13,525 // ] //r.14+
13,526 }
13,527 // else if (memcmp(PseudoLinkedPointer+4+4+4*wordlen, wrd, wordlen) < 0) // go RP
13,528 else if (strFLAG < 0) // go RP
13,529 { // No ?W after RW - go RP
13,530 memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4 + 4, 4 ); //RP
13,531 PseudoLinkedPointer = PseudoLinkedPointerNEW;
13,532 }
13,533 {
13,534 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,535 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8; //RP
13,536 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,537 if (StackPtr > 8192*3-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
13,538 BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
13,539 BSTstack[StackPtr] = 16; ++StackPtr; //LPOffset=0;MPOffset=8;RPOffset=16;
13,540 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,541 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,542 //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
13,543 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,544 // [ //r.14+
13,545 memcpy( &PseudoLinkedPointer_64, &LEAF[8 + 8], 8 );
13,546 // ] //r.14+
13,547 }
13,548 else { FoundInLinkedList = 1; // wrd is RW
13,549 // Counter [
13,550 if (BSTorBtree == 2) {
13,551 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,552 }
13,553 //memcpy( &CounterOccurrences, &FourGramL[(KeySize+1+4)-4], 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
13,554 //if (CounterOccurrences<999999999) CounterOccurrences++; // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
13,555 //memcpy( &FourGramL[(KeySize+1+4)-4], &CounterOccurrences, 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
13,556 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,557 //fwrite(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
13,558 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,559 // [ //r.14+
13,560 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &FourGramL[0], (KeySize+1+8) );
13,561 if (BSTorBtree == 2) {
13,562 fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
13,563 } else { // ##### 64bit memory manipulations [
13,564 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
13,565 } // ##### 64bit memory manipulations ]
13,566 // ] //r.14+
13,567 // Counter ]
13,568 }
13,569 WORDcountAttemptsToPut++;
13,570 // ***** 'P W P' section 2 ]
13,571 // *****
13,572 }
13,573 else // RW empty - go MP
13,574 { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
13,575 PseudoLinkedPointer = PseudoLinkedPointerNEW;

```

```

13,576 //      }
13,577 //      {
13,578 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,579 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8; //MP
13,580 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,581 if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
13,582 BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
13,583 BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
13,584 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,585 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,586 //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
13,587 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,588 // [ //r.14+
13,589 memcpy( &PseudoLinkedPointer_64, &LEAF[8], 8 );
13,590 // ] //r.14+
13,591 //      }
13,592 //    }
13,593 //    else { FoundInLinkedList = 1; // wrd is LW
13,594 // Counter [
13,595 //    if (BSTorBtree == 2) {
13,596 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,597 //    }
13,598 //memcpy( &CounterOccurrences, &FourGramL[(KeySize+1+4)-4], 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
13,599 //if (CounterOccurrences<999999999) CounterOccurrences++; // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
13,600 //memcpy( &FourGramL[(KeySize+1+4)-4], &CounterOccurrences, 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
13,601 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,602 //fwrite(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
13,603 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,604 // [ //r.14+
13,605 memcpy( &LEAF[8 + 8 + 8], &FourGramL[0], (KeySize+1+8) );
13,606 //    if (BSTorBtree == 2) {
13,607 fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
13,608 //    } else { // ##### 64bit memory manipulations [
13,609 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
13,610 //    } // ##### 64bit memory manipulations ]
13,611 // ] //r.14+
13,612 // Counter ]
13,613 //    }
13,614 WORDcountAttemptsToPut++;
13,615 // ***** 'P W P' section ]
13,616 //    } // while
13,617 WORDcountAttemptsToPut--; // - 1 due to BST way of counting i.e. direct hash hit is not counted only successors
13,618 // }
13,619 // 1] Search ]
13,620 if (FoundInLinkedList == 0) NotFoundKeys++; //r.18
13,621 if (FoundInLinkedList == 1) FoundKeys++; //r.18
13,622 if (FoundInLinkedList == 2) NotFoundKeys++; //r.18
13,623
13,624 if (FoundInLinkedList == 0)
13,625 {
13,626 /*
13,627 // ===== [ The whole section/sub-fragment 2 is commented due to great time differences for Internal_vs_External memory accesses - it is far more
cheap to have the STACK overhead (moved to sub-fragment 1) ] ===== [
13,628 // 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search [
13,629 // 'TracingSearch' is the same as 'Search' except that adds the trail in my simulated stack,
13,630 // the goal is not to waste time in 'Search' by dealing with no needed trail in case of not 'Insert'.
13,631 // Simulated stack contains pairs of 'Address of ParentLEAF' + 'Offset of ParentPointer in ParentLEAF i.e. 0 for LP, 4 for MP, 8 for RP'.
13,632 // 'Offset ...' saves unnecessary comparisons of NEWword which after splitting goes up.

```

```

13,633         memcpy( &PseudoLinkedPointer, BufStart+Slot, 4 );
13,634         StackPtr = 0;
13,635         while (PseudoLinkedPointer != 0)
13,636         {
13,637 // ***** 'P W P' section [
13,638 // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
13,639 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
13,640 // here ALWAYS LW exists: no need for existence check - line below
13,641 // if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
13,642         if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wordlen) > 0) // go LP
13,643         { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 0, 4 ); //LP
13,644         if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
13,645         BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
13,646         BSTstack[StackPtr] = 0; ++StackPtr; //LPoffset=0;MPoffset=4;RPOffset=8;
13,647         PseudoLinkedPointer = PseudoLinkedPointerNEW;
13,648         }
13,649         else if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wordlen) < 0) // go RP or MP
13,650         { // RW existence check - line below:
13,651         if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 ) // RW exists
13,652         { // Here all 'P W P' section is repeated; the way of handling case when dynamic number of words in leaf
13,653 // *****
13,654 // ***** 'P W P' section 2 [
13,655 // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
13,656 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
13,657 // here ALWAYS RW exists: no need for existence check - line below
13,658 // if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
13,659         if (memcmp(PseudoLinkedPointer+4+4+4+wordlen, wrd, wordlen) > 0) // go MP
13,660         { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
13,661         if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
13,662         BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
13,663         BSTstack[StackPtr] = 4; ++StackPtr; //LPoffset=0;MPoffset=4;RPOffset=8;
13,664         PseudoLinkedPointer = PseudoLinkedPointerNEW;
13,665         }
13,666         else if (memcmp(PseudoLinkedPointer+4+4+4+wordlen, wrd, wordlen) < 0) // go RP
13,667         { // No ?W after RW - go RP
13,668         memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4 + 4, 4 ); //RP
13,669         if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
13,670         BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
13,671         BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=4;RPOffset=8;
13,672         PseudoLinkedPointer = PseudoLinkedPointerNEW;
13,673         }
13,674         else FoundInLinkedList = 1; // wrd is RW
13,675 // ***** 'P W P' section 2 ]
13,676 // *****
13,677         }
13,678         else // RW empty - go MP
13,679         { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
13,680         if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
13,681         BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
13,682         BSTstack[StackPtr] = 4; ++StackPtr; //LPoffset=0;MPoffset=4;RPOffset=8;
13,683         PseudoLinkedPointer = PseudoLinkedPointerNEW;
13,684         }
13,685         }
13,686         else FoundInLinkedList = 1; // wrd is LW
13,687 // ***** 'P W P' section ]
13,688         } // while
13,689 // 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search ]
13,690 // ===== [ The whole section/sub-fragment 2 is commented due to great time differences for Internal_vs_External memory accesses - it is far more

```

```

cheap to have the STACK overhead (moved to sub-fragment 1) ] ===== ]
13,691 */
13,692
13,693 // 3] Insert Iterative [
13,694 //   There are total 4 situations:
13,695 //   Case #1: Outer NODE(including ROOT) [ ][ ][ ][LW][ ]
13,696 //   Case #2: Outer NODE(including ROOT) [ ][ ][ ][LW][RW] Split Occurs ----
13,697 //   Case #3: ROOT [LP][MP][ ][LW][ ] 'wrUP' (wrklen bytes)
13,698 //   Case #4: Inner NODE(including ROOT) [LP][MP][RP][LW][RW] Split Occurs --- | &
13,699 //                                     | | 'PseudoLinkedPointerNEW' (ptr to NEW LEAF)
13,700 //   There are total 2 situations for PARENT LEAF: <----- ARE GOING UP
13,701 //   Case #3: [LP][MP][ ][LW][ ]
13,702 //   Case #4: [LP][MP][RP][LW][RW] Split Occurs
13,703
13,704 // ~ First deal alongely with the OUTER NODE(LEAF) where Search stopped i.e Case #1 & Case #2:
13,705 //   POffsetInLEAF = BSTstack[--StackPtr];
13,706 //   PseudoLinkedPointer_64 = BSTstack[--StackPtr];
13,707 // NOTE: ONE LEAF IS FULL ONLY WHEN LAST CELL FOR KEY(here RW) EXISTS!
13,708 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wrklen) != 0 )
13,709 //if ( *(char *) (PseudoLinkedPointer+4+4+4+wrklen) != 0 ) // If LEAF is full: Case #2
13,710 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,711 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
13,712 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,713 //fread(&SomeByte, 1, 1, fp_outRG);
13,714 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,715 // [ //r.14+
13,716 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
13,717 //   if (BSTorBtree == 2) {
13,718 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,719 //fread(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fread++;
13,720 //   } else { // ##### 64bit memory manipulations [
13,721 //memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64, 8+8+2*(KeySize+1+8) );
13,722 //   } // ##### 64bit memory manipulations ]
13,723 //memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (KeySize+1+8)], 1 );
13,724 // ] //r.14+
13,725 //   if (SomeByte != 0) // RW exists
13,726 //   { SplitOccured = 1; WORDcountDistinct++;
13,727 //     // ALlocate NEW LEAF:
13,728 //     if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrklen + 4 + 4 + 4 < GRMBLFoolAgain((int)wrklen) ) // +4 more for BST instead of LL; + more(see
LEAF)
13,729 //     {
13,730 //     memcpy( &PseudoLinkedPointerNEW, &bufend[LetterOffset], 4 );
13,731 //     bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
13,732 //     bufend[LetterOffset] = bufend[LetterOffset] + 2*wrklen;
13,733 //     if (MAXusedBuffer[wrklen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrklen] = (unsigned long)(bufend[LetterOffset] -
BufStart);}
13,734 //     }
13,735 //     // BUG IN R.17 and prior - the next 'if' is only for RAM (y/Y) not for SSD (z/Z):
13,736 //     if (BSTorBtree == 2) { //r.18
13,737 //BUGGYoffset = (BufEnd_64-(0+14));
13,738 //     } else { // ##### 64bit memory manipulations [
13,739 //BUGGYoffset = (BufEnd_64-(unsigned long long)pointerflush_64);
13,740 //     } // ##### 64bit memory manipulations ]
13,741 //     if( 8 + 8 + 8 + 2*(KeySize+1+8) < size_in64_L14 - BUGGYoffset ) // the longest wrklen is LongestLineInclusive but actual is LongestLineInclusive(+ CR char)
13,742 //     {
13,743 //     PseudoLinkedPointerNEW_64 = BufEnd_64;
13,744 //     BufEnd_64 = BufEnd_64 + 8 + 8 + 8;
13,745 //     BufEnd_64 = BufEnd_64 + 2*(KeySize+1+8);

```

```

13,746         }
13,747     else
13,748     { printf( "\nLeprechaun: Failure! Increment 'Memory for B-trees'!\n" );
13,749     fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, 11TOaDigits, 10), _ui64toaKAZEcomma((unsigned long)WORDcountDistinct, 11TOaDigits2, 10) );
13,750     fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14, 11TOaDigits, 10) );
13,751     fprintf( fp_outLOG, "Total Attempts to Find/Put WORDS into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11TOaDigits, 10) );
13,752     fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
13,753     fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for B-trees'!\n\n");
13,754     exit( 7 );
13,755     }
13,756     if (POffsetInLEAF == 0) // wrd < LW
13,757     {
13,758         //      memcpy( wrdUP, PseudoLinkedPointer+4+4+4, wrdlen ); // LW up
13,759         //      memcpy( PseudoLinkedPointer+4+4+4, wrd, wrdlen ); // wrd go to OLD LEAF
13,760         //      memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
13,761         //      *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
13,762         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,763         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
13,764         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,765         //fread(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
13,766         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,767         //fwrite(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
13,768         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
13,769         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,770         //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
13,771         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
13,772         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,773         //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
13,774         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
13,775         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,776         //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
13,777         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,778         // [ //r.14+
13,779         memcpy( &wrdUP[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
13,780         memcpy( &LEAF[8 + 8 + 8], &wrd[0], (KeySize+1+8) );
13,781         memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
13,782         // Here reordering (of writing wrdAUX) is needed to avoid seek the position NEW and stupidly to seek again OLD/current position!
13,783         memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
13,784         if (BSTorBtree == 2) {
13,785             fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,786             fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
13,787         } else { // ##### 64bit memory manipulations [
13,788             memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
13,789             // ##### 64bit memory manipulations ]
13,790             PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
13,791             if (BSTorBtree == 2) {
13,792                 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,793                 fwrite(&wrdAUX[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
13,794             } else { // ##### 64bit memory manipulations [
13,795                 memcpy( (char *)PseudoLinkedPointerAUX_64, &wrdAUX[0], (KeySize+1+8) );
13,796                 // ##### 64bit memory manipulations ]
13,797             } // ] //r.14+
13,798         }
13,799     if (POffsetInLEAF == 8) // LW < wrd < RW
13,800     {
13,801         //      memcpy( wrdUP, wrd, wrdlen ); // wrd up
13,802         //      memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF

```

```

13,803 //          *(char *)(&PseudoLinkedPointer+4+4+4+wrklen) = 0;    // RW mark unused in OLD LEAF
13,804          memcpy( wrdUP, wrd, (KeySize+1+8) );    // wrd up
13,805          // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,806          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
13,807          //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,808          //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
13,809          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
13,810          //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,811          //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
13,812          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
13,813          //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,814          //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
13,815          // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,816          // [ //r.14+
13,817          memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
13,818          // Here reordering (of writing wrdAUX) is needed to avoid seek the position NEW and stupidly to seek again OLD/current position!
13,819          memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
13,820          if (BSTorBtree == 2) {
13,821          fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,822          fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
13,823          } else { // ##### 64bit memory manipulations [
13,824          memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
13,825          } // ##### 64bit memory manipulations ]
13,826          PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
13,827          if (BSTorBtree == 2) {
13,828          fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,829          fwrite(&wrdAUX[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
13,830          } else { // ##### 64bit memory manipulations [
13,831          memcpy( (char *)PseudoLinkedPointerAUX_64, &wrdAUX[0], (KeySize+1+8) );
13,832          } // ##### 64bit memory manipulations ]
13,833          // ] //r.14+
13,834          }
13,835          if (PoffsetInLEAF == 16) // wrd > RW
13,836          {
13,837          memcpy( wrdUP, PseudoLinkedPointer+4+4+4+wrklen, wrklen ); // RW up
13,838          *(char *)(&PseudoLinkedPointer+4+4+4+wrklen) = 0;    // RW mark unused in OLD LEAF
13,839          memcpy( PseudoLinkedPointerNEW+4+4+4, wrd, wrklen );    // wrd go to NEW LEAF
13,840          // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,841          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
13,842          //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,843          //fread(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
13,844          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
13,845          //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,846          //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
13,847          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
13,848          //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,849          //fwrite(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
13,850          // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,851          // [ //r.14+
13,852          memcpy( &wrdUP[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
13,853          memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
13,854          if (BSTorBtree == 2) {
13,855          fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,856          fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
13,857          } else { // ##### 64bit memory manipulations [
13,858          memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
13,859          } // ##### 64bit memory manipulations ]
13,860          // ] //r.14+

```

```

13,861 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one! Here NO need!
13,862 PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
13,863 if (BSTorBtree == 2) {
13,864     fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,865     fwrite(&wrd[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
13,866     } else { // ##### 64bit memory manipulations [
13,867 memcpy( (char *)PseudoLinkedPointerAUX_64, &wrd[0], (KeySize+1+8) );
13,868     } // ##### 64bit memory manipulations ]
13,869 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one! Here NO need!
13,870 }
13,871 }
13,872 else // If LEAF is not full: Case #1
13,873 { SplitOccured = 0; WORDcountDistinct++;
13,874     if (POffsetInLEAF == 0) // wrd < [LW][ ] so [LW][ ] -> [ ][LW] -> [wrd][LW]
13,875     {
13,876 //         memcpy( PseudoLinkedPointer+4+4+4+wrdlen, PseudoLinkedPointer+4+4+4, wrdlen );
13,877 //         memcpy( PseudoLinkedPointer+4+4+4, wrd, wrdlen );
13,878 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,879 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
13,880 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,881 //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
13,882 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
13,883 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,884 //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
13,885 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
13,886 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,887 //fwrite(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
13,888 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,889 // [ //r.14+
13,890 memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
13,891 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &wrdAUX[0], (KeySize+1+8) );
13,892 memcpy( &LEAF[8 + 8 + 8], &wrd[0], (KeySize+1+8) );
13,893     if (BSTorBtree == 2) {
13,894 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,895 fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
13,896     } else { // ##### 64bit memory manipulations [
13,897 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
13,898     } // ##### 64bit memory manipulations ]
13,899 // ] //r.14+
13,900 }
13,901 }
13,902 if (POffsetInLEAF == 8) // wrd > [LW][ ] so [LW][ ] -> [LW][wrd]
13,903 {
13,904 //     memcpy( PseudoLinkedPointer+4+4+4+wrdlen, wrd, wrdlen );
13,905 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one! Here NO need!
13,906 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (KeySize+1+8);
13,907     if (BSTorBtree == 2) {
13,908 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,909 fwrite(&wrd[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
13,910     } else { // ##### 64bit memory manipulations [
13,911 memcpy( (char *)PseudoLinkedPointerAUX_64, &wrd[0], (KeySize+1+8) );
13,912     } // ##### 64bit memory manipulations ]
13,913 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one! Here NO need!
13,914 }
13,915 }
13,916
13,917 if (SplitOccured != 0)
13,918 {

```



```

13,919 // ~ Second deal with the INNER NODE(S) i.e Case #3 & Case #4:
13,920 while (StackPtr != 0 || SplitOccured != 0)
13,921 {
13,922     // 'PseudoLinkedPointerNEW' is new LEAF to be inserted
13,923     // 'wrUP' is NEW word to be inserted
13,924     if (StackPtr != 0)
13,925     {
13,926         POffsetInLEAF = BSTstack[--StackPtr];
13,927         PseudoLinkedPointer_64 = BSTstack[--StackPtr];
13,928 //if ( *(char *) (PseudoLinkedPointer+4+4+4+wrklen) != 0 ) // If LEAF is full: Case #4
13,929 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,930 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
13,931 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,932 //fread(&SomeByte, 1, 1, fp_outRG);
13,933 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
13,934 // [ //r.14+
13,935 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
13,936 if (BSTorBtree == 2) {
13,937     fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,938     fread(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fread++;
13,939     } else { // ##### 64bit memory manipulations [
13,940     memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64, 8+8+2*(KeySize+1+8) );
13,941     } // ##### 64bit memory manipulations ]
13,942     memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (KeySize+1+8)], 1 );
13,943     // ] //r.14+
13,944     if (SomeByte != 0 ) // RW exists
13,945     { SplitOccured = 1;
13,946     //     memcpy( wrUPold, wrUP, wrklen ); // LW up
13,947     //     PseudoLinkedPointerNEWold = PseudoLinkedPointerNEW;
13,948     //     memcpy( wrUPold, wrUP, (KeySize+1+8) );
13,949     //     PseudoLinkedPointerNEWold_64 = PseudoLinkedPointerNEW_64;
13,950     // ALlocate NEW LEAF:
13,951     //     if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrklen + 4 + 4 + 4 < GRMELFoolAgain[(int)wrklen] ) // +4 more for BST instead of LL; + more(see
LEAF)
13,952     //     {
13,953     //         memcpy( &PseudoLinkedPointerNEW, &bufend[LetterOffset], 4 );
13,954     //         bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
13,955     //         bufend[LetterOffset] = bufend[LetterOffset] + 2*wrklen;
13,956     //         if (MAXusedBuffer[wrklen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrklen] = (unsigned long)(bufend[LetterOffset] -
BufStart);}
13,957     //     }
13,958     // BUG IN R.17 and prior - the next 'if' is only for RAM (y/Y) not for SSD (z/Z):
13,959     // if (BSTorBtree == 2) { //r.18
13,960     BUGGYoffset = (BufEnd_64-(0+14));
13,961     // } else { // ##### 64bit memory manipulations [
13,962     BUGGYoffset = (BufEnd_64-(unsigned long long)pointerflush_64);
13,963     // } // ##### 64bit memory manipulations ]
13,964     // if( 8 + 8 + 8 + 2*(KeySize+1+8) < size_in64_L14 - BUGGYoffset ) // the longest wrklen is LongestLineInclusive but actual is LongestLineInclusive(+ CR char)
13,965     // {
13,966     PseudoLinkedPointerNEW_64 = BufEnd_64;
13,967     BufEnd_64 = BufEnd_64 + 8 + 8 + 8;
13,968     BufEnd_64 = BufEnd_64 + 2*(KeySize+1+8);
13,969     // [ //r.14+
13,970     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64;
13,971     //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
13,972     //fread(&LEAFNEW[0], 8+8+2*(LongestLineInclusive+1+4), 1, fp_outRG);
13,973     // In fact above three lines are slow, the only need is ZEROed LEAFNEW.
13,974     memset(&LEAFNEW[0], 0, 8+8+2*(KeySize+1+8));

```

```

13,975         // ] //r.14+
13,976     }
13,977     else
13,978     { printf( "\nLeprechaun: Failure! Increment 'Memory for B-trees'!\n" );
13,979 //      fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, 11TOaDigits, 10), _ui64toaKAZEcomma((unsigned long
long)WORDcountDistinct, 11TOaDigits2, 10) );
13,980     fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14, 11TOaDigits, 10) );
13,981     fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11TOaDigits, 10) );
13,982     fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
13,983     fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for B-trees'!\n\n");
13,984     exit( 7 );
13,985 }
13,986 if (POffsetInLEAF == 0) // wrdUPold < LW
13,987 {
13,988 //     memcpy( wrdUP, PseudoLinkedPointer+4+4+4, wrdlen ); // LW up
13,989 //     memcpy( PseudoLinkedPointer+4+4+4, wrdUPold, wrdlen ); // wrdUPold go to OLD LEAF
13,990 //     memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
13,991 //     *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
13,992 //     // [LP](PseudoLinkedPointerNEWold)[MP][RP](wrdUPold)[LW][RW] -----
13,993 //     // pair [LW] PseudoLinkedPointerNEW goes up |
13,994 //     // PseudoLinkedPointer: PseudoLinkedPointerNEW: |
13,995 //     // [LP](PseudoLinkedPointerNEWold)[ ](wrdUPold) [MP][RP][ ][RW] <----
13,996 //     // no need to put zero in RP because logic is based on words existence:
13,997 //     memcpy( PseudoLinkedPointerNEW+0, PseudoLinkedPointer+4, 4 );
13,998 //     memcpy( PseudoLinkedPointerNEW+4, PseudoLinkedPointer+8, 4 );
13,999 //     memcpy( PseudoLinkedPointer+4, &PseudoLinkedPointerNEWold, 4 );
14,000 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
14,001 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
14,002 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,003 //fread(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
14,004 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,005 //fwrite(&wrdUPold[0], (LongestLineInclusive+1+4), 1, fp_outRG);
14,006 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
14,007 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,008 //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
14,009 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
14,010 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,011 //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
14,012 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
14,013 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,014 //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
14,015 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
14,016 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,017 //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
14,018 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 0;
14,019 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,020 //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
14,021 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
14,022 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,023 //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
14,024 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8;
14,025 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,026 //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
14,027 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
14,028 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,029 //fwrite(&PseudoLinkedPointerNEWold_64, 8, 1, fp_outRG);
14,030 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
14,031 // [ //r.14+

```

```

14,032 memcpy( &wrdUP[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
14,033 memcpy( &LEAF[8 + 8 + 8], &wrdUPold[0], (KeySize+1+8) );
14,034 memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
14,035 memcpy( &LEAFNEW[8 + 8 + 8], &wrdAUX[0], (KeySize+1+8) );
14,036 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
14,037 memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[8], 8 );
14,038 memcpy( &LEAFNEW[0], &PseudoLinkedPointerAUXdumbo_64, 8 );
14,039 memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[16], 8 );
14,040 memcpy( &LEAFNEW[8], &PseudoLinkedPointerAUXdumbo_64, 8 );
14,041 memcpy( &LEAF[8], &PseudoLinkedPointerNEWold_64, 8 );
14,042 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
14,043 if (BSTorBtree == 2) {
14,044 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,045 fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
14,046 } else { // ##### 64bit memory manipulations [
14,047 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
14,048 } // ##### 64bit memory manipulations ]
14,049 PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64;
14,050 if (BSTorBtree == 2) {
14,051 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,052 fwrite(&LEAFNEW[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
14,053 } else { // ##### 64bit memory manipulations [
14,054 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAFNEW[0], 8+8+8+2*(KeySize+1+8) );
14,055 } // ##### 64bit memory manipulations ]
14,056 // ] //r.14+
14,057 }
14,058 if (PoffsetInLEAF == 8) // LW < wrdUPold < RW
14,059 {
14,060 // memcpy( wrdUP, wrdUPold, wrdlen ); // wrdUPold up
14,061 // memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW go to NEW LEAF
14,062 // *(char *)PseudoLinkedPointer+4+4+4+wrdlen = 0; // RW mark unused in OLD LEAF
14,063 // // [LP][MP](PseudoLinkedPointerNEWold)[RP][LW](wrdUPold)[RW] -----
14,064 // // pair [wrdUPold] PseudoLinkedPointerNEW goes up |
14,065 // // PseudoLinkedPointer: PseudoLinkedPointerNEW: |
14,066 // // [LP][MP][ ][LW] (PseudoLinkedPointerNEWold)[RP][ ][RW] <----
14,067 // // no need to put zero in RP because logic is based on words existence:
14,068 // memcpy( PseudoLinkedPointerNEW+0, &PseudoLinkedPointerNEWold, 4 );
14,069 // memcpy( PseudoLinkedPointerNEW+4, PseudoLinkedPointer+8, 4 );
14,070 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
14,071 memcpy( wrdUP, wrdUPold, (KeySize+1+8) );
14,072 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
14,073 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,074 //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
14,075 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
14,076 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,077 //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
14,078 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
14,079 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,080 //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
14,081 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 0;
14,082 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,083 //fwrite(&PseudoLinkedPointerNEWold_64, 8, 1, fp_outRG);
14,084 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
14,085 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,086 //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
14,087 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8;
14,088 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,089 //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);

```

```

14,090 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
14,091 // [ //r.14+
14,092 memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
14,093 memcpy( &LEAFNEW[8 + 8 + 8], &wrdAUX[0], (KeySize+1+8) );
14,094 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
14,095 memcpy( &LEAFNEW[0], &PseudoLinkedPointerNEWold_64, 8 );
14,096 memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[16], 8 );
14,097 memcpy( &LEAFNEW[8], &PseudoLinkedPointerAUXdumbo_64, 8 );
14,098 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
14,099 if (BSTorBtree == 2) {
14,100 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,101 fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
14,102 } else { // ##### 64bit memory manipulations [
14,103 memcpy( (char *)&PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
14,104 } // ##### 64bit memory manipulations ]
14,105 PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64;
14,106 if (BSTorBtree == 2) {
14,107 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,108 fwrite(&LEAFNEW[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
14,109 } else { // ##### 64bit memory manipulations [
14,110 memcpy( (char *)&PseudoLinkedPointerAUX_64, &LEAFNEW[0], 8+8+2*(KeySize+1+8) );
14,111 } // ##### 64bit memory manipulations ]
14,112 // ] //r.14+
14,113 }
14,114 if (PoffsetInLEAF == 16) // wrdUPold > RW
14,115 {
14,116     memcpy( wrdUP, PseudoLinkedPointer+4+4+4+wrdlen, wrdlen ); // RW up
14,117     *(char *)&(PseudoLinkedPointer+4+4+4+wrdlen) = 0; // RW mark unused in OLD LEAF
14,118     memcpy( PseudoLinkedPointerNEW+4+4+4, wrdUPold, wrdlen ); // wrdUPold go to NEW LEAF
14,119     // [LP][MP][RP](PseudoLinkedPointerNEWold)[LW][RW](wrdUPold) -----
14,120     // pair [RW] PseudoLinkedPointerNEW goes up |
14,121     // PseudoLinkedPointer: PseudoLinkedPointerNEW: |
14,122     // [LP][MP][LW] [RP](PseudoLinkedPointerNEWold)[LW](wrdUPold) <---
14,123     // no need to put zero in RP because logic is based on words existence:
14,124     memcpy( PseudoLinkedPointerNEW+0, PseudoLinkedPointer+8, 4 );
14,125     memcpy( PseudoLinkedPointerNEW+4, &PseudoLinkedPointerNEWold, 4 );
14,126 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
14,127 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
14,128 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,129 //fread(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
14,130 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
14,131 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,132 //fwrite(&OneChar_ieByte, 1, 1, fp_outRG); // Write ZERO ASCII code
14,133 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8 + 8 + 8;
14,134 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,135 //fwrite(&wrdUPold[0], (LongestLineInclusive+1+4), 1, fp_outRG);
14,136 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
14,137 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,138 //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
14,139 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 0;
14,140 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,141 //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
14,142 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64 + 8;
14,143 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,144 //fwrite(&PseudoLinkedPointerNEWold_64, 8, 1, fp_outRG);
14,145 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
14,146 // [ //r.14+
14,147 memcpy( &wrdUP[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );

```

```

14,148     memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &OneChar_ieByte, 1 );
14,149     memcpy( &LEAFNEW[8 + 8 + 8], &wrdUPold[0], (KeySize+1+8) );
14,150     memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[16], 8 );
14,151     memcpy( &LEAFNEW[0], &PseudoLinkedPointerAUXdumbo_64, 8 );
14,152     memcpy( &LEAFNEW[8], &PseudoLinkedPointerNEWold_64, 8 );
14,153     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
14,154     if (BSTorBtree == 2) {
14,155         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,156         fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
14,157         } else { // ##### 64bit memory manipulations [
14,158     memcpy( (char *)&PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
14,159         } // ##### 64bit memory manipulations ]
14,160     PseudoLinkedPointerAUX_64 = PseudoLinkedPointerNEW_64;
14,161     if (BSTorBtree == 2) {
14,162         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,163         fwrite(&LEAFNEW[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
14,164         } else { // ##### 64bit memory manipulations [
14,165     memcpy( (char *)&PseudoLinkedPointerAUX_64, &LEAFNEW[0], 8+8+8+2*(KeySize+1+8) );
14,166         } // ##### 64bit memory manipulations ]
14,167     // ] //r.14+
14,168 }
14,169 }
14,170 else // If LEAF is not full: Case #3
14,171 { SplitOccured = 0;
14,172     if (POffsetInLEAF == 0) // wrdUP < [LW][ ] so [LW][ ] -> [ ][LW] -> [wrdUP][LW]
14,173     {
14,174         //     memcpy( PseudoLinkedPointer+4+4+4+wrdlen, PseudoLinkedPointer+4+4+4, wrdlen );
14,175         //     memcpy( PseudoLinkedPointer+4+4+4, wrdUP, wrdlen );
14,176         //         // [LP][MP][ ] -> [LP][ ][MP] -> [LP][np][MP]
14,177         //     memcpy( PseudoLinkedPointer+8, PseudoLinkedPointer+4, 4 );
14,178         //     memcpy( PseudoLinkedPointer+4, &PseudoLinkedPointerNEW, 4 );
14,179         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
14,180         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
14,181         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,182         //fread(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
14,183         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
14,184         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,185         //fwrite(&wrdAUX[0], (LongestLineInclusive+1+4), 1, fp_outRG);
14,186         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
14,187         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,188         //fwrite(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
14,189         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
14,190         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,191         //fread(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
14,192         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
14,193         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,194         //fwrite(&PseudoLinkedPointerAUXdumbo_64, 8, 1, fp_outRG);
14,195         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
14,196         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,197         //fwrite(&PseudoLinkedPointerNEW_64, 8, 1, fp_outRG);
14,198         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
14,199         // [ //r.14+
14,200     memcpy( &wrdAUX[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
14,201     memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &wrdAUX[0], (KeySize+1+8) );
14,202     memcpy( &LEAF[8 + 8 + 8], &wrdUP[0], (KeySize+1+8) );
14,203     memcpy( &PseudoLinkedPointerAUXdumbo_64, &LEAF[8], 8 );
14,204     memcpy( &LEAF[8 + 8], &PseudoLinkedPointerAUXdumbo_64, 8 );
14,205     memcpy( &LEAF[8], &PseudoLinkedPointerNEW_64, 8 );

```

```

14,206         if (BSTorBtree == 2) {
14,207             fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,208             fwrite(&LEAF[0], 8*8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
14,209             } else { // ##### 64bit memory manipulations [
14,210             memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8*8+2*(KeySize+1+8) );
14,211             } // ##### 64bit memory manipulations ]
14,212             // ] //r.14+
14,213         }
14,214     if (POffsetInLEAF == 8) // wrdUP > [LW][ ] so [LW][ ] -> [LW][wrdUP]
14,215     {
14,216         memcpy( PseudoLinkedPointer+4+4+4+wrdlen, wrdUP, wrdlen );
14,217         // [LP][MP][ ] -> [LP][MP][np]
14,218         memcpy( PseudoLinkedPointer+8, &PseudoLinkedPointerNEW, 4 );
14,219         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
14,220         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
14,221         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,222         //fwrite(&wrdUP[0], (LongestLineInclusive+1+4), 1, fp_outRG);
14,223         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 16;
14,224         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,225         //fwrite(&PseudoLinkedPointerNEW_64, 8, 1, fp_outRG);
14,226         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
14,227         // [ //r.14+
14,228         memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &wrdUP[0], (KeySize+1+8) );
14,229         memcpy( &LEAF[8 + 8], &PseudoLinkedPointerNEW_64, 8 );
14,230         if (BSTorBtree == 2) {
14,231             fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,232             fwrite(&LEAF[0], 8*8+2*(KeySize+1+8), 1, fp_outRG); Total_fwrite++;
14,233             } else { // ##### 64bit memory manipulations [
14,234             memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8*8+2*(KeySize+1+8) );
14,235             } // ##### 64bit memory manipulations ]
14,236             // ] //r.14+
14,237         }
14,238         break;
14,239     }
14,240 }
14,241 else // Empty stack means ROOT and more over ROOT is already splitted(Case #4 is off)
14,242 {
14,243     // If LEAF is not full: Case #3
14,244     // THIS IS WHERE A NEW(SECOND) LEAF 'PseudoLinkedPointerROOT' must be allocated:
14,245     // if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrdlen + 4 + 4 + 4 < GRMBLPoolAgain((int)wrdlen) ) // +4 more for BST instead of LL; + more(see
LEAF)
14,246     {
14,247         memcpy( &PseudoLinkedPointerROOT, &bufend[LetterOffset], 4 );
14,248         bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
14,249         memcpy( bufend[LetterOffset], wrdUP, wrdlen );
14,250         bufend[LetterOffset] = bufend[LetterOffset] + 2*wrdlen;
14,251         if (MAXusedBuffer[wrdlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrdlen] = (unsigned long)(bufend[LetterOffset] -
BufStart);}
14,252     }
14,253     // BUG IN R.17 and prior - the next 'if' is only for RAM (y/Y) not for SSD (z/Z):
14,254     if (BSTorBtree == 2) { //r.18
14,255         BUGGYoffset = (BufEnd_64-(0+14));
14,256         } else { // ##### 64bit memory manipulations [
14,257         BUGGYoffset = (BufEnd_64-(unsigned long long)pointerflush_64);
14,258         } // ##### 64bit memory manipulations ]
14,259         if( 8 + 8 + 8 + 2*(KeySize+1+8) < size_in64_L14 - BUGGYoffset ) // the longest wrdlen is LongestLineInclusive but actual is LongestLineInclusive(+ CR char)
14,260         {
14,261             PseudoLinkedPointerROOT_64 = BufEnd_64;

```

```

14,262         BufEnd_64 = BufEnd_64 + 8 + 8 + 8;
14,263         BufEnd_64 = BufEnd_64 + 2*(KeySize+1+8);
14,264         PseudoLinkedPointerAUX_64 = PseudoLinkedPointerROOT_64 + 8 + 8 + 8;
14,265         if (BSTorBtree == 2) {
14,266             fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,267             fwrite(&wrdUP[0], (KeySize+1+8), 1, fp_outRG); Total_fwrite++;
14,268         } else { // ##### 64bit memory manipulations [
14,269             memcpy( (char *)PseudoLinkedPointerAUX_64, &wrdUP[0], (KeySize+1+8) );
14,270         } // ##### 64bit memory manipulations ]
14,271     }
14,272     else
14,273     { printf( "\nLeprechaun: Failure! Increment 'Memory for B-trees'!\n" );
14,274       fprintf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, 11ToaDigits, 10), _ui64toaKAZEcomma((unsigned long)WORDcountDistinct, 11ToaDigits2, 10) );
14,275       fprintf( fp_outLOG, "Allocated memory: %s bytes\n", _ui64toaKAZEcomma(size_in64_L14, 11ToaDigits, 10) );
14,276       fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11ToaDigits, 10) );
14,277       fprintf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
14,278       fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for B-trees'!\n\n");
14,279       exit( 7 );
14,280     }
14,281     // Here          -- 'PseudoLinkedPointerROOT' --
14,282     //              |                      |
14,283     // 'PseudoLinkedPointer' <--          --> 'PseudoLinkedPointerNEW'
14,284     //              (LW)                      (RW)
14,285     // memcpy( PseudoLinkedPointerROOT, &PseudoLinkedPointer, 4 ); // LP
14,286     // memcpy( PseudoLinkedPointerROOT+4, &PseudoLinkedPointerNEW, 4 ); // MP
14,287     // Here must NEW ROOT be updated i.e. HASH table(SLOT) must point it:
14,288     // memcpy( BufStart+Slot, &PseudoLinkedPointerROOT, 4 );
14,289     PseudoLinkedPointerAUX_64 = PseudoLinkedPointerROOT_64;
14,290     if (BSTorBtree == 2) {
14,291         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,292         fwrite(&PseudoLinkedPointer_64, 8, 1, fp_outRG); Total_fwrite++;
14,293         } else { // ##### 64bit memory manipulations [
14,294             memcpy( (char *)PseudoLinkedPointerAUX_64, &PseudoLinkedPointer_64, 8 );
14,295         } // ##### 64bit memory manipulations ]
14,296     PseudoLinkedPointerAUX_64 = PseudoLinkedPointerROOT_64 + 8;
14,297     if (BSTorBtree == 2) {
14,298         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,299         fwrite(&PseudoLinkedPointerNEW_64, 8, 1, fp_outRG); Total_fwrite++;
14,300         } else { // ##### 64bit memory manipulations [
14,301             memcpy( (char *)PseudoLinkedPointerAUX_64, &PseudoLinkedPointerNEW_64, 8 );
14,302         } // ##### 64bit memory manipulations ]
14,303     memcpy( BufStart+Slot, &PseudoLinkedPointerROOT_64, 8 );
14,304     break; //because it is ROOT without split
14,305 }
14,306     } // while
14,307 } //if (SplitOccured != 0)
14,308 // 3] Insert Iterative ]
14,309 } //if (FoundInLinkedList == 0)
14,310 // ##### B-tree order 3 64bit ]
14,311 } //if ( (Slot>>HashChunkSizeInBITS) == RipPasses ) {
14,312 } //for (BuildingBlocksSTRIDE=0;
14,313
14,314 RipPasses++;
14,315 if (RipPasses <= (1<<<(HashInBITS_GLOBAL-HashChunkSizeInBITS_GLOBAL))-1) goto WhyTheHellForIsNotWorking;
14,316 //} // for( RipPasses = 1-1; RipPasses <= (1<<<(HashInBITS-HashChunkSizeInBITS))-1; RipPasses++ )
14,317
14,318 // Counting trees [

```

Listing: Nakamichi Ryuugan-ditto-1TB btree.c; Last version: 2021-Aug-30; Font: MxPlus ToshibaTxL2 8x16.ttf; Downloadable at: www.sanmayce.com/Nakamichi/Kaidanji.zip


```

14,377 while (PseudoLinkedPointer_64 != 0)
14,378 {
14,379     if (StackPtr > 8192*3-1) { printf( "\nLeprechaun: Failure! B-tree simulated stack overflow, too high B-tree!\n" ); exit( 13 );}
14,380     BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //ptr to Rwr
14,381 //if ( *(char *) (PseudoLinkedPointer + 4 + 4 + 4 + i%31+1) == 0 ) {memcpy( PseudoLinkedPointer + 4 + 4, &BufStart[NumberOfSLOTS*4], 4 );}
14,382 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
14,383 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
14,384 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,385 //fread(&SomeByte, 1, 1, fp_outRG);
14,386 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
14,387 // [ //r.14+
14,388 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
14,389 if (BSTorBtree == 2) {
14,390     fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,391     fread(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fread++;
14,392     } else { // ##### 64bit memory manipulations [
14,393     memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64, 8+8+2*(KeySize+1+8) );
14,394     } // ##### 64bit memory manipulations ]
14,395     memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (KeySize+1+8)], 1 );
14,396     // ] //r.14+
14,397 if (SomeByte == 0 ) // RW exists not
14,398 {
14,399     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8;
14,400     if (BSTorBtree == 2) {
14,401     fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,402     fwrite(&NULLs_64, 8, 1, fp_outRG); Total_fwrite++;
14,403     } else { // ##### 64bit memory manipulations [
14,404     memcpy( (char *)PseudoLinkedPointerAUX_64, &NULLs_64, 8 );
14,405     } // ##### 64bit memory manipulations ]
14,406     // [ //r.14+
14,407     memcpy( &LEAF[8 + 8], &NULLs_64, 8 );
14,408     // ] //r.14+
14,409 }
14,410 //     memcpy( &PseudoLinkedPointerNEWleft, PseudoLinkedPointer, 4 );
14,411 //     memcpy( &PseudoLinkedPointerNEWMiddle, PseudoLinkedPointer + 4, 4 );
14,412 //     memcpy( &PseudoLinkedPointerNEWRright, PseudoLinkedPointer + 4 + 4, 4 );
14,413 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
14,414 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
14,415 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,416 //fread(&PseudoLinkedPointerNEWleft_64, 8, 1, fp_outRG);
14,417 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
14,418 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,419 //fread(&PseudoLinkedPointerNEWMiddle_64, 8, 1, fp_outRG);
14,420 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8;
14,421 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,422 //fread(&PseudoLinkedPointerNEWRright_64, 8, 1, fp_outRG);
14,423 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
14,424 // [ //r.14+
14,425 memcpy( &PseudoLinkedPointerNEWleft_64, &LEAF[0], 8 );
14,426 memcpy( &PseudoLinkedPointerNEWMiddle_64, &LEAF[8], 8 );
14,427 memcpy( &PseudoLinkedPointerNEWRright_64, &LEAF[8+8], 8 );
14,428 // ] //r.14+
14,429 // Give first from right to left non-zero PTR
14,430 if (PseudoLinkedPointerNEWRright_64 != 0 )
14,431 { memcpy( PseudoLinkedPointer + 4 + 4, &BufStart[NumberOfSLOTS*4], 4 );
14,432 PseudoLinkedPointer = PseudoLinkedPointerNEWRright;
14,433 }
14,434 {

```

```

14,435     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8;
14,436         if (BSTorBtree == 2) {
14,437     fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,438     fwrite(&NULLs_64, 8, 1, fp_outRG); Total_fwrite++;
14,439         } else { // ##### 64bit memory manipulations [
14,440             memcpy( (char *)&PseudoLinkedPointerAUX_64, &NULLs_64, 8 );
14,441         } // ##### 64bit memory manipulations ]
14,442     PseudoLinkedPointer_64 = PseudoLinkedPointerNEWright_64;
14,443 }
14,444     else if (PseudoLinkedPointerNEWmiddle_64 !=0 )
14,445 //         { memcpy( PseudoLinkedPointer + 4, &BufStart[NumberOfSLOTS*4], 4 );
14,446 //             PseudoLinkedPointer = PseudoLinkedPointerNEWmiddle;
14,447 //         }
14,448     {
14,449     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
14,450         if (BSTorBtree == 2) {
14,451     fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,452     fwrite(&NULLs_64, 8, 1, fp_outRG); Total_fwrite++;
14,453         } else { // ##### 64bit memory manipulations [
14,454             memcpy( (char *)&PseudoLinkedPointerAUX_64, &NULLs_64, 8 );
14,455         } // ##### 64bit memory manipulations ]
14,456     PseudoLinkedPointer_64 = PseudoLinkedPointerNEWmiddle_64;
14,457 }
14,458     else if (PseudoLinkedPointerNEWleft_64 !=0 )
14,459 //         { memcpy( PseudoLinkedPointer, &BufStart[NumberOfSLOTS*4], 4 );
14,460 //             PseudoLinkedPointer = PseudoLinkedPointerNEWleft;
14,461 //         }
14,462     {
14,463     PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
14,464         if (BSTorBtree == 2) {
14,465     fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,466     fwrite(&NULLs_64, 8, 1, fp_outRG); Total_fwrite++;
14,467         } else { // ##### 64bit memory manipulations [
14,468             memcpy( (char *)&PseudoLinkedPointerAUX_64, &NULLs_64, 8 );
14,469         } // ##### 64bit memory manipulations ]
14,470     PseudoLinkedPointer_64 = PseudoLinkedPointerNEWleft_64;
14,471 }
14,472     else
14,473     {
14,474         PseudoLinkedPointer_64 = 0;
14,475     }
14,476 }
14,477 if (LevelsInCorona_Not_Counting_ROOT < StackPtr) LevelsInCorona_Not_Counting_ROOT = StackPtr; //r.14
14,478 if (StackPtr == 0) break;
14,479 PseudoLinkedPointer_64 = BSTstack[--StackPtr];
14,480 //     memcpy( &PseudoLinkedPointerNEWleft, PseudoLinkedPointer, 4 );
14,481 //     memcpy( &PseudoLinkedPointerNEWmiddle, PseudoLinkedPointer + 4, 4 );
14,482 //     memcpy( &PseudoLinkedPointerNEWright, PseudoLinkedPointer + 4 + 4, 4 );
14,483 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
14,484 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
14,485 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,486 //fread(&PseudoLinkedPointerNEWleft_64, 8, 1, fp_outRG);
14,487 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8;
14,488 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,489 //fread(&PseudoLinkedPointerNEWmiddle_64, 8, 1, fp_outRG);
14,490 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8;
14,491 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,492 //fread(&PseudoLinkedPointerNEWright_64, 8, 1, fp_outRG);

```

```

14,493 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
14,494 // [ //r.14+
14,495 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
14,496 if (BSTorBtree == 2) {
14,497 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,498 fread(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG); Total_fread++;
14,499 } else { // ##### 64bit memory manipulations [
14,500 memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64, 8+8+2*(KeySize+1+8) );
14,501 } // ##### 64bit memory manipulations ]
14,502 memcpy( &PseudoLinkedPointerNEWleft_64, &LEAF[0], 8 );
14,503 memcpy( &PseudoLinkedPointerNEWMiddle_64, &LEAF[8], 8 );
14,504 memcpy( &PseudoLinkedPointerNEWright_64, &LEAF[8+8], 8 );
14,505 // ] //r.14+
14,506 if (PseudoLinkedPointerNEWleft_64 + PseudoLinkedPointerNEWMiddle_64 + PseudoLinkedPointerNEWright_64 == 0) // One LEAF is PRINTED when LP=0 MP=0 RP=0
14,507 {
14,508 // memcpy( wrd, PseudoLinkedPointer + 4 + 4 + 4, i%31+1 );
14,509 // fwrite(wrd, i%31+1, 1, fp_out);
14,510 // fwrite(CBdLFa, 2, 1, fp_out);
14,511 // if ( *(char *) (PseudoLinkedPointer + 4 + 4 + 4 + i%31+1) != 0 )
14,512 // { memcpy( wrd, PseudoLinkedPointer + 4 + 4 + 4 + i%31+1, i%31+1 );
14,513 // fwrite(wrd, i%31+1, 1, fp_out);
14,514 // fwrite(CBdLFa, 2, 1, fp_out);
14,515 // }
14,516 // PseudoLinkedPointer = 0;
14,517 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
14,518 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
14,519 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,520 //fread(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
14,521 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
14,522 // [ //r.14+
14,523 memcpy( &wrd[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
14,524 // ] //r.14+
14,525 // Counter [
14,526 //memcpy( &CounterOccurrences, &wrd[(KeySize+1+4)-4], 4 );
14,527 //if (CounterOccurrences<999999999) CounterOccurrences++; // Starting from ZERO! Because when insertion happened there was no setting counter to 1.
14,528 // Counter ]
14,529 WORDcountBOTTOM++; //Nakamichi
14,530 WORDcountBOTTOMPerMatchLen++;
14,531 if (*argv[k_FIX] == 'Y' || *argv[k_FIX] == 'Z')
14,532 //if (CounterOccurrences>1) // For Nakamichi
14,533 // fprintf(fp_out, "%s\t%s\r\n", _ui64toaKAZEzerocomma(CounterOccurrences, 11ToaDigits2, 10)+(26-11), wrd); WORDcountBOTTOM++;
14,534 if (*argv[k_FIX] == 'y' || *argv[k_FIX] == 'z')
14,535 // NO-DUMP: [
14,536 WORDcountBOTTOM++;
14,537 // fprintf(fp_out, "%s\r\n", wrd); WORDcountBOTTOM++;
14,538 // NO-DUMP: ]
14,539 //fwrite(CBdLFa, 2, 1, fp_out);
14,540 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
14,541 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
14,542 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,543 //fread(&SomeByte, 1, 1, fp_outRG);
14,544 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
14,545 // [ //r.14+
14,546 memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (KeySize+1+8)], 1 );
14,547 // ] //r.14+
14,548 if (SomeByte != 0) // RW exists
14,549 {
14,550 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!

```

```

14,551 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
14,552 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
14,553 //fread(&wrd[0], (LongestLineInclusive+1+4), 1, fp_outRG);
14,554 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
14,555 // [ //r.14+
14,556 memcpy( &wrd[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
14,557 // ] //r.14+
14,558 // Counter [
14,559 //memcpy( &CounterOccurrences, &wrd[(KeySize+1+4)-4], 4 );
14,560 //if (CounterOccurrences<999999999) CounterOccurrences++; // Starting from ZERO! Because when insertion happened there was no setting counter to 1.
14,561 // Counter ]
14,562 WORDcountBOTTOM++; //Nakamichi
14,563 WORDcountBOTTOMPerMatchLen++;
14,564 if (*argv[k_FIX] == 'Y' || *argv[k_FIX] == 'Z')
14,565 //if (CounterOccurrences>1) // For Nakamichi
14,566 // fprintf(fp_out, "%s\\t%s\\r\\n", _ui64toaKAZEcomma(CounterOccurrences, 11ToaDigits2, 10)+(26-11), wrd); WORDcountBOTTOM++;
14,567 if (*argv[k_FIX] == 'y' || *argv[k_FIX] == 'z')
14,568 // NO-DUMP: [
14,569 WORDcountBOTTOM++;
14,570 // fprintf(fp_out, "%s\\r\\n", wrd); WORDcountBOTTOM++;
14,571 // NO-DUMP: ]
14,572 //fwrite(CRdLFa, 2, 1, fp_out);
14,573 }
14,574 PseudoLinkedPointer_64 = 0;
14,575 NumberOfLEAFs++; //r.14
14,576 }
14,577 }
14,578 // $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ B-tree order 3 traverse 64bit ]
14,579
14,580 } //if (PseudoLinkedPointer_64 != 0)
14,581 // Stats for one tree only:
14,582 //printf( "\\nLeprechaun: Total keys (j=%d) into B-trees order 3 (during traversal/dump): %s\\n", j, _ui64toaKAZEcomma(WORDcountBOTTOMPerMatchLen, 11ToaDigits, 10) );
14,583
14,584 } //for( jjj = 0;
14,585 } //for( j = 0;
14,586
14,587 // The dump file with ripped data [
14,588 //fclose(fp_out);
14,589 // The dump file with ripped data ]
14,590
14,591 fprintf( fp_outLOG, "Number Of Hash Collisions(Distinct WORDs - Number Of Trees): %s\\n", _ui64toaKAZEcomma(WORDcountDistinct - NumberOfTrees, 11ToaDigits, 10) );
14,592 fprintf( fp_outLOG, "Number Of Trees(GREATER THE BETTER): %s\\n", _ui64toaKAZEcomma(NumberOfTrees, 11ToaDigits, 10) );
14,593 fprintf( fp_outLOG, "Number Of LEAFs(littler THE BETTER) not counting ROOT LEAFs: %s\\n", _ui64toaKAZEcomma(NumberOfLEAFs-NumberOfTrees, 11ToaDigits, 10) );
14,594 fprintf( fp_outLOG, "Highest Tree not counting ROOT Level i.e. CORONA levels(littler THE BETTER): %s\\n", _ui64toaKAZEcomma(LevelsInCorona_Not_Counting_ROOT-1, 11ToaDigits, 10) );
14,595
14,596 fprintf( fp_outLOG, "Used value for third parameter in KB: %s\\n", _ui64toaKAZEcomma(Thunderwith, 11ToaDigits, 10) );
14,597 fprintf( fp_outLOG, "Use next time as third parameter: %s\\n", _ui64toaKAZEcomma((((BufEnd_64-(unsigned long long)pointerflush_64)+1)>>10)+1, 11ToaDigits, 10) );
14,598 fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11ToaDigits, 10) );
14,599
14,600 // DUMP and STATS ]]]
14,601
14,602 printf( "\\n");
14,603 printf( "Leprechaun: Number Of Total/Distinct/Undistinct keys/BBs (all orders): %s/%s/%s\\n", _ui64toaKAZEcomma(WORDcount, 11ToaDigits, 10),
14,604 _ui64toaKAZEcomma(WORDcountDistinct, 11ToaDigits2, 10), _ui64toaKAZEcomma(WORDcount-WORDcountDistinct, 11ToaDigits3, 10) );
14,605 printf( "Leprechaun: Number Of TREES(GREATER THE BETTER): %s\\n", _ui64toaKAZEcomma(NumberOfTrees, 11ToaDigits, 10) );
14,606 printf( "Leprechaun: Number Of LEAFs(littler THE BETTER) not counting ROOT LEAFs: %s\\n", _ui64toaKAZEcomma(NumberOfLEAFs-NumberOfTrees, 11ToaDigits, 10) );
14,607 printf( "Leprechaun: Highest Tree not counting ROOT Level i.e. CORONA levels(littler THE BETTER): %s\\n", _ui64toaKAZEcomma(LevelsInCorona_Not_Counting_ROOT-1,

```

```

11TOaDigits, 10) );
14,607 printf( "Leprechaun: Used value for B-trees pool in KB: %s\n", _ui64toaKAZEcomma(Thunderwith, 11TOaDigits, 10) );
14,608 printf( "Leprechaun: Use next time for B-trees pool in KB: %s\n", _ui64toaKAZEcomma(((BufEnd_64-(unsigned long long)pointerflush_64)+1)>>10)+1, 11TOaDigits, 10) );
14,609 // printf( "Leprechaun: Total Attempts to Find/Put keys into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11TOaDigits, 10) );
14,610 printf( "Leprechaun: Total keys (all orders) into B-trees order 3 (during traversal/dump): %s\n", _ui64toaKAZEcomma(WORDcountBOTTOM, 11TOaDigits, 10) );
14,611 printf( "Leprechaun: Total keys MISSES/HITS (all orders) into B-trees order 3: %s/%s\n", _ui64toaKAZEcomma(NotFoundKeys, 11TOaDigits, 10),
_ui64toaKAZEcomma(FoundKeys, 11TOaDigits2, 10) );
14,612 printf( "\n");
14,613 // Stats... ]
14,614
14,615 printf( "Leprechaun: B-trees building speed (counting ROOT LEAFs): %s LEAFs/s\n", _ui64toaKAZEcomma((double)(NumberOfLEAFs)/(double)(time2 - time1 + 1), 11TOaDigits3,
10));
14,616 printf( "Leprechaun: B-trees traverse speed (counting ROOT LEAFs): %s LEAFs/s\n", _ui64toaKAZEcomma((double)(NumberOfLEAFs)/(double)(time(NULL) - time2 + 1),
11TOaDigits3, 10));
14,617 SkipDestroyingTheTrees:
14,618 printf( "Leprechaun: Total Searches-n-Inserts Per Second: %s SNIPS\n", _ui64toaKAZEcomma((double)(WORDcount)/(double)(time(NULL) - time1 + 1), 11TOaDigits3, 10));
14,619 printf( "Leprechaun: RAM needed to house B-trees (relative to the file being ripped): %sN = %sMB\n", _ui64toaKAZEcomma((BufEnd_64-(unsigned long
long)pointerflush_64+1)/size_inLINESIXFOUR, 11TOaDigits3, 10), _ui64toaKAZEcomma(1+((BufEnd_64-(unsigned long long)pointerflush_64+1)>>20), 11TOaDigits2, 10));
14,620 //if defined(ExternalRAM)
14,621 if (BSTorBTree == 2) {
14,622 printf( "Leprechaun: Total IOPS for %s 'freads' and %s 'fwrites' (of packets %d bytes long) during loading traversing all orders: %s IOPS\n",
_ui64toaKAZEcomma(Total_fread, 11TOaDigits, 10), _ui64toaKAZEcomma(Total_fwrite, 11TOaDigits2, 10), 8+8+8+2*(LongestLineInclusive+1+8),
_ui64toaKAZEcomma((double)(Total_fwrite+Total_fread)/(double)(time(NULL) - time1 + 1), 11TOaDigits3, 10));
14,623 }
14,624 //endif
14,625 printf( "\n");
14,626
14,627 //exit(0);
14,628
14,629 } // b_tree]
14,630
14,631 // Leprechaun BBhex ]
14,632
14,633
14,634 // https://github.com/XKCP/XKCP/blob/master/Standalone/CompactFIPS202/C/Keccak-more-compact.c
14,635 /*
14,636 #define FOR(i,n) for(i=0; i<n; ++i)
14,637 typedef unsigned char u8;
14,638 typedef unsigned long long int u64;
14,639 typedef unsigned int ui;
14,640
14,641 void Keccak(ui r, ui c, const u8 *in, u64 inLen, u8 sfx, u8 *out, u64 outLen);
14,642 void FIPS202_SHAKE128(const u8 *in, u64 inLen, u8 *out, u64 outLen) { Keccak(1344, 256, in, inLen, 0x1F, out, outLen); }
14,643 void FIPS202_SHAKE256(const u8 *in, u64 inLen, u8 *out, u64 outLen) { Keccak(1088, 512, in, inLen, 0x1F, out, outLen); }
14,644 void FIPS202_SHA3_224(const u8 *in, u64 inLen, u8 *out) { Keccak(1152, 448, in, inLen, 0x06, out, 28); }
14,645 void FIPS202_SHA3_256(const u8 *in, u64 inLen, u8 *out) { Keccak(1088, 512, in, inLen, 0x06, out, 32); }
14,646 void FIPS202_SHA3_384(const u8 *in, u64 inLen, u8 *out) { Keccak(832, 768, in, inLen, 0x06, out, 48); }
14,647 void FIPS202_SHA3_512(const u8 *in, u64 inLen, u8 *out) { Keccak(576, 1024, in, inLen, 0x06, out, 64); }
14,648
14,649 int LFSR86540(u8 *R) { (*R)=((*R)<<1)^(((*R)&0x80)?0x71:0); return ((*R)&2)>>1; }
14,650 #define ROL(a,o) (((u64)a)<<o)^(((u64)a)>>(64-o)))
14,651 static u64 load64(const u8 *x) { ui i; u64 u=0; FOR(i,8) { u<<=8; u|=x[7-i]; } return u; }
14,652 static void store64(u8 *x, u64 u) { ui i; FOR(i,8) { x[i]=u; u>>=8; } }
14,653 static void xor64(u8 *x, u64 u) { ui i; FOR(i,8) { x[i]^=u; u>>=8; } }
14,654 #define rL(x,y) load64((u8*)s+8*(x+5*y))
14,655 #define wL(x,y,l) store64((u8*)s+8*(x+5*y),l)
14,656 #define XL(x,y,l) xor64((u8*)s+8*(x+5*y),l)
14,657 void KeccakF1600(void *s)

```

```

14,658 {
14,659     ui r,x,y,i,j,Y; u8 R=0x01; u64 C[5],D;
14,660     for(i=0; i<24; i++) {
14,661         FOR(x,5) C[x]=rL(x,0)^rL(x,1)^rL(x,2)^rL(x,3)^rL(x,4); FOR(x,5) { D=C[(x+4)%5]^ROL(C[(x+1)%5],1); FOR(y,5) XL(x,y,D); }
14,662         x=1; y=r=0; D=rL(x,y); FOR(j,24) { r+=j+1; Y=(2*x+3*y)%5; x=y; y=Y; C[0]=rL(x,y); wL(x,y,ROL(D,r%64)); D=C[0]; }
14,663         FOR(y,5) { FOR(x,5) C[x]=rL(x,y); FOR(x,5) wL(x,y,C[x]^("C[(x+1)%5]&C[(x+2)%5])); }
14,664         FOR(j,7) if (LFSR86540(&R)) XL(0,0,(u64)1<<((1<j)-1));
14,665     }
14,666 }
14,667 void Keccak(ui r, ui c, const u8 *in, u64 inLen, u8 sfx, u8 *out, u64 outLen)
14,668 {
14,669     u8 s[200]; ui R=r/8; ui i,b=0; FOR(i,200) s[i]=0;
14,670     while(inLen>0) { b=(inLen<R)?inLen:R; FOR(i,b) s[i]^=in[i]; in+=b; inLen-=b; if (b==R) { KeccakF1600(s); b=0; } }
14,671     s[b]^=sfx; if((sfx&0x80)&&(b==(R-1))) KeccakF1600(s); s[R-1]^=0x80; KeccakF1600(s);
14,672     while(outLen>0) { b=(outLen<R)?outLen:R; FOR(i,b) out[i]=s[i]; out+=b; outLen-=b; if(outLen>0) KeccakF1600(s); }
14,673 }
14,674 */
14,675
14,676 // SHA3 [[[
14,677
14,678 /*
14,679 Implementation by the Keccak, Keyak and Ketje Teams, namely, Guido Bertoni,
14,680 Joan Daemen, Michaël Peeters, Gilles Van Assche and Ronny Van Keer, hereby
14,681 denoted as "the implementer".
14,682
14,683 For more information, feedback or questions, please refer to our websites:
14,684 http://keccak.noekeon.org/
14,685 http://keyak.noekeon.org/
14,686 http://ketje.noekeon.org/
14,687
14,688 To the extent possible under law, the implementer has waived all copyright
14,689 and related or neighboring rights to the source code in this file.
14,690 http://creativecommons.org/publicdomain/zero/1.0/
14,691 */
14,692
14,693 /*
14,694 =====
14,695 The purpose of this source file is to demonstrate a readable and compact
14,696 implementation of all the Keccak instances approved in the FIPS 202 standard,
14,697 including the hash functions and the extendable-output functions (XOFs).
14,698
14,699 We focused on clarity and on source-code compactness,
14,700 rather than on the performance.
14,701
14,702 The advantages of this implementation are:
14,703 + The source code is compact, after removing the comments, that is. :-)
14,704 + There are no tables with arbitrary constants.
14,705 + For clarity, the comments link the operations to the specifications using
14,706   the same notation as much as possible.
14,707 + There is no restriction in cryptographic features. In particular,
14,708   the SHAKE128 and SHAKE256 XOFs can produce any output length.
14,709 + The code does not use much RAM, as all operations are done in place.
14,710
14,711 The drawbacks of this implementation are:
14,712 - There is no message queue. The whole message must be ready in a buffer.
14,713 - It is not optimized for performance.
14,714
14,715 The implementation is even simpler on a little endian platform. Just define the

```

```
14,716 LITTLE_ENDIAN symbol in that case.
14,717
14,718 For a more complete set of implementations, please refer to
14,719 the Keccak Code Package at https://github.com/gvanas/KeccakCodePackage
14,720
14,721 For more information, please refer to:
14,722 * [Keccak Reference] http://keccak.nokeon.org/Keccak-reference-3.0.pdf
14,723 * [Keccak Specifications Summary] http://keccak.nokeon.org/specs\_summary.html
14,724
14,725 This file uses UTF-8 encoding, as some comments use Greek letters.
14,726 =====
14,727 */
14,728
14,729 //void Keccak(unsigned int rate, unsigned int capacity, const unsigned char *input, unsigned long long int inputByteLen, unsigned char delimitedSuffix, unsigned char
*output, unsigned long long int outputByteLen);
14,730
14,731 // #include "sha3.h"
14,732 // sha3.h [
14,733 #define SHA3_256_DIGEST_LEN 32
14,734 #define SHA3_512_DIGEST_LEN 64
14,735 #define SHA3_256_BLOCK_LEN 136
14,736 #define SHA3_512_BLOCK_LEN 72
14,737 #define SHA3_256_B64_LEN 43
14,738 #define SHA3_512_B64_LEN 86
14,739 #define SHA3_256_DIGEST_STR_LEN (SHA3_256_DIGEST_LEN * 2 + 1)
14,740 #define SHA3_512_DIGEST_STR_LEN (SHA3_512_DIGEST_LEN * 2 + 1)
14,741 void Keccak(unsigned int rate, unsigned int capacity, const unsigned char *input, unsigned long long int inputByteLen, unsigned char delimitedSuffix, unsigned char
*output, unsigned long long int outputByteLen);
14,742 void FIPS202_SHAKE128(const unsigned char *input, unsigned int inputByteLen, unsigned char *output, int outputByteLen);
14,743 void FIPS202_SHAKE256(const unsigned char *input, unsigned int inputByteLen, unsigned char *output, int outputByteLen);
14,744 void FIPS202_SHA3_224(const unsigned char *input, unsigned int inputByteLen, unsigned char *output);
14,745 void FIPS202_SHA3_256(const unsigned char *input, unsigned int inputByteLen, unsigned char *output);
14,746 void FIPS202_SHA3_384(const unsigned char *input, unsigned int inputByteLen, unsigned char *output);
14,747 void FIPS202_SHA3_512(const unsigned char *input, unsigned int inputByteLen, unsigned char *output);
14,748 // sha3.h ]
14,749
14,750 void FIPS202_SHAKE128(const unsigned char *input, unsigned int inputByteLen, unsigned char *output, int outputByteLen)
14,751 {
14,752     Keccak(1344, 256, input, inputByteLen, 0x1F, output, outputByteLen);
14,753 }
14,754
14,755 void FIPS202_SHAKE256(const unsigned char *input, unsigned int inputByteLen, unsigned char *output, int outputByteLen)
14,756 {
14,757     Keccak(1088, 512, input, inputByteLen, 0x1F, output, outputByteLen);
14,758 }
14,759
14,760 void FIPS202_SHA3_224(const unsigned char *input, unsigned int inputByteLen, unsigned char *output)
14,761 {
14,762     Keccak(1152, 448, input, inputByteLen, 0x06, output, 28);
14,763 }
14,764
14,765 void FIPS202_SHA3_256(const unsigned char *input, unsigned int inputByteLen, unsigned char *output)
14,766 {
14,767     Keccak(1088, 512, input, inputByteLen, 0x06, output, 32);
14,768 }
14,769
14,770 void FIPS202_SHA3_384(const unsigned char *input, unsigned int inputByteLen, unsigned char *output)
14,771 {
```

```
14,772     Keccak(832, 768, input, inputByteLen, 0x06, output, 48);
14,773 }
14,774
14,775 void FIPS202_SHA3_512(const unsigned char *input, unsigned int inputByteLen, unsigned char *output)
14,776 {
14,777     Keccak(576, 1024, input, inputByteLen, 0x06, output, 64);
14,778 }
14,779
14,780 /*
14,781 =====
14,782 Technicalities
14,783 =====
14,784 */
14,785
14,786 typedef unsigned char UINT8;
14,787 typedef unsigned long long int UINT64;
14,788 typedef UINT64 tKeccakLane;
14,789
14,790 #ifndef LITTLE_ENDIAN
14,791
14,792 static UINT64 load64(const UINT8 *x)
14,793 {
14,794     int i;
14,795     UINT64 u=0;
14,796
14,797     for(i=7; i>=0; --i) {
14,798         u <<= 8;
14,799         u |= x[i];
14,800     }
14,801     return u;
14,802 }
14,803
14,804 static void store64(UINT8 *x, UINT64 u)
14,805 {
14,806     unsigned int i;
14,807
14,808     for(i=0; i<8; ++i) {
14,809         x[i] = u;
14,810         u >>= 8;
14,811     }
14,812 }
14,813
14,814 static void xor64(UINT8 *x, UINT64 u)
14,815 {
14,816     unsigned int i;
14,817
14,818     for(i=0; i<8; ++i) {
14,819         x[i] ^= u;
14,820         u >>= 8;
14,821     }
14,822 }
14,823 #endif
14,824
14,825 /*
14,826 =====
14,827 A readable and compact implementation of the Keccak-f[1600] permutation.
14,828 =====
14,829 */
```



```

14,830
14,831 #define ROL64(a, offset) (((UINT64)a) << offset) ^ (((UINT64)a) >> (64-offset)))
14,832 #define i(x, y) ((x)+5*(y))
14,833
14,834 #ifdef LITTLE_ENDIAN
14,835     #define readLane(x, y)      (((tKeccakLane*)state)[i(x, y)])
14,836     #define writeLane(x, y, lane) (((tKeccakLane*)state)[i(x, y)]) = (lane)
14,837     #define XORLane(x, y, lane) (((tKeccakLane*)state)[i(x, y)]) ^= (lane)
14,838 #else
14,839     #define readLane(x, y)      load64((UINT8*)state+sizeof(tKeccakLane)*i(x, y))
14,840     #define writeLane(x, y, lane) store64((UINT8*)state+sizeof(tKeccakLane)*i(x, y), lane)
14,841     #define XORLane(x, y, lane) xor64((UINT8*)state+sizeof(tKeccakLane)*i(x, y), lane)
14,842 #endif
14,843
14,844 int LFSR86540(UINT8 *LFSR)
14,845 {
14,846     int result = ((*LFSR) & 0x01) != 0;
14,847     if (((*LFSR) & 0x80) != 0)
14,848         // Primitive polynomial over GF(2): x^8+x^6+x^5+x^4+1
14,849         (*LFSR) = ((*LFSR) << 1) ^ 0x71;
14,850     else
14,851         (*LFSR) <<= 1;
14,852     return result;
14,853 }
14,854
14,855 void KeccakF1600_StatePermute(void *state)
14,856 {
14,857     unsigned int round, x, y, j, t;
14,858     UINT8 LFSRstate = 0x01;
14,859
14,860     for(round=0; round<24; round++) {
14,861         // == 8 step (see [Keccak Reference, Section 2.3.2]) ==
14,862         tKeccakLane C[5], D;
14,863
14,864         // Compute the parity of the columns
14,865         for(x=0; x<5; x++)
14,866             C[x] = readLane(x, 0) ^ readLane(x, 1) ^ readLane(x, 2) ^ readLane(x, 3) ^ readLane(x, 4);
14,867         for(x=0; x<5; x++) {
14,868             // Compute the 8 effect for a given column
14,869             D = C[(x+4)%5] ^ ROL64(C[(x+1)%5], 1);
14,870             // Add the 8 effect to the whole column
14,871             for (y=0; y<5; y++)
14,872                 XORLane(x, y, D);
14,873         }
14,874     }
14,875
14,876     { // == p and π steps (see [Keccak Reference, Sections 2.3.3 and 2.3.4]) ==
14,877         tKeccakLane current, temp;
14,878         // Start at coordinates (1 0)
14,879         x = 1; y = 0;
14,880         current = readLane(x, y);
14,881         // Iterate over ((0 1)(2 3))^t * (1 0) for 0 ≤ t ≤ 23
14,882         for(t=0; t<24; t++) {
14,883             // Compute the rotation constant r = (t+1)(t+2)/2
14,884             unsigned int r = ((t+1)*(t+2)/2)%64;
14,885             // Compute ((0 1)(2 3)) * (x y)
14,886             unsigned int Y = (2*x+3*y)%5; x = y; y = Y;
14,887             // Swap current and state(x,y), and rotate

```

```

14,888         temp = readLane(x, y);
14,889         writeLane(x, y, ROL64(current, r));
14,890         current = temp;
14,891     }
14,892 }
14,893
14,894 { // ===  $\chi$  step (see [Keccak Reference, Section 2.3.1]) ===
14,895     tKeccakLane temp[5];
14,896     for(y=0; y<5; y++) {
14,897         // Take a copy of the plane
14,898         for(x=0; x<5; x++)
14,899             temp[x] = readLane(x, y);
14,900         // Compute  $\chi$  on the plane
14,901         for(x=0; x<5; x++)
14,902             writeLane(x, y, temp[x] ^((~temp[(x+1)%5]) & temp[(x+2)%5]));
14,903     }
14,904 }
14,905
14,906 { // ===  $\iota$  step (see [Keccak Reference, Section 2.3.5]) ===
14,907     for(j=0; j<7; j++) {
14,908         unsigned int bitPosition = (1<<j)-1; //2^j-1
14,909         if (LFSR86540(&LFSRstate))
14,910             XORLane(0, 0, (tKeccakLane)1<<bitPosition);
14,911     }
14,912 }
14,913 }
14,914 }
14,915
14,916 /*
14,917 =====
14,918 A readable and compact implementation of the Keccak sponge functions
14,919 that use the Keccak-f[1600] permutation.
14,920 =====
14,921 */
14,922
14,923 #include <string.h>
14,924 #define MIN(a, b) ((a) < (b) ? (a) : (b))
14,925
14,926 void Keccak(unsigned int rate, unsigned int capacity, const unsigned char *input, unsigned long long int inputByteLen, unsigned char delimitedSuffix, unsigned char
14,927 *output, unsigned long long int outputByteLen)
14,928 {
14,929     UINT8 state[200];
14,930     unsigned int rateInBytes = rate/8;
14,931     unsigned int blockSize = 0;
14,932     unsigned int i;
14,933
14,934     if (((rate + capacity) != 1600) || ((rate % 8) != 0))
14,935         return;
14,936
14,937     // === Initialize the state ===
14,938     memset(state, 0, sizeof(state));
14,939
14,940     // === Absorb all the input blocks ===
14,941     while(inputByteLen > 0) {
14,942         blockSize = MIN(inputByteLen, rateInBytes);
14,943         for(i=0; i<blockSize; i++)
14,944             state[i] ^= input[i];
14,945         input += blockSize;

```

```
14,945         inputByteLen -= blockSize;
14,946
14,947         if (blockSize == rateInBytes) {
14,948             KeccakF1600_StatePermute(state);
14,949             blockSize = 0;
14,950         }
14,951     }
14,952
14,953     // === Do the padding and switch to the squeezing phase ===
14,954     // Absorb the last few bits and add the first bit of padding (which coincides with the delimiter in delimitedSuffix)
14,955     state[blockSize] ^= delimitedSuffix;
14,956     // If the first bit of padding is at position rate-1, we need a whole new block for the second bit of padding
14,957     if (((delimitedSuffix & 0x80) != 0) && (blockSize == (rateInBytes-1)))
14,958         KeccakF1600_StatePermute(state);
14,959     // Add the second bit of padding
14,960     state[rateInBytes-1] ^= 0x80;
14,961     // Switch to the squeezing phase
14,962     KeccakF1600_StatePermute(state);
14,963
14,964     // === Squeeze out all the output blocks ===
14,965     while(outputByteLen > 0) {
14,966         blockSize = MIN(outputByteLen, rateInBytes);
14,967         memcpy(output, state, blockSize);
14,968         output += blockSize;
14,969         outputByteLen -= blockSize;
14,970
14,971         if (outputByteLen > 0)
14,972             KeccakF1600_StatePermute(state);
14,973     }
14,974 }
14,975
14,976 // SHA3 ]]]
14,977
14,978 // MAIN[
14,979
14,980 int main( int argc, char *argv[] ) {
14,981     FILE *fp;
14,982     FILE *fp_outLOG;
14,983     uint32_t Filehash;
14,984     //int SourceSize;
14,985     //int TargetSize;
14,986     uint64_t SourceSize;
14,987     uint64_t TargetSize;
14,988     uint64_t TargetSize1;
14,989     uint64_t TargetSize2;
14,990     uint64_t VerifySize;
14,991     uint64_t DoOffset;
14,992     char* SourceBlock=NULL;
14,993     char* SourceBlockREVERSED=NULL;
14,994     char* TargetBlock=NULL;
14,995     char* VerifyBlock=NULL;
14,996     char* Nakamichi = ".Nakamichi\0";
14,997     char* LZSSE      = ".L17.LZSSE2\0";
14,998     char NewFileName[256];
14,999     char NewFileName2[256];
15,000     // clock_t clocks1, clocks2;
15,001     uint64_t OneMB=1024*1024;
15,002     //uint64_t SevenGB = OneMB*5120;
```

```
15,003 uint64_t SevenGB = OneMB*MAXsizeDecompressionRAMpoolInMB; // 2021-Jun-23
15,004 unsigned long long ticksTOTAL=0, ticksStart;
15,005
15,006 char* NakamichiSHA3 = "Nakamichi";
15,007 char* NakamichiSHA3q = "q";
15,008
15,009 char *pointerALIGN;
15,010 int i, j;
15,011 uint64_t i64; // 2020-Feb-26
15,012 clock_t clocks3, clocks4;
15,013 double duration;
15,014 double durationGENERIC;
15,015 int BandwidthFlag=0;
15,016
15,017 unsigned long long k;
15,018 unsigned long long k1;
15,019 unsigned long long kMAXIMUM;
15,020 unsigned long long k2;
15,021 unsigned long long k3;
15,022 int Trials;
15,023
15,024 // LZSSE2 [
15,025 #ifdef _N_alone
15,026 #else
15,027 LZSSE2_OptimalParseState *s;
15,028 #endif
15,029 // LZSSE2 ]
15,030
15,031 #if defined(_WIN32_ENVIRONMENT_)
15,032     unsigned long long size_inLINESIXFOUR;
15,033     unsigned long long size_inLINESIXFOURlog;
15,034 #else
15,035     size_t size_inLINESIXFOUR;
15,036     size_t size_inLINESIXFOURlog;
15,037 #endif /* defined(_WIN32_ENVIRONMENT_) */
15,038
15,039 #ifdef _N_HIGH_PRIORITY
15,040     DWORD dwError, dwPriClass;
15,041 #endif
15,042
15,043 char *TwoDigitHEXlist[256] = {
15,044 "00\0",
15,045 "01\0",
15,046 "02\0",
15,047 "03\0",
15,048 "04\0",
15,049 "05\0",
15,050 "06\0",
15,051 "07\0",
15,052 "08\0",
15,053 "09\0",
15,054 "0A\0",
15,055 "0B\0",
15,056 "0C\0",
15,057 "0D\0",
15,058 "0E\0",
15,059 "0F\0",
15,060 "10\0",
```

15,061 "11\0",
15,062 "12\0",
15,063 "13\0",
15,064 "14\0",
15,065 "15\0",
15,066 "16\0",
15,067 "17\0",
15,068 "18\0",
15,069 "19\0",
15,070 "1A\0",
15,071 "1E\0",
15,072 "1C\0",
15,073 "1D\0",
15,074 "1E\0",
15,075 "1F\0",
15,076 "20\0",
15,077 "21\0",
15,078 "22\0",
15,079 "23\0",
15,080 "24\0",
15,081 "25\0",
15,082 "26\0",
15,083 "27\0",
15,084 "28\0",
15,085 "29\0",
15,086 "2A\0",
15,087 "2B\0",
15,088 "2C\0",
15,089 "2D\0",
15,090 "2E\0",
15,091 "2F\0",
15,092 "30\0",
15,093 "31\0",
15,094 "32\0",
15,095 "33\0",
15,096 "34\0",
15,097 "35\0",
15,098 "36\0",
15,099 "37\0",
15,100 "38\0",
15,101 "39\0",
15,102 "3A\0",
15,103 "3B\0",
15,104 "3C\0",
15,105 "3D\0",
15,106 "3E\0",
15,107 "3F\0",
15,108 "40\0",
15,109 "41\0",
15,110 "42\0",
15,111 "43\0",
15,112 "44\0",
15,113 "45\0",
15,114 "46\0",
15,115 "47\0",
15,116 "48\0",
15,117 "49\0",
15,118 "4A\0",

15,119 "4B\0",
15,120 "4C\0",
15,121 "4D\0",
15,122 "4E\0",
15,123 "4F\0",
15,124 "50\0",
15,125 "51\0",
15,126 "52\0",
15,127 "53\0",
15,128 "54\0",
15,129 "55\0",
15,130 "56\0",
15,131 "57\0",
15,132 "58\0",
15,133 "59\0",
15,134 "5A\0",
15,135 "5B\0",
15,136 "5C\0",
15,137 "5D\0",
15,138 "5E\0",
15,139 "5F\0",
15,140 "60\0",
15,141 "61\0",
15,142 "62\0",
15,143 "63\0",
15,144 "64\0",
15,145 "65\0",
15,146 "66\0",
15,147 "67\0",
15,148 "68\0",
15,149 "69\0",
15,150 "6A\0",
15,151 "6B\0",
15,152 "6C\0",
15,153 "6D\0",
15,154 "6E\0",
15,155 "6F\0",
15,156 "70\0",
15,157 "71\0",
15,158 "72\0",
15,159 "73\0",
15,160 "74\0",
15,161 "75\0",
15,162 "76\0",
15,163 "77\0",
15,164 "78\0",
15,165 "79\0",
15,166 "7A\0",
15,167 "7B\0",
15,168 "7C\0",
15,169 "7D\0",
15,170 "7E\0",
15,171 "7F\0",
15,172 "80\0",
15,173 "81\0",
15,174 "82\0",
15,175 "83\0",
15,176 "84\0",

15,177 "85\0",
15,178 "86\0",
15,179 "87\0",
15,180 "88\0",
15,181 "89\0",
15,182 "8A\0",
15,183 "8B\0",
15,184 "8C\0",
15,185 "8D\0",
15,186 "8E\0",
15,187 "8F\0",
15,188 "90\0",
15,189 "91\0",
15,190 "92\0",
15,191 "93\0",
15,192 "94\0",
15,193 "95\0",
15,194 "96\0",
15,195 "97\0",
15,196 "98\0",
15,197 "99\0",
15,198 "9A\0",
15,199 "9B\0",
15,200 "9C\0",
15,201 "9D\0",
15,202 "9E\0",
15,203 "9F\0",
15,204 "A0\0",
15,205 "A1\0",
15,206 "A2\0",
15,207 "A3\0",
15,208 "A4\0",
15,209 "A5\0",
15,210 "A6\0",
15,211 "A7\0",
15,212 "A8\0",
15,213 "A9\0",
15,214 "AA\0",
15,215 "AB\0",
15,216 "AC\0",
15,217 "AD\0",
15,218 "AE\0",
15,219 "AF\0",
15,220 "B0\0",
15,221 "B1\0",
15,222 "B2\0",
15,223 "B3\0",
15,224 "B4\0",
15,225 "B5\0",
15,226 "B6\0",
15,227 "B7\0",
15,228 "B8\0",
15,229 "B9\0",
15,230 "BA\0",
15,231 "BB\0",
15,232 "BC\0",
15,233 "BD\0",
15,234 "BE\0",

15,235 "BF\0",
15,236 "C0\0",
15,237 "C1\0",
15,238 "C2\0",
15,239 "C3\0",
15,240 "C4\0",
15,241 "C5\0",
15,242 "C6\0",
15,243 "C7\0",
15,244 "C8\0",
15,245 "C9\0",
15,246 "CA\0",
15,247 "CB\0",
15,248 "CC\0",
15,249 "CD\0",
15,250 "CE\0",
15,251 "CF\0",
15,252 "D0\0",
15,253 "D1\0",
15,254 "D2\0",
15,255 "D3\0",
15,256 "D4\0",
15,257 "D5\0",
15,258 "D6\0",
15,259 "D7\0",
15,260 "D8\0",
15,261 "D9\0",
15,262 "DA\0",
15,263 "DB\0",
15,264 "DC\0",
15,265 "DD\0",
15,266 "DE\0",
15,267 "DF\0",
15,268 "E0\0",
15,269 "E1\0",
15,270 "E2\0",
15,271 "E3\0",
15,272 "E4\0",
15,273 "E5\0",
15,274 "E6\0",
15,275 "E7\0",
15,276 "E8\0",
15,277 "E9\0",
15,278 "EA\0",
15,279 "EB\0",
15,280 "EC\0",
15,281 "ED\0",
15,282 "EE\0",
15,283 "EF\0",
15,284 "F0\0",
15,285 "F1\0",
15,286 "F2\0",
15,287 "F3\0",
15,288 "F4\0",
15,289 "F5\0",
15,290 "F6\0",
15,291 "F7\0",
15,292 "F8\0",


```
15,293 "F9\0",
15,294 "FA\0",
15,295 "FB\0",
15,296 "FC\0",
15,297 "FD\0",
15,298 "FE\0",
15,299 "FF\0"
15,300 };
15,301
15,302 if (argc == 2) {} else { // 2020-Jan-13
15,303     printf("
15,304     printf("
15,305     printf("
15,306     printf("
15,307     printf("
15,308     printf("
15,309     printf("
15,310     printf("
15,311     printf("
15,312     printf("
15,313     printf("
15,314     printf("
15,315     printf("
15,316     printf("
15,317     printf("
15,318     printf("
15,319     printf("
15,320     printf("
15,321     printf("
15,322     printf("
15,323     printf("
15,324     printf("
15,325     printf("
15,326     printf("
15,327     printf("
15,328     printf("
15,329     printf("
15,330     printf("
15,331     printf("
15,332     printf("
15,333     printf("
15,334     printf("
15,335
15,336 #ifdef Kaidanji
15,337     printf("Nakamichi 'Kaidanji', written by Kaze, inspired by Haruhiko Okumura sharing, based on Nobuo Ito's LZSS source, muffinesque tips by m^2, Jim Dempsey and Kirill
Kryukov enforced.\n");
15,338     printf("This latest (2021-Aug-28) compile has B-trees only matchfinder - called 'SUPRALITE'; Downloadable at: https://twitter.com/Sanmayce\n");
15,339 #else
15,340     printf("Nakamichi 'Ryuugan-ditto-1TB', written by Kaze, inspired by Haruhiko Okumura sharing, based on Nobuo Ito's LZSS source, muffinesque tips by m^2, Jim Dempsey
and Kirill Kryukov enforced.\n");
15,341     printf("This latest (2021-Aug-28) compile has B-trees only matchfinder - called 'LITE'; Downloadable at: https://twitter.com/Sanmayce\n");
15,342 #endif
15,343 //     printf("https://twitter.com/Sanmayce/status/1091780310407233536\n");
15,344     printf("\n");
15,345     printf("Nakamichi 'Kaidanji' SUPRALITE highlights:\n");
15,346     printf("\n");
15,347     printf("- Back to supersimplisticism - 'Kaidanji' returns refined. Simply, ripping only orders 4,6,8,10,12,14,16,18,36,64 and then quickly trying to expand the match
in the found position.\n");
15,348     printf("\n");
15,349     printf("Nakamichi 'Ryuugan-ditto-1TB'/'Dragoneye' LITE highlights:\n");
15,350     printf("\n");
15,351     printf("- The Zennish LZSS Microdeduplicator, a texttoy (file-to-file [de]compressor) written in plain C, 100% FREE - licenseless it is;\n");
Listing: Nakamichi_Ryuugan-ditto-1TB_btree.c; Last version: 2021-Aug-30; Font: MxPlus_ToshibaTxL2_8x16.ttf; Downloadable at: www.sanmayce.com/Nakamichi/Kaidanji.zip
```

```

15,352 printf("- Single-threaded Non-SIMD console tool written entirely by yours truly Sanmayce;\n");
15,353 printf("- The first Lempel-Ziv-Storer-Szymanski (LZSS) compressor using 1TB sliding window;\n");
15,354 printf("- It targets superfast decompression of huge/tera corpora of textual data;\n");
15,355 printf("\n");
15,356 printf("- LITE means no memmem() (Railgun) invocations are to be made, nah, back to minimalistic Railgun since 2021-Mar-12;\n");
15,357 printf("- Compressing at Linear Rate due to Matchfinding based on B-trees only;\n");
15,358 printf("- The Leprechaunesque (Internal/External) B-trees order 3 (2 keys MAX) are brutally-optimized;\n");
15,359 printf("\n");
15,360 printf("- Ability to deduplicate (as little as) 64 bytes long chunks 1TB backwards;\n");
15,361 printf("- Ability to deduplicate 256[+] bytes long chunks 1TB backwards - when SHA3 enabled;\n");
15,362 printf("\n");
15,363 printf("- CHUNKABLE! Since there are no headers/metadata in .Nakamichi files, arbitrary chunks/blocks can be compressed-n-concatenated;\n");
15,364 printf("\n");
15,365 printf("- SCALABLE! Gets faster when more Physical or/and External RAM is available;\n");
15,366 printf("- Dynamical (as command line parameters) selection of HASH & B-trees memory blocks;\n");
15,367 printf("- AUTO-RESETTING PASSES (when TargetBuffer is filled then automatically restart building by doubling the passes);\n");
15,368 printf("- DEFRAGMENTED B-trees are in use during compression stage, their leaves contain only BBs appearing 2[+] times;\n");
15,369 printf("\n");
15,370 printf("- Compileable under Windows and Linux, most suitable to run on 3TB machines;\n");
15,371 printf("- Many thanks go to: Haruhiko Okumura, Nobuo Ito, Hamid Buzidi, Landon Noll, J. Andrew Rogers, Aleksey Vaneev (https://github.com/avaneev/prvhash);\n");
15,372 printf("\n");
15,373 printf("Sanmayce, Nakamichi's author, here, some more notes:\n");
15,374 printf("\n");
15,375 printf("- The B-trees form the second layer, the first being HASH table handled by FNV1A-Pippip - the fastest lookup-hash;\n");
15,376 printf("- Hashpot a.k.a. hashpools (residing in Physical RAM) could be tuned via command line parameter, thus lessening the B-trees heights;\n");
15,377 printf("- The building of B-trees is done dynamically-n-automatically in PASSES, thus LOCALITY/LOCALIZATION leads to cache-friendliness, for example, instead of confusing/blinding\n");
15,378 printf("the SSD controller with building 2^27 ~= 128M B-trees at a time, 'PASSES' revision lowers the \"noise/mayhem\" 128 times by processing e.g. (20bit) 1M B-trees at a time;\n");
15,379 printf("\n");
15,380 printf("- For so long, the main goal was to throw it in battle, thus to forge one battle-ready *nix/windows tool with superb hash/search/decompress capabilities.\n");
15,381 printf("- Straight up, a wish of mine is to have the latest English Wikipedia XML dump decompressed by Nakamichi into physical RAM and then followed by Kazahana's superfast exact/wildcard/fuzzy etudes.\n");
15,382 printf("- Through the years, such a transparent decompressor (load booster) has been losing momentum and kinda became unnecessary, but not for me, when HDDs had 40MB/s sequential read speeds it was ...;\n");
15,383 printf("boosty, now with SSDs reading at 1700+MB/s, not so smacky, yet uploading (on 128GB machines, 20GB+72GB needed) the whole Wikipedia ~20GB (instead of 72GB) is better, sizewise.\n");
15,384 printf("\n");
15,385 printf("- Currently, having 16 (4,6,8,10,12,14,16,18,36,64,24,28,48,56,128,256) MatchLens being hashed (plus 1 being phantomic/temporary), 29bit hashtable results in: (16+1)*8*(2^29) = (16+1)*8*(1<<29) =\n");
15,386 printf("69632MB hashpot (the cluster of all hashpools), each slot houses a QWORD (being an offset/address of a B-tree root).\n");
15,387 printf("- In near future, wanna see a machine (256GB RAM) crunching the 77,265,813,300 bytes file with this command line (it requires 2x72GB + 68GB + 2930GB = 212GB physical RAM + 2.9TB SSD RAM);\n");
15,388 printf("D:\\>timer64 Nakamichi.exe enwiki-20200420-pages-articles.xml enwiki-20200420-pages-articles.xml.Nakamichi 29 3000000 e\n");
15,389 printf("- In not so far future, wanna see a machine (4096GB RAM) crunching the 72GB file with this command line (it requires 2x72GB + 272GB + 2930GB = 3.3TB physical RAM);\n");
15,390 printf("D:\\>timer64 Nakamichi.exe enwiki-20200420-pages-articles.xml enwiki-20200420-pages-articles.xml.Nakamichi 31 3000000 i\n");
15,391 printf("\n");
15,392 printf("Q: - HOW TO BOOST [BUILDING/COMPRESSION] SPEED?\n");
15,393 printf("A: - Now, the name of the game is \"RAM in spades\", if we have a lot then speed is going to jump a lot. How?\n");
15,394 printf("- First, by increasing the hash-pool - 20 (number of bits) was used in 'Taishukan' resulting in 2^20 ~= 1M hash-slots a.k.a. B-trees' roots per keysize, naturally, the bigger the faster since B-trees will\n");
15,395 printf("have less height and will be more - aiming to reach the ideal case - having only hash-slots with no actual B-trees! Back to 'Taishukan', the testfile is ~77MB, it suggests using 128M slots to lower\n");
15,396 printf("height/branching of B-trees - 2^27 leads to using 27. Feartime, going from 20 to 27: 8*(15+1)*2^(20) = 134,217,793 bytes; with the speed-up hash needs 8*(15+1)*2^(27) = 17,179,869,184 bytes, yes 16GB.\n");
15,397 printf("- Second, by lowering passes in building the B-trees - the trick is to increase the number in compile-time parameter -DSpeedUpBuilding=32 - (didn't make it as an execute-time parameter, lazy).\n");

```

```
15,398 printf(" If we look at the 'Taishukan' console log it tells us that (Target-Buffer 106 MB) x 64 passes = 6784MB are needed for building the B-trees in ONE PASS ONLY,
so compile with -DSpeedUpBuilding=6784 instead.\n");
15,399 printf("\n");
15,400 printf("Enfun!\n");
15,401 printf("\n");
15,402
15,403 /*
15,404 printf("Note0: Nakamichi 'Dragoneye', (AUTO-RESETTING PASSES and DEFRAGMENTED B-trees revision) is 100% FREE, licenseless that is.\n");
15,405 printf("Note1: Hamid Buzidi's LzTurbo ([a] FASTEST [Textual] Decompressor, Levels 19/29/39) retains kingship, his TurboBench (2017-Apr-07) proves the supremacy of
LzTurbo, Turbo-Amazing!\n");
15,406 printf("Note2: Conor Stokes' LZSSE2 ([a] FASTEST Textual Decompressor, Level 17) is embedded, all credits along with many thanks go to him.\n");
15,407 // printf("Note3: 'Ryuugan' predecessors are Washigan, Okamigan, Zato, Tsubame, Tengu-Tsuyo, Tengu, Rakka, Kokuen, Kinroba, Yoko, Kinutora, Jiten, Butsuhiro,
Suiken, Keigan, Kumataka, Washi, Aratama, Hitomi, Nekomata, Kitsune, Kinezumi, Sanbashi, Kaiko, Inazuma, Zangetsu, Hanabi, Hanazakari, Sanshi and Kaidanji.\n");
15,408 // printf("Note4: This variant is the SLOWEST compressor under the Sun! This compile can handle files up to 5120MB.\n");
15,409 printf("Note3: The matchfinder is either 'Railgun_Trolldom' (matches longer than 18, except 36 and 64) or Leprechaun's B-tree order 3.\n");
15,410 printf("Note4: Instead of 'mm_loadu_si128' 'mm_lddqu_si128' is used.\n");
15,411 // printf("Note7: The lookahead 'Tsuyo' heuristic which looks one char ahead is applied thrice, still not strengthened, though.\n");
15,412 // printf("Note8: The compile made 2017-Oct-22, the decompression time measuring is done in 16x8 passes choosing the top score from 64 back-to-back runs - the
goal - to enter [maximal] Turbo Mode.\n");
15,413 // printf("Note9: In GP/SSE4.1/AVX2 compile, the 24 matches become 3xQWORD/1xQWORD+1xXMMWORD/1xYMMWORD.\n");
15,414 printf("Note5: Maximum compression ratio is 28:1, for 256 bytes long matches within 1TB Sliding Window.\n");
15,415 printf("Note6: Please send me (at sanmayce@sanmayce.com) decompression results obtained on machines with fast CPU-RAM subsystems.\n");
15,416 printf("Note7: In this compile, clock() was replaced with time() - to counter bigtime stats misreporting.\n");
15,417 printf("Note8: Multi-way hashing allows each KeySize to occupy its own HASH pool, thus less RAM is in use - the LEAF is smaller.\n");
15,418 printf("Note9: In this revision, B-tree heuristics are in use, allowing skipping many unnecessary memmem() invocations.\n");
15,419 // printf("NoteF: In this compile, B-tree is the primary matchfinder, memmem() is used for 18+ matches (except 24,28,36,48,64), and for marginal cases.\n");
15,420 printf("NoteA: The file being compressed should be 4 bytes or longer due to Building-Blocks being in range 4..18, 24,28,36,48,56,64,128.\n");
15,421 printf("NoteB: In this compile, the key sizes in the LEAF are not HEXed i.e. not doubled. Also, the hash in use: FNV1A_Pippip_Yurii, the fastest one known to me.\n");
15,422 printf("NoteC: SHA3-224 is enabled at compile time, it is optional, that is. Currently only 256 long matches are SHA3fied.\n");
15,423 */
15,424 // printf("NoteH: The SHA3_224 hash for 'Nakamichi' should be: 6f6f29a1a07131340613af7eae0924af7cf605c6eb78f964b601fdf, is: ");
15,425 }
15,426
15,427 //FIPS202_SHA3_224((unsigned char *)NakamichiSHA3, 9, (unsigned char *)SHA328bytes);
15,428 //SHA3-224
15,429 //Nakamichi
15,430 //6f6f29a1a07131340613af7eae0924af7cf605c6eb78f964b601fdf
15,431 //
15,432 //SHA3-224
15,433 //q
15,434 //774057ce10e1b255cfa747982782e969231ef434a057622021ff5b9c
15,435 //for (i=0; i<28; i++)
15,436 // printf("%s",TwoDigitHEXlist[SHA328bytes[i]]);
15,437 //printf("\n");
15,438
15,439 crc64_generate_table(); //2020-Nov-20
15,440 crc32c_generate_table(); //2020-Nov-20
15,441
15,442 crc32cn_generate_table(); //2020-Dec-31
15,443 crc32kn_generate_table(); //2020-Dec-31
15,444 crc32k2n_generate_table(); //2020-Dec-31
15,445
15,446 // CRC-32C("Sanmayce") should be 0xDFA9D91A with 0x1EDC6F41, but it is 0xDFA9D91A with 0x82F63B78
15,447 //printf("%08I32X\n", crc32cn_sw("Sanmayce", 8)); // Indeed DFA9D91A
15,448 //printf("%08I32X\n", crc32c_sw("Sanmayce", 8)); // Indeed DFA9D91A
15,449
15,450 if (argc==1) {
15,451 //printf("Usage: Nakamichi filename\n"); exit(13);
```


Page 261 of 466

```
15,579     return(0);
15,580 }
15,581 */
15,582
15,583 /*
15,584 // C program to demonstrate
15,585 // example of time() function.
15,586 #include <stdio.h>
15,587 #include <time.h>
15,588 int main()
15,589 {
15,590     time_t seconds;
15,591     // Stores time seconds
15,592     time(&seconds);
15,593     printf("Seconds since January 1, 1970 = %ld\n", seconds);
15,594     return 0;
15,595 }
15,596 */
15,597
15,598 #ifdef _N_HIGH_PRIORITY
15,599     if(!SetPriorityClass(GetCurrentProcess(), HIGH_PRIORITY_CLASS))
15,600     {
15,601         // _tprintf(TEXT("Already REALTIME_PRIORITY.\n"));
15,602         // goto Cleanup;
15,603     }
15,604     if(!SetPriorityClass(GetCurrentProcess(), REALTIME_PRIORITY_CLASS))
15,605     {
15,606         // _tprintf(TEXT("Already REALTIME_PRIORITY.\n"));
15,607         // goto Cleanup;
15,608     }
15,609     // Display priority class
15,610
15,611     dwPriClass = GetPriorityClass(GetCurrentProcess());
15,612
15,613     //_tprintf(TEXT("Current priority class is 0x%x\n"), dwPriClass);
15,614
15,615     if (argc == 2) {} else // 2020-Jan-13
15,616         if (dwPriClass==0x00000080) printf("Current priority class is HIGH_PRIORITY_CLASS.\n");
15,617     if (argc == 2) {} else // 2020-Jan-13
15,618         if (dwPriClass==0x00000100) printf("Current priority class is REALTIME_PRIORITY_CLASS.\n");
15,619 #endif
15,620
15,621 #ifdef _N_HIGH_PRIORITY
15,622 // https://habrahabr.ru/post/113682/
15,623 // Andrew Aksyonoff [
15,624 /*
15,625 SetProcessAffinityMask(GetCurrentProcess(), 1); // 2020-Dec-09, causes problems in real-time priority with newer Windows 10
15,626 */
15,627 //volatile int zomg = 1;
15,628 //for ( int i=1; i<1000000000; i++ )
15,629 //    zomg *= i;
15,630 // Andrew Aksyonoff ]
15,631 #endif
15,632
15,633 if (argc==3) BandwidthFlag=1;
15,634 BandwidthFlag=0; // Disable memcpy test
15,635 //if (BandwidthFlag) Trials=64; else Trials=1;
15,636 if (argc==3) Trials=64; else Trials=1;
```

```
15,637 if ((fp = fopen(argv[1], "rb")) == NULL) {
15,638     printf("Nakamichi: Can't open '%s' file.\n", argv[1]); exit(13);
15,639 }
15,640
15,641 #if defined(_WIN32_ENVIRONMENT_)
15,642     // 64bit:
15,643     _lseeki64( fileno(fp), 0L, SEEK_END );
15,644     size_inLINESIXFOUR = _telli64( fileno(fp) );
15,645     _lseeki64( fileno(fp), 0L, SEEK_SET );
15,646 #else
15,647     // 64bit:
15,648     fseeko( fp, 0L, SEEK_END );
15,649     size_inLINESIXFOUR = ftello( fp );
15,650     fseeko( fp, 0L, SEEK_SET );
15,651 #endif /* defined(_WIN32_ENVIRONMENT_) */
15,652 SourceSize = (uint64_t)size_inLINESIXFOUR;
15,653
15,654 if( SourceSize < 4 ) // 2020-Jan-01
15,655     { printf("Nakamichi: The file being compressed should be 4 bytes or longer due to Building-Blocks being in range 4..!\n"); exit(15); }
15,656
15,657 GLOBAL_SourceSize = SourceSize; //btree matchfinder needed
15,658
15,659 //fseek(fp, 0, SEEK_END);
15,660 //SourceSize = ftell(fp);
15,661 //fseek(fp, 0, SEEK_SET);
15,662
15,663 // If filename ends in '.Nakamichi' then mode is decompression otherwise compression.
15,664 if (strcmp(argv[1]+(strlen(argv[1])-strlen(Nakamichi)), Nakamichi) == 0) {
15,665     // DECOMPRESSING [
15,666     if (argc == 2) {} else // 2020-Jan-13
15,667         printf("Allocating Source-Buffer %s MB ... \n", _ui64toaKAZEcomma((SourceSize+512)>>20, 11ToaDigits2, 10) );
15,668     SourceBlock = (char*)malloc(SourceSize+512);
15,669     if( SourceBlock == NULL )
15,670         { printf("Nakamichi: Needed memory (%luMB) allocation denied!\n", (SourceSize+512)>>20); exit(13); }
15,671
15,672     // !!! Ensure the sentinel - the first byte after the last one should be non 0x83 !!!
15,673     *(char*)(SourceBlock+SourceSize) = !0x83;
15,674
15,675     // TargetBlock = (char*)malloc(1111*1024*1024+512); // This was enwik9 setting
15,676     // TargetBlock = (char*)malloc(OneMB*2330+512); // This is GTCC_General_Textual_Compression_Corpus.tar 2,443,181,056 setting
15,677     if (argc == 2) {} else // 2020-Jan-13
15,678         printf("Allocating Target-Buffer %s MB ... \n", _ui64toaKAZEcomma((SevenGB+512)>>20, 11ToaDigits2, 10) );
15,679     TargetBlock = (char*)malloc(SevenGB+512); // 5GB
15,680     if( TargetBlock == NULL )
15,681         { printf("Nakamichi: Needed memory (%luMB) allocation denied!\n", (SevenGB+512)>>20); free(SourceBlock); exit(13); }
15,682     fread(SourceBlock, 1, SourceSize, fp);
15,683     fclose(fp);
15,684     //printf("Decompressing %lu bytes ... \n", SourceSize );
15,685     if (argc == 2) {} else // 2020-Jan-13
15,686         printf("Decompressing %s bytes (being the compressed stream) ... \n", _ui64toaKAZEcomma(SourceSize, 11ToaDigits2, 10) );
15,687     if (argc==3) {
15,688         // Warm up...
15,689         printf("Warming up ... \n");
15,690         for (i = 1; i <= Trials; i++) {
15,691             TargetSize = Decompress(TargetBlock, SourceBlock, SourceSize);
15,692             TargetSize1=SourceSize;
15,693         }
15,694         // Warm up...]
```

```

15,695 // Attempt to pick up the Turbomode score... [
15,696 clocks0 = clock();
15,697 ticksTOTAL = 0;
15,698 #if defined(_icl_mumbo_jumbo_)
15,699 ticksStart = GetRDTSC();
15,700 #endif
15,701 k1=0;
15,702 printf("RAM-to-RAM performance:\n");
15,703 if (TargetSize < (1UL<<20)) {Trials=8192; printf("The file is smaller than 1MB, increasing trials to %d.\n",Trials);}
15,704 for (j = 1; j <= 16*3; j++) {
15,705     clocks1 = clock();
15,706     while (clocks1 == clock());
15,707     clocks1 = clock();
15,708     for (i = 1; i <= Trials; i++) {
15,709         TargetSize = Decompress(TargetBlock, SourceBlock, SourceSize);
15,710     }
15,711     clocks2 = clock();
15,712     k = (((double)CLOCKS_PER_SEC*TargetSize/(double)(clocks2 - clocks1 + 1))); k=k*Trials; k=k>>20;
15,713     if (j%16==0) printf("%d MB/s\n", k); else printf("%d MB/s; ", k);
15,714     if (j%16==0) {
15,715         printf("Enforcing 17 seconds idling to avoid throttling ...\n", k);
15,716         clocks1 = clock();
15,717         while (clocks1 + (double)CLOCKS_PER_SEC*17 > clock() );
15,718     }
15,719     if (k1<k) k1=k;
15,720 }
15,721 k=k1;
15,722 k2=k;
15,723 #if defined(_icl_mumbo_jumbo_)
15,724 ticksTOTAL = ticksTOTAL + GetRDTSC() - ticksStart;
15,725 printf("This CPU seems to be working at %s MHz, or more due to ensleeping.\n", _ui64toaKAZEcomma(ticksTOTAL/((double)(clock() - clocks0 +
15,726 #endif
15,727 } else
15,728     TargetSize = Decompress(TargetBlock, SourceBlock, SourceSize);
15,729 // Attempt to pick up the Turbomode score... ]
15,730 if (argc==3) {
15,731     printf("RAM-to-RAM (peak) performance: %d MB/s.\n", k);
15,732 }
15,733 strcpy(NewFileName, argv[1]);
15,734 *( NewFileName + strlen(argv[1])-strlen(Nakamichi) ) = '\0';
15,735 if (argc == 2) {} else // 2020-Jan-13
15,736 printf("Source-file-Hash(FNV1A_YoshimitsuTRIAD) = 0x%s\n", _ui64toaKAZEzerocomma4(FNV1A_Hash_YoshimitsuTRIAD(SourceBlock, SourceSize), 11ToaDigits2, 16)+(26-
15,737 8-1) );
15,738 if (argc == 2) {} else // 2020-Jan-13
15,739 printf("Target-file-Hash(FNV1A_YoshimitsuTRIAD) = 0x%s\n", _ui64toaKAZEzerocomma4(FNV1A_Hash_YoshimitsuTRIAD(TargetBlock, TargetSize), 11ToaDigits2, 16)+(26-
15,740 8-1) );
15,741 if (argc!=3) { // 2020-Feb-02, do not dump if in bench mode [
15,742 // 2020-Jan-13 [
15,743 // https://stackoverflow.com/questions/16888339/what-is-the-simplest-way-to-write-to-stdout-in-binary-mode
15,744 // The line below crashes under Windows XP?!
15,745 //if (!freopen(NULL, "wb", stdout)) { printf("Can't set output stream to binary mode.\n"); exit(23); }
15,746 //https://stackoverflow.com/questions/23107609/is-there-way-to-set-stdout-to-binary-mode
15,747 //if defined(_WIN32
15,748 #if defined(_WIN32_ENVIRONMENT_)

```



```
15,750 setmode(fileno(stdout), O_BINARY);
15,751 setmode(fileno(stdin), O_BINARY);
15,752 #endif
15,753 //https://github.com/KirillKryukov/naf/blob/4e737ce8852f5bde2ca5fb31ee6d2b9c34f1a5cb/unnaf/src/files.c#L29
15,754
15,755 // if (TargetSize <= OneMB) {
15,756 if (TargetSize <= 3800ULL*1024*1024) { // 2020-Jun-08, to boost dumping speed in SCB
15,757     fwrite(TargetBlock, 1, TargetSize, stdout);
15,758 } else {
15,759     for (DoOffset = 0; DoOffset+OneMB < TargetSize; DoOffset=DoOffset+OneMB) {
15,760         fwrite(TargetBlock+DoOffset, 1, OneMB, stdout);
15,761     }
15,762     //if (DoOffset+OneMB >= TargetSize) {
15,763         fwrite(TargetBlock+DoOffset, 1, TargetSize - DoOffset, stdout);
15,764     }
15,765     fclose(stdout);
15,766 // 2020-Jan-13 ]
15,767
15,768 /*
15,769 if ((fp = fopen(NewFileName, "wb")) == NULL) {
15,770     printf("Nakamichi: Can't write '%s' file.\n", NewFileName); exit(13);
15,771 }
15,772 //fwrite(TargetBlock, 1, TargetSize, fp); // Caramba: It doesn't work when file is 4+GB long!
15,773 if (TargetSize <= OneMB) {
15,774     fwrite(TargetBlock, 1, TargetSize, fp);
15,775 } else {
15,776     for (DoOffset = 0; DoOffset+OneMB < TargetSize; DoOffset=DoOffset+OneMB) {
15,777         fwrite(TargetBlock+DoOffset, 1, OneMB, fp);
15,778     }
15,779     //if (DoOffset+OneMB >= TargetSize) {
15,780         fwrite(TargetBlock+DoOffset, 1, TargetSize - DoOffset, fp);
15,781     }
15,782     fclose(fp);
15,783 */
15,784 } //if (argc!=3) { // 2020-Feb-02, do not dump if in bench mode [
15,785
15,786 // LZSSE2 [
15,787 #ifdef _N_alone
15,788 #else
15,789 if (argc==3) {
15,790     free(TargetBlock);
15,791     free(SourceBlock);
15,792     memcpy(NewFileName2, argv[1], (strlen(argv[1])-strlen(Nakamichi)));
15,793     memcpy(NewFileName2+strlen(argv[1])-strlen(Nakamichi), LZSSE, strlen(LZSSE)+1); //+1 to add NULL
15,794 if ((fp = fopen(NewFileName2, "rb")) == NULL) {
15,795     printf("Nakamichi: Can't open '%s' file.\n", NewFileName2); exit(13);
15,796 }
15,797 #if defined(_WIN32_ENVIRONMENT_)
15,798     // 64bit:
15,799     _lseeki64( fileno(fp), 0L, SEEK_END );
15,800     size_inLINESIXFOUR = _telli64( fileno(fp) );
15,801     _lseeki64( fileno(fp), 0L, SEEK_SET );
15,802 #else
15,803     // 64bit:
15,804     fseeko( fp, 0L, SEEK_END );
15,805     size_inLINESIXFOUR = ftello( fp );
15,806     fseeko( fp, 0L, SEEK_SET );
15,807 #endif /* defined(_WIN32_ENVIRONMENT_) */
```

```

15,808 SourceSize = (uint64_t)size_inLINESIXFOUR;
15,809
15,810 printf("Allocating Source-Buffer %s MB ... \n", _ui64toaKAZEcomma((SourceSize+512)>>20, 11TOaDigits2, 10) );
15,811 SourceBlock = (char*)malloc(SourceSize+512);
15,812 if( SourceBlock == NULL )
15,813     { printf("Nakamichi: Needed memory (%luMB) allocation denied!\n", (SourceSize+512)>>20); exit(13); }
15,814 // TargetBlock = (char*)malloc(1111*1024*1024+512); // This was enwik9 setting
15,815 // TargetBlock = (char*)malloc(OneMB*2330+512); // This is GTCC_General_Textual_Compression_Corpus.tar 2,443,181,056 setting
15,816 printf("Allocating Target-Buffer %s MB ... \n", _ui64toaKAZEcomma((SevenGB+512)>>20, 11TOaDigits2, 10) );
15,817 TargetBlock = (char*)malloc(SevenGB+512); // 5GB
15,818 if( TargetBlock == NULL )
15,819     { printf("Nakamichi: Needed memory (%luMB) allocation denied!\n", (SevenGB+512)>>20); free(SourceBlock); exit(13); }
15,820 fread(SourceBlock, 1, SourceSize, fp);
15,821 fclose(fp);
15,822 //printf("Decompressing %lu bytes ... \n", SourceSize );
15,823 printf("Decompressing '%s' (%s bytes, being the compressed stream) ... \n", NewFileName2, _ui64toaKAZEcomma(SourceSize, 11TOaDigits2, 10) );
15,824 // Warm up...[
15,825 printf("Warming up ... \n");
15,826 for (i = 1; i <= Trials; i++) {
15,827     TargetSize = LZSSE2-Decompress( SourceBlock, SourceSize, TargetBlock, TargetSize );
15,828     TargetSize2=SourceSize;
15,829 }
15,830 // Warm up...[
15,831 // Attempt to pick up the Turbomode score... [
15,832 clocks0 = clock();
15,833 ticksTOTAL = 0;
15,834 #if defined(_icl_mumbo_jumbo_)
15,835 ticksStart = GetRDTSC();
15,836 #endif
15,837 k1=0;
15,838 printf("RAM-to-RAM performance:\n");
15,839 if (TargetSize < (1UL<<20)) {Trials=8192; printf("The file is smaller than 1MB, increasing trials to %d.\n",Trials);}
15,840 for (j = 1; j <= 16*3; j++) {
15,841     clocks1 = clock();
15,842     while (clocks1 == clock());
15,843     clocks1 = clock();
15,844     for (i = 1; i <= Trials; i++) {
15,845         TargetSize = LZSSE2-Decompress( SourceBlock, SourceSize, TargetBlock, TargetSize );
15,846     }
15,847     clocks2 = clock();
15,848     k = (((double)CLOCKS_PER_SEC*TargetSize/(double)(clocks2 - clocks1 + 1))); k=k*Trials; k=k>>20;
15,849     if (j%16==0) printf("%d MB/s\n", k); else printf("%d MB/s; ", k);
15,850     if (j%16==0) {
15,851         printf("Enforcing 17 seconds idling to avoid throttling ... \n", k);
15,852         clocks1 = clock();
15,853         while (clocks1 + (double)CLOCKS_PER_SEC*17 > clock() );
15,854     }
15,855     if (k1<k) k1=k;
15,856 }
15,857 k=k1;
15,858 k3=k;
15,859 #if defined(_icl_mumbo_jumbo_)
15,860 ticksTOTAL = ticksTOTAL + GetRDTSC() - ticksStart;
15,861 printf("This CPU seems to be working at %s MHz, or more due to ensleeping.\n", _ui64toaKAZEcomma(ticksTOTAL/((double)(clock() - clocks0 + 1)/(double)CLOCKS_PER_SEC)/1000000, 11TOaDigits, 10));
15,862 #endif
15,863
15,864 // Attempt to pick up the Turbomode score... ]

```

```
15,865         printf("RAM-to-RAM (peak) performance: %d MB/s.\n", k);
15,866     printf("Nakamichi 'Ryuugan' vs LZSSE2 17, c.size: %.2fx\n", (double)(TargetSize1) / (double)(TargetSize2) );
15,867     printf("LZSSE2 17 vs Nakamichi 'Ryuugan', d.rate: %.2fx\n", (double)(k3) / (double)(k2) );
15,868     printf("Bottomline:\n", k);
15,869     printf("Nakamichi 'Ryuugan' expanding %.2fx to %s at %d MB/s.\n", (double)(TargetSize) / (double)(TargetSize1), _ui64toaKAZEcomma(TargetSize, 11TOaDigits2, 10), k2 );
15,870 }
15,871 #endif
15,872 // LZSSE2 ]
15,873 // DECOMPRESSING ]
15,874 } else {
15,875 // COMPRESSING [
15,876
15,877 if (argc==6) { // overwrite the 'DEFINE' defaults
15,878
15,879 HashInBITS_GLOBAL= atoi(argv[3]);
15,880 HashChunkSizeInBITS_GLOBAL= atoi(argv[3]);
15,881
15,882 if (*argv[5]=='i') {
15,883 BSTorBtree=3;
15,884 }
15,885 if (*argv[5]=='I') {
15,886 BSTorBtree=3;
15,887 if (HashInBITS_GLOBAL > 7) HashChunkSizeInBITS_GLOBAL= HashInBITS_GLOBAL - 7;
15,888 }
15,889 if (*argv[5]=='e') {
15,890 BSTorBtree=2;
15,891 }
15,892 if (*argv[5]=='E') {
15,893 BSTorBtree=2;
15,894 if (HashInBITS_GLOBAL > 7) HashChunkSizeInBITS_GLOBAL= HashInBITS_GLOBAL - 7;
15,895 }
15,896
15,897 RAMPoolInKB_GLOBAL= atoi(argv[4]);
15,898
15,899 #ifdef LITE
15,900 printf("This compile uses B-trees only, no memmem() invocations - it compresses worse but much faster.\n"); // 2020-Feb-14
15,901 #endif
15,902 }
15,903 }
15,904
15,905 Trials=256;
15,906 printf("Allocating Source-Buffer %s MB ... \n", _ui64toaKAZEcomma((SourceSize+512)>>20, 11TOaDigits2, 10) );
15,907 SourceBlock = (char*)malloc(SourceSize+512);
15,908 if( SourceBlock == NULL )
15,909     { printf("Nakamichi: Needed memory (%sMB) allocation denied!\n", _ui64toaKAZEcomma((SourceSize+512)>>20, 11TOaDigits2, 10)); exit(13); }
15,910
15,911 GLOBAL_SourceBlock = SourceBlock; //btree matchfinder needed
15,912 GLOBAL_CurrentPositionForReading_TAILforLookAhead = SourceBlock; //btree matchfinder needed
15,913
15,914 for (i = 0; i < MatchLensNUM; i++)
15,915     GLOBAL_CurrentPositionForReading_TAILforLookAhead_ARRAY[i] = SourceBlock; // 2020-Jan-25
15,916
15,917 #ifdef LITE
15,918 #else
15,919 printf("Allocating Source-Buffer %s MB (REVERSED) ... \n", _ui64toaKAZEcomma((SourceSize+512)>>20, 11TOaDigits2, 10) );
15,920 SourceBlockREVERSED = (char*)malloc(SourceSize+512);
15,921 if( SourceBlockREVERSED == NULL )
15,922     { printf("Nakamichi: Needed memory (%sMB) allocation denied!\n", _ui64toaKAZEcomma((SourceSize+512)>>20, 11TOaDigits2, 10)); exit(13); }
```

```
15,923 #endif
15,924
15,925 printf("Allocating Target-Buffer %s MB ... \n", _ui64toaKAZEcomma((SourceSize+512+(unsigned long long)SpeedUpBuilding*1024*1024)>>20, 11TOaDigits2, 10) );
15,926 TargetBlock = (char*)malloc(SourceSize+512+(unsigned long long)SpeedUpBuilding*1024*1024); /*+32*1024*1024, some files may be expanded instead of compressed.
15,927 if( TargetBlock == NULL )
15,928 { printf("Nakamichi: Needed memory (%sMB) allocation denied!\n", _ui64toaKAZEcomma((SourceSize+512+(unsigned long long)SpeedUpBuilding*1024*1024)>>20,
11TOaDigits2, 10)); free(SourceBlock); exit(13); }
15,929 // Allocating before the compression, TO TRY TO AVOID JUST-MAPPING [
15,930 //printf("Allocating Verification-Buffer %s MB ... \n", _ui64toaKAZEcomma((SourceSize+512)>>20, 11TOaDigits2, 10) );
15,931 //VerifyBlock = (char*)malloc(SourceSize+512); // 2019-Dec-04
15,932 //if( VerifyBlock == NULL )
15,933 //{ printf("Nakamichi: Needed memory (%sMB) allocation denied!\n", _ui64toaKAZEcomma((SourceSize+512)>>20, 11TOaDigits2, 10)); exit(13); }
15,934 // Allocating before the compression, TO TRY TO AVOID JUST-MAPPING ]
15,935 fread(SourceBlock, 1, SourceSize, fp);
15,936 fclose(fp);
15,937
15,938 #ifdef LITE
15,939 #else
15,940 for (i64 = 1; i64 <= SourceSize; i64++) { // 2020-Feb-26
15,941     SourceBlockREVERSED[i64-1]=SourceBlock[SourceSize-i64]; // 2020-Feb-26 // Again in 2020-Apr-12
15,942 }
15,943 #endif
15,944
15,945 printf("Source-file-Hash(FNV1A_YoshimitsuTRIAD) = 0x%s\n", _ui64toaKAZEzerocomma4(FNV1A_Hash_YoshimitsuTRIAD(SourceBlock, SourceSize), 11TOaDigits2, 16)+(26-8-1) );
// 2020-Jun-08
15,946
15,947 // Leprechaun BBhex [[
15,948 // B_tree(argc, argv, SourceBlock, SourceSize); // This is the generic one (2019-Aug-06, the shared one on GIST/LTCB), it inserts ALL keys into B-trees, good for
general searching also, but too heavy for compression.
15,949 //B_tree_Non_Unique_Only(argc, argv, SourceBlock, SourceSize, TargetBlock); // This one adds another pass before the old one, it uses the last (the buffer for
verification) N in size malloc() for building B-trees - for speed. Then it proceeds again with the same pass but builds either on E or I while checking whether the
current/candidate key is with occurrence >1, if so, then skip it and not insert it. SHARED on centminmod forum
15,950 B_tree_Non_Unique_Only_DEFRAGMENTED(argc, argv, SourceBlock, SourceSize, TargetBlock); // This is the successor of above/fragmented revision of 2019-Dec-19 - it has
defragmented B-trees - all leaves are in one continuous block
15,951
15,952 // Leprechaun BBhex ]]
15,953
15,954 //printf("Compressing %lu bytes ... \n", SourceSize );
15,955 printf("Compressing %s bytes ... \n", _ui64toaKAZEcomma(SourceSize, 11TOaDigits2, 10) );
15,956 clocks1 = clock();
15,957 while (clocks1 == clock());
15,958 clocks1 = clock();
15,959 time1=time(NULL); //fix of bigtime
15,960 TargetSize = Compress(TargetBlock, SourceBlock, SourceBlockREVERSED, SourceSize);
15,961 //free(SourceBlockREVERSED); // 2019-Dec-07
15,962 TargetSize1=TargetSize;
15,963 clocks2 = clock();
15,964 //k = (((double)CLOCKS_PER_SEC*SourceSize/(double)(clocks2 - clocks1 + 1))); //k=k>>10;
15,965 k=(double)SourceSize/(double)(time(NULL) - time1 + 1);
15,966 printf("RAM-to-RAM performance: %d B/s.\n", k);
15,967 strcpy(NewFileName, argv[1]);
15,968 strcat(NewFileName, Nakamichi);
15,969 //printf("Compressed to %d bytes.\n", TargetSize );
15,970 printf("Compressed to %s bytes.\n", _ui64toaKAZEcomma(TargetSize, 11TOaDigits2, 10) );
15,971 Filehash = FNV1A_Hash_YoshimitsuTRIAD(SourceBlock, SourceSize);
15,972 printf("Source-file-Hash(FNV1A_YoshimitsuTRIAD) = 0x%s\n", _ui64toaKAZEzerocomma4(Filehash, 11TOaDigits2, 16)+(26-8-1) );
15,973 printf("Target-file-Hash(FNV1A_YoshimitsuTRIAD) = 0x%s\n", _ui64toaKAZEzerocomma4(FNV1A_Hash_YoshimitsuTRIAD(TargetBlock, TargetSize), 11TOaDigits2, 16)+(26-
8-1) );
```

```

15,974 //if ((fp = fopen(NewFileName, "wb")) == NULL) {
15,975 if ((fp = fopen(argv[2], "wb")) == NULL) { // 2020-Jan-13:
15,976 //     printf("Nakamichi: Can't write '%s' file.\n", NewFileName); exit(13);
15,977     printf("Nakamichi: Can't write '%s' file.\n", argv[2]); exit(13); // 2020-Jan-13:
15,978 }
15,979 //fwrite(TargetBlock, 1, TargetSize, fp); // Caramba: It doesn't work when file is 4+GB long!
15,980 if (TargetSize <= OneMB) {
15,981     fwrite(TargetBlock, 1, TargetSize, fp);
15,982 } else {
15,983     for (DoOffset = 0; DoOffset+OneMB < TargetSize; DoOffset=DoOffset+OneMB) {
15,984         fwrite(TargetBlock+DoOffset, 1, OneMB, fp);
15,985     }
15,986     //if (DoOffset+OneMB >= TargetSize) {
15,987         fwrite(TargetBlock+DoOffset, 1, TargetSize - DoOffset, fp);
15,988     }
15,989     fclose(fp);
15,990
15,991 #ifdef LITE
15,992 #else
15,993     printf("Decompressing %s (being the compressed stream) bytes ...\n", _ui64toaKAZEcomma(TargetSize, 11ToADigits2, 10) );
15,994     printf("RAM-to-RAM performance: ");
15,995     kMAXIMUM = 0;
15,996     if (SourceSize < (1LL<<20)) Trials = 7777; else Trials = 77;
15,997     for (j = 1; j <= 5; j++) {
15,998         clocks1 = clock();
15,999         while (clocks1 == clock());
16,000         clocks1 = clock();
16,001     for (i = 1; i <= Trials; i++) {
16,002         //VerifySize = Decompress(VerifyBlock, TargetBlock, TargetSize); // 2019-Dec-07
16,003         VerifySize = Decompress(SourceBlockREVERSED, TargetBlock, TargetSize); // 2019-Dec-07
16,004     }
16,005         clocks2 = clock();
16,006         k1 = (((double)CLOCKS_PER_SEC*VerifySize/(double)(clocks2 - clocks1 + 1))); k1=k1*Trials; k1=k1>>20;
16,007         printf("%s MB/s (Trials = %s); ", _ui64toaKAZEcomma(k1, 11ToADigits3, 10), _ui64toaKAZEcomma(Trials, 11ToADigits2, 10));
16,008         if (k1 > kMAXIMUM) kMAXIMUM = k1;
16,009         Trials = Trials*2;
16,010     } //j
16,011     k1 = kMAXIMUM;
16,012     printf(": %d MB/s\n", k1);
16,013     if (VerifySize == SourceSize) printf("Verification (input and output sizes match) OK.\n"); else printf("Verification (input and output sizes mismatch)
FAILED!\n");
16,014     //if (memcmp(SourceBlock, SourceBlockREVERSED, SourceSize)==0) printf("Verification (input and output blocks match) OK.\n"); else printf("Verification (input
and output blocks mismatch) FAILED!\n"); // 2019-Dec-07
16,015     if (memcmp(SourceBlock, SourceBlockREVERSED, SourceSize)==0) printf("Verification (input and output blocks match) OK.\n"); else printf("Verification (input
and output blocks mismatch) FAILED!\n"); // 2019-Dec-07
16,016 #endif
16,017
16,018 // LZSSE2 [
16,019 #ifdef _N_alone
16,020 #else
16,021
16,022     printf("LZSSE2: Compressing with LZSSE2 (level 17) %s bytes ...\n", _ui64toaKAZEcomma(SourceSize, 11ToADigits2, 10) );
16,023     clocks1 = clock();
16,024     while (clocks1 == clock());
16,025     clocks1 = clock();
16,026     s = LZSSE2_MakeOptimalParseState(SourceSize);
16,027     if (s==NULL) { printf("Errorful compression, allocation stage!\n"); exit(13); };
16,028     TargetSize = LZSSE2_CompressOptimalParse( s, SourceBlock, SourceSize, TargetBlock, SourceSize+3*1024*1024, 17 );

```

```

16,029     TargetSize2=TargetSize;
16,030     if (TargetSize == 0) { printf("Errorful compression!\n"); exit(13); };
16,031     LZSSE2_FreeOptimalParseState(s);
16,032     //printf("LZSSE2: Compressed to %d bytes.\n", TargetSize );
16,033     printf("LZSSE2: Compressed to %s bytes.\n", _ui64toaKAZEcomma(TargetSize, 11TOaDigits2, 10) );
16,034     clocks2 = clock();
16,035     k = (((double)CLOCKS_PER_SEC*SourceSize/(double)(clocks2 - clocks1 + 1))); k=k>>10;
16,036     printf("LZSSE2: RAM-to-RAM performance: %d KB/s.\n", k);
16,037     // Another operation (I/O) TO TRY TO AVOID JUST-MAPPING [
16,038     strcpy(NewFileName2, argv[1]);
16,039     strcat(NewFileName2, LZSSE2);
16,040     if ((fp = fopen(NewFileName2, "wb")) == NULL) {
16,041         printf("Nakamichi: Can't write '%s' file.\n", NewFileName2); exit(13);
16,042     }
16,043     fwrite(TargetBlock, 1, TargetSize, fp); // Caramba: It doesn't work when file is 4+GB long!
16,044     fclose(fp);
16,045     // Another operation (I/O) TO TRY TO AVOID JUST-MAPPING ]
16,046     //     printf("LZSSE2: Allocating Verification-Buffer %s MB ... \n", _ui64toaKAZEcomma((SourceSize+512)>>20, 11TOaDigits2, 10) );
16,047     //     VerifyBlock = (char*)malloc(SourceSize+512);
16,048     //     if( VerifyBlock == NULL )
16,049     //     { printf("Nakamichi: Needed memory (%sMB) allocation denied!\n", _ui64toaKAZEcomma((SourceSize+512)>>20, 11TOaDigits2, 10)); exit(13); }
16,050     printf("LZSSE2: Decompressing %s bytes (being the compressed stream) ... \n", _ui64toaKAZEcomma(TargetSize, 11TOaDigits2, 10) );
16,051     printf("LZSSE2: RAM-to-RAM performance: ");
16,052     kMAXIMUM = 0;
16,053     if (SourceSize < (1LL<<20)) Trials = 7777; else Trials = 77;
16,054     for (j = 1; j <= 5; j++) {
16,055         clocks1 = clock();
16,056         while (clocks1 == clock());
16,057         clocks1 = clock();
16,058         for (i = 1; i <= Trials; i++) {
16,059             //VerifySize = LZSSE2-Decompress( TargetBlock, TargetSize, VerifyBlock, SourceSize ); // 2019-Dec-07
16,060             VerifySize = LZSSE2-Decompress( TargetBlock, TargetSize, SourceBlockREVERSED, SourceSize ); // 2019-Dec-07
16,061         }
16,062         clocks2 = clock();
16,063         k2 = (((double)CLOCKS_PER_SEC*VerifySize/(double)(clocks2 - clocks1 + 1))); k2=k2*Trials; k2=k2>>20;
16,064         printf("%s MB/s (Trials = %s); ", _ui64toaKAZEcomma(k2, 11TOaDigits3, 10), _ui64toaKAZEcomma(Trials, 11TOaDigits2, 10));
16,065         if (k2 > kMAXIMUM) kMAXIMUM = k2;
16,066         Trials = Trials*2;
16,067     } //j
16,068     k2 = kMAXIMUM;
16,069     printf(": %d MB/s\n",k2);
16,070     if(VerifySize == SourceSize) printf("LZSSE2: Verification (input and output sizes match) OK.\n"); else printf("LZSSE2: Verification (input and output sizes
mismatch) FAILED!\n");
16,071     //if (memcmp(SourceBlock, VerifyBlock, SourceSize)==0) printf("LZSSE2: Verification (input and output blocks match) OK.\n"); else printf("LZSSE2: Verification
(input and output blocks mismatch) FAILED!\n"); // 2019-Dec-07
16,072     if (memcmp(SourceBlock, SourceBlockREVERSED, SourceSize)==0) printf("LZSSE2: Verification (input and output blocks match) OK.\n"); else printf("LZSSE2:
Verification (input and output blocks mismatch) FAILED!\n"); // 2019-Dec-07
16,073 #endif
16,074 // LZSSE2 ]
16,075
16,076 #ifdef _N_alone
16,077 #else
16,078
16,079     printf("LZSSE2 vs Nakamichi 'Ryuugan', c.size: %.2fx\n", (double)(TargetSize2) / (double)(TargetSize1) );
16,080     if (k1 != 0)
16,081         printf("LZSSE2 vs Nakamichi 'Ryuugan', d.rate: %.2fx\n", (double)(k2) / (double)(k1) );
16,082     //     else
16,083     //         printf("LZSSE2 vs Nakamichi 'Okamigan', quicker: %.2f:1\n", 0 );

```

```

16,084
16,085 printf("Railgun_INVOCATIONS (the-lower-the-better) = %s\n", _ui64toaKAZEcomma(GLOBAL_Railgun_INVOCATIONS, 11TOaDigits2, 10) );
16,086 for (i = 0; i <= 704; i++) { // 2020-Jan-29
16,087     if (GLOBAL_Railgun_INVOCATIONS_ARRAY[i])
16,088         printf("Railgun_INVOCATIONS (for needle %s bytes) = %s\n", _ui64toaKAZEzerocomma(i, 11TOaDigits3, 10)+(26-3),
16,089         _ui64toaKAZEcomma(GLOBAL_Railgun_INVOCATIONS_ARRAY[i], 11TOaDigits2, 10) );
16,090 }
16,090 //printf("Railgun_INVOCATIONS for 004:3 = %s\n", _ui64toaKAZEcomma(GLOBAL_Railgun_INVOCATIONS_004_3_bytes, 11TOaDigits2, 10) );
16,091 //printf("Railgun_INVOCATIONS for 004:2 = %s\n", _ui64toaKAZEcomma(GLOBAL_Railgun_INVOCATIONS_004_2_bytes, 11TOaDigits2, 10) );
16,092 //printf("Railgun_INVOCATIONS for 004:1 = %s\n", _ui64toaKAZEcomma(GLOBAL_Railgun_INVOCATIONS_004_1_bytes, 11TOaDigits2, 10) );
16,093
16,094 #endif
16,095 //free(VerifyBlock); // 2019-Dec-07
16,096
16,097 #ifdef LITE
16,098 #else
16,099     free(SourceBlockREVERSED); // 2019-Dec-07
16,100 #endif
16,101
16,102 // COMPRESSING ]
16,103
16,104 #ifdef LITE
16,105 #else
16,106 // LOG writing [
16,107 if( ( fp_outLOG = fopen( "Nakamichi.LOG", "a+" ) ) == NULL )
16,108 { printf( "Nakamichi: Can't open file Nakamichi.LOG.\n" ); return( 1 ); }
16,109 #if defined(_WIN32_ENVIRONMENT_)
16,110     // 64bit:
16,111 _lseeki64( fileno(fp_outLOG), 0L, SEEK_END );
16,112 size_inLINESIXFOURlog = _telli64( fileno(fp_outLOG) );
16,113 _lseeki64( fileno(fp_outLOG), 0L, SEEK_SET );
16,114 #else
16,115     // 64bit:
16,116 fseeko( fp_outLOG, 0L, SEEK_END );
16,117 size_inLINESIXFOURlog = ftello( fp_outLOG );
16,118 fseeko( fp_outLOG, 0L, SEEK_SET );
16,119 #endif /* defined(_WIN32_ENVIRONMENT_) */
16,120 if ( (uint64_t)size_inLINESIXFOURlog == 0 )
16,121 fprintf( fp_outLOG, "| #1 Filesize      | #2 Filehash | #3 Nakamichi 'EE' c.size/decompressionrate | #4 LZSSE2 c.size/decompressionrate | #5 Filename \n" );
16,122 fprintf( fp_outLOG, "| %s ", _ui64toaKAZEzerocomma(size_inLINESIXFOURlog, 11TOaDigits3, 10)+(26-15) );
16,123 fprintf( fp_outLOG, "| 0x%s ", _ui64toaKAZEzerocomma4(Filehash, 11TOaDigits2, 16)+(26-8-1) );
16,124 fprintf( fp_outLOG, "| %s / %sMB/s", _ui64toaKAZEzerocomma(TargetSize1, 11TOaDigits2, 10)+(26-13), _ui64toaKAZEzerocomma(k1, 11TOaDigits3, 10)+(26-7) );
16,125 #ifdef _N_alone // 2020-Feb-15
16,126 #else
16,127 fprintf( fp_outLOG, "| %s / %sMB/s", _ui64toaKAZEzerocomma(TargetSize2, 11TOaDigits2, 10)+(26-13), _ui64toaKAZEzerocomma(k2, 11TOaDigits3, 10)+(26-7) );
16,128 #endif
16,129 fprintf( fp_outLOG, "| %s \n", argv[1] );
16,130 fclose(fp_outLOG);
16,131 // LOG writing ]
16,132 #endif
16,133 }
16,134
16,135 if (BandwidthFlag) {
16,136 // Benchmark memcpy() [
16,137 pointerALIGN = TargetBlock + 64 - (((size_t)TargetBlock) % 64);
16,138 //offset=64-int((long)data&63);
16,139 printf("Memory pool starting address: %p ... ", pointerALIGN);

```

```

16,140 if (((uintptr_t)(const void *)pointerALIGN & (64 - 1)) == 0) printf( "64 byte aligned, OK\n"); else printf( "NOT 64 byte aligned, FAILURE\n");
16,141 clocks3 = clock();
16,142 while (clocks3 == clock());
16,143 clocks3 = clock();
16,144 printf("Copying a %dMB block 1024 times i.e. %dGB READ + %dGB WRITTEN ...\n", 512, 512, 512);
16,145 for (i = 0; i < 1024; i++) {
16,146     memcpy(pointerALIGN+512*1024*1024, pointerALIGN, 512*1024*1024);
16,147 }
16,148 clocks4 = clock();
16,149 duration = (double) (clocks4 - clocks3 + 1);
16,150 durationGENERIC = duration;
16,151 printf("memcpy(): (%dMB block); %dMB copied in %d clocks or %.3fMB per clock\n", 512, 1024*( 512 ), (int) duration, (double)1024*( 512 )/ ((int) duration));
16,152
16,153 /*
16,154 #ifndef _N_GP
16,155 clocks3 = clock();
16,156 while (clocks3 == clock());
16,157 clocks3 = clock();
16,158 printf("Copying a %dMB block 1024 times i.e. %dGB READ + %dGB WRITTEN ...\n", 512, 512, 512);
16,159 for (i = 0; i < 1024; i++) {
16,160     memcpy_SSE2_4K_prefetched(pointerALIGN+512*1024*1024, pointerALIGN, 512*1024*1024);
16,161 }
16,162 clocks4 = clock();
16,163 duration = (double) (clocks4 - clocks3 + 1);
16,164 printf("memcpy_SSE2_4K_prefetched(): (%dMB block); %dMB copied in %d clocks or %.3fMB per clock\n", 512, 1024*( 512 ), (int) duration, (double)1024*( 512 )/ ((int)
duration));
16,165 #endif
16,166
16,167 #ifdef _N_YMM
16,168 clocks3 = clock();
16,169 while (clocks3 == clock());
16,170 clocks3 = clock();
16,171 printf("Copying a %dMB block 1024 times i.e. %dGB READ + %dGB WRITTEN ...\n", 512, 512, 512);
16,172 for (i = 0; i < 1024; i++) {
16,173     memcpy_AVX_4K_prefetched(pointerALIGN+512*1024*1024, pointerALIGN, 512*1024*1024);
16,174 }
16,175 clocks4 = clock();
16,176 duration = (double) (clocks4 - clocks3 + 1);
16,177 printf("memcpy_AVX_4K_prefetched(): (%dMB block); %dMB copied in %d clocks or %.3fMB per clock\n", 512, 1024*( 512 ), (int) duration, (double)1024*( 512 )/ ((int)
duration));
16,178 #endif
16,179 */
16,180 // Benchmark memcpy() ]
16,181 //k = (((double)1000*TargetSize/(clocks2 - clocks1 + 1))); k=k>>20;
16,182 j = (double)1000*1024*( 512 )/ ((int) durationGENERIC);
16,183 printf("RAM-to-RAM performance vs memcpy() ratio (bigger-the-better): %d%%\n", (int)((double)k*100/j));
16,184 }
16,185
16,186 free(TargetBlock);
16,187 free(SourceBlock);
16,188 exit(0);
16,189 }
16,190
16,191 // See #30 8:3= 2.6 (1MB) section for replacing Backward 'BawBaw' with Forward 'Trolldom'!
16,192 void SearchIntoSlidingWindow(unsigned int* HowManyDittoTagsToEmit, unsigned int* ShortMediumLongOFFSET, uint64_t* retIndex, unsigned int* retMatch, char*
refStart, char* refEnd, char* encStart, char* encEnd, char* src, char* srcR, uint64_t srcSize){
16,193     char* Skip;
16,194     char* FoundAtPosition;

```



```
16,195 char* FoundAtPosition2;
16,196 unsigned int match=0;
16,197 int i;
16,198
16,199 // Too lazy to write Railgun-Reverse, it would save many ugly patches...
16,200 char* refStartSW1 = refEnd-(16-1); // 11b
16,201 char* refStartSW2 = refEnd-(4*8*128-1); // 10b
16,202 char* refStartSW2ryuu = refEnd-(8*8*128-1); // 10b
16,203 char* refStartSW3 = refEnd-(1024*8*128-1); // 01b
16,204 char* refStartSW3ryuu = refEnd-(2*1024*8*128-1); // 01b
16,205 char* refStartSW4 = refEnd-(256*1024*8*128-1); // 00b
16,206 char* refStartSW5 = refEnd-(512*1024*8*128-1); // 00b
16,207 char* refStartSW5ryuu = refEnd-(128LL*2*512*1024*8*128-1); // 00b
16,208 char* refStartSW6ryuu = refEnd-(1024LL*2*512*1024*8*128-1); // 00b
16,209 char* refStartSW1a = refEnd-(16*1024*8*128-1); // 11b
16,210 char* refStartSW1b = refEnd-(64*8*128-1); // 11b
16,211 char* refStartSW1c = refEnd-(256-1); // 11b
16,212 char* refStartSW1d = refEnd-(256*4-1); // 11b
16,213 char* refStartSW1d2 = refEnd-(256*8-1); // 11b
16,214 char* refStartSW1e = refEnd-(64*1024*8*128-1); // 11b
16,215
16,216 char* refStartSW512 = refEnd-(512-1); // 11b
16,217 char* refStartSW128kb = refEnd-(128*8*128-1); // 11b
16,218
16,219 // In order to avoid the unheardof slowness the 256MB may be reduced to 2MB... // --!
16,220 char* refStartHOT = refEnd-(256*8*128-1); // !
16,221 char* refStartHOTTER = refEnd-(4*8*128-1); // !
16,222 char* refStartHOTTERryuu = refEnd-(8*8*128-1); // !
16,223 char* refStartHOTTEST = refEnd-(16-1); // !
16,224 char* refStartCOLDERbig = refEnd-(1024*8*128-1); // !
16,225 char* refStartCOLDERbigryuu = refEnd-(2*1024*8*128-1); // !
16,226 char* refStartCOLDERERbig = refEnd-(2048*8*128-1); // !
16,227 char* refStartCOLDERERERbig = refEnd-(8192*8*128-1); // !
16,228 char* refStartCOLDERERERERbig = refEnd-(512*1024*8*128-1); // <-!
16,229 char* refStartCOLDERERERERERbigryuu = refEnd-(128LL*2*512*1024*8*128-1); // <-!
16,230 char* refStartCOLDESTbig = refStart;
16,231
16,232 *retIndex=0;
16,233 *retMatch=0;
16,234 *ShortMediumLongOFFSET=0;
16,235
16,236 #ifdef ReplaceBruteForceWithRailgunSwampshineBailOut
16,237
16,238 //btree [
16,239 uint64_t LastSeenOffset_PseudoPointer04=0;
16,240 uint64_t LastSeenOffset_PseudoPointer06=0;
16,241 uint64_t LastSeenOffset_PseudoPointer08=0;
16,242 uint64_t LastSeenOffset_PseudoPointer10=0;
16,243 uint64_t LastSeenOffset_PseudoPointer12=0;
16,244 uint64_t LastSeenOffset_PseudoPointer14=0;
16,245 uint64_t LastSeenOffset_PseudoPointer16=0;
16,246 uint64_t LastSeenOffset_PseudoPointer18=0;
16,247 //uint64_t LastSeenOffset_PseudoPointer[MatchLensNUM];
16,248 uint64_t LastSeenOffset_PseudoPointer[MatchLensNUMdummy]; // 2020-Jun-28
16,249 // Before main() are already defined:
16,250 //FILE *fp_outRG; // Global - not to burden the extract/compare function with one more parameter
16,251 //char *GLOBAL_HASHPOT; //btree matchfinder needed, the hash table/pool
16,252 //char* GLOBAL_SourceBlock; //btree matchfinder needed, the file in-memory itself
```

```
16,253 //uint64_t GLOBAL_SourceSize; //btree matchfinder needed, the file in-memory itself
16,254 //char *GLOBAL_CurrentPositionForReading_TAILforLookAhead; //btree matchfinder needed, updated on each position, step 1, that is
16,255
16,256 char wrd[LongestLineInclusive+1+8];
16,257 int jj, mm;
16,258 unsigned long wrdlen;
16,259 unsigned long long FoundInLinkedList, Slot; //r.18
16,260 uint64_t GettingIndexFromArray;
16,261 int KeySize;
16,262 unsigned long long PseudoLinkedPointer_64;
16,263 unsigned long long PseudoLinkedPointerAUX_64;
16,264 int strFLAG;
16,265 unsigned long StackPtr;
16,266 unsigned long long BSTstack [8192*3];
16,267 char SomeByte;
16,268 char *TwoDigitHEXlist[256] = {
16,269 "00\0",
16,270 "01\0",
16,271 "02\0",
16,272 "03\0",
16,273 "04\0",
16,274 "05\0",
16,275 "06\0",
16,276 "07\0",
16,277 "08\0",
16,278 "09\0",
16,279 "0A\0",
16,280 "0B\0",
16,281 "0C\0",
16,282 "0D\0",
16,283 "0E\0",
16,284 "0F\0",
16,285 "10\0",
16,286 "11\0",
16,287 "12\0",
16,288 "13\0",
16,289 "14\0",
16,290 "15\0",
16,291 "16\0",
16,292 "17\0",
16,293 "18\0",
16,294 "19\0",
16,295 "1A\0",
16,296 "1B\0",
16,297 "1C\0",
16,298 "1D\0",
16,299 "1E\0",
16,300 "1F\0",
16,301 "20\0",
16,302 "21\0",
16,303 "22\0",
16,304 "23\0",
16,305 "24\0",
16,306 "25\0",
16,307 "26\0",
16,308 "27\0",
16,309 "28\0",
16,310 "29\0",
```

16,311 "2A\0",
16,312 "2B\0",
16,313 "2C\0",
16,314 "2D\0",
16,315 "2E\0",
16,316 "2F\0",
16,317 "30\0",
16,318 "31\0",
16,319 "32\0",
16,320 "33\0",
16,321 "34\0",
16,322 "35\0",
16,323 "36\0",
16,324 "37\0",
16,325 "38\0",
16,326 "39\0",
16,327 "3A\0",
16,328 "3B\0",
16,329 "3C\0",
16,330 "3D\0",
16,331 "3E\0",
16,332 "3F\0",
16,333 "40\0",
16,334 "41\0",
16,335 "42\0",
16,336 "43\0",
16,337 "44\0",
16,338 "45\0",
16,339 "46\0",
16,340 "47\0",
16,341 "48\0",
16,342 "49\0",
16,343 "4A\0",
16,344 "4B\0",
16,345 "4C\0",
16,346 "4D\0",
16,347 "4E\0",
16,348 "4F\0",
16,349 "50\0",
16,350 "51\0",
16,351 "52\0",
16,352 "53\0",
16,353 "54\0",
16,354 "55\0",
16,355 "56\0",
16,356 "57\0",
16,357 "58\0",
16,358 "59\0",
16,359 "5A\0",
16,360 "5B\0",
16,361 "5C\0",
16,362 "5D\0",
16,363 "5E\0",
16,364 "5F\0",
16,365 "60\0",
16,366 "61\0",
16,367 "62\0",
16,368 "63\0",

16,369 "64\0",
16,370 "65\0",
16,371 "66\0",
16,372 "67\0",
16,373 "68\0",
16,374 "69\0",
16,375 "6A\0",
16,376 "6B\0",
16,377 "6C\0",
16,378 "6D\0",
16,379 "6E\0",
16,380 "6F\0",
16,381 "70\0",
16,382 "71\0",
16,383 "72\0",
16,384 "73\0",
16,385 "74\0",
16,386 "75\0",
16,387 "76\0",
16,388 "77\0",
16,389 "78\0",
16,390 "79\0",
16,391 "7A\0",
16,392 "7B\0",
16,393 "7C\0",
16,394 "7D\0",
16,395 "7E\0",
16,396 "7F\0",
16,397 "80\0",
16,398 "81\0",
16,399 "82\0",
16,400 "83\0",
16,401 "84\0",
16,402 "85\0",
16,403 "86\0",
16,404 "87\0",
16,405 "88\0",
16,406 "89\0",
16,407 "8A\0",
16,408 "8B\0",
16,409 "8C\0",
16,410 "8D\0",
16,411 "8E\0",
16,412 "8F\0",
16,413 "90\0",
16,414 "91\0",
16,415 "92\0",
16,416 "93\0",
16,417 "94\0",
16,418 "95\0",
16,419 "96\0",
16,420 "97\0",
16,421 "98\0",
16,422 "99\0",
16,423 "9A\0",
16,424 "9B\0",
16,425 "9C\0",
16,426 "9D\0",

16,427 "9E\0",
16,428 "9F\0",
16,429 "A0\0",
16,430 "A1\0",
16,431 "A2\0",
16,432 "A3\0",
16,433 "A4\0",
16,434 "A5\0",
16,435 "A6\0",
16,436 "A7\0",
16,437 "A8\0",
16,438 "A9\0",
16,439 "AA\0",
16,440 "AB\0",
16,441 "AC\0",
16,442 "AD\0",
16,443 "AE\0",
16,444 "AF\0",
16,445 "B0\0",
16,446 "B1\0",
16,447 "B2\0",
16,448 "B3\0",
16,449 "B4\0",
16,450 "B5\0",
16,451 "B6\0",
16,452 "B7\0",
16,453 "B8\0",
16,454 "B9\0",
16,455 "BA\0",
16,456 "BB\0",
16,457 "BC\0",
16,458 "BD\0",
16,459 "BE\0",
16,460 "BF\0",
16,461 "C0\0",
16,462 "C1\0",
16,463 "C2\0",
16,464 "C3\0",
16,465 "C4\0",
16,466 "C5\0",
16,467 "C6\0",
16,468 "C7\0",
16,469 "C8\0",
16,470 "C9\0",
16,471 "CA\0",
16,472 "CB\0",
16,473 "CC\0",
16,474 "CD\0",
16,475 "CE\0",
16,476 "CF\0",
16,477 "D0\0",
16,478 "D1\0",
16,479 "D2\0",
16,480 "D3\0",
16,481 "D4\0",
16,482 "D5\0",
16,483 "D6\0",
16,484 "D7\0",

```
16,485 "D8\0",
16,486 "D9\0",
16,487 "DA\0",
16,488 "DB\0",
16,489 "DC\0",
16,490 "DD\0",
16,491 "DE\0",
16,492 "DF\0",
16,493 "E0\0",
16,494 "E1\0",
16,495 "E2\0",
16,496 "E3\0",
16,497 "E4\0",
16,498 "E5\0",
16,499 "E6\0",
16,500 "E7\0",
16,501 "E8\0",
16,502 "E9\0",
16,503 "EA\0",
16,504 "EB\0",
16,505 "EC\0",
16,506 "ED\0",
16,507 "EE\0",
16,508 "EF\0",
16,509 "F0\0",
16,510 "F1\0",
16,511 "F2\0",
16,512 "F3\0",
16,513 "F4\0",
16,514 "F5\0",
16,515 "F6\0",
16,516 "F7\0",
16,517 "F8\0",
16,518 "F9\0",
16,519 "FA\0",
16,520 "FB\0",
16,521 "FC\0",
16,522 "FD\0",
16,523 "FE\0",
16,524 "FF\0"
16,525 };
16,526 //btree ]
16,527
16,528 char * di;
16,529 int LongestCommonPrefix; // 2020-Jun-30
16,530
16,531 // Also, finally it is time to fix the stupid offset (blind for files smaller than the current window) stupidity for small files:
16,532 // Simply assign 'refStart' if it is within the current window i.e. between e.g. 'refEnd-(256*8*128-1)' and 'refEnd':
16,533
16,534 // Nasty bug fixed (pointer getting negative) only here, to be fixed in all the rest variants [
16,535 /*
16,536 if ( refStart >= refEnd-(2048*8*128-1) ) refStartHOT = refStart;          //--!
16,537 if ( refStart >= refEnd-(256*8*128-1) ) refStartHOTTER = refStart;       //--!
16,538 if ( refStart >= refEnd-(2*4*8*128-1) ) refStartHOTTEST = refStart;      //--!
16,539 if ( refStart >= refEnd-(512*1024*8*128-1) ) refStartCOLDERbig = refStart;  //--!
16,540                                     // \ /
16,541 */
16,542 // Nasty bug fixed (pointer getting negative) only here, to be fixed in all the rest variants ]
```

```

16,543
16,544 //      if ( refStart >= refEnd-(256*8*128-1) ) refStartHOT = refStart;
16,545 //      if ( refStart >= refEnd-(4*8*128-1) ) refStartHOTTER = refStart;
16,546 //      if ( refStart >= refEnd-(16-1) ) refStartHOTEST = refStart;
16,547 //      if ( refStart >= refEnd-(1024*8*128-1) ) refStartCOLDERbig = refStart;
16,548 //      if ( refStart >= refEnd-(2048*8*128-1) ) refStartCOLDERERbig = refStart;
16,549
16,550 if ( (256*8*128-1) >= refEnd-refStart ) refStartHOT = refStart;
16,551 if ( (4*8*128-1) >= refEnd-refStart ) refStartHOTTER = refStart;
16,552 if ( (8*8*128-1) >= refEnd-refStart ) refStartHOTTERryuu = refStart;
16,553 if ( (16-1) >= refEnd-refStart ) refStartHOTEST = refStart;
16,554 if ( (1024*8*128-1) >= refEnd-refStart ) refStartCOLDERbig = refStart;
16,555 if ( (2*1024*8*128-1) >= refEnd-refStart ) refStartCOLDERbigryuu = refStart;
16,556 if ( (2048*8*128-1) >= refEnd-refStart ) refStartCOLDERERbig = refStart;
16,557
16,558 if ( (8192*8*128-1) >= refEnd-refStart ) refStartCOLDERERERbig = refStart;
16,559 if ( (512*1024*8*128-1) >= refEnd-refStart ) refStartCOLDERERERERbig = refStart;
16,560 if ( (128LL*2*512*1024*8*128-1) >= refEnd-refStart ) refStartCOLDERERERERbigryuu = refStart;
16,561
16,562 //printf("%d\n", refStartCOLDERbig); //debug
16,563 //printf("%p\n", refStartCOLDERbig); //debug
16,564
16,565 if ( (16-1) >= refEnd-refStart ) refStartSW1 = refStart;
16,566 if ( (4*8*128-1) >= refEnd-refStart ) refStartSW2 = refStart;
16,567 if ( (8*8*128-1) >= refEnd-refStart ) refStartSW2ryuu = refStart;
16,568 if ( (1024*8*128-1) >= refEnd-refStart ) refStartSW3 = refStart;
16,569 if ( (2*1024*8*128-1) >= refEnd-refStart ) refStartSW3ryuu = refStart;
16,570 if ( (256*1024*8*128-1) >= refEnd-refStart ) refStartSW4 = refStart;
16,571 if ( (512*1024*8*128-1) >= refEnd-refStart ) refStartSW5 = refStart;
16,572 if ( (128LL*2*512*1024*8*128-1) >= refEnd-refStart ) refStartSW5ryuu = refStart;
16,573 if ( (1024LL*2*512*1024*8*128-1) >= refEnd-refStart ) refStartSW6ryuu = refStart;
16,574 if ( (16*1024*8*128-1) >= refEnd-refStart ) refStartSW1a = refStart;
16,575 if ( (64*8*128-1) >= refEnd-refStart ) refStartSW1b = refStart;
16,576 if ( (256-1) >= refEnd-refStart ) refStartSW1c = refStart;
16,577
16,578 if ( (256*4-1) >= refEnd-refStart ) refStartSW1d = refStart;
16,579 if ( (256*8-1) >= refEnd-refStart ) refStartSW1d2 = refStart;
16,580 if ( (64*1024*8*128-1) >= refEnd-refStart ) refStartSW1e = refStart;
16,581
16,582 if ( (512-1) >= refEnd-refStart ) refStartSW512 = refStart;
16,583 if ( (128*8*128-1) >= refEnd-refStart ) refStartSW128kb = refStart;
16,584
16,585 // Sizewise/MatchLenWise priority:
16,586 //      4:1=4          (16B) #01 320: (5+1)+1+1+1=32      (1TB)
16,587 //      8:1=8          (16B) #02 256: (5+1)+1+1+1= 28.4  (1TB)
16,588 //     12:1=12         (16B) #03 192: (5+1)+1+1= 24      (1TB)
16,589 //    16:1=Flag        (16B) #04 128: (5+1)+1= 18.2    (1TB)
16,590 //     4:2=2          (4KB) #05 30:2= 15      (512B)
16,591 //     6:2=3          (512B) #06 56: (2+1)+1= 14      (8KB)
16,592 //     8:2=4          (4KB) #07 24:2= 12      (4KB)
16,593 //    12:2=6          (4KB) #08 12:1= 12      (16B)
16,594 //    14:2=7          (512B) #09 56: (3+1)+1= 11.2    (2MB)
16,595 //    16:2=8          (4KB) #10 22:2= 11      (512B)
16,596 //    22:2=11         (512B) #11 64: (5+1)= 10.6    (1TB)
16,597 //    24:2=12         (4KB) #12 30:3= 10      (128KB)
16,598 //    30:2=15         (512B) #13 56: (4+1)+1= 9.3    (512MB)
16,599 //     4:3=1.3        (1MB) #14 28: (2+1)= 9.3      (8KB)
16,600 //     6:3=2          (128KB) #15 36: (2+1)+1= 9      (8KB)

```

```

16,601 //      8:3=2.6          (1MB) #16 56:(5+1)+1=      8 (128GB)
16,602 //     12:3=4           (1MB) #17 24:3=          8 (1MB)
16,603 //    14:3=4.6         (128KB) #18 16:2=          8 (4KB)
16,604 //    16:3=5.3         (1MB) #19  8:1=          8 (16B)
16,605 //   18:(2+1)=6        (8KB) #20 22:3=         7.3 (128KB)
16,606 //   22:3=7.3        (128KB) #21 36:(3+1)+1=     7.2 (2MB)
16,607 //   24:3=8          (1MB) #22 28:(3+1)=        7 (2MB)
16,608 //   28:(2+1)=9.3     (8KB) #23 14:2=          7 (512B)
16,609 //   30:3=10         (128KB) #24 36:(4+1)+1=     6 (512MB)
16,610 //    6:4=1.5         (16MB) #25 24:4=          6 (256MB)
16,611 //    8:4=2          (16MB) #26 18:(2+1)=        6 (8KB)
16,612 //   10:4=2.5        (16MB) #27 12:2=          6 (4KB)
16,613 //   12:4=3          (16MB) #28 28:(4+1)=       5.6 (512MB)
16,614 //   14:4=3.5        (16MB) #29 16:3=          5.3 (1MB)
16,615 //   16:4=4          (16MB)F #30 36:(5+1)+1=     5.1 (128GB)
16,616 //   18:(3+1)=4.5     (2MB) #31 28:(5+1)=        4.6 (128GB)
16,617 //   24:4=6          (256MB) #32 14:3=          4.6 (128KB)
16,618 //   28:(3+1)=7      (2MB) #33 18:(3+1)=        4.5 (2MB)
16,619 //   36:(2+1)+1=9     (8KB) #34 16:4=          4 (16MB)F
16,620 //   56:(2+1)+1=14   (8KB) #35 12:3=          4 (1MB)
16,621 //   18:(4+1)=3.6    (512MB) #36  8:2=          4 (4KB)
16,622 //   28:(4+1)=5.6    (512MB) #37  4:1=          4 (16B)
16,623 //   36:(3+1)+1=7.2  (2MB) #38 18:(4+1)=        3.6 (512MB)
16,624 //   56:(3+1)+1=11.2 (2MB) #39 14:4=          3.5 (16MB)
16,625 //   18:(5+1)=3      (128GB) #40 18:(5+1)=        3 (128GB)
16,626 //   28:(5+1)=4.6    (128GB) #41 12:4=          3 (16MB)
16,627 //   36:(4+1)+1=6    (512MB) #42  6:2=          3 (512B)
16,628 //   56:(4+1)+1=9.3  (512MB) #43  8:3=          2.6 (1MB)
16,629 //   64:(5+1)=10.6   (1TB) #44 10:4=          2.5 (16MB)
16,630 //   36:(5+1)+1=5.1   (128GB) #45  8:4=          2 (16MB)
16,631 //   56:(5+1)+1=8     (128GB) #46  6:3=          2 (128KB)
16,632 //  128:(5+1)+1=18.2  (1TB) #47  4:2=          2 (4KB)
16,633 //  192:(5+1)+1=24    (1TB) #48  6:4=          1.5 (16MB)
16,634 //  256:(5+1)+1+1+1=28.4 (1TB) #49  4:3=          1.3 (1MB)
16,635 // 320:(5+1)+1+1+1+1=32 (1TB)
16,636
16,637 *HowManyDittoTagsToEmit = 0;
16,638
16,639 // *96:2+1+1+1=19.2      (4KB)
16,640 // *72:2+1+1=18         (4KB)
16,641 // *96:3+1+1+1=16     (1MB)
16,642 // *48:2+1=16         (4KB)
16,643 // *72:3+1+1=14.4     (1MB)
16,644 // *96:4+1+1+1=13.7   (256MB)
16,645 // *72:4+1+1=12       (256MB)
16,646 // *48:3+1=12         (1MB)
16,647 // *48:4+1=9.6        (256MB)
16,648
16,649 // *90:2+1+1=22.5       (512B)
16,650 // *120:3+1+1+1=20     (128KB)
16,651 // *60:2+1=20          (512B)
16,652 // *90:3+1+1=18       (128KB)
16,653 // *60:3+1=15         (128KB)
16,654 // *44:2+1=14.6       (512B)
16,655 // *44:3+1=11         (128KB)
16,656
16,657 goto Jan25;
16,658 //btree [

```



```

16,659
16,660 while ( GLOBAL_CurrentPositionForReading_TAILforLookAhead <= encStart ) { // TAIL should become LookAhead (i.e. encStart)
16,661   for (jj=0; jj< MatchLensNUM; jj++) {
16,662     LastSeenOffset_PseudoPointer[jj] = 0; // By Default - Not Seen
16,663     if ( GLOBAL_CurrentPositionForReading_TAILforLookAhead + MatchLens[jj] -1 <= GLOBAL_SourceBlock + GLOBAL_SourceSize -1 ) {
16,664
16,665       // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed [
16,666       //         wrdlen=0;
16,667       //         memset( &wrd[0], 0, (LongestLineInclusive+1+8) ); //r.18, see below, the old nullifier is commented.
16,668       //         for (mm=0; mm< MatchLens[jj]; mm++) {
16,669       //             memcpy( &wrd[ wrdlen ], TwoDigitHEXlist[(unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead+mm)], 2 );
16,670       //             wrdlen++;
16,671       //             wrdlen++;
16,672       //         }
16,673       // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed ]
16,674
16,675       wrdlen=MatchLens[jj];
16,676
16,677 #if defined(_NSHA3) || defined(_NPRV) || defined(_NDD) || defined(_NAquaHash) //2020-Nov-20
16,678
16,679 #ifdef _NPRV //2020-Nov-12 [ Consider not only 256[+] matchlens but 24[+] as well - in order to save memory!
16,680   if (wrdlen > MatchLenAboveWhichHASHkicksin) {
16,681     prvhash42( (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen, (unsigned char *) PRVhash, PRVhashlenInBYTES, 0, 0, 0 );
16,682     //2020-Nov-17, for v2.0 " , 0" was added
16,683     wrdlen=MatchLenAboveWhichHASHkicksin; // very aggressive is 16B or 128b, if collisions happen then increase to 20B or 160b
16,684     memcpy( &wrd[ 0+1 ], (unsigned char *) PRVhash, wrdlen );
16,685   } else
16,686     memcpy( &wrd[ 0+1 ], (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen );
16,687 #endif
16,688 #ifdef _NSHA3
16,689   //if (wrdlen >= 256) {
16,690   if (wrdlen > MatchLenAboveWhichHASHkicksin) {
16,691     FIPS202_SHA3_224((unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen, (unsigned char *) SHA328bytes);
16,692     wrdlen=MatchLenAboveWhichHASHkicksin; //2020-Nov-12
16,693     memcpy( &wrd[ 0+1 ], (unsigned char *) SHA328bytes, wrdlen );
16,694   } else
16,695     memcpy( &wrd[ 0+1 ], (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen );
16,696 #endif
16,697 #ifdef _NDD
16,698   //if (wrdlen >= 256) { // 2020-Jun-13
16,699   if (wrdlen > MatchLenAboveWhichHASHkicksin) {
16,700     //*(uint64_t*)&DD[0] = crc64((unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen);
16,701     //*(uint32_t*)&DD[8] = crc32c_sw((unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen);
16,702     // Above 2 lines are glued into next one:
16,703     DoubleDeuce( (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen );
16,704     wrdlen=MatchLenAboveWhichHASHkicksin; //2020-Nov-12
16,705     memcpy( &wrd[ 0+1 ], (unsigned char *) DD, wrdlen );
16,706   } else
16,707     memcpy( &wrd[ 0+1 ], (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen );
16,708 #endif
16,709 #ifdef _NAquaHash
16,710   //if (wrdlen >= 256) { // 2020-Jun-13
16,711   if (wrdlen > MatchLenAboveWhichHASHkicksinAQUA) {
16,712     //*(uint64_t*)&DD[0] = crc64((unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen);
16,713     //*(uint32_t*)&DD[8] = crc32c_sw((unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen);
16,714     // Above 2 lines are glued into next one:
16,715     DoubleDeuceAES_Gumbotron_YMM( (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen );

```

```
16,716 wrdlen=MatchLenAboveWhichHASHkicksinAQUA; //2020-Nov-12
16,717 memcpy( &wrd[ 0+1 ], (unsigned char *)DDAES, wrdlen );
16,718
16,719 } else
16,720 memcpy( &wrd[ 0+1 ], (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen );
16,721 #endif
16,722
16,723 #else
16,724 memcpy( &wrd[ 0+1 ], (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen );
16,725 #endif
16,726 memcpy( &wrd[ 0 ], &wrdlen, 1 ); // The keysize is 1..255, pretty enough since my needs are 1..28 (up to SHA3-224).
16,727 /*
16,728 #ifdef _NSHA3
16,729 // if (wrdlen < 28)
16,730 if (wrdlen != 28) // 2020-Jun-13
16,731 memcpy( &wrd[ 0+1 ], (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen );
16,732 else
16,733 memcpy( &wrd[ 0+1 ], (unsigned char *) SHA328bytes, wrdlen );
16,734 #else
16,735 // if (wrdlen < 28)
16,736 memcpy( &wrd[ 0+1 ], (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen );
16,737 // else
16,738 // memcpy( &wrd[ 0+1 ], (unsigned char *) SHA328bytes, wrdlen );
16,739 #endif
16,740 */
16,741 //Slot = FNV1A_Hash_Jesteress_27bit(wrd +1, wrdlen);
16,742 Slot = FNV1A_Pippip_Yurii(wrd +1, wrdlen); // Changed 2019-Nov-28
16,743
16,744 Slot = Slot<<3;
16,745 // NEW NEW NEW [ //r.18
16,746 // In here Slot is not within a single pool but in MatchLensNUM sub-pools i.e. MatchLensNUM-way i.e. MatchLensNUM hashpots:
16,747 /*
16,748 for (GettingIndexOfArray=0; GettingIndexOfArray<MatchLensNUM; GettingIndexOfArray++) {
16,749 // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed [
16,750 if ( (wrdlen>>1)==MatchLens[GettingIndexOfArray] ) break;
16,751 if ( (wrdlen)>>1)==MatchLens[GettingIndexOfArray] ) break;
16,752 // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed [
16,753 }
16,754 */
16,755 GettingIndexOfArray=jj; // same as above atrocity
16,756 Slot = Slot +(GettingIndexOfArray*(1LL<<HashInBITS_GLOBAL)*8); // CAUTION: 'wrdlen' is halved here, when a new revision comes with 1:1 keysize then change it.
16,757 // NEW NEW NEW [ //r.18
16,758 //KeySize = LongestLineInclusive; // 'LongestLineInclusive' is for 1-way hashing where different KeySizes can occupy same HASH pool.
16,759 KeySize = wrdlen; //r.18, In case multi-way hashing: either the MAX allowed or the 'wrdlen'.
16,760
16,761 //Slot = 0; // One Tree only!
16,762 memcpy( &PseudoLinkedPointer_64, GLOBAL_HASHPOT+Slot, 8 );
16,763 // 1] Search [ Read the 8 bytes field into LastSeenOffset_PseudoPointer[jj]
16,764
16,765
16,766
16,767
16,768
16,769
16,770 if (PseudoLinkedPointer_64 == 0) // means EMPTY-SLOT
16,771 {
16,772 // Impossible, they all have already been inserted...
16,773 }
```

```

16,774         else // means USED-SLOT
16,775         { FoundInLinkedList = 0;
16,776         StackPtr = 0;
16,777         //         while (PseudoLinkedPointer != 0 && FoundInLinkedList == 0)
16,778         //         while (PseudoLinkedPointer_64 != 0 && FoundInLinkedList == 0)
16,779         {
16,780         // ***** 'P W P' section [
16,781         // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
16,782         // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
16,783         // here ALWAYS LW exists: no need for existence check - line below
16,784         // if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
16,785         // if (memcmp(PseudoLinkedPointer+4+4+4,wordlen) > 0) // go LP
16,786         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
16,787         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
16,788         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
16,789         //fread(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
16,790         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
16,791         // [ //r.14+
16,792         PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
16,793         if (BSTorBtree == 2) {
16,794         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
16,795         fread(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG);
16,796         } else { // ##### 64bit memory manipulations [
16,797         memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64, 8+8+2*(KeySize+1+8) );
16,798         } // ##### 64bit memory manipulations ]
16,799         memcpy( &FourGramL[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
16,800         // ] //r.14+
16,801         //strFLAG=strcmpKAZE13(FourGramL, wrd);
16,802         strFLAG=memcmp(FourGramL+1, wrd+1, wordlen); // Since 2019-Feb-07, it is not ASCIIZ string but zero byte is the length of the 1..
16,803         if (strFLAG > 0) // go LP
16,804         // { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 0, 4 ); //LP
16,805         //     PseudoLinkedPointer = PseudoLinkedPointerNEW;
16,806         // }
16,807         {
16,808         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
16,809         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 0; //LP
16,810         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
16,811         if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
16,812         BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
16,813         BSTstack[StackPtr] = 0; ++StackPtr; //LPOffset=0;MPOffset=8;RPOffset=16;
16,814         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
16,815         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
16,816         //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
16,817         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
16,818         // [ //r.14+
16,819         memcpy( &PseudoLinkedPointer_64, &LEAF[0], 8 );
16,820         // ] //r.14+
16,821         }
16,822         // else if (memcmp(PseudoLinkedPointer+4+4+4,wordlen) < 0) // go RP or MP
16,823         else if (strFLAG < 0) // go RP or MP
16,824         { // RW existence check - line below:
16,825         // if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 ) // RW exists
16,826         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
16,827         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
16,828         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
16,829         //fread(&SomeByte, 1, 1, fp_outRG);
16,830         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
16,831         // [ //r.14+

```

```

16,832         memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (KeySize+1+8)], 1 );
16,833         // ] //r.14+
16,834         if (SomeByte != 0 ) // RW exists
16,835             { // Here all 'P W P' section is repeated; the way of handling case when dynamic number of words in leaf
16,836 // *****
16,837 // ***** 'P W P' section 2 [
16,838 // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
16,839 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
16,840         // here ALWAYS RW exists: no need for existence check - line below
16,841         // if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
16,842 //         if (memcmp(PseudoLinkedPointer+4+4+4+wordlen, wrd, wordlen) > 0) // go MP
16,843         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
16,844         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
16,845         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
16,846         //fread(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
16,847         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
16,848         // [ //r.14+
16,849         memcpy( &FourGramL[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
16,850         // ] //r.14+
16,851         //strFLAG=strcmpKAZE13(FourGramL, wrd);
16,852         strFLAG=memcmp(FourGramL+1, wrd+1, wordlen); // Since 2019-Feb-07, it is not ASCIIZ string but zero byte is the length of the 1..
16,853         if (strFLAG > 0) // go MP
16,854 //             { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
16,855 //             PseudoLinkedPointer = PseudoLinkedPointerNEW;
16,856 //             }
16,857         {
16,858         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
16,859         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8; //MP
16,860         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
16,861         if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
16,862         BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
16,863         BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
16,864         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
16,865         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
16,866         //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
16,867         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
16,868         // [ //r.14+
16,869         memcpy( &PseudoLinkedPointer_64, &LEAF[8], 8 );
16,870         // ] //r.14+
16,871         }
16,872 //         else if (memcmp(PseudoLinkedPointer+4+4+4+wordlen, wrd, wordlen) < 0) // go RP
16,873         else if (strFLAG < 0) // go RP
16,874 //             { // No ?W after RW - go RP
16,875 //             memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4 + 4, 4 ); //RP
16,876 //             PseudoLinkedPointer = PseudoLinkedPointerNEW;
16,877 //             }
16,878         {
16,879         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
16,880         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8; //RP
16,881         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
16,882         if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
16,883         BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
16,884         BSTstack[StackPtr] = 16; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
16,885         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
16,886         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
16,887         //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
16,888         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
16,889         // [ //r.14+

```

```

16,890         memcpy( &PseudoLinkedPointer_64, &LEAF[8 + 8], 8 );
16,891         // ] //r.14+
16,892     }
16,893     else { FoundInLinkedList = 1; // wrd is RW
16,894     // Counter [
16,895         if (BSTorBtree == 2) {
16,896             fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
16,897         }
16,898         //memcpy( &CounterOccurrences, &FourGramL[(KeySize+1+4)-4], 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
16,899         memcpy( &LastSeenOffset_PseudoPointer[jj], &FourGramL[(KeySize+1+8)-8], 8 );
16,900         //if (CounterOccurrences<999999999) CounterOccurrences++; // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
16,901         //memcpy( &FourGramL[(KeySize+1+4)-4], &CounterOccurrences, 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
16,902         memcpy( &FourGramL[(KeySize+1+8)-8], &GLOBAL_CurrentPositionForReading_TAILforLookAhead, 8 ); // 2020-Jun-30: In 32bit code it is 4 not 8:
printf("%d",sizeof(char*));
16,903         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
16,904         //fwrite(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
16,905         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
16,906         // [ //r.14+
16,907         memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &FourGramL[0], (KeySize+1+8) );
16,908         if (BSTorBtree == 2) {
16,909             fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG);
16,910         } else { // ##### 64bit memory manipulations [
16,911             memcpy( (char *)&PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
16,912         } // ##### 64bit memory manipulations ]
16,913         // ] //r.14+
16,914         // Counter ]
16,915     }
16,916 // ***** 'P W P' section 2 ]
16,917 // *****
16,918     }
16,919     else // RW empty - go MP
16,920     {
16,921         memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
16,922         PseudoLinkedPointer = PseudoLinkedPointerNEW;
16,923     }
16,924     // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
16,925     //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8; //MP
16,926     // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
16,927     if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
16,928     BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
16,929     BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
16,930     // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
16,931     //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
16,932     //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
16,933     // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
16,934     // [ //r.14+
16,935     memcpy( &PseudoLinkedPointer_64, &LEAF[8], 8 );
16,936     // ] //r.14+
16,937     }
16,938     }
16,939     else { FoundInLinkedList = 1; // wrd is LW
16,940     // Counter [
16,941         if (BSTorBtree == 2) {
16,942             fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
16,943         }
16,944         //memcpy( &CounterOccurrences, &FourGramL[(KeySize+1+4)-4], 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
16,945         memcpy( &LastSeenOffset_PseudoPointer[jj], &FourGramL[(KeySize+1+8)-8], 8 );
16,946         //if (CounterOccurrences<999999999) CounterOccurrences++; // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset

```

```

16,947 //memcpy( &FourGramL[(KeySize+1+4)-4], &CounterOccurrences, 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
16,948 memcpy( &FourGramL[(KeySize+1+8)-8], &GLOBAL_CurrentPositionForReading_TAILforLookAhead, 8 ); // 2020-Jun-30: In 32bit code it is 4 not 8:
printf("%d",sizeof(char*));
16,949 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
16,950 //fwrite(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
16,951 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
16,952 // [ //r.14+
16,953 memcpy( &LEAF[8 + 8 + 8], &FourGramL[0], (KeySize+1+8) );
16,954 if (BSTorBtree == 2) {
16,955 fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG);
16,956 } else { // ##### 64bit memory manipulations [
16,957 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
16,958 } // ##### 64bit memory manipulations ]
16,959 // ] //r.14+
16,960 // Counter ]
16,961 }
16,962 // ***** 'P W P' section ]
16,963 } // while
16,964 }
16,965
16,966
16,967
16,968 // 1] Search ] Write (uint64_t)(GLOBAL_CurrentPositionForReading_TAILforLookAhead) into the 8 bytes field
16,969 }
16,970 } //for (jj=0;
16,971
16,972 // Debug the ratio difference [
16,973 // Save the first value, in order to start from zero:
16,974 //if (GLOBAL_CurrentPositionForReading_TAILforLookAhead_ZERO_debug == NULL) GLOBAL_CurrentPositionForReading_TAILforLookAhead_ZERO_debug =
GLOBAL_CurrentPositionForReading_TAILforLookAhead;
16,975 //
16,976 //printf("\nGLOBAL_CurrentPositionForReading_TAILforLookAhead = %s (catching up with encStart = %s): ",
_ui64toaKAZEzerocomma4((uint64_t)GLOBAL_CurrentPositionForReading_TAILforLookAhead-(uint64_t)GLOBAL_CurrentPositionForReading_TAILforLookAhead_ZERO_debug, 11ToaDigits2,
16)+(26-8-1), _ui64toaKAZEzerocomma4((uint64_t)encStart-(uint64_t)GLOBAL_CurrentPositionForReading_TAILforLookAhead_ZERO_debug, 11ToaDigits3, 16)+(26-8-1) );
16,977 // for (i=0; i< MatchLensNUM; i++) {
16,978 // if (LastSeenOffset_PseudoPointer[i] == 0)
16,979 // printf(" %s ", _ui64toaKAZEzerocomma4(LastSeenOffset_PseudoPointer[i], 11ToaDigits2, 16)+(26-8-1) );
16,980 // else
16,981 // printf(" %s ", _ui64toaKAZEzerocomma4(LastSeenOffset_PseudoPointer[i] -(uint64_t)GLOBAL_CurrentPositionForReading_TAILforLookAhead_ZERO_debug,
11ToaDigits2, 16)+(26-8-1) );
16,982 // }
16,983 //printf("\n");
16,984 // Debug the ratio difference ]
16,985
16,986 GLOBAL_CurrentPositionForReading_TAILforLookAhead++;
16,987 } // At the end of this loop we should have (in LastSeenOffset_PseudoPointer[]) all matches found for position 'encStart'.
16,988
16,989 //btree ]
16,990
16,991
16,992 Jan25:
16,993 //btree [ 2020-Jan-25, improved updating - not causing fallbacks to memmem() by introducing GLOBAL_CurrentPositionForReading_TAILforLookAhead_ARRAY[]
16,994
16,995 // step #1 of 2: Updating up to the end of Sliding Window
16,996
16,997 for (jj=0; jj< MatchLensNUMdummy; jj++) { // 2020-Jun-28
16,998 LastSeenOffset_PseudoPointer[jj] = 0; // By Default - Not Seen
16,999 }

```

```
17,000
17,001 for (jj=0; jj< MatchLensNUM; jj++) {
17,002     LastSeenOffset_PseudoPointer[jj] = 0; // By Default - Not Seen
17,003 //while ( GLOBAL_CurrentPositionForReading_TAILforLookAhead <= encStart ) { // TAIL should become LookAhead (i.e. encStart)
17,004 while ( GLOBAL_CurrentPositionForReading_TAILforLookAhead_ARRAY[jj] + MatchLens[jj] -1 <= encStart -1 ) { // TAIL should cover all positions up to the end of Sliding
Window i.e. GLUED to the LookAheadBuffer i.e. EncStart ->...][EncStart
17,005     //if ( GLOBAL_CurrentPositionForReading_TAILforLookAhead_ARRAY[jj] + MatchLens[jj] -1 <= GLOBAL_SourceBlock + GLOBAL_SourceSize -1 ) {
17,006
17,007     // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed [
17,008 //         wrdlen=0;
17,009 //         memset( &wrd[0], 0, (LongestLineInclusive+1*8) ); //r.18, see below, the old nullifier is commented.
17,010 //         for (mm=0; mm< MatchLens[jj]; mm++) {
17,011 //             memcpy( &wrd[ wrdlen ], TwoDigitHEXlist[(unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead+mm)], 2 );
17,012 //             wrdlen++;
17,013 //             wrdlen++;
17,014 //         }
17,015 // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed ]
17,016
17,017     wrdlen=MatchLens[jj];
17,018
17,019 #if defined(_NSHA3) || defined(_NPRV) || defined(_NDD) || defined(_NAquaHash) //2020-Nov-20
17,020
17,021 #ifdef _NPRV //2020-Nov-12 [ Consider not only 256[+] matchlens but 24[+] as well - in order to save memory!
17,022     if (wrdlen > MatchLenAboveWhichHASHkicksin) {
17,023         prvhash42( (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead_ARRAY[jj]), wrdlen, (unsigned char *) PRVhash, PRVhashlenInBYTES, 0, 0,
0 ); //2020-Nov-17, for v2.0 ", 0" was added
17,024         wrdlen=MatchLenAboveWhichHASHkicksin; // very aggressive is 16B or 128b, if collisions happen then increase to 20B or 160b
17,025         memcpy( &wrd[ 0+1 ], (unsigned char *) PRVhash, wrdlen );
17,026     } else
17,027         memcpy( &wrd[ 0+1 ], (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead_ARRAY[jj]), wrdlen );
17,028 #endif
17,029 #ifdef _NSHA3
17,030     //if (wrdlen >= 256) {
17,031     if (wrdlen > MatchLenAboveWhichHASHkicksin) {
17,032         FIPS202_SHA3_224((unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead_ARRAY[jj]), wrdlen, (unsigned char *) SHA328bytes);
17,033         wrdlen=MatchLenAboveWhichHASHkicksin; //2020-Nov-12
17,034         memcpy( &wrd[ 0+1 ], (unsigned char *) SHA328bytes, wrdlen );
17,035     } else
17,036         memcpy( &wrd[ 0+1 ], (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead_ARRAY[jj]), wrdlen );
17,037 #endif
17,038 #ifdef _NDD
17,039     //if (wrdlen >= 256) { // 2020-Jun-13
17,040     if (wrdlen > MatchLenAboveWhichHASHkicksin) {
17,041         //(uint64_t*)&DD[0] = crc64((unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead_ARRAY[jj]), wrdlen);
17,042         //(uint32_t*)&DD[8] = crc32c_sw((unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead_ARRAY[jj]), wrdlen);
17,043         // Above 2 lines are glued into next one:
17,044         DoubleDeuce( (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead_ARRAY[jj]), wrdlen );
17,045         wrdlen=MatchLenAboveWhichHASHkicksin; //2020-Nov-12
17,046         memcpy( &wrd[ 0+1 ], (unsigned char *) DD, wrdlen );
17,047     } else
17,048         memcpy( &wrd[ 0+1 ], (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead_ARRAY[jj]), wrdlen );
17,049 #endif
17,050 #ifdef _NAquaHash
17,051     //if (wrdlen >= 256) { // 2020-Jun-13
17,052     if (wrdlen > MatchLenAboveWhichHASHkicksinAQUA) {
17,053         //(uint64_t*)&DD[0] = crc64((unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead_ARRAY[jj]), wrdlen);
17,054         //(uint32_t*)&DD[8] = crc32c_sw((unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead_ARRAY[jj]), wrdlen);
17,055         //(uint32_t*)&DD[8] = crc32c_sw((unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead_ARRAY[jj]), wrdlen);
```

```

17,056 // Above 2 lines are glued into next one:
17,057 DoubleDeuceAES_Gumbotron_YMM( (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead_ARRAY[jj]), wrdlen );
17,058 wrdlen=MatchLenAboveWhichHASHkicksinAQUA; //2020-Nov-12
17,059 memcpy( &wrd[ 0+1 ], (unsigned char *)DDAES, wrdlen );
17,060
17,061 } else
17,062     memcpy( &wrd[ 0+1 ], (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead_ARRAY[jj]), wrdlen );
17,063 #endif
17,064
17,065 #else
17,066     memcpy( &wrd[ 0+1 ], (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead_ARRAY[jj]), wrdlen );
17,067 #endif
17,068     memcpy( &wrd[ 0 ], &wrdlen, 1 ); // The keysize is 1..255, pretty enough since my needs are 1..28 (up to SHA3-224).
17,069 /*
17,070 #ifdef _NSHA3
17,071 //
17,072     if (wrdlen != 28) // 2020-Jun-13
17,073         memcpy( &wrd[ 0+1 ], (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead_ARRAY[jj]), wrdlen );
17,074     else
17,075         memcpy( &wrd[ 0+1 ], (unsigned char *)SHA328bytes, wrdlen );
17,076 #else
17,077 //
17,078     if (wrdlen < 28)
17,079         memcpy( &wrd[ 0+1 ], (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead_ARRAY[jj]), wrdlen );
17,080     else
17,081         memcpy( &wrd[ 0+1 ], (unsigned char *)SHA328bytes, wrdlen );
17,082 */
17,083
17,084 //Slot = FNV1A_Hash_Jesteress_27bit(wrd +1, wrdlen);
17,085 Slot = FNV1A_Pippip_Yurii(wrd +1, wrdlen); // Changed 2019-Nov-28
17,086
17,087 Slot = Slot<<3;
17,088 // NEW NEW NEW [ //r.18
17,089 // In here Slot is not within a single pool but in MatchLensNUM sub-pools i.e. MatchLensNUM-way i.e. MatchLensNUM hashpots:
17,090 /*
17,091 for (GettingIndexOfArray=0; GettingIndexOfArray<MatchLensNUM; GettingIndexOfArray++) {
17,092     // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed [
17,093     // if ( (wrdlen>>1)==MatchLens[GettingIndexOfArray] ) break;
17,094     if ( (wrdlen)==MatchLens[GettingIndexOfArray] ) break;
17,095     // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed ]
17,096 }
17,097 */
17,098 GettingIndexOfArray=jj; // same as above atrocity
17,099 Slot = Slot +(GettingIndexOfArray*(1LL<<HashInBITS_GLOBAL)*8); // CAUTION: 'wrdlen' is halved here, when a new revision comes with 1:1 keysize then change it.
17,100 // NEW NEW NEW [ //r.18
17,101
17,102 //KeySize = LongestLineInclusive; // 'LongestLineInclusive' is for 1-way hashing where different KeySizes can occupy same HASH pool.
17,103 KeySize = wrdlen; //r.18, In case multi-way hashing: either the MAX allowed or the 'wrdlen'.
17,104
17,105 //Slot = 0; // One Tree only!
17,106 memcpy( &PseudoLinkedPointer_64, GLOBAL_HASHPOT+Slot, 8 );
17,107
17,108 // 1] Search [ Read the 8 bytes field into LastSeenOffset_PseudoPointer[jj]
17,109
17,110
17,111     if (PseudoLinkedPointer_64 == 0) // means EMPTY-SLOT
17,112     {
17,113         // Impossible, they all have already been inserted...

```



```

17,114         }
17,115         else // means USED-SLOT
17,116         { FoundInLinkedList = 0;
17,117         StackPtr = 0;
17,118         // while (PseudoLinkedPointer != 0 && FoundInLinkedList == 0)
17,119         while (PseudoLinkedPointer_64 != 0 && FoundInLinkedList == 0)
17,120         {
17,121         // ***** 'P W P' section [
17,122         // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
17,123         // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
17,124         // here ALWAYS LW exists: no need for existence check - line below
17,125         // if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
17,126         // if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wordlen) > 0) // go LP
17,127         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,128         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
17,129         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
17,130         //fread(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
17,131         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,132         // [ //r.14+
17,133         PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
17,134         if (BSTorBtree == 2) {
17,135         fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
17,136         fread(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG);
17,137         } else { // ##### 64bit memory manipulations [
17,138         memcpy( &LEAF[0], (char *)PseudoLinkedPointerAUX_64, 8+8+8+2*(KeySize+1+8) );
17,139         } // ##### 64bit memory manipulations ]
17,140         memcpy( &FourGramL[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
17,141         // ] //r.14+
17,142         //strFLAG=strcmpKAZE13(FourGramL, wrd);
17,143         strFLAG=memcmp(FourGramL +1, wrd +1, wordlen); // Since 2019-Feb-07, it is not ASCIIZ string but zero byte is the length of the 1..
17,144         if (strFLAG > 0) // go LP
17,145         { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 0, 4 ); //LP
17,146         // PseudoLinkedPointer = PseudoLinkedPointerNEW;
17,147         // }
17,148         {
17,149         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,150         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 0; //LP
17,151         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,152         if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
17,153         BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
17,154         BSTstack[StackPtr] = 0; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
17,155         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,156         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
17,157         //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
17,158         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,159         // [ //r.14+
17,160         memcpy( &PseudoLinkedPointer_64, &LEAF[0], 8 );
17,161         // ] //r.14+
17,162         }
17,163         // else if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wordlen) < 0) // go RP or MP
17,164         else if (strFLAG < 0) // go RP or MP
17,165         { // RW existence check - line below:
17,166         // if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 ) // RW exists
17,167         // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,168         //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
17,169         //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
17,170         //fread(&SomeByte, 1, 1, fp_outRG);
17,171         // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!

```

```

17,172          // [ //r.14+
17,173          memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (KeySize+1+8)], 1 );
17,174          // ] //r.14+
17,175          if (SomeByte != 0 ) // RW exists
17,176              { // Here all 'P W P' section is repeated; the way of handling case when dynamic number of words in leaf
17,177          // ***** 'P W P' section 2 [
17,178          // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
17,179          // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
17,180          // here ALWAYS RW exists: no need for existence check - line below
17,181          // if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
17,182          // if (memcmp(PseudoLinkedPointer+4+4+4+wordlen, wrd, wordlen) > 0) // go MP
17,183          // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,184          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);
17,185          //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
17,186          //fread(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
17,187          // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,188          // [ //r.14+
17,189          memcpy( &FourGramL[0], &LEAF[8 + 8 + 8 + (KeySize+1+8)], (KeySize+1+8) );
17,190          // ] //r.14+
17,191          //strFLAG=strcmpKAZE13(FourGramL, wrd);
17,192          strFLAG=memcmp(FourGramL + 1, wrd + 1, wordlen); // Since 2019-Feb-07, it is not ASCIIZ string but zero byte is the length of the 1..
17,193          if (strFLAG > 0) // go MP
17,194              {
17,195                  memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
17,196                  PseudoLinkedPointer = PseudoLinkedPointerNEW;
17,197              }
17,198          {
17,199          // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,200          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8; //MP
17,201          // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,202          if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
17,203          BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
17,204          BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
17,205          // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,206          //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
17,207          //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
17,208          // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,209          // [ //r.14+
17,210          memcpy( &PseudoLinkedPointer_64, &LEAF[8], 8 );
17,211          // ] //r.14+
17,212          }
17,213          // else if (memcmp(PseudoLinkedPointer+4+4+4+wordlen, wrd, wordlen) < 0) // go RP
17,214          else if (strFLAG < 0) // go RP
17,215              { // No ?W after RW - go RP
17,216                  memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4 + 4, 4 ); //RP
17,217                  PseudoLinkedPointer = PseudoLinkedPointerNEW;
17,218              }
17,219          {
17,220          // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,221          //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8; //RP
17,222          // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,223          if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
17,224          BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
17,225          BSTstack[StackPtr] = 16; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
17,226          // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,227          //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
17,228          //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
17,229          // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!

```

```

17,230 // [ //r.14+
17,231 memcpy( &PseudoLinkedPointer_64, &LEAF[8 + 8], 8 );
17,232 // ] //r.14+
17,233 }
17,234 else { FoundInLinkedList = 1; // wrd is RW
17,235 // Counter [
17,236 if (BSTorBtree == 2) {
17,237 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
17,238 }
17,239 //memcpy( &CounterOccurrences, &FourGramL[(KeySize+1+4)-4], 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
17,240 memcpy( &LastSeenOffset_PseudoPointer[jj], &FourGramL[(KeySize+1+8)-8], 8 );
17,241 //if (CounterOccurrences<999999999) CounterOccurrences++; // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
17,242 //memcpy( &FourGramL[(KeySize+1+4)-4], &CounterOccurrences, 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
17,243 memcpy( &FourGramL[(KeySize+1+8)-8], &GLOBAL_CurrentPositionForReading_TAILforLookAhead_ARRAY[jj], 8 ); // 2020-Jun-30: In 32bit code it is 4 not 8:
printf("%d", sizeof(char*));
17,244 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,245 //fwrite(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
17,246 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,247 // [ //r.14+
17,248 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &FourGramL[0], (KeySize+1+8) );
17,249 if (BSTorBtree == 2) {
17,250 fwrite(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG);
17,251 } else { // ##### 64bit memory manipulations [
17,252 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+8+2*(KeySize+1+8) );
17,253 } // ##### 64bit memory manipulations ]
17,254 // ] //r.14+
17,255 // Counter ]
17,256 }
17,257 // ***** 'P W P' section 2 ]
17,258 // ++++++
17,259 }
17,260 else // RW empty - go MP
17,261 // { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
17,262 // PseudoLinkedPointer = PseudoLinkedPointerNEW;
17,263 // }
17,264 {
17,265 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,266 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8; //MP
17,267 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,268 if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
17,269 BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
17,270 BSTstack[StackPtr] = 8; ++StackPtr; //LPOffset=0;MPOffset=8;RPOffset=16;
17,271 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,272 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
17,273 //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
17,274 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,275 // [ //r.14+
17,276 memcpy( &PseudoLinkedPointer_64, &LEAF[8], 8 );
17,277 // ] //r.14+
17,278 }
17,279 }
17,280 else { FoundInLinkedList = 1; // wrd is LW
17,281 // Counter [
17,282 if (BSTorBtree == 2) {
17,283 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
17,284 }
17,285 //memcpy( &CounterOccurrences, &FourGramL[(KeySize+1+4)-4], 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
17,286 memcpy( &LastSeenOffset_PseudoPointer[jj], &FourGramL[(KeySize+1+8)-8], 8 );

```

```

17,287 //if (CounterOccurrences<999999999) CounterOccurrences++; // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
17,288 //memcpy( &FourGramL[(KeySize+1+4)-4], &CounterOccurrences, 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
17,289 memcpy( &FourGramL[(KeySize+1+8)-8], &GLOBAL_CurrentPositionForReading_TAILforLookAhead_ARRAY[jj], 8 ); // 2020-Jun-30: In 32bit code it is 4 not 8:
printf("%d",sizeof(char*));
17,290 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,291 //fwrite(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
17,292 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,293 // [ //r.14+
17,294 memcpy( &LEAF[8 + 8 + 8], &FourGramL[0], (KeySize+1+8) );
17,295 if (BSTorBtree == 2) {
17,296 fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG);
17,297 } else { // ##### 64bit memory manipulations [
17,298 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
17,299 } // ##### 64bit memory manipulations ]
17,300 // ] //r.14+
17,301 // Counter ]
17,302 }
17,303 // ***** 'P W P' section ]
17,304 } // while
17,305 }
17,306
17,307
17,308 // 1] Search ] Write (uint64_t)(GLOBAL_CurrentPositionForReading_TAILforLookAhead) into the 8 bytes field
17,309 //}
17,310
17,311 GLOBAL_CurrentPositionForReading_TAILforLookAhead_ARRAY[jj]++;
17,312 } // At the end of this loop we should have (in LastSeenOffset_PseudoPointer[]) all matches found for position 'encStart'.
17,313 } //for (jj=0;
17,314
17,315
17,316 // 2021-Jan-12 [
17,317 /*
17,318 if ( (uint64_t)(encStart - src) >= 0x26C021-2 ) {
17,319 printf("\nBefore step #2 of 2:\n");
17,320 printf("src = %p\n", src);
17,321 printf("encStart = %p\n", encStart);
17,322 printf("(uint64_t)(encStart - src) = %016lx\n", (uint64_t)(encStart - src));
17,323
17,324 for (jj=0; jj< MatchLensNUM; jj++) {
17,325 printf("LastSeenOffset_PseudoPointer[%02d] = %016lx; \n", jj, LastSeenOffset_PseudoPointer[jj] );
17,326 }
17,327 printf("After step #2 of 2:\n");
17,328 }
17,329 */
17,330 // 2021-Jan-12 ]
17,331
17,332 // step #2 of 2: Loading into LastSeenOffset_PseudoPointer[] the actual LookAheadBuffer i.e. EncStart
17,333 GLOBAL_CurrentPositionForReading_TAILforLookAhead = encStart;
17,334 while ( GLOBAL_CurrentPositionForReading_TAILforLookAhead <= encStart ) { // TAIL should become LookAhead (i.e. encStart)
17,335 for (jj=0; jj< MatchLensNUM; jj++) {
17,336 LastSeenOffset_PseudoPointer[jj] = 0; // By Default - Not Seen
17,337 if ( GLOBAL_CurrentPositionForReading_TAILforLookAhead + MatchLens[jj] -1 <= GLOBAL_SourceBlock + GLOBAL_SourceSize -1 ) {
17,338
17,339 // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed [
17,340 // wrdlen=0;
17,341 // memset( &wrd[0], 0, (LongestLineInclusive+1+8) ); //r.18, see below, the old nullifier is commented.
17,342 // for (mm=0; mm< MatchLens[jj]; mm++) {
17,343 // memcpy( &wrd[ wrdlen ], TwoDigitHEXlist[(unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead+mm)], 2 );

```

```
17,344 // wrdlen++;
17,345 // wrdlen++;
17,346 // }
17,347 // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed ]
17,348
17,349 wrdlen=MatchLens[jj];
17,350
17,351 #if defined(_NSHA3) || defined(_NPRV) || defined(_NDD) || defined(_NAquaHash) //2020-Nov-20
17,352
17,353 #ifdef _NPRV //2020-Nov-12 [ Consider not only 256[+] matchlens but 24[+] as well - in order to save memory!
17,354     if (wrdlen > MatchLenAboveWhichHASHkicksin) {
17,355         prvhash42( (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen, (unsigned char *) PRVhash, PRVhashlenInBYTES, 0, 0, 0 );
17,356         //2020-Nov-17, for v2.0 ", 0" was added
17,357         wrdlen=MatchLenAboveWhichHASHkicksin; // very aggressive is 16B or 128b, if collisions happen then increase to 20B or 160b
17,358         memcpy( &wrd[ 0+1 ], (unsigned char *) PRVhash, wrdlen );
17,359     } else
17,360         memcpy( &wrd[ 0+1 ], (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen );
17,361 #endif
17,362 #ifdef _NSHA3
17,363     //if (wrdlen >= 256) {
17,364     if (wrdlen > MatchLenAboveWhichHASHkicksin) {
17,365         FIPS202_SHA3_224( (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen, (unsigned char *) SHA328bytes);
17,366         wrdlen=MatchLenAboveWhichHASHkicksin; //2020-Nov-12
17,367         memcpy( &wrd[ 0+1 ], (unsigned char *) SHA328bytes, wrdlen );
17,368     } else
17,369         memcpy( &wrd[ 0+1 ], (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen );
17,370 #endif
17,371 #ifdef _NDD
17,372     //if (wrdlen >= 256) { // 2020-Jun-13
17,373     if (wrdlen > MatchLenAboveWhichHASHkicksin) {
17,374         //*(uint64_t*)&DD[0] = crc64( (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen);
17,375         //*(uint32_t*)&DD[8] = crc32c_sw( (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen);
17,376         // Above 2 lines are glued into next one:
17,377         DoubleDeuce( (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen );
17,378         wrdlen=MatchLenAboveWhichHASHkicksin; //2020-Nov-12
17,379         memcpy( &wrd[ 0+1 ], (unsigned char *) DD, wrdlen );
17,380     } else
17,381         memcpy( &wrd[ 0+1 ], (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen );
17,382 #endif
17,383 #ifdef _NAquaHash
17,384     //if (wrdlen >= 256) { // 2020-Jun-13
17,385     if (wrdlen > MatchLenAboveWhichHASHkicksinAQUA) {
17,386         //*(uint64_t*)&DD[0] = crc64( (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen);
17,387         //*(uint32_t*)&DD[8] = crc32c_sw( (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen);
17,388         // Above 2 lines are glued into next one:
17,389         DoubleDeuceAES_Gumbotron_YMM( (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen );
17,390         wrdlen=MatchLenAboveWhichHASHkicksinAQUA; //2020-Nov-12
17,391         memcpy( &wrd[ 0+1 ], (unsigned char *) DD_AES, wrdlen );
17,392     } else
17,393         memcpy( &wrd[ 0+1 ], (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen );
17,394 #endif
17,395 #else
17,396     memcpy( &wrd[ 0+1 ], (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen );
17,397 #endif
17,398 #endif
17,399     memcpy( &wrd[ 0 ], &wrdlen, 1 ); // The keysize is 1..255, pretty enough since my needs are 1..28 (up to SHA3-224).
17,400
```

```

17,401 /*
17,402 #ifdef _NSHA3
17,403 //      if (wrdlen < 28)
17,404         if (wrdlen != 28) // 2020-Jun-13
17,405             memcpy( &wrd[ 0+1 ], (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen );
17,406         else
17,407             memcpy( &wrd[ 0+1 ], (unsigned char *) SHA328bytes, wrdlen );
17,408 #else
17,409 //      if (wrdlen < 28)
17,410         memcpy( &wrd[ 0+1 ], (unsigned char *) (GLOBAL_CurrentPositionForReading_TAILforLookAhead), wrdlen );
17,411         else
17,412             memcpy( &wrd[ 0+1 ], (unsigned char *) SHA328bytes, wrdlen );
17,413 #endif
17,414 */
17,415
17,416 //Slot = FNV1A_Hash_Jesteress_27bit(wrd +1, wrdlen);
17,417 Slot = FNV1A_Pippip_Yurii(wrd +1, wrdlen); // Changed 2019-Nov-28
17,418
17,419 Slot = Slot<<3;
17,420 // NEW NEW NEW [ //r.18
17,421 // In here Slot is not within a single pool but in MatchLensNUM sub-pools i.e. MatchLensNUM-way i.e. MatchLensNUM hashpots:
17,422 /*
17,423 for (GettingIndexofArray=0; GettingIndexofArray<MatchLensNUM; GettingIndexofArray++) {
17,424     // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed [
17,425     //      if ( (wrdlen>1)==MatchLens[GettingIndexofArray] ) break;
17,426     if ( (wrdlen)==MatchLens[GettingIndexofArray] ) break;
17,427     // Since 2019-Feb-07 the length of keys is 1:1 i.e. not HEXed ]
17,428 }
17,429 */
17,430 GettingIndexofArray=jj; // same as above atrocity
17,431 Slot = Slot + (GettingIndexofArray*(1LL<<HashInBITS_GLOBAL)*8); // CAUTION: 'wrdlen' is halved here, when a new revision comes with 1:1 keysize then change it.
17,432 // NEW NEW NEW ] //r.18
17,433
17,434 //KeySize = LongestLineInclusive; // 'LongestLineInclusive' is for 1-way hashing where different KeySizes can occupy same HASH pool.
17,435 KeySize = wrdlen; //r.18, In case multi-way hashing: either the MAX allowed or the 'wrdlen'.
17,436
17,437 //Slot = 0; // One Tree only!
17,438 memcpy( &PseudoLinkedPointer_64, GLOBAL_HASHPOT+Slot, 8 );
17,439
17,440 // 1] Search [ Read the 8 bytes field into LastSeenOffset_PseudoPointer[jj]
17,441
17,442         if (PseudoLinkedPointer_64 == 0) // means EMPTY-SLOT
17,443         {
17,444             // Impossible, they all have already been inserted...
17,445         }
17,446         else // means USED-SLOT
17,447         { FoundInLinkedList = 0;
17,448           StackPtr = 0;
17,449           //      while (PseudoLinkedPointer != 0 && FoundInLinkedList == 0)
17,450           while (PseudoLinkedPointer_64 != 0 && FoundInLinkedList == 0)
17,451           {
17,452           // ***** 'P W P' section [
17,453           // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
17,454           // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
17,455           // here ALWAYS LW exists: no need for existence check - line below
17,456           // if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
17,457           //      if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wrdlen) > 0) // go LP
17,458           // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!

```

```

17,459 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8;
17,460 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
17,461 //fread(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
17,462 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,463 // [ //r.14+
17,464 PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64;
17,465 if (BSTorBtree == 2) {
17,466 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
17,467 fread(&LEAF[0], 8+8+8+2*(KeySize+1+8), 1, fp_outRG);
17,468 } else { // ##### 64bit memory manipulations [
17,469 memcpy( &LEAF[0], (char *)&PseudoLinkedPointerAUX_64, 8+8+8+2*(KeySize+1+8) );
17,470 } // ##### 64bit memory manipulations ]
17,471 memcpy( &FourGramL[0], &LEAF[8 + 8 + 8], (KeySize+1+8) );
17,472 // ] //r.14+
17,473 //strFLAG=strcmpKAZE13(FourGramL, wrd);
17,474 strFLAG=memcmp(FourGramL +1, wrd +1, wrdlen); // Since 2019-Feb-07, it is not ASCIIZ string but zero byte is the length of the 1..
17,475 if (strFLAG > 0) // go LP
17,476 { // { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 0, 4 ); //LP
17,477 // PseudoLinkedPointer = PseudoLinkedPointerNEW;
17,478 // }
17,479 {
17,480 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,481 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 0; //LP
17,482 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,483 if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
17,484 BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
17,485 BSTstack[StackPtr] = 0; ++StackPtr; //LPoffset=0;MPoffset=8;RPoffset=16;
17,486 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,487 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
17,488 //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
17,489 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,490 // [ //r.14+
17,491 memcpy( &PseudoLinkedPointer_64, &LEAF[0], 8 );
17,492 // ] //r.14+
17,493 }
17,494 // else if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wrdlen) < 0) // go RP or MP
17,495 // else if (strFLAG < 0) // go RP or MP
17,496 // { // RW existence check - line below:
17,497 // if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 ) // RW exists
17,498 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,499 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4); //RW
17,500 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
17,501 //fread(&SomeByte, 1, 1, fp_outRG);
17,502 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,503 // [ //r.14+
17,504 memcpy( &SomeByte, &LEAF[8 + 8 + 8 + (KeySize+1+8)], 1 );
17,505 // ] //r.14+
17,506 if (SomeByte != 0 ) // RW exists
17,507 { // Here all 'P W P' section is repeated; the way of handling case when dynamic number of words in leaf
17,508 // *****
17,509 // ***** 'P W P' section 2 [
17,510 // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
17,511 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
17,512 // here ALWAYS RW exists: no need for existence check - line below
17,513 // if ( *(char *) (PseudoLinkedPointer+4+4+4+wrdlen) != 0 )
17,514 // if (memcmp(PseudoLinkedPointer+4+4+4+wrdlen, wrd, wrdlen) > 0) // go MP
17,515 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,516 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8 + 8 + 8 + (LongestLineInclusive+1+4);

```

Listing: Nakamichi Ryuugan-ditto-1TB_btree.c; Last version: 2021-Aug-30; Font: MxPlus ToshibaTxL2 8x16.ttf; Downloadable at: www.sanmayce.com/Nakamichi/Kaidanji.zip


```

2020-Jun-30: In 32bit code it is 4 not 8: printf("%d",sizeof(char*));
17,575 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,576 //fwrite(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
17,577 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,578 // [ //r.14+
17,579 memcpy( &LEAF[8 + 8 + 8 + (KeySize+1+8)], &FourGramL[0], (KeySize+1+8) );
17,580 // if (BSTorBtree == 2) {
17,581 fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG);
17,582 } else { // ##### 64bit memory manipulations [
17,583 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
17,584 } // ##### 64bit memory manipulations ]
17,585 // ] //r.14+
17,586 // Counter ]
17,587 }
17,588 // ***** 'P W P' section 2 ]
17,589 // *****
17,590 }
17,591 else // RW empty - go MP
17,592 { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
17,593 PseudoLinkedPointer = PseudoLinkedPointerNEW;
17,594 }
17,595 {
17,596 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,597 //PseudoLinkedPointerAUX_64 = PseudoLinkedPointer_64 + 8; //MP
17,598 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,599 if (StackPtr > 8192*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); exit( 13 );}
17,600 BSTstack[StackPtr] = PseudoLinkedPointer_64; ++StackPtr; //pt to visited leaf
17,601 BSTstack[StackPtr] = 8; ++StackPtr; //LPOffset=0;MPOffset=8;RPOffset=16;
17,602 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,603 //fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
17,604 //fread(&PseudoLinkedPointer_64, 8, 1, fp_outRG);
17,605 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,606 // [ //r.14+
17,607 memcpy( &PseudoLinkedPointer_64, &LEAF[8], 8 );
17,608 // ] //r.14+
17,609 }
17,610 }
17,611 else { FoundInLinkedList = 1; // wrd is LW
17,612 // Counter [
17,613 if (BSTorBtree == 2) {
17,614 fsetpos(fp_outRG, (const fpos_t *)&PseudoLinkedPointerAUX_64);
17,615 }
17,616 //memcpy( &CounterOccurrences, &FourGramL[(KeySize+1+4)-4], 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
17,617 memcpy( &LastSeenOffset_PseudoPointer[jj], &FourGramL[(KeySize+1+8)-8], 8 );
17,618 //if (CounterOccurrences<999999999) CounterOccurrences++; // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
17,619 //memcpy( &FourGramL[(KeySize+1+4)-4], &CounterOccurrences, 4 ); // r.18, 4 becomes 8, instead of counter it houses LastSeenOffset
17,620 // memcpy( &FourGramL[(KeySize+1+8)-8], &GLOBAL_CurrentPositionForReading_TAILforLookAhead, 8 ); // 2020-Feb-02, commented - NO UPDATING! //
2020-Jun-30: In 32bit code it is 4 not 8: printf("%d",sizeof(char*));
17,621 // [ //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,622 //fwrite(&FourGramL[0], (LongestLineInclusive+1+4), 1, fp_outRG);
17,623 // ] //r.14+ Optimized I/O i.e. reading a LEAF at once not LEAF's elements one-by-one!
17,624 // [ //r.14+
17,625 memcpy( &LEAF[8 + 8 + 8], &FourGramL[0], (KeySize+1+8) );
17,626 // if (BSTorBtree == 2) {
17,627 fwrite(&LEAF[0], 8+8+2*(KeySize+1+8), 1, fp_outRG);
17,628 } else { // ##### 64bit memory manipulations [
17,629 memcpy( (char *)PseudoLinkedPointerAUX_64, &LEAF[0], 8+8+2*(KeySize+1+8) );
17,630 } // ##### 64bit memory manipulations ]

```

```

17,631         // ] //r.14+
17,632         // Counter ]
17,633     }
17,634 // ***** 'P W P' section ]
17,635     } // while
17,636 }
17,637
17,638 // 1] Search ] Write (uint64_t)(GLOBAL_CurrentPositionForReading_TAILforLookAhead) into the 8 bytes field
17,639 }
17,640 } //for (jj=0;
17,641
17,642 GLOBAL_CurrentPositionForReading_TAILforLookAhead++;
17,643 } // At the end of this loop we should have (in LastSeenOffset_PseudoPointer[]) all matches found for position 'encStart'.
17,644
17,645 //btree ] 2020-Jan-25, improved updating - not causing fallbacks to memmem() by introducing GLOBAL_CurrentPositionForReading_TAILforLookAhead_ARRAY[]
17,646
17,647 //printf("\nLastSeenOffset_PseudoPointer[7]: %lu\n", LastSeenOffset_PseudoPointer[7]); //debugprint
17,648 // LastSeenOffset_PseudoPointer[9] is 64
17,649 // LastSeenOffset_PseudoPointer[8] is 36
17,650 // LastSeenOffset_PseudoPointer[7] is 18
17,651 // LastSeenOffset_PseudoPointer[6] is 16
17,652 // LastSeenOffset_PseudoPointer[5] is 14
17,653 // LastSeenOffset_PseudoPointer[4] is 12
17,654 // LastSeenOffset_PseudoPointer[3] is 10
17,655 // LastSeenOffset_PseudoPointer[2] is 8
17,656 // LastSeenOffset_PseudoPointer[1] is 6
17,657 // LastSeenOffset_PseudoPointer[0] is 4
17,658
17,659
17,660 // 2021-Jan-12 [
17,661 /*
17,662 if ( (uint64_t)(encStart - src) >= 0x26C021-2 ) {
17,663     printf("encStart first five bytes: %02x %02x %02x %02x %02x\n", *(encStart+0), *(encStart+1), *(encStart+2), *(encStart+3), *(encStart+4) );
17,664     for (jj=0; jj< MatchLensNUM; jj++) {
17,665         //printf("LastSeenOffset_PseudoPointer[%02d] = %016lx; (uint64_t)((char *)LastSeenOffset_PseudoPointer[%02d] - src) = %016lx\n", jj,
LastSeenOffset_PseudoPointer[jj], jj, (uint64_t)((char *)LastSeenOffset_PseudoPointer[jj] - src) );
17,666         printf("LastSeenOffset_PseudoPointer[%02d] = %016lx; ", jj, LastSeenOffset_PseudoPointer[jj] );
17,667         printf("(uint64_t)((char *)LastSeenOffset_PseudoPointer[%02d] - src) = %016lx\n", jj, (uint64_t)((char *)LastSeenOffset_PseudoPointer[jj] - src) );
17,668     }
17,669     for (jj=0; jj< MatchLensNUM; jj++) {
17,670         if ( src <= (char *)LastSeenOffset_PseudoPointer[jj] ) printf("LastSeenOffset_PseudoPointer[%02d] first five bytes: %02x %02x %02x %02x %02x\n", jj, *((char
*)(LastSeenOffset_PseudoPointer[jj])+0), *((char *)LastSeenOffset_PseudoPointer[jj]+1), *((char *)LastSeenOffset_PseudoPointer[jj]+2), *((char
*)(LastSeenOffset_PseudoPointer[jj])+3), *((char *)LastSeenOffset_PseudoPointer[jj]+4) );
17,671     }
17,672     exit(1);
17,673 }
17,674 */
17,675
17,676 // CARAMBA! When a key is not found then its corresponding 'LastSeenOffset_PseudoPointer[??]' should be ZERO! 0! Why the hell (in below dump, where only first 4 bytes
match: order 4 vs EncStart) LastSeenOffset_PseudoPointer[01] is not 0?!
17,677 // '3' is the first 3,123,123 bytes of stringology file.
17,678 /*
17,679 E:\KAZE_Smxt_Benchmarks\Nakamichi_Double-Deuce_C_source_binaries>Nakamichi_DD-192.exe 3 3.Nakamichi 20 1000 i
17,680
17,681 Compressing 3,123,123 bytes ...
17,682 /; Each rotation means 64KB are encoded; Speed: 0,069,176 B/s; Done 79%; Compression Ratio: 4.69:1; Matches(16/24/48): 5,533/4,192/2,335; 128[+] long matches: 1,950;
ETA: 0.00 hours
17,683 Before step #2 of 2:

```

```
17,684 src = 00440020
17,685 encStart = 006AC03F
17,686 (uint64_t)(encStart - src) = 00000000026c01f
17,687 LastSeenOffset_PseudoPointer[00] = 000000000067d486;
17,688 LastSeenOffset_PseudoPointer[01] = 0000000000000004;
17,689 LastSeenOffset_PseudoPointer[02] = 0000000000000004;
17,690 LastSeenOffset_PseudoPointer[03] = 0000000000000004;
17,691 LastSeenOffset_PseudoPointer[04] = 0000000000000004;
17,692 LastSeenOffset_PseudoPointer[05] = 0000000000000004;
17,693 LastSeenOffset_PseudoPointer[06] = 0000000000000004;
17,694 LastSeenOffset_PseudoPointer[07] = 0000000000000004;
17,695 LastSeenOffset_PseudoPointer[08] = 0000000000000004;
17,696 LastSeenOffset_PseudoPointer[09] = 0000000000000000;
17,697 LastSeenOffset_PseudoPointer[10] = 0000000000000004;
17,698 LastSeenOffset_PseudoPointer[11] = 0000000000000004;
17,699 LastSeenOffset_PseudoPointer[12] = 0000000000000004;
17,700 LastSeenOffset_PseudoPointer[13] = 0000000000000004;
17,701 LastSeenOffset_PseudoPointer[14] = 0000000000000000;
17,702 After step #2 of 2:
17,703 encStart first five bytes: 46 46 46 46 46
17,704 LastSeenOffset_PseudoPointer[00] = 000000000069e8c3; (uint64_t)((char *)LastSeenOffset_PseudoPointer[00] - src) = 00000000025e8a3
17,705 LastSeenOffset_PseudoPointer[01] = 000000000004d93d; (uint64_t)((char *)LastSeenOffset_PseudoPointer[01] - src) = 00000000ffc0d91d
17,706 LastSeenOffset_PseudoPointer[02] = 00000000000488ad; (uint64_t)((char *)LastSeenOffset_PseudoPointer[02] - src) = 00000000ffc0888d
17,707 LastSeenOffset_PseudoPointer[03] = 0000000000043b8c; (uint64_t)((char *)LastSeenOffset_PseudoPointer[03] - src) = 00000000ffc03b6c
17,708 LastSeenOffset_PseudoPointer[04] = 000000000003f2f4; (uint64_t)((char *)LastSeenOffset_PseudoPointer[04] - src) = 00000000ffbf2d4
17,709 LastSeenOffset_PseudoPointer[05] = 000000000003acbd; (uint64_t)((char *)LastSeenOffset_PseudoPointer[05] - src) = 00000000ffbfac9d
17,710 LastSeenOffset_PseudoPointer[06] = 0000000000036bcd; (uint64_t)((char *)LastSeenOffset_PseudoPointer[06] - src) = 00000000ffbf6bad
17,711 LastSeenOffset_PseudoPointer[07] = 00000000000330fa; (uint64_t)((char *)LastSeenOffset_PseudoPointer[07] - src) = 00000000ffbf30da
17,712 LastSeenOffset_PseudoPointer[08] = 0000000000018378; (uint64_t)((char *)LastSeenOffset_PseudoPointer[08] - src) = 00000000ffbd8358
17,713 LastSeenOffset_PseudoPointer[09] = 000000000000852; (uint64_t)((char *)LastSeenOffset_PseudoPointer[09] - src) = 00000000ffbc0832
17,714 LastSeenOffset_PseudoPointer[10] = 0000000000028ccf; (uint64_t)((char *)LastSeenOffset_PseudoPointer[10] - src) = 00000000ffbe8caf
17,715 LastSeenOffset_PseudoPointer[11] = 00000000000229a4; (uint64_t)((char *)LastSeenOffset_PseudoPointer[11] - src) = 00000000ffbe2984
17,716 LastSeenOffset_PseudoPointer[12] = 000000000000b27a; (uint64_t)((char *)LastSeenOffset_PseudoPointer[12] - src) = 00000000ffbc25a
17,717 LastSeenOffset_PseudoPointer[13] = 000000000000545e; (uint64_t)((char *)LastSeenOffset_PseudoPointer[13] - src) = 00000000ffbc543e
17,718 LastSeenOffset_PseudoPointer[14] = 000000000000068; (uint64_t)((char *)LastSeenOffset_PseudoPointer[14] - src) = 00000000ffbc0048
17,719 LastSeenOffset_PseudoPointer[00] first five bytes: 46 46 46 46 30
17,720 */
17,721 // Indeed, the offset 25e8a3 in '3' file does contain '46 46 46 46 30' - indeed the last seen such sequence is at this position. The rest 01..14 should be ZERO00!!!
17,722 // 2021-Jan-12 ]
17,723
17,724 // Due to the two lines added (as the NastyBugFix, the fix is to fill the 8bytes field for 'Occurrences' with zeroes, since it is used afterwards as LastSeenOffset[])
from '2021-Jan-13', now stringology bugs is crushed :P
17,725 // 2021-Jan-13 ]
17,726 /*
17,727 //; Each rotation means 64KB are encoded; Speed: 0,040,825 B/s; Done 79%; Compression Ratio: 4.36:1; Matches(16/24/48): 5,537/4,199/2,345; 128[+] long matches: 9,878;
ETA: 0.00 hours
17,728 Before step #2 of 2:
17,729 src = 00440020
17,730 encStart = 006AC03F
17,731 (uint64_t)(encStart - src) = 00000000026c01f
17,732 LastSeenOffset_PseudoPointer[00] = 000000000067d486;
17,733 LastSeenOffset_PseudoPointer[01] = 0000000000000000;
17,734 LastSeenOffset_PseudoPointer[02] = 0000000000000000;
17,735 LastSeenOffset_PseudoPointer[03] = 0000000000000000;
17,736 LastSeenOffset_PseudoPointer[04] = 0000000000000000;
17,737 LastSeenOffset_PseudoPointer[05] = 0000000000000000;
17,738 LastSeenOffset_PseudoPointer[06] = 0000000000000000;
17,739 LastSeenOffset_PseudoPointer[07] = 0000000000000000;
```

```
17,740 LastSeenOffset_PseudoPointer[08] = 0000000000000000;
17,741 LastSeenOffset_PseudoPointer[09] = 0000000000000000;
17,742 LastSeenOffset_PseudoPointer[10] = 0000000000000000;
17,743 LastSeenOffset_PseudoPointer[11] = 0000000000000000;
17,744 LastSeenOffset_PseudoPointer[12] = 0000000000000000;
17,745 LastSeenOffset_PseudoPointer[13] = 0000000000000000;
17,746 LastSeenOffset_PseudoPointer[14] = 0000000000000000;
17,747 After step #2 of 2:
17,748 encStart first five bytes: 46 46 46 46 46
17,749 LastSeenOffset_PseudoPointer[00] = 000000000069e8c3; (uint64_t)((char *)LastSeenOffset_PseudoPointer[00] - src) = 00000000025e8a3
17,750 LastSeenOffset_PseudoPointer[01] = 0000000000000000; (uint64_t)((char *)LastSeenOffset_PseudoPointer[01] - src) = 00000000ffbbffe0
17,751 LastSeenOffset_PseudoPointer[02] = 0000000000000000; (uint64_t)((char *)LastSeenOffset_PseudoPointer[02] - src) = 00000000ffbbffe0
17,752 LastSeenOffset_PseudoPointer[03] = 0000000000000000; (uint64_t)((char *)LastSeenOffset_PseudoPointer[03] - src) = 00000000ffbbffe0
17,753 LastSeenOffset_PseudoPointer[04] = 0000000000000000; (uint64_t)((char *)LastSeenOffset_PseudoPointer[04] - src) = 00000000ffbbffe0
17,754 LastSeenOffset_PseudoPointer[05] = 0000000000000000; (uint64_t)((char *)LastSeenOffset_PseudoPointer[05] - src) = 00000000ffbbffe0
17,755 LastSeenOffset_PseudoPointer[06] = 0000000000000000; (uint64_t)((char *)LastSeenOffset_PseudoPointer[06] - src) = 00000000ffbbffe0
17,756 LastSeenOffset_PseudoPointer[07] = 0000000000000000; (uint64_t)((char *)LastSeenOffset_PseudoPointer[07] - src) = 00000000ffbbffe0
17,757 LastSeenOffset_PseudoPointer[08] = 0000000000000000; (uint64_t)((char *)LastSeenOffset_PseudoPointer[08] - src) = 00000000ffbbffe0
17,758 LastSeenOffset_PseudoPointer[09] = 0000000000000000; (uint64_t)((char *)LastSeenOffset_PseudoPointer[09] - src) = 00000000ffbbffe0
17,759 LastSeenOffset_PseudoPointer[10] = 0000000000000000; (uint64_t)((char *)LastSeenOffset_PseudoPointer[10] - src) = 00000000ffbbffe0
17,760 LastSeenOffset_PseudoPointer[11] = 0000000000000000; (uint64_t)((char *)LastSeenOffset_PseudoPointer[11] - src) = 00000000ffbbffe0
17,761 LastSeenOffset_PseudoPointer[12] = 0000000000000000; (uint64_t)((char *)LastSeenOffset_PseudoPointer[12] - src) = 00000000ffbbffe0
17,762 LastSeenOffset_PseudoPointer[13] = 0000000000000000; (uint64_t)((char *)LastSeenOffset_PseudoPointer[13] - src) = 00000000ffbbffe0
17,763 LastSeenOffset_PseudoPointer[14] = 0000000000000000; (uint64_t)((char *)LastSeenOffset_PseudoPointer[14] - src) = 00000000ffbbffe0
17,764 LastSeenOffset_PseudoPointer[00] first five bytes: 46 46 46 46 30
17,765
17,766 E:\KAZE_Smxt_Benchmarks\Nakamichi_Double-Deuce_C_source_binaries>
17,767 */
17,768 // Indeed, the offset 25e8a3 in '3' file does contain '46 46 46 46 30' - indeed the last seen such sequence is at this position. The rest 01..14 should be ZERO00!!!
They are now!
17,769 // 2021-Jan-13 ]
17,770
17,771 #ifdef Kaidanji
17,772 if ( refStart <= (char *)LastSeenOffset_PseudoPointer[MatchLensNUM-1] ) { // 2021-Jun-21, changed 6 to 'MatchLensNUM-1'
17,773
17,774     LongestCommonPrefix = memcmp_CommonPrefixLength ( (char *)LastSeenOffset_PseudoPointer[MatchLensNUM-1], encStart, 256); // 2021-Jun-21, changed 6 to
'MatchLensNUM-1'
17,775
17,776     if (LongestCommonPrefix >= MatchLens[MatchLensNUM-1]) { // 2021-Jun-21, changed 16 to 'MatchLens[MatchLensNUM-1]'
17,777         //int MatchLens[MatchLensNUMdummy]={4,6,8,10,12,14,16,18,36,64,24,28,48,56,128,256};
17,778         ///          0,1,2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f
17,779         for (i=MatchLensNUM-1; i<MatchLensNUMdummy; i++) // 2021-Jun-21, changed 6 to 'MatchLensNUM-1'
17,780             if (LongestCommonPrefix >= MatchLens[i]) LastSeenOffset_PseudoPointer[i]=LastSeenOffset_PseudoPointer[MatchLensNUM-1]; // 2021-Jun-21, changed 6 to
'MatchLensNUM-1'
17,781     }
17,782 }
17,783 #endif
17,784
17,785 //goto NoFreeTag;
17,786 // 2021-Mar-11 [ Adding 512:4=128 (16MB) encoded as FFFFFFFF*0xF3 (i.e. LSB->MSB: 3FFFFFFF) replacing 6:4 (16MB)
17,787 // #00 512:4=128 (16MB)
17,788 #if defined(BtreeHEURISTIC)
17,789     if ( LastSeenOffset_PseudoPointer[14] != 0 )
17,790         //int MatchLens[MatchLensNUMdummy]={4,6,8,10,12,14,16,18,36,64,24,28,48,56,128,256};
17,791         ///          0,1,2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f
17,792 #endif
17,793 {
17,794
```

Listing: Nakamichi Ryuugan-ditto-1TB btree.c; Last version: 2021-Aug-30; Font: MxPlus ToshibaTxL2 8x16.ttf; Downloadable at: www.sanmayce.com/Nakamichi/Kaidanji.zip

Listing: Nakamichi_Ryuugan-ditto-1TB_btree.c; Last version: 2021-Aug-30; Font: MxPlus ToshibaTxL2 8x16.ttf; Downloadable at: www.sanmayce.com/Nakamichi/Kaidanji.zip

```

17,900 #endif
17,901
17,902         if (FoundAtPosition!=NULL) {
17,903             *retMatch=64*4;
17,904             // The first 8 bits should be 0x93, in order to retain 4bytes-longness of 'retIndex' we play dirty - not tagging the 5bytes chunk with 0x93
but as "usual" - the address subchunk:
17,905             *retIndex=(((refEnd-FoundAtPosition)<<0)&0xFFFFFFFF);
17,906             *ShortMediumLongOFFSET=5+1 +9999;
17,907             *HowManyDittoTagsToEmit = 3;
17,908             return;
17,909         }
17,910     }
17,911
17,912     } // B-TREE heuristic
17,913 #endif
17,914
17,915 Skip = NULL;
17,916
17,917 #if defined(BtreeHEURISTIC)
17,918     if ( LastSeenOffset_PseudoPointer[9] != 0 )
17,919 #endif
17,920     { // Search into ... If 64 is not found then skip all 64[+] - see the roster above.
17,921
17,922     // In order to skip 192,256,320 it is better to try 128 before 192,256,320 - they are slow! [
17,923     // #04 128:(5+1)+1= 18.2 (1TB)
17,924     if (refStartSW6ryuu >= refStart) {
17,925         // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW6ryuu, encStart, (uint64_t)(refEnd-refStartSW6ryuu), 64*2);
17,926         Skip = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW6ryuu - src) -(uint64_t)(refEnd-refStartSW6ryuu -1), srcR + (srcSize - 1) - (encStart - src) -(64*2-1),
(uint64_t)(refEnd-refStartSW6ryuu), 64*2);
17,927     }
17,928     // In order to skip 192,256,320 it is better to try 128 before 192,256,320 - they are slow! ]
17,929
17,930     if (Skip!=NULL) {
17,931
17,932     // #01 704:(5+1)+1+1+1+1+1+1+1+1=44 (1TB)
17,933     if (refStartSW6ryuu >= refStart) {
17,934         // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW6ryuu, encStart, (uint64_t)(refEnd-refStartSW6ryuu), 64*11);
17,935         FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW6ryuu - src) -(uint64_t)(refEnd-refStartSW6ryuu -1), srcR + (srcSize - 1) - (encStart - src)
-(64*11-1), (uint64_t)(refEnd-refStartSW6ryuu), 64*11);
17,936         if (FoundAtPosition2!=NULL) {
17,937             FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(64*11-1);
17,938         } else FoundAtPosition = NULL;
17,939
17,940         if (FoundAtPosition!=NULL) {
17,941             *retMatch=64*11;
17,942             // The first 8 bits should be 0x93, in order to retain 4bytes-longness of 'retIndex' we play dirty - not tagging the 5bytes chunk with 0x93
but as "usual" - the address subchunk:
17,943             *retIndex=(((refEnd-FoundAtPosition)<<0)&0xFFFFFFFF);
17,944             *ShortMediumLongOFFSET=5+1 +9999;
17,945             *HowManyDittoTagsToEmit = 10;
17,946             return;
17,947         }
17,948     }
17,949
17,950     // #01 320:(5+1)+1+1+1+1=32 (1TB)
17,951     if (refStartSW6ryuu >= refStart) {
17,952         // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW6ryuu, encStart, (uint64_t)(refEnd-refStartSW6ryuu), 64*5);
17,953         FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW6ryuu - src) -(uint64_t)(refEnd-refStartSW6ryuu -1), srcR + (srcSize - 1) - (encStart - src)

```

```

-(64*5-1), (uint64_t)(refEnd-refStartSW6ryuu), 64*5);
17,954     if (FoundAtPosition2!=NULL) {
17,955 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(64*5-1);
17,956     } else FoundAtPosition = NULL;
17,957
17,958         if (FoundAtPosition!=NULL) {
17,959             *retMatch=64*5;
17,960             // The first 8 bits should be 0x93, in order to retain 4bytes-longness of 'retIndex' we play dirty - not tagging the 5bytes chunk with 0x93
but as "usual" - the address subchunk:
17,961             *retIndex=((refEnd-FoundAtPosition)<<0)&0xFFFFFFFF);
17,962             *ShortMediumLongOFFSET=5*1 +9999;
17,963             *HowManyDittoTagsToEmit = 4;
17,964             return;
17,965         }
17,966     }
17,967
17,968 // #03 192:(5+1)+1+1= 24      (1TB)
17,969 if (refStartSW6ryuu >= refStart) {
17,970 //     FoundAtPosition = Railgun_BawBaw_reverse (refStartSW6ryuu, encStart, (uint64_t)(refEnd-refStartSW6ryuu), 64*3);
17,971 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW6ryuu - src) -(uint64_t)(refEnd-refStartSW6ryuu -1), srcR + (srcSize - 1) - (encStart - src)
-(64*3-1), (uint64_t)(refEnd-refStartSW6ryuu), 64*3);
17,972     if (FoundAtPosition2!=NULL) {
17,973 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(64*3-1);
17,974     } else FoundAtPosition = NULL;
17,975
17,976         if (FoundAtPosition!=NULL) {
17,977             *retMatch=64*3;
17,978             // The first 8 bits should be 0x93, in order to retain 4bytes-longness of 'retIndex' we play dirty - not tagging the 5bytes chunk with 0x93
but as "usual" - the address subchunk:
17,979             *retIndex=((refEnd-FoundAtPosition)<<0)&0xFFFFFFFF);
17,980             *ShortMediumLongOFFSET=5*1 +9999;
17,981             *HowManyDittoTagsToEmit = 2;
17,982             return;
17,983         }
17,984     }
17,985 }
17,986 }
17,987
17,988     } // B-TREE heuristic
17,989
17,990
17,991 #if defined(BtreeHEURISTIC)
17,992     if ( LastSeenOffset_PseudoPointer[9] != 0 )
17,993         //int MatchLens[MatchLensNUMdummy]={4,6,8,10,12,14,16,18,36,64,24,28,48,56,128,256};
17,994         ///      0,1,2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f
17,995 #endif
17,996     { // Search into ... If 64 is not found then skip all 64[+] - see the roster above.
17,997
17,998 // 90:2+1+1=22.5      (512B)
17,999 if (refStartSW512 >= refStart) {
18,000 FoundAtPosition = Railgun_BawBaw_reverse (refStartSW512, encStart, (uint32_t)(refEnd-refStartSW512), 30*3); // Uncommented , 2021-Mar-11, below 4 lines need reversed
file block
18,001 //     FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW512 - src) -(uint32_t)(refEnd-refStartSW512 -1), srcR + (srcSize - 1) - (encStart -
src) -(30*3-1), (uint32_t)(refEnd-refStartSW512), 30*3);
18,002 //         if (FoundAtPosition2!=NULL) {
18,003 //             FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(30*3-1);
18,004 //         } else FoundAtPosition = NULL;
18,005

```



```

18,006         if (FoundAtPosition!=NULL) {
18,007             *retMatch=30*3;
18,008             // The first four bits should be:
18,009             *retIndex=((refEnd-FoundAtPosition)<<7)&0xFF80)!0x7C; // (4+3)C
18,010             *ShortMediumLongOFFSET=2;
18,011             *HowManyDittoTagsToEmit = 2;
18,012             return;
18,013         }
18,014     }
18,015
18,016         } // B-TREE heuristic
18,017
18,018 #if defined(BtreeHEURISTIC)
18,019     if ( LastSeenOffset_PseudoPointer[9] != 0 )
18,020         //int MatchLens[MatchLensNUMdummy]={4,6,8,10,12,14,16,18,36,64,24,28,48,56,128,256};
18,021         ///          0,1,2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f
18,022 #endif
18,023     { // Search into ... If 64 is not found then skip all 64[+] - see the roster above.
18,024
18,025         // 120:3+1+1+1=20      (128KB)
18,026         if (refStartSW128kb >= refStart) {
18,027             FoundAtPosition = Railgun_BawBaw_reverse (refStartSW128kb, encStart, (uint32_t)(refEnd-refStartSW128kb), 30*4); // Uncommented , 2021-Mar-11, below 4 lines need
reversed file block
18,028             FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW128kb - src) -(uint32_t)(refEnd-refStartSW128kb -1), srcR + (srcSize - 1) -
(encStart - src) -(30*4-1), (uint32_t)(refEnd-refStartSW128kb), 30*4);
18,029             //             if (FoundAtPosition2!=NULL) {
18,030             //                 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(30*4-1);
18,031             //             } else FoundAtPosition = NULL;
18,032
18,033             if (FoundAtPosition!=NULL) {
18,034                 *retMatch=30*4;
18,035                 // The first four bits should be:
18,036                 *retIndex=((refEnd-FoundAtPosition)<<7)&0xFFFF80)!0x3C;
18,037                 *ShortMediumLongOFFSET=3;
18,038                 *HowManyDittoTagsToEmit = 3;
18,039                 return;
18,040             }
18,041         }
18,042
18,043         } // B-TREE heuristic
18,044
18,045 #if defined(BtreeHEURISTIC)
18,046     if ( LastSeenOffset_PseudoPointer[13] != 0 ) // MatchLens[MatchLensNUM]={4,6,8,10,12,14,16,18,36,64,24,28,48};
18,047         //int MatchLens[MatchLensNUMdummy]={4,6,8,10,12,14,16,18,36,64,24,28,48,56,128,256};
18,048         ///          0,1,2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f
18,049 #endif
18,050     { // Search into ... If 48 is not found then skip all 48[+] - see the roster above.
18,051
18,052         // 60:2+1=20      (512B)
18,053         if (refStartSW512 >= refStart) {
18,054             FoundAtPosition = Railgun_BawBaw_reverse (refStartSW512, encStart, (uint32_t)(refEnd-refStartSW512), 30*2); // Uncommented , 2021-Mar-11, below 4 lines need reversed
file block
18,055             FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW512 - src) -(uint32_t)(refEnd-refStartSW512 -1), srcR + (srcSize - 1) - (encStart -
src) -(30*2-1), (uint32_t)(refEnd-refStartSW512), 30*2);
18,056             //             if (FoundAtPosition2!=NULL) {
18,057             //                 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(30*2-1);
18,058             //             } else FoundAtPosition = NULL;
18,059

```

```

18,060         if (FoundAtPosition!=NULL) {
18,061             *retMatch=30*2;
18,062             // The first four bits should be:
18,063             *retIndex=((refEnd-FoundAtPosition)<<7)&0xFF80)!0x7C; // (4+3)C
18,064             *ShortMediumLongOFFSET=2;
18,065             *HowManyDittoTagsToEmit = 1;
18,066             return;
18,067         }
18,068     }
18,069
18,070     } // B-TREE heuristic
18,071
18,072 #if defined(BtreeHEURISTIC)
18,073     if ( LastSeenOffset_PseudoPointer[9] != 0 )
18,074         //int MatchLens[MatchLensNumDummy]={4,6,8,10,12,14,16,18,36,64,24,28,48,56,128,256};
18,075         ///          0,1,2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f
18,076 #endif
18,077     { // Search into ... If 64 is not found then skip all 64[+] - see the roster above.
18,078
18,079     // #03a 96:2+1+1+1=19.2      (4KB)
18,080     if (refStartSW2 >= refStart) {
18,081         FoundAtPosition = Railgun_BawBaw_reverse (refStartSW2, encStart, (uint32_t)(refEnd-refStartSW2), 24*4); // Uncommented , 2021-Mar-11, below 4 lines need reversed file
        block
18,082         //         FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW2 - src) -(uint32_t)(refEnd-refStartSW2 -1), srcR + (srcSize - 1) - (encStart - src)
        -(24*4-1), (uint32_t)(refEnd-refStartSW2), 24*4);
18,083         //         if (FoundAtPosition2!=NULL) {
18,084         //             FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(24*4-1);
18,085         //         } else FoundAtPosition = NULL;
18,086
18,087         if (FoundAtPosition!=NULL) {
18,088             *retMatch=24*4;
18,089             // The first four bits should be:
18,090
18,091             *retIndex=((refEnd-FoundAtPosition)<<4)&0xFFF0)!0x0008; // xx ... x[00LL] // 1000b = 8
18,092             *ShortMediumLongOFFSET=2;
18,093             *HowManyDittoTagsToEmit = 3;
18,094             return;
18,095         }
18,096     }
18,097
18,098
18,099 goto Jan27;
18,100 if (Skip!=NULL) {
18,101
18,102 // #04 128:(5+1)+1=      18.2      (1TB)
18,103 if (refStartSW6ryuu >= refStart) {
18,104 //         FoundAtPosition = Railgun_BawBaw_reverse (refStartSW6ryuu, encStart, (uint64_t)(refEnd-refStartSW6ryuu), 64*2);
18,105 FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW6ryuu - src) -(uint64_t)(refEnd-refStartSW6ryuu -1), srcR + (srcSize - 1) - (encStart - src)
        -(64*2-1), (uint64_t)(refEnd-refStartSW6ryuu), 64*2);
18,106         if (FoundAtPosition2!=NULL) {
18,107 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(64*2-1);
18,108         } else FoundAtPosition = NULL;
18,109
18,110         if (FoundAtPosition!=NULL) {
18,111             *retMatch=64*2;
18,112             // The first 8 bits should be 0x93, in order to retain 4bytes-longness of 'retIndex' we play dirty - not tagging the 5bytes chunk with 0x93
        but as "usual" - the address subchunk:
18,113             *retIndex=((refEnd-FoundAtPosition)<<0)&0xFFFFFFFF);

```

```

18,114         *ShortMediumLongOFFSET=5+1 +9999;
18,115         *HowManyDittoTagsToEmit = 1;
18,116         return;
18,117     }
18,118 }
18,119
18,120 }
18,121 Jan27:
18,122
18,123
18,124 #if defined(BtreeHEURISTIC)
18,125     if ( LastSeenOffset_PseudoPointer[14] != 0 )
18,126         //int MatchLens[MatchLensNUMdummy]={4,6,8,10,12,14,16,18,36,64,24,28,48,56,128,256};
18,127         ///          0,1,2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f
18,128 #endif
18,129     { // Search into ... If 48 is not found then skip all 48[+] - see the roster above.
18,130
18,131 //#04 128:(5+1)+1=    18.2    (1TB)
18,132 if (refStartSW6ryuu >= refStart) {
18,133
18,134 #ifndef BtreeHEURISTIC
18,135 //      FoundAtPosition = Railgun_BawBaw_reverse (refStartSW6ryuu, encStart, (uint64_t)(refEnd-refStartSW6ryuu), 64);
18,136 FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW6ryuu - src) -(uint64_t)(refEnd-refStartSW6ryuu -1), srcR + (srcSize - 1) - (encStart - src)
18,137 -(64*2-1), (uint64_t)(refEnd-refStartSW6ryuu), 64*2);
18,138 if (FoundAtPosition2!=NULL) {
18,139 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(64*2-1);
18,140 } else FoundAtPosition = NULL;
18,141 #else
18,142 if ( refStartSW6ryuu <= (char *)LastSeenOffset_PseudoPointer[14] ) {
18,143     if ( refEnd - (char *)LastSeenOffset_PseudoPointer[14] < (64*2) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
18,144 first position of the LookAhead!
18,145 // Fallback to memmem(): [
18,146 FoundAtPosition = Railgun_BawBaw_reverse (refStartSW6ryuu, encStart, (uint64_t)(refEnd-refStartSW6ryuu), 64);
18,147 FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW6ryuu - src) -(uint64_t)(refEnd-refStartSW6ryuu -1), srcR + (srcSize - 1) - (encStart - src)
18,148 -(64*2-1), (uint64_t)(refEnd-refStartSW6ryuu), 64*2);
18,149 if (FoundAtPosition2!=NULL) {
18,150 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(64*2-1);
18,151 } else FoundAtPosition = NULL;
18,152 // Fallback to memmem(): ]
18,153 } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[14];
18,154 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
18,155 %p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
18,156 } else FoundAtPosition = NULL;
18,157 #endif
18,158
18,159 if (FoundAtPosition!=NULL) {
18,160     *retMatch=64*2;
18,161 // The first 8 bits should be 0x93, in order to retain 4bytes-longness of 'retIndex' we play dirty - not tagging the 5bytes chunk with 0x93
18,162 but as "usual" - the address subchunk:
18,163 *retIndex=((refEnd-FoundAtPosition)<<0)&0xFFFFFFFF);
18,164 *ShortMediumLongOFFSET=5+1 +9999;
18,165 *HowManyDittoTagsToEmit = 1;
18,166 return;
18,167 }
18,168 }
18,169 } // B-TREE heuristic
18,170
18,171

```

```

18,167 // 90:3+1+1=18      (128KB)
18,168 if (refStartSW128kb >= refStart) {
18,169 FoundAtPosition = Railgun_BawBaw_reverse (refStartSW128kb, encStart, (uint32_t)(refEnd-refStartSW128kb), 30*3); // Uncommented , 2021-Mar-11, below 4 lines need
reversed file block
18,170 // FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW128kb - src) -(uint32_t)(refEnd-refStartSW128kb -1), srcR + (srcSize - 1) -
(encStart - src) -(30*3-1), (uint32_t)(refEnd-refStartSW128kb), 30*3);
18,171 // if (FoundAtPosition2!=NULL) {
18,172 // FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(30*3-1);
18,173 // } else FoundAtPosition = NULL;
18,174
18,175 if (FoundAtPosition!=NULL) {
18,176 *retMatch=30*3;
18,177 // The first four bits should be:
18,178 *retIndex=(((refEnd-FoundAtPosition)<<7)&0xFFFF80)!0x3C;
18,179 *ShortMediumLongOFFSET=3;
18,180 *HowManyDittoTagsToEmit = 2;
18,181 return;
18,182 }
18,183 }
18,184
18,185 // #04a 72:2+1+1=18      (4KB)
18,186 if (refStartSW2 >= refStart) {
18,187 FoundAtPosition = Railgun_BawBaw_reverse (refStartSW2, encStart, (uint32_t)(refEnd-refStartSW2), 24*3); // Uncommented , 2021-Mar-11, below 4 lines need reversed file
block
18,188 // FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW2 - src) -(uint32_t)(refEnd-refStartSW2 -1), srcR + (srcSize - 1) - (encStart - src)
-(24*3-1), (uint32_t)(refEnd-refStartSW2), 24*3);
18,189 // if (FoundAtPosition2!=NULL) {
18,190 // FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(24*3-1);
18,191 // } else FoundAtPosition = NULL;
18,192
18,193 if (FoundAtPosition!=NULL) {
18,194 *retMatch=24*3;
18,195 // The first four bits should be:
18,196 // 1000b = 8
18,197 *retIndex=(((refEnd-FoundAtPosition)<<4)&0xFFF0)!0x0008; // xx ... x[00LL]
18,198 *ShortMediumLongOFFSET=2;
18,199 *HowManyDittoTagsToEmit = 2;
18,200 return;
18,201 }
18,202 }
18,203
18,204 // #04b 96:3+1+1+1=16      (1MB)
18,205 if (refStartSW3 >= refStart) {
18,206 FoundAtPosition = Railgun_BawBaw_reverse (refStartSW3, encStart, (uint32_t)(refEnd-refStartSW3), 24*4); // Uncommented , 2021-Mar-11, below 4 lines need reversed file
block
18,207 // FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW3 - src) -(uint32_t)(refEnd-refStartSW3 -1), srcR + (srcSize - 1) - (encStart - src)
-(24*4-1), (uint32_t)(refEnd-refStartSW3), 24*4);
18,208 // if (FoundAtPosition2!=NULL) {
18,209 // FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(24*4-1);
18,210 // } else FoundAtPosition = NULL;
18,211
18,212 if (FoundAtPosition!=NULL) {
18,213 *retMatch=24*4;
18,214 // The first four bits should be:
18,215 // 0100b = 4
18,216 *retIndex=(((refEnd-FoundAtPosition)<<4)&0xFFFFF0)!0x0004; // xx ... x[00LL]
18,217 *ShortMediumLongOFFSET=3;
18,218 *HowManyDittoTagsToEmit = 3;

```

```

18,219         return;
18,220     }
18,221 }
18,222
18,223     } // B-TREE heuristic
18,224
18,225 #if defined(BtreeHEURISTIC)
18,226     if ( LastSeenOffset_PseudoPointer[12] != 0 ) // MatchLens[MatchLensNUM]={4,6,8,10,12,14,16,18,36,64,24,28,48};
18,227 #endif
18,228     { // Search into ... If 48 is not found then skip all 48[+] - see the roster above.
18,229
18,230 //#04c 48:2+1=16           (4KB)
18,231 if (refStartSW2 >= refStart) {
18,232
18,233 #ifndef BtreeHEURISTIC
18,234 //     FoundAtPosition = Railgun_BawBaw_reverse (refStartSW2, encStart, (uint32_t)(refEnd-refStartSW2), 24*2);
18,235 FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW2 - src) -(uint32_t)(refEnd-refStartSW2 -1), srcR + (srcSize - 1) - (encStart - src) -(24*2-
1), (uint32_t)(refEnd-refStartSW2), 24*2);
18,236     if (FoundAtPosition2!=NULL) {
18,237 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(24*2-1);
18,238     } else FoundAtPosition = NULL;
18,239 #else
18,240     if ( refStartSW2 <= (char *)LastSeenOffset_PseudoPointer[12] ) {
18,241         if ( refEnd - (char *)LastSeenOffset_PseudoPointer[12] < (48) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
first position of the LookAhead!
18,242             // Fallback to memmem(): [
18,243 //     FoundAtPosition = Railgun_BawBaw_reverse (refStartSW2, encStart, (uint32_t)(refEnd-refStartSW2), 24*2);
18,244 FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW2 - src) -(uint32_t)(refEnd-refStartSW2 -1), srcR + (srcSize - 1) - (encStart - src) -(24*2-
1), (uint32_t)(refEnd-refStartSW2), 24*2);
18,245     if (FoundAtPosition2!=NULL) {
18,246 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(24*2-1);
18,247     } else FoundAtPosition = NULL;
18,248         // Fallback to memmem(): ]
18,249     } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[12];
18,250 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
%p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
18,251     } else FoundAtPosition = NULL;
18,252 #endif
18,253
18,254     if (FoundAtPosition!=NULL) {
18,255         *retMatch=24*2;
18,256         // The first four bits should be:
18,257
18,258         *retIndex=(((refEnd-FoundAtPosition)<<4)&0xFFF0)!0x0008; // xx ... x[00LL] // 1000b = 8
18,259         *ShortMediumLongOFFSET=2;
18,260         *HowManyDittoTagsToEmit = 1;
18,261         return;
18,262     }
18,263 }
18,264
18,265     } // B-TREE heuristic
18,266
18,267 #if defined(BtreeHEURISTIC)
18,268     if ( LastSeenOffset_PseudoPointer[13] != 0 ) // MatchLens[MatchLensNUM]={4,6,8,10,12,14,16,18,36,64,24,28,48};
18,269     //int MatchLens[MatchLensNUMdummy]={4,6,8,10,12,14,16,18,36,64,24,28,48,56,128,256};
18,270     ///
18,271     //0,1,2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f
18,272 #endif
18,272     { // Search into ... If 48 is not found then skip all 48[+] - see the roster above.

```

```
18,273
18,274 // 60:3+1=15 (128KB)
18,275 if (refStartSW128kb >= refStart) {
18,276 FoundAtPosition = Railgun_BawBaw_reverse (refStartSW128kb, encStart, (uint32_t)(refEnd-refStartSW128kb), 30*2); // Uncommented , 2021-Mar-11, below 4 lines need
reversed file block
18,277 // FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW128kb - src) -(uint32_t)(refEnd-refStartSW128kb -1), srcR + (srcSize - 1) -
(encStart - src) -(30*2-1), (uint32_t)(refEnd-refStartSW128kb), 30*2);
18,278 // if (FoundAtPosition2!=NULL) {
18,279 // FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(30*2-1);
18,280 // } else FoundAtPosition = NULL;
18,281
18,282 if (FoundAtPosition!=NULL) {
18,283 *retMatch=30*2;
18,284 // The first four bits should be:
18,285 *retIndex=(((refEnd-FoundAtPosition)<<7)&0xFFFF80)!0x3C;
18,286 *ShortMediumLongOFFSET=3;
18,287 *HowManyDittoTagsToEmit = 1;
18,288 return;
18,289 }
18,290 }
18,291
18,292 } // B-TREE heuristic
18,293
18,294 #if defined(BtreeHEURISTIC)
18,295 if ( LastSeenOffset_PseudoPointer[11] != 0 ) // 2020-Jan-01
18,296 //int MatchLens[MatchLensNUMdummy]={4,6,8,10,12,14,16,18,36,64,24,28,48,56,128,256};
18,297 //0,1,2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f
18,298 #endif
18,299 { // Search into ... If 18 is not found then skip all 18[+] - see the roster above.
18,300
18,301 //05 30:2= 15 (512B)
18,302 if (refStartSW512 >= refStart) {
18,303 FoundAtPosition = Railgun_BawBaw_reverse (refStartSW512, encStart, (uint32_t)(refEnd-refStartSW512), 30); // Uncommented , 2021-Mar-11, below 4 lines need reversed
file block
18,304 // FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW512 - src) -(uint32_t)(refEnd-refStartSW512 -1), srcR + (srcSize - 1) - (encStart -
src) -(30-1), (uint32_t)(refEnd-refStartSW512), 30);
18,305 // if (FoundAtPosition2!=NULL) {
18,306 // FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(30-1);
18,307 // } else FoundAtPosition = NULL;
18,308
18,309 if (FoundAtPosition!=NULL) {
18,310 *retMatch=30;
18,311 // The first four bits should be:
18,312 *retIndex=(((refEnd-FoundAtPosition)<<7)&0xFF80)!0x7C; // (4+3)C
18,313 *ShortMediumLongOFFSET=2;
18,314 return;
18,315 }
18,316 }
18,317
18,318 } // B-TREE heuristic
18,319
18,320 #if defined(BtreeHEURISTIC)
18,321 if ( LastSeenOffset_PseudoPointer[8] != 0 )
18,322 //int MatchLens[MatchLensNUMdummy]={4,6,8,10,12,14,16,18,36,64,24,28,48,56,128,256};
18,323 //0,1,2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f
18,324 #endif
18,325 { // Search into ... If 36 is not found then skip all 36[+] - see the roster above.
18,326
```

```

18,327 // 44:2+1=14.6      (512B)
18,328 if (refStartSW512 >= refStart) {
18,329 FoundAtPosition = Railgun_BawBaw_reverse (refStartSW512, encStart, (uint32_t)(refEnd-refStartSW512), 22*2); // Uncommented , 2021-Mar-11, below 4 lines need reversed
file block
18,330 // FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW512 - src) -(uint32_t)(refEnd-refStartSW512 -1), srcR + (srcSize - 1) - (encStart -
src) -(22*2-1), (uint32_t)(refEnd-refStartSW512), 22*2);
18,331 // if (FoundAtPosition2!=NULL) {
18,332 // FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(22*2-1);
18,333 // } else FoundAtPosition = NULL;
18,334
18,335 if (FoundAtPosition!=NULL) {
18,336     *retMatch=22*2;
18,337     // The first four bits should be:
18,338     *retIndex=((refEnd-FoundAtPosition)<<7)&0xFF80)!0x6C; // (4+3)C
18,339     *ShortMediumLongOFFSET=2;
18,340     *HowManyDittoTagsToEmit = 1;
18,341     return;
18,342 }
18,343 }
18,344
18,345 } // B-TREE heuristic
18,346
18,347 #if defined(BtreeHEURISTIC)
18,348 if ( LastSeenOffset_PseudoPointer[9] != 0 )
18,349 //int MatchLens[MatchLensNUMdummy]={4,6,8,10,12,14,16,18,36,64,24,28,48,56,128,256};
18,350 /// 0,1,2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f
18,351 #endif
18,352 { // Search into ... If 64 is not found then skip all 64[+] - see the roster above.
18,353
18,354 // #05a 72:3+1+1=14.4      (1MB)
18,355 if (refStartSW3 >= refStart) {
18,356 FoundAtPosition = Railgun_BawBaw_reverse (refStartSW3, encStart, (uint32_t)(refEnd-refStartSW3), 24*3); // Uncommented , 2021-Mar-11, below 4 lines need reversed file
block
18,357 // FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW3 - src) -(uint32_t)(refEnd-refStartSW3 -1), srcR + (srcSize - 1) - (encStart - src)
-(24*3-1), (uint32_t)(refEnd-refStartSW3), 24*3);
18,358 // if (FoundAtPosition2!=NULL) {
18,359 // FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(24*3-1);
18,360 // } else FoundAtPosition = NULL;
18,361
18,362 if (FoundAtPosition!=NULL) {
18,363     *retMatch=24*3;
18,364     // The first four bits should be:
18,365                                     // 0100b = 4
18,366     *retIndex=((refEnd-FoundAtPosition)<<4)&0xFFFFF0)!0x0004; // xx ... x[OOLL]
18,367     *ShortMediumLongOFFSET=3;
18,368     *HowManyDittoTagsToEmit = 2;
18,369     return;
18,370 }
18,371 }
18,372
18,373 } // B-TREE heuristic
18,374
18,375 #if defined(BtreeHEURISTIC)
18,376 if ( LastSeenOffset_PseudoPointer[13] != 0 ) // MatchLens[MatchLensNUM]={4,6,8,10,12,14,16,18,36,64,24,28,48};
18,377 #endif
18,378 { // Search into ... If 48 is not found then skip all 48[+] - see the roster above.
18,379
18,380 // #06 56:(2+1)+1=      14      (8KB)

```

```

18,381 if (refStartSW2ryuu >= refStart) {
18,382
18,383 #ifndef BtreeHEURISTIC
18,384 //      FoundAtPosition = Railgun_BawBaw_reverse (refStartSW2ryuu, encStart, (uint32_t)(refEnd-refStartSW2ryuu), 28*2);
18,385 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW2ryuu - src) - (uint32_t)(refEnd-refStartSW2ryuu - 1), srcR + (srcSize - 1) - (encStart - src)
-(28*2-1), (uint32_t)(refEnd-refStartSW2ryuu), 28*2);
18,386 if (FoundAtPosition2!=NULL) {
18,387 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(28*2-1);
18,388 } else FoundAtPosition = NULL;
18,389 #else
18,390 if ( refStartSW2ryuu <= (char *)LastSeenOffset_PseudoPointer[13] ) {
18,391 if ( refEnd - (char *)LastSeenOffset_PseudoPointer[13] < (56) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
first position of the LookAhead!
18,392 // Fallback to memmem(): [
18,393 //      FoundAtPosition = Railgun_BawBaw_reverse (refStartSW2ryuu, encStart, (uint32_t)(refEnd-refStartSW2ryuu), 28*2);
18,394 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW2ryuu - src) - (uint32_t)(refEnd-refStartSW2ryuu - 1), srcR + (srcSize - 1) - (encStart - src)
-(28*2-1), (uint32_t)(refEnd-refStartSW2ryuu), 28*2);
18,395 if (FoundAtPosition2!=NULL) {
18,396 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(28*2-1);
18,397 } else FoundAtPosition = NULL;
18,398 // Fallback to memmem(): ]
18,399 } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[13];
18,400 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
%p,%p\n", refStartSW3,refEnd, FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0]); //debugprint
18,401 } else FoundAtPosition = NULL;
18,402 #endif
18,403
18,404 if (FoundAtPosition!=NULL) {
18,405     *retMatch=28*2;
18,406     // The first four bits should be:
18,407                                     // 1000b = 8
18,408     *retIndex=((refEnd-FoundAtPosition)<<3)&0xFFF8)!0x07; // 111, 32:(2+1)=10.6
18,409     *ShortMediumLongOFFSET=2+1 +3333;
18,410     *HowManyDittoTagsToEmit = 1;
18,411     return;
18,412 }
18,413 }
18,414
18,415 } // B-TREE heuristic
18,416
18,417 #if defined(BtreeHEURISTIC)
18,418 if ( LastSeenOffset_PseudoPointer[9] != 0 )
18,419 #endif
18,420 { // Search into ... If 64 is not found then skip all 64[+] - see the roster above.
18,421
18,422 // #06a 96:4+1+1+1=13.7 (256MB)
18,423 if (refStartSW4 >= refStart) {
18,424 //      FoundAtPosition = Railgun_BawBaw_reverse (refStartSW4, encStart, (uint32_t)(refEnd-refStartSW4), 24*4);
18,425 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW4 - src) - (uint32_t)(refEnd-refStartSW4 - 1), srcR + (srcSize - 1) - (encStart - src) -(24*4-
1), (uint32_t)(refEnd-refStartSW4), 24*4);
18,426 if (FoundAtPosition2!=NULL) {
18,427 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(24*4-1);
18,428 } else FoundAtPosition = NULL;
18,429
18,430 if (FoundAtPosition!=NULL) {
18,431     *retMatch=24*4;
18,432     // The first four bits should be:
18,433                                     // 0000b = 0x0; 6<<2

```



```

18,434         *retIndex=((refEnd-FoundAtPosition)<<4)&0xFFFFFFFF!0x00; // xx ... x[LL00]
18,435         *ShortMediumLongOFFSET=4;
18,436         *HowManyDittoTagsToEmit = 3;
18,437         return;
18,438     }
18,439 }
18,440
18,441 // #06b 72:4+1+1=12      (256MB)
18,442 if (refStartSW4 >= refStart) {
18,443     // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW4, encStart, (uint32_t)(refEnd-refStartSW4), 24*3);
18,444     FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW4 - src) - (uint32_t)(refEnd-refStartSW4 - 1), srcR + (srcSize - 1) - (encStart - src) - (24*3-1), (uint32_t)(refEnd-refStartSW4), 24*3);
18,445     if (FoundAtPosition2!=NULL) {
18,446         FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (24*3-1);
18,447     } else FoundAtPosition = NULL;
18,448
18,449     if (FoundAtPosition!=NULL) {
18,450         *retMatch=24*3;
18,451         // The first four bits should be:
18,452
18,453         *retIndex=((refEnd-FoundAtPosition)<<4)&0xFFFFFFFF!0x00; // xx ... x[LL00]
18,454         *ShortMediumLongOFFSET=4;
18,455         *HowManyDittoTagsToEmit = 3;
18,456         return;
18,457     }
18,458 }
18,459
18,460     } // B-TREE heuristic
18,461
18,462 #if defined(BtreeHEURISTIC)
18,463     if ( LastSeenOffset_PseudoPointer[12] != 0 )
18,464 #endif
18,465     { // Search into ... If 48 is not found then skip all 48[+] - see the roster above.
18,466
18,467     // #06c 48:3+1=12      (1MB)
18,468     if (refStartSW3 >= refStart) {
18,469
18,470     #ifndef BtreeHEURISTIC
18,471     // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW3, encStart, (uint32_t)(refEnd-refStartSW3), 24*2);
18,472     FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW3 - src) - (uint32_t)(refEnd-refStartSW3 - 1), srcR + (srcSize - 1) - (encStart - src) - (24*2-1), (uint32_t)(refEnd-refStartSW3), 24*2);
18,473     if (FoundAtPosition2!=NULL) {
18,474         FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (24*2-1);
18,475     } else FoundAtPosition = NULL;
18,476 #else
18,477     if ( refStartSW3 <= (char *)LastSeenOffset_PseudoPointer[12] ) {
18,478         if ( refEnd - (char *)LastSeenOffset_PseudoPointer[12] < (48) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the first position of the LookAhead!
18,479             // Fallback to memmem(): [
18,480             // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW3, encStart, (uint32_t)(refEnd-refStartSW3), 24*2);
18,481             FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW3 - src) - (uint32_t)(refEnd-refStartSW3 - 1), srcR + (srcSize - 1) - (encStart - src) - (24*2-1), (uint32_t)(refEnd-refStartSW3), 24*2);
18,482             if (FoundAtPosition2!=NULL) {
18,483                 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (24*2-1);
18,484             } else FoundAtPosition = NULL;
18,485             // Fallback to memmem(): ]
18,486             } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[12];
18,487 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:

```

```

%p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
18,488     } else FoundAtPosition = NULL;
18,489 #endif
18,490
18,491     if (FoundAtPosition!=NULL) {
18,492         *retMatch=24*2;
18,493         // The first four bits should be:
18,494                                     // 0100b = 4
18,495         *retIndex=((refEnd-FoundAtPosition)<<4)&0xFFFF0!0x0004; // xx ... x[00LL]
18,496         *ShortMediumLongOFFSET=3;
18,497         *HowManyDittoTagsToEmit = 1;
18,498         return;
18,499     }
18,500 }
18,501
18,502     } // B-TREE heuristic
18,503
18,504 #if defined(BtreeHEURISTIC)
18,505     if ( LastSeenOffset_PseudoPointer[10] != 0 ) // 2020-Jan-01
18,506 #endif
18,507     { // Search into ... If 18 is not found then skip all 18[+] - see the roster above.
18,508
18,509     // #07 24:2=          12      (4KB)
18,510     /*
18,511     // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) [
18,512     if (refStartHOTTER >= refStart)
18,513     if (refStartHOTTER < refEnd) {
18,514     FoundAtPosition = Railgun_Trolldom_64(refStartHOTTER, encStart, (uint32_t)(refEnd-refStartHOTTER), 24);
18,515     if (FoundAtPosition!=NULL) {
18,516         *retMatch=24;
18,517         // The first four bits should be:
18,518                                     // 1000b = 8
18,519         *retIndex=((refEnd-FoundAtPosition)<<4)&0xFFF0!0x0008; // xx ... x[00LL]
18,520         *ShortMediumLongOFFSET=2;
18,521         return;
18,522     }
18,523 }
18,524 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) ]
18,525 */
18,526 if (refStartSW2 >= refStart) {
18,527 // 2020-Jan-01 [
18,528 #ifndef BtreeHEURISTIC
18,529 // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW2, encStart, (uint32_t)(refEnd-refStartSW2), 24);
18,530 FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW2 - src) -(uint32_t)(refEnd-refStartSW2 -1), srcR + (srcSize - 1) - (encStart - src) -(24-1), (uint32_t)(refEnd-refStartSW2), 24);
18,531 if (FoundAtPosition2!=NULL) {
18,532 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(24-1);
18,533 } else FoundAtPosition = NULL;
18,534 #else
18,535 if ( refStartSW2 <= (char *)LastSeenOffset_PseudoPointer[10] ) {
18,536 if ( refEnd - (char *)LastSeenOffset_PseudoPointer[10] < (24) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the first position of the LookAhead!
18,537 // Fallback to memmem(): [
18,538 // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW2, encStart, (uint32_t)(refEnd-refStartSW2), 24);
18,539 FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW2 - src) -(uint32_t)(refEnd-refStartSW2 -1), srcR + (srcSize - 1) - (encStart - src) -(24-1), (uint32_t)(refEnd-refStartSW2), 24);
18,540 if (FoundAtPosition2!=NULL) {
18,541 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(24-1);

```

```

18,542         } else FoundAtPosition = NULL;
18,543         // Fallback to memmem(): ]
18,544         } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[10];
18,545 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
%p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
18,546         } else FoundAtPosition = NULL;
18,547 #endif
18,548 // 2020-Jan-01 ]
18,549
18,550         if (FoundAtPosition!=NULL) {
18,551             *retMatch=24;
18,552             // The first four bits should be:
18,553                                     // 1000b = 8
18,554             *retIndex=((refEnd-FoundAtPosition)<<4)&0xFFF0)!0x0008; // xx ... x[OOLL]
18,555             *ShortMediumLongOFFSET=2;
18,556             return;
18,557         }
18,558     }
18,559     } // B-TREE heuristic
18,560
18,561 // #08 12:1=          12      (16B)
18,562 /*
18,563 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) [
18,564 if (refStartHOTEST >= refStart)
18,565 if (refStartHOTEST < refEnd) {
18,566     FoundAtPosition = Railgun_Trolldom_64(refStartHOTEST, encStart, (uint32_t)(refEnd-refStartHOTEST), 12);
18,567     if (FoundAtPosition!=NULL) {
18,568         *retMatch=12;
18,569         // The first four bits should be:
18,570                                     // 0111b = 7
18,571         *retIndex=((refEnd-FoundAtPosition)<<4)&0xFFF0)!0x0007; // xx ... x[OOLL]
18,572         *ShortMediumLongOFFSET=1;
18,573         return;
18,574     }
18,575 }
18,576 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) ]
18,577 */
18,578 #if defined(BtreeHEURISTIC)
18,579     if ( LastSeenOffset_PseudoPointer[4] != 0 )
18,580 #endif
18,581 { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
18,582     if (refStartSW1 >= refStart) {
18,583
18,584 #ifndef BtreeHEURISTIC
18,585 // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW1, encStart, (uint32_t)(refEnd-refStartSW1), 12);
18,586 FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW1 - src) -(uint32_t)(refEnd-refStartSW1 -1), srcR + (srcSize - 1) - (encStart - src) -(12-1), (uint32_t)(refEnd-refStartSW1), 12);
18,587 if (FoundAtPosition2!=NULL) {
18,588 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(12-1);
18,589 } else FoundAtPosition = NULL;
18,590 #else
18,591 if ( refStartSW1 <= (char *)LastSeenOffset_PseudoPointer[4] ) {
18,592 if ( refEnd - (char *)LastSeenOffset_PseudoPointer[4] < (12) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
first position of the LookAhead!
18,593 // Fallback to memmem(): [
18,594 FoundAtPosition = Railgun_BawBaw_reverse (refStartSW1, encStart, (uint32_t)(refEnd-refStartSW1), 12);

```

```

18,597 FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW1 - src) -(uint32_t)(refEnd-refStartSW1 -1), srcR + (srcSize - 1) - (encStart - src) -(12-
1), (uint32_t)(refEnd-refStartSW1), 12);
18,598     if (FoundAtPosition2!=NULL) {
18,599         FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(12-1);
18,600     } else FoundAtPosition = NULL;
18,601         // Fallback to memmem(): ]
18,602     } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[4];
18,603 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
%p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
18,604     } else FoundAtPosition = NULL;
18,605 #endif
18,606
18,607 // DEBUGged [ ----- THE BUG FIX: ADDING -1 AVOIDS OVERLAPPING -----
18,608 /*
18,609 //
18,610 FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW1 - src) -(uint32_t)(refEnd-refStartSW1 -1), srcR + (srcSize - 1) - (encStart - src) -(12-
1), (uint32_t)(refEnd-refStartSW1), 12);
18,611     if (FoundAtPosition2!=NULL) {
18,612
18,613         printf("\n 1 = %d\n", (uint32_t)( (srcSize - 1) - (refStartSW1 - src) -(uint32_t)(refEnd-refStartSW1) ));
18,614         printf(" 2 = %d\n", (uint32_t)( (srcSize - 1) - (encStart - src) -(12-1) ));
18,615
18,616         if ( FoundAtPosition != src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(12-1) ) {
18,617
18,618             printf("\n %p refStartSW1:                                [", refStartSW1);
18,619             for (i=0; i<15; i++) {
18,620                 printf("%c",*(refStartSW1 +i));
18,621             }
18,622             printf("]");
18,623             printf("\n %p srcR + (srcSize - 1) - (refStartSW1 - src) -(uint32_t)(refEnd-refStartSW1): [", srcR + (srcSize - 1) - (refStartSW1 - src) -(uint32_t)(refEnd-
refStartSW1));
18,624             for (i=0; i<15; i++) {
18,625                 printf("%c",*(srcR + (srcSize - 1) - (refStartSW1 - src) -(uint32_t)(refEnd-refStartSW1) +i));
18,626             }
18,627             printf("]\n");
18,628
18,629
18,630             printf("\n %p encStart:                                [", encStart);
18,631             for (i=0; i<12; i++) {
18,632                 printf("%c",*(encStart +i));
18,633             }
18,634             printf("]");
18,635             printf("\n %p srcR + (srcSize - 1) - (encStart - src) -(12-1): [", srcR + (srcSize - 1) - (encStart - src) -(12-1));
18,636             for (i=0; i<12; i++) {
18,637                 printf("%c",*(srcR + (srcSize - 1) - (encStart - src) -(12-1) +i));
18,638             }
18,639             printf("]\n");
18,640         */
18,641
18,642 //printf("\n (uint32_t)(refEnd-refStartSW1)=%d\n", (uint32_t)(refEnd-refStartSW1)); //(uint32_t)(refEnd-refStartSW1)=15
18,643
18,644 // 1 = 200108
18,645 // 2 = 200097
18,646
18,647 // 000000000043640D refStartSW1:                                [.' Something, s]
18,648 // 000000000049818C srcR + (srcSize - 1) - (refStartSW1 - src) -(uint32_t)(refEnd-refStartSW1): [os ,gnihtemoS ']
18,649
18,650 // 000000000043641C encStart:                                [omething, so]

```

```

18,651 // 0000000000498181 srcR + (srcSize - 1) - (encStart - src) -(12-1): [os ,gnihtemo]
18,652
18,653 //12:1 FoundAtPosition = 0000000000000000
18,654
18,655 //FoundAtPosition2 = 000000000049818C
18,656 //Mismatch!
18,657
18,658
18,659 //
18,660 // 000000000043640D refStartSW1:
18,661 //
18,662 // 000000000043641C encStart:
18,663 //
18,664 // 000000000049818C srcR + (srcSize - 1) - (refStartSW1 - src) -(uint32_t)(refEnd-refStartSW1):
18,665 //
18,666 // 0000000000498181 srcR + (srcSize - 1) - (encStart - src) -(12-1):
18,667
18,668 // 'omething, so'
18,669 // 0812 line: "She said, 'Master Ridley.' She said some crazy thing when we came in the door. 'Play the man,' she said, 'Master Ridley.' Something, something, something."
18,670 // 3127 line: But now there was a long morning's walk until noon, and if the men were silent it was because there was everything to think about and much to remember. Perhaps later in the morning, when the sun was up and had warmed them, they would begin to talk, or just say the things they remembered, to be sure they were there, to be absolutely certain that things were safe in them. Montag felt the slow stir of words, the slow simmer. And when it came to his turn, what could he say, what could he offer on a day like this, to make the trip a little easier? To everything there is a season. Yes. A time to break down, and a time to build up. Yes. A time to keep silence and a time to speak. Yes, all that. But what else. What else? Something, something...
18,671
18,672 // printf("\n12:1 FoundAtPosition = %p\n",FoundAtPosition);
18,673 // printf("\nFoundAtPosition2 = %p\n",FoundAtPosition2);
18,674
18,675 //printf ("Mismatch!\n"); exit(13);}
18,676 // DEBUGed ]
18,677
18,678     if (FoundAtPosition!=NULL) {
18,679         *retMatch=12;
18,680         // The first four bits should be:
18,681
18,682         *retIndex=(((refEnd-FoundAtPosition)<<4)&0xF0)!0x0007; // xx ... x[00LL]
18,683         *ShortMediumLongOFFSET=1;
18,684         return;
18,685     }
18,686 }
18,687
18,688     } // B-TREE heuristic
18,689
18,690 #if defined(BtreeHEURISTIC)
18,691     if ( LastSeenOffset_PseudoPointer[13] != 0 ) // MatchLens[MatchLensNUM]={4,6,8,10,12,14,16,18,36,64,24,28,48};
18,692 #endif
18,693     { // Search into ... If 48 is not found then skip all 48[+] - see the roster above.
18,694
18,695     //09 56:(3+1)+1= 11.2 (2MB)
18,696     if (refStartSW3ryuu >= refStart) {
18,697
18,698     #ifndef BtreeHEURISTIC
18,699     // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW3ryuu, encStart, (uint32_t)(refEnd-refStartSW3ryuu), 28*2);
18,700     FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW3ryuu - src) -(uint32_t)(refEnd-refStartSW3ryuu -1), srcR + (srcSize - 1) - (encStart - src) -(28*2-1), (uint32_t)(refEnd-refStartSW3ryuu), 28*2);
18,701     if (FoundAtPosition2!=NULL) {
18,702     FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(28*2-1);

```

```

18,703         } else FoundAtPosition = NULL;
18,704 #else
18,705         if ( refStartSW3ryuu <= (char *)LastSeenOffset_PseudoPointer[13] ) {
18,706             if ( refEnd - (char *)LastSeenOffset_PseudoPointer[13] < (56) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
first position of the LookAhead!
18,707                 // Fallback to memmem(): [
18,708                 FoundAtPosition = Railgun_BawBaw_reverse( refStartSW3ryuu, encStart, (uint32_t)(refEnd-refStartSW3ryuu), 28*2);
18,709                 FoundAtPosition2 = Railgun_Trollldom_64( srcR + (srcSize - 1) - (refStartSW3ryuu - src) -(uint32_t)(refEnd-refStartSW3ryuu -1), srcR + (srcSize - 1) - (encStart - src)
-(28*2-1), (uint32_t)(refEnd-refStartSW3ryuu), 28*2);
18,710                 if (FoundAtPosition2!=NULL) {
18,711                     FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(28*2-1);
18,712                 } else FoundAtPosition = NULL;
18,713                 // Fallback to memmem(): ]
18,714                 } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[13];
18,715 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
%p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
18,716                 } else FoundAtPosition = NULL;
18,717 #endif
18,718
18,719         if (FoundAtPosition!=NULL) {
18,720             *retMatch=28*2;
18,721             // The first four bits should be:
18,722                                     // 0101b = 5
18,723             *retIndex=((refEnd-FoundAtPosition)<<3)&0xFFFFF8)!0x06; // 011, 32:(3+1)=8
18,724             *ShortMediumLongOFFSET=3+1+3333;
18,725             *HowManyDittoTagsToEmit = 1;
18,726             return;
18,727         }
18,728     }
18,729
18,730     } // B-TREE heuristic
18,731
18,732 #if defined(BtreeHEURISTIC)
18,733     if ( LastSeenOffset_PseudoPointer[8] != 0 ) // MatchLens[MatchLensNUM]={4,6,8,10,12,14,16,18,36,64,24,28,48};
18,734     //int MatchLens[MatchLensNUMdummy]={4,6,8,10,12,14,16,18,36,64,24,28,48,56,128,256};
18,735     /// 0,1,2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f
18,736 #endif
18,737     { // Search into ... If 36 is not found then skip all 36[+] - see the roster above.
18,738
18,739     // 44:3+1=11 (128KB)
18,740     if (refStartSW128kb >= refStart) {
18,741         FoundAtPosition = Railgun_BawBaw_reverse( refStartSW128kb, encStart, (uint32_t)(refEnd-refStartSW128kb), 22*2); // Uncommented , 2021-Mar-11, below 4 lines need
reversed file block
18,742         FoundAtPosition2 = Railgun_Trollldom_64( srcR + (srcSize - 1) - (refStartSW128kb - src) -(uint32_t)(refEnd-refStartSW128kb -1), srcR + (srcSize - 1) -
(encStart - src) -(22*2-1), (uint32_t)(refEnd-refStartSW128kb), 22*2);
18,743         if (FoundAtPosition2!=NULL) {
18,744             FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(22*2-1);
18,745         } else FoundAtPosition = NULL;
18,746
18,747         if (FoundAtPosition!=NULL) {
18,748             *retMatch=22*2;
18,749             // The first four bits should be:
18,750             *retIndex=((refEnd-FoundAtPosition)<<7)&0xFFFF80)!0x2C;
18,751             *ShortMediumLongOFFSET=3;
18,752             *HowManyDittoTagsToEmit = 1;
18,753             return;
18,754         }
18,755     }

```

```

18,756
18,757     } // B-TREE heuristic
18,758
18,759 #if defined(BtreeHEURISTIC)
18,760     if ( LastSeenOffset_PseudoPointer[7] != 0 )
18,761         //int MatchLens[MatchLensNUMdummy]={4,6,8,10,12,14,16,18,36,64,24,28,48,56,128,256};
18,762         ///          0,1,2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f
18,763 #endif
18,764     { // Search into ... If 18 is not found then skip all 18[+] - see the roster above.
18,765
18,766     // #10 22:2=      11      (512B)
18,767     if (refStartSW512 >= refStart) {
18,768         FoundAtPosition = Railgun_BawBaw_reverse (refStartSW512, encStart, (uint32_t)(refEnd-refStartSW512), 22); // Uncommented , 2021-Mar-11, below 4 lines need reversed
18,769         // FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW512 - src) - (uint32_t)(refEnd-refStartSW512 - 1), srcR + (srcSize - 1) - (encStart -
18,770         // src) - (22-1), (uint32_t)(refEnd-refStartSW512), 22);
18,771         //         if (FoundAtPosition2!=NULL) {
18,772         //             FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (22-1);
18,773         //         } else FoundAtPosition = NULL;
18,774
18,775         if (FoundAtPosition!=NULL) {
18,776             *retMatch=22;
18,777             // The first four bits should be:
18,778             *retIndex=((refEnd-FoundAtPosition)<<7)&0xFF80!0x6C; // (4+2)C
18,779             *ShortMediumLongOFFSET=2;
18,780             return;
18,781         }
18,782     }
18,783     } // B-TREE heuristic
18,784
18,785 #if defined(BtreeHEURISTIC)
18,786     if ( LastSeenOffset_PseudoPointer[9] != 0 )
18,787 #endif
18,788     { // Search into ... If 64 is not found then skip all 64[+] - see the roster above.
18,789
18,790     // #11 64:(5+1)=      10.6      (1TB)
18,791     if (refStartSW6ryuu >= refStart) {
18,792
18,793     #ifndef BtreeHEURISTIC
18,794         // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW6ryuu, encStart, (uint64_t)(refEnd-refStartSW6ryuu), 64);
18,795         FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW6ryuu - src) - (uint64_t)(refEnd-refStartSW6ryuu - 1), srcR + (srcSize - 1) - (encStart - src)
18,796         // -(64-1), (uint64_t)(refEnd-refStartSW6ryuu), 64);
18,797         //         if (FoundAtPosition2!=NULL) {
18,798         //             FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (64-1);
18,799         //         } else FoundAtPosition = NULL;
18,800     #else
18,801         if ( refStartSW6ryuu <= (char *)LastSeenOffset_PseudoPointer[9] ) {
18,802             if ( refEnd - (char *)LastSeenOffset_PseudoPointer[9] < (64) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
18,803             // first position of the LookAhead!
18,804             // Fallback to memmem(): [
18,805             FoundAtPosition = Railgun_BawBaw_reverse (refStartSW6ryuu, encStart, (uint64_t)(refEnd-refStartSW6ryuu), 64);
18,806             FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW6ryuu - src) - (uint64_t)(refEnd-refStartSW6ryuu - 1), srcR + (srcSize - 1) - (encStart - src)
18,807             // -(64-1), (uint64_t)(refEnd-refStartSW6ryuu), 64);
18,808             //         if (FoundAtPosition2!=NULL) {
18,809             //             FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (64-1);
18,810             //         } else FoundAtPosition = NULL;
18,811             // Fallback to memmem(): ]

```

```

18,809             } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[9];
18,810 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
%p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
18,811             } else FoundAtPosition = NULL;
18,812 #endif
18,813
18,814             if (FoundAtPosition!=NULL) {
18,815                 *retMatch=64;
18,816                 // The first 8 bits should be 0x93, in order to retain 4bytes-longness of 'retIndex' we play dirty - not tagging the 5bytes chunk with 0x93
but as "usual" - the address subchunk:
18,817                 *retIndex=(((refEnd-FoundAtPosition)<<0)&0xFFFFFFFF);
18,818                 *ShortMediumLongOFFSET=5+1+9999;
18,819                 return;
18,820             }
18,821 }
18,822
18,823     } // B-TREE heuristic
18,824
18,825 #if defined(BtreeHEURISTIC)
18,826     if ( LastSeenOffset_PseudoPointer[11] != 0 ) // 2020-Jan-01
18,827         //int MatchLens[MatchLensNUMdummy]={4,6,8,10,12,14,16,18,36,64,24,28,48,56,128,256};
18,828         ///
18,829         //0,1,2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f
18,829 #endif
18,830     { // Search into ... If 18 is not found then skip all 18[+] - see the roster above.
18,831
18,832 //#12 30:3=      10 (128KB)
18,833     if (refStartSW128kb >= refStart) {
18,834         FoundAtPosition = Railgun_BawBaw_reverse (refStartSW128kb, encStart, (uint32_t)(refEnd-refStartSW128kb), 30); // Uncommented , 2021-Mar-11, below 4 lines need
reversed file block
18,835 //         FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW128kb - src) -(uint32_t)(refEnd-refStartSW128kb -1), srcR + (srcSize - 1) -
(encStart - src) -(30-1), (uint32_t)(refEnd-refStartSW128kb), 30);
18,836 //         if (FoundAtPosition2!=NULL) {
18,837 //             FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(30-1);
18,838 //         } else FoundAtPosition = NULL;
18,839
18,840         if (FoundAtPosition!=NULL) {
18,841             *retMatch=30;
18,842             // The first four bits should be:
18,843             *retIndex=(((refEnd-FoundAtPosition)<<7)&0xFFFF80)|0x3C;
18,844             *ShortMediumLongOFFSET=3;
18,845             return;
18,846         }
18,847     }
18,848
18,849     } // B-TREE heuristic
18,850
18,851 #if defined(BtreeHEURISTIC)
18,852     if ( LastSeenOffset_PseudoPointer[12] != 0 )
18,853 #endif
18,854     { // Search into ... If 48 is not found then skip all 48[+] - see the roster above.
18,855
18,856 //#12a 48:4+1=9.6      (256MB)
18,857     if (refStartSW4 >= refStart) {
18,858
18,859 #ifndef BtreeHEURISTIC
18,860 //         FoundAtPosition = Railgun_BawBaw_reverse (refStartSW4, encStart, (uint32_t)(refEnd-refStartSW4), 24*2);
18,861         FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW4 - src) -(uint32_t)(refEnd-refStartSW4 -1), srcR + (srcSize - 1) - (encStart - src) -(24*2-
1), (uint32_t)(refEnd-refStartSW4), 24*2);

```



```

18,862         if (FoundAtPosition2!=NULL) {
18,863     FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(24*2-1);
18,864         } else FoundAtPosition = NULL;
18,865 #else
18,866         if ( refStartSW4 <= (char *)LastSeenOffset_PseudoPointer[12] ) {
18,867             if ( refEnd - (char *)LastSeenOffset_PseudoPointer[12] < (48) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
first position of the LookAhead!
18,868                 // Fallback to memmem(): [
18,869 //         FoundAtPosition = Railgun_BawBaw_reverse (refStartSW4, encStart, (uint32_t)(refEnd-refStartSW4), 24*2);
18,870     FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW4 - src) -(uint32_t)(refEnd-refStartSW4 -1), srcR + (srcSize - 1) - (encStart - src) -(24*2-
1), (uint32_t)(refEnd-refStartSW4), 24*2);
18,871         if (FoundAtPosition2!=NULL) {
18,872     FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(24*2-1);
18,873         } else FoundAtPosition = NULL;
18,874             // Fallback to memmem(): ]
18,875             } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[12];
18,876 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
%p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
18,877         } else FoundAtPosition = NULL;
18,878 #endif
18,879
18,880         if (FoundAtPosition!=NULL) {
18,881             *retMatch=24*2;
18,882             // The first four bits should be:
18,883                                     // 0000b = 0x0; 6<<2
18,884             *retIndex=((refEnd-FoundAtPosition)<<4)&0xFFFFFFFF)!0x00; // xx ... x[LL00]
18,885             *ShortMediumLongOFFSET=4;
18,886             *HowManyDittoTagsToEmit = 1;
18,887             return;
18,888         }
18,889     }
18,890
18,891     } // B-TREE heuristic
18,892
18,893
18,894 #if defined(BtreeHEURISTIC)
18,895     if ( LastSeenOffset_PseudoPointer[13] != 0 )
18,896 #endif
18,897     { // Search into ... If 48 is not found then skip all 48[+] - see the roster above.
18,898
18,899 //#13 56:(4+1)+1=          9.3 (512MB)
18,900     if (refStartSW5 >= refStart) {
18,901
18,902 #ifndef BtreeHEURISTIC
18,903 //         FoundAtPosition = Railgun_BawBaw_reverse (refStartSW5, encStart, (uint32_t)(refEnd-refStartSW5), 28*2);
18,904     FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW5 - src) -(uint32_t)(refEnd-refStartSW5 -1), srcR + (srcSize - 1) - (encStart - src) -(28*2-
1), (uint32_t)(refEnd-refStartSW5), 28*2);
18,905         if (FoundAtPosition2!=NULL) {
18,906     FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(28*2-1);
18,907         } else FoundAtPosition = NULL;
18,908 #else
18,909         if ( refStartSW5 <= (char *)LastSeenOffset_PseudoPointer[13] ) {
18,910             if ( refEnd - (char *)LastSeenOffset_PseudoPointer[13] < (56) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
first position of the LookAhead!
18,911                 // Fallback to memmem(): [
18,912 //         FoundAtPosition = Railgun_BawBaw_reverse (refStartSW5, encStart, (uint32_t)(refEnd-refStartSW5), 28*2);
18,913     FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW5 - src) -(uint32_t)(refEnd-refStartSW5 -1), srcR + (srcSize - 1) - (encStart - src) -(28*2-
1), (uint32_t)(refEnd-refStartSW5), 28*2);

```

```

18,914         if (FoundAtPosition2!=NULL) {
18,915 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(28*2-1);
18,916         } else FoundAtPosition = NULL;
18,917         // Fallback to memmem(): ]
18,918         } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[13];
18,919 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
%p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
18,920         } else FoundAtPosition = NULL;
18,921 #endif
18,922
18,923         if (FoundAtPosition!=NULL) {
18,924             *retMatch=28*2;
18,925             // The first 8 bits should be 0x93, in order to retain 4bytes-longness of 'retIndex' we play dirty - not tagging the 5bytes chunk with 0x93
but as "usual" - the address subchunk:
18,926             *retIndex=(((refEnd-FoundAtPosition)<<3)&0xFFFFFFFF8)!0x05; // 101, 32:(4+1)=6.4
18,927             *ShortMediumLongOFFSET=4+1 +3333;
18,928             *HowManyDittoTagsToEmit = 1;
18,929             return;
18,930         }
18,931     }
18,932
18,933     } // B-TREE heuristic
18,934
18,935
18,936 #if defined(BtreeHEURISTIC)
18,937     if ( LastSeenOffset_PseudoPointer[11] != 0 ) // 2020-Jan-01
18,938 #endif
18,939     { // Search into ... If 18 is not found then skip all 18[+] - see the roster above.
18,940
18,941 //#14 28:(2+1)=          9.3   (8KB)
18,942 if (refStartSW2ryuu >= refStart) {
18,943
18,944 // 2020-Jan-01 [
18,945 #ifndef BtreeHEURISTIC
18,946 // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW2ryuu, encStart, (uint32_t)(refEnd-refStartSW2ryuu), 28);
18,947 FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW2ryuu - src) -(uint32_t)(refEnd-refStartSW2ryuu -1), srcR + (srcSize - 1) - (encStart - src)
-(28-1), (uint32_t)(refEnd-refStartSW2ryuu), 28);
18,948         if (FoundAtPosition2!=NULL) {
18,949 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(28-1);
18,950         } else FoundAtPosition = NULL;
18,951 #else
18,952         if ( refStartSW2ryuu <= (char *)LastSeenOffset_PseudoPointer[11] ) {
18,953             if ( refEnd - (char *)LastSeenOffset_PseudoPointer[11] < (28) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
first position of the LookAhead!
18,954 // 2020-Feb-02: No need of FALLBACK to memmem() when 'refEnd - (char *)LastSeenOffset_PseudoPointer[11] == (28)' is true, because when SHA3 is used it will search for
SHA3 in the ... Uncompressed Block!
18,955 // refEnd=40=encStart is LookAheadBuffer
18,956 //
18,957 // 1         2         3         \
18,958 // 12345678901234567890123456789!0
18,959 // Bombastica Playlist Ecstatica!tica
18,960 // [ ] [ ]
18,961 // ^^^ ^^^
18,962 // 36*(4-1) is within 1..39
18,963 // if (40-36 < 4) then memmem()
18,964 // The buffer starts at refStartSW3ryuu and is long (uint32_t)(refEnd-refStartSW3ryuu) or:
18,965 // 1..(40-1) or 1..39
18,966

```

```

18,967 /*
18,968 // debugg [[[[[[[
18,969 printf("refEnd - (char *)LastSeenOffset_PseudoPointer[11] = %d\n", refEnd - (char *)LastSeenOffset_PseudoPointer[11]);
18,970 // 0 1 2 3
18,971 // Src Src+1 Src+2 Src+3
18,972 // refEnd
18,973 // refEnd-Src = 3-0 = 3
18,974 printf("LastSeenOffset_PseudoPointer[11]-src= %x\n", (char *)LastSeenOffset_PseudoPointer[11]-src);
18,975 printf("refEnd-src = %x\n", refEnd-src);
18,976 for (di=(char *)LastSeenOffset_PseudoPointer[11]; di<=refEnd+1; di++)
18,977 printf("[%c]", *di);
18,978 printf("\n");
18,979
18,980 // For alice29.txt:
18,981 //
18,982 // refEnd - (char *)LastSeenOffset_PseudoPointer[11] = 8
18,983 // LastSeenOffset_PseudoPointer[11]-src= 2303 ' * * * *
18,984 // refEnd-src = 230b
18,985 // [ ][ ][ ][*][ ][ ][ ][ ][ ]
18,986 // refEnd - (char *)LastSeenOffset_PseudoPointer[11] = 8
18,987 // LastSeenOffset_PseudoPointer[11]-src= 230b
18,988 // refEnd-src = 2313
18,989 // [ ][ ][ ][*][ ][ ][ ][ ][ ]
18,990 // 22c0 66 65 65 2c 20 61 6e 64 20 68 6f 74 20 62 75 74 74 65 72 65 64 20 74 6f 61 73 74 2c 29 20 73 68 fee, and hot buttered toast,) sh
18,991 // 22e0 65 20 76 65 72 79 20 73 6f 6f 6e 20 66 69 6e 69 73 68 65 64 0d 0a 69 74 20 6f 66 66 2e 0d 0a 0d e very soon finished..it off....
18,992 // 2300 0a 20 20 20 20 20 2a 20 20 20 20 20 20 2a 20 20 20 20 20 20 2a 20 20 20 20 20 20 2a 20 . * * * *
18,993 // ^[ ! ] [ ]
18,994 // 2320 20 20 20 20 20 20 2a 20 20 20 20 20 20 2a 20 20 20 20 20 20 2a 0d 0a 0d 0a 20 20 20 20 20 [ * * *....
18,995 // 2340 20 20 20 20 2a 20 20 20 20 20 20 2a 20 20 20 20 20 2a 20 20 20 20 20 20 2a 20 20 20 * * * *
18,996 // 2360 20 20 20 20 2a 20 20 20 20 20 20 2a 0d 0a 0d 0a 20 20 20 20 2a 20 20 20 20 20 2a 20 * *.... * *
18,997 // 2380 20 20 20 20 20 2a 20 20 20 20 20 2a 20 20 20 20 2a 20 20 20 20 20 20 2a 20 20 2a 20 * * * *
18,998 // 23a0 20 20 20 20 20 2a 0d 0a 0d 0a 20 60 57 68 61 74 20 61 20 63 75 72 69 6f 75 73 20 66 65 65 *.... 'What a curious fee
18,999 // 23c0 6c 69 6e 67 21 27 20 73 61 69 64 20 41 6c 69 63 65 3b 20 60 49 20 6d 75 73 74 20 62 65 20 73 68 ling!' said Alice; 'I must be sh
19,000 // Before commenting the two lines (where updating happens), they have // 2020-Feb-02 tag:
19,001 // Railgun_INVOCATIONS (the-lower-the-better) = 41,640
19,002 // Railgun_INVOCATIONS (for needle 004 bytes) = 21,929
19,003 // Railgun_INVOCATIONS (for needle 006 bytes) = 12 VANISHES
19,004 // Railgun_INVOCATIONS (for needle 008 bytes) = 12,350
19,005 // Railgun_INVOCATIONS (for needle 010 bytes) = 3 VANISHES
19,006 // Railgun_INVOCATIONS (for needle 012 bytes) = 4,426
19,007 // Railgun_INVOCATIONS (for needle 014 bytes) = 6 VANISHES
19,008 // Railgun_INVOCATIONS (for needle 016 bytes) = 1 VANISHES
19,009 // Railgun_INVOCATIONS (for needle 022 bytes) = 1,827
19,010 // Railgun_INVOCATIONS (for needle 024 bytes) = 4 VANISHES
19,011 // Railgun_INVOCATIONS (for needle 028 bytes) = 2 <- VANISHES
19,012 // Railgun_INVOCATIONS (for needle 030 bytes) = 346
19,013 // Railgun_INVOCATIONS (for needle 036 bytes) = 2 VANISHES
19,014 // Railgun_INVOCATIONS (for needle 044 bytes) = 208
19,015 // Railgun_INVOCATIONS (for needle 056 bytes) = 167
19,016 // Railgun_INVOCATIONS (for needle 060 bytes) = 95
19,017 // Railgun_INVOCATIONS (for needle 072 bytes) = 71
19,018 // Railgun_INVOCATIONS (for needle 090 bytes) = 54
19,019 // Railgun_INVOCATIONS (for needle 096 bytes) = 73
19,020 // Railgun_INVOCATIONS (for needle 120 bytes) = 27
19,021 // Railgun_INVOCATIONS (for needle 128 bytes) = 29
19,022 // Railgun_INVOCATIONS (for needle 192 bytes) = 2
19,023 // Railgun_INVOCATIONS (for needle 256 bytes) = 2
19,024 // Railgun_INVOCATIONS (for needle 320 bytes) = 2

```

```

19,025 // Railgun_INVOCATIONS (for needle 704 bytes) = 2
19,026 // After commenting the two lines (where updating happens), they have // 2020-Feb-02 tag:
19,027 // Railgun_INVOCATIONS (the-lower-the-better) = 41,579
19,028 // Railgun_INVOCATIONS (for needle 004 bytes) = 21,919
19,029 // Railgun_INVOCATIONS (for needle 008 bytes) = 12,340
19,030 // Railgun_INVOCATIONS (for needle 012 bytes) = 4,415
19,031 // Railgun_INVOCATIONS (for needle 022 bytes) = 1,827
19,032 // Railgun_INVOCATIONS (for needle 030 bytes) = 346
19,033 // Railgun_INVOCATIONS (for needle 044 bytes) = 208
19,034 // Railgun_INVOCATIONS (for needle 056 bytes) = 167
19,035 // Railgun_INVOCATIONS (for needle 060 bytes) = 95
19,036 // Railgun_INVOCATIONS (for needle 072 bytes) = 71
19,037 // Railgun_INVOCATIONS (for needle 090 bytes) = 54
19,038 // Railgun_INVOCATIONS (for needle 096 bytes) = 73
19,039 // Railgun_INVOCATIONS (for needle 120 bytes) = 27
19,040 // Railgun_INVOCATIONS (for needle 128 bytes) = 29
19,041 // Railgun_INVOCATIONS (for needle 192 bytes) = 2
19,042 // Railgun_INVOCATIONS (for needle 256 bytes) = 2
19,043 // Railgun_INVOCATIONS (for needle 320 bytes) = 2
19,044 // Railgun_INVOCATIONS (for needle 704 bytes) = 2
19,045 // debugg ]]]]]]]
19,046 */
19,047 // Fallback to memmem(): [
19,048 // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW2ryuu, encStart, (uint32_t)(refEnd-refStartSW2ryuu), 28);
19,049 FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW2ryuu - src) - (uint32_t)(refEnd-refStartSW2ryuu - 1), srcR + (srcSize - 1) - (encStart - src)
-(28-1), (uint32_t)(refEnd-refStartSW2ryuu), 28);
19,050 if (FoundAtPosition2!=NULL) {
19,051 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(28-1);
19,052 } else FoundAtPosition = NULL;
19,053 // Fallback to memmem(): ]
19,054 } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[11];
19,055 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
%p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
19,056 } else FoundAtPosition = NULL;
19,057 #endif
19,058 // 2020-Jan-01 ]
19,059
19,060 if (FoundAtPosition!=NULL) {
19,061 *retMatch=28;
19,062 // The first four bits should be:
19,063 // 1000b = 8
19,064 *retIndex=((refEnd-FoundAtPosition)<<3)&0xFFF8)!0x07; // 111, 32:(2+1)=10.6
19,065 *ShortMediumLongOFFSET=2+1 +3333;
19,066 return;
19,067 }
19,068 }
19,069
19,070 } // B-TREE heuristic
19,071
19,072 #if defined(BtreeHEURISTIC)
19,073 if ( LastSeenOffset_PseudoPointer[8] != 0 )
19,074 #endif
19,075 { // Search into ... If 36 is not found then skip all 36[+] - see the roster above.
19,076
19,077 //#15 36:(2+1)+1= 9 (8KB)
19,078 if (refStartSW2ryuu >= refStart) {
19,079
19,080 #ifndef BtreeHEURISTIC

```

```

19,081 // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW2ryuu, encStart, (uint32_t)(refEnd-refStartSW2ryuu), 18*2);
19,082 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW2ryuu - src) - (uint32_t)(refEnd-refStartSW2ryuu - 1), srcR + (srcSize - 1) - (encStart - src)
-(18*2-1), (uint32_t)(refEnd-refStartSW2ryuu), 18*2);
19,083 if (FoundAtPosition2!=NULL) {
19,084 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (18*2-1);
19,085 } else FoundAtPosition = NULL;
19,086 #else
19,087 if ( refStartSW2ryuu <= (char *)LastSeenOffset_PseudoPointer[8] ) {
19,088 if ( refEnd - (char *)LastSeenOffset_PseudoPointer[8] < (36) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
first position of the LookAhead!
19,089 // Fallback to memmem(): [
19,090 FoundAtPosition = Railgun_BawBaw_reverse (refStartSW2ryuu, encStart, (uint32_t)(refEnd-refStartSW2ryuu), 18*2);
19,091 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW2ryuu - src) - (uint32_t)(refEnd-refStartSW2ryuu - 1), srcR + (srcSize - 1) - (encStart - src)
-(18*2-1), (uint32_t)(refEnd-refStartSW2ryuu), 18*2);
19,092 if (FoundAtPosition2!=NULL) {
19,093 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (18*2-1);
19,094 } else FoundAtPosition = NULL;
19,095 // Fallback to memmem(): ]
19,096 } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[8];
19,097 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
%p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
19,098 } else FoundAtPosition = NULL;
19,099 #endif
19,100
19,101 if (FoundAtPosition!=NULL) {
19,102 *retMatch=18*2;
19,103 // The first four bits should be:
19,104 // 1000b = 8
19,105 *retIndex=((refEnd-FoundAtPosition)<<3)&0xFFF8)!0x03; // 110, 18:(2+1)=6
19,106 *ShortMediumLongOFFSET=2+1+3333;
19,107 *HowManyDittoTagsToEmit = 1;
19,108 return;
19,109 }
19,110 }
19,111
19,112 } // B-TREE heuristic
19,113
19,114 #if defined(BtreeHEURISTIC)
19,115 if ( LastSeenOffset_PseudoPointer[13] != 0 )
19,116 #endif
19,117 { // Search into ... If 48 is not found then skip all 48[+] - see the roster above.
19,118
19,119 //#16 56:(5+1)+1= 8 (128GB)
19,120 if (refStartSW5ryuu >= refStart) {
19,121
19,122 #ifndef BtreeHEURISTIC
19,123 FoundAtPosition = Railgun_BawBaw_reverse (refStartSW5ryuu, encStart, (uint64_t)(refEnd-refStartSW5ryuu), 28*2);
19,124 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW5ryuu - src) - (uint64_t)(refEnd-refStartSW5ryuu - 1), srcR + (srcSize - 1) - (encStart - src)
-(28*2-1), (uint64_t)(refEnd-refStartSW5ryuu), 28*2);
19,125 if (FoundAtPosition2!=NULL) {
19,126 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (28*2-1);
19,127 } else FoundAtPosition = NULL;
19,128 #else
19,129 if ( refStartSW5ryuu <= (char *)LastSeenOffset_PseudoPointer[13] ) {
19,130 if ( refEnd - (char *)LastSeenOffset_PseudoPointer[13] < (56) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
first position of the LookAhead!
19,131 // Fallback to memmem(): [
19,132 FoundAtPosition = Railgun_BawBaw_reverse (refStartSW5ryuu, encStart, (uint64_t)(refEnd-refStartSW5ryuu), 28*2);

```

```

19,133 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW5ryuu - src) - (uint64_t)(refEnd-refStartSW5ryuu -1), srcR + (srcSize - 1) - (encStart - src)
-(28*2-1), (uint64_t)(refEnd-refStartSW5ryuu), 28*2);
19,134     if (FoundAtPosition2!=NULL) {
19,135 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(28*2-1);
19,136     } else FoundAtPosition = NULL;
19,137         // Fallback to memmem(): ]
19,138     } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[13];
19,139 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
%p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
19,140     } else FoundAtPosition = NULL;
19,141 #endif
19,142
19,143     if (FoundAtPosition!=NULL) {
19,144         *retMatch=28*2;
19,145         // The first 8 bits should be 0x93, in order to retain 4bytes-longness of 'retIndex' we play dirty - not tagging the 5bytes chunk with 0x93
but as "usual" - the address subchunk:
19,146         *retIndex=(((refEnd-FoundAtPosition)<<3)&0xFFFFFFFF8)!0x04; // 001, 32:(5+1)=5.3
19,147         *ShortMediumLongOFFSET=5+1+3333;
19,148         *HowManyDittoTagsToEmit = 1;
19,149         return;
19,150     }
19,151 }
19,152
19,153     } // B-TREE heuristic
19,154
19,155 #if defined(BtreeHEURISTIC)
19,156     if ( LastSeenOffset_PseudoPointer[10] != 0 ) // 2020-Jan-01
19,157 #endif
19,158     { // Search into ... If 18 is not found then skip all 18[+] - see the roster above.
19,159
19,160 //#17 24:3=          8      (1MB)
19,161 /*
19,162 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) [
19,163 if (refStartHOT >= refStart)
19,164 if (refStartHOT < refEnd) {
19,165 FoundAtPosition = Railgun_Trollldom_64(refStartHOT, encStart, (uint32_t)(refEnd-refStartHOT), 24);
19,166     if (FoundAtPosition!=NULL) {
19,167         *retMatch=24;
19,168         // The first four bits should be:
19,169                                     // 0100b = 4
19,170         *retIndex=(((refEnd-FoundAtPosition)<<4)&0xFFFFF0)!0x0004; // xx ... x[OOLL]
19,171         *ShortMediumLongOFFSET=3;
19,172         return;
19,173     }
19,174 }
19,175 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) ]
19,176
19,177 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) [
19,178 if (refStartCOLDERbig >= refStart)
19,179 if (refStartCOLDERbig < refEnd) {
19,180 FoundAtPosition = Railgun_Trollldom_64(refStartCOLDERbig, encStart, (uint32_t)(refEnd-refStartCOLDERbig), 24);
19,181     if (FoundAtPosition!=NULL) {
19,182         *retMatch=24;
19,183         // The first four bits should be:
19,184                                     // 0100b = 4
19,185         *retIndex=(((refEnd-FoundAtPosition)<<4)&0xFFFFF0)!0x0004; // xx ... x[OOLL]
19,186         *ShortMediumLongOFFSET=3;
19,187         return;

```

```

19,188     }
19,189 }
19,190 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) ]
19,191 /*
19,192 if (refStartSW3 >= refStart) {
19,193 // 2020-Jan-01 [
19,194 #ifndef BtreeHEURISTIC
19,195 //     FoundAtPosition = Railgun_BawBaw_reverse (refStartSW3, encStart, (uint32_t)(refEnd-refStartSW3), 24);
19,196 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW3 - src) - (uint32_t)(refEnd-refStartSW3 - 1), srcR + (srcSize - 1) - (encStart - src) - (24-1), (uint32_t)(refEnd-refStartSW3), 24);
19,197     if (FoundAtPosition2!=NULL) {
19,198 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (24-1);
19,199     } else FoundAtPosition = NULL;
19,200 #else
19,201     if ( refStartSW3 <= (char *)LastSeenOffset_PseudoPointer[10] ) {
19,202         if ( refEnd - (char *)LastSeenOffset_PseudoPointer[10] < (24) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the first position of the LookAhead!
19,203             // Fallback to memmem(): [
19,204 //         FoundAtPosition = Railgun_BawBaw_reverse (refStartSW3, encStart, (uint32_t)(refEnd-refStartSW3), 24);
19,205 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW3 - src) - (uint32_t)(refEnd-refStartSW3 - 1), srcR + (srcSize - 1) - (encStart - src) - (24-1), (uint32_t)(refEnd-refStartSW3), 24);
19,206         if (FoundAtPosition2!=NULL) {
19,207 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (24-1);
19,208         } else FoundAtPosition = NULL;
19,209         // Fallback to memmem(): ]
19,210         } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[10];
19,211 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p: %p,%p\n", refStartSW3,refEnd, FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0]); //debugprint
19,212     } else FoundAtPosition = NULL;
19,213 #endif
19,214 // 2020-Jan-01 ]
19,215
19,216     if (FoundAtPosition!=NULL) {
19,217         *retMatch=24;
19,218         // The first four bits should be:
19,219
19,220         *retIndex=(((refEnd-FoundAtPosition)<<4)&0xFFFFF0)!0x0004; // xx ... x[00LL] // 0100b = 4
19,221         *ShortMediumLongOFFSET=3;
19,222         return;
19,223     }
19,224 }
19,225
19,226     } // B-TREE heuristic
19,227
19,228 // #18 16:2=      8      (4KB)
19,229 /*
19,230 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) [
19,231 if (refStartHOTTER >= refStart)
19,232 if (refStartHOTTER < refEnd) {
19,233 FoundAtPosition = Railgun_Trollldom_64(refStartHOTTER, encStart, (uint32_t)(refEnd-refStartHOTTER), 16);
19,234     if (FoundAtPosition!=NULL) {
19,235         *retMatch=16;
19,236         // The first four bits should be:
19,237
19,238         *retIndex=(((refEnd-FoundAtPosition)<<4)&0xFFF0)!0x0002; // xx ... x[00LL] // 0010b = 2
19,239         *ShortMediumLongOFFSET=2;
19,240         return;
19,241     }

```

```

19,242 }
19,243 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) ]
19,244 */
19,245 #if defined(BtreeHEURISTIC)
19,246     if ( LastSeenOffset_PseudoPointer[6] != 0 )
19,247 #endif
19,248     { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
19,249
19,250     if (refStartSW2 >= refStart) {
19,251
19,252     // 2020-Jan-01 [
19,253     #ifndef BtreeHEURISTIC
19,254     //     FoundAtPosition = Railgun_BawBaw_reverse (refStartSW2, encStart, (uint32_t)(refEnd-refStartSW2), 16);
19,255     FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW2 - src) - (uint32_t)(refEnd-refStartSW2 - 1), srcR + (srcSize - 1) - (encStart - src) - (16-1), (uint32_t)(refEnd-refStartSW2), 16);
19,256     if (FoundAtPosition2!=NULL) {
19,257     FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (16-1);
19,258     } else FoundAtPosition = NULL;
19,259     #else
19,260     if ( refStartSW2 <= (char *)LastSeenOffset_PseudoPointer[6] ) {
19,261     if ( refEnd - (char *)LastSeenOffset_PseudoPointer[6] < (16) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the first position of the LookAhead!
19,262     // Fallback to memmem(): [
19,263     FoundAtPosition = Railgun_BawBaw_reverse (refStartSW2, encStart, (uint32_t)(refEnd-refStartSW2), 16);
19,264     FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW2 - src) - (uint32_t)(refEnd-refStartSW2 - 1), srcR + (srcSize - 1) - (encStart - src) - (16-1), (uint32_t)(refEnd-refStartSW2), 16);
19,265     if (FoundAtPosition2!=NULL) {
19,266     FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (16-1);
19,267     } else FoundAtPosition = NULL;
19,268     // Fallback to memmem(): ]
19,269     } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[6];
19,270 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p: %p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
19,271     } else FoundAtPosition = NULL;
19,272 #endif
19,273 // 2020-Jan-01 ]
19,274
19,275     if (FoundAtPosition!=NULL) {
19,276     *retMatch=16;
19,277     // The first four bits should be:
19,278
19,279     *retIndex=(((refEnd-FoundAtPosition)<<4)&0xFFF0)!0x0002; // xx ... x[OOLL] // 0010b = 2
19,280     *ShortMediumLongOFFSET=2;
19,281     return;
19,282     }
19,283 }
19,284 } // B-TREE heuristic
19,285
19,286 // #19 8:1= 8 (16B)
19,287 /*
19,288 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) [
19,289 if (refStartHOTEST >= refStart)
19,290 if (refStartHOTEST < refEnd) {
19,291 FoundAtPosition = Railgun_Trollldom_64(refStartHOTEST, encStart, (uint32_t)(refEnd-refStartHOTEST), 8);
19,292 if (FoundAtPosition!=NULL) {
19,293 *retMatch=8;
19,294 // The first four bits should be:
19,295
19,296 // 1011b = B

```



```

19,296             *retIndex=(((refEnd-FoundAtPosition)<<4)&0xFFF0)!0x000B; // xx ... x[OOLL]
19,297             *ShortMediumLongOFFSET=1;
19,298             return;
19,299         }
19,300     }
19,301     // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) ]
19,302     */
19,303     #if defined(BtreeHEURISTIC)
19,304         if ( LastSeenOffset_PseudoPointer[2] != 0 )
19,305     #endif
19,306         { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
19,307
19,308         if (refStartSW1 >= refStart) {
19,309
19,310         #ifndef BtreeHEURISTIC
19,311             // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW1, encStart, (uint32_t)(refEnd-refStartSW1), 8);
19,312             FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW1 - src) -(uint32_t)(refEnd-refStartSW1 -1), srcR + (srcSize - 1) - (encStart - src) -(8-1),
19,313             (uint32_t)(refEnd-refStartSW1), 8);
19,314             if (FoundAtPosition2!=NULL) {
19,315             FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(8-1);
19,316             } else FoundAtPosition = NULL;
19,317         #else
19,318             if ( refStartSW1 <= (char *)LastSeenOffset_PseudoPointer[2] ) {
19,319                 if ( refEnd - (char *)LastSeenOffset_PseudoPointer[2] < (8) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
19,320                 first position of the LookAhead!
19,321                 // Fallback to memmem(): [
19,322                 FoundAtPosition = Railgun_BawBaw_reverse (refStartSW1, encStart, (uint32_t)(refEnd-refStartSW1), 8);
19,323                 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW1 - src) -(uint32_t)(refEnd-refStartSW1 -1), srcR + (srcSize - 1) - (encStart - src) -(8-1),
19,324                 (uint32_t)(refEnd-refStartSW1), 8);
19,325                 if (FoundAtPosition2!=NULL) {
19,326                 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(8-1);
19,327                 } else FoundAtPosition = NULL;
19,328                 // Fallback to memmem(): ]
19,329                 } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[2];
19,330             //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
19,331             %p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
19,332             } else FoundAtPosition = NULL;
19,333         #endif
19,334
19,335         if (FoundAtPosition!=NULL) {
19,336             *retMatch=8;
19,337             // The first four bits should be:
19,338
19,339             // 1011b = B
19,340             *retIndex=(((refEnd-FoundAtPosition)<<4)&0xFFF0)!0x000B; // xx ... x[OOLL]
19,341             *ShortMediumLongOFFSET=1;
19,342             return;
19,343         }
19,344     } // B-TREE heuristic
19,345
19,346     //#20 22:3= 7.3 (128KB)
19,347     #if defined(BtreeHEURISTIC)
19,348         if ( LastSeenOffset_PseudoPointer[7] != 0 )
19,349             //int MatchLens[MatchLensNUMdummy]={4,6,8,10,12,14,16,18,36,64,24,28,48,56,128,256};
19,350             /// 0,1,2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f
19,351     #endif
19,352         { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
19,353

```

```

19,350 if (refStartSW128kb >= refStart) {
19,351 FoundAtPosition = Railgun_BawBaw_reverse (refStartSW128kb, encStart, (uint32_t)(refEnd-refStartSW128kb), 22); // Uncommented , 2021-Mar-11, below 4 lines need
reversed file block
19,352 // FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW128kb - src) -(uint32_t)(refEnd-refStartSW128kb -1), srcR + (srcSize - 1) -
(encStart - src) -(22-1), (uint32_t)(refEnd-refStartSW128kb), 22);
19,353 // if (FoundAtPosition2!=NULL) {
19,354 // FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(22-1);
19,355 // } else FoundAtPosition = NULL;
19,356
19,357 if (FoundAtPosition!=NULL) {
19,358 *retMatch=22;
19,359 // The first four bits should be:
19,360 *retIndex=((refEnd-FoundAtPosition)<<7)&0xFFFF80)!0x2C;
19,361 *ShortMediumLongOFFSET=3;
19,362 return;
19,363 }
19,364 }
19,365 } // B-TREE heuristic
19,366
19,367 // #21 36:(3+1)+1= 7.2 (2MB)
19,368 #if defined(BtreeHEURISTIC)
19,369 if ( LastSeenOffset_PseudoPointer[8] != 0 )
19,370 #endif
19,371 { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
19,372
19,373 if (refStartSW3ryuu >= refStart) {
19,374
19,375 #ifndef BtreeHEURISTIC
19,376 // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW3ryuu, encStart, (uint32_t)(refEnd-refStartSW3ryuu), 18*2);
19,377 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW3ryuu - src) -(uint32_t)(refEnd-refStartSW3ryuu -1), srcR + (srcSize - 1) - (encStart - src)
-(18*2-1), (uint32_t)(refEnd-refStartSW3ryuu), 18*2);
19,378 if (FoundAtPosition2!=NULL) {
19,379 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(18*2-1);
19,380 } else FoundAtPosition = NULL;
19,381 #else
19,382 if ( refStartSW3ryuu <= (char *)LastSeenOffset_PseudoPointer[8] ) {
19,383 if ( refEnd - (char *)LastSeenOffset_PseudoPointer[8] < (36) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
first position of the LookAhead!
19,384 // Fallback to memmem(): [
19,385 // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW3ryuu, encStart, (uint32_t)(refEnd-refStartSW3ryuu), 18*2);
19,386 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW3ryuu - src) -(uint32_t)(refEnd-refStartSW3ryuu -1), srcR + (srcSize - 1) - (encStart - src)
-(18*2-1), (uint32_t)(refEnd-refStartSW3ryuu), 18*2);
19,387 if (FoundAtPosition2!=NULL) {
19,388 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(18*2-1);
19,389 } else FoundAtPosition = NULL;
19,390 // Fallback to memmem(): ]
19,391 } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[8];
19,392 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
%p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
19,393 } else FoundAtPosition = NULL;
19,394 #endif
19,395
19,396 if (FoundAtPosition!=NULL) {
19,397 *retMatch=18*2;
19,398 // The first four bits should be:
19,399 // 0101b = 5
19,400 *retIndex=((refEnd-FoundAtPosition)<<3)&0xFFFFF8)!0x02; // 010, 18:(3+1)=4.5
19,401 *ShortMediumLongOFFSET=3+1+3333;

```

```

19,402             *HowManyDittoTagsToEmit = 1;
19,403             return;
19,404         }
19,405     }
19,406     } // B-TREE heuristic
19,407
19,408     // #22 28:(3+1)= 7 (2MB)
19,409     #if defined(BtreeHEURISTIC)
19,410         if ( LastSeenOffset_PseudoPointer[11] != 0 ) // 2020-Jan-01
19,411     #endif
19,412         { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
19,413
19,414         if (refStartSW3ryuu >= refStart) {
19,415             // 2020-Jan-01 [
19,416             #ifndef BtreeHEURISTIC
19,417             // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW3ryuu, encStart, (uint32_t)(refEnd-refStartSW3ryuu), 28);
19,418             FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW3ryuu - src) - (uint32_t)(refEnd-refStartSW3ryuu - 1), srcR + (srcSize - 1) - (encStart - src)
19,419             -(28-1), (uint32_t)(refEnd-refStartSW3ryuu), 28);
19,420             if (FoundAtPosition2!=NULL) {
19,421                 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(28-1);
19,422             } else FoundAtPosition = NULL;
19,423         #else
19,424             if ( refStartSW3ryuu <= (char *)LastSeenOffset_PseudoPointer[11] ) {
19,425                 if ( refEnd - (char *)LastSeenOffset_PseudoPointer[11] < (28) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
19,426                 first position of the LookAhead!
19,427                 // Fallback to memmem(): [
19,428                 FoundAtPosition = Railgun_BawBaw_reverse (refStartSW3ryuu, encStart, (uint32_t)(refEnd-refStartSW3ryuu), 28);
19,429                 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW3ryuu - src) - (uint32_t)(refEnd-refStartSW3ryuu - 1), srcR + (srcSize - 1) - (encStart - src)
19,430                 -(28-1), (uint32_t)(refEnd-refStartSW3ryuu), 28);
19,431                 if (FoundAtPosition2!=NULL) {
19,432                     FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(28-1);
19,433                 } else FoundAtPosition = NULL;
19,434                 // Fallback to memmem(): [
19,435                 } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[11];
19,436             //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
19,437             %p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
19,438             } else FoundAtPosition = NULL;
19,439         #endif
19,440         // 2020-Jan-01 ]
19,441
19,442         if (FoundAtPosition!=NULL) {
19,443             *retMatch=28;
19,444             // The first four bits should be:
19,445             // 0101b = 5
19,446             *retIndex=(((refEnd-FoundAtPosition)<<3)&0xFFFFF8)!0x06; // 011, 32:(3+1)=8
19,447             *ShortMediumLongOFFSET=3+1 +3333;
19,448             return;
19,449         }
19,450     } // B-TREE heuristic
19,451
19,452     // #23 14:2= 7 (512B)
19,453     #if defined(BtreeHEURISTIC)
19,454         if ( LastSeenOffset_PseudoPointer[5] != 0 )
19,455     #endif
19,456         { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
19,457
19,458         if (refStartSW512 >= refStart) {

```

```
19,456
19,457 // 2020-Jan-01 [
19,458 #ifndef BtreeHEURISTIC
19,459 //      FoundAtPosition = Railgun_BawBaw_reverse (refStartSW512, encStart, (uint32_t)(refEnd-refStartSW512), 14);
19,460 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW512 - src) - (uint32_t)(refEnd-refStartSW512 - 1), srcR + (srcSize - 1) - (encStart - src) -
(14-1), (uint32_t)(refEnd-refStartSW512), 14);
19,461     if (FoundAtPosition2!=NULL) {
19,462 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(14-1);
19,463     } else FoundAtPosition = NULL;
19,464 #else
19,465     if ( refStartSW512 <= (char *)LastSeenOffset_PseudoPointer[5] ) {
19,466         if ( refEnd - (char *)LastSeenOffset_PseudoPointer[5] < (14) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
first position of the LookAhead!
19,467             // Fallback to memmem(): [
19,468 //      FoundAtPosition = Railgun_BawBaw_reverse (refStartSW512, encStart, (uint32_t)(refEnd-refStartSW512), 14);
19,469 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW512 - src) - (uint32_t)(refEnd-refStartSW512 - 1), srcR + (srcSize - 1) - (encStart - src) -
(14-1), (uint32_t)(refEnd-refStartSW512), 14);
19,470     if (FoundAtPosition2!=NULL) {
19,471 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(14-1);
19,472     } else FoundAtPosition = NULL;
19,473         // Fallback to memmem(): ]
19,474     } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[5];
19,475 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
%p,%p\n", refStartSW3,refEnd, FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0]); //debugprint
19,476     } else FoundAtPosition = NULL;
19,477 #endif
19,478 // 2020-Jan-01 ]
19,479
19,480     if (FoundAtPosition!=NULL) {
19,481         *retMatch=14;
19,482         // The first four bits should be:
19,483         *retIndex=(((refEnd-FoundAtPosition)<<7)&0xFF80)!0x5C; // (4+1)C
19,484         *ShortMediumLongOFFSET=2;
19,485         return;
19,486     }
19,487 }
19,488 } // B-TREE heuristic
19,489
19,490 #if defined(BtreeHEURISTIC)
19,491     if ( LastSeenOffset_PseudoPointer[8] != 0 )
19,492 #endif
19,493 { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
19,494
19,495 //#24 36:(4+1)+1=      6   (512MB)
19,496 if (refStartSW5 >= refStart) {
19,497
19,498 #ifndef BtreeHEURISTIC
19,499 //      FoundAtPosition = Railgun_BawBaw_reverse (refStartSW5, encStart, (uint32_t)(refEnd-refStartSW5), 18*2);
19,500 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW5 - src) - (uint32_t)(refEnd-refStartSW5 - 1), srcR + (srcSize - 1) - (encStart - src) -(18*2-
1), (uint32_t)(refEnd-refStartSW5), 18*2);
19,501     if (FoundAtPosition2!=NULL) {
19,502 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(18*2-1);
19,503     } else FoundAtPosition = NULL;
19,504 #else
19,505     if ( refStartSW5 <= (char *)LastSeenOffset_PseudoPointer[8] ) {
19,506         if ( refEnd - (char *)LastSeenOffset_PseudoPointer[8] < (36) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
first position of the LookAhead!
19,507             // Fallback to memmem(): [
```

```

19,508 //      FoundAtPosition = Railgun_BawBaw_reverse (refStartSW5, encStart, (uint32_t)(refEnd-refStartSW5), 18*2);
19,509 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW5 - src) -(uint32_t)(refEnd-refStartSW5 -1), srcR + (srcSize - 1) - (encStart - src) -(18*2-
1), (uint32_t)(refEnd-refStartSW5), 18*2);
19,510      if (FoundAtPosition2!=NULL) {
19,511 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(18*2-1);
19,512      } else FoundAtPosition = NULL;
19,513      // Fallback to memmem(): ]
19,514      } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[8];
19,515 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
%p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
19,516      } else FoundAtPosition = NULL;
19,517 #endif
19,518
19,519      if (FoundAtPosition!=NULL) {
19,520          *retMatch=18*2;
19,521          // The first 8 bits should be 0x93, in order to retain 4bytes-longness of 'retIndex' we play dirty - not tagging the 5bytes chunk with 0x93
but as "usual" - the address subchunk:
19,522          *retIndex=((refEnd-FoundAtPosition)<<3)&0xFFFFFFFF8)!0x01; // 100, 18:(4+1)=3.6
19,523          *ShortMediumLongOFFSET=4+1+3333;
19,524          *HowManyDittoTagsToEmit = 1;
19,525          return;
19,526      }
19,527 }
19,528
19,529      } // B-TREE heuristic
19,530
19,531 #if defined(BtreeHEURISTIC)
19,532      if ( LastSeenOffset_PseudoPointer[10] != 0 ) // 2020-Jan-01
19,533 #endif
19,534      { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
19,535
19,536 //#25 24:4=          6 (256MB)
19,537 if (refStartSW4 >= refStart) {
19,538
19,539 // 2020-Jan-01 [
19,540 #ifndef BtreeHEURISTIC
19,541 //      FoundAtPosition = Railgun_BawBaw_reverse (refStartSW4, encStart, (uint32_t)(refEnd-refStartSW4), 48-24);
19,542 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW4 - src) -(uint32_t)(refEnd-refStartSW4 -1), srcR + (srcSize - 1) - (encStart - src) -(48-
24-1), (uint32_t)(refEnd-refStartSW4), 48-24);
19,543      if (FoundAtPosition2!=NULL) {
19,544 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(48-24-1);
19,545      } else FoundAtPosition = NULL;
19,546 #else
19,547      if ( refStartSW4 <= (char *)LastSeenOffset_PseudoPointer[10] ) {
19,548          if ( refEnd - (char *)LastSeenOffset_PseudoPointer[10] < (24) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
first position of the LookAhead!
19,549              // Fallback to memmem(): [
19,550 //      FoundAtPosition = Railgun_BawBaw_reverse (refStartSW4, encStart, (uint32_t)(refEnd-refStartSW4), 48-24);
19,551 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW4 - src) -(uint32_t)(refEnd-refStartSW4 -1), srcR + (srcSize - 1) - (encStart - src) -(48-
24-1), (uint32_t)(refEnd-refStartSW4), 48-24);
19,552      if (FoundAtPosition2!=NULL) {
19,553 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(48-24-1);
19,554      } else FoundAtPosition = NULL;
19,555      // Fallback to memmem(): ]
19,556      } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[10];
19,557 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
%p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
19,558      } else FoundAtPosition = NULL;

```

```

19,559 #endif
19,560 // 2020-Jan-01 ]
19,561
19,562     if (FoundAtPosition!=NULL) {
19,563         *retMatch=48-24;
19,564         // The first four bits should be:
19,565                                     // 0000b = 0x0; 6<<2
19,566         *retIndex=((refEnd-FoundAtPosition)<<4)&0xFFFFF0)!0x00; // xx ... x[LL00]
19,567         *ShortMediumLongOFFSET=4;
19,568         return;
19,569     }
19,570 }
19,571 } // B-TREE heuristic // 2020-Jan-04
19,572
19,573 #if defined(BtreeHEURISTIC)
19,574     if ( LastSeenOffset_PseudoPointer[7] != 0 ) // 2020-Jan-01
19,575 #endif
19,576     { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
19,577     // #26 18:(2+1)= 6 (8KB)
19,578     if (refStartSW2ryuu >= refStart) {
19,579
19,580     // 2020-Jan-01 [
19,581     #ifndef BtreeHEURISTIC
19,582     // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW2ryuu, encStart, (uint32_t)(refEnd-refStartSW2ryuu), 18);
19,583     FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW2ryuu - src) - (uint32_t)(refEnd-refStartSW2ryuu -1), srcR + (srcSize - 1) - (encStart - src)
19,584     -(18-1), (uint32_t)(refEnd-refStartSW2ryuu), 18);
19,585     if (FoundAtPosition2!=NULL) {
19,586     FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(18-1);
19,587     //printf("\nFoundAtPosition 18:(2+1): %lu\n", FoundAtPosition); //debugprint
19,588     } else FoundAtPosition = NULL;
19,589     #else
19,590     if ( refStartSW2ryuu <= (char *)LastSeenOffset_PseudoPointer[7] ) {
19,591     if ( refEnd - (char *)LastSeenOffset_PseudoPointer[7] < (18) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
19,592     first position of the LookAhead!
19,593     // Fallback to memmem(): [
19,594     FoundAtPosition = Railgun_BawBaw_reverse (refStartSW2ryuu, encStart, (uint32_t)(refEnd-refStartSW2ryuu), 18);
19,595     FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW2ryuu - src) - (uint32_t)(refEnd-refStartSW2ryuu -1), srcR + (srcSize - 1) - (encStart - src)
19,596     -(18-1), (uint32_t)(refEnd-refStartSW2ryuu), 18);
19,597     if (FoundAtPosition2!=NULL) {
19,598     FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(18-1);
19,599     //printf("\nFoundAtPosition 18:(2+1): %lu\n", FoundAtPosition); //debugprint
19,600     } else FoundAtPosition = NULL;
19,601     // Fallback to memmem(): ]
19,602     } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[7];
19,603     //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
19,604     %p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
19,605     } else FoundAtPosition = NULL;
19,606 #endif
19,607 // 2020-Jan-01 ]
19,608
19,609     if (FoundAtPosition!=NULL) {
19,610         *retMatch=18;
19,611         // The first four bits should be:
19,612                                     // 1000b = 8
19,613         *retIndex=((refEnd-FoundAtPosition)<<3)&0xFFF8)!0x03; // 110, 18:(2+1)=6
19,614         *ShortMediumLongOFFSET=2+1+3333;
19,615         return;
19,616     }

```

```

19,613 }
19,614     } // B-TREE heuristic
19,615
19,616 // #27 12:2=          6      (4KB)
19,617 /*
19,618 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) [
19,619 if (refStartHOTTER >= refStart)
19,620 if (refStartHOTTER < refEnd) {
19,621 FoundAtPosition = Railgun_Trollldom_64(refStartHOTTER, encStart, (uint32_t)(refEnd-refStartHOTTER), 12);
19,622     if (FoundAtPosition!=NULL) {
19,623         *retMatch=12;
19,624         // The first four bits should be:
19,625                                     // 0110b = 6
19,626         *retIndex=((refEnd-FoundAtPosition)<<4)&0xFFF0!0x0006; // xx ... x[OOLL]
19,627         *ShortMediumLongOFFSET=2;
19,628         return;
19,629     }
19,630 }
19,631 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) ]
19,632 */
19,633 #if defined(BtreeHEURISTIC)
19,634     if ( LastSeenOffset_PseudoPointer[4] != 0 )
19,635 #endif
19,636     { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
19,637
19,638 if (refStartSW2 >= refStart) {
19,639 // 2020-Jan-01 [
19,640 #ifndef BtreeHEURISTIC
19,641 // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW2, encStart, (uint32_t)(refEnd-refStartSW2), 12);
19,642 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW2 - srcR) -(uint32_t)(refEnd-refStartSW2 -1), srcR + (srcSize - 1) - (encStart - srcR) -(12-1), (uint32_t)(refEnd-refStartSW2), 12);
19,643     if (FoundAtPosition2!=NULL) {
19,644 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(12-1);
19,645     } else FoundAtPosition = NULL;
19,646 #else
19,647     if ( refStartSW2 <= (char *)LastSeenOffset_PseudoPointer[4] ) {
19,648         if ( refEnd - (char *)LastSeenOffset_PseudoPointer[4] < (12) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the first position of the LookAhead!
19,649             // Fallback to memmem(): [
19,650 // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW2, encStart, (uint32_t)(refEnd-refStartSW2), 12);
19,651 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW2 - srcR) -(uint32_t)(refEnd-refStartSW2 -1), srcR + (srcSize - 1) - (encStart - srcR) -(12-1), (uint32_t)(refEnd-refStartSW2), 12);
19,652     if (FoundAtPosition2!=NULL) {
19,653 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(12-1);
19,654     } else FoundAtPosition = NULL;
19,655         // Fallback to memmem(): ]
19,656     } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[4];
19,657 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p: %p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
19,658     } else FoundAtPosition = NULL;
19,659 #endif
19,660 // 2020-Jan-01 ]
19,661
19,662     if (FoundAtPosition!=NULL) {
19,663         *retMatch=12;
19,664         // The first four bits should be:
19,665                                     // 0110b = 6
19,666         *retIndex=((refEnd-FoundAtPosition)<<4)&0xFFF0!0x0006; // xx ... x[OOLL]

```

```

19,667             *ShortMediumLongOFFSET=2;
19,668             return;
19,669         }
19,670     }
19,671     } // B-TREE heuristic
19,672
19,673 // #28 28:(4+1)= 5.6 (512MB)
19,674 #if defined(BtreeHEURISTIC)
19,675     if ( LastSeenOffset_PseudoPointer[11] != 0 ) // 2020-Jan-01
19,676 #endif
19,677     { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
19,678
19,679     if (refStartSW5 >= refStart) {
19,680 // 2020-Jan-01 [
19,681 #ifndef BtreeHEURISTIC
19,682 // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW5, encStart, (uint32_t)(refEnd-refStartSW5), 28);
19,683 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW5 - src) - (uint32_t)(refEnd-refStartSW5 - 1), srcR + (srcSize - 1) - (encStart - src) - (28-1), (uint32_t)(refEnd-refStartSW5), 28);
19,684     if (FoundAtPosition2!=NULL) {
19,685 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (28-1);
19,686     } else FoundAtPosition = NULL;
19,687 #else
19,688     if ( refStartSW5 <= (char *)LastSeenOffset_PseudoPointer[11] ) {
19,689         if ( refEnd - (char *)LastSeenOffset_PseudoPointer[11] < (28) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the first position of the LookAhead!
19,690             // Fallback to memmem(): [
19,691 // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW5, encStart, (uint32_t)(refEnd-refStartSW5), 28);
19,692 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW5 - src) - (uint32_t)(refEnd-refStartSW5 - 1), srcR + (srcSize - 1) - (encStart - src) - (28-1), (uint32_t)(refEnd-refStartSW5), 28);
19,693     if (FoundAtPosition2!=NULL) {
19,694 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (28-1);
19,695     } else FoundAtPosition = NULL;
19,696         // Fallback to memmem(): [
19,697     } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[11];
19,698 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p: %p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
19,699     } else FoundAtPosition = NULL;
19,700 #endif
19,701 // 2020-Jan-01 ]
19,702
19,703     if (FoundAtPosition!=NULL) {
19,704         *retMatch=28;
19,705         // The first 8 bits should be 0x93, in order to retain 4bytes-longness of 'retIndex' we play dirty - not tagging the 5bytes chunk with 0x93
but as "usual" - the address subchunk:
19,706         *retIndex=((refEnd-FoundAtPosition)<<3)&0xFFFFFFFF8)!0x05; // 101, 32:(4+1)=6.4
19,707         *ShortMediumLongOFFSET=4+1 +3333;
19,708         return;
19,709     }
19,710 }
19,711 } // B-TREE heuristic
19,712
19,713 // #29 16:3= 5.3 (1MB)
19,714 /*
19,715 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) [
19,716 if (refStartHOT >= refStart)
19,717 if (refStartHOT < refEnd) {
19,718 FoundAtPosition = Railgun_Trollldom_64(refStartHOT, encStart, (uint32_t)(refEnd-refStartHOT), 16);
19,719     if (FoundAtPosition!=NULL) {

```



```

19,720         *retMatch=16;
19,721         // The first four bits should be:
19,722                                     // 0001b = 1
19,723         *retIndex=((refEnd-FoundAtPosition)<<4)&0xFFFFF0)!0x0001; // xx ... x[OOLL]
19,724         *ShortMediumLongOFFSET=3;
19,725         return;
19,726     }
19,727 }
19,728 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) ]
19,729
19,730 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) [
19,731 if (refStartCOLDERbig >= refStart)
19,732 if (refStartCOLDERbig < refEnd) {
19,733 FoundAtPosition = Railgun_Trollldom_64(refStartCOLDERbig, encStart, (uint32_t)(refEnd-refStartCOLDERbig), 16);
19,734     if (FoundAtPosition!=NULL) {
19,735         *retMatch=16;
19,736         // The first four bits should be:
19,737                                     // 0001b = 1
19,738         *retIndex=((refEnd-FoundAtPosition)<<4)&0xFFFFF0)!0x0001; // xx ... x[OOLL]
19,739         *ShortMediumLongOFFSET=3;
19,740         return;
19,741     }
19,742 }
19,743 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) ]
19,744 */
19,745 #if defined(BtreeHEURISTIC)
19,746     if ( LastSeenOffset_PseudoPointer[6] != 0 )
19,747 #endif
19,748     { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
19,749
19,750     if (refStartSW3 >= refStart) {
19,751
19,752 #ifndef BtreeHEURISTIC
19,753 //         FoundAtPosition = Railgun_BawBaw_reverse (refStartSW3, encStart, (uint32_t)(refEnd-refStartSW3), 16);
19,754 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW3 - src) -(uint32_t)(refEnd-refStartSW3 -1), srcR + (srcSize - 1) - (encStart - src) -(16-1), (uint32_t)(refEnd-refStartSW3), 16);
19,755         if (FoundAtPosition2!=NULL) {
19,756 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(16-1);
19,757         } else FoundAtPosition = NULL;
19,758 #else
19,759         if ( refStartSW3 <= (char *)LastSeenOffset_PseudoPointer[6] ) {
19,760             if ( refEnd - (char *)LastSeenOffset_PseudoPointer[6] < (16) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the first position of the LookAhead!
19,761                 // Fallback to memmem(): [
19,762 //         FoundAtPosition = Railgun_BawBaw_reverse (refStartSW3, encStart, (uint32_t)(refEnd-refStartSW3), 16);
19,763 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW3 - src) -(uint32_t)(refEnd-refStartSW3 -1), srcR + (srcSize - 1) - (encStart - src) -(16-1), (uint32_t)(refEnd-refStartSW3), 16);
19,764         if (FoundAtPosition2!=NULL) {
19,765 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(16-1);
19,766         } else FoundAtPosition = NULL;
19,767         // Fallback to memmem(): ]
19,768         } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[6];
19,769 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p: %p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
19,770         } else FoundAtPosition = NULL;
19,771 #endif
19,772
19,773         if (FoundAtPosition!=NULL) {

```

```

19,774             *retMatch=16;
19,775             // The first four bits should be:
19,776                                     // 0001b = 1
19,777             *retIndex=(((refEnd-FoundAtPosition)<<4)&0xFFFFF0)!0x0001; // xx ... x[OOLL]
19,778             *ShortMediumLongOFFSET=3;
19,779             return;
19,780         }
19,781     }
19,782     } // B-TREE heuristic
19,783
19,784 #if defined(BtreeHEURISTIC)
19,785     if ( LastSeenOffset_PseudoPointer[8] != 0 )
19,786 #endif
19,787     { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
19,788
19,789     // #30 36:(5+1)+1= 5.1 (128GB)
19,790     if (refStartSW5ryuu >= refStart) {
19,791
19,792     #ifndef BtreeHEURISTIC
19,793     // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW5ryuu, encStart, (uint64_t)(refEnd-refStartSW5ryuu), 18*2);
19,794     FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW5ryuu - src) - (uint64_t)(refEnd-refStartSW5ryuu - 1), srcR + (srcSize - 1) - (encStart - src)
19,795     -(18*2-1), (uint64_t)(refEnd-refStartSW5ryuu), 18*2);
19,796     if (FoundAtPosition2!=NULL) {
19,797     FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(18*2-1);
19,798     } else FoundAtPosition = NULL;
19,799     #else
19,800     if ( refStartSW5ryuu <= (char *)LastSeenOffset_PseudoPointer[8] ) {
19,801     if ( refEnd - (char *)LastSeenOffset_PseudoPointer[8] < (36) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
19,802     first position of the LookAhead!
19,803     // Fallback to memmem(): [
19,804     FoundAtPosition = Railgun_BawBaw_reverse (refStartSW5ryuu, encStart, (uint64_t)(refEnd-refStartSW5ryuu), 18*2);
19,805     FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW5ryuu - src) - (uint64_t)(refEnd-refStartSW5ryuu - 1), srcR + (srcSize - 1) - (encStart - src)
19,806     -(18*2-1), (uint64_t)(refEnd-refStartSW5ryuu), 18*2);
19,807     if (FoundAtPosition2!=NULL) {
19,808     FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(18*2-1);
19,809     } else FoundAtPosition = NULL;
19,810     // Fallback to memmem(): [
19,811     } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[8];
19,812     //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
19,813     %p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
19,814     } else FoundAtPosition = NULL;
19,815     #endif
19,816
19,817     if (FoundAtPosition!=NULL) {
19,818     *retMatch=18*2;
19,819     // The first 8 bits should be 0x93, in order to retain 4bytes-longness of 'retIndex' we play dirty - not tagging the 5bytes chunk with 0x93
19,820     but as "usual" - the address subchunk:
19,821     *retIndex=(((refEnd-FoundAtPosition)<<3)&0xFFFFFFFF)!0x00; // 000, 18:(5+1)=3
19,822     *ShortMediumLongOFFSET=5+1 +3333;
19,823     *HowManyDittoTagsToEmit = 1;
19,824     return;
19,825     }
19,826     } // B-TREE heuristic
19,827
19,828 #if defined(BtreeHEURISTIC)
19,829     if ( LastSeenOffset_PseudoPointer[11] != 0 ) // 2020-Jan-01

```

```
19,827 #endif
19,828     { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
19,829
19,830 // #31 28:(5+1)= 4.6 (128GB)
19,831 if (refStartSW5ryuu >= refStart) {
19,832
19,833 // 2020-Jan-01 [
19,834 #ifndef BtreeHEURISTIC
19,835 // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW5ryuu, encStart, (uint64_t)(refEnd-refStartSW5ryuu), 28);
19,836 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW5ryuu - src) - (uint64_t)(refEnd-refStartSW5ryuu - 1), srcR + (srcSize - 1) - (encStart - src)
19,837 -(28-1), (uint64_t)(refEnd-refStartSW5ryuu), 28);
19,838 if (FoundAtPosition2!=NULL) {
19,839 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (28-1);
19,840 } else FoundAtPosition = NULL;
19,841 #else
19,842 if ( refStartSW5ryuu <= (char *)LastSeenOffset_PseudoPointer[11] ) {
19,843 if ( refEnd - (char *)LastSeenOffset_PseudoPointer[11] < (28) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
19,844 first position of the LookAhead!
19,845 // Fallback to memmem(): [
19,846 FoundAtPosition = Railgun_BawBaw_reverse (refStartSW5ryuu, encStart, (uint64_t)(refEnd-refStartSW5ryuu), 28);
19,847 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW5ryuu - src) - (uint64_t)(refEnd-refStartSW5ryuu - 1), srcR + (srcSize - 1) - (encStart - src)
19,848 -(28-1), (uint64_t)(refEnd-refStartSW5ryuu), 28);
19,849 if (FoundAtPosition2!=NULL) {
19,850 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (28-1);
19,851 } else FoundAtPosition = NULL;
19,852 // Fallback to memmem(): ]
19,853 } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[11];
19,854 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
19,855 %p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
19,856 } else FoundAtPosition = NULL;
19,857 #endif
19,858 // 2020-Jan-01 ]
19,859
19,860 if (FoundAtPosition!=NULL) {
19,861 *retMatch=28;
19,862 // The first 8 bits should be 0x93, in order to retain 4bytes-longness of 'retIndex' we play dirty - not tagging the 5bytes chunk with 0x93
19,863 but as "usual" - the address subchunk:
19,864 *retIndex=((refEnd-FoundAtPosition)<<3)&0xFFFFFFFF8)!0x04; // 001, 32:(5+1)=5.3
19,865 *ShortMediumLongOFFSET=5+1 +3333;
19,866 return;
19,867 }
19,868 } // B-TREE heuristic
19,869
19,870 // #32 14:3= 4.6 (128KB)
19,871 #if defined(BtreeHEURISTIC)
19,872 if ( LastSeenOffset_PseudoPointer[5] != 0 )
19,873 #endif
19,874 { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
19,875
19,876 if (refStartSW128kb >= refStart) {
19,877
19,878 #ifndef BtreeHEURISTIC
19,879 // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW128kb, encStart, (uint32_t)(refEnd-refStartSW128kb), 14);
19,880 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW128kb - src) - (uint32_t)(refEnd-refStartSW128kb - 1), srcR + (srcSize - 1) - (encStart - src)
19,881 -(14-1), (uint32_t)(refEnd-refStartSW128kb), 14);
19,882 if (FoundAtPosition2!=NULL) {
19,883 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (14-1);
19,884 }
19,885 #endif
19,886 }
```

```

19,879         } else FoundAtPosition = NULL;
19,880 #else
19,881         if ( refStartSW128kb <= (char *)LastSeenOffset_PseudoPointer[5] ) {
19,882             if ( refEnd - (char *)LastSeenOffset_PseudoPointer[5] < (14) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
first position of the LookAhead!
19,883                 // Fallback to memmem(): [
19,884                 FoundAtPosition = Railgun_BawBaw_reverse (refStartSW128kb, encStart, (uint32_t)(refEnd-refStartSW128kb), 14);
19,885                 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW128kb - src) -(uint32_t)(refEnd-refStartSW128kb -1), srcR + (srcSize - 1) - (encStart - src)
-(14-1), (uint32_t)(refEnd-refStartSW128kb), 14);
19,886                 if (FoundAtPosition2!=NULL) {
19,887                     FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(14-1);
19,888                 } else FoundAtPosition = NULL;
19,889                 // Fallback to memmem(): ]
19,890                 } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[5];
19,891 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
%p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
19,892             } else FoundAtPosition = NULL;
19,893 #endif
19,894
19,895         if (FoundAtPosition!=NULL) {
19,896             *retMatch=14;
19,897             // The first four bits should be:
19,898             *retIndex=((refEnd-FoundAtPosition)<<7)&0xFFFF80)!0x1C;
19,899             *ShortMediumLongOFFSET=3;
19,900             return;
19,901         }
19,902     }
19,903     } // B-TREE heuristic
19,904
19,905     // refEnd=40=encStart is LookAheadBuffer
19,906     //
19,907     // 1      2      3      4
19,908     // 12345678901234567890123456789!0
19,909     // Bombastica Playlist Ecstatica!tica
19,910     // [ ] [ ]
19,911     // ^^^ ^^^
19,912     // 36*(4-1) is within 1..39
19,913     // if (40-36 < 4) then memmem()
19,914     // The buffer starts at refStartSW3ryuu and is long (uint32_t)(refEnd-refStartSW3ryuu) or:
19,915     // 1..(40-1) or 1..39
19,916     // SearchIntoSlidingWindow(&HowManyDittoTagsToEmit, &ShortMediumLongOFFSET, &index, &match, refStart,
&src[srcIndex], &src[srcIndex], encEnd, src, srcR, srcSize);
19,917     // SearchIntoSlidingWindow(unsigned int* HowManyDittoTagsToEmit, unsigned int* ShortMediumLongOFFSET, uint64_t* retIndex, unsigned int* retMatch, char* refStart, char*
refEnd, char* encStart, char* encEnd, char* src, char* srcR, uint64_t srcSize ){
19,918
19,919     // #33 18:(3+1)= 4.5 (2MB)
19,920     #if defined(BtreeHEURISTIC)
19,921         if ( LastSeenOffset_PseudoPointer[7] != 0 )
19,922     #endif
19,923         { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
19,924
19,925         if (refStartSW3ryuu >= refStart) {
19,926
19,927         #ifndef BtreeHEURISTIC
19,928         // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW3ryuu, encStart, (uint32_t)(refEnd-refStartSW3ryuu), 18);
19,929         FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW3ryuu - src) -(uint32_t)(refEnd-refStartSW3ryuu -1), srcR + (srcSize - 1) - (encStart - src)
-(18-1), (uint32_t)(refEnd-refStartSW3ryuu), 18);
19,930         if (FoundAtPosition2!=NULL) {

```

```

19,931 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(18-1);
19,932 //printf("\nFoundAtPosition 18:(3+1): %lu\n", FoundAtPosition); //debugprint
19,933 } else FoundAtPosition = NULL;
19,934 #else
19,935     if ( refStartSW3ryuu <= (char *)LastSeenOffset_PseudoPointer[7] ) {
19,936         if ( refEnd - (char *)LastSeenOffset_PseudoPointer[7] < (18) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
first position of the LookAhead!
19,937             // Fallback to memmem(): [
19,938             FoundAtPosition = Railgun_BawBaw_reverse (refStartSW3ryuu, encStart, (uint32_t)(refEnd-refStartSW3ryuu), 18);
19,939             FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW3ryuu - src) -(uint32_t)(refEnd-refStartSW3ryuu -1), srcR + (srcSize - 1) - (encStart - src)
-(18-1), (uint32_t)(refEnd-refStartSW3ryuu), 18);
19,940             if (FoundAtPosition2!=NULL) {
19,941                 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(18-1);
19,942                 //printf("\nFoundAtPosition 18:(3+1): %lu\n", FoundAtPosition); //debugprint
19,943             } else FoundAtPosition = NULL;
19,944             // Fallback to memmem(): ]
19,945             } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[7];
19,946 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
%p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
19,947         } else FoundAtPosition = NULL;
19,948 #endif
19,949
19,950     if (FoundAtPosition!=NULL) {
19,951         *retMatch=18;
19,952         // The first four bits should be:
19,953                                     // 0101b = 5
19,954         *retIndex=((refEnd-FoundAtPosition)<<3)&0xFFFFF8)!0x02; // 010, 18:(3+1)=4.5
19,955         *ShortMediumLongOFFSET=3+1 +3333;
19,956         return;
19,957     }
19,958 }
19,959     } // B-TREE heuristic
19,960
19,961 // #34 16:4= 4 (16MB)F
19,962 #if defined(BtreeHEURISTIC)
19,963     if ( LastSeenOffset_PseudoPointer[6] != 0 )
19,964 #endif
19,965     { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
19,966
19,967     if (refStartSW1a >= refStart) {
19,968
19,969     #ifndef BtreeHEURISTIC
19,970 // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW1a, encStart, (uint32_t)(refEnd-refStartSW1a), 16);
19,971 FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW1a - src) -(uint32_t)(refEnd-refStartSW1a -1), srcR + (srcSize - 1) - (encStart - src) -(16-
1), (uint32_t)(refEnd-refStartSW1a), 16);
19,972         if (FoundAtPosition2!=NULL) {
19,973             FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(16-1);
19,974         } else FoundAtPosition = NULL;
19,975     #else
19,976         if ( refStartSW1a <= (char *)LastSeenOffset_PseudoPointer[6] ) {
19,977             if ( refEnd - (char *)LastSeenOffset_PseudoPointer[6] < (16) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
first position of the LookAhead!
19,978                 // Fallback to memmem(): [
19,979 // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW1a, encStart, (uint32_t)(refEnd-refStartSW1a), 16);
19,980 FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW1a - src) -(uint32_t)(refEnd-refStartSW1a -1), srcR + (srcSize - 1) - (encStart - src) -(16-
1), (uint32_t)(refEnd-refStartSW1a), 16);
19,981                 if (FoundAtPosition2!=NULL) {
19,982                     FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(16-1);

```

```

19,983     } else FoundAtPosition = NULL;
19,984         // Fallback to memmem(): ]
19,985     } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[6];
19,986 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
%p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
19,987     } else FoundAtPosition = NULL;
19,988 #endif
19,989
19,990     if (FoundAtPosition!=NULL) {
19,991         *retMatch=16;
19,992         // The first four bits should be:
19,993
19,994         *retIndex=((refEnd-FoundAtPosition)<<8)&0xFFFFF00)!0xA3; // xx ... x[LL00] // 0011b = 0x3; 11xx0011b, xx==11
19,995         *ShortMediumLongOFFSET=4;
19,996         return;
19,997     }
19,998 }
19,999     } // B-TREE heuristic
20,000
20,001 // #35 12:3=         4      (1MB)
20,002 /*
20,003 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) [
20,004 if (refStartHOT >= refStart)
20,005 if (refStartHOT < refEnd) {
20,006 FoundAtPosition = Railgun_Trolldom_64(refStartHOT, encStart, (uint32_t)(refEnd-refStartHOT), 12);
20,007     if (FoundAtPosition!=NULL) {
20,008         *retMatch=12;
20,009         // The first four bits should be:
20,010
20,011         *retIndex=((refEnd-FoundAtPosition)<<4)&0xFFFFF0)!0x0005; // xx ... x[OOLL] // 0101b = 5
20,012         *ShortMediumLongOFFSET=3;
20,013         return;
20,014     }
20,015 }
20,016 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) ]
20,017
20,018 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) [
20,019 if (refStartCOLDERbig >= refStart)
20,020 if (refStartCOLDERbig < refEnd) {
20,021 FoundAtPosition = Railgun_Trolldom_64(refStartCOLDERbig, encStart, (uint32_t)(refEnd-refStartCOLDERbig), 12);
20,022     if (FoundAtPosition!=NULL) {
20,023         *retMatch=12;
20,024         // The first four bits should be:
20,025
20,026         *retIndex=((refEnd-FoundAtPosition)<<4)&0xFFFFF0)!0x0005; // xx ... x[OOLL] // 0101b = 5
20,027         *ShortMediumLongOFFSET=3;
20,028         return;
20,029     }
20,030 }
20,031 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) ]
20,032 /*
20,033 #if defined(BtreeHEURISTIC)
20,034     if ( LastSeenOffset_PseudoPointer[4] != 0 )
20,035 #endif
20,036     { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
20,037
20,038 if (refStartSW3 >= refStart) {
20,039

```

```

20,040 #ifndef BtreeHEURISTIC
20,041 //      FoundAtPosition = Railgun_BawBaw_reverse (refStartSW3, encStart, (uint32_t)(refEnd-refStartSW3), 12);
20,042 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW3 - src) -(uint32_t)(refEnd-refStartSW3 -1), srcR + (srcSize - 1) - (encStart - src) -(12-
1), (uint32_t)(refEnd-refStartSW3), 12);
20,043     if (FoundAtPosition2!=NULL) {
20,044 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(12-1);
20,045     } else FoundAtPosition = NULL;
20,046 #else
20,047     if ( refStartSW3 <= (char *)LastSeenOffset_PseudoPointer[4] ) {
20,048         if ( refEnd - (char *)LastSeenOffset_PseudoPointer[4] < (12) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
first position of the LookAhead!
20,049             // Fallback to memmem(): [
20,050 //      FoundAtPosition = Railgun_BawBaw_reverse (refStartSW3, encStart, (uint32_t)(refEnd-refStartSW3), 12);
20,051 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW3 - src) -(uint32_t)(refEnd-refStartSW3 -1), srcR + (srcSize - 1) - (encStart - src) -(12-
1), (uint32_t)(refEnd-refStartSW3), 12);
20,052     if (FoundAtPosition2!=NULL) {
20,053 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(12-1);
20,054     } else FoundAtPosition = NULL;
20,055         // Fallback to memmem(): ]
20,056     } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[4];
20,057 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
%p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
20,058     } else FoundAtPosition = NULL;
20,059 #endif
20,060
20,061     if (FoundAtPosition!=NULL) {
20,062         *retMatch=12;
20,063         // The first four bits should be:
20,064
20,065         *retIndex=((refEnd-FoundAtPosition)<<4)&0xFFFFF0)!0x0005; // xx ... x[OOLL] // 0101b = 5
20,066         *ShortMediumLongOFFSET=3;
20,067         return;
20,068     }
20,069 }
20,070 } // B-TREE heuristic
20,071
20,072 // #36 8:2= 4 (4KB)
20,073 /*
20,074 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) [
20,075 if (refStartHOTTER >= refStart)
20,076 if (refStartHOTTER < refEnd) {
20,077 FoundAtPosition = Railgun_Trollldom_64(refStartHOTTER, encStart, (uint32_t)(refEnd-refStartHOTTER), 8);
20,078     if (FoundAtPosition!=NULL) {
20,079         *retMatch=8;
20,080         // The first four bits should be:
20,081
20,082         *retIndex=((refEnd-FoundAtPosition)<<4)&0xFFF0)!0x000A; // xx ... x[OOLL] // 1010b = A
20,083         *ShortMediumLongOFFSET=2;
20,084         return;
20,085     }
20,086 }
20,087 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) ]
20,088 */
20,089 #if defined(BtreeHEURISTIC)
20,090     if ( LastSeenOffset_PseudoPointer[2] != 0 )
20,091 #endif
20,092     { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
20,093

```

```

20,094 if (refStartSW2 >= refStart) {
20,095
20,096 #ifndef BtreeHEURISTIC
20,097 //      FoundAtPosition = Railgun_BawBaw_reverse (refStartSW2, encStart, (uint32_t)(refEnd-refStartSW2), 8);
20,098 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW2 - src) - (uint32_t)(refEnd-refStartSW2 - 1), srcR + (srcSize - 1) - (encStart - src) - (8-1),
(uint32_t)(refEnd-refStartSW2), 8);
20,099 if (FoundAtPosition2!=NULL) {
20,100 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (8-1);
20,101 } else FoundAtPosition = NULL;
20,102 #else
20,103 if ( refStartSW2 <= (char *)LastSeenOffset_PseudoPointer[2] ) {
20,104 if ( refEnd - (char *)LastSeenOffset_PseudoPointer[2] < (8) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
first position of the LookAhead!
20,105 // Fallback to memmem(): [
20,106 //      FoundAtPosition = Railgun_BawBaw_reverse (refStartSW2, encStart, (uint32_t)(refEnd-refStartSW2), 8);
20,107 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW2 - src) - (uint32_t)(refEnd-refStartSW2 - 1), srcR + (srcSize - 1) - (encStart - src) - (8-1),
(uint32_t)(refEnd-refStartSW2), 8);
20,108 if (FoundAtPosition2!=NULL) {
20,109 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (8-1);
20,110 } else FoundAtPosition = NULL;
20,111 // Fallback to memmem(): ]
20,112 } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[2];
20,113 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
%p,%p\n", refStartSW3,refEnd, FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0]); //debugprint
20,114 } else FoundAtPosition = NULL;
20,115 #endif
20,116
20,117 if (FoundAtPosition!=NULL) {
20,118 *retMatch=8;
20,119 // The first four bits should be:
20,120 // 1010b = A
20,121 *retIndex=(((refEnd-FoundAtPosition)<<4)&0xFFF0)!0x000A; // xx ... x[00LL]
20,122 *ShortMediumLongOFFSET=2;
20,123 return;
20,124 }
20,125 }
20,126 } // B-TREE heuristic
20,127
20,128 // #37 4:1= 4 (16B)
20,129 /*
20,130 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) [
20,131 if (refStartHOTEST >= refStart)
20,132 if (refStartHOTEST < refEnd) {
20,133 FoundAtPosition = Railgun_Trollldom_64(refStartHOTEST, encStart, (uint32_t)(refEnd-refStartHOTEST), 4);
20,134 if (FoundAtPosition!=NULL) {
20,135 *retMatch=4;
20,136 // The first four bits should be:
20,137 // 1111b = F
20,138 *retIndex=(((refEnd-FoundAtPosition)<<4)&0xFFF0)!0x000F; // xx ... x[00LL]
20,139 *ShortMediumLongOFFSET=1;
20,140 return;
20,141 }
20,142 }
20,143 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) ]
20,144 */
20,145 #if defined(BtreeHEURISTIC)
20,146 if ( LastSeenOffset_PseudoPointer[0] != 0 )
20,147 #endif

```



```
20,148 { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
20,149
20,150 if (refStartSW1 >= refStart) {
20,151
20,152 #ifndef BtreeHEURISTIC
20,153 //GLOBAL_Railgun_INVOCATIONS_004_1_bytes++;
20,154 // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW1, encStart, (uint32_t)(refEnd-refStartSW1), 4);
20,155 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW1 - src) - (uint32_t)(refEnd-refStartSW1 - 1), srcR + (srcSize - 1) - (encStart - src) - (4-1),
(uint32_t)(refEnd-refStartSW1), 4);
20,156 if (FoundAtPosition2!=NULL) {
20,157 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (4-1);
20,158 } else FoundAtPosition = NULL;
20,159 #else
20,160 if ( refStartSW1 <= (char *)LastSeenOffset_PseudoPointer[0] ) {
20,161 if ( refEnd - (char *)LastSeenOffset_PseudoPointer[0] < (4) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
first position of the LookAhead!
20,162 // Fallback to memmem(): [
20,163 //GLOBAL_Railgun_INVOCATIONS_004_1_bytes++;
20,164 // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW1, encStart, (uint32_t)(refEnd-refStartSW1), 4);
20,165 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW1 - src) - (uint32_t)(refEnd-refStartSW1 - 1), srcR + (srcSize - 1) - (encStart - src) - (4-1),
(uint32_t)(refEnd-refStartSW1), 4);
20,166 if (FoundAtPosition2!=NULL) {
20,167 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (4-1);
20,168 } else FoundAtPosition = NULL;
20,169 // Fallback to memmem(): ]
20,170 } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[0];
20,171 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
%p,%p\n", refStartSW3,refEnd, FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0]); //debugprint
20,172 } else FoundAtPosition = NULL;
20,173 #endif
20,174
20,175 if (FoundAtPosition!=NULL) {
20,176 *retMatch=4;
20,177 // The first four bits should be:
20,178 // 1111b = F
20,179 *retIndex=((refEnd-FoundAtPosition)<<4)&0xFFF0)!0x000F; // xx ... x[OOLL]
20,180 *ShortMediumLongOFFSET=1;
20,181 return;
20,182 }
20,183 }
20,184 } // B-TREE heuristic
20,185
20,186 //38 18:(4+1)= 3.6 (512MB)
20,187 #if defined(BtreeHEURISTIC)
20,188 if ( LastSeenOffset_PseudoPointer[7] != 0 )
20,189 #endif
20,190 { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
20,191
20,192 if (refStartSW5 >= refStart) {
20,193
20,194 #ifndef BtreeHEURISTIC
20,195 // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW5, encStart, (uint32_t)(refEnd-refStartSW5), 18);
20,196 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW5 - src) - (uint32_t)(refEnd-refStartSW5 - 1), srcR + (srcSize - 1) - (encStart - src) - (18-
1), (uint32_t)(refEnd-refStartSW5), 18);
20,197 if (FoundAtPosition2!=NULL) {
20,198 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (18-1);
20,199 //printf("\nFoundAtPosition 18:(4+1): %lu\n", FoundAtPosition); //debugprint
20,200 } else FoundAtPosition = NULL;
```

```

20,201 #else
20,202     if ( refStartSW5 <= (char *)LastSeenOffset_PseudoPointer[7] ) {
20,203         if ( refEnd - (char *)LastSeenOffset_PseudoPointer[7] < (18) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
first position of the LookAhead!
20,204             // Fallback to memmem(): [
20,205             FoundAtPosition = Railgun_BawBaw_reverse (refStartSW5, encStart, (uint32_t)(refEnd-refStartSW5), 18);
20,206             FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW5 - src) - (uint32_t)(refEnd-refStartSW5 - 1), srcR + (srcSize - 1) - (encStart - src) - (18-
1), (uint32_t)(refEnd-refStartSW5), 18);
20,207             if (FoundAtPosition2!=NULL) {
20,208                 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (18-1);
20,209                 //printf("\nFoundAtPosition 18:(4+1): %lu\n", FoundAtPosition); //debugprint
20,210             } else FoundAtPosition = NULL;
20,211             // Fallback to memmem(): ]
20,212             } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[7];
20,213 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
%p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
20,214             } else FoundAtPosition = NULL;
20,215 #endif
20,216
20,217     if (FoundAtPosition!=NULL) {
20,218         *retMatch=18;
20,219         // The first 8 bits should be 0x93, in order to retain 4bytes-longness of 'retIndex' we play dirty - not tagging the 5bytes chunk with 0x93
but as "usual" - the address subchunk:
20,220         *retIndex=((refEnd-FoundAtPosition)<<3)&0xFFFFFFFF8)!0x01; // 100, 18:(4+1)=3.6
20,221         *ShortMediumLongOFFSET=4+1 +3333;
20,222         return;
20,223     }
20,224 }
20,225     } // B-TREE heuristic
20,226
20,227 // #39 14:4=          3.5 (16MB)
20,228 #if defined(BtreeHEURISTIC)
20,229     if ( LastSeenOffset_PseudoPointer[5] != 0 )
20,230 #endif
20,231     { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
20,232
20,233     if (refStartSW1a >= refStart) {
20,234
20,235     #ifndef BtreeHEURISTIC
20,236     // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW1a, encStart, (uint32_t)(refEnd-refStartSW1a), 14);
20,237     FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW1a - src) - (uint32_t)(refEnd-refStartSW1a - 1), srcR + (srcSize - 1) - (encStart - src) - (14-
1), (uint32_t)(refEnd-refStartSW1a), 14);
20,238     if (FoundAtPosition2!=NULL) {
20,239     FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (14-1);
20,240     } else FoundAtPosition = NULL;
20,241 #else
20,242     if ( refStartSW1a <= (char *)LastSeenOffset_PseudoPointer[5] ) {
20,243         if ( refEnd - (char *)LastSeenOffset_PseudoPointer[5] < (14) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
first position of the LookAhead!
20,244             // Fallback to memmem(): [
20,245             FoundAtPosition = Railgun_BawBaw_reverse (refStartSW1a, encStart, (uint32_t)(refEnd-refStartSW1a), 14);
20,246             FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW1a - src) - (uint32_t)(refEnd-refStartSW1a - 1), srcR + (srcSize - 1) - (encStart - src) - (14-
1), (uint32_t)(refEnd-refStartSW1a), 14);
20,247             if (FoundAtPosition2!=NULL) {
20,248                 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (14-1);
20,249             } else FoundAtPosition = NULL;
20,250             // Fallback to memmem(): ]
20,251             } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[5];

```

```

20,252 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
%p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
20,253     } else FoundAtPosition = NULL;
20,254 #endif
20,255
20,256     if (FoundAtPosition!=NULL) {
20,257         *retMatch=14;
20,258         // The first four bits should be:
20,259                                     // 0011b = 0x3; 11xx0011b, xx==11
20,260         *retIndex=(((refEnd-FoundAtPosition)<<8)&0xFFFFFFFF00)!0xB3; // xx ... x[LL00]
20,261         *ShortMediumLongOFFSET=4;
20,262         return;
20,263     }
20,264 }
20,265     } // B-TREE heuristic
20,266
20,267 // #40 18:(5+1)=      3   (128GB)
20,268 #if defined(BtreeHEURISTIC)
20,269     if ( LastSeenOffset_PseudoPointer[7] != 0 )
20,270 #endif
20,271     { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
20,272
20,273     if (refStartSW5ryuu >= refStart) {
20,274
20,275     #ifndef BtreeHEURISTIC
20,276     // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW5ryuu, encStart, (uint64_t)(refEnd-refStartSW5ryuu), 18);
20,277     FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW5ryuu - src) -(uint64_t)(refEnd-refStartSW5ryuu -1), srcR + (srcSize - 1) - (encStart - src)
20,278     -(18-1), (uint64_t)(refEnd-refStartSW5ryuu), 18);
20,279     if (FoundAtPosition2!=NULL) {
20,280     FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(18-1);
20,281     //printf("\nFoundAtPosition 18:(5+1): %lu\n", FoundAtPosition);//debugprint
20,282     } else FoundAtPosition = NULL;
20,283     #else
20,284     if ( refStartSW5ryuu <= (char *)LastSeenOffset_PseudoPointer[7] ) {
20,285     if ( refEnd - (char *)LastSeenOffset_PseudoPointer[7] < (18) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
20,286     first position of the LookAhead!
20,287     // Fallback to memmem(): [
20,288     FoundAtPosition = Railgun_BawBaw_reverse (refStartSW5ryuu, encStart, (uint64_t)(refEnd-refStartSW5ryuu), 18);
20,289     FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW5ryuu - src) -(uint64_t)(refEnd-refStartSW5ryuu -1), srcR + (srcSize - 1) - (encStart - src)
20,290     -(18-1), (uint64_t)(refEnd-refStartSW5ryuu), 18);
20,291     if (FoundAtPosition2!=NULL) {
20,292     FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(18-1);
20,293     //printf("\nFoundAtPosition 18:(5+1): %lu\n", FoundAtPosition);//debugprint
20,294     } else FoundAtPosition = NULL;
20,295     // Fallback to memmem(): [
20,296     } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[7];
20,297     //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
20,298     %p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
20,299     } else FoundAtPosition = NULL;
20,300 #endif
20,301
20,302     if (FoundAtPosition!=NULL) {
20,303         *retMatch=18;
20,304         // The first 8 bits should be 0x93, in order to retain 4bytes-longness of 'retIndex' we play dirty - not tagging the 5bytes chunk with 0x93
20,305         but as "usual" - the address subchunk:
20,306         *retIndex=(((refEnd-FoundAtPosition)<<3)&0xFFFFFFFFF8)!0x00; // 000, 18:(5+1)=3
20,307         *ShortMediumLongOFFSET=5+1 +3333;
20,308         return;

```

```

20,304     }
20,305 }
20,306     } // B-TREE heuristic
20,307
20,308 // #41 12:4=          3    (16MB)
20,309 #if defined(BtreeHEURISTIC)
20,310     if ( LastSeenOffset_PseudoPointer[4] != 0 )
20,311 #endif
20,312     { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
20,313
20,314     if (refStartSW1a >= refStart) {
20,315
20,316 #ifndef BtreeHEURISTIC
20,317 //         FoundAtPosition = Railgun_BawBaw_reverse (refStartSW1a, encStart, (uint32_t)(refEnd-refStartSW1a), 12);
20,318 FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW1a - src) - (uint32_t)(refEnd-refStartSW1a - 1), srcR + (srcSize - 1) - (encStart - src) - (12-
1), (uint32_t)(refEnd-refStartSW1a), 12);
20,319     if (FoundAtPosition2!=NULL) {
20,320 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (12-1);
20,321     } else FoundAtPosition = NULL;
20,322 #else
20,323     if ( refStartSW1a <= (char *)LastSeenOffset_PseudoPointer[4] ) {
20,324         if ( refEnd - (char *)LastSeenOffset_PseudoPointer[4] < (12) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
first position of the LookAhead!
20,325             // Fallback to memmem(): [
20,326 //         FoundAtPosition = Railgun_BawBaw_reverse (refStartSW1a, encStart, (uint32_t)(refEnd-refStartSW1a), 12);
20,327 FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW1a - src) - (uint32_t)(refEnd-refStartSW1a - 1), srcR + (srcSize - 1) - (encStart - src) - (12-
1), (uint32_t)(refEnd-refStartSW1a), 12);
20,328     if (FoundAtPosition2!=NULL) {
20,329 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (12-1);
20,330     } else FoundAtPosition = NULL;
20,331             // Fallback to memmem(): ]
20,332     } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[4];
20,333 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
%p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
20,334     } else FoundAtPosition = NULL;
20,335 #endif
20,336
20,337     if (FoundAtPosition!=NULL) {
20,338         *retMatch=12;
20,339         // The first four bits should be:
20,340
20,341         *retIndex=((refEnd-FoundAtPosition)<<8)&0xFFFFFFFF)!0xC3; // xx ... x[LL00] // 0011b = 0x3; 11xx0011b, xx==11
20,342         *ShortMediumLongOFFSET=4;
20,343         return;
20,344     }
20,345 }
20,346     } // B-TREE heuristic
20,347
20,348 // #42 6:2=          3    (512B)
20,349 #if defined(BtreeHEURISTIC)
20,350     if ( LastSeenOffset_PseudoPointer[1] != 0 )
20,351 #endif
20,352     { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
20,353
20,354     if (refStartSW512 >= refStart) {
20,355
20,356 #ifndef BtreeHEURISTIC
20,357 //         FoundAtPosition = Railgun_BawBaw_reverse (refStartSW512, encStart, (uint32_t)(refEnd-refStartSW512), 6);

```

```

20,358 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW512 - src) - (uint32_t)(refEnd-refStartSW512 -1), srcR + (srcSize - 1) - (encStart - src) -
(6-1), (uint32_t)(refEnd-refStartSW512), 6);
20,359     if (FoundAtPosition2!=NULL) {
20,360 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(6-1);
20,361     } else FoundAtPosition = NULL;
20,362 #else
20,363     if ( refStartSW512 <= (char *)LastSeenOffset_PseudoPointer[1] ) {
20,364         if ( refEnd - (char *)LastSeenOffset_PseudoPointer[1] < (6) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
first position of the LookAhead!
20,365             // Fallback to memmem(): [
20,366 //         FoundAtPosition = Railgun_BawBaw_reverse (refStartSW512, encStart, (uint32_t)(refEnd-refStartSW512), 6);
20,367 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW512 - src) - (uint32_t)(refEnd-refStartSW512 -1), srcR + (srcSize - 1) - (encStart - src) -
(6-1), (uint32_t)(refEnd-refStartSW512), 6);
20,368         if (FoundAtPosition2!=NULL) {
20,369 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(6-1);
20,370         } else FoundAtPosition = NULL;
20,371             // Fallback to memmem(): ]
20,372         } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[1];
20,373 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
%p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
20,374         } else FoundAtPosition = NULL;
20,375 #endif
20,376
20,377     if (FoundAtPosition!=NULL) {
20,378         *retMatch=6;
20,379         // The first four bits should be:
20,380         *retIndex=((refEnd-FoundAtPosition)<<7)&0xFF80)!0x4C; // (4+0)C
20,381         *ShortMediumLongOFFSET=2;
20,382         return;
20,383     }
20,384 }
20,385     } // B-TREE heuristic
20,386
20,387 // #43 8:3=          2.6 (1MB)
20,388 /*
20,389 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) [
20,390 if (refStartHOT >= refStart)
20,391 if (refStartHOT < refEnd) {
20,392 FoundAtPosition = Railgun_Trollldom_64(refStartHOT, encStart, (uint32_t)(refEnd-refStartHOT), 8);
20,393     if (FoundAtPosition!=NULL) {
20,394         *retMatch=8;
20,395         // The first four bits should be:
20,396
20,397         *retIndex=((refEnd-FoundAtPosition)<<4)&0xFFFFF0)!0x0009; // xx ... x[OOLL] // 1001b = 9
20,398         *ShortMediumLongOFFSET=3;
20,399         return;
20,400     }
20,401 }
20,402 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) ]
20,403
20,404 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) [
20,405 if (refStartCOLDERbig >= refStart)
20,406 if (refStartCOLDERbig < refEnd) {
20,407 FoundAtPosition = Railgun_Trollldom_64(refStartCOLDERbig, encStart, (uint32_t)(refEnd-refStartCOLDERbig), 8);
20,408     if (FoundAtPosition!=NULL) {
20,409         *retMatch=8;
20,410         // The first four bits should be:
20,411
20,411         // 1001b = 9

```

```

20,412         *retIndex=((refEnd-FoundAtPosition)<<4)&0xFFFFF0)!0x0009; // xx ... x[OOLL]
20,413         *ShortMediumLongOFFSET=3;
20,414         return;
20,415     }
20,416 }
20,417 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) ]
20,418 /*
20,419 // B-TREE heuristic #2:
20,420 #if defined(BtreeHEURISTIC)
20,421     if ( LastSeenOffset_PseudoPointer[2] != 0 )
20,422 #endif
20,423     { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
20,424
20,425     if (refStartSW3 >= refStart) {
20,426
20,427     #ifndef BtreeHEURISTIC
20,428     // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW3, encStart, (uint32_t)(refEnd-refStartSW3), 8);
20,429     FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW3 - src) -(uint32_t)(refEnd-refStartSW3 -1), srcR + (srcSize - 1) - (encStart - src) -(8-1),
20,430     (uint32_t)(refEnd-refStartSW3), 8);
20,431     if (FoundAtPosition2!=NULL) {
20,432     FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(8-1);
20,433     } else FoundAtPosition = NULL;
20,434 #else
20,435     if ( refStartSW3 <= (char *)LastSeenOffset_PseudoPointer[2] ) {
20,436     if ( refEnd - (char *)LastSeenOffset_PseudoPointer[2] < (8) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
20,437     first position of the LookAhead!
20,438     // Fallback to memmem(): [
20,439     FoundAtPosition = Railgun_BawBaw_reverse (refStartSW3, encStart, (uint32_t)(refEnd-refStartSW3), 8);
20,440     FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW3 - src) -(uint32_t)(refEnd-refStartSW3 -1), srcR + (srcSize - 1) - (encStart - src) -(8-1),
20,441     (uint32_t)(refEnd-refStartSW3), 8);
20,442     if (FoundAtPosition2!=NULL) {
20,443     FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(8-1);
20,444     } else FoundAtPosition = NULL;
20,445     // Fallback to memmem(): ]
20,446     } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[2];
20,447 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
20,448 %p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
20,449 } else FoundAtPosition = NULL;
20,450 #endif
20,451
20,452 // ForwardSearch [
20,453 // FoundAtPosition2 = Railgun_Trollldom (srcR + (srcSize - 1) - (refStartSW3 - src) -(uint32_t)(refEnd-refStartSW3), srcR + (srcSize - 1) - (encStart - src) -(8-1),
20,454 // (uint32_t)(refEnd-refStartSW3), 8);
20,455 //
20,456 printf("\nrefStartSW3, encStart                                     = %p, %p", refStartSW3,
20,457 encStart);
20,458 printf("\nsrcR + (srcSize - 1) - (refStartSW3 - src) -(uint32_t)(refEnd-refStartSW3), srcR + (srcSize - 1) - (encStart - src) -(8-1) = %p, %p\n\n", srcR + (srcSize - 1) - (refStartSW3 - src), srcR + (srcSize - 1) - (encStart - src));
20,459 printf("\nrefStartSW3:                                     [");
20,460 for (i=0; i<8; i++) {
20,461     printf("%c",*(refStartSW3 +i));
20,462 }
20,463 printf("]");
20,464 printf("\nsrcR + (srcSize - 1) - (refStartSW3 - src) -(uint32_t)(refEnd-refStartSW3): [");
20,465 for (i=0; i<8; i++) {
20,466     printf("%c",*(srcR + (srcSize - 1) - (refStartSW3 - src) -(uint32_t)(refEnd-refStartSW3) +i));

```

```

20,463 }
20,464 printf("\n");
20,465
20,466 printf("\nencStart:                [");
20,467 for (i=0; i<8; i++) {
20,468   printf("%c",*(encStart +i));
20,469 }
20,470 printf("]");
20,471 printf("\nsrcR + (srcSize - 1) - (encStart - src) -(8-1): [");
20,472 for (i=0; i<8; i++) {
20,473   printf("%c",*(srcR + (srcSize - 1) - (encStart - src) -(8-1) +i));
20,474 }
20,475 printf("]\n");
20,476 */
20,477
20,478 // D:\Nakamichi_Washigan+Nakamichi_Washigan+ForwardSearch>"Nakamichi_Washigan+.exe" Fahrenheit_451_-_Ray_Bradbury.txt
20,479 // -----
20,480 // refStartSW3, encStart                                     = 00000000002377C0, 00000000002377C0
20,481 // srcR + (srcSize - 1) - (refStartSW3 - src) -(uint32_t)(refEnd-refStartSW3), srcR + (srcSize - 1) - (encStart - src) -(8-1) = 00000000002B6D88, 00000000002B6D88
20,482
20,483 // refStartSW3:                                             [Fahrenhe]
20,484 // srcR + (srcSize - 1) - (refStartSW3 - src) -(uint32_t)(refEnd-refStartSW3): [F          ]
20,485
20,486 // encStart:                                             [Fahrenhe]
20,487 // srcR + (srcSize - 1) - (encStart - src) -(8-1): [ehnerhaF]
20,488 // -----
20,489 // refStartSW3, encStart                                     = 00000000002377C0, 00000000002377C1
20,490 // srcR + (srcSize - 1) - (refStartSW3 - src) -(uint32_t)(refEnd-refStartSW3), srcR + (srcSize - 1) - (encStart - src) -(8-1) = 00000000002B6D88, 00000000002B6D87
20,491
20,492 // refStartSW3:                                             [Fahrenhe]
20,493 // srcR + (srcSize - 1) - (refStartSW3 - src) -(uint32_t)(refEnd-refStartSW3): [aF          ]
20,494
20,495 // encStart:                                             [ahrenhei]
20,496 // srcR + (srcSize - 1) - (encStart - src) -(8-1): [iehnerha]
20,497 // -----
20,498 // refStartSW3, encStart                                     = 00000000002377C0, 00000000002377C2
20,499 // srcR + (srcSize - 1) - (refStartSW3 - src) -(uint32_t)(refEnd-refStartSW3), srcR + (srcSize - 1) - (encStart - src) -(8-1) = 00000000002B6D88, 00000000002B6D86
20,500
20,501 // refStartSW3:                                             [Fahrenhe]
20,502 // srcR + (srcSize - 1) - (refStartSW3 - src) -(uint32_t)(refEnd-refStartSW3): [haF          ]
20,503
20,504 // encStart:                                             [hrenheit]
20,505 // srcR + (srcSize - 1) - (encStart - src) -(8-1): [tiehnerh]
20,506 // -----
20,507
20,508 // Actual picture:
20,509 // 1 2 3 4 5 6 7 8 9
20,510 // -----
20,511 // 1 2 3 4 2_3 2 3 9
20,512 //      < BawBawReverse
20,513 // 9 3 2 3_2 4 3 2 1
20,514 //      > Trolldom
20,515 // Paired with '_' is the match - close to the lookahead.
20,516
20,517 //      if (FoundAtPosition2!=NULL) {
20,518 //          printf("\nFoundAtPosition = %p\n",FoundAtPosition);
20,519 //          printf("\nFoundAtPosition2 = %p\n",FoundAtPosition2);
20,520 //          printf("\n%p\n\n",src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(8-1));

```

Page 352 of 466


```

20,575         } // B-TREE heuristic
20,576
20,577 // #45 8:4=          2    (16MB)
20,578 #if defined(BtreeHEURISTIC)
20,579     if ( LastSeenOffset_PseudoPointer[2] != 0 )
20,580 #endif
20,581     { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
20,582
20,583     if (refStartSW1a >= refStart) {
20,584
20,585     #ifndef BtreeHEURISTIC
20,586     //         FoundAtPosition = Railgun_BawBaw_reverse (refStartSW1a, encStart, (uint32_t)(refEnd-refStartSW1a), 8);
20,587     FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW1a - src) - (uint32_t)(refEnd-refStartSW1a - 1), srcR + (srcSize - 1) - (encStart - src) - (8-1), (uint32_t)(refEnd-refStartSW1a), 8);
20,588     if (FoundAtPosition2!=NULL) {
20,589     FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (8-1);
20,590     } else FoundAtPosition = NULL;
20,591 #else
20,592     if ( refStartSW1a <= (char *)LastSeenOffset_PseudoPointer[2] ) {
20,593     if ( refEnd - (char *)LastSeenOffset_PseudoPointer[2] < (8) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
first position of the LookAhead!
20,594     // Fallback to memmem(): [
20,595     //         FoundAtPosition = Railgun_BawBaw_reverse (refStartSW1a, encStart, (uint32_t)(refEnd-refStartSW1a), 8);
20,596     FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW1a - src) - (uint32_t)(refEnd-refStartSW1a - 1), srcR + (srcSize - 1) - (encStart - src) - (8-1), (uint32_t)(refEnd-refStartSW1a), 8);
20,597     if (FoundAtPosition2!=NULL) {
20,598     FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (8-1);
20,599     } else FoundAtPosition = NULL;
20,600     // Fallback to memmem(): ]
20,601     } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[2];
20,602 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
%p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
20,603     } else FoundAtPosition = NULL;
20,604 #endif
20,605
20,606     if (FoundAtPosition!=NULL) {
20,607     *retMatch=8;
20,608     // The first four bits should be:
20,609
20,610     *retIndex=(((refEnd-FoundAtPosition)<<8)&0xFFFFF00)!0xE3; // xx ... x[LL00] // 0011b = 0x3; 11xx0011b, xx==11
20,611     *ShortMediumLongOFFSET=4;
20,612     return;
20,613     }
20,614 }
20,615     } // B-TREE heuristic
20,616
20,617 // #46 6:3=          2    (128KB)
20,618 #if defined(BtreeHEURISTIC)
20,619     if ( LastSeenOffset_PseudoPointer[1] != 0 )
20,620 #endif
20,621     { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
20,622
20,623     if (refStartSW128kb >= refStart) {
20,624
20,625     #ifndef BtreeHEURISTIC
20,626     //         FoundAtPosition = Railgun_BawBaw_reverse (refStartSW128kb, encStart, (uint32_t)(refEnd-refStartSW128kb), 6);
20,627     FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW128kb - src) - (uint32_t)(refEnd-refStartSW128kb - 1), srcR + (srcSize - 1) - (encStart - src) - (6-1), (uint32_t)(refEnd-refStartSW128kb), 6);

```

```

20,628         if (FoundAtPosition2!=NULL) {
20,629     FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(6-1);
20,630     } else FoundAtPosition = NULL;
20,631 #else
20,632     if ( refStartSW128kb <= (char *)LastSeenOffset_PseudoPointer[1] ) {
20,633         if ( refEnd - (char *)LastSeenOffset_PseudoPointer[1] < (6) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
first position of the LookAhead!
20,634             // Fallback to memmem(): [
20,635             FoundAtPosition = Railgun_BawBaw_reverse (refStartSW128kb, encStart, (uint32_t)(refEnd-refStartSW128kb), 6);
20,636     FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW128kb - src) -(uint32_t)(refEnd-refStartSW128kb -1), srcR + (srcSize - 1) - (encStart - src)
-(6-1), (uint32_t)(refEnd-refStartSW128kb), 6);
20,637         if (FoundAtPosition2!=NULL) {
20,638     FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(6-1);
20,639         } else FoundAtPosition = NULL;
20,640             // Fallback to memmem(): ]
20,641             } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[1];
20,642 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
%p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
20,643         } else FoundAtPosition = NULL;
20,644 #endif
20,645
20,646     if (FoundAtPosition!=NULL) {
20,647         *retMatch=6;
20,648         // The first four bits should be:
20,649         *retIndex=((refEnd-FoundAtPosition)<<7)&0xFFFF80)!0x0C;
20,650         *ShortMediumLongOFFSET=3;
20,651         return;
20,652     }
20,653 }
20,654 } // B-TREE heuristic
20,655
20,656 //47 4:2=          2      (4KB)
20,657 /*
20,658 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) [
20,659 if (refStartHOTTER >= refStart)
20,660 if (refStartHOTTER < refEnd) {
20,661     FoundAtPosition = Railgun_Trollldom_64(refStartHOTTER, encStart, (uint32_t)(refEnd-refStartHOTTER), 4);
20,662     if (FoundAtPosition!=NULL) {
20,663         *retMatch=4;
20,664         // The first four bits should be:
20,665                                     // 1110b = E
20,666         *retIndex=((refEnd-FoundAtPosition)<<4)&0xFFF0)!0x000E; // xx ... x[00LL]
20,667         *ShortMediumLongOFFSET=2;
20,668         return;
20,669     }
20,670 }
20,671 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) ]
20,672 */
20,673 #if defined(BtreeHEURISTIC)
20,674     if ( LastSeenOffset_PseudoPointer[0] != 0 )
20,675 #endif
20,676     { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
20,677
20,678     if (refStartSW2 >= refStart) {
20,679
20,680 #ifndef BtreeHEURISTIC
20,681 // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW2, encStart, (uint32_t)(refEnd-refStartSW2), 4);
20,682 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW2 - src) -(uint32_t)(refEnd-refStartSW2 -1), srcR + (srcSize - 1) - (encStart - src) -(4-1),

```

```

(uint32_t)(refEnd-refStartSW2), 4);
20,683     if (FoundAtPosition2!=NULL) {
20,684 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(4-1);
20,685     } else FoundAtPosition = NULL;
20,686 #else
20,687     if ( refStartSW2 <= (char *)LastSeenOffset_PseudoPointer[0] ) {
20,688         if ( refEnd - (char *)LastSeenOffset_PseudoPointer[0] < (4) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
first position of the LookAhead!
20,689             // Fallback to memmem(): [
20,690 //GLOBAL_Railgun_INVOCATIONS_004_2_bytes++;
20,691 // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW2, encStart, (uint32_t)(refEnd-refStartSW2), 4);
20,692 FoundAtPosition2 = Railgun_Trolldom_64 (srcR + (srcSize - 1) - (refStartSW2 - src) -(uint32_t)(refEnd-refStartSW2 -1), srcR + (srcSize - 1) - (encStart - src) -(4-1),
(uint32_t)(refEnd-refStartSW2), 4);
20,693         if (FoundAtPosition2!=NULL) {
20,694 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(4-1);
20,695         } else FoundAtPosition = NULL;
20,696             // Fallback to memmem(): ]
20,697         } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[0];
20,698 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
%p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
20,699         } else FoundAtPosition = NULL;
20,700 #endif
20,701
20,702     if (FoundAtPosition!=NULL) {
20,703
20,704
20,705 #ifdef Kaidanji
20,706 /*
20,707 if (LastSeenOffset_PseudoPointer[0]!=0) {
20,708     printf("LastSeenOffset_PseudoPointer[0]                : %s\n",_ui64toaKAZEcomma( (uint64_t)LastSeenOffset_PseudoPointer[0] , 11ToaDigits, 10));
//debugggggggggg
20,709     printf("%c%c%c%c%c = %c%c%c%c%c\n", *((char *)LastSeenOffset_PseudoPointer[0]+0),*((char *)LastSeenOffset_PseudoPointer[0]+1),*((char
*)LastSeenOffset_PseudoPointer[0]+2),*((char *)LastSeenOffset_PseudoPointer[0]+3),*((char *)LastSeenOffset_PseudoPointer[0]+4),
*((char*)(0+encStart)),*((char*)(1+encStart)),*((char*)(2+encStart)),*((char*)(3+encStart)),*((char*)(4+encStart))); //debuggggggggggg
20,710     LongestCommonPrefix = memcmp_CommonPrefixLength ( (char *)LastSeenOffset_PseudoPointer[0], encStart, 256);
20,711     printf("LongestCommonPrefix                : %s\n",_ui64toaKAZEcomma( LongestCommonPrefix , 11ToaDigits, 10)); //debuggggggggggg
20,712 }
20,713 */
20,714 #endif
20,715
20,716
20,717         *retMatch=4;
20,718         // The first four bits should be:
20,719                                     // 1110b = E
20,720         *retIndex=((refEnd-FoundAtPosition)<<4)&0xFFF0|0x000E; // xx ... x[00LL]
20,721         *ShortMediumLongOFFSET=2;
20,722         return;
20,723     }
20,724 }
20,725 } // B-TREE heuristic
20,726
20,727 goto skip64;
20,728 //48 6:4= 1.5 (16MB)
20,729 #if defined(BtreeHEURISTIC)
20,730     if ( LastSeenOffset_PseudoPointer[1] != 0 )
20,731 #endif
20,732     { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
20,733

```

```

20,734 if (refStartSW1a >= refStart) {
20,735
20,736 #ifndef BtreeHEURISTIC
20,737 //      FoundAtPosition = Railgun_BawBaw_reverse (refStartSW1a, encStart, (uint32_t)(refEnd-refStartSW1a), 6);
20,738 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW1a - src) - (uint32_t)(refEnd-refStartSW1a - 1), srcR + (srcSize - 1) - (encStart - src) - (6-1), (uint32_t)(refEnd-refStartSW1a), 6);
20,739 if (FoundAtPosition2!=NULL) {
20,740 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (6-1);
20,741 } else FoundAtPosition = NULL;
20,742 #else
20,743 if ( refStartSW1a <= (char *)LastSeenOffset_PseudoPointer[1] ) {
20,744 if ( refEnd - (char *)LastSeenOffset_PseudoPointer[1] < (6) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the first position of the LookAhead!
20,745 // Fallback to memmem(): [
20,746 //      FoundAtPosition = Railgun_BawBaw_reverse (refStartSW1a, encStart, (uint32_t)(refEnd-refStartSW1a), 6);
20,747 FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW1a - src) - (uint32_t)(refEnd-refStartSW1a - 1), srcR + (srcSize - 1) - (encStart - src) - (6-1), (uint32_t)(refEnd-refStartSW1a), 6);
20,748 if (FoundAtPosition2!=NULL) {
20,749 FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) - (6-1);
20,750 } else FoundAtPosition = NULL;
20,751 // Fallback to memmem(): ]
20,752 } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[1];
20,753 //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p: %p,%p\n", refStartSW3,refEnd, FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0]); //debugprint
20,754 } else FoundAtPosition = NULL;
20,755 #endif
20,756
20,757 if (FoundAtPosition!=NULL) {
20,758     *retMatch=6;
20,759     // The first four bits should be:
20,760
20,761     *retIndex=(((refEnd-FoundAtPosition)<<8)&0xFFFFF00)!0xF3; // xx ... x[LL00] // 0011b = 0x3; 11xx0011b, xx==11
20,762     *ShortMediumLongOFFSET=4;
20,763     return;
20,764 }
20,765 }
20,766 } // B-TREE heuristic
20,767 skip64:
20,768
20,769 //49 4:3= 1.3 (1MB)
20,770 /*
20,771 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) [
20,772 if (refStartHOT >= refStart)
20,773 if (refStartHOT < refEnd) {
20,774 FoundAtPosition = Railgun_Trollldom_64(refStartHOT, encStart, (uint32_t)(refEnd-refStartHOT), 4);
20,775 if (FoundAtPosition!=NULL) {
20,776     *retMatch=4;
20,777     // The first four bits should be:
20,778
20,779     *retIndex=(((refEnd-FoundAtPosition)<<4)&0xFFFFF0)!0x000D; // xx ... x[OOLL] // 1101b = D
20,780     *ShortMediumLongOFFSET=3;
20,781     return;
20,782 }
20,783 }
20,784 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) ]
20,785
20,786 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) [
20,787 if (refStartCOLDERbig >= refStart)

```

```

20,788 if (refStartCOLDERbig < refEnd) {
20,789 FoundAtPosition = Railgun_Trollldom_64(refStartCOLDERbig, encStart, (uint32_t)(refEnd-refStartCOLDERbig), 4);
20,790     if (FoundAtPosition!=NULL) {
20,791         *retMatch=4;
20,792         // The first four bits should be:
20,793                                     // 1101b = D
20,794         *retIndex=((refEnd-FoundAtPosition)<<4)&0xFFFFF0|0x000D; // xx ... x[OOLL]
20,795         *ShortMediumLongOffset=3;
20,796         return;
20,797     }
20,798 }
20,799 // Pre-emptive strike, matches should be sought close to the lookahead (cache-friendliness) ]
20,800 */
20,801 #if defined(BtreeHEURISTIC)
20,802     if ( LastSeenOffset_PseudoPointer[0] != 0 )
20,803 #endif
20,804     { // LastSeenOffset_PseudoPointer[0,1,2,3,4,5,6,7] is 4,6,8,10,12,14,16,18
20,805
20,806     if (refStartSW3 >= refStart) {
20,807
20,808     #ifndef BtreeHEURISTIC
20,809     // FoundAtPosition = Railgun_BawBaw_reverse (refStartSW3, encStart, (uint32_t)(refEnd-refStartSW3), 4);
20,810     FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW3 - src) -(uint32_t)(refEnd-refStartSW3 -1), srcR + (srcSize - 1) - (encStart - src) -(4-1),
20,811     (uint32_t)(refEnd-refStartSW3), 4);
20,812     if (FoundAtPosition2!=NULL) {
20,813     FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(4-1);
20,814     } else FoundAtPosition = NULL;
20,815     #else
20,816     if ( refStartSW3 <= (char *)LastSeenOffset_PseudoPointer[0] ) {
20,817         if ( refEnd - (char *)LastSeenOffset_PseudoPointer[0] < (4) ) { // RETURN TO memmem(), the match cannot overlap with 'refEnd' position - it is the
20,818         first position of the LookAhead!
20,819         // Fallback to memmem(): [
20,820         //GLOBAL_Railgun_INVOCATIONS_004_3_bytes++;
20,821         FoundAtPosition = Railgun_BawBaw_reverse (refStartSW3, encStart, (uint32_t)(refEnd-refStartSW3), 4);
20,822         FoundAtPosition2 = Railgun_Trollldom_64 (srcR + (srcSize - 1) - (refStartSW3 - src) -(uint32_t)(refEnd-refStartSW3 -1), srcR + (srcSize - 1) - (encStart - src) -(4-1),
20,823         (uint32_t)(refEnd-refStartSW3), 4);
20,824         if (FoundAtPosition2!=NULL) {
20,825         FoundAtPosition = src + (srcSize - 1) - (FoundAtPosition2 - srcR) -(4-1);
20,826         } else FoundAtPosition = NULL;
20,827         // Fallback to memmem(): ]
20,828         } else FoundAtPosition = (char *)LastSeenOffset_PseudoPointer[0];
20,829         //if (FoundAtPosition != (char *)LastSeenOffset_PseudoPointer[0]) printf("\nrefStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = %p-%p:
20,830         %p,%p\n", refStartSW3,refEnd, FoundAtPosition,(char *)LastSeenOffset_PseudoPointer[0]); //debugprint
20,831         } else FoundAtPosition = NULL;
20,832     #endif
20,833
20,834 // Note1: Looking how within 'Sherlock' the Btree values are not boundary compliant, we have to fall back to memmem(), see the dump below:
20,835 // Note2: Tsuyo was disabled by uncommenting 'goto Skip...'
20,836 // Compressing 3,714,387 bytes ...
20,837 //
20,838 // refStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = 00430020-00430047: 00000000,00430045
20,839 //
20,840 // refStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = 00430020-00430048: 00000000,00430046
20,841 // !; Each rotation means 64KB are encoded; Speed: 0,032,769 B/s; Done 1%; Compression Ratio: 1.82:1; Matches(16/24/48): 103/46/1; 128[+] long matches: 0; ETA: 0.00
20,842 days
20,843 // refStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = 00430020-00449475: 00000000,00449472
20,844 //

```

```

20,841 // refStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = 00430020-00449476: 00000000,00449473
20,842 //
20,843 // refStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = 00430020-00449477: 00000000,00449474
20,844 // /; Each rotation means 64KB are encoded; Speed: 0,065,536 B/s; Done 3%; Compression Ratio: 2.00:1; Matches(16/24/48): 268/63/1; 128[+] long matches: 0; ETA: 0.00
days
20,845 // refStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = 00430020-0045B144: 0045500C,0045B141
20,846 // \; Each rotation means 64KB are encoded; Speed: 0,054,613 B/s; Done 35%; Compression Ratio: 2.63:1; Matches(16/24/48): 7,710/702/6; 128[+] long matches: 0; ETA:
0.00 days
20,847 // refStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = 0047D423-0057D422: 00000000,0057D41F
20,848 //
20,849 // refStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = 0047D424-0057D423: 00000000,0057D420
20,850 // !; Each rotation means 64KB are encoded; Speed: 0,034,152 B/s; Done 65%; Compression Ratio: 2.81:1; Matches(16/24/48): 18,255/1,938/15; 128[+] long matches: 0; ETA:
0.00 days
20,851 // refStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = 0058AE76-0068AE75: 00000000,0068AE73
20,852 //
20,853 // refStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = 0058AE77-0068AE76: 00000000,0068AE74
20,854 // \; Each rotation means 64KB are encoded; Speed: 0,024,342 B/s; Done 91%; Compression Ratio: 2.88:1; Matches(16/24/48): 28,524/3,262/27; 128[+] long matches: 20;
ETA: 0.00 days
20,855 // refStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = 00679F88-00779F87: 00000000,00779F85
20,856 //
20,857 // refStartSW3-refEnd: FoundAtPosition, (char *)LastSeenOffset_PseudoPointer[0] = 00679F89-00779F88: 00000000,00779F86
20,858 // !; Each rotation means 64KB are encoded; Speed: 0,021,595 B/s; Done 100%; Compression Ratio: 2.89:1; Matches(16/24/48): 31,867/3,627/28; 128[+] long matches: 21;
ETA: 0.00 days
20,859 // NumberOfFullLiterals (lower-the-better): 487
20,860 // Tsuyo_HEURISTIC_APPLIED_thrice_back-to-back: 0
20,861 // NumberOf(Tiny)Matches[Micro]Window (4)[16B]: 1291
20,862 // NumberOfMatches[Bheema]Window [128GB window]: 23
20,863 // RAM-to-RAM performance: 21595 B/s.
20,864 // Compressed to 1,286,110 bytes.
20,865 // Source-file-Hash(FNV1A_YoshimitsuTRIAD) = 0x667c,8403
20,866 // Target-file-Hash(FNV1A_YoshimitsuTRIAD) = 0xec71,2979
20,867 // Decompressing 1,286,110 (being the compressed stream) bytes ...
20,868 // RAM-to-RAM performance: 442 MB/s.
20,869 // Verification (input and output sizes match) OK.
20,870 // Verification (input and output blocks match) OK.
20,871
20,872
20,873     if (FoundAtPosition!=NULL) {
20,874         *retMatch=4;
20,875         // The first four bits should be:
20,876
20,877         *retIndex=(((refEnd-FoundAtPosition)<<4)&0xFFFFF0)!0x000D; // xx ... x[00LL] // 1101b = D
20,878         *ShortMediumLongOFFSET=3;
20,879         return;
20,880     }
20,881 }
20,882 } // B-TREE heuristic
20,883
20,884 /*
20,885 // #35 3:2= 1.5 (256B)
20,886 if (refStartSW1c >= refStart) {
20,887     FoundAtPosition = Railgun_Baw_reverse (refStartSW1c, encStart, (uint32_t)(refEnd-refStartSW1c), 3);
20,888     if (FoundAtPosition!=NULL) {
20,889         *retMatch=3;
20,890         // The first four bits should be:
20,891         *retIndex=(((refEnd-FoundAtPosition)<<8)&0xFF00)!0x03; // xx ... x[LL00]
20,892         *ShortMediumLongOFFSET=2;
20,893         return;

```

```
20,894     }
20,895 }
20,896 */
20,897
20,898
20,899 #else
20,900 while(refStart < refEnd){
20,901     match=SlidingWindowVsLookAheadBuffer(refStart,refEnd,encStart,encEnd);
20,902     if(match > *retMatch){
20,903         *retMatch=match;
20,904         *retIndex=refEnd-refStart;
20,905     }
20,906     if(*retMatch >= Min_Match_BAILOUT_Length) break;
20,907     refStart++;
20,908 }
20,909 #endif
20,910 }
20,911
20,912 unsigned int SlidingWindowVsLookAheadBuffer( char* refStart, char* refEnd, char* encStart,char* encEnd){
20,913     int ret = 0;
20,914     while(refStart[ret] == encStart[ret]){
20,915         if(&refStart[ret] >= refEnd) break;
20,916         if(&encStart[ret] >= encEnd) break;
20,917         ret++;
20,918         if(ret >= Min_Match_BAILOUT_Length) break;
20,919     }
20,920     return ret;
20,921 }
20,922
20,923 uint64_t Compress(char* ret, char* src, char* srcR, uint64_t srcSize){
20,924     int FlagPlusONEbyte; // to tag the 2+1,3+1,4+1,5+1 offsets
20,925     int FlagPlusONEbyte1; // to tag the 2+1,3+1,4+1,5+1 offsets
20,926     uint64_t srcIndex=0;
20,927     uint64_t retIndex=0;
20,928     //unsigned int index=0; // Washigan
20,929     uint64_t index=0; // Ryuugan
20,930         //unsigned int index1=0;// Tsuyo
20,931         uint64_t index1=0;// Tsuyo
20,932     unsigned int match=0;
20,933     unsigned int match1=0;// Tsuyo
20,934     unsigned int notMatch=0;
20,935     // unsigned char* notMatchStart=NULL; // This line causes error when /TP is used, so next one is the fix:
20,936     char* notMatchStart=NULL;
20,937     char* refStart=NULL;
20,938         char* refStart1=NULL;
20,939     char* encEnd=NULL;
20,940         char* encEnd1=NULL;
20,941     int Melnitchka=0;
20,942     char *Auberge[4] = {"\\0","/\\0","-\\0","\\\\0"};
20,943     int ProgressIndicator;
20,944     FILE *fpratio;
20,945
20,946     unsigned int NumberOfFullLiterals=0;
20,947     int GLOBALmediumT=0;
20,948     int GLOBALshortT=0;
20,949     int GLOBALtinyT=0;
20,950     int GLOBAL24B=0;
20,951     int GLOBAL32B=0;
```

```

20,952 int GLOBAL48B=0;
20,953 int GLOBAL3=0;
20,954 int GLOBAL5bytes=0;
20,955 int GLOBAL6bytes=0;
20,956 unsigned int ShortMediumLongOFFSET=0;
20,957     unsigned int ShortMediumLongOFFSET1=0; // Tsuyo
20,958     int TsuyoHEURISTIC;
20,959     long SPD;
20,960     long long TsuyoHEURISTICAPPLIED=0;
20,961     unsigned int HowManyDittoTagsToEmit;
20,962     unsigned int HowManyDittoTagsToEmit1;
20,963
20,964 if ((fpratio = fopen("Nakamichi.ratio-graph.csv", "a")) == NULL) {
20,965     printf("Nakamichi: Can't write '%s' file.\n", "Nakamichi.ratio-graph.csv"); exit(13);
20,966 }
20,967
20,968 while(srcIndex < srcSize){
20,969     if(srcIndex>=REF_SIZE)
20,970         refStart=&src[srcIndex-REF_SIZE];
20,971     else
20,972         refStart=src;
20,973     if(srcIndex>=srcSize-ENC_SIZE)
20,974         encEnd=&src[srcSize];
20,975     else
20,976         encEnd=&src[srcIndex+ENC_SIZE];
20,977     // Fixing the stupid 'search-beyond-end' bug:
20,978     if(srcIndex+ENC_SIZE < srcSize) {
20,979         SearchIntoSlidingWindow(&HowManyDittoTagsToEmit, &ShortMediumLongOFFSET,&index,&match,refStart,&src[srcIndex],&src[srcIndex],encEnd, src, srcR,
srcSize);
20,980         FlagPlusONEbyte = 0;
20,981         if (ShortMediumLongOFFSET > 9999) {ShortMediumLongOFFSET=ShortMediumLongOFFSET-9999; FlagPlusONEbyte = 1;}
20,982         if (ShortMediumLongOFFSET > 3333) {ShortMediumLongOFFSET=ShortMediumLongOFFSET-3333; FlagPlusONEbyte = 2;}
20,983 // Large_traffic_log_file_of_a_popular_website_fp.log:
20,984 // 1: Size: 2,000,938; Speed: 15051 B/s; NumberOfFullLiterals (lower-the-better): 579; Matches(24/48): 526,996/142,715
20,985 // 2: Size: 2,000,557; Speed: 10839 B/s; NumberOfFullLiterals (lower-the-better): 579; Matches(24/48): 526,920/142,857
20,986 // 3: Size: 2,000,557; Speed: 8323 B/s; NumberOfFullLiterals (lower-the-better): 579; Matches(24/48): 526,920/142,857
20,987
20,988 #ifdef GREEDY
20,989 goto SKIPTsuyo;
20,990 #endif
20,991
20,992 for (TsuyoHEURISTIC=1; TsuyoHEURISTIC<=3; TsuyoHEURISTIC++) // This is 'Zato' i.e. double 'Tsuyo' - the lookahead heuristic looks one more char ahead. [One level
nested i.e. 1..2-1 due to tiny gains.]
20,993 {
20,994 // Tsuyo [
20,995 if (ShortMediumLongOFFSET != 0)
20,996 if (srcIndex+(1) < srcSize) {
20,997     if(srcIndex+(1)>=REF_SIZE)
20,998         refStart1=&src[srcIndex+(1)-REF_SIZE];
20,999     else
21,000         refStart1=src;
21,001     if(srcIndex+(1)>=srcSize-ENC_SIZE)
21,002         encEnd1=&src[srcSize];
21,003     else
21,004         encEnd1=&src[srcIndex+(1)+ENC_SIZE];
21,005     if(srcIndex+(1)+ENC_SIZE < srcSize) {
21,006
21,007

```



```

        SearchIntoSlidingWindow(&HowManyDittoTagsToEmit1,&ShortMediumLongOFFSET1,&index1,&match1,refStart1,&src[srcIndex+(1)],&src[srcIndex+(1)],encEnd1, src, srcR, srcSize);
21,008     FlagPlusONEbyte1 = 0;
21,009     if (ShortMediumLongOFFSET1 > 9999) {ShortMediumLongOFFSET1=ShortMediumLongOFFSET1-9999; FlagPlusONEbyte1 = 1;}
21,010     if (ShortMediumLongOFFSET1 > 3333) {ShortMediumLongOFFSET1=ShortMediumLongOFFSET1-3333; FlagPlusONEbyte1 = 2;}
21,011     if (ShortMediumLongOFFSET1 != 0)
21,012         if (match/((ShortMediumLongOFFSET+HowManyDittoTagsToEmit) * (1+1)) >= match1/((ShortMediumLongOFFSET1+HowManyDittoTagsToEmit1)) { // a brash
heuristic
21,013             break; // Exit the 'for' - simple heuristic, still not handling YES-NO-YES cases...
21,014         }
21,015     else {
21,016         if (TsuyoHEURISTIC==3) TsuyoHEURISTICAPPLIED++;
21,017         match=0; // Pretend nothing to find.
21,018
21,019 // 1of2
21,020         if(notMatch==0){
21,021             notMatchStart=&ret[retIndex];
21,022             retIndex++;
21,023         }
21,024         //else if (notMatch==(127-64-32)) {
21,025         else if (notMatch==(127-64-32-16 -(7)-1)) { //127-64-32-16 -(7)=8 lower one more to 7 - for 'ditto' tag
21,026             NumberOfFullLiterals++;
21,027             /*notMatchStart=(unsigned char)((127-64-32)<<3);
21,028             *notMatchStart=(unsigned char)((127-64-32-16 -(7)-1)<<(4));
21,029             *notMatchStart=(unsigned char)((127-64-32-16 -(7)-1)<<(4)) | 0x03; // Entag it as Literal
21,030             notMatch=0;
21,031             notMatchStart=&ret[retIndex];
21,032             retIndex++;
21,033         }
21,034         ret[retIndex]=src[srcIndex];
21,035         retIndex++;
21,036         notMatch++;
21,037         srcIndex++;
21,038         if ((srcIndex-1) % (1<<16) > srcIndex % (1<<16)) {
21,039             ProgressIndicator = (int)((srcIndex+1)*(double)100/(srcSize+1));
21,040             //SPD=((double)CLOCKS_PER_SEC*srcIndex/(double)(clock() - clocks1 + 1));
21,041             SPD=(double)srcIndex/(double)(time(NULL) - time1 + 1);
21,042             if (SPD > 99999999) SPD=99999999;
21,043             // printf("%s; Each rotation means 64KB are encoded; Speed: %s B/s; Done %d%%; Compression Ratio: %.2f:1; Matches(16/24/48): %s/%s/%s;
128[+] long matches: %s; ETA: %.2f days \r", Auberge[Melnitchka++], _ui64toaKAZEzerocomma(SPD, 11ToaDigits4, 10)+(26-9), ProgressIndicator, (double)(srcIndex) /
(double)(retIndex), _ui64toaKAZEcomma(GLOBAL24B, 11ToaDigits2, 10), _ui64toaKAZEcomma(GLOBAL32B, 11ToaDigits, 10), _ui64toaKAZEcomma(GLOBAL48B, 11ToaDigits3, 10),
_ui64toaKAZEcomma(GLOBAL5bytes, 11ToaDigits5, 10), ((double)(srcSize-srcIndex+1)/(SPD+1))/(3600*24));
21,044             printf("%s; Each rotation means 64KB are encoded; Speed: %s B/s; Done %d%%; Compression Ratio: %.2f:1; Matches(16/24/48): %s/%s/%s; 128[+]
long matches: %s; ETA: %.2f hours \r", Auberge[Melnitchka++], _ui64toaKAZEzerocomma(SPD, 11ToaDigits4, 10)+(26-9), ProgressIndicator, (double)(srcIndex) /
(double)(retIndex), _ui64toaKAZEcomma(GLOBAL24B, 11ToaDigits2, 10), _ui64toaKAZEcomma(GLOBAL32B, 11ToaDigits, 10), _ui64toaKAZEcomma(GLOBAL48B, 11ToaDigits3, 10),
_ui64toaKAZEcomma(GLOBAL5bytes, 11ToaDigits5, 10), ((double)(srcSize-srcIndex+1)/(SPD+1))/(3600));
21,045             Melnitchka = Melnitchka & 3; // 0 1 2 3: 00 01 10 11
21,046             fflush(stdout);
21,047         }
21,048 // 2of2
21,049         ShortMediumLongOFFSET=ShortMediumLongOFFSET1;
21,050         HowManyDittoTagsToEmit=HowManyDittoTagsToEmit1;
21,051         FlagPlusONEbyte = 0;
21,052         if (FlagPlusONEbyte1 == 1) FlagPlusONEbyte = 1; // Inherit the +1byte tag
21,053         if (FlagPlusONEbyte1 == 2) FlagPlusONEbyte = 2; // Inherit the +1byte tag
21,054         index=index1;
21,055         match=match1;
21,056
21,057     }

```

```

21,058
21,059     }
21,060 }
21,061 // Tsuyo ]
21,062 }
21,063 SKIPTsuyo:
21,064     if ( ShortMediumLongOFFSET==1 && match==4 ) GLOBALtinyT++;
21,065     if ( ShortMediumLongOFFSET==1 && match==8 ) GLOBALshortT++;
21,066     if ( ShortMediumLongOFFSET==1 && match==12 ) GLOBALmediumT++;
21,067     if ( match==16 ) GLOBAL24E++;
21,068     if ( match==24 ) GLOBAL32E++;
21,069     if ( match==48 ) GLOBAL48E++;
21,070     if ( match==3 ) GLOBAL3++;
21,071     //if ( ShortMediumLongOFFSET==5 ) GLOBAL5bytes++;
21,072     if ( match>=128 ) GLOBAL5bytes++;
21,073     if ( ShortMediumLongOFFSET==6 ) GLOBAL6bytes++;
21,074 }
21,075 else
21,076     match=0; // Nothing to find.
21,077 //if ( match<Min_Match_Length ) {
21,078 //if ( match<Min_Match_Length || match<8 ) {
21,079     if ( match==0 ) {
21,080         if(notMatch==0){
21,081             notMatchStart=&ret[retIndex];
21,082             retIndex++;
21,083         }
21,084         //else if (notMatch==(127-64-32)) {
21,085         else if (notMatch==(127-64-32-16 -(7)-1)) {
21,086             NumberOfFullLiterals++;
21,087             //*notMatchStart=(unsigned char)((127-64-32)<<3);
21,088             *notMatchStart=(unsigned char)((127-64-32-16 -(7)-1)<<(4));
21,089             *notMatchStart=(unsigned char)((127-64-32-16 -(7)-1)<<(4)) | 0x03; // Entag it as Literal
21,090             notMatch=0;
21,091             notMatchStart=&ret[retIndex];
21,092             retIndex++;
21,093         }
21,094         ret[retIndex]=src[srcIndex];
21,095         retIndex++;
21,096         notMatch++;
21,097         srcIndex++;
21,098         if ((srcIndex-1) % (1<<16) > srcIndex % (1<<16)) {
21,099             ProgressIndicator = (int)( (srcIndex+1)*(double)100/(srcSize+1) );
21,100             //SPD=((double)CLOCKS_PER_SEC*srcIndex/(double)(clock() - clocks1 + 1));
21,101             SPD=(double)srcIndex/(double)(time(NULL) - time1 + 1);
21,102             if (SPD > 99999999) SPD=99999999;
21,103             // printf("%s; Each rotation means 64KB are encoded; Speed: %s B/s; Done %d%%; Compression Ratio: %.2f:1; Matches(16/24/48): %s/%s/%s;
128[+] long matches: %s; ETA: %.2f days \r", Auberge[Melnitchka++], _ui64toaKAZEzerocomma(SPD, 11ToaDigits4, 10)+(26-9), ProgressIndicator, (double)(srcIndex) /
(double)(retIndex), _ui64toaKAZEcomma(GLOBAL24B, 11ToaDigits2, 10), _ui64toaKAZEcomma(GLOBAL32B, 11ToaDigits, 10), _ui64toaKAZEcomma(GLOBAL48B, 11ToaDigits3, 10),
_ui64toaKAZEcomma(GLOBAL5bytes, 11ToaDigits5, 10), ((double)(srcSize-srcIndex+1)/(SPD+1))/(3600*24) );
21,104             printf("%s; Each rotation means 64KB are encoded; Speed: %s B/s; Done %d%%; Compression Ratio: %.2f:1; Matches(16/24/48): %s/%s/%s; 128[+]
long matches: %s; ETA: %.2f hours \r", Auberge[Melnitchka++], _ui64toaKAZEzerocomma(SPD, 11ToaDigits4, 10)+(26-9), ProgressIndicator, (double)(srcIndex) /
(double)(retIndex), _ui64toaKAZEcomma(GLOBAL24B, 11ToaDigits2, 10), _ui64toaKAZEcomma(GLOBAL32B, 11ToaDigits, 10), _ui64toaKAZEcomma(GLOBAL48B, 11ToaDigits3, 10),
_ui64toaKAZEcomma(GLOBAL5bytes, 11ToaDigits5, 10), ((double)(srcSize-srcIndex+1)/(SPD+1))/(3600) );
21,105             Melnitchka = Melnitchka & 3; // 0 1 2 3: 00 01 10 11
21,106             fflush(stdout);
21,107             fprintf( fpratio, "%.2f,", (double)(srcIndex) / (double)(retIndex) );
21,108         }
21,109     } else {

```

```

21,110         if(notMatch > 0){
21,111             *notMatchStart=(unsigned char)((notMatch)<<(4));
21,112             *notMatchStart=(unsigned char)((notMatch)<<(4)) ! 0x03; // Entag it as Literal
21,113             notMatch=0;
21,114         }
21,115 // -----|
21,116 //      \ /
21,117
21,118         //ret[retIndex] = 0x80; // Assuming seventh/fifteenth bit is zero i.e. LONG MATCH i.e. Min_Match_BAILOUT_Length*4
21,119         //if ( match==Min_Match_BAILOUT_Length ) ret[retIndex] = 0xC0; // 8bit&7bit set, SHORT MATCH if seventh/fifteenth bit is not zero i.e.
Min_Match_BAILOUT_Length
21,120 //      / \
21,121 // -----|
21,122 /*
21,123         ret[retIndex] = 0x01; // Assuming seventh/fifteenth bit is zero i.e. LONG MATCH i.e. Min_Match_BAILOUT_Length*4
21,124         if ( match==Min_Match_BAILOUT_Length ) ret[retIndex] = 0x03; // 2bit&1bit set, LONG MATCH if 2bit is not zero i.e. Min_Match_BAILOUT_Length
21,125 */
21,126 // No need of above, during compression we demanded lowest 2bits to be not 00.
21,127         // 1bit+3bits+12bits:
21,128         //ret[retIndex] = ret[retIndex] ! ((match-Min_Match_Length)<<(4));
21,129         //ret[retIndex] = ret[retIndex] ! (((index-Min_Match_Length) & 0xF00)>>8);
21,130         // 1bit+1bit+14bits:
21,131         //ret[retIndex] = ret[retIndex] ! ((match-Min_Match_Length)<<(8-(LengthBITS+1))); // No need to set the matchlength
21,132 // The fragment below is outrageously ineffective - instead of 8bit&7bit I have to use the lower TWO bits i.e. 2bit&1bit as flags, thus in decompressing one WORD can
be fetched instead of two BYTE loads followed by SHR by 2.
21,133 // -----|
21,134 //      \ /
21,135         //ret[retIndex] = ret[retIndex] ! (((index-Min_Match_Length) & 0x3F00)>>8); // 2*4+8=14
21,136         //retIndex++;
21,137         //ret[retIndex] = (char)((index-Min_Match_Length) & 0x00FF);
21,138         //retIndex++;
21,139 //      / \
21,140 // -----|
21,141         // Now the situation is like LOW:HIGH i.e. FF:3F i.e. 0x3FFF, 16bit&15bit used as flags,
21,142         // should become LOW:HIGH i.e. FC:FF i.e. 0xFFFC, 2bit&1bit used as flags.
21,143 /*
21,144         ret[retIndex] = ret[retIndex] ! (((index-Min_Match_Length) & 0x00FF)<<2); // 6+8=14
21,145         //ret[retIndex] = ret[retIndex] ! (((index-Min_Match_Length) & 0x00FF)<<1); // 7+8=15
21,146         retIndex++;
21,147         ret[retIndex] = (char)(((index-Min_Match_Length) & 0x3FFF)>>6);
21,148         //ret[retIndex] = (char)(((index-Min_Match_Length) & 0x7FFF)>>7);
21,149         retIndex++;
21,150 */
21,151 // No need of above, during compression we demanded lowest 2bits to be not 00, use the full 16bits and get rid of the stupid '+/-' Min_Match_Length.
21,152         //if (index>0xFFFF) {printf ("\nFatal error: Overflow!\n"); exit(13);}
21,153         //memcpy(&ret[retIndex],&index,2+1); // copy lower 2 bytes
21,154         //retIndex++;
21,155         //retIndex++;
21,156         //retIndex++;
21,157
21,158         if (FlagPlusONEbyte == 2) {
21,159             ret[retIndex]=0x03;
21,160             memcpy(&ret[retIndex+1],&index,ShortMediumLongOFFSET-1);
21,161         } else if (FlagPlusONEbyte == 1) {
21,162             ret[retIndex]=0x93;
21,163             memcpy(&ret[retIndex+1],&index,ShortMediumLongOFFSET-1);
21,164         } else
21,165             memcpy(&ret[retIndex],&index,ShortMediumLongOFFSET);

```

```

21,166         retIndex = retIndex + ShortMediumLongOFFSET;
21,167         while (HowManyDittoTagsToEmit-->0) {
21,168             ret[retIndex++] = 0x83;
21,169         }
21,170 //         / \
21,171 // -----!
21,172         srcIndex += match;
21,173         if ((srcIndex - match) % (1 << 16) > srcIndex % (1 << 16)) {
21,174             ProgressIndicator = (int)((srcIndex + 1) * (double)100 / (srcSize + 1));
21,175             //SPD = (((double)CLOCK5_PER_SEC * srcIndex / (double)(clock() - clocks1 + 1)));
21,176             SPD = (double)srcIndex / (double)(time(NULL) - time1 + 1);
21,177             if (SPD > 99999999) SPD = 99999999;
21,178 //             printf("%s; Each rotation means 64KB are encoded; Speed: %s B/s; Done %d%%; Compression Ratio: %.2f:1; Matches(16/24/48): %s/%s/%s;
128[+] long matches: %s; ETA: %.2f days \r", Auberge[Melnitchka++], _ui64toaKAZEzerocomma(SPD, 11ToaDigits4, 10) + (26 - 9), ProgressIndicator, (double)(srcIndex) /
(double)(retIndex), _ui64toaKAZEcomma(GLOBAL24B, 11ToaDigits2, 10), _ui64toaKAZEcomma(GLOBAL32B, 11ToaDigits, 10), _ui64toaKAZEcomma(GLOBAL48B, 11ToaDigits3, 10),
_ui64toaKAZEcomma(GLOBAL5bytes, 11ToaDigits5, 10), ((double)(srcSize - srcIndex + 1) / (SPD + 1)) / (3600 * 24));
21,179             printf("%s; Each rotation means 64KB are encoded; Speed: %s B/s; Done %d%%; Compression Ratio: %.2f:1; Matches(16/24/48): %s/%s/%s; 128[+]
long matches: %s; ETA: %.2f hours \r", Auberge[Melnitchka++], _ui64toaKAZEzerocomma(SPD, 11ToaDigits4, 10) + (26 - 9), ProgressIndicator, (double)(srcIndex) /
(double)(retIndex), _ui64toaKAZEcomma(GLOBAL24B, 11ToaDigits2, 10), _ui64toaKAZEcomma(GLOBAL32B, 11ToaDigits, 10), _ui64toaKAZEcomma(GLOBAL48B, 11ToaDigits3, 10),
_ui64toaKAZEcomma(GLOBAL5bytes, 11ToaDigits5, 10), ((double)(srcSize - srcIndex + 1) / (SPD + 1)) / (3600));
21,180             Melnitchka = Melnitchka & 3; // 0 1 2 3: 00 01 10 11
21,181             fflush(stdout);
21,182             fprintf(fpratio, "%.2f, ", (double)(srcIndex) / (double)(retIndex));
21,183         }
21,184     }
21,185 }
21,186 if(notMatch > 0){
21,187     *notMatchStart = (unsigned char)((notMatch) << (4));
21,188     *notMatchStart = (unsigned char)((notMatch) << (4)) | 0x03; // Entag it as Literal
21,189 }
21,190 //SPD = (((double)CLOCK5_PER_SEC * srcIndex / (double)(clock() - clocks1 + 1)));
21,191 SPD = (double)srcIndex / (double)(time(NULL) - time1 + 1);
21,192 if (SPD > 99999999) SPD = 99999999;
21,193 //             printf("%s; Each rotation means 64KB are encoded; Speed: %s B/s; Done %d%%; Compression Ratio: %.2f:1; Matches(16/24/48): %s/%s/%s;
128[+] long matches: %s; ETA: %.2f days \n", Auberge[Melnitchka++], _ui64toaKAZEzerocomma(SPD, 11ToaDigits4, 10) + (26 - 9), 100, (double)(srcIndex) / (double)(retIndex),
_ui64toaKAZEcomma(GLOBAL24B, 11ToaDigits2, 10), _ui64toaKAZEcomma(GLOBAL32B, 11ToaDigits, 10), _ui64toaKAZEcomma(GLOBAL48B, 11ToaDigits3, 10), _ui64toaKAZEcomma(GLOBAL5bytes,
11ToaDigits5, 10), 0);
21,194             printf("%s; Each rotation means 64KB are encoded; Speed: %s B/s; Done %d%%; Compression Ratio: %.2f:1; Matches(16/24/48): %s/%s/%s; 128[+]
long matches: %s; ETA: %.2f hours \n", Auberge[Melnitchka++], _ui64toaKAZEzerocomma(SPD, 11ToaDigits4, 10) + (26 - 9), 100, (double)(srcIndex) / (double)(retIndex),
_ui64toaKAZEcomma(GLOBAL24B, 11ToaDigits2, 10), _ui64toaKAZEcomma(GLOBAL32B, 11ToaDigits, 10), _ui64toaKAZEcomma(GLOBAL48B, 11ToaDigits3, 10), _ui64toaKAZEcomma(GLOBAL5bytes,
11ToaDigits5, 10), 0.0); // 2020-Jan-11, it was 0 now 0.0 - huge number?!
21,195             fflush(stdout);
21,196             fprintf(fpratio, "%.2f\n", (double)(srcIndex) / (double)(retIndex));
21,197             printf("NumberOfFullLiterals (lower-the-better): %d\n", NumberOfFullLiterals);
21,198 //             printf("Tsuyo_HEURISTIC_APPLIED_thrice_back-to-back: %d\n", TsuyoHEURISTICAPPLIED);
21,199 //printf("NumberOf(Micro)Matches[Tiny]Window (%d)[%d]: %d\n", 3, 256, GLOBAL3);
21,200 //printf("NumberOf(Tiny)Matches[Micro]Window (%d)[%dB]: %d\n", 4, 16, GLOBALtinyT);
21,201 //printf("NumberOf(Short)Matches[Tiny]Window (%d): %d\n", 8, GLOBALshortT);
21,202 //printf("NumberOf(Medium)Matches[Tiny]Window (%d): %d\n", 12, GLOBALmediumT);
21,203 //printf("NumberOf(ExtraLong)Matches[Long]Window (%d): %d\n", 24, GLOBAL24B);
21,204 //printf("NumberOf(Huge)Matches[Long]Window (%d): %d\n", 48, GLOBAL48B);
21,205 //printf("NumberOfMatches[Bheema]Window [128GB window]: %d\n", GLOBAL6bytes);
21,206
21,207             fclose(fpratio);
21,208
21,209             return retIndex;
21,210 }
21,211

```

```

21,212 // Nakamichi_Ryuugan+ quick stats:
21,213 /*
21,214 08/27/2017 02:49 PM <DIR> ..
21,215 04/06/2017 08:44 PM 152,089 alice29.txt
21,216 08/27/2017 12:48 PM 56,526 alice29.txt.L17.LZSSE2
21,217 08/27/2017 12:48 PM 67,579 alice29.txt.Nakamichi
21,218 04/06/2017 08:44 PM 260,569 Fahrenheit_451_-_Ray_Bradbury.txt
21,219 08/27/2017 12:50 PM 100,387 Fahrenheit_451_-_Ray_Bradbury.txt.L17.LZSSE2
21,220 08/27/2017 12:50 PM 115,818 Fahrenheit_451_-_Ray_Bradbury.txt.Nakamichi
21,221 04/06/2017 08:44 PM 1,820,160 Fleurs_du_mal.tar
21,222 08/27/2017 10:58 AM 557,422 Fleurs_du_mal.tar.L17.LZSSE2
21,223 08/27/2017 10:58 AM 414,512 Fleurs_du_mal.tar.Nakamichi
21,224 04/06/2017 08:44 PM 8,798 IBM_Deep_Blue_vs_Kasparov.pgn
21,225 08/27/2017 12:49 PM 4,803 IBM_Deep_Blue_vs_Kasparov.pgn.L17.LZSSE2
21,226 08/27/2017 12:49 PM 5,153 IBM_Deep_Blue_vs_Kasparov.pgn.Nakamichi
21,227 08/27/2017 10:46 AM 167,936 Nakamichi_Ryuugan+_(1xQWORD+1xXMM)_Intel_15.0_32bit_SSE41.exe
21,228 04/06/2017 08:44 PM 3,680,256 Resident_Evil_(10-books).tar
21,229 08/27/2017 02:49 PM 1,343,243 Resident_Evil_(10-books).tar.L17.LZSSE2
21,230 08/27/2017 02:49 PM 1,289,461 Resident_Evil_(10-books).tar.Nakamichi
21,231 04/06/2017 08:44 PM 698,072 The_Death_Ship_-_B.Traven.pdf.txt
21,232 08/27/2017 12:55 PM 258,683 The_Death_Ship_-_B.Traven.pdf.txt.L17.LZSSE2
21,233 08/27/2017 12:55 PM 277,889 The_Death_Ship_-_B.Traven.pdf.txt.Nakamichi
21,234 07/02/2017 10:06 AM 4,445,260 The_Project_Gutenberg_EBook_of_The_King_James_Bible_kjv10.txt
21,235 08/27/2017 12:44 PM 1,390,894 The_Project_Gutenberg_EBook_of_The_King_James_Bible_kjv10.txt.L17.LZSSE2
21,236 08/27/2017 12:44 PM 1,356,427 The_Project_Gutenberg_EBook_of_The_King_James_Bible_kjv10.txt.Nakamichi
21,237 */
21,238
21,239 // 2021-Jan-12 [
21,240 uint64_t Decompress (char* ret, char* src, uint64_t srcSize) {
21,241 char* retLOCAL = ret;
21,242 char* srcLOCAL = src;
21,243 char* srcEndLOCAL = src+srcSize;
21,244 unsigned int DWORDtrio;
21,245 unsigned int DWORDtrio2;
21,246 unsigned int DWORDtrioDumbo;
21,247 unsigned int MatchLen;
21,248 unsigned int Gear;
21,249 uint64_t QWORDquartet;
21,250 unsigned char MatchLenLUT0[8]; // LookUpTable
21,251 // The first 3 bits in section 'Bheema' OOL, 00 is offset len; L=1 long, L=0 short match:
21,252 MatchLenLUT0[0]=18; // 000, 18:(5+1)=3
21,253 MatchLenLUT0[1]=18; // 100, 18:(4+1)=3.6
21,254 MatchLenLUT0[2]=18; // 010, 18:(3+1)=4.5
21,255 MatchLenLUT0[3]=18; // 110, 18:(2+1)=6
21,256 MatchLenLUT0[4]=28; // 001, 32:(5+1)=5.3
21,257 MatchLenLUT0[5]=28; // 101, 32:(4+1)=6.4
21,258 MatchLenLUT0[6]=28; // 011, 32:(3+1)=8
21,259 MatchLenLUT0[7]=28; // 111, 32:(2+1)=10.6
21,260 // Instead of using array consider using QWORD variable and shifting it side to side:
21,261 // uint64_t MatchLenLUT = 0x2030405012121424; // (MatchLenLUT<<<((7-OOL)<<3))>>>8*7
21,262 // QWORD: [1byte][2byte][3byte][4byte][5byte][6byte][7byte][8byte]
21,263 // LSB MSB
21,264 // 00=0 [OOL_5][ 2 ][ 3 ][ 4 ][ 5 ]
21,265 // 00=1 [OOL_5][ 2 ][ 3 ][ 4 ]
21,266 // 00=2 [OOL_5][ 2 ][ 3 ]
21,267 // 00=3 [OOL_5][ 2 ]
21,268 while (srcLOCAL < srcEndLOCAL) {
21,269 DWORDtrio = *(unsigned int*)srcLOCAL;

```

```

21,270 // MatchLen = (DWORDtrio&0x0C); // 0!4!8!12 // Commented and moved where it is used, 2020-Dec-17
21,271
21,272
21,273 // !1stLSB !2ndLSB !3rdLSB !
21,274 // -----
21,275 // !00!LL!xxxx!xxxxxxxx!xxxxxx!xx!
21,276 // -----
21,277 // [1bit 16bit 24bit]
21,278 // LL = 00b means Long MatchLength, (4-LL)<<2 or 16
21,279 // LL = 01b means Long MatchLength, (4-LL)<<2 or 12
21,280 // LL = 10b means Long MatchLength, (4-LL)<<2 or 8
21,281 // LL = 11b means Long MatchLength, (4-LL)<<2 or 4
21,282 // xxxx0011b Literals for xxxx in 1..15-7, Matches for 10..15 i.e. Sliding Window is 4*8-8=24 or 16MB
21,283 // 00 = 00b MatchOffset, 0xFFFFFFFF>>00, 4 bytes long i.e. Sliding Window is 4*8-LL-00=4*8-4=28 or 256MB
21,284 // 00 = 01b MatchOffset, 0xFFFFFFFF>>00, 3 bytes long i.e. Sliding Window is 3*8-LL-00=3*8-4=20 or 1MB
21,285 // 00 = 10b MatchOffset, 0xFFFFFFFF>>00, 2 bytes long i.e. Sliding Window is 2*8-LL-00=2*8-4=12 or 4KB
21,286 // 00 = 11b MatchOffset, 0xFFFFFFFF>>00, 1 byte long i.e. Sliding Window is 1*8-LL-00=1*8-4=4 or 16B
21,287
21,288 if ( (DWORDtrio & 0x0F) == 0x03 ) {
21,289 if ( ((DWORDtrio & 0xFF)>>4) <= 7 ) { // 9 not (actually used) used, only 1..7 for literals, 8 for 'ditto' tag
21,290 // Make next (nested branch) unbranched (https://godbolt.org/#):
21,291 // x = (symbolA <= symbolB) ? countA : 0;
21,292 // x = ((-(symbolA <= symbolB)) & countA);
21,293 if ( ((DWORDtrio & 0xFF)>>4) == 0 ) {
21,294 // srcLOCAL-2
21,295 //
21,296 // srcLOCAL
21,297 // QWORD: [1byte][2byte][3byte][4byte][5byte][6byte][7byte][8byte]
21,298 // LSB MSB
21,299 // 00=0 [00L_5][ 2 ][ 3 ][ 4 ][ 5 ]
21,300 // 00=1 [00L_5][ 2 ][ 3 ][ 4 ]
21,301 // 00=2 [00L_5][ 2 ][ 3 ]
21,302 // 00=3 [00L_5][ 2 ]
21,303 MatchLen = (DWORDtrio>>8)&0x7;
21,304 QWORDquartet = *(uint64_t*)(srcLOCAL+1- (3+(MatchLen&0x3)) ); // Safe since 5bytes backwards reads are less than MatchLengths used here.
21,305 QWORDquartet = QWORDquartet >> (((3+(MatchLen&0x3))<<3)+3);
21,306 srcLOCAL+= 5-(MatchLen&0x3)+1-(1);
21,307 ditto1:
21,308 #ifdef _N_GP
21,309 memcpy(retLOCAL, (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) ), 32);
21,310 #endif
21,311 #ifdef _N_XMM
21,312 NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) +16*(0) ), retLOCAL +16*(0));
21,313 NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) +16*(1) ), retLOCAL +16*(1));
21,314 #endif
21,315 #ifdef _N_YMM
21,316 SlowCopy256bit( (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) +32*(0) ), retLOCAL +32*(0));
21,317 #endif
21,318 retLOCAL+= MatchLenLUT0[MatchLen];
21,319 if (*(unsigned char*)(++srcLOCAL) == 0x83) goto ditto1;
21,320 // *(uint32_t*)(retLOCAL+4*(0)) = *(uint32_t*)(retLOCAL-((DWORDtrio&0xFFFF)>>8)+4*(0));
21,321 // retLOCAL+= 3;
21,322 // srcLOCAL+= 2;
21,323 } else {
21,324 // #ifdef _N_GP
21,325 // memcpy(retLOCAL, (const char *) ( (uint64_t)(srcLOCAL+1) ), 16);
21,326 // #endif
21,327 // #ifdef _N_XMM

```

```

21,328 //                NotSoSlowCopy128bit( (const char *) ( (uint64_t)(srcLOCAL+1) ), retLOCAL );
21,329 //                #endif
21,330                *(uint64_t*)(retLOCAL) = *(uint64_t*)((const char *) ( (uint64_t)(srcLOCAL+1) ));
21,331                retLOCAL+= ((DWORDtrio & 0xFF)>>4);
21,332                srcLOCAL+= ((DWORDtrio & 0xFF)>>4)+1;
21,333                }
21,334                } else if ( ((DWORDtrio & 0xFF)>>4) > 9 ) {
21,335                if ( ((DWORDtrio & 0xFF)>>4) < 0xF ) { // 2021-Mar-11
21,336                #ifdef _N_GP
21,337                memcpy(retLOCAL, (const char *) ( (uint64_t)(retLOCAL-(DWORDtrio>>8)) ), 16); // No need of DWORDtrio&0xFFFFFFFF
21,338                #endif
21,339                #ifdef _N_XMM
21,340                NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-(DWORDtrio>>8))+16*(0) ), retLOCAL +16*(0));
21,341                #endif
21,342                #ifdef _N_YMM
21,343                //NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-(DWORDtrio>>8))+16*(0) ), retLOCAL +16*(0));
21,344                SlowCopy256bit( (const char *) ( (uint64_t)(retLOCAL-(DWORDtrio>>8))+16*(0) ), retLOCAL +16*(0)); // 2020-Dec-17
21,345                #endif
21,346                retLOCAL+= (18 - ((DWORDtrio & 0xFF)>>4))<<1;
21,347                srcLOCAL+= 4; // 6!8!10!12!14!16 in 16MB window
21,348                } else { // 2021-Mar-11
21,349                memcpy(retLOCAL, (const char *) ( (uint64_t)(retLOCAL-(DWORDtrio>>8)) ), Sandokan);
21,350                retLOCAL+= Sandokan;
21,351                srcLOCAL+= 4;
21,352                }
21,353                } else { // i.e. 8!9
21,354                QWORDquartet = *(uint64_t*)(srcLOCAL+1- (3+(0)) ); // Safe since 5bytes backwards reads are less than MatchLengths used here.
21,355                QWORDquartet = QWORDquartet >> (((3+(0))<<3)+3-(3)); // Full 5x8=40bit i.e. 1TB
21,356                srcLOCAL+= 5-(0)+1-(1);
21,357                ditto2:
21,358                #ifdef _N_GP
21,359                memcpy(retLOCAL, (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) ), 64);
21,360                #endif
21,361                #ifdef _N_XMM
21,362                NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) +16*(0) ), retLOCAL +16*(0));
21,363                NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) +16*(1) ), retLOCAL +16*(1));
21,364                NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) +16*(2) ), retLOCAL +16*(2));
21,365                NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) +16*(3) ), retLOCAL +16*(3));
21,366                #endif
21,367                #ifdef _N_YMM
21,368                SlowCopy256bit( (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) +32*(0) ), retLOCAL +32*(0));
21,369                SlowCopy256bit( (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) +32*(1) ), retLOCAL +32*(1));
21,370                #endif
21,371                retLOCAL+= 64;
21,372                if (*(unsigned char*)(++srcLOCAL) == 0x83) goto ditto2;
21,373                }
21,374                } else if ( (DWORDtrio & 0x0f) == 0x0C ) {
21,375                // 0011LLGx xxxxxxxx xxxxxxxx, 11LL is matchlength, 2bytes window: 1+8=512B; 3bytes window: 1+8+8=128KB
21,376                Gear = (DWORDtrio>>3)&0x8; // 0/8 is 3/2 bytes respectively
21,377                /*(uint64_t*)(retLOCAL+8*(0)) = *(uint64_t*)((retLOCAL-( (DWORDtrio&((0xFFFFFFFF)>>Gear))>>7 ))+8*(0));
21,378                srcLOCAL+= 3-(Gear>>3) -(1); // ! -(1) is for 'ditto' tag
21,379                ditto4:
21,380                #ifdef _N_GP
21,381                memcpy(retLOCAL, (const char *) ( (uint64_t)(retLOCAL-((DWORDtrio&((0xFFFFFFFF)>>Gear))>>7)) ), 32);
21,382                #endif
21,383                #ifdef _N_XMM
21,384                NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-((DWORDtrio&((0xFFFFFFFF)>>Gear))>>7))+16*(0) ), retLOCAL +16*(0));
21,385                NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-((DWORDtrio&((0xFFFFFFFF)>>Gear))>>7))+16*(1) ), retLOCAL +16*(1));

```

```

21,386         #endif
21,387         #ifdef _N_YMM
21,388             SlowCopy256bit( (const char *) ( (uint64_t)(retLOCAL-((DWORDtrio&((0xFFFFFFFF)>>Gear))>>7))+32*(0) ), retLOCAL +32*(0));
21,389         #endif
21,390         retLOCAL+= (DWORDtrio & 0x3C)>>1; // (12*0)>>1=6 ! (12+16)>>1=14 ! (12+32)>>1=22 ! (12+48)>>1=30
21,391 if (*(unsigned char*)(++srcLOCAL) == 0x83) goto ditto4;
21,392     } else if ( (DWORDtrio & 0x03) == 0x00 ) {
21,393         MatchLen = (DWORDtrio&0x0C); // 0!4!8!12 // Commented and moved where it is used, 2020-Dec-17
21,394         // MatchLen 0!4!8 <<1 0!8!16
21,395         MatchLen=MatchLen<<1; // To avoid 'LEA'
21,396         srcLOCAL+= 4-(MatchLen>>3) -(1); // 24:2, 24:3, 24:4 ! -(1) is for 'ditto' tag
21,397 ditto3:
21,398         #ifdef _N_GP
21,399             *(uint64_t*)(retLOCAL+8*(0)) = *(uint64_t*)((retLOCAL-( (DWORDtrio&(0xFFFFFFFF)>>MatchLen))>>4 ))+8*(0));
21,400             *(uint64_t*)(retLOCAL+8*(1)) = *(uint64_t*)((retLOCAL-( (DWORDtrio&(0xFFFFFFFF)>>MatchLen))>>4 ))+8*(1));
21,401             *(uint64_t*)(retLOCAL+8*(2)) = *(uint64_t*)((retLOCAL-( (DWORDtrio&(0xFFFFFFFF)>>MatchLen))>>4 ))+8*(2));
21,402         #endif
21,403         #ifdef _N_XMM
21,404             *(uint64_t*)(retLOCAL+8*(0)) = *(uint64_t*)((retLOCAL-( (DWORDtrio&(0xFFFFFFFF)>>MatchLen))>>4 ))+8*(0));
21,405             NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-( (DWORDtrio&(0xFFFFFFFF)>>MatchLen))>>4 ))+8*(1) ), retLOCAL +8*(1));
21,406         #endif
21,407         #ifdef _N_YMM
21,408             SlowCopy256bit( (const char *) ( (uint64_t)(retLOCAL-( (DWORDtrio&(0xFFFFFFFF)>>MatchLen))>>4 )) ), retLOCAL);
21,409         #endif
21,410         retLOCAL+= 24;
21,411 if (*(unsigned char*)(++srcLOCAL) == 0x83) goto ditto3;
21,412     } else {
21,413         DWORDtrioDumbo = (DWORDtrio & 0x03)<<3; // To avoid 'LEA'
21,414         MatchLen = (DWORDtrio&0x0C); // 0!4!8!12 // Commented and moved where it is used, 2020-Dec-17
21,415         #ifdef _N_GP
21,416             memcpy(retLOCAL, (const char *) ( (uint64_t)(retLOCAL-(DWORDtrio&(0xFFFFFFFF)>>DWORDtrioDumbo))>>4) ), 16);
21,417         #endif
21,418         #ifdef _N_XMM
21,419             NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-(DWORDtrio&(0xFFFFFFFF)>>DWORDtrioDumbo))>>4) ), retLOCAL );
21,420         #endif
21,421         #ifdef _N_YMM
21,422             //NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-(DWORDtrio&(0xFFFFFFFF)>>DWORDtrioDumbo))>>4) ), retLOCAL );
21,423             SlowCopy256bit( (const char *) ( (uint64_t)(retLOCAL-(DWORDtrio&(0xFFFFFFFF)>>DWORDtrioDumbo))>>4) ), retLOCAL ); // 2020-Dec-17
21,424         #endif
21,425         retLOCAL+= 16-MatchLen;
21,426         srcLOCAL+= 4-(DWORDtrioDumbo>>3);
21,427     }
21,428 }
21,429 return (uint64_t)(retLOCAL - ret);
21,430 }
21,431
21,432
21,433 // 2020-Dec-18 [
21,434
21,435 uint64_t Decompress2020Dec18 (char* ret, char* src, uint64_t srcSize) {
21,436     char* retLOCAL = ret;
21,437     char* srcLOCAL = src;
21,438     char* srcEndLOCAL = src+srcSize;
21,439     unsigned int DWORDtrio;
21,440     unsigned int DWORDtrio2;
21,441     unsigned int DWORDtrioDumbo;
21,442     unsigned int MatchLen;
21,443     unsigned int Gear;

```



```

21,444 uint64_t QWORDquartet;
21,445 unsigned char MatchLenLUT0[8]; // LookUpTable
21,446 // The first 3 bits in section 'Bheema' OOL, 00 is offset len; L=1 long, L=0 short match:
21,447 MatchLenLUT0[0]=18; // 000, 18:(5+1)=3
21,448 MatchLenLUT0[1]=18; // 100, 18:(4+1)=3.6
21,449 MatchLenLUT0[2]=18; // 010, 18:(3+1)=4.5
21,450 MatchLenLUT0[3]=18; // 110, 18:(2+1)=6
21,451 MatchLenLUT0[4]=28; // 001, 32:(5+1)=5.3
21,452 MatchLenLUT0[5]=28; // 101, 32:(4+1)=6.4
21,453 MatchLenLUT0[6]=28; // 011, 32:(3+1)=8
21,454 MatchLenLUT0[7]=28; // 111, 32:(2+1)=10.6
21,455 // Instead of using array consider using QWORD variable and shifting it side to side:
21,456 // uint64_t MatchLenLUT = 0x2030405012121424; // (MatchLenLUT<<((7-OOL)<<3))>>8*7
21,457 // QWORD: [1byte][2byte][3byte][4byte][5byte][6byte][7byte][8byte]
21,458 //      LSB                                     MSB
21,459 // 00=0      [OOL_5][ 2 ][ 3 ][ 4 ][ 5 ]
21,460 // 00=1      [OOL_5][ 2 ][ 3 ][ 4 ]
21,461 // 00=2      [OOL_5][ 2 ][ 3 ]
21,462 // 00=3      [OOL_5][ 2 ]
21,463 while (srcLOCAL < srcEndLOCAL) {
21,464     DWORDtrio = *(unsigned int*)srcLOCAL;
21,465     //      MatchLen = (DWORDtrio&0x0C); // 0!4!8!12 // Commented and moved where it is used, 2020-Dec-17
21,466     /*
21,467     #ifndef _N_GP
21,468     #ifdef _N_prefetch_64
21,469         _mm_prefetch((char*)(srcLOCAL + 64), _MM_HINT_T0);
21,470     #endif
21,471     #ifdef _N_prefetch_128
21,472         _mm_prefetch((char*)(srcLOCAL + 64*2), _MM_HINT_T0);
21,473     #endif
21,474     #ifdef _N_prefetch_4096
21,475         _mm_prefetch((char*)(srcLOCAL + 64*64), _MM_HINT_T0);
21,476     #endif
21,477     #endif
21,478     */
21,479
21,480     // Sizewise/MatchLenWise priority:
21,481     /*#01 704:(5+1)+1+1+1+1+1+1+1+1=44      (1TB)
21,482     /*#02 320:(5+1)+1+1+1+1=      32      (1TB)
21,483     /*#03 256:(5+1)+1+1+1=      28.4      (1TB)
21,484     /*#04 192:(5+1)+1+1=      24      (1TB)
21,485     /*#05  90:2+1+1=      22.5      (512B)
21,486     /*#06  44:2+1=      22      (512B)
21,487     /*#07 120:3+1+1+1=      20      (128KB)
21,488     /*#08  60:2+1=      20      (512B)
21,489     /*#09  96:2+1+1+1=      19.2      (4KB)
21,490     /*#10 128:(5+1)+1=      18.2      (1TB)
21,491     /*#11  90:3+1+1=      18      (128KB)
21,492     /*#12  72:2+1+1=      18      (4KB)
21,493     /*#13  96:3+1+1+1=      16      (1MB)
21,494     /*#14  48:2+1=      16      (4KB)
21,495     /*#15  60:3+1=      15      (128KB)
21,496     /*#16  30:2=      15      (512B)
21,497     /*#17  72:3+1+1=      14.4      (1MB)
21,498     /*#18  56:(2+1)+1=      14      (8KB)
21,499     /*#19  96:4+1+1+1=      13.7      (256MB)
21,500     /*#20  72:4+1+1=      12      (256MB)
21,501     /*#21  48:3+1=      12      (1MB)

```

```

21,502 // #22 24:2= 12 (4KB)
21,503 // #23 12:1= 12 (16B)
21,504 // #24 56:(3+1)+1= 11.2 (2MB)
21,505 // #25 44:3+1= 11 (128KB)
21,506 // #26 22:2= 11 (512B)
21,507 // #27 64:(5+1)= 10.6 (1TB)
21,508 // #28 30:3= 10 (128KB)
21,509 // #29 48:4+1= 9.6 (256MB)
21,510 // #30 56:(4+1)+1= 9.3 (512MB)
21,511 // #31 28:(2+1)= 9.3 (8KB)
21,512 // #32 36:(2+1)+1= 9 (8KB)
21,513 // #33 56:(5+1)+1= 8 (128GB)
21,514 // #34 24:3= 8 (1MB)
21,515 // #35 16:2= 8 (4KB)
21,516 // #36 8:1= 8 (16B)
21,517 // #37 22:3= 7.3 (128KB)
21,518 // #38 36:(3+1)+1= 7.2 (2MB)
21,519 // #39 28:(3+1)= 7 (2MB)
21,520 // #40 14:2= 7 (512B)
21,521 // #41 36:(4+1)+1= 6 (512MB)
21,522 // #42 24:4= 6 (256MB)
21,523 // #43 18:(2+1)= 6 (8KB)
21,524 // #44 12:2= 6 (4KB)
21,525 // #45 28:(4+1)= 5.6 (512MB)
21,526 // #46 16:3= 5.3 (1MB)
21,527 // #47 36:(5+1)+1= 5.1 (128GB)
21,528 // #48 28:(5+1)= 4.6 (128GB)
21,529 // #49 14:3= 4.6 (128KB)
21,530 // #50 18:(3+1)= 4.5 (2MB)
21,531 // #51 16:4= 4 (16MB)F
21,532 // #52 12:3= 4 (1MB)
21,533 // #53 8:2= 4 (4KB)
21,534 // #54 4:1= 4 (16B)
21,535 // #55 18:(4+1)= 3.6 (512MB)
21,536 // #56 14:4= 3.5 (16MB)
21,537 // #57 18:(5+1)= 3 (128GB)
21,538 // #58 12:4= 3 (16MB)
21,539 // #59 6:2= 3 (512B)
21,540 // #60 8:3= 2.6 (1MB)
21,541 // #61 10:4= 2.5 (16MB)
21,542 // #62 8:4= 2 (16MB)
21,543 // #63 6:3= 2 (128KB)
21,544 // #64 4:2= 2 (4KB)
21,545 // #65 6:4= 1.5 (16MB)
21,546 // #66 4:3= 1.3 (1MB)
21,547
21,548 // !1stLSB !2ndLSB !3rdLSB !
21,549 // -----
21,550 // !00!LL!xxxx!xxxxxxxx!xxxxxx!xx!
21,551 // -----
21,552 // [1bit 16bit] 24bit]
21,553 // LL = 00b means Long MatchLength, (4-LL)<<2 or 16
21,554 // LL = 01b means Long MatchLength, (4-LL)<<2 or 12
21,555 // LL = 10b means Long MatchLength, (4-LL)<<2 or 8
21,556 // LL = 11b means Long MatchLength, (4-LL)<<2 or 4
21,557 // xxxx0011b Literals for xxxx in 1..15-7, Matches for 10..15 i.e. Sliding Window is 4*8-8=24 or 16MB
21,558 // 00 = 00b MatchOffset, 0xFFFFFFFF>>00, 4 bytes long i.e. Sliding Window is 4*8-LL-00=4*8-4=28 or 256MB
21,559 // 00 = 01b MatchOffset, 0xFFFFFFFF>>00, 3 bytes long i.e. Sliding Window is 3*8-LL-00=3*8-4=20 or 1MB

```

```

21,560 // 00 = 10b MatchOffset, 0xFFFFFFFF>>00, 2 bytes long i.e. Sliding Window is 2*8-LL-00=2*8-4=12 or 4KB
21,561 // 00 = 11b MatchOffset, 0xFFFFFFFF>>00, 1 byte long i.e. Sliding Window is 1*8-LL-00=1*8-4=4 or 16B
21,562
21,563 // switch ((DWORDtrio & 0x0F)) {
21,564 // case 0x03:
21,565 // break;
21,566 // case 0x0C:
21,567 // break;
21,568 // default:
21,569 // break;
21,570 // }
21,571
21,572 // if ( (DWORDtrio & 0x0F) == 0x03 ) {
21,573 // switch ((DWORDtrio & 0x0F)) { // 2020-Dec-18
21,574 // case 0x03: // 2020-Dec-18
21,575 // if ( ((DWORDtrio & 0xFF)>>4) <= 7 ) { // 9 not used, only 1..7 for literals, 8 for 'ditto' tag
21,576 // // Make next (nested branch) unbranched (https://godbolt.org/#):
21,577 // // x = (symbolA <= symbolB) ? countA : 0;
21,578 // // x = ((-(symbolA <= symbolB)) & countA);
21,579 // if ( ((DWORDtrio & 0xFF)>>4) == 0 ) {
21,580 // // srcLOCAL-2
21,581 // // srcLOCAL
21,582 // // srcLOCAL+1
21,583 // // QWORD: [1byte][2byte][3byte][4byte][5byte][6byte][7byte][8byte]
21,584 // // LSB MSB
21,585 // // 00=0 [00L_5][ 2 ][ 3 ][ 4 ][ 5 ]
21,586 // // 00=1 [00L_5][ 2 ][ 3 ][ 4 ]
21,587 // // 00=2 [00L_5][ 2 ][ 3 ]
21,588 // // 00=3 [00L_5][ 2 ]
21,589 MatchLen = (DWORDtrio>>8)&0x7;
21,590 QWORDquartet = *(uint64_t*)(srcLOCAL+1- (3+(MatchLen&0x3)) ); // Safe since 5bytes backwards reads are less than MatchLengths used here.
21,591 QWORDquartet = QWORDquartet >> (((3+(MatchLen&0x3))<<3)+3);
21,592 srcLOCAL+= 5-(MatchLen&0x3)+1-(1);
21,593 ditto1:
21,594 #ifdef _N_GP
21,595 memcpy(retLOCAL, (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) ), 32);
21,596 #endif
21,597 #ifdef _N_XMM
21,598 NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) +16*(0) ), retLOCAL +16*(0));
21,599 NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) +16*(1) ), retLOCAL +16*(1));
21,600 #endif
21,601 #ifdef _N_YMM
21,602 SlowCopy256bit( (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) +32*(0) ), retLOCAL +32*(0));
21,603 #endif
21,604
21,605 // 2021-Jan-10, careless bug in 'Stringology' [
21,606 if ( (uint64_t)(retLOCAL - ret) >= 0x26C021-2 )
21,607 {
21,608 // 1st iteration: 0x26C021-2 is 0026c01f \
21,609 // 2nd iteration: 0x26C021-1 is 0026c057 --> 57-1f=56 decimal
21,610 // 2nd iteration: 0x26C021-0 is 0026c057 /
21,611 /*
21,612 (uint64_t)(retLOCAL - ret) = 2539551
21,613 (uint64_t)(retLOCAL - ret) = 0026c01f
21,614 067CC03F: '46' 1f = 0x26C021-2
21,615 067CC040: '46' 20 = 0x26C021-1
21,616 067CC041: '30' 21 = 0x26C021-0
21,617 067CC042: '39'

```

Listing: Nakamichi Ryuugan-ditto-1TB btree.c; Last version: 2021-Aug-30; Font: MxPlus ToshibaTxL2 8x16.ttf; Downloadable at: www.sanmayce.com/Nakamichi/Kaidanji.zip

```

21,676 /*
21,677 printf("(uint64_t)(retLOCAL - ret) = %llu\n", (uint64_t)(retLOCAL - ret) );
21,678 printf("(uint64_t)(retLOCAL - ret) = %08x\n", (uint64_t)(retLOCAL - ret) );
21,679 printf("%p: '%02x'\n", retLOCAL+0, *(retLOCAL+0) );
21,680 printf("%p: '%02x'\n", retLOCAL+1, *(retLOCAL+1) );
21,681 printf("%p: '%02x'\n", retLOCAL+2, *(retLOCAL+2) );
21,682 printf("%p: '%02x'\n", retLOCAL+3, *(retLOCAL+3) );
21,683 printf("%p: '%02x'\n", retLOCAL+4, *(retLOCAL+4) );
21,684 printf(".....\n");
21,685 printf("%p: '%02x'\n", retLOCAL+27, *(retLOCAL+27) );
21,686 printf("The problematic/buggy LUT; MatchLen = %d; %d\n", MatchLenLUT0[MatchLen], MatchLen );
21,687 printf("5-(MatchLen&0x3)+1-(1)= %d\n", 5-(MatchLen&0x3)+1-(1) );
21,688 //exit(1);
21,689 */
21,690
21,691 }
21,692 // 2021-Jan-10, careless bug in 'Stringology' ]
21,693
21,694         retLOCAL+= MatchLenLUT0[MatchLen];
21,695 if (*(unsigned char*)(++srcLOCAL) == 0x83) goto ditto1;
21,696 //         *(uint32_t*)(retLOCAL+4*(0)) = *(uint32_t*)(retLOCAL-((DWORDtrio&0xFFFF)>>8)+4*(0));
21,697 //         retLOCAL+= 3;
21,698 //         srcLOCAL+= 2;
21,699     } else {
21,700 //         #ifdef _N_GP
21,701 //             memcpy(retLOCAL, (const char *) ( (uint64_t)(srcLOCAL+1) ), 16);
21,702 //         #endif
21,703 //         #ifdef _N_XMM
21,704 //             NotSoSlowCopy128bit( (const char *) ( (uint64_t)(srcLOCAL+1) ), retLOCAL );
21,705 //         #endif
21,706         *(uint64_t*)(retLOCAL) = *(uint64_t*)((const char *) ( (uint64_t)(srcLOCAL+1) ));
21,707         retLOCAL+= ((DWORDtrio & 0xFF)>>4);
21,708         srcLOCAL+= ((DWORDtrio & 0xFF)>>4)+1;
21,709     }
21,710 } else if ( ((DWORDtrio & 0xFF)>>4) > 9 ) {
21,711     #ifdef _N_GP
21,712         memcpy(retLOCAL, (const char *) ( (uint64_t)(retLOCAL-(DWORDtrio>>8)) ), 16); // No need of DWORDtrio&0xFFFFFFFF
21,713     #endif
21,714     #ifdef _N_XMM
21,715         NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-(DWORDtrio>>8))+16*(0) ), retLOCAL +16*(0));
21,716     #endif
21,717     #ifdef _N_YMM
21,718         //NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-(DWORDtrio>>8))+16*(0) ), retLOCAL +16*(0));
21,719         SlowCopy256bit( (const char *) ( (uint64_t)(retLOCAL-(DWORDtrio>>8))+16*(0) ), retLOCAL +16*(0)); // 2020-Dec-17
21,720     #endif
21,721     retLOCAL+= (18 - ((DWORDtrio & 0xFF)>>4))<<1;
21,722     srcLOCAL+= 4; // 6!8!10!12!14!16 in 16MB window
21,723 } else { // i.e. 8!9
21,724     QWORDquartet = *(uint64_t*)(srcLOCAL+1- (3+(0)) ); // Safe since 5bytes backwards reads are less than MatchLengths used here.
21,725     QWORDquartet = QWORDquartet >> (((3+(0))<<3)+3-(3)); // Full 5x8=40bit i.e. 1TB
21,726     srcLOCAL+= 5-(0)+1-(1);
21,727 ditto2:
21,728     #ifdef _N_GP
21,729         memcpy(retLOCAL, (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) ), 64);
21,730     #endif
21,731     #ifdef _N_XMM
21,732         NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) +16*(0) ), retLOCAL +16*(0));
21,733         NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) +16*(1) ), retLOCAL +16*(1));

```

```

21,734         NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) +16*(2) ), retLOCAL +16*(2));
21,735         NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) +16*(3) ), retLOCAL +16*(3));
21,736     #endif
21,737     #ifdef _N_YMM
21,738         SlowCopy256bit( (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) +32*(0) ), retLOCAL +32*(0));
21,739         SlowCopy256bit( (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) +32*(1) ), retLOCAL +32*(1));
21,740     #endif
21,741     retLOCAL+= 64;
21,742     if (*(unsigned char*)(++srcLOCAL) == 0x83) goto ditto2;
21,743     // To reduce compressed size further consider:
21,744     // if (uint64_t)(retLOCAL - ret)<<((1LL)<<(37-8)) srcLOCAL+= 5-(MatchLen&0x3)+1 -1; // gain for 512-MB files
21,745     // if (uint64_t)(retLOCAL - ret)<<((1LL)<<(37-8-8)) srcLOCAL+= 5-(MatchLen&0x3)+1 -1-1; // gain for 2-MB files
21,746 }
21,747     break; // 2020-Dec-18
21,748     case 0x0C: // 2020-Dec-18
21,749 // } else if ( (DWORDtrio & 0x0f) == 0x0C ) {
21,750 // // 0011LLGx xxxxxxxx xxxxxxxx, 11LL is matchlength, 2bytes window: 1+8=512B; 3bytes window: 1+8+8=128KB
21,751 Gear = (DWORDtrio>>3)&0x8; // 0/8 is 3/2 bytes respectively
21,752 //*(uint64_t*)(retLOCAL+8*(0)) = *(uint64_t*)((retLOCAL-( (DWORDtrio&((0xFFFFFFFF)>>Gear))>>7 ))+8*(0));
21,753 srcLOCAL+= 3-(Gear>>3) -1; // ! -1 is for 'ditto' tag
21,754 ditto4:
21,755     #ifdef _N_GP
21,756         memcpy(retLOCAL, (const char *) ( (uint64_t)(retLOCAL-((DWORDtrio&((0xFFFFFFFF)>>Gear))>>7)) ), 32);
21,757     #endif
21,758     #ifdef _N_XMM
21,759         NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-((DWORDtrio&((0xFFFFFFFF)>>Gear))>>7))+16*(0) ), retLOCAL +16*(0));
21,760         NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-((DWORDtrio&((0xFFFFFFFF)>>Gear))>>7))+16*(1) ), retLOCAL +16*(1));
21,761     #endif
21,762     #ifdef _N_YMM
21,763         SlowCopy256bit( (const char *) ( (uint64_t)(retLOCAL-((DWORDtrio&((0xFFFFFFFF)>>Gear))>>7))+32*(0) ), retLOCAL +32*(0));
21,764     #endif
21,765     retLOCAL+= (DWORDtrio & 0x3C)>>1; // (12+0)>>1=6 ! (12+16)>>1=14 ! (12+32)>>1=22 ! (12+48)>>1=30
21,766     if (*(unsigned char*)(++srcLOCAL) == 0x83) goto ditto4;
21,767     break; // 2020-Dec-18
21,768     default: // 2020-Dec-18
21,769 // } else if ( (DWORDtrio & 0x03) == 0x00 ) {
21,770 if ( (DWORDtrio & 0x03) == 0x00 ) {
21,771     MatchLen = (DWORDtrio&0x0C); // 0!4!8!12 // Commented and moved where it is used, 2020-Dec-17
21,772     // MatchLen 0!4!8 <<1 0!8!16
21,773     MatchLen=MatchLen<<1; // To avoid 'LEA'
21,774     srcLOCAL+= 4-(MatchLen>>3) -1; // 24:2, 24:3, 24:4 ! -1 is for 'ditto' tag
21,775 ditto3:
21,776     #ifdef _N_GP
21,777         *(uint64_t*)(retLOCAL+8*(0)) = *(uint64_t*)((retLOCAL-( (DWORDtrio&(0xFFFFFFFF)>>MatchLen))>>4 ))+8*(0));
21,778         *(uint64_t*)(retLOCAL+8*(1)) = *(uint64_t*)((retLOCAL-( (DWORDtrio&(0xFFFFFFFF)>>MatchLen))>>4 ))+8*(1));
21,779         *(uint64_t*)(retLOCAL+8*(2)) = *(uint64_t*)((retLOCAL-( (DWORDtrio&(0xFFFFFFFF)>>MatchLen))>>4 ))+8*(2));
21,780     #endif
21,781     #ifdef _N_XMM
21,782         *(uint64_t*)(retLOCAL+8*(0)) = *(uint64_t*)((retLOCAL-( (DWORDtrio&(0xFFFFFFFF)>>MatchLen))>>4 ))+8*(0));
21,783         NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-( (DWORDtrio&(0xFFFFFFFF)>>MatchLen))>>4 ))+8*(1) ), retLOCAL +8*(1));
21,784     #endif
21,785     #ifdef _N_YMM
21,786         SlowCopy256bit( (const char *) ( (uint64_t)(retLOCAL-( (DWORDtrio&(0xFFFFFFFF)>>MatchLen))>>4 )) ), retLOCAL);
21,787     #endif
21,788     retLOCAL+= 24;
21,789     if (*(unsigned char*)(++srcLOCAL) == 0x83) goto ditto3;
21,790 } else {
21,791     DWORDtrioDumbo = (DWORDtrio & 0x03)<<3; // To avoid 'LEA'

```

```

21,792 MatchLen = (DWORDtrio&0x0C); // 0!4!8!12 // Commented and moved where it is used, 2020-Dec-17
21,793 #ifdef _N_GP
21,794 memcpy(retLOCAL, (const char *) ( (uint64_t)(retLOCAL-((DWORDtrio&(0xFFFFFFFF>>DWORDtrioDumbo))>>4)) ), 16);
21,795 #endif
21,796 #ifdef _N_XMM
21,797 NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-((DWORDtrio&(0xFFFFFFFF>>DWORDtrioDumbo))>>4)) ), retLOCAL );
21,798 #endif
21,799 #ifdef _N_YMM
21,800 //NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-((DWORDtrio&(0xFFFFFFFF>>DWORDtrioDumbo))>>4)) ), retLOCAL );
21,801 SlowCopy256bit( (const char *) ( (uint64_t)(retLOCAL-((DWORDtrio&(0xFFFFFFFF>>DWORDtrioDumbo))>>4)) ), retLOCAL ); // 2020-Dec-17
21,802 #endif
21,803 retLOCAL+= 16-MatchLen;
21,804 srcLOCAL+= 4-(DWORDtrioDumbo>>3);
21,805 }
21,806 break; // 2020-Dec-18
21,807 } // 2020-Dec-18
21,808 }
21,809 return (uint64_t)(retLOCAL - ret);
21,810 }
21,811
21,812 // 2020-Dec-18 ]
21,813
21,814 // 2020-Dec-19, the GCC 7.3.0 Assembly:
21,815 // gcc -O3 -m64 -S -mavx2 -fomit-frame-pointer Nakamichi_Ryuugan-ditto-1TB_btree.c -o Nakamichi_DD_LITE_GCC_64bit_plus-320MB-buffer_YMM.asm -D_N_YMM -D_N_prefetch_4096
-D_N_alone -D_N_HIGH_PRIORITY -DHashInBITS=24 -DHashChunkSizeInBITS=24 -DRAMpoolInKB=5120 -DBtreeHEURISTIC -D_WIN32_ENVIRONMENT_ -DLongestLineInclusive=128 -
DSpeedUpBuilding=320 -DLITE -D_NDD
21,816 /*
21,817 .seh_proc Decompress
21,818 Decompress:
21,819 pushq %r15
21,820 .seh_pushreg %r15
21,821 pushq %r14
21,822 .seh_pushreg %r14
21,823 pushq %r13
21,824 .seh_pushreg %r13
21,825 pushq %r12
21,826 .seh_pushreg %r12
21,827 pushq %rbp
21,828 .seh_pushreg %rbp
21,829 pushq %rdi
21,830 .seh_pushreg %rdi
21,831 pushq %rsi
21,832 .seh_pushreg %rsi
21,833 pushq %rbx
21,834 .seh_pushreg %rbx
21,835 subq $24, %rsp
21,836 .seh_stackalloc 24
21,837 .seh_endprologue
21,838 movabsq $2025524839297716754, %rdi
21,839 addq %rdx, %r8
21,840 movq %rcx, %rax
21,841 movq %rdi, 8(%rsp)
21,842 movq %rcx, %r12
21,843 cmpq %r8, %rdx
21,844 jnb .L2997
21,845 movl $-1, %ebx
21,846 movl $16, %ebp
21,847 movl $4, %edi

```

```
21,848 movl    $3, %r13d
21,849 movl    $2, %r11d
21,850 movl    $16777215, %r10d
21,851 .p2align 4,,10
21,852 .L2996:
21,853 movl    (%rdx), %r9d
21,854 movl    %r9d, %ecx
21,855 andl    $15, %ecx
21,856 cmpl    $3, %ecx
21,857 je      .L2985
21,858 cmpl    $12, %ecx
21,859 jne     .L3005
21,860 movl    %r9d, %ecx
21,861 movl    %r11d, %r15d
21,862 shrl    $3, %ecx
21,863 andl    $8, %ecx
21,864 movl    %ecx, %esi
21,865 shrl    $3, %esi
21,866 subl    %esi, %r15d
21,867 movl    %r10d, %esi
21,868 sarl    %cl, %esi
21,869 addq    %r15, %rdx
21,870 movl    %esi, %ecx
21,871 andl    %r9d, %ecx
21,872 shrl    %r9d
21,873 shrl    $7, %ecx
21,874 andl    $30, %r9d
21,875 negq    %rcx
21,876 .p2align 4,,10
21,877 .L2993:
21,878 vmovdqu (%rax,%rcx), %ymm0
21,879 addq    $1, %rdx
21,880 vmovdqu %ymm0, (%rax)
21,881 addq    %r9, %rax
21,882 cmpb    $-125, (%rdx)
21,883 je      .L2993
21,884 cmpq    %r8, %rdx
21,885 jb      .L2996
21,886 .L3007:
21,887 subq    %r12, %rax
21,888 vzeroupper
21,889 .L3004:
21,890 addq    $24, %rsp
21,891 popq    %rbx
21,892 popq    %rsi
21,893 popq    %rdi
21,894 popq    %rbp
21,895 popq    %r12
21,896 popq    %r13
21,897 popq    %r14
21,898 popq    %r15
21,899 ret
21,900 .p2align 4,,10
21,901 .L3005:
21,902 testb    $3, %r9b
21,903 je      .L3006
21,904 leal    0(%r9,8), %ecx
21,905 movl    %ebx, %esi
```



```
21,906 movq    %rax, %r15
21,907 andl    $24, %ecx
21,908 shr1    %cl, %esi
21,909 shr1    $3, %ecx
21,910 andl    %r9d, %esi
21,911 andl    $12, %r9d
21,912 shr1    $4, %esi
21,913 subq    %rsi, %r15
21,914 movl    %ebp, %esi
21,915 subl    %r9d, %esi
21,916 vmovdqu (%r15), %ymm0
21,917 movl    %esi, %r9d
21,918 movl    %edi, %esi
21,919 subl    %ecx, %esi
21,920 vmovdqu %ymm0, (%rax)
21,921 addq    %r9, %rax
21,922 addq    %rsi, %rdx
21,923 cmpq    %r8, %rdx
21,924 jb     .L2996
21,925 jmp    .L3007
21,926 .p2align 4,,10
21,927 .L2985:
21,928 movl    %r9d, %ecx
21,929 shr1    $4, %ecx
21,930 andl    $15, %ecx
21,931 cmpl    $7, %ecx
21,932 ja     .L2987
21,933 testl   %ecx, %ecx
21,934 je     .L3008
21,935 movq    1(%rdx), %r9
21,936 movq    %r9, (%rax)
21,937 movl    %ecx, %r9d
21,938 addl    $1, %ecx
21,939 addq    %rcx, %rdx
21,940 addq    %r9, %rax
21,941 cmpq    %r8, %rdx
21,942 jb     .L2996
21,943 jmp    .L3007
21,944 .p2align 4,,10
21,945 .L2987:
21,946 cmpl    $9, %ecx
21,947 jbe     .L2991
21,948 movq    %rax, %rsi
21,949 shr1    $8, %r9d
21,950 addq    $4, %rdx
21,951 subq    %r9, %rsi
21,952 vmovdqu (%rsi), %ymm0
21,953 movl    $18, %esi
21,954 subl    %ecx, %esi
21,955 movl    %esi, %ecx
21,956 vmovdqu %ymm0, (%rax)
21,957 addl    %ecx, %ecx
21,958 addq    %rcx, %rax
21,959 cmpq    %r8, %rdx
21,960 jb     .L2996
21,961 jmp    .L3007
21,962 .p2align 4,,10
21,963 .L3006:
```

```
21,964 leal    (%r9,%r9), %ecx
21,965 movl    %r13d, %r15d
21,966 andl    $24, %ecx
21,967 movl    %ecx, %esi
21,968 shrl    $3, %esi
21,969 subl    %esi, %r15d
21,970 movl    %ebx, %esi
21,971 shrl    %cl, %esi
21,972 addq    %r15, %rdx
21,973 movl    %esi, %ecx
21,974 andl    %r9d, %ecx
21,975 shrl    $4, %ecx
21,976 negq    %rcx
21,977 .p2align 4,,10
21,978 .L2995:
21,979 vmovdqu (%rax,%rcx), %ymm0
21,980 addq    $1, %rdx
21,981 addq    $24, %rax
21,982 vmovdqu %ymm0, -24(%rax)
21,983 cmpb    $-125, (%rdx)
21,984 je      .L2995
21,985 cmpq    %r8, %rdx
21,986 jb      .L2996
21,987 jmp     .L3007
21,988 .p2align 4,,10
21,989 .L3008:
21,990 shrl    $8, %r9d
21,991 movl    $1, %r14d
21,992 movl    %r9d, %r15d
21,993 andl    $7, %r9d
21,994 andl    $3, %r15d
21,995 movzbl 8(%rsp,%r9), %r9d
21,996 leal    3(%r15), %esi
21,997 subq    %rsi, %r14
21,998 movq    %rsi, %rcx
21,999 movq    (%rdx,%r14), %rsi
22,000 leal    3(%rcx,8), %ecx
22,001 movl    $5, %r14d
22,002 subl    %r15d, %r14d
22,003 shrq    %cl, %rsi
22,004 addq    %r14, %rdx
22,005 movq    %rsi, %rcx
22,006 negq    %rcx
22,007 .p2align 4,,10
22,008 .L2989:
22,009 vmovdqu (%rax,%rcx), %ymm0
22,010 addq    $1, %rdx
22,011 vmovdqu %ymm0, (%rax)
22,012 addq    %r9, %rax
22,013 cmpb    $-125, (%rdx)
22,014 je      .L2989
22,015 cmpq    %r8, %rdx
22,016 jb      .L2996
22,017 jmp     .L3007
22,018 .p2align 4,,10
22,019 .L2991:
22,020 movq    -2(%rdx), %rcx
22,021 movq    %rax, %rsi
```

```

22,022 addq    $5, %rdx
22,023 shrq    $24, %rcx
22,024 subq    %rcx, %rsi
22,025 movq    %rsi, %rcx
22,026 addq    $32, %rcx
22,027 .p2align 4,,10
22,028 .L2992:
22,029 vmovdqu -32(%rcx), %ymm0
22,030 addq    $1, %rdx
22,031 addq    $64, %rax
22,032 addq    $64, %rcx
22,033 vmovdqu %ymm0, -64(%rax)
22,034 vmovdqu -64(%rcx), %ymm0
22,035 vmovdqu %ymm0, -32(%rax)
22,036 cmpb    $-125, (%rdx)
22,037 je      .L2992
22,038 cmpq    %r8, %rdx
22,039 jb      .L2996
22,040 jmp      .L3007
22,041 .p2align 4,,10
22,042 .L2997:
22,043 xorl    %eax, %eax
22,044 jmp      .L3004
22,045 .seh_endproc
22,046 */
22,047
22,048 // 2020-Dec-18, the cyclical function size: (0023c + 4) - 00046 = 506 bytes
22,049 // For a long time I knew that 'switch' statement makes different things than the 'if' counterparts but was stubborn not to use it, kinda stupid me.
22,050 // The 'if' counterpart is 2020-Dec-17 - further below - the end of the Assembly tells how more fragmented the 'if' is - not good.
22,051 /*
22,052 ; 18 branches [
22,053 ; unconditional: 6 'jmp'
22,054 000c0 e9 e6 00 00 00 jmp .B2.21
22,055 00101 e9 a5 00 00 00 jmp .B2.21
22,056 0015d eb 4c jmp .B2.21
22,057 001d4 eb d5 jmp .B2.21
22,058 001f8 eb b1 jmp .B2.21
22,059 00237 e9 6f ff ff ff jmp .B2.21
22,060 ; conditional: 5 'je'
22,061 000be 74 e6 je .B2.7
22,062 000ff 74 da je .B2.11
22,063 00109 74 54 je .B2.18
22,064 0015b 74 e5 je .B2.16
22,065 001a9 74 e6 je .B2.19
22,066 ; conditional: 3 'jne'
22,067 00 jne .B2.13
22,068 00 jne .B2.24
22,069 00 jne .B2.26
22,070 ; conditional: 3 'ja'
22,071 00 jae .B2.23
22,072 00064 77 5f ja .B2.9
22,073 00 ja .B2.25
22,074 ; conditional: 1 'jb'
22,075 ff jb .B2.3
22,076 ; ]
22,077 */
22,078 /*
22,079 ; mark_description "Intel(R) C++ Compiler XE for applications running on Intel(R) 64, Version 15.0.0.108 Build 20140726";

```

```
22,080 ; mark_description "-TP -O3 -arch:CORE-AVX2 -D_N_YMM -D_N_prefetch_4096 -D_N_HIGH_PRIORITY -D_icl_mumbo_jumbo -Facs -DHashInBIT";
22,081 ; mark_description "S=24 -DHashChunkSizeInBITS=24 -DRAMpoolInKB=5120 -DBtreeHEURISTIC -D_WIN32_ENVIRONMENT -DLongestLineInclusi";
22,082 ; mark_description "ve=128 -DSpeedUpBuilding=320 -D_NDD -DLITE -D_N_alone";
22,083
22,084 ?Decompress@@YA_KPEAD0_K0Z      PROC
22,085 ; parameter 1: rcx
22,086 ; parameter 2: rdx
22,087 ; parameter 3: r8
22,088 .B2.1::
22,089 000000 48 83 ec 38      sub rsp, 56
22,090 000004 4c 03 c2      add r8, rdx
22,091 000007 41 b1 12      mov r9b, 18
22,092 00000a 41 b2 1c      mov r10b, 28
22,093 00000d 48 89 c8      mov rax, rcx
22,094 000010 44 88 4c 24 28    mov BYTE PTR [40+rsp], r9b
22,095 000015 44 88 4c 24 29    mov BYTE PTR [41+rsp], r9b
22,096 00001a 44 88 4c 24 2a    mov BYTE PTR [42+rsp], r9b
22,097 00001f 44 88 4c 24 2b    mov BYTE PTR [43+rsp], r9b
22,098 000024 44 88 54 24 2c    mov BYTE PTR [44+rsp], r10b
22,099 000029 44 88 54 24 2d    mov BYTE PTR [45+rsp], r10b
22,100 00002e 44 88 54 24 2e    mov BYTE PTR [46+rsp], r10b
22,101 000033 44 88 54 24 2f    mov BYTE PTR [47+rsp], r10b
22,102 000038 49 3b d0      cmp rdx, r8
22,103 00003b 0f 83 78 01 00    jae .B2.23
22,104 00
22,105
22,106 .B2.2::
22,107 000041 48 89 6c 24 20    mov QWORD PTR [32+rsp], rbp
22,108
22,109 .B2.3::
22,110 000046 44 8b 12      mov r10d, DWORD PTR [rdx]
22,111 000049 44 89 d5      mov ebp, r10d
22,112 00004c 83 e5 0f      and ebp, 15
22,113 00004f 83 fd 03      cmp ebp, 3
22,114 000052 0f 85 ae 00 00    jne .B2.13
22,115 00
22,116
22,117 .B2.4::
22,118 000058 45 0f b6 ca      movzx r9d, r10b
22,119 00005c 41 c1 e9 04      shr r9d, 4
22,120 000060 41 83 f9 07      cmp r9d, 7
22,121 000064 77 5f          ja .B2.9
22,122
22,123 .B2.5::
22,124 000066 45 85 c9      test r9d, r9d
22,125 000069 0f 85 55 01 00    jne .B2.24
22,126 00
22,127
22,128 .B2.6::
22,129 00006f 41 c1 ea 08      shr r10d, 8
22,130 000073 49 83 e2 07      and r10, 7
22,131 000077 45 89 d1      mov r9d, r10d
22,132 00007a 41 83 e1 03      and r9d, 3
22,133 00007e 41 8d 69 03      lea ebp, DWORD PTR [3+r9]
22,134 000082 46 8d 1c cd 1b    lea r11d, DWORD PTR [27+r9*8]
22,135 00 00 00      neg rbp
22,136 00008a 48 f7 dd      neg r9d
22,137 00008d 41 f7 d9
```

```
22,138 00090 48 03 ea      add rbp, rdx
22,139 00093 41 83 c1 05     add r9d, 5
22,140 00097 49 03 d1       add rdx, r9
22,141 0009a c4 e2 a3 f7 6d
22,142 01                shrx rbp, QWORD PTR [1+rbp], r11
22,143 000a0 46 0f b6 4c 14
22,144 28                movzx r9d, BYTE PTR [40+rsp+r10]
22,145
22,146 .B2.7::
22,147 000a6 49 89 c2       mov r10, rax
22,148 000a9 48 ff c2       inc rdx
22,149 000ac 4c 2b d5       sub r10, rbp
22,150 000af c4 c1 7e 6f 02 vmovdqu ymm0, YMMWORD PTR [r10]
22,151 000b4 c5 fe 7f 00     vmovdqu YMMWORD PTR [rax], ymm0
22,152 000b8 49 03 c1       add rax, r9
22,153 000bb 80 3a 83       cmp BYTE PTR [rdx], -125
22,154 000be 74 e6         je .B2.7
22,155 000c0 e9 e6 00 00 00 jmp .B2.21
22,156
22,157 .B2.9::
22,158 000c5 41 83 f9 09     cmp r9d, 9
22,159 000c9 0f 87 07 01 00
22,160 00                ja .B2.25
22,161
22,162 .B2.10::
22,163 000cf 48 8b 6a fe     mov rbp, QWORD PTR [-2+rdx]
22,164 000d3 48 83 c2 05     add rdx, 5
22,165 000d7 48 c1 ed 18     shr rbp, 24
22,166
22,167 .B2.11::
22,168 000db 49 89 c1       mov r9, rax
22,169 000de 48 ff c2       inc rdx
22,170 000e1 4c 2b cd       sub r9, rbp
22,171 000e4 c4 c1 7e 6f 01 vmovdqu ymm0, YMMWORD PTR [r9]
22,172 000e9 c5 fe 7f 00     vmovdqu YMMWORD PTR [rax], ymm0
22,173 000ed c4 c1 7e 6f 49
22,174 20                vmovdqu ymm1, YMMWORD PTR [32+r9]
22,175 000f3 c5 fe 7f 48 20 vmovdqu YMMWORD PTR [32+rax], ymm1
22,176 000f8 48 83 c0 40     add rax, 64
22,177 000fc 80 3a 83       cmp BYTE PTR [rdx], -125
22,178 000ff 74 da         je .B2.11
22,179 00101 e9 a5 00 00 00 jmp .B2.21
22,180
22,181 .B2.13::
22,182 00106 83 fd 0c       cmp ebp, 12
22,183 00109 74 54         je .B2.18
22,184
22,185 .B2.14::
22,186 0010b 44 89 d5       mov ebp, r10d
22,187 0010e 83 e5 03       and ebp, 3
22,188 00111 0f 85 e3 00 00
22,189 00                jne .B2.26
22,190
22,191 .B2.15::
22,192 00117 44 89 d5       mov ebp, r10d
22,193 0011a 41 bb ff ff ff
22,194 ff                mov r11d, -1
22,195 00120 83 e5 0c       and ebp, 12
```

```

22,196 00123 03 ed      add ebp, ebp
22,197 00125 41 89 e9    mov r9d, ebp
22,198 00128 41 c1 e9 03   shr r9d, 3
22,199 0012c c4 c2 53 f7 eb shrx ebp, r11d, ebp
22,200 00131 41 f7 d9      neg r9d
22,201 00134 41 83 c1 03    add r9d, 3
22,202 00138 44 23 d5      and r10d, ebp
22,203 0013b 41 c1 ea 04   shr r10d, 4
22,204 0013f 49 03 d1      add rdx, r9
22,205
22,206 .B2.16::
22,207 00142 48 89 c5      mov rbp, rax
22,208 00145 48 ff c2      inc rdx
22,209 00148 49 2b ea      sub rbp, r10
22,210 0014b c5 fe 6f 45 00 vmovdqu ymm0, YMMWORD PTR [rbp]
22,211 00150 c5 fe 7f 00   vmovdqu YMMWORD PTR [rax], ymm0
22,212 00154 48 83 c0 18    add rax, 24
22,213 00158 80 3a 83      cmp BYTE PTR [rdx], -125
22,214 0015b 74 e5         je .B2.16
22,215 0015d eb 4c         jmp .B2.21
22,216
22,217 .B2.18::
22,218 0015f 44 89 d5      mov ebp, r10d
22,219 00162 41 bb ff ff ff 00 mov r11d, 16777215
22,220 00168 c1 ed 03      shr ebp, 3
22,221 0016b 83 e5 08      and ebp, 8
22,222 0016e 41 89 e9      mov r9d, ebp
22,223 00171 41 c1 e9 03   shr r9d, 3
22,224 00175 c4 c2 53 f7 eb shrx ebp, r11d, ebp
22,225 0017a 41 f7 d9      neg r9d
22,226 0017d 41 83 c1 02    add r9d, 2
22,227 00181 41 23 ea      and ebp, r10d
22,228 00184 41 83 e2 3c    and r10d, 60
22,229 00188 c1 ed 07      shr ebp, 7
22,230 0018b 49 03 d1      add rdx, r9
22,231 0018e 41 d1 ea      shr r10d, 1
22,232
22,233
22,234 .B2.19::
22,235 00191 49 89 c1      mov r9, rax
22,236 00194 48 ff c2      inc rdx
22,237 00197 4c 2b cd      sub r9, rbp
22,238 0019a c4 c1 7e 6f 01 vmovdqu ymm0, YMMWORD PTR [r9]
22,239 0019f c5 fe 7f 00   vmovdqu YMMWORD PTR [rax], ymm0
22,240 001a3 49 03 c2      add rax, r10
22,241 001a6 80 3a 83      cmp BYTE PTR [rdx], -125
22,242 001a9 74 e6         je .B2.19
22,243
22,244 .B2.21::
22,245 001ab 49 3b d0      cmp rdx, r8
22,246 001ae 0f 82 92 fe ff jb .B2.3
22,247 ff
22,248
22,249 .B2.22::
22,250 001b4 48 8b 6c 24 20 mov rbp, QWORD PTR [32+rsp]
22,251
22,252 .B2.23::
22,253 001b9 48 2b c1      sub rax, rcx

```

```

22,254 001bc c5 f8 77      vzeroupper
22,255 001bf 48 83 c4 38      add rsp, 56
22,256 001c3 c3              ret
22,257
22,258 .B2.24::
22,259 001c4 48 8b 6a 01      mov rbp, QWORD PTR [1+rdx]
22,260 001c8 48 89 28          mov QWORD PTR [rax], rbp
22,261 001cb 49 03 c1          add rax, r9
22,262 001ce 41 ff c1          inc r9d
22,263 001d1 49 03 d1          add rdx, r9
22,264 001d4 eb d5          jmp .B2.21
22,265
22,266 .B2.25::
22,267 001d6 41 c1 ea 08      shr r10d, 8
22,268 001da 48 83 c2 04      add rdx, 4
22,269 001de 41 f7 d9      neg r9d
22,270 001e1 49 f7 da      neg r10
22,271 001e4 4c 03 d0      add r10, rax
22,272 001e7 43 8d 6c 09 24  lea ebp, DWORD PTR [36+r9+r9]
22,273 001ec c4 c1 7e 6f 02  vmovdqu ymm0, YMMWORD PTR [r10]
22,274 001f1 c5 fe 7f 00      vmovdqu YMMWORD PTR [rax], ymm0
22,275 001f5 48 03 c5          add rax, rbp
22,276 001f8 eb b1          jmp .B2.21
22,277
22,278 .B2.26::
22,279 001fa c1 e5 03          shl ebp, 3
22,280 001fd 41 b9 ff ff ff      mov r9d, -1
22,281 ff                      shrx r11d, r9d, ebp
22,282 00203 c4 42 53 f7 d9      and r11d, r10d
22,283 00208 45 23 da          and r10d, 12
22,284 0020b 41 83 e2 0c      shr r11d, 4
22,285 0020f 41 c1 eb 04      neg r10d
22,286 00213 41 f7 da          add r10d, 16
22,287 00216 41 83 c2 10      neg r11
22,288 0021a 49 f7 db          add r11, rax
22,289 0021d 4c 03 d8          shr ebp, 3
22,290 00220 c1 ed 03          neg ebp
22,291 00223 f7 dd          add ebp, 4
22,292 00225 83 c5 04          vmovdqu ymm0, YMMWORD PTR [r11]
22,293 00228 c4 c1 7e 6f 03      vmovdqu YMMWORD PTR [rax], ymm0
22,294 0022d c5 fe 7f 00      add rax, r10
22,295 00231 49 03 c2          add rdx, rbp
22,296 00234 48 03 d5          jmp .B2.21
22,297 00237 e9 6f ff ff ff      ALIGN 16
22,298 0023c 0f 1f 40 00
22,299
22,300 .B2.27::
22,301 ; mark_end;
22,302 ?Decompress@@YA_KPEAD0_K@Z ENDP
22,303 */
22,304
22,305 uint64_t Decompress2020Dec17 (char* ret, char* src, uint64_t srcSize) {
22,306 char* retLOCAL = ret;
22,307 char* srcLOCAL = src;
22,308 char* srcEndLOCAL = src+srcSize;
22,309 unsigned int DWORDtrio;
22,310 unsigned int DWORDtrio2;
22,311 unsigned int DWORDtrioDumbo;

```

```

22,312 unsigned int MatchLen;
22,313 unsigned int Gear;
22,314 uint64_t QWORDquartet;
22,315 unsigned char MatchLenLUT0[8]; // LookUpTable
22,316 // The first 3 bits in section 'Bheema' OOL, 00 is offset len; L=1 long, L=0 short match:
22,317 MatchLenLUT0[0]=18; // 000, 18:(5+1)=3
22,318 MatchLenLUT0[1]=18; // 100, 18:(4+1)=3.6
22,319 MatchLenLUT0[2]=18; // 010, 18:(3+1)=4.5
22,320 MatchLenLUT0[3]=18; // 110, 18:(2+1)=6
22,321 MatchLenLUT0[4]=28; // 001, 32:(5+1)=5.3
22,322 MatchLenLUT0[5]=28; // 101, 32:(4+1)=6.4
22,323 MatchLenLUT0[6]=28; // 011, 32:(3+1)=8
22,324 MatchLenLUT0[7]=28; // 111, 32:(2+1)=10.6
22,325 // Instead of using array consider using QWORD variable and shifting it side to side:
22,326 // uint64_t MatchLenLUT = 0x2030405012121424; // (MatchLenLUT<<((7-OOL)<<3))>>8*7
22,327 // QWORD: [1byte][2byte][3byte][4byte][5byte][6byte][7byte][8byte]
22,328 //      LSB                                     MSB
22,329 // 00=0      [OOL_5][ 2 ][ 3 ][ 4 ][ 5 ]
22,330 // 00=1      [OOL_5][ 2 ][ 3 ][ 4 ]
22,331 // 00=2      [OOL_5][ 2 ][ 3 ]
22,332 // 00=3      [OOL_5][ 2 ]
22,333 while (srcLOCAL < srcEndLOCAL) {
22,334     DWORDtrio = *(unsigned int*)srcLOCAL;
22,335     //      MatchLen = (DWORDtrio&0x0C); // 0!4!8!12 // Commented and moved where it is used, 2020-Dec-17
22,336 /*
22,337 #ifndef _N_GP
22,338 #ifdef _N_prefetch_64
22,339     _mm_prefetch((char*)(srcLOCAL + 64), _MM_HINT_T0);
22,340 #endif
22,341 #ifdef _N_prefetch_128
22,342     _mm_prefetch((char*)(srcLOCAL + 64*2), _MM_HINT_T0);
22,343 #endif
22,344 #ifdef _N_prefetch_4096
22,345     _mm_prefetch((char*)(srcLOCAL + 64*64), _MM_HINT_T0);
22,346 #endif
22,347 #endif
22,348 */
22,349
22,350 // Sizewise/MatchLenWise priority:
22,351 // #01 704: (5+1)+1+1+1+1+1+1+1+1+1=44      (1TB)
22,352 // #02 320: (5+1)+1+1+1+1=32      (1TB)
22,353 // #03 256: (5+1)+1+1+1=28.4      (1TB)
22,354 // #04 192: (5+1)+1+1=24      (1TB)
22,355 // #05 90: 2+1+1=22.5      (512B)
22,356 // #06 44: 2+1=22      (512B)
22,357 // #07 120: 3+1+1+1=20      (128KB)
22,358 // #08 60: 2+1=20      (512B)
22,359 // #09 96: 2+1+1+1=19.2      (4KB)
22,360 // #10 128: (5+1)+1=18.2      (1TB)
22,361 // #11 90: 3+1+1=18      (128KB)
22,362 // #12 72: 2+1+1=18      (4KB)
22,363 // #13 96: 3+1+1+1=16      (1MB)
22,364 // #14 48: 2+1=16      (4KB)
22,365 // #15 60: 3+1=15      (128KB)
22,366 // #16 30: 2=15      (512B)
22,367 // #17 72: 3+1+1=14.4      (1MB)
22,368 // #18 56: (2+1)+1=14      (8KB)
22,369 // #19 96: 4+1+1+1=13.7      (256MB)

```



```

22,370 // #20 72:4+1+1= 12 (256MB)
22,371 // #21 48:3+1= 12 (1MB)
22,372 // #22 24:2= 12 (4KB)
22,373 // #23 12:1= 12 (16B)
22,374 // #24 56:(3+1)+1= 11.2 (2MB)
22,375 // #25 44:3+1= 11 (128KB)
22,376 // #26 22:2= 11 (512B)
22,377 // #27 64:(5+1)= 10.6 (1TB)
22,378 // #28 30:3= 10 (128KB)
22,379 // #29 48:4+1= 9.6 (256MB)
22,380 // #30 56:(4+1)+1= 9.3 (512MB)
22,381 // #31 28:(2+1)= 9.3 (8KB)
22,382 // #32 36:(2+1)+1= 9 (8KB)
22,383 // #33 56:(5+1)+1= 8 (128GB)
22,384 // #34 24:3= 8 (1MB)
22,385 // #35 16:2= 8 (4KB)
22,386 // #36 8:1= 8 (16B)
22,387 // #37 22:3= 7.3 (128KB)
22,388 // #38 36:(3+1)+1= 7.2 (2MB)
22,389 // #39 28:(3+1)= 7 (2MB)
22,390 // #40 14:2= 7 (512B)
22,391 // #41 36:(4+1)+1= 6 (512MB)
22,392 // #42 24:4= 6 (256MB)
22,393 // #43 18:(2+1)= 6 (8KB)
22,394 // #44 12:2= 6 (4KB)
22,395 // #45 28:(4+1)= 5.6 (512MB)
22,396 // #46 16:3= 5.3 (1MB)
22,397 // #47 36:(5+1)+1= 5.1 (128GB)
22,398 // #48 28:(5+1)= 4.6 (128GB)
22,399 // #49 14:3= 4.6 (128KB)
22,400 // #50 18:(3+1)= 4.5 (2MB)
22,401 // #51 16:4= 4 (16MB)F
22,402 // #52 12:3= 4 (1MB)
22,403 // #53 8:2= 4 (4KB)
22,404 // #54 4:1= 4 (16B)
22,405 // #55 18:(4+1)= 3.6 (512MB)
22,406 // #56 14:4= 3.5 (16MB)
22,407 // #57 18:(5+1)= 3 (128GB)
22,408 // #58 12:4= 3 (16MB)
22,409 // #59 6:2= 3 (512B)
22,410 // #60 8:3= 2.6 (1MB)
22,411 // #61 10:4= 2.5 (16MB)
22,412 // #62 8:4= 2 (16MB)
22,413 // #63 6:3= 2 (128KB)
22,414 // #64 4:2= 2 (4KB)
22,415 // #65 6:4= 1.5 (16MB)
22,416 // #66 4:3= 1.3 (1MB)
22,417
22,418 // !1stLSB !2ndLSB !3rdLSB !
22,419 // -----
22,420 // !00!LL!xxxx!xxxxxxxx!xxxxxx!xx!
22,421 // -----
22,422 // [1bit 16bit] 24bit]
22,423 // LL = 00b means Long MatchLength, (4-LL)<<2 or 16
22,424 // LL = 01b means Long MatchLength, (4-LL)<<2 or 12
22,425 // LL = 10b means Long MatchLength, (4-LL)<<2 or 8
22,426 // LL = 11b means Long MatchLength, (4-LL)<<2 or 4
22,427 // xxxx0011b Literals for xxxx in 1..15-7, Matches for 10..15 i.e. Sliding Window is 4*8-8=24 or 16MB

```

```

22,428 // 00 = 00b MatchOffset, 0xFFFFFFFF>>00, 4 bytes long i.e. Sliding Window is 4*8-LL-00=4*8-4=28 or 256MB
22,429 // 00 = 01b MatchOffset, 0xFFFFFFFF>>00, 3 bytes long i.e. Sliding Window is 3*8-LL-00=3*8-4=20 or 1MB
22,430 // 00 = 10b MatchOffset, 0xFFFFFFFF>>00, 2 bytes long i.e. Sliding Window is 2*8-LL-00=2*8-4=12 or 4KB
22,431 // 00 = 11b MatchOffset, 0xFFFFFFFF>>00, 1 byte long i.e. Sliding Window is 1*8-LL-00=1*8-4=4 or 16B
22,432
22,433 if ( (DWORDtrio & 0x0F) == 0x03 ) {
22,434 if ( ((DWORDtrio & 0xFF)>>4) <= 7 ) { // 9 not used, only 1..7 for literals, 8 for 'ditto' tag
22,435 // Make next (nested branch) unbranched (https://godbolt.org/#):
22,436 // x = (symbolA <= symbolB) ? countA : 0;
22,437 // x = ((-(symbolA <= symbolB)) & countA);
22,438 if ( ((DWORDtrio & 0xFF)>>4) == 0 ) {
22,439 // srcLOCAL-2
22,440 // srcLOCAL
22,441 // srcLOCAL+1
22,442 // QWORD: [1byte][2byte][3byte][4byte][5byte][6byte][7byte][8byte]
22,443 // LSB MSB
22,444 // 00=0 [00L_5][ 2 ][ 3 ][ 4 ][ 5 ]
22,445 // 00=1 [00L_5][ 2 ][ 3 ][ 4 ]
22,446 // 00=2 [00L_5][ 2 ][ 3 ]
22,447 // 00=3 [00L_5][ 2 ]
22,448 MatchLen = (DWORDtrio>>8)&0x7;
22,449 QWORDquartet = *(uint64_t*)(srcLOCAL+1- (3+(MatchLen&0x3)) ); // Safe since 5bytes backwards reads are less than MatchLengths used here.
22,450 QWORDquartet = QWORDquartet >> (((3+(MatchLen&0x3))<<3)+3);
22,451 srcLOCAL+= 5-(MatchLen&0x3)+1-(1);
22,452 ditto1:
22,453 #ifdef _N_GP
22,454 memcpy(retLOCAL, (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) ), 32);
22,455 #endif
22,456 #ifdef _N_XMM
22,457 NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) +16*(0) ), retLOCAL +16*(0));
22,458 NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) +16*(1) ), retLOCAL +16*(1));
22,459 #endif
22,460 #ifdef _N_YMM
22,461 SlowCopy256bit( (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) +32*(0) ), retLOCAL +32*(0));
22,462 #endif
22,463 retLOCAL+= MatchLenLUT0[MatchLen];
22,464 if (*(unsigned char*)(++srcLOCAL) == 0x83) goto ditto1;
22,465 // *(uint32_t*)(retLOCAL+4*(0)) = *(uint32_t*)(retLOCAL-((DWORDtrio&0xFFFF)>>8)+4*(0));
22,466 // retLOCAL+= 3;
22,467 // srcLOCAL+= 2;
22,468 } else {
22,469 // #ifdef _N_GP
22,470 // memcpy(retLOCAL, (const char *) ( (uint64_t)(srcLOCAL+1) ), 16);
22,471 // #endif
22,472 // #ifdef _N_XMM
22,473 // NotSoSlowCopy128bit( (const char *) ( (uint64_t)(srcLOCAL+1) ), retLOCAL );
22,474 // #endif
22,475 *(uint64_t*)(retLOCAL) = *(uint64_t*)((const char *) ( (uint64_t)(srcLOCAL+1) ));
22,476 retLOCAL+= ((DWORDtrio & 0xFF)>>4);
22,477 srcLOCAL+= ((DWORDtrio & 0xFF)>>4)+1;
22,478 }
22,479 } else if ( ((DWORDtrio & 0xFF)>>4) > 9 ) {
22,480 #ifdef _N_GP
22,481 memcpy(retLOCAL, (const char *) ( (uint64_t)(retLOCAL-(DWORDtrio>>8)) ), 16); // No need of DWORDtrio&0xFFFFFFFF
22,482 #endif
22,483 #ifdef _N_XMM
22,484 NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-(DWORDtrio>>8))+16*(0) ), retLOCAL +16*(0));
22,485 #endif

```

```

22,486         #ifdef _N_YMM
22,487             //NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-(DWORDtrio>>8))+16*(0) ), retLOCAL +16*(0));
22,488             SlowCopy256bit( (const char *) ( (uint64_t)(retLOCAL-(DWORDtrio>>8))+16*(0) ), retLOCAL +16*(0)); // 2020-Dec-17
22,489         #endif
22,490         retLOCAL+= (18 - ((DWORDtrio & 0xFF)>>4))<<1;
22,491         srcLOCAL+= 4; // 6!8!10!12!14!16 in 16MB window
22,492     } else { // i.e. 8!9
22,493         QWORDquartet = *(uint64_t*)(srcLOCAL+1- (3+(0)) ); // Safe since 5bytes backwards reads are less than MatchLengths used here.
22,494         QWORDquartet = QWORDquartet >> (((3+(0))<<3)+3-(3)); // Full 5x8=40bit i.e. 1TB
22,495         srcLOCAL+= 5-(0)+1-(1);
22,496     ditto2:
22,497         #ifdef _N_GP
22,498             memcpy(retLOCAL, (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) ), 64);
22,499         #endif
22,500         #ifdef _N_XMM
22,501             NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) +16*(0) ), retLOCAL +16*(0));
22,502             NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) +16*(1) ), retLOCAL +16*(1));
22,503             NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) +16*(2) ), retLOCAL +16*(2));
22,504             NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) +16*(3) ), retLOCAL +16*(3));
22,505         #endif
22,506         #ifdef _N_YMM
22,507             SlowCopy256bit( (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) +32*(0) ), retLOCAL +32*(0));
22,508             SlowCopy256bit( (const char *) ( (uint64_t)(retLOCAL-QWORDquartet) +32*(1) ), retLOCAL +32*(1));
22,509         #endif
22,510         retLOCAL+= 64;
22,511     if (*(unsigned char*)(++srcLOCAL) == 0x83) goto ditto2;
22,512         // To reduce compressed size further consider:
22,513         // if (uint64_t)(retLOCAL - ret)<<((1LL)<<((37-8))) srcLOCAL+= 5-(MatchLen&0x3)+1 -1; // gain for 512-MB files
22,514         // if (uint64_t)(retLOCAL - ret)<<((1LL)<<((37-8-8))) srcLOCAL+= 5-(MatchLen&0x3)+1 -1-1; // gain for 2-MB files
22,515     }
22,516     } else if ( (DWORDtrio & 0x0f) == 0x0C ) {
22,517         // 0011LLGx xxxxxxxx xxxxxxxx, 11LL is matchlength, 2bytes window: 1+8=512B; 3bytes window: 1+8+8=128KB
22,518         Gear = (DWORDtrio>>3)&0x8; // 0/8 is 3/2 bytes respectively
22,519         //*(uint64_t*)(retLOCAL+8*(0)) = *(uint64_t*)((retLOCAL-( (DWORDtrio&((0xFFFFFFFF)>>Gear))>>7 ))+8*(0));
22,520         srcLOCAL+= 3-(Gear>>3) -1; // ! -1 is for 'ditto' tag
22,521     ditto4:
22,522         #ifdef _N_GP
22,523             memcpy(retLOCAL, (const char *) ( (uint64_t)(retLOCAL-((DWORDtrio&((0xFFFFFFFF)>>Gear))>>7)) ), 32);
22,524         #endif
22,525         #ifdef _N_XMM
22,526             NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-((DWORDtrio&((0xFFFFFFFF)>>Gear))>>7))+16*(0) ), retLOCAL +16*(0));
22,527             NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-((DWORDtrio&((0xFFFFFFFF)>>Gear))>>7))+16*(1) ), retLOCAL +16*(1));
22,528         #endif
22,529         #ifdef _N_YMM
22,530             SlowCopy256bit( (const char *) ( (uint64_t)(retLOCAL-((DWORDtrio&((0xFFFFFFFF)>>Gear))>>7))+32*(0) ), retLOCAL +32*(0));
22,531         #endif
22,532         retLOCAL+= (DWORDtrio & 0x3C)>>1; // (12+0)>>1=6 ! (12+16)>>1=14 ! (12+32)>>1=22 ! (12+48)>>1=30
22,533     if (*(unsigned char*)(++srcLOCAL) == 0x83) goto ditto4;
22,534     } else if ( (DWORDtrio & 0x03) == 0x00 ) {
22,535         MatchLen = (DWORDtrio&0x0C); // 0!4!8!12 // Commented and moved where it is used, 2020-Dec-17
22,536         // MatchLen 0!4!8 <<1 0!8!16
22,537         MatchLen=MatchLen<<1; // To avoid 'LEA'
22,538         srcLOCAL+= 4-(MatchLen>>3) -1; // 24:2, 24:3, 24:4 ! -1 is for 'ditto' tag
22,539     ditto3:
22,540         #ifdef _N_GP
22,541             *(uint64_t*)(retLOCAL+8*(0)) = *(uint64_t*)((retLOCAL-( (DWORDtrio&0xFFFFFFFF)>>MatchLen))>>4 )+8*(0));
22,542             *(uint64_t*)(retLOCAL+8*(1)) = *(uint64_t*)((retLOCAL-( (DWORDtrio&0xFFFFFFFF)>>MatchLen))>>4 )+8*(1));
22,543             *(uint64_t*)(retLOCAL+8*(2)) = *(uint64_t*)((retLOCAL-( (DWORDtrio&0xFFFFFFFF)>>MatchLen))>>4 )+8*(2));

```

```

22,544         #endif
22,545         #ifdef _N_XMM
22,546         *(uint64_t*)(retLOCAL+8*(0)) = *(uint64_t*)((retLOCAL-( (DWORDtrio&(0xFFFFFFFF)>>MatchLen))>>4 ))+8*(0));
22,547         NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-( (DWORDtrio&(0xFFFFFFFF)>>MatchLen))>>4 ))+8*(1) , retLOCAL +8*(1));
22,548         #endif
22,549         #ifdef _N_YMM
22,550             SlowCopy256bit( (const char *) ( (uint64_t)(retLOCAL-( (DWORDtrio&(0xFFFFFFFF)>>MatchLen))>>4 )) , retLOCAL);
22,551         #endif
22,552         retLOCAL+= 24;
22,553         if (*(unsigned char*)(++srcLOCAL) == 0x83) goto ditto3;
22,554     } else {
22,555         DWORDtrioDumbo = (DWORDtrio & 0x03)<<3; // To avoid 'LEA'
22,556         MatchLen = (DWORDtrio&0x0C); // 0!4!8!12 // Commented and moved where it is used, 2020-Dec-17
22,557         #ifdef _N_GP
22,558             memcpy(retLOCAL, (const char *) ( (uint64_t)(retLOCAL-( (DWORDtrio&(0xFFFFFFFF)>>DWORDtrioDumbo))>>4 )) , 16);
22,559         #endif
22,560         #ifdef _N_XMM
22,561             NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-( (DWORDtrio&(0xFFFFFFFF)>>DWORDtrioDumbo))>>4 )) , retLOCAL );
22,562         #endif
22,563         #ifdef _N_YMM
22,564             //NotSoSlowCopy128bit( (const char *) ( (uint64_t)(retLOCAL-( (DWORDtrio&(0xFFFFFFFF)>>DWORDtrioDumbo))>>4 )) , retLOCAL );
22,565             SlowCopy256bit( (const char *) ( (uint64_t)(retLOCAL-( (DWORDtrio&(0xFFFFFFFF)>>DWORDtrioDumbo))>>4 )) , retLOCAL ); // 2020-Dec-17
22,566         #endif
22,567         retLOCAL+= 16-MatchLen;
22,568         srcLOCAL+= 4-(DWORDtrioDumbo>>3);
22,569     }
22,570 }
22,571 return (uint64_t)(retLOCAL - ret);
22,572 }
22,573
22,574 // 2020-Dec-17 variant:
22,575 // Core Function Size: 0023c - 00046 + 4 = 506 bytes
22,576 /*
22,577 ; mark_description "Intel(R) C++ Compiler XE for applications running on Intel(R) 64, Version 15.0.0.108 Build 20140726";
22,578 ; mark_description "-TP -O3 -arch:CORE-AVX2 -D_N_YMM -D_N_prefetch_4096 -D_N_HIGH_PRIORITY -D_icl_mumbo_jumbo_ -Facs -DHashInBIT";
22,579 ; mark_description "S=24 -DHashChunkSizeInBITS=24 -DRAMpoolInKB=5120 -DBtreeHEURISTIC -D_WIN32_ENVIRONMENT_ -DLongestLineInclusi";
22,580 ; mark_description "ve=128 -DSpeedUpBuilding=320 -D_NDD -DLITE -D_N_alone";
22,581
22,582
22,583 ?Decompress@@YA_KPEAD0_K0Z      PROC
22,584 ; parameter 1: rcx
22,585 ; parameter 2: rdx
22,586 ; parameter 3: r8
22,587 .B2.1::
22,588     000000 48 83 ec 38      sub rsp, 56
22,589     000004 4c 03 c2          add r8, rdx
22,590     000007 41 b1 12          mov r9b, 18
22,591     00000a 41 b2 1c          mov r10b, 28
22,592     00000d 48 89 c8          mov rax, rcx
22,593     00010 44 88 4c 24 28    mov BYTE PTR [40+rsp], r9b
22,594     00015 44 88 4c 24 29    mov BYTE PTR [41+rsp], r9b
22,595     0001a 44 88 4c 24 2a    mov BYTE PTR [42+rsp], r9b
22,596     0001f 44 88 4c 24 2b    mov BYTE PTR [43+rsp], r9b
22,597     00024 44 88 54 24 2c    mov BYTE PTR [44+rsp], r10b
22,598     00029 44 88 54 24 2d    mov BYTE PTR [45+rsp], r10b
22,599     0002e 44 88 54 24 2e    mov BYTE PTR [46+rsp], r10b
22,600     00033 44 88 54 24 2f    mov BYTE PTR [47+rsp], r10b
22,601     00038 49 3b d0          cmp rdx, r8

```

```

22,602 0003b 0f 83 25 01 00
22,603 00 jae .B2.19
22,604
22,605 .B2.2::
22,606 00041 48 89 6c 24 20 mov QWORD PTR [32+rsp], rbp
22,607
22,608 .B2.3::
22,609 00046 44 8b 12 mov r10d, DWORD PTR [rdx]
22,610 00049 44 89 d5 mov ebp, r10d
22,611 0004c 83 e5 0f and ebp, 15
22,612 0004f 83 fd 03 cmp ebp, 3
22,613 00052 0f 85 ab 00 00 jne .B2.13
22,614 00
22,615
22,616 .B2.4::
22,617 00058 45 0f b6 ca movzx r9d, r10b
22,618 0005c 41 c1 e9 04 shr r9d, 4
22,619 00060 41 83 f9 07 cmp r9d, 7
22,620 00064 77 5f ja .B2.9
22,621
22,622 .B2.5::
22,623 00066 45 85 c9 test r9d, r9d
22,624 00069 0f 85 02 01 00 jne .B2.20
22,625 00
22,626
22,627 .B2.6::
22,628 0006f 41 c1 ea 08 shr r10d, 8
22,629 00073 49 83 e2 07 and r10, 7
22,630 00077 45 89 d1 mov r9d, r10d
22,631 0007a 41 83 e1 03 and r9d, 3
22,632 0007e 41 8d 69 03 lea ebp, DWORD PTR [3+r9]
22,633 00082 46 8d 1c cd 1b lea r11d, DWORD PTR [27+r9*8]
22,634 00 00 00
22,635 0008a 48 f7 dd neg rbp
22,636 0008d 41 f7 d9 neg r9d
22,637 00090 48 03 ea add rbp, rdx
22,638 00093 41 83 c1 05 add r9d, 5
22,639 00097 49 03 d1 add rdx, r9
22,640 0009a c4 e2 a3 f7 6d
22,641 01 shrx rbp, QWORD PTR [1+rbp], r11
22,642 000a0 46 0f b6 4c 14 movzx r9d, BYTE PTR [40+rsp+r10]
22,643 28
22,644
22,645 .B2.7::
22,646 000a6 49 89 c2 mov r10, rax
22,647 000a9 48 ff c2 inc rdx
22,648 000ac 4c 2b d5 sub r10, rbp
22,649 000af c4 c1 7e 6f 02 vmovdqu ymm0, YMMWORD PTR [r10]
22,650 000b4 c5 fe 7f 00 vmovdqu YMMWORD PTR [rax], ymm0
22,651 000b8 49 03 c1 add rax, r9
22,652 000bb 80 3a 83 cmp BYTE PTR [rdx], -125
22,653 000be 74 e6 je .B2.7
22,654 000c0 e9 93 00 00 00 jmp .B2.17
22,655
22,656 .B2.9::
22,657 000c5 41 83 f9 09 cmp r9d, 9
22,658 000c9 0f 87 b4 00 00
22,659 00 ja .B2.21

```

```
22,660
22,661 .B2.10::
22,662 000cf 48 8b 6a fe      mov rbp, QWORD PTR [-2+rdx]
22,663 000d3 48 83 c2 05      add rdx, 5
22,664 000d7 48 c1 ed 18      shr rbp, 24
22,665
22,666 .B2.11::
22,667 000db 49 89 c1          mov r9, rax
22,668 000de 48 ff c2          inc rdx
22,669 000e1 4c 2b cd          sub r9, rbp
22,670 000e4 c4 c1 7e 6f 01    vmovdqu ymm0, YMMWORD PTR [r9]
22,671 000e9 c5 fe 7f 00      vmovdqu YMMWORD PTR [rax], ymm0
22,672 000ed c4 c1 7e 6f 49    vmovdqu ymm1, YMMWORD PTR [32+r9]
22,673 20                      vmovdqu YMMWORD PTR [32+rax], ymm1
22,674 000f3 c5 fe 7f 48 20    add rax, 64
22,675 000f8 48 83 c0 40      cmp BYTE PTR [rdx], -125
22,676 000fc 80 3a 83          je .B2.11
22,677 000ff 74 da            jmp .B2.17
22,678 00101 eb 55
22,679
22,680 .B2.13::
22,681 00103 83 fd 0c          cmp ebp, 12
22,682 00106 0f 85 9b 00 00    jne .B2.22
22,683 00
22,684
22,685 .B2.14::
22,686 0010c 44 89 d5          mov ebp, r10d
22,687 0010f 41 bb ff ff ff    mov r11d, 16777215
22,688 00
22,689 00115 c1 ed 03          shr ebp, 3
22,690 00118 83 e5 08          and ebp, 8
22,691 0011b 41 89 e9          mov r9d, ebp
22,692 0011e 41 c1 e9 03      shr r9d, 3
22,693 00122 c4 c2 53 f7 eb    shrx ebp, r11d, ebp
22,694 00127 41 f7 d9          neg r9d
22,695 0012a 41 83 c1 02      add r9d, 2
22,696 0012e 41 23 ea          and ebp, r10d
22,697 00131 41 83 e2 3c      and r10d, 60
22,698 00135 c1 ed 07          shr ebp, 7
22,699 00138 49 03 d1          add rdx, r9
22,700 0013b 41 d1 ea          shr r10d, 1
22,701
22,702 .B2.15::
22,703 0013e 49 89 c1          mov r9, rax
22,704 00141 48 ff c2          inc rdx
22,705 00144 4c 2b cd          sub r9, rbp
22,706 00147 c4 c1 7e 6f 01    vmovdqu ymm0, YMMWORD PTR [r9]
22,707 0014c c5 fe 7f 00      vmovdqu YMMWORD PTR [rax], ymm0
22,708 00150 49 03 c2          add rax, r10
22,709 00153 80 3a 83          cmp BYTE PTR [rdx], -125
22,710 00156 74 e6            je .B2.15
22,711
22,712 .B2.17::
22,713 00158 49 3b d0          cmp rdx, r8
22,714 0015b 0f 82 e5 fe ff    jb .B2.3
22,715 ff
22,716
22,717 .B2.18::
```

```
22,718 00161 48 8b 6c 24 20 mov rbp, QWORD PTR [32+rsp]
22,719
22,720 .B2.19::
22,721 00166 48 2b c1 sub rax, rcx
22,722 00169 c5 f8 77 vzeroupper
22,723 0016c 48 83 c4 38 add rsp, 56
22,724 00170 c3 ret
22,725
22,726 .B2.20::
22,727 00171 48 8b 6a 01 mov rbp, QWORD PTR [1+rdx]
22,728 00175 48 89 28 mov QWORD PTR [rax], rbp
22,729 00178 49 03 c1 add rax, r9
22,730 0017b 41 ff c1 inc r9d
22,731 0017e 49 03 d1 add rdx, r9
22,732 00181 eb d5 jmp .B2.17
22,733
22,734 .B2.21::
22,735 00183 41 c1 ea 08 shr r10d, 8
22,736 00187 48 83 c2 04 add rdx, 4
22,737 0018b 41 f7 d9 neg r9d
22,738 0018e 49 f7 da neg r10
22,739 00191 4c 03 d0 add r10, rax
22,740 00194 43 8d 6c 09 24 lea ebp, DWORD PTR [36+r9+r9]
22,741 00199 c4 c1 7e 6f 02 vmovdqu ymm0, YMMWORD PTR [r10]
22,742 0019e c5 fe 7f 00 vmovdqu YMMWORD PTR [rax], ymm0
22,743 001a2 48 03 c5 add rax, rbp
22,744 001a5 eb b1 jmp .B2.17
22,745
22,746 .B2.22::
22,747 001a7 44 89 d5 mov ebp, r10d
22,748 001aa 83 e5 03 and ebp, 3
22,749 001ad 75 4b jne .B2.26
22,750
22,751 .B2.23::
22,752 001af 44 89 d5 mov ebp, r10d
22,753 001b2 41 bb ff ff ff
22,754 ff mov r11d, -1
22,755 001b8 83 e5 0c and ebp, 12
22,756 001bb 03 ed add ebp, ebp
22,757 001bd 41 89 e9 mov r9d, ebp
22,758 001c0 41 c1 e9 03 shr r9d, 3
22,759 001c4 c4 c2 53 f7 eb shrx ebp, r11d, ebp
22,760 001c9 41 f7 d9 neg r9d
22,761 001cc 41 83 c1 03 add r9d, 3
22,762 001d0 44 23 d5 and r10d, ebp
22,763 001d3 41 c1 ea 04 shr r10d, 4
22,764 001d7 49 03 d1 add rdx, r9
22,765
22,766 .B2.24::
22,767 001da 48 89 c5 mov rbp, rax
22,768 001dd 48 ff c2 inc rdx
22,769 001e0 49 2b ea sub rbp, r10
22,770 001e3 c5 fe 6f 45 00 vmovdqu ymm0, YMMWORD PTR [rbp]
22,771 001e8 c5 fe 7f 00 vmovdqu YMMWORD PTR [rax], ymm0
22,772 001ec 48 83 c0 18 add rax, 24
22,773 001f0 80 3a 83 cmp BYTE PTR [rdx], -125
22,774 001f3 74 e5 je .B2.24
22,775 001f5 e9 5e ff ff ff jmp .B2.17
```

```

22,776
22,777 .B2.26::
22,778 001fa c1 e5 03      shl ebp, 3
22,779 001fd 41 b9 ff ff ff
22,780 ff                  mov r9d, -1
22,781 00203 c4 42 53 f7 d9 shr r11d, r9d, ebp
22,782 00208 45 23 da      and r11d, r10d
22,783 0020b 41 83 e2 0c    and r10d, 12
22,784 0020f 41 c1 eb 04    shr r11d, 4
22,785 00213 41 f7 da      neg r10d
22,786 00216 41 83 c2 10    add r10d, 16
22,787 0021a 49 f7 db      neg r11
22,788 0021d 4c 03 d8      add r11, rax
22,789 00220 c1 ed 03      shr ebp, 3
22,790 00223 f7 dd          neg ebp
22,791 00225 83 c5 04      add ebp, 4
22,792 00228 c4 c1 7e 6f 03 vmovdqu ymm0, YMMWORD PTR [r11]
22,793 0022d c5 fe 7f 00    vmovdqu YMMWORD PTR [rax], ymm0
22,794 00231 49 03 c2      add rax, r10
22,795 00234 48 03 d5      add rdx, rbp
22,796 00237 e9 1c ff ff ff jmp .B2.17
22,797 0023c 0f 1f 40 00    ALIGN      16
22,798
22,799 .B2.27::
22,800 ; mark_end;
22,801 ?Decompress@@YA_KPEAD0_K@Z ENDP
22,802 */
22,803
22,804 /*
22,805 ; 'Ryuugan-ditto-1TB' decompression loop, 251-4c+5=522 bytes long, 148 instructions long:
22,806 ; mark_description "Intel(R) C++ Compiler XE for applications running on Intel(R) 64, Version 15.0.0.108 Build 20140726";
22,807 ; mark_description "-TP -O3 -archSSE4.1 -D_N_XMM -D_N_prefetch_4096 -D_N_HIGH_PRIORITY -D_icl_mumbo_jumbo_ -F&cs";
22,808 ; 17 branches [
22,809 ; unconditional: 6 'jmp'
22,810 ; jmp .B2.17
22,811 ; jmp .B2.17
22,812 ; jmp .B2.17
22,813 ; jmp .B2.17
22,814 ; jmp .B2.17
22,815 ; jmp .B2.17
22,816 ; conditional: 4 'je'
22,817 ; je .B2.7
22,818 ; je .B2.11
22,819 ; je .B2.15
22,820 ; je .B2.24
22,821 ; conditional: 4 'jne'
22,822 ; jne .B2.13
22,823 ; jne .B2.20
22,824 ; jne .B2.22
22,825 ; jne .B2.26
22,826 ; conditional: 2 'ja'
22,827 ; ja .B2.9
22,828 ; ja .B2.21
22,829 ; conditional: 1 'jb'
22,830 ; jb .B2.3
22,831 ; ]
22,832 _TEXT      SEGMENT      'CODE'
22,833 ALIGN      16

```



```
22,834 PUBLIC ?Decompress@@YA_KPEAD0_K@Z
22,835 ?Decompress@@YA_KPEAD0_K@Z PROC
22,836 ; parameter 1: rcx
22,837 ; parameter 2: rdx
22,838 ; parameter 3: r8
22,839 .B2.1::
22,840 00000 48 83 ec 38 sub rsp, 56
22,841 00004 4d 89 c1 mov r9, r8
22,842 00007 4c 03 ca add r9, rdx
22,843 0000a 49 89 c8 mov r8, rcx
22,844 0000d 41 b2 12 mov r10b, 18
22,845 00010 41 b3 1c mov r11b, 28
22,846 00013 44 88 54 24 28 mov BYTE PTR [40+rsp], r10b
22,847 00018 4c 89 c0 mov rax, r8
22,848 0001b 44 88 54 24 29 mov BYTE PTR [41+rsp], r10b
22,849 00020 49 3b d1 cmp rdx, r9
22,850 00023 44 88 54 24 2a mov BYTE PTR [42+rsp], r10b
22,851 00028 44 88 54 24 2b mov BYTE PTR [43+rsp], r10b
22,852 0002d 44 88 5c 24 2c mov BYTE PTR [44+rsp], r11b
22,853 00032 44 88 5c 24 2d mov BYTE PTR [45+rsp], r11b
22,854 00037 44 88 5c 24 2e mov BYTE PTR [46+rsp], r11b
22,855 0003c 44 88 5c 24 2f mov BYTE PTR [47+rsp], r11b
22,856 00041 0f 83 45 01 00 jae .B2.19
22,857 00 jae .B2.19
22,858 .B2.2::
22,859 00047 48 89 6c 24 20 mov QWORD PTR [32+rsp], rbp
22,860 .B2.3::
22,861 0004c 44 8b 12 mov r10d, DWORD PTR [rdx]
22,862 0004f 45 89 d3 mov r11d, r10d
22,863 00052 44 89 d5 mov ebp, r10d
22,864 00055 41 83 e3 0f and r11d, 15
22,865 00059 83 e5 0c and ebp, 12
22,866 0005c 41 83 fb 03 cmp r11d, 3
22,867 00060 0f 85 c1 00 00 jne .B2.13
22,868 00 jne .B2.13
22,869 .B2.4::
22,870 00066 41 0f b6 ea movzx ebp, r10b
22,871 0006a c1 ed 04 shr ebp, 4
22,872 0006d 83 fd 07 cmp ebp, 7
22,873 00070 77 65 ja .B2.9
22,874 .B2.5::
22,875 00072 85 ed test ebp, ebp
22,876 00074 0f 85 1a 01 00 jne .B2.20
22,877 00 jne .B2.20
22,878 .B2.6::
22,879 0007a 41 c1 ea 08 shr r10d, 8
22,880 0007e 49 83 e2 07 and r10, 7
22,881 00082 44 89 d5 mov ebp, r10d
22,882 00085 83 e5 03 and ebp, 3
22,883 00088 44 8d 5d 03 lea r11d, DWORD PTR [3+rbp]
22,884 0008c 8d 0c ed 1b 00 lea ecx, DWORD PTR [27+rbp*8]
22,885 00 00 lea ecx, DWORD PTR [27+rbp*8]
22,886 00093 49 f7 db neg r11
22,887 00096 f7 dd neg ebp
22,888 00098 4c 03 da add r11, rdx
22,889 0009b 83 c5 05 add ebp, 5
22,890 0009e 48 03 d5 add rdx, rbp
22,891 000a1 4d 8b 5b 01 mov r11, QWORD PTR [1+r11]
```

```
22,892 000a5 49 d3 eb      shr r11, cl
22,893 000a8 42 0f b6 4c 14
22,894      28      movzx ecx, BYTE PTR [40+rsp+r10]
22,895 .B2.7::
22,896 000ae 48 89 c5      mov rbp, rax
22,897 000b1 48 ff c2      inc rdx
22,898 000b4 49 2b eb      sub rbp, r11
22,899 000b7 f2 0f f0 45 00 lddqu xmm0, XMMWORD PTR [rbp]
22,900 000bc f3 0f 7f 00      movdqu XMMWORD PTR [rax], xmm0
22,901 000c0 f2 0f f0 4d 10 lddqu xmm1, XMMWORD PTR [16+rbp]
22,902 000c5 f3 0f 7f 48 10 movdqu XMMWORD PTR [16+rax], xmm1
22,903 000ca 48 03 c1      add rax, rcx
22,904 000cd 80 3a 83      cmp BYTE PTR [rdx], -125
22,905 000d0 74 dc      je .B2.7
22,906 000d2 e9 a7 00 00 00 jmp .B2.17
22,907 .B2.9::
22,908 000d7 83 fd 09      cmp ebp, 9
22,909 000da 0f 87 c5 00 00
22,910      00      ja .B2.21
22,911 .B2.10::
22,912 000e0 48 8b 4a fe      mov rcx, QWORD PTR [-2+rdx]
22,913 000e4 48 83 c2 05      add rdx, 5
22,914 000e8 48 c1 e9 18      shr rcx, 24
22,915 .B2.11::
22,916 000ec 48 89 c5      mov rbp, rax
22,917 000ef 48 ff c2      inc rdx
22,918 000f2 48 2b e9      sub rbp, rcx
22,919 000f5 f2 0f f0 45 00 lddqu xmm0, XMMWORD PTR [rbp]
22,920 000fa f3 0f 7f 00      movdqu XMMWORD PTR [rax], xmm0
22,921 000fe f2 0f f0 4d 10 lddqu xmm1, XMMWORD PTR [16+rbp]
22,922 00103 f3 0f 7f 48 10 movdqu XMMWORD PTR [16+rax], xmm1
22,923 00108 f2 0f f0 55 20 lddqu xmm2, XMMWORD PTR [32+rbp]
22,924 0010d f3 0f 7f 50 20 movdqu XMMWORD PTR [32+rax], xmm2
22,925 00112 f2 0f f0 5d 30 lddqu xmm3, XMMWORD PTR [48+rbp]
22,926 00117 f3 0f 7f 58 30 movdqu XMMWORD PTR [48+rax], xmm3
22,927 0011c 48 83 c0 40      add rax, 64
22,928 00120 80 3a 83      cmp BYTE PTR [rdx], -125
22,929 00123 74 c7      je .B2.11
22,930 00125 eb 57      jmp .B2.17
22,931 .B2.13::
22,932 00127 41 83 fb 0c      cmp r11d, 12
22,933 0012b 0f 85 96 00 00
22,934      00      jne .B2.22
22,935 .B2.14::
22,936 00131 44 89 d1      mov ecx, r10d
22,937 00134 c1 e9 03      shr ecx, 3
22,938 00137 83 e1 08      and ecx, 8
22,939 0013a 89 cd      mov ebp, ecx
22,940 0013c c1 ed 03      shr ebp, 3
22,941 0013f f7 dd      neg ebp
22,942 00141 83 c5 02      add ebp, 2
22,943 00144 48 03 d5      add rdx, rbp
22,944 00147 bd ff ff ff 00 mov ebp, 16777215
22,945 0014c d3 ed      shr ebp, cl
22,946 0014e 41 23 ea      and ebp, r10d
22,947 00151 41 83 e2 3c      and r10d, 60
22,948 00155 c1 ed 07      shr ebp, 7
22,949 00158 41 d1 ea      shr r10d, 1
```

```
22,950 .B2.15::
22,951 0015b 48 89 c1      mov rcx, rax
22,952 0015e 48 ff c2      inc rdx
22,953 00161 48 2b cd      sub rcx, rbp
22,954 00164 f2 0f f0 01   lddqu xmm0, XMMWORD PTR [rcx]
22,955 00168 f3 0f 7f 00   movdqu XMMWORD PTR [rax], xmm0
22,956 0016c f2 0f f0 49 10 lddqu xmm1, XMMWORD PTR [16+rcx]
22,957 00171 f3 0f 7f 48 10 movdqu XMMWORD PTR [16+rax], xmm1
22,958 00176 49 03 c2      add rax, r10
22,959 00179 80 3a 83      cmp BYTE PTR [rdx], -125
22,960 0017c 74 dd        je .B2.15
22,961 .B2.17::
22,962
22,963 0017e 49 3b d1      cmp rdx, r9
22,964 00181 0f 82 c5 ff  jb .B2.3
22,965 ff
22,966 .B2.18::
22,967 00187 48 8b 6c 24 20 mov rbp, QWORD PTR [32+rsp]
22,968 .B2.19::
22,969 0018c 49 2b c0      sub rax, r8
22,970 0018f 48 83 c4 38   add rsp, 56
22,971 00193 c3           ret
22,972 .B2.20::
22,973 00194 48 8b 4a 01     mov rcx, QWORD PTR [1+rdx]
22,974 00198 48 89 08      mov QWORD PTR [rax], rcx
22,975 0019b 48 03 c5      add rax, rbp
22,976 0019e ff c5         inc ebp
22,977 001a0 48 03 d5      add rdx, rbp
22,978 001a3 eb d9         jmp .B2.17
22,979 .B2.21::
22,980 001a5 41 c1 ea 08     shr r10d, 8
22,981 001a9 48 83 c2 04   add rdx, 4
22,982 001ad f7 dd        neg ebp
22,983 001af 49 f7 da      neg r10
22,984 001b2 4c 03 d0      add r10, rax
22,985 001b5 8d 4c 2d 24   lea ecx, DWORD PTR [36+rbp+rbp]
22,986 001b9 f2 41 0f f0 02 lddqu xmm0, XMMWORD PTR [r10]
22,987 001be f3 0f 7f 00   movdqu XMMWORD PTR [rax], xmm0
22,988 001c2 48 03 c1      add rax, rcx
22,989 001c5 eb b7        jmp .B2.17
22,990 .B2.22::
22,991 001c7 44 89 d1      mov ecx, r10d
22,992 001ca 83 e1 03      and ecx, 3
22,993 001cd 75 4d        jne .B2.26
22,994 .B2.23::
22,995 001cf 03 ed         add ebp, ebp
22,996 001d1 41 89 eb      mov r11d, ebp
22,997 001d4 89 e9      mov ecx, ebp
22,998 001d6 41 c1 eb 03  shr r11d, 3
22,999 001da 41 f7 db      neg r11d
23,000 001dd 41 83 c3 03   add r11d, 3
23,001 001e1 49 03 d3    add rdx, r11
23,002 001e4 41 bb ff ff  mov r11d, -1
23,003 ff
23,004 001ea 41 d3 eb      shr r11d, cl
23,005 001ed 45 23 d3    and r10d, r11d
23,006 001f0 41 c1 ea 04  shr r10d, 4
23,007 .B2.24::
```

```

23,008 001f4 48 89 c5      mov rbp, rax
23,009 001f7 48 ff c2      inc rdx
23,010 001fa 49 2b ea      sub rbp, r10
23,011 001fd 48 8b 4d 00    mov rcx, QWORD PTR [rbp]
23,012 00201 48 89 08      mov QWORD PTR [rax], rcx
23,013 00204 f2 0f f0 45 08 lddqu xmm0, XMMWORD PTR [8+rbp]
23,014 00209 f3 0f 7f 40 08 movdqu XMMWORD PTR [8+rax], xmm0
23,015 0020e 48 83 c0 18    add rax, 24
23,016 00212 80 3a 83      cmp BYTE PTR [rdx], -125
23,017 00215 74 dd        je .B2.24
23,018 00217 e9 62 ff ff ff jmp .B2.17
23,019 .B2.26::
23,020 0021c c1 e1 03      shl ecx, 3
23,021 0021f 41 bb ff ff ff
23,022 ff                mov r11d, -1
23,023 00225 41 d3 eb      shr r11d, cl
23,024 00228 f7 dd      neg ebp
23,025 0022a 45 23 d3      and r10d, r11d
23,026 0022d 83 c5 10      add ebp, 16
23,027 00230 41 c1 ea 04  shr r10d, 4
23,028 00234 49 f7 da      neg r10
23,029 00237 4c 03 d0      add r10, rax
23,030 0023a c1 e9 03      shr ecx, 3
23,031 0023d f7 d9      neg ecx
23,032 0023f 83 c1 04      add ecx, 4
23,033 00242 f2 41 0f f0 02 lddqu xmm0, XMMWORD PTR [r10]
23,034 00247 f3 0f 7f 00    movdqu XMMWORD PTR [rax], xmm0
23,035 0024b 48 03 c5      add rax, rbp
23,036 0024e 48 03 d1      add rdx, rcx
23,037 00251 e9 28 ff ff ff jmp .B2.17
23,038 00256 0f 1f 00 0f 1f
23,039 80 00 00 00 00    ALIGN      16
23,040 .B2.27::
23,041 ?Decompress@@YA_KPEAD0_K@Z ENDP
23,042 _TEXT      ENDS
23,043 */
23,044
23,045 /*
23,046 ; 'Ryuugan+' decompression loop, 239-39+6=518 bytes long, 143 instructions long:
23,047 ; mark_description "Intel(R) C++ Compiler XE for applications running on Intel(R) 64, Version 15.0.0.108 Build 20140726";
23,048 ; mark_description "-TP -O3 -archSSE4.1 -D_N_XMM -D_N_prefetch_4096 -D_N_HIGH_PRIORITY -D_icl_mumbo_jumbo_ -Facs";
23,049 ; 12 branches:
23,050 ; unconditional: 6 'jmp'
23,051 ; conditional: 2 'jne'
23,052 ; 2 'je'
23,053 ; 1 'jbe'
23,054 ; 1 'ja'
23,055
23,056 .B2.3::
23,057 00039 45 8b 18      mov r11d, DWORD PTR [r8]
23,058 0003c 45 89 da      mov r10d, r11d
23,059 0003f 44 89 dd      mov ebp, r11d
23,060 00042 41 83 e2 0f    and r10d, 15
23,061 00046 83 e5 0c      and ebp, 12
23,062 00049 41 83 fa 03    cmp r10d, 3
23,063 0004d 0f 85 1e 01 00    jne .B2.11
23,064 00
23,065 .B2.4::

```

```

23,066 00053 41 0f b6 eb    movzx ebp, r11b
23,067 00057 c1 ed 04      shr ebp, 4
23,068 0005a 83 fd 08      cmp ebp, 8
23,069 0005d 77 71        ja .B2.8
23,070 .B2.5::
23,071 0005f 85 ed          test ebp, ebp
23,072 00061 74 14        je .B2.7
23,073 .B2.6::
23,074 00063 49 8b 48 01    mov rcx, QWORD PTR [1+r8]
23,075 00067 48 89 08      mov QWORD PTR [rax], rcx
23,076 0006a 48 03 c5      add rax, rbp
23,077 0006d ff c5      inc ebp
23,078 0006f 4c 03 c5      add r8, rbp
23,079 00072 e9 bf 01 00 00 jmp .B2.16
23,080 .B2.7::
23,081 00077 41 c1 eb 08      shr r11d, 8
23,082 0007b 41 83 e3 07      and r11d, 7
23,083 0007f 45 89 da      mov r10d, r11d
23,084 00082 41 83 e2 03      and r10d, 3
23,085 00086 41 8d 6a 03      lea ebp, DWORD PTR [3+r10]
23,086 0008a 42 8d 0c d5 1b    lea ecx, DWORD PTR [27+r10*8]
23,087 00 00 00          neg rbp
23,088 00092 48 f7 dd      neg r10d
23,089 00095 41 f7 da      add rbp, r8
23,090 00098 49 03 e8      add r10d, 6
23,091 0009b 41 83 c2 06      add r8, r10
23,092 0009f 4d 03 c2      mov rbp, QWORD PTR [1+rbp]
23,093 000a2 48 8b 6d 01      shr rbp, cl
23,094 000a6 48 d3 ed      neg rbp
23,095 000a9 48 f7 dd      add rbp, rax
23,096 000ac 48 03 e8
23,097 000af 42 0f b6 4c 1c    movzx ecx, BYTE PTR [32+rsp+r11]
23,098 20
23,099 000b5 f2 0f f0 45 00    lddqu xmm0, XMMWORD PTR [rbp]
23,100 000ba f3 0f 7f 00      movdqu XMMWORD PTR [rax], xmm0
23,101 000be f2 0f f0 4d 10    lddqu xmm1, XMMWORD PTR [16+rbp]
23,102 000c3 f3 0f 7f 48 10    movdqu XMMWORD PTR [16+rax], xmm0
23,103 000c8 48 03 c1      add rax, rcx
23,104 000cb e9 66 01 00 00 jmp .B2.16
23,105 .B2.8::
23,106 000d0 83 fd 09      cmp ebp, 9
23,107 000d3 76 25      jbe .B2.10
23,108 .B2.9::
23,109 000d5 41 c1 eb 08      shr r11d, 8
23,110 000d9 49 83 c0 04      add r8, 4
23,111 000dd f7 dd      neg ebp
23,112 000df 49 f7 db      neg r11
23,113 000e2 4c 03 d8      add r11, rax
23,114 000e5 8d 4c 2d 24      lea ecx, DWORD PTR [36+rbp+rbp]
23,115 000e9 f2 41 0f f0 03      lddqu xmm0, XMMWORD PTR [r11]
23,116 000ee f3 0f 7f 00      movdqu XMMWORD PTR [rax], xmm0
23,117 000f2 48 03 c1      add rax, rcx
23,118 000f5 e9 3c 01 00 00 jmp .B2.16
23,119 .B2.10::
23,120 000fa 41 c1 eb 08      shr r11d, 8
23,121 000fe 41 83 e3 07      and r11d, 7
23,122 00102 45 89 da      mov r10d, r11d
23,123 00105 41 83 e2 03      and r10d, 3

```

```

23,124 00109 41 8d 6a 03      lea ebp, DWORD PTR [3+r10]
23,125 0010d 42 8d 0c d5 1b      lea ecx, DWORD PTR [27+r10*8]
23,126 00 00 00      neg rbp
23,127 00115 48 f7 dd      neg r10d
23,128 00118 41 f7 da      add rbp, r8
23,129 0011b 49 03 e8      add r10d, 6
23,130 0011e 41 83 c2 06      add r8, r10
23,131 00122 4d 03 c2      mov rbp, QWORD PTR [1+rbp]
23,132 00125 48 8b 6d 01      shr rbp, cl
23,133 00129 48 d3 ed      neg rbp
23,134 0012c 48 f7 dd      add rbp, rax
23,135 0012f 48 03 e8
23,136 00132 42 0f b6 4c 1c      movzx ecx, BYTE PTR [40+rsp+r11]
23,137 28      lddqu xmm0, XMMWORD PTR [rbp]
23,138 00138 f2 0f f0 45 00      movdqu XMMWORD PTR [rax], xmm0
23,139 0013d f3 0f 7f 00      lddqu xmm1, XMMWORD PTR [16+rbp]
23,140 00141 f2 0f f0 4d 10      movdqu XMMWORD PTR [16+rax], xmm1
23,141 00146 f3 0f 7f 48 10      lddqu xmm2, XMMWORD PTR [32+rbp]
23,142 0014b f2 0f f0 55 20      movdqu XMMWORD PTR [32+rax], xmm2
23,143 00150 f3 0f 7f 50 20      lddqu xmm3, XMMWORD PTR [48+rbp]
23,144 00155 f2 0f f0 5d 30      movdqu XMMWORD PTR [48+rax], xmm3
23,145 0015a f3 0f 7f 58 30      lddqu xmm4, XMMWORD PTR [64+rbp]
23,146 0015f f2 0f f0 65 40      movdqu XMMWORD PTR [64+rax], xmm4
23,147 00164 f3 0f 7f 60 40      add rax, rcx
23,148 00169 48 03 c1      jmp .B2.16
23,149 0016c e9 c5 00 00 00
23,150 .B2.11::
23,151 00171 41 83 fa 0c      cmp r10d, 12
23,152 00175 74 7b      je .B2.15
23,153 .B2.12::
23,154 00177 44 89 d9      mov ecx, r11d
23,155 0017a 83 e1 03      and ecx, 3
23,156 0017d 75 3c      jne .B2.14
23,157 .B2.13::
23,158 0017f 03 ed      add ebp, ebp
23,159 00181 41 ba ff ff ff      mov r10d, -1
23,160 ff
23,161 00187 89 e9      mov ecx, ebp
23,162 00189 41 d3 ea      shr r10d, cl
23,163 0018c 45 23 da      and r11d, r10d
23,164 0018f 41 c1 eb 04      shr r11d, 4
23,165 00193 49 f7 db      neg r11
23,166 00196 4c 03 d8      add r11, rax
23,167 00199 c1 ed 03      shr ebp, 3
23,168 0019c f7 dd      neg ebp
23,169 0019e 83 c5 04      add ebp, 4
23,170 001a1 4d 8b 13      mov r10, QWORD PTR [r11]
23,171 001a4 4c 89 10      mov QWORD PTR [rax], r10
23,172 001a7 f2 41 0f f0 43      lddqu xmm0, XMMWORD PTR [8+r11]
23,173 08      movdqu XMMWORD PTR [8+rax], xmm0
23,174 001ad f3 0f 7f 40 08      add r8, rbp
23,175 001b2 4c 03 c5      add rax, 24
23,176 001b5 48 83 c0 18      jmp .B2.16
23,177 001b9 eb 7b
23,178 .B2.14::
23,179 001bb c1 e1 03      shl ecx, 3
23,180 001be 41 ba ff ff ff      mov r10d, -1
23,181 ff

```

```

23,182 001c4 41 d3 ea      shr r10d, cl
23,183 001c7 f7 dd      neg ebp
23,184 001c9 45 23 da      and r11d, r10d
23,185 001cc 83 c5 10      add ebp, 16
23,186 001cf 41 c1 eb 04    shr r11d, 4
23,187 001d3 49 f7 db      neg r11
23,188 001d6 4c 03 d8      add r11, rax
23,189 001d9 c1 e9 03      shr ecx, 3
23,190 001dc f7 d9      neg ecx
23,191 001de 83 c1 04      add ecx, 4
23,192 001e1 f2 41 0f f0 03 lddqu xmm0, XMMWORD PTR [r11]
23,193 001e6 f3 0f 7f 00    movdqu XMMWORD PTR [rax], xmm0
23,194 001ea 48 03 c5      add rax, rbp
23,195 001ed 4c 03 c1      add r8, rcx
23,196 001f0 eb 44      jmp .B2.16
23,197 .B2.15::
23,198 001f2 44 89 d9      mov ecx, r11d
23,199 001f5 bd ff ff ff 00 mov ebp, 16777215
23,200 001fa c1 e9 03      shr ecx, 3
23,201 001fd 83 e1 08      and ecx, 8
23,202 00200 d3 ed      shr ebp, cl
23,203 00202 41 23 eb      and ebp, r11d
23,204 00205 41 83 e3 3c    and r11d, 60
23,205 00209 c1 ed 07      shr ebp, 7
23,206 0020c 48 f7 dd      neg rbp
23,207 0020f 48 03 e8      add rbp, rax
23,208 00212 c1 e9 03      shr ecx, 3
23,209 00215 f7 d9      neg ecx
23,210 00217 41 d1 eb      shr r11d, 1
23,211 0021a 83 c1 03      add ecx, 3
23,212 0021d f2 0f f0 45 00    lddqu xmm0, XMMWORD PTR [rbp]
23,213 00222 f3 0f 7f 00      movdqu XMMWORD PTR [rax], xmm0
23,214 00226 f2 0f f0 4d 10 lddqu xmm1, XMMWORD PTR [16+rbp]
23,215 0022b f3 0f 7f 48 10 movdqu XMMWORD PTR [16+rax], xmm1
23,216 00230 49 03 c3      add rax, r11
23,217 00233 4c 03 c1      add r8, rcx
23,218 .B2.16::
23,219 00236 4d 3b c1      cmp r8, r9
23,220 00239 0f 82 fa fd ff  jb .B2.3
23,221 ff
23,222 */
23,223
23,224 /*
23,225 ; 'Washigan+' (3xQWORD -> 1xQWORD+1xXMMWORD) decompression loop, 1a9-1e+6=401 bytes long, 120 instructions long:
23,226 ; mark_description "Intel(R) C++ Intel(R) 64 Compiler XE for applications running on Intel(R) 64, Version 15.0.0.108 Build 20140";
23,227 ; mark_description "-TP -O3 -QxSSE4.1 -D_N_XMM -D_N_prefetch_4096 -D_N_HIGH_PRIORITY -D_icl_mumbo_jumbo_ -Facs";
23,228
23,229 .B2.3::
23,230 0001e 44 8b 1a      mov r11d, DWORD PTR [rdx]
23,231 00021 45 89 da      mov r10d, r11d
23,232 00024 44 89 dd      mov ebp, r11d
23,233 00027 41 83 e2 0f    and r10d, 15
23,234 0002b 83 e5 0c      and ebp, 12
23,235 0002e 41 83 fa 03      cmp r10d, 3
23,236 00032 0f 85 a9 00 00  jne .B2.13
23,237 00
23,238 .B2.4::
23,239 00038 41 0f b6 eb      movzx ebp, r11b

```

```
23,240 0003c c1 ed 04 shr ebp, 4
23,241 0003f 83 fd 08 cmp ebp, 8
23,242 00042 77 36 ja .B2.8
23,243 .B2.5::
23,244 00044 85 ed test ebp, ebp
23,245 00046 75 1e jne .B2.7
23,246 .B2.6::
23,247 00048 41 0f b7 cb movzx ecx, r11w
23,248 0004c 48 83 c2 02 add rdx, 2
23,249 00050 c1 e9 08 shr ecx, 8
23,250 00053 48 f7 d9 neg rcx
23,251 00056 48 03 c8 add rcx, rax
23,252 00059 8b 29 mov ebp, DWORD PTR [rcx]
23,253 0005b 89 28 mov DWORD PTR [rax], ebp
23,254 0005d 48 83 c0 03 add rax, 3
23,255 00061 e9 40 01 00 00 jmp .B2.18
23,256 .B2.7::
23,257 00066 48 8b 4a 01 mov rcx, QWORD PTR [1+rdx]
23,258 0006a 48 89 08 mov QWORD PTR [rax], rcx
23,259 0006d 48 03 c5 add rax, rbp
23,260 00070 ff c5 inc ebp
23,261 00072 48 03 d5 add rdx, rbp
23,262 00075 e9 2c 01 00 00 jmp .B2.18
23,263 .B2.8::
23,264 0007a 83 fd 09 cmp ebp, 9
23,265 0007d 76 25 jbe .B2.10
23,266 .B2.9::
23,267 0007f 41 c1 eb 08 shr r11d, 8
23,268 00083 48 83 c2 04 add rdx, 4
23,269 00087 f7 dd neg ebp
23,270 00089 49 f7 db neg r11
23,271 0008c 4c 03 d8 add r11, rax
23,272 0008f 8d 4c 2d 24 lea ecx, DWORD PTR [36+rbp+rbp]
23,273 00093 f2 41 0f f0 03 lddqu xmm0, XMMWORD PTR [r11]
23,274 00098 f3 0f 7f 00 movdqu XMMWORD PTR [rax], xmm0
23,275 0009c 48 03 c1 add rax, rcx
23,276 0009f e9 02 01 00 00 jmp .B2.18
23,277 .B2.10::
23,278 000a4 8b 4a 01 mov ecx, DWORD PTR [1+rdx]
23,279 000a7 41 89 ca mov r10d, ecx
23,280 000aa 41 c1 ea 03 shr r10d, 3
23,281 000ae 33 ed xor ebp, ebp
23,282 000b0 83 e1 07 and ecx, 7
23,283 000b3 49 f7 da neg r10
23,284 000b6 45 33 db xor r11d, r11d
23,285 000b9 4c 03 d0 add r10, rax
23,286 .B2.11::
23,287 000bc ff c5 inc ebp
23,288 000be f2 43 0f f0 04 lddqu xmm0, XMMWORD PTR [r11+r10]
23,289 13
23,290 000c4 f3 41 0f 7f 04 movdqu XMMWORD PTR [r11+rax], xmm0
23,291 03
23,292 000ca 41 83 c3 10 add r11d, 16
23,293 000ce 3b e9 cmp ebp, ecx
23,294 000d0 76 ea jbe .B2.11
23,295 .B2.12::
23,296 000d2 c1 e5 04 shl ebp, 4
23,297 000d5 48 83 c2 05 add rdx, 5
```



```

23,298 000d9 48 03 c5      add rax, rbp
23,299 000dc e9 c5 00 00 00 jmp .B2.18
23,300 .B2.13::
23,301 000e1 41 83 fa 0c      cmp r10d, 12
23,302 000e5 74 7b          je .B2.17
23,303 .B2.14::
23,304 000e7 44 89 d9      mov ecx, r11d
23,305 000ea 83 e1 03      and ecx, 3
23,306 000ed 75 3c          jne .B2.16
23,307 .B2.15::
23,308 000ef 03 ed          add ebp, ebp
23,309 000f1 41 ba ff ff ff
23,310 ff                  mov r10d, -1
23,311 000f7 89 e9          mov ecx, ebp
23,312 000f9 41 d3 ea      shr r10d, cl
23,313 000fc 45 23 da      and r11d, r10d
23,314 000ff 41 c1 eb 04    shr r11d, 4
23,315 00103 49 f7 db      neg r11
23,316 00106 4c 03 d8      add r11, rax
23,317 00109 c1 ed 03      shr ebp, 3
23,318 0010c f7 dd          neg ebp
23,319 0010e 83 c5 04      add ebp, 4
23,320 00111 4d 8b 13      mov r10, QWORD PTR [r11]
23,321 00114 4c 89 10      mov QWORD PTR [rax], r10
23,322 00117 f2 41 0f f0 43
23,323 08                  lddqu xmm0, XMMWORD PTR [8+r11]
23,324 0011d f3 0f 7f 40 08 movdqu XMMWORD PTR [8+rax], xmm0
23,325 00122 48 03 d5      add rdx, rbp
23,326 00125 48 83 c0 18    add rax, 24
23,327 00129 eb 7b          jmp .B2.18
23,328 .B2.16::
23,329 0012b c1 e1 03      shl ecx, 3
23,330 0012e 41 ba ff ff ff
23,331 ff                  mov r10d, -1
23,332 00134 41 d3 ea      shr r10d, cl
23,333 00137 f7 dd          neg ebp
23,334 00139 45 23 da      and r11d, r10d
23,335 0013c 83 c5 10      add ebp, 16
23,336 0013f 41 c1 eb 04    shr r11d, 4
23,337 00143 49 f7 db      neg r11
23,338 00146 4c 03 d8      add r11, rax
23,339 00149 c1 e9 03      shr ecx, 3
23,340 0014c f7 d9          neg ecx
23,341 0014e 83 c1 04      add ecx, 4
23,342 00151 f2 41 0f f0 03 lddqu xmm0, XMMWORD PTR [r11]
23,343 00156 f3 0f 7f 00    movdqu XMMWORD PTR [rax], xmm0
23,344 0015a 48 03 c5      add rax, rbp
23,345 0015d 48 03 d1      add rdx, rcx
23,346 00160 eb 44          jmp .B2.18
23,347 .B2.17::
23,348 00162 44 89 d9      mov ecx, r11d
23,349 00165 bd ff ff ff 00 mov ebp, 16777215
23,350 0016a c1 e9 03      shr ecx, 3
23,351 0016d 83 e1 08      and ecx, 8
23,352 00170 d3 ed          shr ebp, cl
23,353 00172 41 23 eb      and ebp, r11d
23,354 00175 41 83 e3 3c    and r11d, 60
23,355 00179 c1 ed 07      shr ebp, 7

```

```

23,356 0017c 48 f7 dd      neg rbp
23,357 0017f 48 03 e8      add rbp, rax
23,358 00182 c1 e9 03      shr ecx, 3
23,359 00185 f7 d9      neg ecx
23,360 00187 41 d1 eb      shr r11d, 1
23,361 0018a 83 c1 03      add ecx, 3
23,362 0018d f2 0f f0 45 00    lddqu xmm0, XMMWORD PTR [rbp]
23,363 00192 f3 0f 7f 00      movdqu XMMWORD PTR [rax], xmm0
23,364 00196 f2 0f f0 4d 10    lddqu xmm1, XMMWORD PTR [16+rbp]
23,365 0019b f3 0f 7f 48 10    movdqu XMMWORD PTR [16+rax], xmm1
23,366 001a0 49 03 c3      add rax, r11
23,367 001a3 48 03 d1      add rdx, rcx
23,368 .B2.18::
23,369 001a6 49 3b d1      cmp rdx, r9
23,370 001a9 0f 82 6f fe ff    jb .B2.3
23,371 ff
23,372 */
23,373
23,374 /*
23,375 ; 'Okamigan' (3xQWORD -> 1xQWORD+1xXMMWORD) decompression loop, 18c-21+6=369 bytes long, 106 instructions long:
23,376 ; mark_description "Intel(R) C++ Intel(R) 64 Compiler XE for applications running on Intel(R) 64, Version 15.0.0.108 Build 20140";
23,377 ; mark_description "-TP -O3 -QxSSE4.1 -D_N_XMM -D_N_prefetch_4096 -D_N_HIGH_PRIORITY -D_icl_mumbo_jumbo_ -Facs";
23,378
23,379 .B2.3::
23,380 00021 45 8b 10      mov r10d, DWORD PTR [r8]
23,381 00024 45 89 d3      mov r11d, r10d
23,382 00027 44 89 d5      mov ebp, r10d
23,383 0002a 41 83 e3 0f    and r11d, 15
23,384 0002e 83 e5 0c      and ebp, 12
23,385 00031 41 83 fb 03      cmp r11d, 3
23,386 00035 75 75      jne .B2.9
23,387 .B2.4::
23,388 00037 41 0f b6 ea      movzx ebp, r10b
23,389 0003b c1 ed 04      shr ebp, 4
23,390 0003e 83 fd 08      cmp ebp, 8
23,391 00041 77 44      ja .B2.8
23,392 .B2.5::
23,393 00043 85 ed      test ebp, ebp
23,394 00045 74 14      je .B2.7
23,395 .B2.6::
23,396 00047 49 8b 48 01      mov rcx, QWORD PTR [1+r8]
23,397 0004b 48 89 08      mov QWORD PTR [rax], rcx
23,398 0004e 48 03 c5      add rax, rbp
23,399 00051 ff c5      inc ebp
23,400 00053 4c 03 c5      add r8, rbp
23,401 00056 e9 2e 01 00 00    jmp .B2.16
23,402 .B2.7::
23,403 0005b 41 0f b7 ca      movzx ecx, r10w
23,404 0005f 49 83 c0 02      add r8, 2
23,405 00063 c1 e9 08      shr ecx, 8
23,406 00066 48 f7 d9      neg rcx
23,407 00069 48 03 c8      add rcx, rax
23,408 0006c f2 0f f0 01      lddqu xmm0, XMMWORD PTR [rcx]
23,409 00070 f3 0f 7f 00      movdqu XMMWORD PTR [rax], xmm0
23,410 00074 f2 0f f0 49 10    lddqu xmm1, XMMWORD PTR [16+rcx]
23,411 00079 f3 0f 7f 48 10    movdqu XMMWORD PTR [16+rax], xmm1
23,412 0007e 48 83 c0 20      add rax, 32
23,413 00082 e9 02 01 00 00    jmp .B2.16

```

```
23,414 .B2.8::
23,415 00087 41 c1 ea 08 shr r10d, 8
23,416 0008b 49 83 c0 04 add r8, 4
23,417 0008f f7 dd neg ebp
23,418 00091 49 f7 da neg r10
23,419 00094 4c 03 d0 add r10, rax
23,420 00097 8d 4c 2d 24 lea ecx, DWORD PTR [36+rbp+rbp]
23,421 0009b f2 41 0f f0 02 lddqu xmm0, XMMWORD PTR [r10]
23,422 000a0 f3 0f 7f 00 movdqu XMMWORD PTR [rax], xmm0
23,423 000a4 48 03 c1 add rax, rcx
23,424 000a7 e9 dd 00 00 00 jmp .B2.16
23,425 .B2.9::
23,426 000ac 41 83 fb 0c cmp r11d, 12
23,427 000b0 75 5e jne .B2.13
23,428 .B2.10::
23,429 000b2 41 f7 c2 30 00
23,430 00 00 test r10d, 48
23,431 000b9 75 20 jne .B2.12
23,432 .B2.11::
23,433 000bb 41 0f b7 ca movzx ecx, r10w
23,434 000bf 49 83 c0 02 add r8, 2
23,435 000c3 c1 e9 06 shr ecx, 6
23,436 000c6 48 f7 d9 neg rcx
23,437 000c9 48 03 c8 add rcx, rax
23,438 000cc 48 8b 29 mov rbp, QWORD PTR [rcx]
23,439 000cf 48 89 28 mov QWORD PTR [rax], rbp
23,440 000d2 48 83 c0 06 add rax, 6
23,441 000d6 e9 ae 00 00 00 jmp .B2.16
23,442 .B2.12::
23,443 000db 44 89 d1 mov ecx, r10d
23,444 000de 49 83 c0 04 add r8, 4
23,445 000e2 c1 e9 06 shr ecx, 6
23,446 000e5 48 f7 d9 neg rcx
23,447 000e8 48 03 c8 add rcx, rax
23,448 000eb 49 83 e2 30 and r10, 48
23,449 000ef f2 0f f0 01 lddqu xmm0, XMMWORD PTR [rcx]
23,450 000f3 f3 0f 7f 00 movdqu XMMWORD PTR [rax], xmm0
23,451 000f7 f2 0f f0 49 10 lddqu xmm1, XMMWORD PTR [16+rcx]
23,452 000fc f3 0f 7f 48 10 movdqu XMMWORD PTR [16+rax], xmm1
23,453 00101 f2 0f f0 51 20 lddqu xmm2, XMMWORD PTR [32+rcx]
23,454 00106 f3 0f 7f 50 20 movdqu XMMWORD PTR [32+rax], xmm2
23,455 0010b 49 03 c2 add rax, r10
23,456 0010e eb 79 jmp .B2.16
23,457 .B2.13::
23,458 00110 44 89 d1 mov ecx, r10d
23,459 00113 83 e1 03 and ecx, 3
23,460 00116 75 3c jne .B2.15
23,461 .B2.14::
23,462 00118 03 ed add ebp, ebp
23,463 0011a 41 bb ff ff ff
23,464 ff mov r11d, -1
23,465 00120 89 e9 mov ecx, ebp
23,466 00122 41 d3 eb shr r11d, cl
23,467 00125 45 23 d3 and r10d, r11d
23,468 00128 41 c1 ea 04 shr r10d, 4
23,469 0012c 49 f7 da neg r10
23,470 0012f 4c 03 d0 add r10, rax
23,471 00132 c1 ed 03 shr ebp, 3
```

```

23,472 00135 f7 dd      neg ebp
23,473 00137 83 c5 04      add ebp, 4
23,474 0013a 4d 8b 1a      mov r11, QWORD PTR [r10]
23,475 0013d 4c 89 18      mov QWORD PTR [rax], r11
23,476 00140 f2 41 0f f0 42      lddqu xmm0, XMMWORD PTR [8+r10]
23,477 08                      movdqu XMMWORD PTR [8+rax], xmm0
23,478 00146 f3 0f 7f 40 08      add r8, rbp
23,479 0014b 4c 03 c5      add rax, 24
23,480 0014e 48 83 c0 18      jmp .B2.16
23,481 00152 eb 35
23,482 .B2.15::
23,483 00154 c1 e1 03      shl ecx, 3
23,484 00157 41 bb ff ff ff      mov r11d, -1
23,485 ff                      shr r11d, cl
23,486 0015d 41 d3 eb      neg ebp
23,487 00160 f7 dd      and r10d, r11d
23,488 00162 45 23 d3      add ebp, 16
23,489 00165 83 c5 10      shr r10d, 4
23,490 00168 41 c1 ea 04      neg r10
23,491 0016c 49 f7 da      add r10, rax
23,492 0016f 4c 03 d0      shr ecx, 3
23,493 00172 c1 e9 03      neg ecx
23,494 00175 f7 d9      add ecx, 4
23,495 00177 83 c1 04      lddqu xmm0, XMMWORD PTR [r10]
23,496 0017a f2 41 0f f0 02      movdqu XMMWORD PTR [rax], xmm0
23,497 0017f f3 0f 7f 00      add rax, rbp
23,498 00183 48 03 c5      add r8, rcx
23,499 00186 4c 03 c1
23,500 .B2.16::
23,501 00189 4d 3b c1      cmp r8, r9
23,502 0018c 0f 82 8f fe ff      jb .B2.3
23,503 ff
23,504 */
23,505
23,506 /*
23,507 ; 'Okamigan' decompression loop, 191-21+6=374 bytes long, 108 instructions long:
23,508 ; mark_description "Intel(R) C++ Intel(R) 64 Compiler XE for applications running on Intel(R) 64, Version 15.0.0.108 Build 20140";
23,509 ; mark_description "-TP -O3 -QxSSE4.1 -D_N_XMM -D_N_prefetch_4096 -D_N_HIGH_PRIORITY -FAcs";
23,510
23,511 .B2.3::
23,512 00021 45 8b 10      mov r10d, DWORD PTR [r8]
23,513 00024 45 89 d3      mov r11d, r10d
23,514 00027 44 89 d5      mov ebp, r10d
23,515 0002a 41 83 e3 0f      and r11d, 15
23,516 0002e 83 e5 0c      and ebp, 12
23,517 00031 41 83 fb 03      cmp r11d, 3
23,518 00035 75 75      jne .B2.9
23,519 .B2.4::
23,520 00037 41 0f b6 ea      movzx ebp, r10b
23,521 0003b c1 ed 04      shr ebp, 4
23,522 0003e 83 fd 08      cmp ebp, 8
23,523 00041 77 44      ja .B2.8
23,524 .B2.5::
23,525 00043 85 ed      test ebp, ebp
23,526 00045 74 14      je .B2.7
23,527 .B2.6::
23,528 00047 49 8b 48 01      mov rcx, QWORD PTR [1+r8]
23,529 0004b 48 89 08      mov QWORD PTR [rax], rcx

```

```

23,530 0004e 48 03 c5      add rax, rbp
23,531 00051 ff c5          inc ebp
23,532 00053 4c 03 c5      add r8, rbp
23,533 00056 e9 33 01 00 00  jmp .B2.16
23,534 .B2.7::
23,535 0005b 41 0f b7 ca      movzx ecx, r10w
23,536 0005f 49 83 c0 02      add r8, 2
23,537 00063 c1 e9 08        shr ecx, 8
23,538 00066 48 f7 d9        neg rcx
23,539 00069 48 03 c8        add rcx, rax
23,540 0006c f2 0f f0 01      lddqu xmm0, XMMWORD PTR [rcx]
23,541 00070 f3 0f 7f 00      movdqu XMMWORD PTR [rax], xmm0
23,542 00074 f2 0f f0 49 10    lddqu xmm1, XMMWORD PTR [16+rcx]
23,543 00079 f3 0f 7f 48 10    movdqu XMMWORD PTR [16+rax], xmm1
23,544 0007e 48 83 c0 20      add rax, 32
23,545 00082 e9 07 01 00 00  jmp .B2.16
23,546 .B2.8::
23,547 00087 41 c1 ea 08        shr r10d, 8
23,548 0008b 49 83 c0 04      add r8, 4
23,549 0008f f7 dd          neg ebp
23,550 00091 49 f7 da          neg r10
23,551 00094 4c 03 d0        add r10, rax
23,552 00097 8d 4c 2d 24      lea ecx, DWORD PTR [36+rbp+rbp]
23,553 0009b f2 41 0f f0 02      lddqu xmm0, XMMWORD PTR [r10]
23,554 000a0 f3 0f 7f 00      movdqu XMMWORD PTR [rax], xmm0
23,555 000a4 48 03 c1        add rax, rcx
23,556 000a7 e9 e2 00 00 00  jmp .B2.16
23,557 .B2.9::
23,558 000ac 41 83 fb 0c      cmp r11d, 12
23,559 000b0 75 5e          jne .B2.13
23,560 .B2.10::
23,561 000b2 41 f7 c2 30 00      test r10d, 48
23,562 00 00          jne .B2.12
23,563 000b9 75 20          jne .B2.12
23,564 .B2.11::
23,565 000bb 41 0f b7 ca      movzx ecx, r10w
23,566 000bf 49 83 c0 02      add r8, 2
23,567 000c3 c1 e9 06        shr ecx, 6
23,568 000c6 48 f7 d9        neg rcx
23,569 000c9 48 03 c8        add rcx, rax
23,570 000cc 48 8b 29        mov rbp, QWORD PTR [rcx]
23,571 000cf 48 89 28        mov QWORD PTR [rax], rbp
23,572 000d2 48 83 c0 06      add rax, 6
23,573 000d6 e9 b3 00 00 00  jmp .B2.16
23,574 .B2.12::
23,575 000db 44 89 d1        mov ecx, r10d
23,576 000de 49 83 c0 04      add r8, 4
23,577 000e2 c1 e9 06        shr ecx, 6
23,578 000e5 48 f7 d9        neg rcx
23,579 000e8 48 03 c8        add rcx, rax
23,580 000eb 49 83 e2 30      and r10, 48
23,581 000ef f2 0f f0 01      lddqu xmm0, XMMWORD PTR [rcx]
23,582 000f3 f3 0f 7f 00      movdqu XMMWORD PTR [rax], xmm0
23,583 000f7 f2 0f f0 49 10    lddqu xmm1, XMMWORD PTR [16+rcx]
23,584 000fc f3 0f 7f 48 10    movdqu XMMWORD PTR [16+rax], xmm1
23,585 00101 f2 0f f0 51 20    lddqu xmm2, XMMWORD PTR [32+rcx]
23,586 00106 f3 0f 7f 50 20    movdqu XMMWORD PTR [32+rax], xmm2
23,587 0010b 49 03 c2        add rax, r10

```

```
23,588 0010e eb 7e jmp .B2.16
23,589 .B2.13::
23,590 00110 44 89 d1 mov ecx, r10d
23,591 00113 83 e1 03 and ecx, 3
23,592 00116 75 41 jne .B2.15
23,593 .B2.14::
23,594 00118 03 ed add ebp, ebp
23,595 0011a 41 bb ff ff ff
23,596 ff mov r11d, -1
23,597 00120 89 e9 mov ecx, ebp
23,598 00122 41 d3 eb shr r11d, cl
23,599 00125 45 23 d3 and r10d, r11d
23,600 00128 41 c1 ea 04 shr r10d, 4
23,601 0012c 49 f7 da neg r10
23,602 0012f 4c 03 d0 add r10, rax
23,603 00132 c1 ed 03 shr ebp, 3
23,604 00135 f7 dd neg ebp
23,605 00137 83 c5 04 add ebp, 4
23,606 0013a 4d 8b 1a mov r11, QWORD PTR [r10]
23,607 0013d 4c 89 18 mov QWORD PTR [rax], r11
23,608 00140 4d 8b 5a 08 mov r11, QWORD PTR [8+r10]
23,609 00144 4c 89 58 08 mov QWORD PTR [8+rax], r11
23,610 00148 4d 8b 52 10 mov r10, QWORD PTR [16+r10]
23,611 0014c 4c 89 50 10 mov QWORD PTR [16+rax], r10
23,612 00150 4c 03 c5 add r8, rbp
23,613 00153 48 83 c0 18 add rax, 24
23,614 00157 eb 35 jmp .B2.16
23,615 .B2.15::
23,616 00159 c1 e1 03 shl ecx, 3
23,617 0015c 41 bb ff ff ff
23,618 ff mov r11d, -1
23,619 00162 41 d3 eb shr r11d, cl
23,620 00165 f7 dd neg ebp
23,621 00167 45 23 d3 and r10d, r11d
23,622 0016a 83 c5 10 add ebp, 16
23,623 0016d 41 c1 ea 04 shr r10d, 4
23,624 00171 49 f7 da neg r10
23,625 00174 4c 03 d0 add r10, rax
23,626 00177 c1 e9 03 shr ecx, 3
23,627 0017a f7 d9 neg ecx
23,628 0017c 83 c1 04 add ecx, 4
23,629 0017f f2 41 0f f0 02 lddqu xmm0, XMMWORD PTR [r10]
23,630 00184 f3 0f 7f 00 movdqu XMMWORD PTR [rax], xmm0
23,631 00188 48 03 c5 add rax, rbp
23,632 0018b 4c 03 c1 add r8, rcx
23,633 .B2.16::
23,634 0018e 4d 3b c1 cmp r8, r9
23,635 00191 0f 82 8a fe ff
23,636 ff jb .B2.3
23,637 */
23,638
23,639 /*
23,640 RT:
23,641
23,642
23,643
23,644
23,645
23,646
23,647
23,648
```

```
23, 649 .....
23, 650 .....
23, 651 .....
23, 652 .....
23, 653 .....
23, 654 .....
23, 655 .....
23, 656 .....
23, 657 .....
23, 658 .....
23, 659 .....
23, 660 .....
23, 661 .....
23, 662 .....
23, 663 .....
23, 664 .....
23, 665 .....
23, 666 .....
23, 667 .....
23, 668 .....
23, 669 .....
23, 670 .....
23, 671 .....
23, 672 .....
23, 673 .....
23, 674 .....
23, 675 .....
23, 676 .....
23, 677 .....
23, 678 .....
23, 679 .....
23, 680 .....
23, 681 .....
23, 682 .....
23, 683 .....
23, 684 .....
23, 685 .....
23, 686 .....
23, 687 .....
23, 688 .....
23, 689 .....
23, 690 .....
23, 691 .....
23, 692 .....
23, 693 .....
23, 694 .....
23, 695 .....
23, 696 .....
23, 697 .....
23, 698 .....
23, 699 .....
23, 700 .....
23, 701 .....
23, 702 .....
23, 703 .....
23, 704 .....
23, 705 .....
23, 706 .....
23, 707 .....
23, 708 .....
23, 709 .....
23, 710 .....
23, 711 .....
23, 712 .....
23, 713 .....
23, 714 .....
23, 715 .....
23, 716 .....
23, 717 .....
23, 718 .....
23, 719 .....
23, 720 .....
23, 721 .....
23, 722 .....
23, 723 .....
23, 724 .....
23, 725 .....
23, 726 .....
23, 727 .....
23, 728 .....
23, 729 .....
23, 730 .....
23, 731 .....
23, 732 .....
23, 733 .....
23, 734 .....
23, 735 .....
```

```

23,818 */
23,819 // Railgun_Trolldom (the successor of Railgun_Swampshine_BailOut - avoiding second pattern comparison in BMH2 and pseudo-BMH4), copyleft 2016-Aug-19, Kaze.
23,820 // Railgun_Swampshine_BailOut, copyleft 2016-Aug-10, Kaze.

```



```

23,821 // Internet "home" page: http://www.codeproject.com/Articles/250566/Fastest-strstr-like-function-in-C
23,822 // My homepage (homeserver, often down): http://www.sanmayce.com/Railgun/
23,823 /*
23,824 !!!!!!!!!!!!!!!!!!!!!!!!!!!!! BENCHMARKING GNU's memmem vs Railgun !!!!!!!!!!!!!!!!!!!!!!!!!!!!! [
23,825 Add-on: 2016-Aug-22
23,826
23,827 Two things.
23,828
23,829 First, the fix from the last time was buggy, my apologies, now fixed, quite embarrassing since it is a simple left/right boundary check. It doesn't affect the speed,
it appears as rare pattern hit misses.
23,830 Since I don't believe in saying "sorry" but in making things right, here my attempt to further disgrace my amateurish work follows:
23,831 Two years ago, I didn't pay due attention to adding 'Swampwalker' heuristic to the Railgun_Ennearch, I mean, only quick test was done and no real proofing - this was
due not to a blunder of mine, nor carelessness, but overconfidence in my ability to write "on the fly". Stupid, indeed, however, when a coder gets momentum in writing simple
etudes he starts gaining false confidence of mastering the subject, not good for sure!
23,832 Hopefully, other coders will learn to avoid such full of neglect style.
23,833
23,834 Second, wanted to present the heaviest testbed for search i.e. memmem() functions: it benefits the benchmarking (speed in real application) as well as bug-control.
23,835
23,836 The benchmark is downloadable at my INTERNET drive:
23,837 https://1drv.ms/u/s!AmWWFXGMzDmEglwjlUtnMJrfhosK
23,838
23,839 The speed showdown has three facets:
23,840 - compares the 64bit code generated from GCC 5.10 versus Intel 15.0 compilers;
23,841 - compares four types of datasets - search speed through English texts versus genome ACGT-type data versus binary versus UTF8;
23,842 - compares the tweaked Two-Way algorithm (implemented by Eric Blake) and adopted by GLIBC as memmem() versus my Railgun_Swampshine.
23,843
23,844 Note1: The GLIBC memmem() was taken from latest (2016-08-05) glibc 2.24 tar:
23,845 https://www.gnu.org/software/libc/
23,846 Note2: Eric Blake says that he enhanced the linearity of Two-Way by adding some sublinear paths, well, Railgun is all about sublinearity, so feel free to experiment
with your own testfiles (worst-case-scenarios), just make such a file feed the compressor with it, then we will see how the LINEAR Two-Way behaves versus Railgun_Swampshine.
23,847 Note3: Just copy-and-paste 'Railgun_Swampshine' or 'Railgun_Ennearch' from the benchmark's source.
23,848
23,849 So the result on Core 2 Q9550s @2.83GHz DDR2 @666MHz / i5-2430M @3.00GHz DDR3 @666MHz:
23,850 -----
23,851 | Searcher                                | GNU/GLIBC memmem()          | Railgun_Swampshine          | Railgun_Trollidom          |
23,852 |-----|-----|-----|-----|-----|-----|-----|-----|
23,853 | Testfile\Compiler                       | Intel 15.0 | GCC 5.10          | Intel 15.0 | GCC 5.10          | Intel 15.0 | GCC 5.10          |
23,854 |-----|-----|-----|-----|-----|-----|-----|-----|
23,855 | Size: 27,703 bytes                      | 4506/- | 5330/14725         | 13198/- | 11581/15171       | 19105/22449 | 15493/21642       |
23,856 | Name: An_Interview_with_Carlos_Castaneda.TXT | | | | | | | |
23,857 | LATENCY-WISE: Number of 'mемmem()' Invocations: 308,062 | | | | | | | |
23,858 | THROUGHPUT-WISE: Number of Total bytes Traversed: 3,242,492,648 | | | | | | | |
23,859 |-----|-----|-----|-----|-----|-----|-----|-----|
23,860 | Size: 2,347,772 bytes                   | 190/- | 226/244           | 1654/- | 1729/1806         | 1794/1822 | 1743/1809         |
23,861 | Name: Gutenberg_EBook_Don_Quixote_996_(ANSI).txt | | | | | | | |
23,862 | LATENCY-WISE: Number of 'mемmem()' Invocations: 14,316,954 | | | | | | | |
23,863 | THROUGHPUT-WISE: Number of Total bytes Traversed: 6,663,594,719,173 | | | | | | | |
23,864 |-----|-----|-----|-----|-----|-----|-----|-----|
23,865 | Size: 899,425 bytes                    | 582/- | 760/816           | 3094/- | 2898/3088         | 3255/3289 | 2915/3322         |
23,866 | Name: Gutenberg_EBook_Dokoe_by_Hakucho_Masamune_(Japanese_UTF8).txt | | | | | | | |
23,867 | LATENCY-WISE: Number of 'mемmem()' Invocations: 3,465,806 | | | | | | | |
23,868 | THROUGHPUT-WISE: Number of Total bytes Traversed: 848,276,034,315 | | | | | | | |
23,869 |-----|-----|-----|-----|-----|-----|-----|-----|
23,870 | Size: 4,487,433 bytes                   | 104/- | 109/116           | 445/- | 458/417           | 450/411 | 467/425           |
23,871 | Name: Dragonfly_genome_shotgun_sequence_(ACGT_alphabet).fasta | | | | | | | |
23,872 | LATENCY-WISE: Number of 'mемmem()' Invocations: 20,540,375 | | | | | | | |
23,873 | THROUGHPUT-WISE: Number of Total bytes Traversed: 13,592,530,857,131 | | | | | | | |
23,874 |-----|-----|-----|-----|-----|-----|-----|-----|

```

23,884
23,885

23,894
33,895 Just to see how faster is Kern's 7std in decompression (its level 12 is 377,331 MB/s faster) on Core 2 Q9550 @ 3.00GHz DDR2 @666MHz:

23,895 Just to see how faster is rain's zstd in decompression (its level 12 is 377-331 MB/s faster), on core 2 Q9550S @2.83GHz DDR2 @666MHz:
23,896

```
[REDACTED]
```

```
23.897 D:\Nakamichi Kintaro++ source executables 64bit (GCC510-vs-Intel150) (TW-vs-RG) BENCHMARK>Nakamichi Kintaro++ Intel 15.0 64bit.exe Agatha Christie 85-
```

ebooks_(French)_TXT.tar

23,898 Nakamichi "Kintaro++", written by Kaze based on Nobuo Ito's LZSS source, babealicious suggestion by m^2 enforced, muffinsque suggestion by Jim Dempsey enforced.
23,899 Note1: This compile can handle files up to 1711MB

```
23,900 Note2: The matchfinder/memmem() is Railgun_Trollldom.
```

```
23.901 Current priority class is HIGH_PRIORITY_CLASS.
23.902 Compressing 32007168 bytes
```

```
23,903 1; Each rotation means 64KB are encoded; Done 100%; Compression Ratio: 3.53:1
```

```
23,904 NumberOfFullLiterals (lower-the-better): 164
23,905 NumberOfFullLiteralsHeuristic (bigger-the-better): 104222
```

23,906 Legend: WindowSizes: 1/2/3/4=Tiny/Short/Medium/Long

```
23,907 NumberOf(Tiny)Matches[Short]Window (4)[2]: 226869
23,909 NumberOf(Tiny)Matches[Short]Window (6)[2]: 226869
```

23.908 NumberOf(Short)Matches[Short]Window (8)[2]: 119810
23.909 NumberOf(Medium)Matches[Short]Window (12)[2]: 71202

```
23,910 NumberOf(Long)Matches[Short]Window (16)[2]: 31955
```

23,911 NumberOfMaxLongMatches[ShortWindow (24)][2]: 7078
23,912 NumberOfTinyMatches[MediumWindow (5)][3]: 257313

```
23,913 NumberOf(Short)Matches[Medium]Window (9)[3]: 526493
```

```
23,914 NumberOf(Medium)Matches[Medium]Window (13)[3]: 285579
23,915 NumberOf(Long)Matches[Medium]Window (17)[3]: 158873
```

```
23,916 NumberOf(MaxLong)Matches[Medium]Window (24)[3]: 51276
```

```
23,917 NumberOf(Tiny)Matches[Long]Window (6)[4]: 41075
23,918 NumberOf(Short)Matches[Long]Window (10)[4]: 240454
```

```
23,919 NumberOf(Short)Matches[Long]Window (10): 23225
23,919 NumberOf(Medium)Matches[Long]Window (14): 258653
```

```
23, 920 NumberOf(Long)Matches[Long]Window (18) [4]: 2090007
23, 921 NumberOf(Max[Long)Matches[Long]Window / 24) [4]: 1000000
```

```
23,921 NumberOfMaxLongMatches[6000]window (24)[4]. 150525
23,922 RAM-to-RAM performance: 605 bytes/s.
```

```
23,923 Compressed to 9076876 bytes.
```

Latency	Latency-wise	Number of members	Invocations	102,091,852
23,924	Latency-wise	Number of members	Invocations	102,091,852

Listing: Nakamichi Ryuugan-ditto-1TB btree.c; Last version: 2021-Aug-30; Font: MxPlus ToshibaTxL2 8x16.ttf; Downloadable at: www.sanmayce.com/Nakamichi/Kaidanji.zip

```
23,978 // &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
23,979 // i-(PRIMALposition-1)+(count-1) >= 0
23,980 // &pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4
23,981
23,982 // "FIX" from 2014-Apr-27:
23,983 // Because (count-1) is negative, above fours are reduced to next twos:
23,984 // i-(PRIMALposition-1)+(count-1) >= 0
23,985 // &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
23,986 // The line below is BUGGY:
23,987 //if ( (i-(PRIMALposition-1) >= 0) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) && (&pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4) ) {
23,988 // The line below is NOT OKAY, in fact so stupid, grrr, not a blunder, not carelessness, but overconfidence in writing "on the fly":
23,989 //if ( ((signed int)(i-(PRIMALposition-1)+(count-1)) >= 0) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) ) {
23,990 // FIX from 2016-Aug-10 (two times failed to do simple boundary checks, pfu):
23,991 if ( ((signed int)(i-(PRIMALposition-1)) >= 0) && (&pbTarget[i-(PRIMALposition-1)]+((PRIMALlengthCANDIDATE-4+1)-1) <= pbTargetMax - 4) ) {
23,992     if ( *(uint32_t *)&pbTarget[i-(PRIMALposition-1)] == *(uint32_t *)&(pbPattern-(PRIMALposition-1))) { // This fast check ensures not missing a match (for
remainder) when going under 0 in loop below:
23,993         count = PRIMALlengthCANDIDATE-4+1;
23,994         while ( count > 0 && *(uint32_t *)&(pbPattern-(PRIMALposition-1)+count-1) == *(uint32_t *)&(&pbTarget[i-(PRIMALposition-1)]+(count-1)) )
23,995             count = count-4;
23,996         if ( count <= 0 ) return(pbTarget+i-(PRIMALposition-1));
23,997     }
23,998 }
23,999 }
24,000 ...
24,001 */
24,002 // Railgun_Swampshine_BailOut, copyleft 2014-Jan-31, Kaze.
24,003 // Caution: For better speed the case 'if (cbPattern==1)' was removed, so Pattern must be longer than 1 char.
24,004 #define NeedleThreshold2vs4swampLITE 9+10 // Should be bigger than 9. BMH2 works up to this value (inclusive), if bigger then BMH4 takes over. Should be <=255
otherwise the 0!1 BMH2 should be used.
24,005 char * Railgun_Troldom_64 (char * pbTarget, char * pbPattern, uint64_t cbTarget, uint32_t cbPattern)
24,006 {
24,007     char * pbTargetMax = pbTarget + cbTarget;
24,008     register uint32_t ulHashPattern;
24,009     //signed long count;
24,010     signed long long count; // 2020-Jan-11
24,011
24,012     unsigned char bm_Horspool_Order2[256*256]; // Bitwise soon...
24,013     unsigned char bm_Horspool_Order2bitwise[(256*256)>>3]; // Bitwise soon...
24,014     //uint32_t i, Gulliver;
24,015     uint64_t i, Gulliver;
24,016
24,017     //uint32_t PRIMALposition, PRIMALpositionCANDIDATE;
24,018     //uint32_t PRIMALlength, PRIMALlengthCANDIDATE;
24,019     //uint32_t j, FoundAtPosition;
24,020
24,021     uint64_t PRIMALposition, PRIMALpositionCANDIDATE;
24,022     uint64_t PRIMALlength, PRIMALlengthCANDIDATE;
24,023     uint64_t j, FoundAtPosition;
24,024
24,025     // Quadruplet [
24,026         //char * pbTargetMax = pbTarget + cbTarget;
24,027         //register unsigned long ulHashPattern;
24,028         unsigned long ulHashTarget;
24,029         //unsigned long count;
24,030         unsigned long countSTATIC;
24,031         unsigned char SINGLET;
24,032         unsigned long Quadruplet2nd;
24,033         unsigned long Quadruplet3rd;
```

```

24,034 unsigned long Quadruplet4th;
24,035 unsigned long AdvanceHopperGrass;
24,036 // Quadruplet ]
24,037
24,038 // 2020-Jan-11 [
24,039 // uint64_t A=3123123123, B=5123123123;
24,040 //if ((signed int)A > 0) printf("(signed int)3billion OK\n"); else printf("(signed int)3billion Bug\n");
24,041 //if ((signed int)B > 0) printf("(signed int)5billion OK\n"); else printf("(signed int)5billion Bug\n");
24,042 //if ((signed long long)A > 0) printf("(signed long long)3billion OK\n"); else printf("(signed long long)3billion Bug\n");
24,043 //if ((signed long long)B > 0) printf("(signed long long)5billion OK\n"); else printf("(signed long long)5billion Bug\n");
24,044
24,045 //(signed int)3billion Bug
24,046 //(signed int)5billion OK
24,047 //(signed long long)3billion OK
24,048 //(signed long long)5billion OK
24,049 // 2020-Jan-11 ]
24,050
24,051 GLOBAL_Railgun_INVOCATIONS++; // 2020-Jan-29
24,052 GLOBAL_Railgun_INVOCATIONS_ARRAY[cbPattern]++; // 2020-Jan-29
24,053
24,054 if (cbPattern > cbTarget) return(NULL);
24,055
24,056 #ifdef LITE
24,057 return(NULL); // 2020-Feb-14
24,058 #endif
24,059
24,060 if ( cbPattern<4 ) {
24,061     // SSE2 i.e. 128bit Assembly rules here, Mischa knows best:
24,062     // ...
24,063     pbTarget = pbTarget+cbPattern;
24,064     ulHashPattern = ( (*char*)(pbPattern)<<8 ) + *(pbPattern+(cbPattern-1));
24,065     if ( cbPattern==3 ) {
24,066         for ( ;; ) {
24,067             if ( ulHashPattern == ( (*char*)(pbTarget-3)<<8 ) + *(pbTarget-1) ) {
24,068                 if ( (*char*)(pbPattern+1) == (*char*)(pbTarget-2) ) return((pbTarget-3));
24,069             }
24,070             if ( (char)(ulHashPattern>>8) != *(pbTarget-2) ) {
24,071                 pbTarget++;
24,072                 if ( (char)(ulHashPattern>>8) != *(pbTarget-2) ) pbTarget++;
24,073             }
24,074             pbTarget++;
24,075             if (pbTarget > pbTargetMax) return(NULL);
24,076         }
24,077     } else {
24,078     }
24,079     for ( ;; ) {
24,080         if ( ulHashPattern == ( (*char*)(pbTarget-2)<<8 ) + *(pbTarget-1) ) return((pbTarget-2));
24,081         if ( (char)(ulHashPattern>>8) != *(pbTarget-1) ) pbTarget++;
24,082         pbTarget++;
24,083         if (pbTarget > pbTargetMax) return(NULL);
24,084     }
24,085 } else { //if ( cbPattern<4 )
24,086     if ( cbPattern<=NeedleThreshold2vs4swampLITE ) {
24,087
24,088 // This is the awesome 'Railgun_Quadruplet', it did outperform EVERYWHERE the fastest strstr (back in old GLIBCes ~2003, by the Dutch hacker Stephen R. van den Berg),
suitable for short haystacks ~100bytes.
24,089 // Caution: For better speed the case 'if (cbPattern==1)' was removed, so Pattern must be longer than 1 char.
24,090 // char * Railgun_Quadruplet (char * pbTarget, char * pbPattern, unsigned long cbTarget, unsigned long cbPattern)

```

```

24,091 // ...
24,092 //   if (cbPattern > cbTarget) return(NULL);
24,093 //} else { //if ( cbPattern<4)
24,094 if (cbTarget<777) // This value is arbitrary(don't know how exactly), it ensures(at least must) better performance than 'Boyer_Moore_Horspool'.
24,095 {
24,096     pbTarget = pbTarget+cbPattern;
24,097     ulHashPattern = *(unsigned long *) (pbPattern);
24,098 //     countSTATIC = cbPattern-1;
24,099
24,100     //SINGLET = *(char *) (pbPattern);
24,101     SINGLET = ulHashPattern & 0xFF;
24,102     Quadruplet2nd = SINGLET<<8;
24,103     Quadruplet3rd = SINGLET<<16;
24,104     Quadruplet4th = SINGLET<<24;
24,105
24,106     for ( ;; )
24,107     {
24,108         AdvanceHopperGrass = 0;
24,109         ulHashTarget = *(unsigned long *) (pbTarget-cbPattern);
24,110
24,111         if ( ulHashPattern == ulHashTarget ) { // Three unnecessary comparisons here, but 'AdvanceHopperGrass' must be calculated - it has a higher priority.
24,112 //             count = countSTATIC;
24,113 //             while ( count && *(char *) (pbPattern+1+(countSTATIC-count)) == *(char *) (pbTarget-cbPattern+1+(countSTATIC-count)) ) {
24,114 //                 if ( countSTATIC==AdvanceHopperGrass+count && SINGLET != *(char *) (pbTarget-cbPattern+1+(countSTATIC-count)) ) AdvanceHopperGrass++;
24,115 //                 count--;
24,116 //             }
24,117             count = cbPattern-1;
24,118             while ( count && *(char *) (pbPattern+(cbPattern-count)) == *(char *) (pbTarget-count) ) {
24,119                 if ( cbPattern-1==AdvanceHopperGrass+count && SINGLET != *(char *) (pbTarget-count) ) AdvanceHopperGrass++;
24,120                 count--;
24,121             }
24,122             if ( count == 0 ) return((pbTarget-cbPattern));
24,123         } else { // The goal here: to avoid memory accesses by stressing the registers.
24,124             if ( Quadruplet2nd != (ulHashTarget & 0x0000FF00) ) {
24,125                 AdvanceHopperGrass++;
24,126                 if ( Quadruplet3rd != (ulHashTarget & 0x00FF0000) ) {
24,127                     AdvanceHopperGrass++;
24,128                     if ( Quadruplet4th != (ulHashTarget & 0xFF000000) ) AdvanceHopperGrass++;
24,129                 }
24,130             }
24,131         }
24,132         AdvanceHopperGrass++;
24,133
24,134         pbTarget = pbTarget + AdvanceHopperGrass;
24,135         if (pbTarget > pbTargetMax)
24,136             return(NULL);
24,137     }
24,138 } else if (cbTarget<77777) { // The warmup/overhead is lowered from 64K down to 8K, however the bitwise additional instructions quickly start hurting the
24,139 // throughput/traversal.
24,140 // The below bitwise 0!1 BMH2 gives 1427 bytes/s for 'Don_Quixote' with Intel:
24,141 // The below bitwise 0!1 BMH2 gives 1242 bytes/s for 'Don_Quixote' with GCC:
24,142 //     } else { //if ( cbPattern<4 )
24,143 //         if ( cbPattern<=NeedleThreshold2vs4Decumanus ) {
24,144 //             // BMH order 2, needle should be >=4:
24,145             ulHashPattern = *(uint32_t *) (pbPattern); // First four bytes
24,146             //for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]=0;}
24,147             for (i=0; i < (256*256)>>3; i++) {bm_Horspool_Order2bitwise[i]=0;}

```

```

24,148 //for (i=0; i < cbPattern-1; i++) bm_Horspool_Order2[*(unsigned short *) (pbPattern+i)]=1;
24,149 for (i=0; i < cbPattern-2+1; i++) bm_Horspool_Order2bitwise[*(unsigned short *) (pbPattern+i)>>3]= bm_Horspool_Order2bitwise[*(unsigned short
24,150 *) (pbPattern+i)>>3] | (1<<((*(unsigned short *) (pbPattern+i))&0x7));
24,151 i=0;
24,152 while (i <= cbTarget-cbPattern) {
24,153     Gulliver = 1; // 'Gulliver' is the skip
24,154     //if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+cbPattern-1-1]] != 0 ) {
24,155         if ( ( bm_Horspool_Order2bitwise[*(unsigned short *)&pbTarget[i+cbPattern-1-1]]>>3] & (1<<((*(unsigned short *)&pbTarget[i+cbPattern-1-
24,156 1])&0x7)) ) != 0 ) {
24,157             //if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+cbPattern-1-2]] == 0 ) Gulliver = cbPattern-(2-1)-2; else {
24,158                 if ( ( bm_Horspool_Order2bitwise[*(unsigned short *)&pbTarget[i+cbPattern-1-2]]>>3] & (1<<((*(unsigned short
24,159 *)&pbTarget[i+cbPattern-1-2])&0x7)) ) == 0 ) Gulliver = cbPattern-(2-1)-2; else {
24,160                     if ( *(uint32_t *)&pbTarget[i] == ulHashPattern) { // This fast check ensures not missing a match (for remainder) when going
24,161 under 0 in loop below:
24,162                         count = cbPattern-4+1;
24,163                         while ( count > 0 && *(uint32_t *) (pbPattern+count-1) == *(uint32_t *) (&pbTarget[i]+(count-1)) )
24,164                             count = count-4;
24,165                         if ( count <= 0 ) return(pbTarget+i);
24,166                     }
24,167                 } else Gulliver = cbPattern-(2-1);
24,168                 i = i + Gulliver;
24,169                 //GlobalI++; // Comment it, it is only for stats.
24,170             }
24,171             return(NULL);
24,172         } else { // if ( cbPattern<=NeedleThreshold2vs4Decumanus )
24,173             // BMH order 2, needle should be >=4:
24,174             ulHashPattern = *(uint32_t *) (pbPattern); // First four bytes
24,175             for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]=0;}
24,176             for (i=0; i < cbPattern-1; i++) bm_Horspool_Order2[*(unsigned short *) (pbPattern+i)]=1;
24,177             i=0;
24,178             while (i <= cbTarget-cbPattern) {
24,179                 Gulliver = 1; // 'Gulliver' is the skip
24,180                 if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+cbPattern-1-1]] != 0 ) {
24,181                     if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+cbPattern-1-2]] == 0 ) Gulliver = cbPattern-(2-1)-2; else {
24,182                         if ( *(uint32_t *)&pbTarget[i] == ulHashPattern) { // This fast check ensures not missing a match (for remainder) when going
24,183 under 0 in loop below:
24,184                             count = cbPattern-4+1;
24,185                             while ( count > 0 && *(uint32_t *) (pbPattern+count-1) == *(uint32_t *) (&pbTarget[i]+(count-1)) )
24,186                                 count = count-4;
24,187                             if ( count <= 0 ) return(pbTarget+i);
24,188                         }
24,189                     } else Gulliver = cbPattern-(2-1);
24,190                     i = i + Gulliver;
24,191                     //GlobalI++; // Comment it, it is only for stats.
24,192                 }
24,193                 return(NULL);
24,194             }
24,195             // Slower than Swampshine's simple 0!1 segment:
24,196             /*
24,197             PRIMAlength=0;
24,198             for (i=0+(1); i < cbPattern-2+1+(1)-(1); i++) { // -(1) because the last BB order 2 has no counterpart(s)
24,199                 FoundAtPosition = cbPattern;
24,200                 PRIMAlength=0;
24,201                 while ( PRIMAlengthCANDIDATE <= (FoundAtPosition-1) ) {
24,202                     j = PRIMAlengthCANDIDATE + 1;

```

```

24,201     while ( j <= (FoundAtPosition-1) ) {
24,202         if ( *(unsigned short *) (pbPattern+PRIMALpositionCANDIDATE-(1)) == *(unsigned short *) (pbPattern+j-(1)) ) FoundAtPosition = j;
24,203         j++;
24,204     }
24,205     PRIMALpositionCANDIDATE++;
24,206 }
24,207 PRIMALlengthCANDIDATE = (FoundAtPosition-1)-i+(2);
24,208 if (PRIMALlengthCANDIDATE >= PRIMALlength) {PRIMALposition=i; PRIMALlength = PRIMALlengthCANDIDATE;}
24,209 }
24,210 PRIMALlengthCANDIDATE = cbPattern;
24,211 cbPattern = PRIMALlength;
24,212 pbPattern = pbPattern + (PRIMALposition-1);
24,213 if (cbPattern<4) {
24,214     cbPattern = PRIMALlengthCANDIDATE;
24,215     pbPattern = pbPattern - (PRIMALposition-1);
24,216 }
24,217 if (cbPattern == PRIMALlengthCANDIDATE) {
24,218     // BMH order 2, needle should be >=4:
24,219     ulHashPattern = *(uint32_t *) (pbPattern); // First four bytes
24,220     for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]=0;}
24,221     for (i=0; i < cbPattern-1; i++) bm_Horspool_Order2[*(unsigned short *) (pbPattern+i)]=1;
24,222     i=0;
24,223     while (i <= cbTarget-cbPattern) {
24,224         Gulliver = 1; // 'Gulliver' is the skip
24,225         if ( bm_Horspool_Order2[*(unsigned short *) &pbTarget[i+cbPattern-1-1]] != 0 ) {
24,226             if ( bm_Horspool_Order2[*(unsigned short *) &pbTarget[i+cbPattern-1-1-2]] == 0 ) Gulliver = cbPattern-(2-1)-2; else {
24,227                 if ( *(uint32_t *) &pbTarget[i] == ulHashPattern) { // This fast check ensures not missing a match (for remainder) when going
under 0 in loop below:
24,228                     count = cbPattern-4+1;
24,229                     while ( count > 0 && *(uint32_t *) (pbPattern+count-1) == *(uint32_t *) (&pbTarget[i]+(count-1)) )
24,230                         count = count-4;
24,231                     if ( count <= 0 ) return(pbTarget+i);
24,232                 }
24,233             }
24,234         } else Gulliver = cbPattern-(2-1);
24,235         i = i + Gulliver;
24,236         //GlobalI++; // Comment it, it is only for stats.
24,237     }
24,238     return(NULL);
24,239 } else { //if (cbPattern == PRIMALlengthCANDIDATE) {
24,240 // BMH Order 2 [
24,241     ulHashPattern = *(uint32_t *) (pbPattern); // First four bytes
24,242     for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]= cbPattern-1;} // cbPattern-(Order-1) for Horspool; 'memset' if not optimized
24,243     // The above 'for' gives 1424 bytes/s for 'Don_Quixote' with Intel:
24,244     // The above 'for' gives 1431 bytes/s for 'Don_Quixote' with GCC:
24,245     // The below 'memset' gives 1389 bytes/s for 'Don_Quixote' with Intel:
24,246     // The below 'memset' gives 1432 bytes/s for 'Don_Quixote' with GCC:
24,247     //memset(&bm_Horspool_Order2[0], cbPattern-1, 256*256); // Why why? It is 1700:1000 slower than above 'for'!
24,248     for (i=0; i < cbPattern-1; i++) bm_Horspool_Order2[*(unsigned short *) (pbPattern+i)]=i; // Rightmost appearance/position is needed
24,249     i=0;
24,250     while (i <= cbTarget-cbPattern) {
24,251         Gulliver = bm_Horspool_Order2[*(unsigned short *) &pbTarget[i+cbPattern-1-1]];
24,252         if ( Gulliver != cbPattern-1 ) { // CASE #2: if equal means the pair (char order 2) is not found i.e. Gulliver remains intact, skip the whole
pattern and fall back (Order-1) chars i.e. one char for Order 2
24,253         if ( Gulliver == cbPattern-2 ) { // CASE #1: means the pair (char order 2) is found
24,254             if ( *(uint32_t *) &pbTarget[i] == ulHashPattern) {
24,255                 count = cbPattern-4+1;
24,256                 while ( count > 0 && *(uint32_t *) (pbPattern+count-1) == *(uint32_t *) (&pbTarget[i]+(count-1)) )

```



```

24,257                                     count = count-4;
24,258 // If we miss to hit then no need to compare the original: Needle
24,259 if ( count <= 0 ) {
24,260 // I have to add out-of-range checks...
24,261 // i-(PRIMALposition-1) >= 0
24,262 // &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
24,263 // i-(PRIMALposition-1)+(count-1) >= 0
24,264 // &pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4
24,265
24,266 // "FIX" from 2014-Apr-27:
24,267 // Because (count-1) is negative, above fours are reduced to next twos:
24,268 // i-(PRIMALposition-1)+(count-1) >= 0
24,269 // &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
24,270 // The line below is BUGGY:
24,271 //if ( (i-(PRIMALposition-1) >= 0) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) && (&pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4) ) {
24,272 // The line below is NOT OKAY, in fact so stupid, grrr, not a blunder, not carelessness, but overconfidence in writing "on the fly":
24,273 //if ( ((signed int)(i-(PRIMALposition-1)+(count-1)) >= 0) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) ) {
24,274 // FIX from 2016-Aug-10 (two times failed to do simple boundary checks, pfu):
24,275 if ( ((signed long long)(i-(PRIMALposition-1)) >= 0) && (&pbTarget[i-(PRIMALposition-1)]+((PRIMALlengthCANDIDATE-4+1)-1) <= pbTargetMax - 4) ) { // 2020-jan-11
24,276     if ( *(uint32_t *)&pbTarget[i-(PRIMALposition-1)] == *(uint32_t *)(&pbPattern-(PRIMALposition-1))) { // This fast check ensures not missing a match (for
remainder) when going under 0 in loop below:
24,277         count = PRIMALlengthCANDIDATE-4+1;
24,278         while ( count > 0 && *(uint32_t *)(&pbPattern-(PRIMALposition-1)+count-1) == *(uint32_t *)(&pbTarget[i-(PRIMALposition-1)]+(count-1)) )
24,279             count = count-4;
24,280         if ( count <= 0 ) return(pbTarget+i-(PRIMALposition-1));
24,281     }
24,282 }
24,283 }
24,284     }
24,285     Gulliver = 1;
24,286 } else
24,287     Gulliver = cbPattern - Gulliver - 2; // CASE #3: the pair is found and not as suffix i.e. rightmost position
24,288 }
24,289 i = i + Gulliver;
24,290 //GlobalI++; // Comment it, it is only for stats.
24,291 }
24,292 return(NULL);
24,293 // BMH Order 2 ]
24,294 } //if (cbPattern == PRIMALlengthCANDIDATE) {
24,295 /*
24,296
24,297 /*
24,298 So the result on Core 2 Q9550s @2.83GHz:
24,299
24,300 | testfile\Searcher | GNU/GLIBC memmem() | Railgun_Swampshine | Railgun_Troldom |
24,301 |-----|-----|-----|-----|-----|
24,302 | Compiler | Intel 15.0 | GCC 5.10 | Intel 15.0 | GCC 5.10 | Intel 15.0 | GCC 5.10 |
24,303 |-----|-----|-----|-----|-----|
24,304 | The_Project_Gutenberg_EBook_of_Don | 190 | 226 | 1654 | 1729 | 1147 | 1764 |
24,305 | _Quixote_996_(ANSI).txt | | | | | | |
24,306 | 2,347,772 bytes | | | | | | |
24,307 |-----|-----|-----|-----|-----|
24,308 | The_Project_Gutenberg_EBook_of_Dokoe | 582 | 760 | 3094 | 2898 | 2410 | 3036 |
24,309 | _by_Hakucho_Masamune_(Japanese_UTF-8).txt | | | | | | |
24,310 | 899,425 bytes | | | | | | |
24,311 |-----|-----|-----|-----|-----|
24,312 | Dragonfly_genome_shotgun_sequence | 104 | 109 | 445 | 458 | 484 | 553 |
24,313 | _(ACGT_alphabet).fasta | | | | | | |

```

```

24,314 | 4,487,433 bytes
24,315 |-----|
24,316 | LAOTZU_Wu_Wei_(BINARY).pdf | 99 | 144 | 629 | 580 | 185 | 570 |
24,317 | 954,035 bytes |
24,318 |-----|
24,319 Below segment (when compiled with Intel) is very slow, see Railgun_Troldom two sub-columns above, compared to GCC:
24,320 */
24,321 /*
24,322 // BMH Order 2 [
24,323         ulHashPattern = *(uint32_t *) (pbPattern); // First four bytes
24,324         for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i] = (cbPattern-1);} // cbPattern-(Order-1) for Horspool; 'memset' if not optimized
24,325         // The above 'for' is translated by Intel as:
24,326 // .B5.21::
24,327 // 0013f 83 c0 40      add eax, 64
24,328 // 00142 66 0f 7f 44 14
24,329 // 60      movdqa XMMWORD PTR [96+rsp+rdx], xmm0
24,330 // 00148 3d 00 00 01 00      cmp eax, 65536
24,331 // 0014d 66 0f 7f 44 14
24,332 // 70      movdqa XMMWORD PTR [112+rsp+rdx], xmm0
24,333 // 00153 66 0f 7f 84 14
24,334 // 80 00 00 00      movdqa XMMWORD PTR [128+rsp+rdx], xmm0
24,335 // 0015c 66 0f 7f 84 14
24,336 // 90 00 00 00      movdqa XMMWORD PTR [144+rsp+rdx], xmm0
24,337 // 00165 89 c2      mov edx, eax
24,338 // 00167 72 d6      jb .B5.21
24,339         //memset(&bm_Horspool_Order2[0], cbPattern-1, 256*256); // Why why? It is 1700:1000 slower than above 'for'!?
24,340         // The above 'memset' is translated by Intel as:
24,341 // 00127 41 b8 00 00 01
24,342 // 00      mov r8d, 65536
24,343 // 0012d 44 8b 26      mov r12d, DWORD PTR [rsi]
24,344 // 00130 e8 fc ff ff ff      call _intel_fast_memset
24,345         // ! The problem is that 256*256, 64KB, is already too much, going bitwise i.e. 8KB is not that better, when 'cbPattern-1' is bigger than 255 - an
unsigned char - then
24,346         // we must switch to 0!1 table i.e. present or not. Since we are in 'if ( cbPattern<=NeedleThreshold2vs4swampLITE ) {' branch and
NeedleThreshold2vs4swampLITE, by default, is 19 - it is okay to use 'memset'. !
24,347         for (i=0; i < cbPattern-1; i++) bm_Horspool_Order2[*(unsigned short *) (pbPattern+i)]=i; // Rightmost appearance/position is needed
24,348         i=0;
24,349         while (i <= cbTarget-cbPattern) {
24,350             Gulliver = bm_Horspool_Order2[*(unsigned short *) &pbTarget[i+cbPattern-1]];
24,351             if ( Gulliver != cbPattern-1 ) { // CASE #2: if equal means the pair (char order 2) is not found i.e. Gulliver remains intact, skip the whole
pattern and fall back (Order-1) chars i.e. one char for Order 2
24,352             if ( Gulliver == cbPattern-2 ) { // CASE #1: means the pair (char order 2) is found
24,353                 if ( *(uint32_t *) &pbTarget[i] == ulHashPattern ) {
24,354                     count = cbPattern-4+1;
24,355                     while ( count > 0 && *(uint32_t *) (pbPattern+count-1) == *(uint32_t *) (&pbTarget[i]+(count-1)) )
24,356                         count = count-4;
24,357                     if ( count <= 0 ) return(pbTarget+i);
24,358                 }
24,359                 Gulliver = 1;
24,360             } else
24,361                 Gulliver = cbPattern - Gulliver - 2; // CASE #3: the pair is found and not as suffix i.e. rightmost position
24,362             }
24,363             i = i + Gulliver;
24,364             //GlobalI++; // Comment it, it is only for stats.
24,365         }
24,366         return(NULL);
24,367 // BMH Order 2 ]
24,368 */

```

```

24,369 // Above fragment in Assembly:
24,370 /*
24,371 ; mark_description "Intel(R) C++ Intel(R) 64 Compiler XE for applications running on Intel(R) 64, Version 15.0.0.108 Build 20140";
24,372 ; mark_description "-O3 -QxSSE2 -D_N_XMM -D_N_prefetch_4096 -D_N_Branchfull -D_N_HIGH_PRIORITY -FA";
24,373 ALIGN      16
24,374 .B6.1::      ; Preds .B6.0
24,375     push      rbx                      ;3435.1
24,376     push      r13                     ;3435.1
24,377     push      r15                     ;3435.1
24,378     push      rbp                      ;3435.1
24,379     mov       eax, 65592                ;3435.1
24,380     call      __chkstk                 ;3435.1
24,381     sub       rsp, 65592                ;3435.1
24,382     cmp       r9d, r8d                 ;3460.18
24,383     ja       .B6.25                    ;3460.18
24,384           ; Prob 28%
24,384           ; LOE rdx rcx rbx rsi rdi r12 r14 r8d r9d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
24,385 .B6.3::      ; Preds .B6.1
24,386     mov       r13d, DWORD PTR [rdx]      ;3491.33
24,387     lea       ebp, DWORD PTR [-1+r9]     ;3492.67
24,388     movzx     eax, bpl                  ;3492.67
24,389     xor       r10d, r10d                ;3492.4
24,390     movd      xmm0, eax                 ;3492.67
24,391     xor       eax, eax                  ;3492.4
24,392     punpcklbw xmm0, xmm0               ;3492.67
24,393     punpcklwd xmm0, xmm0               ;3492.67
24,394     punpckldq xmm0, xmm0               ;3492.67
24,395     punpcklqdq xmm0, xmm0              ;3492.67
24,396           ; LOE rdx rcx rbx rsi rdi r10 r12 r14 eax ebp r8d r9d r13d xmm0 xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
24,397 .B6.4::      ; Preds .B6.4 .B6.3
24,398     add       eax, 64                   ;3492.4
24,399     movdqa    XMMWORD PTR [48+rsp+r10], xmm0 ;3492.33
24,400     cmp       eax, 65536                 ;3492.4
24,401     movdqa    XMMWORD PTR [64+rsp+r10], xmm0 ;3492.33
24,402     movdqa    XMMWORD PTR [80+rsp+r10], xmm0 ;3492.33
24,403     movdqa    XMMWORD PTR [96+rsp+r10], xmm0 ;3492.33
24,404     mov      r10d, eax                 ;3492.4
24,405     jnb     .B6.4                     ;3492.4
24,406           ; Prob 99%
24,406           ; LOE rdx rcx rbx rsi rdi r10 r12 r14 eax ebp r8d r9d r13d xmm0 xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
24,407 .B6.5::      ; Preds .B6.4
24,408     test      ebp, ebp                  ;3515.28
24,409     je       .B6.12                    ;3515.28
24,410           ; Prob 50%
24,410           ; LOE rdx rcx rbx rsi rdi r12 r14 ebp r8d r9d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
24,411 .B6.6::      ; Preds .B6.5
24,412     mov      eax, 1                     ;3515.4
24,413     lea      r11d, DWORD PTR [-1+r9]    ;3515.4
24,414     mov      r15d, r11d                ;3515.4
24,415     xor      r10d, r10d                ;3515.4
24,416     shr     r15d, 1                    ;3515.4
24,417     test    r15d, r15d                 ;3515.4
24,418     jbe     .B6.10                      ;3515.4
24,419           ; Prob 15%
24,419           ; LOE rdx rcx rbx rsi rdi r12 r14 eax ebp r8d r9d r10d r11d r13d r15d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
24,420 .B6.8::      ; Preds .B6.6 .B6.8
24,421     lea      eax, DWORD PTR [r10+r10]    ;3515.36
24,422     movzx     ebx, WORD PTR [rax+rdx]     ;3515.75
24,423     mov      BYTE PTR [48+rsp+rbx], al   ;3515.36
24,424     lea      eax, DWORD PTR [1+r10+r10] ;3515.36
24,425     inc      r10d                      ;3515.4
24,426     cmp     r10d, r15d                 ;3515.4

```

```

24,427      movzx    ebx, WORD PTR [rax+rdx]                ;3515.75
24,428      mov     BYTE PTR [48+rsp+rbx], al              ;3515.36
24,429      jb      .B6.8                                ;3515.4
24,430      ; LOE rdx rcx rsi rdi r12 r14 ebp r8d r9d r10d r11d r13d r15d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
24,431 .B6.9::      ; Preds .B6.8
24,432      lea     eax, DWORD PTR [1+r10+r10]             ;3515.4
24,433      ; LOE rdx rcx rbx rsi rdi r12 r14 eax ebp r8d r9d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
24,434 .B6.10::    ; Preds .B6.9 .B6.6
24,435      dec     eax                                    ;3515.36
24,436      cmp     eax, r11d                             ;3515.4
24,437      jae     .B6.12                                ;3515.4
24,438      ; LOE rax rcx rcx rbx rsi rdi r12 r14 ebp r8d r9d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
24,439 .B6.11::    ; Preds .B6.10
24,440      movzx    r10d, WORD PTR [rax+rdx]              ;3515.75
24,441      mov     BYTE PTR [48+rsp+r10], al             ;3515.36
24,442      ; LOE rdx rcx rbx rsi rdi r12 r14 ebp r8d r9d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
24,443 .B6.12::    ; Preds .B6.5 .B6.10 .B6.11
24,444      xor     r10d, r10d                            ;3516.4
24,445      lea     r15d, DWORD PTR [-3+r9]                ;3522.27
24,446      movsxd   r15, r15d                            ;3522.7
24,447      sub     r8d, r9d                             ;3517.16
24,448      lea     r11d, DWORD PTR [-2+r9]                ;3520.32
24,449      ; LOE rdx rcx rsi rdi r12 r14 r15 ebp r8d r9d r10d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
24,450 .B6.13::    ; Preds .B6.12 .B6.24
24,451      lea     eax, DWORD PTR [-2+r9+r10]             ;3518.78
24,452      movzx    ebx, WORD PTR [rax+rcx]              ;3518.55
24,453      movzx    eax, BYTE PTR [48+rsp+rbx]           ;3518.16
24,454      cmp     eax, ebp                             ;3519.32
24,455      je      .B6.24                                ;3519.32
24,456      ; LOE rdx rcx rsi rdi r12 r14 r15 eax ebp r8d r9d r10d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
24,457 .B6.14::    ; Preds .B6.13
24,458      cmp     eax, r11d                             ;3520.32
24,459      jne     .B6.23                                ;3520.32
24,460      ; LOE rdx rcx rsi rdi r12 r14 r15 eax ebp r8d r9d r10d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
24,461 .B6.15::    ; Preds .B6.14
24,462      mov     eax, r10d                             ;3521.25
24,463      add     rax, rcx                              ;3521.25
24,464      cmp     r13d, DWORD PTR [rax]                 ;3521.40
24,465      je      .B6.17                                ;3521.40
24,466      ; LOE rax rdx rcx rsi rdi r12 r14 r15 ebp r8d r9d r10d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
24,467 .B6.16::    ; Preds .B6.26 .B6.15
24,468      mov     eax, 1                                ;3527.6
24,469      jmp     .B6.24                                ;3527.6
24,470      ; LOE rdx rcx rsi rdi r12 r14 r15 eax ebp r8d r9d r10d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
24,471 .B6.17::    ; Preds .B6.15
24,472      mov     rbx, r15                              ;3522.7
24,473      test    r15, r15                             ;3523.23
24,474      jle     .B6.22                                ;3523.23
24,475      ; LOE rax rdx rcx rbx rsi rdi r12 r14 r15 ebp r8d r9d r10d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
24,476 .B6.18::    ; Preds .B6.17
24,477      mov     QWORD PTR [32+rsp], rsi               ;
24,478      ; LOE rax rdx rcx rbx rdi r12 r14 r15 ebp r8d r9d r10d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
24,479 .B6.19::    ; Preds .B6.20 .B6.18
24,480      mov     esi, DWORD PTR [-1+rbx+rdx]           ;3523.58
24,481      cmp     esi, DWORD PTR [-1+rbx+rax]           ;3523.79
24,482      jne     .B6.26                                ;3523.79
24,483      ; LOE rax rdx rcx rbx rdi r12 r14 r15 ebp r8d r9d r10d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
24,484 .B6.20::    ; Preds .B6.19

```

```

24,485      add      rbx, -4                      ;3524.22
24,486      test     rbx, rbx                    ;3523.23
24,487      jg       .B6.19                      ;3523.23
24,488                      ; LOE rax rdx rcx rbx rdi r12 r14 r15 ebp r8d r9d r10d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
24,489 .B6.21::      ; Preds .B6.20
24,490      mov      rsi, QWORD PTR [32+rsp]      ;
24,491                      ; LOE rax rbx rsi rdi r12 r14 xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
24,492 .B6.22::      ; Preds .B6.17 .B6.21
24,493      add      rsp, 65592                  ;3525.32
24,494      pop      rbp                        ;3525.32
24,495      pop      r15                       ;3525.32
24,496      pop      r13                       ;3525.32
24,497      pop      rbx                      ;3525.32
24,498      ret                                ;3525.32
24,499                      ; LOE
24,500 .B6.23::      ; Preds .B6.14
24,501      neg      eax                      ;3529.17
24,502      add      eax, r9d                  ;3529.17
24,503      add      eax, -2                   ;3529.40
24,504                      ; LOE rdx rcx rsi rdi r12 r14 r15 ebp r8d r9d r10d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
24,505 .B6.24::      ; Preds .B6.16 .B6.23 .B6.13
24,506      add      r10d, eax                 ;3531.13
24,507      cmp      r10d, r8d                ;3517.25
24,508      jbe      .B6.13                   ;3517.25
24,509                      ; LOE rdx rcx rsi rdi r12 r14 r15 ebp r8d r9d r10d r11d r13d xmm6 xmm7 xmm8 xmm9 xmm10 xmm11 xmm12 xmm13 xmm14 xmm15
24,510 .B6.25::      ; Preds .B6.1 .B6.24
24,511      xor      eax, eax                 ;3534.10
24,512      add      rsp, 65592               ;3534.10
24,513      pop      rbp                      ;3534.10
24,514      pop      r15                     ;3534.10
24,515      pop      r13                     ;3534.10
24,516      pop      rbx                     ;3534.10
24,517      ret                                ;3534.10
24,518                      ; LOE
24,519 .B6.26::      ; Preds .B6.19           ; Infreq
24,520      mov      rsi, QWORD PTR [32+rsp]    ;
24,521      jmp      .B6.16                   ; Prob 100%
24,522 */
24,523
24,524 // GCC 5.10; >gcc -O3 -m64 -fomit-frame-pointer
24,525 /*
24,526 Railgun_Trolldom:
24,527 pushq   %r15
24,528 .seh_pushreg %r15
24,529 movl    $65592, %eax
24,530 pushq   %r14
24,531 .seh_pushreg %r14
24,532 pushq   %r13
24,533 .seh_pushreg %r13
24,534 pushq   %r12
24,535 .seh_pushreg %r12
24,536 pushq   %rbp
24,537 .seh_pushreg %rbp
24,538 pushq   %rdi
24,539 .seh_pushreg %rdi
24,540 pushq   %rsi
24,541 .seh_pushreg %rsi
24,542 pushq   %rbx

```

```
24,543 .seh_pushreg    %rbx
24,544 call    ___chkstk_ms
24,545 subq    %rax, %rsp
24,546 .seh_stackalloc 65592
24,547 .seh_endprologue
24,548 cmpl    %r9d, %r8d
24,549 movq    %rcx, %rbx
24,550 movq    %rdx, %rdi
24,551 movl    %r8d, %r12d
24,552 movl    %r9d, %esi
24,553 jb      .L118
24,554 movl    (%rdx), %ebp
24,555 leal    -1(%r9), %edx
24,556 movl    $65536, %r8d
24,557 leaq    48(%rsp), %rcx
24,558 movzbl %dl, %edx
24,559 call    memset
24,560 movl    %esi, %r11d
24,561 subl    $1, %r11d
24,562 je      .L119
24,563 xorl    %eax, %eax
24,564 .p2align 4,,10
24,565 .L113:
24,566 movzwl (%rdi,%rax), %edx
24,567 movb    %al, 48(%rsp,%rdx)
24,568 addq    $1, %rax
24,569 cmpl    %eax, %r11d
24,570 ja      .L113
24,571 .L112:
24,572 leal    -4(%rsi), %r9d
24,573 movl    %r12d, %r8d
24,574 xorl    %edx, %edx
24,575 leal    -3(%rsi), %eax
24,576 shrl    $2, %r9d
24,577 subl    %esi, %r8d
24,578 leal    -2(%rsi), %r10d
24,579 movslq %eax, %r14
24,580 negq    %r9
24,581 movl    %eax, 44(%rsp)
24,582 leaq    -1(%r14), %r15
24,583 salq    $2, %r9
24,584 leaq    (%rdi,%r14), %r13
24,585 jmp     .L117
24,586 .p2align 4,,10
24,587 .L130:
24,588 movl    %r10d, %eax
24,589 subl    %ecx, %eax
24,590 cmpl    %r10d, %ecx
24,591 je      .L129
24,592 .L114:
24,593 addl    %eax, %edx
24,594 cmpl    %r8d, %edx
24,595 ja      .L118
24,596 .L117:
24,597 leal    (%rdx,%r10), %eax
24,598 movzwl (%rbx,%rax), %eax
24,599 movzbl 48(%rsp,%rax), %ecx
24,600 cmpl    %r11d, %ecx
```

```
24,601 jne .L130
24,602 movl %r11d, %eax
24,603 addl %eax, %edx
24,604 cmpl %r8d, %edx
24,605 jbe .L117
24,606 .L118:
24,607 xorl %eax, %eax
24,608 jmp .L128
24,609 .p2align 4,,10
24,610 .L129:
24,611 movl %edx, %ecx
24,612 movl $1, %eax
24,613 leaq (%rbx,%rcx), %r12
24,614 cmpl (%r12), %ebp
24,615 jne .L114
24,616 movl 44(%rsp), %esi
24,617 testl %esi, %esi
24,618 jle .L124
24,619 movl (%r12,%r15), %esi
24,620 cmpl %esi, (%rdi,%r15)
24,621 jne .L114
24,622 addq %r14, %rcx
24,623 xorl %eax, %eax
24,624 addq %rbx, %rcx
24,625 jmp .L116
24,626 .p2align 4,,10
24,627 .L132:
24,628 movl -5(%r13,%rax), %esi
24,629 subq $4, %rax
24,630 cmpl -1(%rcx,%rax), %esi
24,631 jne .L131
24,632 .L116:
24,633 cmpq %rax, %r9
24,634 jne .L132
24,635 .L124:
24,636 movq %r12, %rax
24,637 .L128:
24,638 addq $65592, %rsp
24,639 popq %rbx
24,640 popq %rsi
24,641 popq %rdi
24,642 popq %rbp
24,643 popq %r12
24,644 popq %r13
24,645 popq %r14
24,646 popq %r15
24,647 ret
24,648 .p2align 4,,10
24,649 .L131:
24,650 movl $1, %eax
24,651 jmp .L114
24,652 .L119:
24,653 xorl %r11d, %r11d
24,654 jmp .L112
24,655 */
24,656
24,657 } //if (cbTarget<777)
24,658
```

Listing: Nakamichi Ryuugan-ditto-1TB btree.c; Last version: 2021-Aug-30; Font: MxPlus ToshibaTxL2 8x16.ttf; Downloadable at: www.sanmayce.com/Nakamichi/Kaidanji.zip

Listing: Nakamichi Ryuugan-ditto-1TB btree.c; Last version: 2021-Aug-30; Font: MxPlus ToshibaTxL2 8x16.ttf; Downloadable at: www.sanmayce.com/Nakamichi/Kaidanji.zip

```
24,765 Step 31_00: {}1234567890qwertyuiopasdfghjklz[xc][v?] ! For position #31 the initial boundaries are PRIMALpositionCANDIDATE=LeftBoundary=31,
RightBoundary=FoundAtPosition-1, the CANDIDATE PRIMAL string length is RightBoundary-LeftBoundary+(2)=(33-1)-31+(2)=03 !
24,766 Step 31_01: 1234567890qwertyuiopasdfghjklz[{xc}][v?] ! Searching for 'xc', FoundAtPosition = 33, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(33-1)-31+(2)=03
!
24,767     if (PRIMALlengthCANDIDATE >= PRIMALlength) {PRIMALposition=PRIMALpositionCANDIDATE; PRIMALlength = PRIMALlengthCANDIDATE;}
24,768     Result:
24,769     PRIMALposition=01 PRIMALlength=33, NewNeedle = '1234567890qwertyuiopasdfghjklzxcv'
24,770
24,771
24,772     PRIMALlength=00; FoundAtPosition=33;
24,773 Step 01_00: {}[vv]vvvvvvvvvvvvvvvvvvvvvvvv[vv] ! For position #01 the initial boundaries are PRIMALpositionCANDIDATE=LeftBoundary=01,
RightBoundary=FoundAtPosition-1, the CANDIDATE PRIMAL string length is RightBoundary-LeftBoundary+(2)=(33-1)-01+(2)=33 !
24,774 Step 01_01: [{v(v)}v]vvvvvvvvvvvvvvvvvvvvvvvv ! Searching for 'vv', FoundAtPosition = 02, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(02-1)-01+(2)=02
!
24,775     if (PRIMALlengthCANDIDATE >= PRIMALlength) {PRIMALposition=PRIMALpositionCANDIDATE; PRIMALlength = PRIMALlengthCANDIDATE;}
24,776 Step 02_00: {}v[vv]vvvvvvvvvvvvvvvvvvvvvvvv[vv] ! For position #02 the initial boundaries are PRIMALpositionCANDIDATE=LeftBoundary=02,
RightBoundary=FoundAtPosition-1, the CANDIDATE PRIMAL string length is RightBoundary-LeftBoundary+(2)=(33-1)-02+(2)=32 !
24,777 Step 02_01: v[{v(v)}v]vvvvvvvvvvvvvvvvvvvvvvvv ! Searching for 'vv', FoundAtPosition = 03, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(03-1)-02+(2)=02
!
24,778     if (PRIMALlengthCANDIDATE >= PRIMALlength) {PRIMALposition=PRIMALpositionCANDIDATE; PRIMALlength = PRIMALlengthCANDIDATE;}
24,779 ...
24,780 Step 31_00: {}vvvvvvvvvvvvvvvvvvvvvvvvvvvv[vv][v?] ! For position #31 the initial boundaries are PRIMALpositionCANDIDATE=LeftBoundary=31,
RightBoundary=FoundAtPosition-1, the CANDIDATE PRIMAL string length is RightBoundary-LeftBoundary+(2)=(33-1)-31+(2)=03 !
24,781 Step 31_01: vvvvvvvvvvvvvvvvvvvvvvvvvvv[{v(v)}v] ! Searching for 'vv', FoundAtPosition = 32, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(32-1)-31+(2)=02
!
24,782     if (PRIMALlengthCANDIDATE >= PRIMALlength) {PRIMALposition=PRIMALpositionCANDIDATE; PRIMALlength = PRIMALlengthCANDIDATE;}
24,783     Result:
24,784     PRIMALposition=31 PRIMALlength=02, NewNeedle = 'vv'
24,785
24,786
24,787     PRIMALlength=00; FoundAtPosition=33;
24,788 Step 01_00: {}[vv]vvvvvvvvBOOMSHAKALAKAvvvvvvvvv[vv] ! For position #01 the initial boundaries are PRIMALpositionCANDIDATE=LeftBoundary=01,
RightBoundary=FoundAtPosition-1, the CANDIDATE PRIMAL string length is RightBoundary-LeftBoundary+(2)=(33-1)-01+(2)=33 !
24,789 Step 01_01: [{v(v)}v]vvvvvvvvBOOMSHAKALAKAvvvvvvvvv ! Searching for 'vv', FoundAtPosition = 02, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(02-1)-01+(2)=02
!
24,790     if (PRIMALlengthCANDIDATE >= PRIMALlength) {PRIMALposition=PRIMALpositionCANDIDATE; PRIMALlength = PRIMALlengthCANDIDATE;}
24,791 Step 02_00: {}v[vv]vvvvvvvvBOOMSHAKALAKAvvvvvvvvv[vv] ! For position #02 the initial boundaries are PRIMALpositionCANDIDATE=LeftBoundary=02,
RightBoundary=FoundAtPosition-1, the CANDIDATE PRIMAL string length is RightBoundary-LeftBoundary+(2)=(33-1)-02+(2)=32 !
24,792 Step 02_01: v[{v(v)}v]vvvvvvvvBOOMSHAKALAKAvvvvvvvvv ! Searching for 'vv', FoundAtPosition = 03, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(03-1)-02+(2)=02
!
24,793     if (PRIMALlengthCANDIDATE >= PRIMALlength) {PRIMALposition=PRIMALpositionCANDIDATE; PRIMALlength = PRIMALlengthCANDIDATE;}
24,794 ...
24,795 Step 09_00: {}vvvvvvvv[vv]BOOMSHAKALAKAvvvvvvvvv[vv] ! For position #09 the initial boundaries are PRIMALpositionCANDIDATE=LeftBoundary=09,
RightBoundary=FoundAtPosition-1, the CANDIDATE PRIMAL string length is RightBoundary-LeftBoundary+(2)=(33-1)-09+(2)=25 !
24,796 Step 09_01: vvvvvvvv[{vv}]BOOMSHAKALAKA(vv)vvvvvvvv ! Searching for 'vv', FoundAtPosition = 24, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(24-1)-09+(2)=16
!
24,797 Step 09_02: vvvvvvvv[v{v}]BOOMSHAKALAKA[vv]vvvvvvvv ! Searching for 'vB', FoundAtPosition = 24, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(24-1)-09+(2)=16
!
24,798 Step 09_03: vvvvvvvv[vv]{BO}OMSHAKALAKA[vv]vvvvvvvv ! Searching for 'BO', FoundAtPosition = 24, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(24-1)-09+(2)=16
!
24,799 Step 09_04: vvvvvvvv[vv]B{OO}MSHAKALAKA[vv]vvvvvvvv ! Searching for 'OO', FoundAtPosition = 24, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(24-1)-09+(2)=16
!
24,800 Step 09_05: vvvvvvvv[vv]BO{OM}SHAKALAKA[vv]vvvvvvvv ! Searching for 'OM', FoundAtPosition = 24, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(24-1)-09+(2)=16
!
24,801 Step 09_06: vvvvvvvv[vv]BOO{MS}HAKALAKA[vv]vvvvvvvv ! Searching for 'MS', FoundAtPosition = 24, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(24-1)-09+(2)=16
!
24,802 Step 09_07: vvvvvvvv[vv]BOOM{SH}AKALAKA[vv]vvvvvvvv ! Searching for 'SH', FoundAtPosition = 24, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(24-1)-09+(2)=16
!
```

```
24,803 Step 09_08: vvvvvvvv[vv]BOOMS{HA}KALAKA[vv]vvvvvvvv ! Searching for 'HA', FoundAtPosition = 24, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(24-1)-09+(2)=16
!
24,804 Step 09_09: vvvvvvvv[vv]BOOMSH{AK}AL{AK}Avvvvvvvvv ! Searching for 'AK', FoundAtPosition = 21, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(21-1)-09+(2)=13
!
24,805 Step 09_10: vvvvvvvv[vv]BOOMSHA{KA}L{AK}Avvvvvvvvv ! Searching for 'KA', FoundAtPosition = 21, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(21-1)-09+(2)=13
!
24,806 Step 09_11: vvvvvvvv[vv]BOOMSHAK{AL}{AK}Avvvvvvvvv ! Searching for 'AL', FoundAtPosition = 21, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(21-1)-09+(2)=13
!
24,807 Step 09_12: vvvvvvvv[vv]BOOMSHAKA{L}{A}K}Avvvvvvvvv ! Searching for 'LA', FoundAtPosition = 21, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(21-1)-09+(2)=13
!
24,808     if (PRIMALlengthCANDIDATE >= PRIMALlength) {PRIMALposition=PRIMALpositionCANDIDATE; PRIMALlength = PRIMALlengthCANDIDATE;}
24,809 ...
24,810 Step 31_00: {}vvvvvvvv[vv]BOOMSHAKALAKAvvvvvvvvv[v?] ! For position #31 the initial boundaries are PRIMALpositionCANDIDATE=LeftBoundary=31,
RightBoundary=FoundAtPosition-1, the CANDIDATE PRIMAL string length is RightBoundary-LeftBoundary+(2)=(33-1)-31+(2)=03 !
24,811 Step 31_01: vvvvvvvvvvBOOMSHAKALAKAvvvvvvvv[{v(v)}v] ! Searching for 'vv', FoundAtPosition = 32, PRIMALlengthCANDIDATE=RightBoundary-LeftBoundary+(2)=(32-1)-31+(2)=02
!
24,812     if (PRIMALlengthCANDIDATE >= PRIMALlength) {PRIMALposition=PRIMALpositionCANDIDATE; PRIMALlength = PRIMALlengthCANDIDATE;}
24,813     Result:
24,814     PRIMALposition=09 PRIMALlength=13, NewNeedle = 'vvBOOMSHAKALA'
24,815 */
24,816
24,817 // Here we have 4 or bigger NewNeedle, apply order 2 for pbPattern[i+(PRIMALposition-1)] with length 'PRIMALlength' and compare the pbPattern[i] with length
'cbPattern':
24,818 PRIMALlengthCANDIDATE = cbPattern;
24,819 cbPattern = PRIMALlength;
24,820 pbPattern = pbPattern + (PRIMALposition-1);
24,821
24,822 // Revision 2 commented section [
24,823 /*
24,824 if (cbPattern-1 <= 255) {
24,825 // BMH Order 2 [
24,826     ulHashPattern = *(uint32_t *) (pbPattern); // First four bytes
24,827     for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]= cbPattern-1;} // cbPattern-(Order-1) for Horspool; 'memset' if not optimized
24,828     for (i=0; i < cbPattern-1; i++) bm_Horspool_Order2[(unsigned short *) (pbPattern+i)]=i; // Rightmost appearance/position is needed
24,829     i=0;
24,830     while (i <= cbTarget-cbPattern) {
24,831         Gulliver = bm_Horspool_Order2[(unsigned short *) &pbTarget[i+cbPattern-1]];
24,832         if ( Gulliver != cbPattern-1 ) { // CASE #2: if equal means the pair (char order 2) is not found i.e. Gulliver remains intact, skip the whole
pattern and fall back (Order-1) chars i.e. one char for Order 2
24,833             if ( Gulliver == cbPattern-2 ) { // CASE #1: means the pair (char order 2) is found
24,834                 if ( *(uint32_t *) &pbTarget[i] == ulHashPattern) {
24,835                     count = cbPattern-4+1;
24,836                     while ( count > 0 && *(uint32_t *) (pbPattern+count-1) == *(uint32_t *) (&pbTarget[i]+(count-1)) )
24,837                         count = count-4;
24,838 // If we miss to hit then no need to compare the original: Needle
24,839 if ( count <= 0 ) {
24,840 // I have to add out-of-range checks...
24,841 // i-(PRIMALposition-1) >= 0
24,842 // &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
24,843 // i-(PRIMALposition-1)+(count-1) >= 0
24,844 // &pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4
24,845
24,846 // "FIX" from 2014-Apr-27:
24,847 // Because (count-1) is negative, above fours are reduced to next twos:
24,848 // i-(PRIMALposition-1)+(count-1) >= 0
24,849 // &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
24,850 // The line below is BUGGY:
24,851 //if ( (i-(PRIMALposition-1) >= 0) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) && (&pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4) ) {
```

```

24,852 // The line below is NOT OKAY, in fact so stupid, grrr, not a blunder, not carelessness, but overconfidence in writing "on the fly":
24,853 //if ( ((signed int)(i-(PRIMALposition-1)+(count-1)) >= 0) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) ) {
24,854 // FIX from 2016-Aug-10 (two times failed to do simple boundary checks, pfu):
24,855 if ( ((signed long long)(i-(PRIMALposition-1)) >= 0) && (&pbTarget[i-(PRIMALposition-1)]+((PRIMALlengthCANDIDATE-4+1)-1) <= pbTargetMax - 4) ) { // 2020-jan-11
24,856     if ( *(uint32_t *)&pbTarget[i-(PRIMALposition-1)] == *(uint32_t *)(&pbPattern-(PRIMALposition-1))) { // This fast check ensures not missing a match (for
remainder) when going under 0 in loop below:
24,857         count = PRIMALlengthCANDIDATE-4+1;
24,858         while ( count > 0 && *(uint32_t *)(&pbPattern-(PRIMALposition-1)+count-1) == *(uint32_t *)(&pbTarget[i-(PRIMALposition-1)]+(count-1)) )
24,859             count = count-4;
24,860         if ( count <= 0 ) return(pbTarget+i-(PRIMALposition-1));
24,861     }
24,862 }
24,863 }
24,864     }
24,865     Gulliver = 1;
24,866 } else
24,867     Gulliver = cbPattern - Gulliver - 2; // CASE #3: the pair is found and not as suffix i.e. rightmost position
24,868 }
24,869 i = i + Gulliver;
24,870 //GlobalI++; // Comment it, it is only for stats.
24,871 }
24,872 return(NULL);
24,873 // BMH Order 2 ]
24,874 } else {
24,875     // BMH order 2, needle should be >=4:
24,876     ulHashPattern = *(uint32_t *)(&pbPattern); // First four bytes
24,877     for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]=0;}
24,878     for (i=0; i < cbPattern-1; i++) bm_Horspool_Order2[*(unsigned short *)(&pbPattern+i)]=1;
24,879     i=0;
24,880     while (i <= cbTarget-cbPattern) {
24,881         Gulliver = 1; // 'Gulliver' is the skip
24,882         if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+cbPattern-1-1]] != 0 ) {
24,883             if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+cbPattern-1-2]] == 0 ) Gulliver = cbPattern-(2-1)-2; else {
24,884                 if ( *(uint32_t *)&pbTarget[i] == ulHashPattern) { // This fast check ensures not missing a match (for remainder) when going
under 0 in loop below:
24,885                     count = cbPattern-4+1;
24,886                     while ( count > 0 && *(uint32_t *)(&pbPattern+count-1) == *(uint32_t *)(&pbTarget[i]+(count-1)) )
24,887                         count = count-4;
24,888 // If we miss to hit then no need to compare the original: Needle
24,889 if ( count <= 0 ) {
24,890 // I have to add out-of-range checks...
24,891 // i-(PRIMALposition-1) >= 0
24,892 // &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
24,893 // i-(PRIMALposition-1)+(count-1) >= 0
24,894 // &pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4
24,895 // "FIX" from 2014-Apr-27:
24,896 // Because (count-1) is negative, above fours are reduced to next twos:
24,897 // i-(PRIMALposition-1)+(count-1) >= 0
24,898 // &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
24,900 // The line below is BUGGY:
24,901 //if ( (i-(PRIMALposition-1)) >= 0) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) && (&pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4) ) {
24,902 // The line below is NOT OKAY, in fact so stupid, grrr, not a blunder, not carelessness, but overconfidence in writing "on the fly":
24,903 //if ( ((signed int)(i-(PRIMALposition-1)+(count-1)) >= 0) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) ) {
24,904 // FIX from 2016-Aug-10 (two times failed to do simple boundary checks, pfu):
24,905 if ( ((signed long long)(i-(PRIMALposition-1)) >= 0) && (&pbTarget[i-(PRIMALposition-1)]+((PRIMALlengthCANDIDATE-4+1)-1) <= pbTargetMax - 4) ) { // 2020-jan-11
24,906     if ( *(uint32_t *)&pbTarget[i-(PRIMALposition-1)] == *(uint32_t *)(&pbPattern-(PRIMALposition-1))) { // This fast check ensures not missing a match (for
remainder) when going under 0 in loop below:

```

```

24,907         count = PRIMALlengthCANDIDATE-4+1;
24,908         while ( count > 0 && *(uint32_t *) (pbPattern-(PRIMALposition-1)+count-1) == *(uint32_t *) (&pbTarget[i-(PRIMALposition-1)]+(count-1)) )
24,909             count = count-4;
24,910         if ( count <= 0 ) return(pbTarget+i-(PRIMALposition-1));
24,911     }
24,912 }
24,913 }
24,914     }
24,915     }
24,916     } else Gulliver = cbPattern-(2-1);
24,917     i = i + Gulliver;
24,918     //GlobalI++; // Comment it, it is only for stats.
24,919 }
24,920 return(NULL);
24,921 }
24,922 */
24,923 // Revision 2 commented section ]
24,924
24,925     if ( cbPattern<=NeedleThreshold2vs4swampLITE ) {
24,926
24,927         // BMH order 2, needle should be >=4:
24,928         ulHashPattern = *(uint32_t *) (pbPattern); // First four bytes
24,929         for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]=0;}
24,930         // Above line is translated by Intel as:
24,931 // 0044c 41 b8 00 00 01
24,932 //          00          mov r8d, 65536
24,933 // 00452 44 89 5c 24 20  mov DWORD PTR [32+rsp], r11d
24,934 // 00457 44 89 54 24 60  mov DWORD PTR [96+rsp], r10d
24,935 // 0045c e8 fc ff ff ff  call _intel_fast_memset
24,936         for (i=0; i < cbPattern-1; i++) bm_Horspool_Order2[* (unsigned short *) (pbPattern+i)]=1;
24,937         i=0;
24,938         while (i <= cbTarget-cbPattern) {
24,939             Gulliver = 1; // 'Gulliver' is the skip
24,940             if ( bm_Horspool_Order2[* (unsigned short *) &pbTarget[i+cbPattern-1-1]] != 0 ) {
24,941                 if ( bm_Horspool_Order2[* (unsigned short *) &pbTarget[i+cbPattern-1-1-2]] == 0 ) Gulliver = cbPattern-(2-1)-2; else {
24,942                     if ( *(uint32_t *) &pbTarget[i] == ulHashPattern) { // This fast check ensures not missing a match (for remainder) when going
under 0 in loop below:
24,943                         count = cbPattern-4+1;
24,944                         while ( count > 0 && *(uint32_t *) (pbPattern+count-1) == *(uint32_t *) (&pbTarget[i]+(count-1)) )
24,945                             count = count-4;
24,946
24,947         if (cbPattern != PRIMALlengthCANDIDATE) { // No need of same comparison when Needle and NewNeedle are equal!
24,948         // If we miss to hit then no need to compare the original: Needle
24,949         if ( count <= 0 ) {
24,950         // I have to add out-of-range checks...
24,951         // i-(PRIMALposition-1) >= 0
24,952         // &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
24,953         // i-(PRIMALposition-1)+(count-1) >= 0
24,954         // &pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4
24,955
24,956         // "FIX" from 2014-Apr-27:
24,957         // Because (count-1) is negative, above fours are reduced to next twos:
24,958         // i-(PRIMALposition-1)+(count-1) >= 0
24,959         // &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
24,960         // The line below is BUGGY:
24,961         //if ( (i-(PRIMALposition-1)) >= 0 ) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) && (&pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4) ) {
24,962         // The line below is NOT OKAY, in fact so stupid, grrr, not a blunder, not carelessness, but overconfidence in writing "on the fly":
24,963         //if ( ((signed int)(i-(PRIMALposition-1)+(count-1))) >= 0 ) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) ) {

```

```

24,964 // FIX from 2016-Aug-10 (two times failed to do simple boundary checks, pfu):
24,965 if ( ((signed long long)(i-(PRIMALposition-1)) >= 0) && (&pbTarget[i-(PRIMALposition-1)]+((PRIMALlengthCANDIDATE-4+1)-1) <= pbTargetMax - 4) ) { // 2020-jan-11
24,966     if ( *(uint32_t *)&pbTarget[i-(PRIMALposition-1)] == *(uint32_t *)(&pbPattern-(PRIMALposition-1))) { // This fast check ensures not missing a match (for
remainder) when going under 0 in loop below:
24,967         count = PRIMALlengthCANDIDATE-4+1;
24,968         while ( count > 0 && *(uint32_t *)(&pbPattern-(PRIMALposition-1)+count-1) == *(uint32_t *)(&pbTarget[i-(PRIMALposition-1)]+(count-1)) )
24,969             count = count-4;
24,970         if ( count <= 0 ) return(pbTarget+i-(PRIMALposition-1));
24,971     }
24,972 }
24,973 }
24,974 } else { //if (cbPattern != PRIMALlengthCANDIDATE)
24,975     if ( count <= 0 ) return(pbTarget+i);
24,976 }
24,977     }
24,978     }
24,979     } else Gulliver = cbPattern-(2-1);
24,980     i = i + Gulliver;
24,981     //GlobalI++; // Comment it, it is only for stats.
24,982 }
24,983 return(NULL);
24,984 }
24,985 } else { // if ( cbPattern<=NeedleThreshold2vs4swampLITE )
24,986     // BMH pseudo-order 4, needle should be >=8+2:
24,987     ulHashPattern = *(uint32_t *)(&pbPattern); // First four bytes
24,988     for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]=0;}
24,989     // In line below we "hash" 4bytes to 2bytes i.e. 16bit table, how to compute TOTAL number of BBs, 'cbPattern - Order + 1' is the number of BBs for
text 'cbPattern' bytes long, for example, for cbPattern=11 'fastest fox' and Order=4 we have BBs = 11-4+1=8:
24,991     //"fast"
24,992     //"aste"
24,993     //"stes"
24,994     //"test"
24,995     //"est "
24,996     //"st f"
24,997     //"t fo"
24,998     //" fox"
24,999     //for (i=0; i < cbPattern-4+1; i++) bm_Horspool_Order2[( *(unsigned short *)(&pbPattern+i+0) + *(unsigned short *)(&pbPattern+i+2) ) & ( (1<<16)-1 )]=1;
25,000     //for (i=0; i < cbPattern-4+1; i++) bm_Horspool_Order2[( *(uint32_t *)(&pbPattern+i+0)>>16)+*(uint32_t *)(&pbPattern+i+0)&0xFFFF) & ( (1<<16)-1
)]]=1;
25,001     // Above line is replaced by next one with better hashing:
25,002     for (i=0; i < cbPattern-4+1; i++) bm_Horspool_Order2[( *(uint32_t *)(&pbPattern+i+0)>>(16-1))+*(uint32_t *)(&pbPattern+i+0)&0xFFFF) & ( (1<<16)-1
)]]=1;
25,003     i=0;
25,004     while (i <= cbTarget-cbPattern) {
25,005         Gulliver = 1;
25,006         //if ( bm_Horspool_Order2[( *(uint32_t *)&pbTarget[i+cbPattern-1-1-2]>>16)+*(uint32_t *)&pbTarget[i+cbPattern-1-1-2]&0xFFFF) & ( (1<<16)-1
)] != 0 ) { // DWORD #1
25,007             // Above line is replaced by next one with better hashing:
25,008             if ( bm_Horspool_Order2[( *(uint32_t *)&pbTarget[i+cbPattern-1-1-2]>>(16-1))+*(uint32_t *)&pbTarget[i+cbPattern-1-1-2]&0xFFFF) & (
(1<<16)-1) ] != 0 ) { // DWORD #1
25,009                 //if ( bm_Horspool_Order2[( *(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]>>16)+*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF) &
( (1<<16)-1) ] == 0 ) Gulliver = cbPattern-(2-1)-2-4; else {
25,010                     // Above line is replaced in order to strengthen the skip by checking the middle DWORD,if the two DWORDs are 'ab' and 'cd' i.e.
[2x][2a][2b][2c][2d] then the middle DWORD is 'bc'.
25,011                     // The respective offsets (backwards) are: -10/-8/-6/-4 for 'xa'/'ab'/'bc'/'cd'.
25,012                     //if ( ( bm_Horspool_Order2[( *(uint32_t *)&pbTarget[i+cbPattern-1-1-2-6]>>16)+*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-6]&0xFFFF) &
& ( (1<<16)-1) ] + ( bm_Horspool_Order2[( *(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]>>16)+*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF) & ( (1<<16)-1) ] ) + (

```

```

bm_Horspool_Order2[( (*uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]>>16)+(*uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]&0xFFFF) & ( (1<<16)-1 ) ] < 3 ) Gulliver = cbPattern-
(2-1)-2-4-2; else {
25,013 // Above line is replaced by next one with better hashing:
25,014 // When using (16-1) right shifting instead of 16 we will have two different pairs (if they are equal), the highest bit being lost do
the job especially for ASCII texts with no symbols in range 128-255.
25,015 // Example for genomesque pair TT+TT being shifted by (16-1):
25,016 // T = 01010100
25,017 // TT = 01010100 01010100
25,018 // TTTT = 01010100 01010100 01010100 01010100
25,019 // TTTT>>16 = 00000000 00000000 01010100 01010100
25,020 // TTTT>>(16-1) = 00000000 00000000 10101000 10101000 <--- Due to the left shift by 1, the 8th bits of 1st and 2nd bytes are populated
- usually they are 0 for English texts & 'ACGT' data.
25,021 //if ( ( bm_Horspool_Order2[( (*uint32_t *)&pbTarget[i+cbPattern-1-1-2-6]>>(16-1))+(*uint32_t *)&pbTarget[i+cbPattern-1-1-2-
6]&0xFFFF) & ( (1<<16)-1 ) ] ) + ( bm_Horspool_Order2[( (*uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]>>(16-1))+(*uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF) & (
(1<<16)-1 ) ] ) + ( bm_Horspool_Order2[( (*uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]>>(16-1))+(*uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]&0xFFFF) & ( (1<<16)-1 ) ] ) < 3 )
Gulliver = cbPattern-(2-1)-2-4-2; else {
25,022 // 'Maximus' uses branched 'if', again.
25,023 if ( \
25,024 ( bm_Horspool_Order2[( (*uint32_t *)&pbTarget[i+cbPattern-1-1-2-6 +1]>>(16-1))+(*uint32_t *)&pbTarget[i+cbPattern-1-1-2-6
+1]&0xFFFF) & ( (1<<16)-1 ) ] ) == 0 \
25,025 || ( bm_Horspool_Order2[( (*uint32_t *)&pbTarget[i+cbPattern-1-1-2-4 +1]>>(16-1))+(*uint32_t *)&pbTarget[i+cbPattern-1-1-2-4
+1]&0xFFFF) & ( (1<<16)-1 ) ] ) == 0 \
25,026 ) Gulliver = cbPattern-(2-1)-2-4-2 +1; else {
25,027 // Above line is not optimized (several a SHR are used), we have 5 non-overlapping WORDs, or 3 overlapping WORDs, within 4 overlapping
DWORDs so:
25,028 // [2x][2a][2b][2c][2d]
25,029 // DWORD #4
25,030 // [2a] (*uint32_t *)&pbTarget[i+cbPattern-1-1-2-6]>>16) = !SHR to be avoided! <--
25,031 // [2x] (*uint32_t *)&pbTarget[i+cbPattern-1-1-2-6]&0xFFFF) = |
25,032 // DWORD #3 |
25,033 // [2b] (*uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]>>16) = !SHR to be avoided! |<--
25,034 // [2a] (*uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF) = ----- |
25,035 // DWORD #2 |
25,036 // [2c] (*uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]>>16) = !SHR to be avoided! |<--
25,037 // [2b] (*uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]&0xFFFF) = ----- |
25,038 // DWORD #1 |
25,039 // [2d] (*uint32_t *)&pbTarget[i+cbPattern-1-1-2-0]>>16) = |
25,040 // [2c] (*uint32_t *)&pbTarget[i+cbPattern-1-1-2-0]&0xFFFF) = ----- |
25,041 //
25,042 // So in order to remove 3 SHR instructions the equal extractions are:
25,043 // DWORD #4
25,044 // [2a] (*uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF) = !SHR to be avoided! <--
25,045 // [2x] (*uint32_t *)&pbTarget[i+cbPattern-1-1-2-6]&0xFFFF) = |
25,046 // DWORD #3 |
25,047 // [2b] (*uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]&0xFFFF) = !SHR to be avoided! |<--
25,048 // [2a] (*uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF) = ----- |
25,049 // DWORD #2 |
25,050 // [2c] (*uint32_t *)&pbTarget[i+cbPattern-1-1-2-0]&0xFFFF) = !SHR to be avoided! |<--
25,051 // [2b] (*uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]&0xFFFF) = ----- |
25,052 // DWORD #1 |
25,053 // [2d] (*uint32_t *)&pbTarget[i+cbPattern-1-1-2-0]>>16) = |
25,054 // [2c] (*uint32_t *)&pbTarget[i+cbPattern-1-1-2-0]&0xFFFF) = ----- |
25,055 //if ( ( bm_Horspool_Order2[( (*uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF)+(*uint32_t *)&pbTarget[i+cbPattern-1-1-2-
6]&0xFFFF) & ( (1<<16)-1 ) ] ) + ( bm_Horspool_Order2[( (*uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]&0xFFFF)+(*uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF) & (
(1<<16)-1 ) ] ) + ( bm_Horspool_Order2[( (*uint32_t *)&pbTarget[i+cbPattern-1-1-2-0]&0xFFFF)+(*uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]&0xFFFF) & ( (1<<16)-1 ) ] ) < 3 )
Gulliver = cbPattern-(2-1)-2-6; else {
25,056 // Since the above Decumanus mumbo-jumbo (3 overlapping lookups vs 2 non-overlapping lookups) is not fast enough we go DuoDecumanus or 3x4:
25,057 // [2y][2x][2a][2b][2c][2d]

```

```

25,058 // DWORD #3
25,059 //         DWORD #2
25,060 //         DWORD #1
25,061 //if ( ( bm_Horspool_Order2[( *(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]>>16)]+(*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF) )
& ( (1<<16)-1) ) ) + ( bm_Horspool_Order2[( *(uint32_t *)&pbTarget[i+cbPattern-1-1-2-8]>>16)]+(*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-8]&0xFFFF) ) & ( (1<<16)-1) ) ) < 2 )
Gulliver = cbPattern-(2-1)-2-8; else {
25,062         if ( *(uint32_t *)&pbTarget[i] == ulHashPattern) {
25,063             // Order 4 [
25,064             // Let's try something "outrageous" like comparing with[out] overlap BBs 4bytes long instead of 1 byte back-to-back:
25,065             // Inhere we are using order 4, 'cbPattern - Order + 1' is the number of BBs for text 'cbPattern' bytes long, for example, for
cbPattern=11 'fastest fox' and Order=4 we have BBs = 11-4+1=8:
25,066             //0:"fast" if the comparison failed here, 'count' is 1; 'Gulliver' is cbPattern-(4-1)-7
25,067             //1:"aste" if the comparison failed here, 'count' is 2; 'Gulliver' is cbPattern-(4-1)-6
25,068             //2:"stes" if the comparison failed here, 'count' is 3; 'Gulliver' is cbPattern-(4-1)-5
25,069             //3:"test" if the comparison failed here, 'count' is 4; 'Gulliver' is cbPattern-(4-1)-4
25,070             //4:"est " if the comparison failed here, 'count' is 5; 'Gulliver' is cbPattern-(4-1)-3
25,071             //5:"st f" if the comparison failed here, 'count' is 6; 'Gulliver' is cbPattern-(4-1)-2
25,072             //6:"t fo" if the comparison failed here, 'count' is 7; 'Gulliver' is cbPattern-(4-1)-1
25,073             //7:" fox" if the comparison failed here, 'count' is 8; 'Gulliver' is cbPattern-(4-1)
25,074             count = cbPattern-4+1;
25,075             // Below comparison is UNIdirectional:
25,076             while ( count > 0 && *(uint32_t *)&(pbPattern+count-1) == *(uint32_t *)&(&pbTarget[i]+(count-1)) )
25,077                 count = count-4;
25,078
25,079 if (cbPattern != PRIMALlengthCANDIDATE) { // No need of same comparison when Needle and NewNeedle are equal!
25,080 // count = cbPattern-4+1 = 23-4+1 = 20
25,081 // boomshakalakazzzzzz[zzzz] 20
25,082 // boomshakalakazz[zzzz]zzzz 20-4
25,083 // boomshakala[kazz]zzzzzzzz 20-8 = 12
25,084 // boomsha[kala]kazzzzzzzzzz 20-12 = 8
25,085 // boo[msha]kalakazzzzzzzzzz 20-16 = 4
25,086
25,087 // If we miss to hit then no need to compare the original: Needle
25,088 if ( count <= 0 ) {
25,089 // I have to add out-of-range checks...
25,090 // i-(PRIMALposition-1) >= 0
25,091 // &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
25,092 // i-(PRIMALposition-1)+(count-1) >= 0
25,093 // &pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4
25,094
25,095 // "FIX" from 2014-Apr-27:
25,096 // Because (count-1) is negative, above fours are reduced to next twos:
25,097 // i-(PRIMALposition-1)+(count-1) >= 0
25,098 // &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
25,099 // The line below is BUGGY:
25,100 //if ( (i-(PRIMALposition-1) >= 0) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) && (&pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4) ) {
25,101 // The line below is NOT OKAY, in fact so stupid, grrr, not a blunder, not carelessness, but overconfidence in writing "on the fly":
25,102 //if ( ((signed int)(i-(PRIMALposition-1)+(count-1)) >= 0) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) ) {
25,103 // FIX from 2016-Aug-10 (two times failed to do simple boundary checks, pfu):
25,104 if ( ((signed long long)(i-(PRIMALposition-1))) >= 0) && (&pbTarget[i-(PRIMALposition-1)]+((PRIMALlengthCANDIDATE-4+1)-1) <= pbTargetMax - 4) ) { // 2020-jan-11
25,105     if ( *(uint32_t *)&pbTarget[i-(PRIMALposition-1)] == *(uint32_t *)&(pbPattern-(PRIMALposition-1))) { // This fast check ensures not missing a match (for
remainder) when going under 0 in loop below:
25,106         count = PRIMALlengthCANDIDATE-4+1;
25,107         while ( count > 0 && *(uint32_t *)&(pbPattern-(PRIMALposition-1)+count-1) == *(uint32_t *)&(&pbTarget[i-(PRIMALposition-1)]+(count-1)) )
25,108             count = count-4;
25,109         if ( count <= 0 ) return(pbTarget+i-(PRIMALposition-1));
25,110     }
25,111 }

```



```

25,112 }
25,113 } else { //if (cbPattern != PRIMALlengthCANDIDATE)
25,114         if ( count <= 0 ) return(pbTarget+i);
25,115 }
25,116
25,117 // In order to avoid only-left or only-right WCS the memcmp should be done as left-to-right and right-to-left AT THE
SAME TIME.
25,118 // Below comparison is BIdirectional. It pays off when needle is 8+++ long:
25,119 //         for (count = cbPattern-4+1; count > 0; count = count-4) {
25,120 //             if ( *(uint32_t *) (pbPattern+count-1) != *(uint32_t *) (&pbTarget[i]+(count-1)) ) {break;};
25,121 //             if ( *(uint32_t *) (pbPattern+(cbPattern-4+1)-count) != *(uint32_t *) (&pbTarget[i]+(cbPattern-4+1)-
count) ) {count = (cbPattern-4+1)-count +(1); break;} // +(1) because two lookups are implemented as one, also no danger of 'count' being 0 because of the fast check outwith
the 'while': if ( *(uint32_t *) &pbTarget[i] == ulHashPattern)
25,122 //         }
25,123 //         if ( count <= 0 ) return(pbTarget+i);
25,124 //         // Checking the order 2 pairs in mismatched DWORD, all the 3:
25,125 //         //if ( bm_Horspool_Order2[(unsigned short *) &pbTarget[i+count-1]] == 0 ) Gulliver = count; // 1 or bigger, as
it should
25,126 //         //if ( bm_Horspool_Order2[(unsigned short *) &pbTarget[i+count-1+1]] == 0 ) Gulliver = count+1; // 1 or
bigger, as it should
25,127 //         //if ( bm_Horspool_Order2[(unsigned short *) &pbTarget[i+count-1+1+1]] == 0 ) Gulliver = count+1+1; // 1 or
bigger, as it should
25,128 //         //         if ( bm_Horspool_Order2[(unsigned short *) &pbTarget[i+count-1]] + bm_Horspool_Order2[(unsigned short
*) &pbTarget[i+count-1+1]] + bm_Horspool_Order2[(unsigned short *) &pbTarget[i+count-1+1+1]] < 3 ) Gulliver = count; // 1 or bigger, as it should, THE
MIN(count, count+1, count+1+1)
25,129 //         // Above compound 'if' guarantees not that Gulliver > 1, an example:
25,130 //         // Needle:   fastest tax
25,131 //         // Window: ...fastast tax...
25,132 //         // After matching ' tax' vs ' tax' and 'fast' vs 'fast' the mismatched DWORD is 'test' vs 'tast':
25,133 //         // 'tast' when factorized down to order 2 yields: 'ta', 'as', 'st' - all the three when summed give 1+1+1=3 i.e.
Gulliver remains 1.
25,134 //         // Roughly speaking, this attempt maybe has its place in worst-case scenarios but not in English text and even
not in ACGT data, that's why I commented it in original 'Shockeroo'.
25,135 //         //if ( bm_Horspool_Order2[( ( *(uint32_t *) &pbTarget[i+count-1]) >> 16) + ( *(uint32_t *) &pbTarget[i+count-1] & 0xFFFF
) & ( (1<<16)-1 )] == 0 ) Gulliver = count; // 1 or bigger, as it should
25,136 //         // Above line is replaced by next one with better hashing:
25,137 //         //         if ( bm_Horspool_Order2[( ( *(uint32_t *) &pbTarget[i+count-1]) >> (16-1)) + ( *(uint32_t *) &pbTarget[i+count-
1] & 0xFFFF ) & ( (1<<16)-1 )] == 0 ) Gulliver = count; // 1 or bigger, as it should
25,138 //         // Order 4 ]
25,139 //         }
25,140 //     }
25,141 //     } else Gulliver = cbPattern-(2-1)-2; // -2 because we check the 4 rightmost bytes not 2.
25,142 //     i = i + Gulliver;
25,143 //     //GlobalI++; // Comment it, it is only for stats.
25,144 }
25,145 return(NULL);
25,146
25,147 // } // if ( cbPattern<=NeedleThreshold2vs4swampLITE )
25,148 // } // if ( cbPattern<=NeedleThreshold2vs4swampLITE )
25,149 } //if ( cbPattern<4 )
25,150 }
25,151 /*
25,152 // For short needles, and mainly haystacks, 'Doublet' is quite effective. Consider it or 'Quadruplet'.
25,153 // Fixed version from 2012-Feb-27.
25,154 // Caution: For better speed the case 'if (cbPattern==1)' was removed, so Pattern must be longer than 1 char.
25,155 char * Railgun_Doublet (char * pbTarget, char * pbPattern, uint32_t cbTarget, uint32_t cbPattern)
25,156 {
25,157     char * pbTargetMax = pbTarget + cbTarget;

```

```
25,158 register uint32_t ulHashPattern;
25,159 uint32_t ulHashTarget, count, countSTATIC;
25,160
25,161 if (cbPattern > cbTarget) return(NULL);
25,162
25,163 countSTATIC = cbPattern-2;
25,164
25,165 pbTarget = pbTarget+cbPattern;
25,166 ulHashPattern = (*(uint16_t *) (pbPattern));
25,167
25,168 for ( ;; ) {
25,169     if ( ulHashPattern == (*(uint16_t *) (pbTarget-cbPattern)) ) {
25,170         count = countSTATIC;
25,171         while ( count && *(char *) (pbPattern+2+(countSTATIC-count)) == *(char *) (pbTarget-cbPattern+2+(countSTATIC-count)) ) {
25,172             count--;
25,173         }
25,174         if ( count == 0 ) return((pbTarget-cbPattern));
25,175     }
25,176     pbTarget++;
25,177     if (pbTarget > pbTargetMax) return(NULL);
25,178 }
25,179 }
25,180 */
25,181
25,182 // Fixed version from 2012-Feb-27.
25,183 // Caution: For better speed the case 'if (cbPattern==1)' was removed, so Pattern must be longer than 1 char.
25,184 char * Railgun_Doublet (char * pbTarget, char * pbPattern, uint32_t cbTarget, uint32_t cbPattern)
25,185 {
25,186     char * pbTargetMax = pbTarget + cbTarget;
25,187     register uint32_t ulHashPattern;
25,188     uint32_t ulHashTarget, count, countSTATIC;
25,189
25,190     if (cbPattern > cbTarget) return(NULL);
25,191
25,192     countSTATIC = cbPattern-2;
25,193
25,194     pbTarget = pbTarget+cbPattern;
25,195     ulHashPattern = (*(uint16_t *) (pbPattern));
25,196
25,197     for ( ;; ) {
25,198         if ( ulHashPattern == (*(uint16_t *) (pbTarget-cbPattern)) ) {
25,199             count = countSTATIC;
25,200             while ( count && *(char *) (pbPattern+2+(countSTATIC-count)) == *(char *) (pbTarget-cbPattern+2+(countSTATIC-count)) ) {
25,201                 count--;
25,202             }
25,203             if ( count == 0 ) return((pbTarget-cbPattern));
25,204         }
25,205         pbTarget++;
25,206         if (pbTarget > pbTargetMax) return(NULL);
25,207     }
25,208 }
25,209
25,210 // Pattern must be >=4
25,211 char * Railgun_BawBaw_reverse (char * pbTarget, char * pbPattern, uint32_t cbTarget, uint32_t cbPattern)
25,212 {
25,213     register uint32_t ulHashPattern;
25,214     signed long count;
25,215
```

```

25,216 unsigned char bm_Horspool_Order2[256*256]; // Bitwise soon...
25,217 uint32_t i, Gulliver;
25,218
25,219 if (cbPattern > cbTarget) return(NULL);
25,220
25,221 // BMH order 2, needle should be >=4:
25,222 ulHashPattern = *(uint32_t *) (pbPattern); // First four bytes
25,223 for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]=0;}
25,224 for (i=0; i < cbPattern-1; i++) bm_Horspool_Order2[(unsigned short *) (pbPattern+i)]=1;
25,225 i=cbTarget;
25,226 while (i >= cbPattern) {
25,227     Gulliver = 1; // 'Gulliver' is the skip
25,228     if ( bm_Horspool_Order2[(unsigned short *)&pbTarget[i-cbPattern]] != 0 ) {
25,229         if ( bm_Horspool_Order2[(unsigned short *)&pbTarget[i-cbPattern+2]] == 0 ) Gulliver = cbPattern-(2-1)-2; else {
25,230             if ( *(uint32_t *)&pbTarget[i-cbPattern] == ulHashPattern) { // This fast check ensures not missing a match (for remainder)
when going under 0 in loop below:
25,231                 count = cbPattern-4+1;
25,232                 while ( count > 0 && *(uint32_t *) (pbPattern+count-1) == *(uint32_t *) (&pbTarget[i-cbPattern]+(count-1)) )
25,233                     count = count-4;
25,234                 if ( count <= 0 ) return(pbTarget+i-cbPattern);
25,235             }
25,236         }
25,237     } else Gulliver = cbPattern-(2-1);
25,238     i = i - Gulliver;
25,239 }
25,240 return(NULL);
25,241 }
25,242
25,243 // Pattern must be >=2
25,244 char * Railgun_Baw_reverse (char * pbTarget, char * pbPattern, uint32_t cbTarget, uint32_t cbPattern)
25,245 {
25,246     register uint16_t ulHashPattern;
25,247     signed long count;
25,248
25,249     unsigned char bm_Horspool_Order2[256*256]; // Bitwise soon...
25,250     uint32_t i, Gulliver;
25,251
25,252     if (cbPattern > cbTarget) return(NULL);
25,253
25,254     // BMH order 2, needle should be >=4:
25,255     ulHashPattern = *(uint32_t *) (pbPattern); // First four bytes
25,256     for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]=0;}
25,257     for (i=0; i < cbPattern-1; i++) bm_Horspool_Order2[(unsigned short *) (pbPattern+i)]=1;
25,258     i=cbTarget;
25,259     while (i >= cbPattern) {
25,260         Gulliver = 1; // 'Gulliver' is the skip
25,261         if ( bm_Horspool_Order2[(unsigned short *)&pbTarget[i-cbPattern]] != 0 ) {
25,262             if ( *(uint16_t *)&pbTarget[i-cbPattern] == ulHashPattern) { // This fast check ensures not missing a match (for remainder)
when going under 0 in loop below:
25,263                 count = cbPattern-2+1;
25,264                 while ( count > 0 && *(uint16_t *) (pbPattern+count-1) == *(uint16_t *) (&pbTarget[i-cbPattern]+(count-1)) )
25,265                     count = count-2;
25,266                 if ( count <= 0 ) return(pbTarget+i-cbPattern);
25,267             }
25,268         } else Gulliver = cbPattern-(2-1);
25,269         i = i - Gulliver;
25,270     }
25,271     return(NULL);

```

```
25,272 }
25,273
25,274 // To Tomisaburo Wakayama:
25,275 /*
```

```
25,276
25,277
25,278
25,279
25,280
25,281
25,282
25,283
25,284
25,285
25,286
25,287
25,288
25,289
25,290
25,291
25,292
25,293
25,294
25,295
25,296
25,297
25,298
25,299
25,300
25,301
25,302
25,303
25,304
25,305
25,306
25,307
25,308
25,309
25,310
25,311
25,312
25,313
25,314
25,315
25,316
25,317
25,318
25,319
25,320
25,321
25,322
25,323
25,324
25,325
25,326
25,327
25,328
25,329
25,330
25,331
25,332
25,333
25,334
25,335
25,336
25,337
25,338
25,339
25,340
25,341
25,342
25,343
25,344
25,345
25,346
25,347
25,348
25,349
25,350
25,351
25,352
25,353
25,354
25,355
25,356
25,357
25,358
25,359
25,360
25,361
25,362
25,363
25,364
25,365
25,366
25,367
25,368
25,369
25,370
25,371
25,372
25,373
25,374
25,375
25,376
25,377
25,378
25,379
25,380
25,381
25,382
25,383
25,384
25,385
25,386
25,387
25,388
25,389
25,390
25,391
25,392
25,393
25,394
25,395
25,396
25,397
25,398
25,399
25,400
25,401
25,402
25,403
25,404
25,405
25,406
25,407
25,408
25,409
25,410
25,411
25,412
25,413
25,414
25,415
25,416
25,417
25,418
25,419
25,420
25,421
25,422
25,423
25,424
25,425
25,426
25,427
25,428
25,429
25,430
25,431
25,432
25,433
25,434
25,435
25,436
25,437
25,438
25,439
25,440
25,441
25,442
25,443
25,444
25,445
25,446
25,447
25,448
25,449
25,450
25,451
25,452
25,453
25,454
25,455
25,456
25,457
25,458
25,459
25,460
25,461
25,462
25,463
25,464
25,465
25,466
25,467
25,468
25,469
25,470
25,471
25,472
25,473
25,474
25,475
25,476
25,477
25,478
25,479
25,480
25,481
25,482
25,483
25,484
25,485
25,486
25,487
25,488
25,489
25,490
25,491
25,492
25,493
25,494
25,495
25,496
25,497
25,498
25,499
25,500
25,501
25,502
25,503
25,504
25,505
25,506
25,507
25,508
25,509
25,510
25,511
25,512
25,513
25,514
25,515
25,516
25,517
25,518
25,519
25,520
25,521
25,522
25,523
25,524
25,525
25,526
25,527
25,528
25,529
25,530
25,531
25,532
25,533
25,534
25,535
25,536
25,537
25,538
25,539
25,540
25,541
25,542
25,543
25,544
25,545
25,546
25,547
25,548
25,549
25,550
25,551
25,552
25,553
25,554
25,555
25,556
25,557
25,558
25,559
25,560
25,561
25,562
25,563
25,564
25,565
25,566
25,567
25,568
25,569
25,570
25,571
25,572
25,573
25,574
25,575
25,576
25,577
25,578
25,579
25,580
25,581
25,582
25,583
25,584
25,585
25,586
25,587
25,588
25,589
25,590
25,591
25,592
25,593
25,594
25,595
25,596
25,597
25,598
25,599
25,600
25,601
25,602
25,603
25,604
25,605
25,606
25,607
25,608
25,609
25,610
25,611
25,612
25,613
25,614
25,615
25,616
25,617
25,618
25,619
25,620
25,621
25,622
25,623
25,624
25,625
25,626
25,627
25,628
25,629
25,630
25,631
25,632
25,633
25,634
25,635
25,636
25,637
25,638
25,639
25,640
25,641
25,642
25,643
25,644
25,645
25,646
25,647
25,648
25,649
25,650
25,651
25,652
25,653
25,654
25,655
25,656
25,657
25,658
25,659
25,660
25,661
25,662
25,663
25,664
25,665
25,666
25,667
25,668
25,669
25,670
25,671
25,672
25,673
25,674
25,675
25,676
25,677
25,678
25,679
25,680
25,681
25,682
25,683
25,684
25,685
25,686
25,687
25,688
25,689
25,690
25,691
25,692
25,693
25,694
25,695
25,696
25,697
25,698
25,699
25,700
25,701
25,702
25,703
25,704
25,705
25,706
25,707
25,708
25,709
25,710
25,711
25,712
25,713
25,714
25,715
25,716
25,717
25,718
25,719
25,720
25,721
25,722
25,723
25,724
25,725
25,726
25,727
25,728
25,729
25,730
25,731
25,732
25,733
25,734
25,735
25,736
25,737
25,738
25,739
25,740
25,741
25,742
25,743
25,744
25,745
25,746
25,747
25,748
25,749
25,750
25,751
25,752
25,753
25,754
25,755
25,756
25,757
25,758
25,759
25,760
25,761
25,762
25,763
25,764
25,765
25,766
25,767
25,768
25,769
25,770
25,771
25,772
25,773
25,774
25,775
25,776
25,777
25,778
25,779
25,780
25,781
25,782
25,783
25,784
25,785
25,786
25,787
25,788
25,789
25,790
25,791
25,792
25,793
25,794
25,795
25,796
25,797
25,798
25,799
25,800
25,801
25,802
25,803
25,804
25,805
25,806
25,807
25,808
25,809
25,810
25,811
25,812
25,813
25,814
25,815
25,816
25,817
25,818
25,819
25,820
25,821
25,822
25,823
25,824
25,825
25,826
25,827
25,828
25,829
25,830
25,831
25,832
25,833
25,834
25,835
25,836
25,837
25,838
25,839
25,840
25,841
25,842
25,843
25,844
25,845
25,846
25,847
25,848
25,849
25,850
25,851
25,852
25,853
25,854
25,855
25,856
25,857
25,858
25,859
25,860
25,861
25,862
25,863
25,864
25,865
25,866
25,867
25,868
25,869
25,870
25,871
25,872
25,873
25,874
25,875
25,876
25,877
25,878
25,879
25,880
25,881
25,882
25,883
25,884
25,885
25,886
25,887
25,888
25,889
25,890
25,891
25,892
25,893
25,894
25,895
25,896
25,897
25,898
25,899
25,900
25,901
25,902
25,903
25,904
25,905
25,906
25,907
25,908
25,909
25,910
25,911
25,912
25,913
25,914
25,915
25,916
25,917
25,918
25,919
25,920
25,921
25,922
25,923
25,924
25,925
25,926
25,927
25,928
25,929
25,930
25,931
25,932
25,933
25,934
25,935
25,936
25,937
25,938
25,939
25,940
25,941
25,942
25,943
25,944
25,945
25,946
25,947
25,948
25,949
25,950
25,951
25,952
25,953
25,954
25,955
25,956
25,957
25,958
25,959
25,960
25,961
25,962
25,963
25,964
25,965
25,966
25,967
25,968
25,969
25,970
25,971
25,972
25,973
25,974
25,975
25,976
25,977
25,978
25,979
25,980
25,981
25,982
25,983
25,984
25,985
25,986
25,987
25,988
25,989
25,990
25,991
25,992
25,993
25,994
25,995
25,996
25,997
25,998
25,999
26,000
```

Page 437 of 466

```

25,472 NoteC: In this latest (2019-Jan-20) compile, clock() was replaced with time() - to counter bigtime stats misreporting.
25,473 NoteD: Multi-way hashing allows each KeySize to occupy its own HASH pool, thus less RAM is in use - the LEAF is smaller.
25,474 Current priority class is REALTIME_PRIORITY_CLASS.
25,475 Allocating Source-Buffer 14 MB ...
25,476 Allocating Source-Buffer 14 MB (REVERSED) ...
25,477 Allocating Target-Buffer 46 MB ...
25,478 Allocating Verification-Buffer 14 MB ...
25,479 Leprechaun: Memory pool for B-tress is 6,612,048 KB.
25,480 Leprechaun: In this revision 8MB 8-way hash is used which results in 8 x 1,048,576 internal B-Trees of order 3.
25,481 Leprechaun: In this revision, 1 pass is to be made.
25,482 Leprechaun: Allocating HASH memory 67,108,929 bytes ... OK
25,483 Leprechaun: Allocating memory for B-tress 6458 MB ... OK
25,484 Leprechaun: Size of input file: 15,583,440
25,485
25,486 Leprechaun: Inserting keys/BBs of order 004 into B-trees, free RAM in B-tree pool is 6,612,048 KB ...
25,487 Leprechaun: Inserting keys/BBs of order 006 into B-trees, free RAM in B-tree pool is 6,604,089 KB ...
25,488 Leprechaun: Inserting keys/BBs of order 008 into B-trees, free RAM in B-tree pool is 6,550,882 KB ...
25,489 Leprechaun: Inserting keys/BBs of order 010 into B-trees, free RAM in B-tree pool is 6,388,585 KB ...
25,490 Leprechaun: Inserting keys/BBs of order 012 into B-trees, free RAM in B-tree pool is 6,055,645 KB ...
25,491 Leprechaun: Inserting keys/BBs of order 014 into B-trees, free RAM in B-tree pool is 5,532,175 KB ...
25,492 Leprechaun: Inserting keys/BBs of order 016 into B-trees, free RAM in B-tree pool is 4,825,455 KB ...
25,493 Leprechaun: Inserting keys/BBs of order 018 into B-trees, free RAM in B-tree pool is 3,954,416 KB ...
25,494
25,495 Leprechaun: Number Of Total/Distinct/Undistinct keys/BBs (all orders): 124,667,440/55,177,480/69,489,960
25,496 Leprechaun: Number Of TREES(GREATER THE BETTER): 6,768,818
25,497 Leprechaun: Number Of LEAFs(littler THE BETTER) not counting ROOT LEAFs: 34,942,972
25,498 Leprechaun: Highest Tree not counting ROOT Level i.e. CORONA levels(littler THE BETTER): 4
25,499 Leprechaun: Used value for B-trees pool in KB: 6,612,048
25,500 Leprechaun: Use next time for B-trees pool in KB: 3,672,584
25,501 Leprechaun: Total keys (all orders) into B-trees order 3 (during traversal/dump): 55,177,480
25,502 Leprechaun: Total keys MISSES/HITS (all orders) into B-trees order 3: 55,177,480/69,489,960
25,503
25,504 Leprechaun: B-trees building speed (counting ROOT LEAFs): 1,069,533 LEAFs/s
25,505 Leprechaun: B-trees traverse speed (counting ROOT LEAFs): 5,958,827 LEAFs/s
25,506 Leprechaun: Total Searches-n-Inserts Per Second: 2,770,387 SNIPS
25,507 Leprechaun: RAM needed to house B-trees (relative to the file being ripped): 241N
25,508
25,509 Compressing 15,583,440 bytes ...
25,510 ^C
25,511
25,512 D:\Nakamichi_Ryuugan-ditto-1TB_bigtime_b-tree_2019-Jan-20>Nakamichi_Ryuugan-ditto-1TB_ssd_GCC730.exe Arabian_Nights_complete.html
25,513 Nakamichi 'Ryuugan-ditto-1TB', written by Kaze, based on Nobuo Ito's LZSS source, babealicious suggestion by m^2 enforced, muffinesque suggestion by Jim Dempsey enforced.
25,514 Note0: Nakamichi 'Dragoneye' is 100% FREE, licenseless that is.
25,515 Note1: Hamid Buzidi's LzTurbo ([a] FASTEST [Textual] Decompressor, Levels 19/29/39) retains kingship, his TurboBench (2017-Apr-07) proves the supremacy of LzTurbo, Turbo-Amazing!
25,516 Note2: Conor Stokes' LZSSE2 ([a] FASTEST Textual Decompressor, Level 17) is embedded, all credits along with many thanks go to him.
25,517 Note3: 'Ryuugan' predecessors are Washigan, Okamigan, Zato, Tsubame, Tengu-Tsuyo, Tengu, Rakka, Kokuen, Kinroba, Yoko, Kinutora, Jiten, Butsuhira, Suiken, Keigan, Kumataka, Washi, Aratama, Hitomi, Nekomata, Kitsune, Kinezumi, Sanbashi, Kaiko, Inazuma, Zangetsu, Hanabi, Hanazakari, Sanshi and Kaidanji.
25,518 Note4: This variant is the SLOWEST compressor under the Sun! This compile can handle files up to 5120MB.
25,519 Note5: The matchfinder is either 'Railgun_Trolldom' (matches longer than 18) or Leprechaun's B-tree order 3 (matches less or equal to 18).
25,520 Note6: Instead of '_mm_loadu_si128' '_mm_lddqu_si128' is used.
25,521 Note7: The lookahead 'Tsuyo' heuristic which looks one char ahead is applied thrice, still not strengthened, though.
25,522 Note8: The compile made 2017-Oct-22, the decompression time measuring is done in 16x8 passes choosing the top score from 64 back-to-back runs - the goal - to enter [maximal] Turbo Mode.
25,523 Note9: In GP/SSE4.1/AVX2 compile, the 24 matches become 3xQWORD/1xQWORD+1xXMMWORD/1xYMMWORD.
25,524 NoteA: Maximum compression ratio is 44:1, for 704 bytes long matches within 1TB Sliding Window.
25,525 NoteB: Please send me (at sanmayce@sanmayce.com) decompression results obtained on machines with fast CPU-RAM subsystems.

```

```

25,526 NoteC: In this latest (2019-Jan-20) compile, clock() was replaced with time() - to counter bigtime stats misreporting.
25,527 NoteD: Multi-way hashing allows each KeySize to occupy its own HASH pool, thus less RAM is in use - the LEAF is smaller.
25,528 Current priority class is REALTIME_PRIORITY_CLASS.
25,529 Allocating Source-Buffer 14 MB ...
25,530 Allocating Source-Buffer 14 MB (REVERSED) ...
25,531 Allocating Target-Buffer 46 MB ...
25,532 Allocating Verification-Buffer 14 MB ...
25,533 Leprechaun: Memory pool for B-tress is 6,612,048 KB.
25,534 Leprechaun: In this revision 8MB 8-way hash is used which results in 8 x 1,048,576 external B-Trees of order 3.
25,535 Leprechaun: In this revision, 1 pass is to be made.
25,536 Leprechaun: Allocating HASH memory 67,108,929 bytes ... OK
25,537 Leprechaun: Allocating/ZEROing 6,770,737,166 bytes swap file ... OK
25,538 Leprechaun: Size of input file: 15,583,440
25,539
25,540 Leprechaun: Inserting keys/BBs of order 004 into B-trees, free RAM in B-tree pool is 6,612,048 KB ...
25,541 Leprechaun: Inserting keys/BBs of order 006 into B-trees, free RAM in B-tree pool is 6,604,089 KB ...
25,542 Leprechaun: Inserting keys/BBs of order 008 into B-trees, free RAM in B-tree pool is 6,550,882 KB ...
25,543 Leprechaun: Inserting keys/BBs of order 010 into B-trees, free RAM in B-tree pool is 6,388,585 KB ...
25,544 Leprechaun: Inserting keys/BBs of order 012 into B-trees, free RAM in B-tree pool is 6,055,645 KB ...
25,545 Leprechaun: Inserting keys/BBs of order 014 into B-trees, free RAM in B-tree pool is 5,532,175 KB ...
25,546 Leprechaun: Inserting keys/BBs of order 016 into B-trees, free RAM in B-tree pool is 4,825,455 KB ...
25,547 Leprechaun: Inserting keys/BBs of order 018 into B-trees, free RAM in B-tree pool is 3,954,416 KB ...
25,548
25,549 Leprechaun: Number Of Total/Distinct/Undistinct keys/BBs (all orders): 124,667,440/55,177,480/69,489,960
25,550 Leprechaun: Number Of TREES(GREATER THE BETTER): 6,768,818
25,551 Leprechaun: Number Of LEAFs(littler THE BETTER) not counting ROOT LEAFs: 34,942,972
25,552 Leprechaun: Highest Tree not counting ROOT Level i.e. CORONA levels(littler THE BETTER): 4
25,553 Leprechaun: Used value for B-trees pool in KB: 6,612,048
25,554 Leprechaun: Use next time for B-trees pool in KB: 3,672,584
25,555 Leprechaun: Total keys (all orders) into B-trees order 3 (during traversal/dump): 55,177,480
25,556 Leprechaun: Total keys MISSES/HITS (all orders) into B-trees order 3: 55,177,480/69,489,960
25,557
25,558 Leprechaun: B-trees building speed (counting ROOT LEAFs): 13,322 LEAFs/s
25,559 Leprechaun: B-trees traverse speed (counting ROOT LEAFs): 42,693 LEAFs/s
25,560 Leprechaun: Total Searches-n-Inserts Per Second: 30,354 SNIPS
25,561 Leprechaun: RAM needed to house B-trees (relative to the file being ripped): 241N
25,562 Leprechaun: Total IOPS for 402,619,834 'freads' and 268,170,926 'fwrites' (of packets 106 bytes long) during loading traversing all orders: 163,328 IOPS
25,563
25,564 Compressing 15,583,440 bytes ...
25,565 ^C
25,566
25,567 D:\Nakamichi_Ryuugan-ditto-1TB_bigtime_b-tree_2019-Jan-20>MakeEXEs_Ryuugan-ditto-1TB_btree_GCC.bat
25,568
25,569 D:\Nakamichi_Ryuugan-ditto-1TB_bigtime_b-tree_2019-Jan-20>gcc -O3 -msse4.1 -fomit-frame-pointer Nakamichi_Ryuugan-ditto-1TB_btree.c -o Nakamichi_Ryuugan-ditto-1TB_RAM_GCC730.exe -D_N_XMM -D_N_prefetch_4096 -D_N_alone -D_N_HI
25,570 GH_PRIORITY -DHashInBITS=20 -DHashChunkSizeInBITS=20 -DRAMpoolInKB=13612048
25,571
25,572 D:\Nakamichi_Ryuugan-ditto-1TB_bigtime_b-tree_2019-Jan-20>gcc -O3 -msse4.1 -fomit-frame-pointer Nakamichi_Ryuugan-ditto-1TB_btree.c -o Nakamichi_Ryuugan-ditto-1TB_SSD_GCC730.exe -D_N_XMM -D_N_prefetch_4096 -D_N_alone -D_N_HI
25,573 GH_PRIORITY -DHashInBITS=20 -DHashChunkSizeInBITS=20 -DRAMpoolInKB=13612048 -DExternalRAM
25,574
25,575 D:\Nakamichi_Ryuugan-ditto-1TB_bigtime_b-tree_2019-Jan-20>Nakamichi_Ryuugan-ditto-1TB_ssd_GCC730.exe Complete_Works_of_Charles_Dickens.txt
25,576 Nakamichi 'Ryuugan-ditto-1TB', written by Kaze, based on Nobuo Ito's LZSS source, babealicious suggestion by m^2 enforced, muffinesque suggestion by Jim Dempsey enforced.
25,577 Note0: Nakamichi 'Dragoneye' is 100% FREE, licenseless that is.
25,578 Note1: Hamid Buzidi's LzTurbo ([a] FASTEST [Textual] Decompressor, Levels 19/29/39) retains kingship, his TurboBench (2017-Apr-07) proves the supremacy of LzTurbo, Turbo-Amazing!
25,579 Note2: Conor Stokes' LZSSE2 ([a] FASTEST Textual Decompressor, Level 17) is embedded, all credits along with many thanks go to him.

```

25,580 Note3: 'Ryuugan' predecessors are Washigan, Okamigan, Zato, Tsubame, Tengu-Tsuyo, Tengu, Rakka, Kokuen, Kinroba, Yoko, Kinutora, Jiten, Butsuhira, Suiken, Keigan, Kumataka, Washi, Aratama, Hitomi, Nekomata, Kitsune, Kinezumi, Sanbashi, Kaiko, Inazuma, Zangetsu, Hanabi, Hanazakari, Sanshi and Kaidanji.

25,581 Note4: This variant is the SLOWEST compressor under the Sun! This compile can handle files up to 5120MB.

25,582 Note5: The matchfinder is either 'RailgunTrolldom' (matches longer than 18) or Leprechaun's B-tree order 3 (matches less or equal to 18).

25,583 Note6: Instead of '_mm_loadu_si128' '_mm_lddqu_si128' is used.

25,584 Note7: The lookahead 'Tsuyo' heuristic which looks one char ahead is applied thrice, still not strengthened, though.

25,585 Note8: The compile made 2017-Oct-22, the decompression time measuring is done in 16x8 passes choosing the top score from 64 back-to-back runs - the goal - to enter [maximal] Turbo Mode.

25,586 Note9: In GP/SSE4.1/AVX2 compile, the 24 matches become 3xQWORD/1xQWORD+1xXMMWORD/1xYMMWORD.

25,587 NoteA: Maximum compression ratio is 44:1, for 704 bytes long matches within 1TB Sliding Window.

25,588 NoteB: Please send me (at sanmayce@sanmayce.com) decompression results obtained on machines with fast CPU-RAM subsystems.

25,589 NoteC: In this latest (2019-Jan-20) compile, clock() was replaced with time() - to counter bigtime stats misreporting.

25,590 NoteD: Multi-way hashing allows each KeySize to occupy its own HASH pool, thus less RAM is in use - the LEAF is smaller.

25,591 Current priority class is REALTIME_PRIORITY_CLASS.

25,592 Allocating Source-Buffer 40 MB ...

25,593 Allocating Source-Buffer 40 MB (REVERSED) ...

25,594 Allocating Target-Buffer 72 MB ...

25,595 Allocating Verification-Buffer 40 MB ...

25,596 Leprechaun: Memory pool for B-trees is 13,612,048 KB.

25,597 Leprechaun: In this revision 8MB 8-way hash is used which results in 8 x 1,048,576 external B-Trees of order 3.

25,598 Leprechaun: In this revision, 1 pass is to be made.

25,599 Leprechaun: Allocating HASH memory 67,108,929 bytes ... OK

25,600 Leprechaun: Allocating/ZEROing 13,938,737,166 bytes swap file ... OK

25,601 Leprechaun: Size of input file: 42,935,960

25,602

25,603 Leprechaun: Inserting keys/BBs of order 004 into B-trees, free RAM in B-tree pool is 13,612,048 KB ...

25,604 Leprechaun: Inserting keys/BBs of order 006 into B-trees, free RAM in B-tree pool is 13,605,572 KB ...

25,605 Leprechaun: Inserting keys/BBs of order 008 into B-trees, free RAM in B-tree pool is 13,546,308 KB ...

25,606 Leprechaun: Inserting keys/BBs of order 010 into B-trees, free RAM in B-tree pool is 13,285,138 KB ...

25,607 Leprechaun: Inserting keys/BBs of order 012 into B-trees, free RAM in B-tree pool is 12,604,428 KB ...

25,608 Leprechaun: Inserting keys/BBs of order 014 into B-trees, free RAM in B-tree pool is 11,354,612 KB ...

25,609 Leprechaun: Inserting keys/BBs of order 016 into B-trees, free RAM in B-tree pool is 9,485,965 KB ...

25,610 Leprechaun: Inserting keys/BBs of order 018 into B-trees, free RAM in B-tree pool is 7,061,527 KB ...

25,611

25,612 Leprechaun: Number Of Total/Distinct/Undistinct keys/BBs (all orders): 343,487,600/139,455,560/204,032,040

25,613 Leprechaun: Number Of TREES(GREATER THE BETTER): 6,869,619

25,614 Leprechaun: Number Of LEAFs(littler THE BETTER) not counting ROOT LEAFs: 97,898,274

25,615 Leprechaun: Highest Tree not counting ROOT Level i.e. CORONA levels(littler THE BETTER): 5

25,616 Leprechaun: Used value for B-trees pool in KB: 13,612,048

25,617 Leprechaun: Use next time for B-trees pool in KB: 9,440,253

25,618 Leprechaun: Total keys (all orders) into B-trees order 3 (during traversal/dump): 139,455,560

25,619 Leprechaun: Total keys MISSES/HITS (all orders) into B-trees order 3: 139,455,560/204,032,040

25,620

25,621 Leprechaun: B-trees building speed (counting ROOT LEAFs): 5,116 LEAFs/s

25,622 Leprechaun: B-trees traverse speed (counting ROOT LEAFs): 10,564 LEAFs/s

25,623 Leprechaun: Total Searches-n-Inserts Per Second: 11,301 SNIPS

25,624 Leprechaun: RAM needed to house B-trees (relative to the file being ripped): 225N

25,625 Leprechaun: Total IOPS for 1,320,112,595 'freads' and 735,431,907 'fwrites' (of packets 106 bytes long) during loading traversing all orders: 67,632 IOPS

25,626

25,627 Compressing 42,935,960 bytes ...

25,628 ^C

25,629 */

25,630

25,631 // 2019-Aug-05:

25,632 /*

25,633 <http://mattmahoney.net/dc/text.html>

25,634

25,635 Large Text Compression Benchmark


```

25,636 Matt Mahoney
25,637 Last update: July 25, 2019.
25,638
25,639
25,640 Program          Compression          Compressed size  Decompressor  Total size  Time (ns/byte)
25,641 -----          -----          -----          -----          -----          -----
25,642 phda9 1.8                Options          enwik8      enwik9      size (zip)  enwik9+prog  Comp Decomp  Mem Alg Note
25,643 cmix v17                -----          -----          -----          -----          -----
25,644                                15,010,414  116,544,849  42,944 xd  116,587,793  86182 86305  6319 CM  83
25,645 ...                                14,877,373  116,394,271  208,263 s  116,602,534  641189 645651  25258 CM  83
25,646
25,647 cabarc 1.00.0601 -m lzx:21          28,465,607  250,756,595  51,917 xd  250,808,853  1619   15   20 LZ77
25,648 sr3                    28,926,691  253,031,980   9,399 s  253,054,625   148  160   68 SR  26
25,649 bzip2 1.0.2            -9          29,008,736  253,977,839  30,036 x  254,007,875   379  129   8 BWT
25,650
25,651 rh5_x64                -window:27 c6          29,078,552  254,220,469  36,744 x  254,257,213   196   9.4  145 ROLZ 48
25,652 RangeCoderC v1.7      c7 26          28,788,013  254,527,369   7,858 x  254,535,227  2460  2436  1116 CM  26
25,653 quad v1.11           -x          29,110,579  256,145,858  13,387 s  256,159,245   956  116   34 ROLZ
25,654 WinACE              -sfx -m5 -d4096        29,481,470  257,237,710   0 xd  257,237,710  1080   77   4
25,655 lzsr 0.01          29,433,834  258,912,605  40,287 x  258,952,892   194   88   6 LZ77 26
25,656 120
25,657
25,658 libzling 20160107 e4          29,721,114  259,475,639  35,582 s  259,511,221    83   27   28 ROLZ 48
25,659 xpv5                  c2          29,963,217  262,525,246  14,371 x  262,539,617  2359  516   9 ROLZ 26
25,660 sr3c 1.0              29,731,019  266,035,006   7,701 x  266,042,707   160  145   5 SR  26
25,661 lzc v0.08            10          30,611,315  266,565,255  11,364 x  266,576,619   302   63  550 LZ77
25,662
25,663 Nakamichi 'Dragoneye'          32,917,888  277,293,058                                1.3      LZSS 85
25,664
25,665 crush 1.00            cx          31,731,711  279,491,430   2,489 s  279,493,919   948   2.9  148 LZ77 60
25,666
25,667 xeloz 0.3.5.3          c889          32,441,272  283,621,211  18,771 s  283,639,982  1079   8  230 LZ77 48
25,668 bzip 0.2              31,563,865  283,908,295  36,808 x  283,945,103   110  120   3 LZIP
25,669 ha 0.98              a2          31,250,524  285,739,328  28,404 x  285,767,732  2010  1800   0.8 PPM
25,670 ulz 0.06             c9          32,945,292  291,028,084  49,450 x  291,077,534   325   1.1  490 LZ77 82
25,671 irolz                33,310,676  292,448,365   4,584 s  292,452,949   274  144   17 ROLZ 26
25,672
25,673 60. Tested by Ilia Muravyov on an Intel Core i7-3770K, 4.8 GHz, 16 GB Corsair Vengeance LP 1800 MHz CL9, Corsair Force GS 240 GB SSD, Windows 7 SP1.
25,674 82. Tested by Ilia Muraviev on an Intel Core i7-4790K @ 4.6GHz, 32GB @ 1866MHz DDR3 RAM, RAMDisk.
25,675 85. Tested by Georgi Marinov on i5-7200U @ 3.1GHz, 8GB @ 2133MHz DDR4 RAM, Windows 10.
25,676
25,677 Decompression rate in nanoseconds per byte, 1.3ns/B:
25,678 enwik9.Nakamichi, 725MB/s is 725x1024x1024B per 1,000,000,000ns
25,679 enwik9                1,000,000,000B per (1,000,000,000B/(725x1024x1024B))x1,000,000,000ns= 1,315,412,850ns
25,680
25,681 Or, Nakamichi decompresses enwik9 in 1.3s on a laptop.
25,682
25,683 Needed memory for Compression:
25,684 4N          + 293N          + (HASHPOOL=5N) <= 302N or 302GB
25,685 [Physical] [Physical|External] [Physical]
25,686
25,687 Needed memory for Decompression:
25,688 <= 2N or 2GB
25,689
25,690 Compression Rate:
25,691 122 B/s or 1000000000/122/3600/24 =~ 95 days
25,692
25,693 D:\TEXTORAMIC_benchmarking>lzbench173 -c4 -i1,15 -o3 -

```

etornado,16/blosclz,9/brieflz/crush,2/csc,5/density,3/fastlz,2/gipfeli/lzo1b,999/lzham,4/lzham24,4/libdeflate,1,12/lz4hc,1,10,12/lizard,19,29,39,49/lzf,1/lzfse/lzg,9/lzham,1/lzjb/lzlib,9/lzma,9/lzrw,5/lzsse2,17/lzsse4,17/lzsse8,17/lzvn/pithy,9/quicklz,3/snappy/slz_zlib,3/ucl_nrv2b,9/ucl_nrv2d,9/ucl_nrv2e,9/xpack,1,9/xz,9/yalz77,12/yappy,99/zlib,1,5,9/zling,4/shrinker/wflz/lzmat enwik9

25,694 lbzbench 1.7.3 (64-bit Windows) Assembled by P.Skibinski

25,695 The results sorted by column number 4:

Compressor name	Compress.	Decompress.	Orig. size	Compr. size	Ratio	Filename
csc 2016-10-13 -5	2.33 MB/s	59 MB/s	1000000000	213296889	21.33	enwik9
lzma 16.04 -9	1.12 MB/s	81 MB/s	1000000000	213337819	21.33	enwik9
xz 5.2.3 -9	1.19 MB/s	79 MB/s	1000000000	213337866	21.33	enwik9
lzham 1.0 -d26 -4	0.81 MB/s	188 MB/s	1000000000	215673584	21.57	enwik9
lzlib 1.8 -9	1.07 MB/s	57 MB/s	1000000000	216832124	21.68	enwik9
tornado 0.6a -16	1.31 MB/s	175 MB/s	1000000000	217735325	21.77	enwik9
lzham24 1.0 -4	1.01 MB/s	188 MB/s	1000000000	227368900	22.74	enwik9
zling 2016-01-10 -4	27 MB/s	144 MB/s	1000000000	259449164	25.94	enwik9
lzham 1.0 -d26 -1	1.97 MB/s	191 MB/s	1000000000	259506946	25.95	enwik9
Nakamichi 'Dragoneye'		725 MB/s		277293058		
crush 1.0 -2	0.35 MB/s	269 MB/s	1000000000	279083341	27.91	enwik9
xpack 2016-06-02 -9	10 MB/s	746 MB/s	1000000000	300716430	30.07	enwik9
libdeflate 0.7 -12	5.65 MB/s	559 MB/s	1000000000	310824785	31.08	enwik9
lizard 1.0 -49	1.38 MB/s	1046 MB/s	1000000000	318854201	31.89	enwik9
lzfse 2017-03-08	47 MB/s	596 MB/s	1000000000	319756993	31.98	enwik9
zlib 1.2.11 -9	17 MB/s	253 MB/s	1000000000	322789230	32.28	enwik9
lizard 1.0 -29	1.44 MB/s	1191 MB/s	1000000000	323348239	32.33	enwik9
zlib 1.2.11 -5	30 MB/s	249 MB/s	1000000000	327365805	32.74	enwik9
ucl_nrv2e 1.03 -9	1.30 MB/s	248 MB/s	1000000000	332405521	33.24	enwik9
ucl_nrv2d 1.03 -9	1.30 MB/s	250 MB/s	1000000000	335533150	33.55	enwik9
lzsse2 2016-05-14 -17	2.26 MB/s	2923 MB/s	1000000000	340270593	34.03	enwik9
ucl_nrv2b 1.03 -9	1.27 MB/s	242 MB/s	1000000000	341785796	34.18	enwik9
lzsse4 2016-05-14 -17	2.51 MB/s	3123 MB/s	1000000000	344520599	34.45	enwik9
lzsse8 2016-05-14 -17	2.33 MB/s	3032 MB/s	1000000000	346637623	34.66	enwik9
libdeflate 0.7 -1	122 MB/s	587 MB/s	1000000000	355647167	35.56	enwik9
xpack 2016-06-02 -1	114 MB/s	575 MB/s	1000000000	358741520	35.87	enwik9
lzo1b 2.09 -999	12 MB/s	459 MB/s	1000000000	363178533	36.32	enwik9
lzmat 1.01	31 MB/s	255 MB/s	1000000000	367262723	36.73	enwik9
lz4hc 1.8.0 -12	11 MB/s	2127 MB/s	1000000000	371677964	37.17	enwik9
lizard 1.0 -19	5.39 MB/s	2507 MB/s	1000000000	372092974	37.21	enwik9
lz4hc 1.8.0 -10	20 MB/s	2145 MB/s	1000000000	373026340	37.30	enwik9
zlib 1.2.11 -1	71 MB/s	245 MB/s	1000000000	378355076	37.84	enwik9
lizard 1.0 -39	5.27 MB/s	2337 MB/s	1000000000	378912990	37.89	enwik9
lzg 1.0.8 -9	0.84 MB/s	447 MB/s	1000000000	386976301	38.70	enwik9
brieflz 1.1.0	86 MB/s	135 MB/s	1000000000	388557049	38.86	enwik9
yalz77 2015-09-19 -12	17 MB/s	269 MB/s	1000000000	394059341	39.41	enwik9
quicklz 1.5.0 -3	39 MB/s	589 MB/s	1000000000	395494056	39.55	enwik9
lzvn 2017-03-08	39 MB/s	708 MB/s	1000000000	395814407	39.58	enwik9
lz4hc 1.8.0 -1	86 MB/s	2030 MB/s	1000000000	400818043	40.08	enwik9
gipfeli 2016-07-13	205 MB/s	318 MB/s	1000000000	411940549	41.19	enwik9
density 0.12.5 beta -3	252 MB/s	263 MB/s	1000000000	432914184	43.29	enwik9
lzrw 15-Jul-1991 -5	101 MB/s	355 MB/s	1000000000	436682071	43.67	enwik9
pithy 2011-12-24 -9	201 MB/s	1200 MB/s	1000000000	437729417	43.77	enwik9
slz_zlib 1.0.0 -3	164 MB/s	213 MB/s	1000000000	478256185	47.83	enwik9
fastlz 0.1 -2	210 MB/s	370 MB/s	1000000000	487260752	48.73	enwik9
yappy 2014-03-22 -99	72 MB/s	1574 MB/s	1000000000	492824491	49.28	enwik9
lzf 3.6 -1	214 MB/s	458 MB/s	1000000000	492987190	49.30	enwik9
blosclz 2015-11-10 -9	170 MB/s	634 MB/s	1000000000	498688572	49.87	enwik9
snappy 1.1.4	214 MB/s	889 MB/s	1000000000	507860747	50.79	enwik9
wflz 2015-09-16	165 MB/s	572 MB/s	1000000000	559915321	55.99	enwik9

```

25,749 lzjb 2010          176 MB/s   329 MB/s   1000000000   665072021   66.51 enwik9
25,750 shrinker 0.1      197 MB/s   5596 MB/s  1000000000   968576855   96.86 enwik9
25,751 */
25,752
25,753 /*
25,754 'Dragon Suffix Roster' revision 6
25,755 -----
25,756 | SORTED SUFFIXES | DRAGON          | LEPRECHAUN      |
25,757 -----
25,758 | -                | DRAGON          | LEPRECHAUN      |
25,759 | -adelic          | dragonadelic    | LEPRECHAUnadelic|
25,760 | -adelically      | dragonadelically| LEPRECHAUnadelically|
25,761 | -AGE             | DRAGONAGE       | LEPRECHAUNAGE    |
25,762 | -AGES            | DRAGONAGES      | LEPRECHAUNAGES   |
25,763 | -ALIA            | DRAGONALIA      | LEPRECHAUNALIA   |
25,764 | -ALIBUS          | DRAGONALIBUS    | LEPRECHAUNALIBUS |
25,765 | -ALIS            | DRAGONALIS      | LEPRECHAUNALIS   |
25,766 | -ALIUM           | DRAGONALIUM     | LEPRECHAUNALIUM  |
25,767 | -ANTHROP         | DRAGONANTHROP   | LEPRECHAUNANTHROP|
25,768 | -ANTHROPE        | DRAGONANTHROPE  | LEPRECHAUNANTHROPE|
25,769 | -ANTHROPES       | DRAGONANTHROPES | LEPRECHAUNANTHROPES|
25,770 | -ANTHROPI        | DRAGONANTHROPI  | LEPRECHAUNANTHROPI|
25,771 | -ANTHROPIA       | DRAGONANTHROPIA | LEPRECHAUNANTHROPIA|
25,772 | -ANTHROPIC        | DRAGONANTHROPIC | LEPRECHAUNANTHROPIC|
25,773 | -ANTHROPICALLY   | DRAGONANTHROPICALLY| LEPRECHAUNANTHROPICALLY|
25,774 | -ANTHROPIES      | DRAGONANTHROPIES| LEPRECHAUNANTHROPIES|
25,775 | -ANTHROPIST      | DRAGONANTHROPIST| LEPRECHAUNANTHROPIST|
25,776 | -ANTHROPISTS     | DRAGONANTHROPISTS| LEPRECHAUNANTHROPISTS|
25,777 | -ANTHROPOUS      | DRAGONANTHROPOUS| LEPRECHAUNANTHROPOUS|
25,778 | -ANTHROPS        | DRAGONANTHROPS  | LEPRECHAUNANTHROPS|
25,779 | -ANTHROPUS       | DRAGONANTHROPUS | LEPRECHAUNANTHROPUS|
25,780 | -ANTHROPY        | DRAGONANTHROPY  | LEPRECHAUNANTHROPY|
25,781 | -ARIA            | DRAGONARIA      | LEPRECHAUNARIA   |
25,782 | -ARIUM           | DRAGONARIUM     | LEPRECHAUNARIUM  |
25,783 | -ARIUMS          | DRAGONARIUMS    | LEPRECHAUNARIUMS |
25,784 | -aroo            | dragonaroo      | leprechaunaroo   |
25,785 | -aroonie         | dragonaroonie   | leprechaunaroonie|
25,786 | -ATA             | DRAGONATA       | LEPRECHAUNATA    |
25,787 | -ATE             | DRAGONATE       | LEPRECHAUNATE     |
25,788 | -ATES            | DRAGONATES      | LEPRECHAUNATES    |
25,789 | -ATICA           | DRAGONATICA     | LEPRECHAUNATICA   |
25,790 | -ATICUM          | DRAGONATICUM    | LEPRECHAUNATICUM  |
25,791 | -ATICUS          | DRAGONATICUS    | LEPRECHAUNATICUS  |
25,792 | -ATRICES         | DRAGONATRICES   | LEPRECHAUNATRICES|
25,793 | -ATRIX           | DRAGONATRIX     | LEPRECHAUNATRIX   |
25,794 | -ATRIKES        | DRAGONATRIKES   | LEPRECHAUNATRIKES|
25,795 | -ATUM            | DRAGONATUM      | LEPRECHAUNATUM    |
25,796 | -ATUS            | DRAGONATUS      | LEPRECHAUNATUS    |
25,797 | -aut             | dragonaut       | LEPRECHAUnaut     |
25,798 | -autic           | dragonautic     | LEPRECHAUnautic   |
25,799 | -autical         | dragonautical   | LEPRECHAUnautical|
25,800 | -autics          | dragonautics    | LEPRECHAUnautics  |
25,801 | -autrix          | dragonautrix    | LEPRECHAUnautrix  |
25,802 | -autte           | dragonautte     | LEPRECHAUnautte   |
25,803 | -BORN            | DRAGONBORN      | LEPRECHAUNBORN    |
25,804 | -BORNS           | DRAGONBORNS     | LEPRECHAUNBORNS   |
25,805 | -CEPHALY         | DRAGONCEPHALY   | LEPRECHAUNCEPHALY|
25,806 | -cholia          | DRAGONcholia    | LEPRECHAUNcholia  |

```

25,807	-choliac	DRAGONcholiac	LEPRECHAUNCHoliac
25,808	-cholian	DRAGONcholian	LEPRECHAUNCHolian
25,809	-cholically	DRAGONcholically	LEPRECHAUNCHolically
25,810	-cholie	DRAGONcholie	LEPRECHAUNCHolie
25,811	-cholies	DRAGONcholies	LEPRECHAUNCHolies
25,812	-cholily	DRAGONcholily	LEPRECHAUNCHolily
25,813	-choliness	DRAGONcholiness	LEPRECHAUNCHoliness
25,814	-cholious	DRAGONcholious	LEPRECHAUNCHolious
25,815	-cholist	DRAGONcholist	LEPRECHAUNCHolist
25,816	-cholists	DRAGONcholists	LEPRECHAUNCHolists
25,817	-cholize	DRAGONcholize	LEPRECHAUNCHolize
25,818	-choly	DRAGONcholy	LEPRECHAUNCHoly
25,819	-clad	dragonclad	leprechaunclad
25,820	-CRAFT	DRAGONCRAFT	LEPRECHAUNCRAFT
25,821	-CRAFTED	DRAGONCRAFTED	LEPRECHAUNCRAFTED
25,822	-CRAFTING	DRAGONCRAFTING	LEPRECHAUNCRAFTING
25,823	-DOM	DRAGONDOM	LEPRECHAUNDOM
25,824	-DOMS	DRAGONDOMS	LEPRECHAUNDOMS
25,825	-eer	dragoneer	LEPRECHAUNEer
25,826	-eers	dragonneers	LEPRECHAUNEers
25,827	-ella	DRAGONella	LEPRECHAUNella
25,828	-entelechial	dragonentelechial	LEPRECHAUNentelechial
25,829	-entelechially	dragonentelechially	LEPRECHAUNentelechially
25,830	-entelechy	dragonentelechy	LEPRECHAUNentelechy
25,831	-erast	DRAGONerast	LEPRECHAUNerast
25,832	-erastic	DRAGONerastic	LEPRECHAUNerastic
25,833	-erasts	DRAGONerasts	LEPRECHAUNerasts
25,834	-ERIE	DRAGONERIE	LEPRECHAUNERIE
25,835	-ERIES	DRAGONERIES	LEPRECHAUNERIES
25,836	-eroo	dragoneroo	leprechauneroo
25,837	-eroonie	dragoneroonie	leprechauneroonie
25,838	-ESE	DRAGONESE	LEPRECHAUNESE
25,839	-ESQUE	DRAGONESQUE	LEPRECHAUNESQUE
25,840	-ESQUELY	DRAGONESQUELY	LEPRECHAUNESQUELY
25,841	-ESS	DRAGONESS	LEPRECHAUNESS
25,842	-ESSES	DRAGONESSES	LEPRECHAUNESSES
25,843	-ETTE	DRAGONETTE	LEPRECHAUNETTE
25,844	-ette	dragonette	LEPRECHAUNette
25,845	-fulness	DRAGONfulness	LEPRECHAUNfulness
25,846	-HEAD	DRAGONHEAD	LEPRECHAUNHEAD
25,847	-HEART	DRAGONHEART	LEPRECHAUNHEART
25,848	-HEARTED	DRAGONHEARTED	LEPRECHAUNHEARTED
25,849	-HEARTEDLY	DRAGONHEARTEDLY	LEPRECHAUNHEARTEDLY
25,850	-HEARTEDNESS	DRAGONHEARTEDNESS	LEPRECHAUNHEARTEDNESS
25,851	-HOOD	DRAGONHOOD	LEPRECHAUNHOOD
25,852	-IA	DRAGONIA	LEPRECHAUNIA
25,853	-IACISM	DRAGONIACISM	LEPRECHAUNIACISM
25,854	-IAL	DRAGONIAL	LEPRECHAUNIAL
25,855	-IAN	DRAGONIAN	LEPRECHAUNIAN
25,856	-IANA	DRAGONIANA	LEPRECHAUNIANA
25,857	-IANICITY	DRAGONIANICITY	LEPRECHAUNIANICITY
25,858	-IANISM	DRAGONIANISM	LEPRECHAUNIANISM
25,859	-ianly	DRAGONianly	LEPRECHAUNianly
25,860	-IC	DRAGONIC	LEPRECHAUNIC
25,861	-ICA	DRAGONICA	LEPRECHAUNICA
25,862	-ICAE	DRAGONICAE	LEPRECHAUNICAE
25,863	-ICAL	DRAGONICAL	LEPRECHAUNICAL
25,864	-ICALIZATION	DRAGONICALIZATION	LEPRECHAUNICALIZATION

25,865	-ICALIZATIONS	DRAGONICALIZATIONS	LEPRECHAUNICALIZATIONS
25,866	-ICALIZATOR	DRAGONICALIZATOR	LEPRECHAUNICALIZATOR
25,867	-ICALIZATORS	DRAGONICALIZATORS	LEPRECHAUNICALIZATORS
25,868	-ICALIZATRESS	DRAGONICALIZATRESS	LEPRECHAUNICALIZATRESS
25,869	-ICALIZATRESSES	DRAGONICALIZATRESSES	LEPRECHAUNICALIZATRESSES
25,870	-ICALIZATRICES	DRAGONICALIZATRICES	LEPRECHAUNICALIZATRICES
25,871	-ICALIZATRIX	DRAGONICALIZATRIX	LEPRECHAUNICALIZATRIX
25,872	-ICALIZATRIKES	DRAGONICALIZATRIKES	LEPRECHAUNICALIZATRIKES
25,873	-ICALIZE-MENT	DRAGONICALIZEMENT	LEPRECHAUNICALIZEMENT
25,874	-ICALIZE-MENTS	DRAGONICALIZEMENTS	LEPRECHAUNICALIZEMENTS
25,875	-ICALIZE	DRAGONICALIZE	LEPRECHAUNICALIZE
25,876	-ICALIZED	DRAGONICALIZED	LEPRECHAUNICALIZED
25,877	-ICALIZER	DRAGONICALIZER	LEPRECHAUNICALIZER
25,878	-ICALIZERESS	DRAGONICALIZERESS	LEPRECHAUNICALIZERESS
25,879	-ICALIZERESSES	DRAGONICALIZERESSES	LEPRECHAUNICALIZERESSES
25,880	-ICALIZERS	DRAGONICALIZERS	LEPRECHAUNICALIZERS
25,881	-ICALIZES	DRAGONICALIZES	LEPRECHAUNICALIZES
25,882	-ICALIZING	DRAGONICALIZING	LEPRECHAUNICALIZING
25,883	-ICALLY	DRAGONICALLY	LEPRECHAUNICALLY
25,884	-ICALNESS	DRAGONICALNESS	LEPRECHAUNICALNESS
25,885	-ICAM	DRAGONICAM	LEPRECHAUNICAM
25,886	-ICARUM	DRAGONICARUM	LEPRECHAUNICARUM
25,887	-ICAS	DRAGONICAS	LEPRECHAUNICAS
25,888	-ICE	DRAGONICE	LEPRECHAUNICE
25,889	-ICHE	DRAGONICHE	LEPRECHAUNICHE
25,890	-ICI	DRAGONICI	LEPRECHAUNICI
25,891	-ICIAN	DRAGONICIAN	LEPRECHAUNICIAN
25,892	-ICIANS	DRAGONICIANS	LEPRECHAUNICIANS
25,893	-ICICITY	DRAGONICICITY	LEPRECHAUNICICITY
25,894	-ICIDAL	DRAGONICIDAL	LEPRECHAUNICIDAL
25,895	-ICIDE	DRAGONICIDE	LEPRECHAUNICIDE
25,896	-ICIS	DRAGONICIS	LEPRECHAUNICIS
25,897	-ICISM	DRAGONICISM	LEPRECHAUNICISM
25,898	-ICISMS	DRAGONICISMS	LEPRECHAUNICISMS
25,899	-icity	dragonicity	leprechaunicity
25,900	-ICIZATION	DRAGONICIZATION	LEPRECHAUNICIZATION
25,901	-ICIZATIONS	DRAGONICIZATIONS	LEPRECHAUNICIZATIONS
25,902	-ICIZATOR	DRAGONICIZATOR	LEPRECHAUNICIZATOR
25,903	-ICIZATORS	DRAGONICIZATORS	LEPRECHAUNICIZATORS
25,904	-ICIZATRESS	DRAGONICIZATRESS	LEPRECHAUNICIZATRESS
25,905	-ICIZATRESSES	DRAGONICIZATRESSES	LEPRECHAUNICIZATRESSES
25,906	-ICIZATRICES	DRAGONICIZATRICES	LEPRECHAUNICIZATRICES
25,907	-ICIZATRIX	DRAGONICIZATRIX	LEPRECHAUNICIZATRIX
25,908	-ICIZATRIKES	DRAGONICIZATRIKES	LEPRECHAUNICIZATRIKES
25,909	-ICIZE-MENT	DRAGONICIZEMENT	LEPRECHAUNICIZEMENT
25,910	-ICIZE-MENTS	DRAGONICIZEMENTS	LEPRECHAUNICIZEMENTS
25,911	-ICIZE	DRAGONICIZE	LEPRECHAUNICIZE
25,912	-ICIZED	DRAGONICIZED	LEPRECHAUNICIZED
25,913	-ICIZER	DRAGONICIZER	LEPRECHAUNICIZER
25,914	-ICIZERESS	DRAGONICIZERESS	LEPRECHAUNICIZERESS
25,915	-ICIZERESSES	DRAGONICIZERESSES	LEPRECHAUNICIZERESSES
25,916	-ICIZERS	DRAGONICIZERS	LEPRECHAUNICIZERS
25,917	-ICIZES	DRAGONICIZES	LEPRECHAUNICIZES
25,918	-ICIZING	DRAGONICIZING	LEPRECHAUNICIZING
25,919	-ICNESS	DRAGONICNESS	LEPRECHAUNICNESS
25,920	-ICO	DRAGONICO	LEPRECHAUNICO
25,921	-ICORUM	DRAGONICORUM	LEPRECHAUNICORUM
25,922	-ICOS	DRAGONICOS	LEPRECHAUNICOS

25,923	-ICS	DRAGONICS	LEPRECHAUNICS
25,924	-ICUM	DRAGONICUM	LEPRECHAUNICUM
25,925	-ICUS	DRAGONICUS	LEPRECHAUNICUS
25,926	-IDAS	DRAGONIDAS	LEPRECHAUNIDAS
25,927	-ier	DRAGONier	LEPRECHAUNier
25,928	-iest	DRAGONiest	LEPRECHAUNiest
25,929	-IFEROUS	DRAGONIFEROUS	LEPRECHAUNIFEROUS
25,930	-IFIABLE	DRAGONIFIABLE	LEPRECHAUNIFIABLE
25,931	-IFIC	DRAGONIFIC	LEPRECHAUNIFIC
25,932	-IFICATION	DRAGONIFICATION	LEPRECHAUNIFICATION
25,933	-IFIED	DRAGONIFIED	LEPRECHAUNIFIED
25,934	-IFIER	DRAGONIFIER	LEPRECHAUNIFIER
25,935	-IFIERS	DRAGONIFIERS	LEPRECHAUNIFIERS
25,936	-IFIES	DRAGONIFIES	LEPRECHAUNIFIES
25,937	-IFY-MENT	DRAGONIFYMENT	LEPRECHAUNIFYMENT
25,938	-IFY-MENTS	DRAGONIFYMENTS	LEPRECHAUNIFYMENTS
25,939	-IFY	DRAGONIFY	LEPRECHAUNIFY
25,940	-IFYING	DRAGONIFYING	LEPRECHAUNIFYING
25,941	-IFYINGS	DRAGONIFYINGS	LEPRECHAUNIFYINGS
25,942	-IGENOUS	DRAGONIGENOUS	LEPRECHAUNIGENOUS
25,943	-IGER	DRAGONIGER	LEPRECHAUNIGER
25,944	-IGEROUS	DRAGONIGEROUS	LEPRECHAUNIGEROUS
25,945	-INA	DRAGONINA	LEPRECHAUNINA
25,946	-INE	DRAGONINE	LEPRECHAUNINE
25,947	-INITY	DRAGONINITY	LEPRECHAUNINITY
25,948	-INUM	DRAGONINUM	LEPRECHAUNINUM
25,949	-INUS	DRAGONINUS	LEPRECHAUNINUS
25,950	-ISH	DRAGONISH	LEPRECHAUNISH
25,951	-ISHLY	DRAGONISHLY	LEPRECHAUNISHLY
25,952	-ISHNESS	DRAGONISHNESS	LEPRECHAUNISHNESS
25,953	-ISM	DRAGONISM	LEPRECHAUNISM
25,954	-ISMS	DRAGONISMS	LEPRECHAUNISMS
25,955	-ISSIMA	DRAGONISSIMA	LEPRECHAUNISSIMA
25,956	-ISSIMAE	DRAGONISSIMAE	LEPRECHAUNISSIMAE
25,957	-ISSIMI	DRAGONISSIMI	LEPRECHAUNISSIMI
25,958	-ISSIMO	DRAGONISSIMO	LEPRECHAUNISSIMO
25,959	-ISSIMUM	DRAGONISSIMUM	LEPRECHAUNISSIMUM
25,960	-ISSIMUS	DRAGONISSIMUS	LEPRECHAUNISSIMUS
25,961	-ISSIMUSES	DRAGONISSIMUSES	LEPRECHAUNISSIMUSES
25,962	-IST	DRAGONIST	LEPRECHAUNIST
25,963	-ISTESS	DRAGONISTESS	LEPRECHAUNISTESS
25,964	-ISTS	DRAGONISTS	LEPRECHAUNISTS
25,965	-ISTSSESSES	DRAGONISTSSESSES	LEPRECHAUNISTSSESSES
25,966	-ITARIAN	DRAGONITARIAN	LEPRECHAUNITARIAN
25,967	-ITARIANISM	DRAGONITARIANISM	LEPRECHAUNITARIANISM
25,968	-ITARY	DRAGONITARY	LEPRECHAUNITARY
25,969	-ITE	DRAGONITE	LEPRECHAUNITE
25,970	-itis	DRAGONitis	LEPRECHAUNITis
25,971	-ITUDE	DRAGONITUDE	LEPRECHAUNITUDE
25,972	-ITUDES	DRAGONITUDES	LEPRECHAUNITUDES
25,973	-ITY	DRAGONITY	LEPRECHAUNITY
25,974	-IUM	DRAGONIUM	LEPRECHAUNIUM
25,975	-IUS	DRAGONIUS	LEPRECHAUNIUS
25,976	-IZABILITY	DRAGONIZABILITY	LEPRECHAUNIZABILITY
25,977	-IZABLE	DRAGONIZABLE	LEPRECHAUNIZABLE
25,978	-IZATION	DRAGONIZATION	LEPRECHAUNIZATION
25,979	-IZATIONS	DRAGONIZATIONS	LEPRECHAUNIZATIONS
25,980	-IZATOR	DRAGONIZATOR	LEPRECHAUNIZATOR

25,981	-IZATORS	DRAGONIZATORS	LEPRECHAUNIZATORS
25,982	-IZATRESS	DRAGONIZATRESS	LEPRECHAUNIZATRESS
25,983	-IZATRESSES	DRAGONIZATRESSES	LEPRECHAUNIZATRESSES
25,984	-IZATRICES	DRAGONIZATRICES	LEPRECHAUNIZATRICES
25,985	-IZATRIX	DRAGONIZATRIX	LEPRECHAUNIZATRIX
25,986	-IZATRIKES	DRAGONIZATRIKES	LEPRECHAUNIZATRIKES
25,987	-IZE-MENT	DRAGONIZEMENT	LEPRECHAUNIZEMENT
25,988	-IZE-MENTS	DRAGONIZEMENTS	LEPRECHAUNIZEMENTS
25,989	-IZE	DRAGONIZE	LEPRECHAUNIZE
25,990	-IZED	DRAGONIZED	LEPRECHAUNIZED
25,991	-IZER	DRAGONIZER	LEPRECHAUNIZER
25,992	-IZERESS	DRAGONIZERESS	LEPRECHAUNIZERESS
25,993	-IZERESSES	DRAGONIZERESSES	LEPRECHAUNIZERESSES
25,994	-IZERS	DRAGONIZERS	LEPRECHAUNIZERS
25,995	-IZES	DRAGONIZES	LEPRECHAUNIZES
25,996	-IZING	DRAGONIZING	LEPRECHAUNIZING
25,997	-KIN	DRAGONKIN	LEPRECHAUNKIN
25,998	-KIND	DRAGONKIND	LEPRECHAUNKIND
25,999	-LAND	DRAGONLAND	LEPRECHAUNLAND
26,000	-LESS	DRAGONLESS	LEPRECHAUNLESS
26,001	-lessness	DRAGONlessness	LEPRECHAUNlessness
26,002	-LET	DRAGONLET	LEPRECHAUNLET
26,003	-LETS	DRAGONLETS	LEPRECHAUNLETS
26,004	-LIKE	DRAGONLIKE	LEPRECHAUNLIKE
26,005	-liness	DRAGONliness	LEPRECHAUNliness
26,006	-linesses	DRAGONlinesses	LEPRECHAUNlinesses
26,007	-LING	DRAGONLING	LEPRECHAUNLING
26,008	-LINGS	DRAGONLINGS	LEPRECHAUNLINGS
26,009	-LORAMA	DRAGONLORAMA	LEPRECHAUNLORAMA
26,010	-LORE	DRAGONLORE	LEPRECHAUNLORE
26,011	-LORES	DRAGONLORES	LEPRECHAUNLORES
26,012	-LORIC	DRAGONLORIC	LEPRECHAUNLORIC
26,013	-LORISH	DRAGONLORISH	LEPRECHAUNLORISH
26,014	-LORIST	DRAGONLORIST	LEPRECHAUNLORIST
26,015	-LORISTIC	DRAGONLORISTIC	LEPRECHAUNLORISTIC
26,016	-LORISTICS	DRAGONLORISTICS	LEPRECHAUNLORISTICS
26,017	-LORISTS	DRAGONLORISTS	LEPRECHAUNLORISTS
26,018	-LY	DRAGONLY	LEPRECHAUNLY
26,019	-NESS	DRAGONNESS	LEPRECHAUNNESS
26,020	-NESSes	DRAGONNESSes	LEPRECHAUNNESSes
26,021	-OCEPHALI	DRAGONOCEPHALI	LEPRECHAUNOCEPHALI
26,022	-OCEPHALIC	DRAGONOCEPHALIC	LEPRECHAUNOCEPHALIC
26,023	-OCEPHALISM	DRAGONOCEPHALISM	LEPRECHAUNOCEPHALISM
26,024	-OCEPHALOUS	DRAGONOCEPHALOUS	LEPRECHAUNOCEPHALOUS
26,025	-OCEPHALUS	DRAGONOCEPHALUS	LEPRECHAUNOCEPHALUS
26,026	-OCEPHALY	DRAGONOCEPHALY	LEPRECHAUNOCEPHALY
26,027	-OCRACIES	DRAGONOCRACIES	LEPRECHAUNOCRACIES
26,028	-OCRACY	DRAGONOCRACY	LEPRECHAUNOCRACY
26,029	-OCRAT	DRAGONOCRAT	LEPRECHAUNOCRAT
26,030	-OCRATIC	DRAGONOCRATIC	LEPRECHAUNOCRATIC
26,031	-OCRATICAL	DRAGONOCRATICAL	LEPRECHAUNOCRATICAL
26,032	-OCRATICALLY	DRAGONOCRATICALLY	LEPRECHAUNOCRATICALLY
26,033	-OCRATS	DRAGONOCRATS	LEPRECHAUNOCRATS
26,034	-ODOULIC	DRAGONODOULIC	LEPRECHAUNODOULIC
26,035	-ODOULIST	DRAGONODOULIST	LEPRECHAUNODOULIST
26,036	-ODOULISTS	DRAGONODOULISTS	LEPRECHAUNODOULISTS
26,037	-ODULE	DRAGONODULE	LEPRECHAUNODULE
26,038	-ODULES	DRAGONODULES	LEPRECHAUNODULES

26,039	-ODULIC	DRAGONODULIC	LEPRECHAUNODULIC
26,040	-ODULIST	DRAGONODULIST	LEPRECHAUNODULIST
26,041	-ODULISTS	DRAGONODULISTS	LEPRECHAUNODULISTS
26,042	-ODULY	DRAGONODULY	LEPRECHAUNODULY
26,043	-ogeny	dragonogeny	leprechaunogeny
26,044	-ognosis	dragonognosis	leprechaunognosis
26,045	-ogony	dragonogony	leprechaunogony
26,046	-ogram	DRAGONogram	LEPRECHAUNogram
26,047	-ogrammatic	DRAGONogrammatic	LEPRECHAUNogrammatic
26,048	-ogrammatical	DRAGONogrammatical	LEPRECHAUNogrammatical
26,049	-ogrammatically	DRAGONogrammatically	LEPRECHAUNogrammatically
26,050	-ogrammed	DRAGONogrammed	LEPRECHAUNogrammed
26,051	-ogramming	DRAGONogramming	LEPRECHAUNogramming
26,052	-ograms	DRAGONograms	LEPRECHAUNograms
26,053	-OGRAPH	DRAGONOGRAPH	LEPRECHAUNOGRAPH
26,054	-GRAPHER	DRAGONGRAPHER	LEPRECHAUNGRAPHER
26,055	-GRAPHERS	DRAGONGRAPHERS	LEPRECHAUNGRAPHERS
26,056	-GRAPHIC	DRAGONGRAPHIC	LEPRECHAUNGRAPHIC
26,057	-GRAPHICAL	DRAGONGRAPHICAL	LEPRECHAUNGRAPHICAL
26,058	-GRAPHICALLY	DRAGONGRAPHICALLY	LEPRECHAUNGRAPHICALLY
26,059	-GRAPHIES	DRAGONGRAPHIES	LEPRECHAUNGRAPHIES
26,060	-GRAPHIST	DRAGONGRAPHIST	LEPRECHAUNGRAPHIST
26,061	-GRAPHISTS	DRAGONGRAPHISTS	LEPRECHAUNGRAPHISTS
26,062	-ographomania	DRAGONographomania	LEPRECHAUNographomania
26,063	-ographomaniac	DRAGONographomaniac	LEPRECHAUNographomaniac
26,064	-ographomaniacs	DRAGONographomaniacs	LEPRECHAUNographomaniacs
26,065	-GRAPHS	DRAGONGRAPHS	LEPRECHAUNGRAPHS
26,066	-GRAPHY	DRAGONGRAPHY	LEPRECHAUNOGRAPHY
26,067	-ogyral	dragonogyral	leprechaunogyral
26,068	-oholic	DRAGONoholic	LEPRECHAUNoholic
26,069	-oholics	DRAGONoholics	LEPRECHAUNoholics
26,070	-OID	DRAGONOID	LEPRECHAUNOID
26,071	-OIDS	DRAGONOIDS	LEPRECHAUNOIDS
26,072	-olagnia	DRAGONolagnia	LEPRECHAUNolagnia
26,073	-olagniac	DRAGONolagniac	LEPRECHAUNolagniac
26,074	-olagniacs	DRAGONolagniacs	LEPRECHAUNolagniacs
26,075	-olagnist	DRAGONolagnist	LEPRECHAUNolagnist
26,076	-olagnists	DRAGONolagnists	LEPRECHAUNolagnists
26,077	-OLATER	DRAGONOLATER	LEPRECHAUNOLATER
26,078	-OLATERS	DRAGONOLATERS	LEPRECHAUNOLATERS
26,079	-OLATRIES	DRAGONOLATRIES	LEPRECHAUNOLATRIES
26,080	-OLATROUS	DRAGONOLATROUS	LEPRECHAUNOLATROUS
26,081	-OLATRY	DRAGONOLATRY	LEPRECHAUNOLATRY
26,082	-OLOGER	DRAGONOLOGER	LEPRECHAUNOLOGER
26,083	-LOGERS	DRAGONOLOGERS	LEPRECHAUNOLOGERS
26,084	-LOGIC	DRAGONOLOGIC	LEPRECHAUNOLOGIC
26,085	-LOGICAL	DRAGONOLOGICAL	LEPRECHAUNOLOGICAL
26,086	-LOGIST	DRAGONOLOGIST	LEPRECHAUNOLOGIST
26,087	-LOGISTS	DRAGONOLOGISTS	LEPRECHAUNOLOGISTS
26,088	-LOGY	DRAGONOLOGY	LEPRECHAUNOLOGY
26,089	-OMACH	DRAGONOMACH	LEPRECHAUNOMACH
26,090	-OMACHAL	DRAGONOMACHAL	LEPRECHAUNOMACHAL
26,091	-OMACHES	DRAGONOMACHES	LEPRECHAUNOMACHES
26,092	-OMACHIAN	DRAGONOMACHIAN	LEPRECHAUNOMACHIAN
26,093	-OMACHICAL	DRAGONOMACHICAL	LEPRECHAUNOMACHICAL
26,094	-OMACHIST	DRAGONOMACHIST	LEPRECHAUNOMACHIST
26,095	-OMACHISTS	DRAGONOMACHISTS	LEPRECHAUNOMACHISTS
26,096	-OMACHY	DRAGONOMACHY	LEPRECHAUNOMACHY

26,097	-OMAGY	DRAGONOMAGY	LEPRECHAUNOMAGY
26,098	-OMANCER	DRAGONOMANCER	LEPRECHAUNOMANCER
26,099	-OMANCERS	DRAGONOMANCERS	LEPRECHAUNOMANCERS
26,100	-OMANCY	DRAGONOMANCY	LEPRECHAUNOMANCY
26,101	-OMANIA	DRAGONOMANIA	LEPRECHAUNOMANIA
26,102	-OMANIAC	DRAGONOMANIAC	LEPRECHAUNOMANIAC
26,103	-OMANIACS	DRAGONOMANIACS	LEPRECHAUNOMANIACS
26,104	-OMANTIC	DRAGONOMANTIC	LEPRECHAUNOMANTIC
26,105	-OMANTICA	DRAGONOMANTICA	LEPRECHAUNOMANTICA
26,106	-OMANTICAL	DRAGONOMANTICAL	LEPRECHAUNOMANTICAL
26,107	-OMANTICALLY	DRAGONOMANTICALLY	LEPRECHAUNOMANTICALLY
26,108	-OMANTICHE	DRAGONOMANTICHE	LEPRECHAUNOMANTICHE
26,109	-OMANTICI	DRAGONOMANTICI	LEPRECHAUNOMANTICI
26,110	-OMANTICISM	DRAGONOMANTICISM	LEPRECHAUNOMANTICISM
26,111	-OMANTICO	DRAGONOMANTICO	LEPRECHAUNOMANTICO
26,112	-OMANTICRATIC	DRAGONOMANTICRATIC	LEPRECHAUNOMANTICRATIC
26,113	-OMANTICS	DRAGONOMANTICS	LEPRECHAUNOMANTICS
26,114	-OMATIC	DRAGONOMATIC	LEPRECHAUNOMATIC
26,115	-OMATICALLY	DRAGONOMATICALLY	LEPRECHAUNOMATICALLY
26,116	-OMATICISM	DRAGONOMATICISM	LEPRECHAUNOMATICISM
26,117	-OMATOGRAPHY	DRAGONOMATOGRAPHY	LEPRECHAUNOMATOGRAPHY
26,118	-OMER	DRAGONOMER	LEPRECHAUNOMER
26,119	-OMERS	DRAGONOMERS	LEPRECHAUNOMERS
26,120	-OMETER	DRAGONOMETER	LEPRECHAUNOMETER
26,121	-OMETRIC	DRAGONOMETRIC	LEPRECHAUNOMETRIC
26,122	-OMETRICAL	DRAGONOMETRICAL	LEPRECHAUNOMETRICAL
26,123	-OMETRICALLY	DRAGONOMETRICALLY	LEPRECHAUNOMETRICALLY
26,124	-OMETRY	DRAGONOMETRY	LEPRECHAUNOMETRY
26,125	-OMIA	DRAGONOMIA	LEPRECHAUNOMIA
26,126	-OMIAE	DRAGONOMIAE	LEPRECHAUNOMIAE
26,127	-OMIC	DRAGONOMIC	LEPRECHAUNOMIC
26,128	-OMICA	DRAGONOMICA	LEPRECHAUNOMICA
26,129	-OMICAL	DRAGONOMICAL	LEPRECHAUNOMICAL
26,130	-OMICALLY	DRAGONOMICALLY	LEPRECHAUNOMICALLY
26,131	-OMICON	DRAGONOMICON	LEPRECHAUNOMICON
26,132	-OMICS	DRAGONOMICS	LEPRECHAUNOMICS
26,133	-OMICUM	DRAGONOMICUM	LEPRECHAUNOMICUM
26,134	-OMICUS	DRAGONOMICUS	LEPRECHAUNOMICUS
26,135	-OMIES	DRAGONOMIES	LEPRECHAUNOMIES
26,136	-OMIST	DRAGONOMIST	LEPRECHAUNOMIST
26,137	-OMISTS	DRAGONOMISTS	LEPRECHAUNOMISTS
26,138	-OMY	DRAGONOMY	LEPRECHAUNOMY
26,139	-ONOMATIC	DRAGONONOMATIC	LEPRECHAUNONOMATIC
26,140	-ONOMATICALLY	DRAGONONOMATICALLY	LEPRECHAUNONOMATICALLY
26,141	-ONOMATICISM	DRAGONONOMATICISM	LEPRECHAUNONOMATICISM
26,142	-ONOMATOGRAPHY	DRAGONONOMATOGRAPHY	LEPRECHAUNONOMATOGRAPHY
26,143	-ONOMIC	DRAGONONOMIC	LEPRECHAUNONOMIC
26,144	-ONOMICS	DRAGONONOMICS	LEPRECHAUNONOMICS
26,145	-ONOMY	DRAGONONOMY	LEPRECHAUNONOMY
26,146	-OPHANIC	DRAGONOPHANIC	LEPRECHAUNOPHANIC
26,147	-OPHANIES	DRAGONOPHANIES	LEPRECHAUNOPHANIES
26,148	-OPHANOUS	DRAGONOPHANOUS	LEPRECHAUNOPHANOUS
26,149	-OPHANOUSLY	DRAGONOPHANOUSLY	LEPRECHAUNOPHANOUSLY
26,150	-OPHANY	DRAGONOPHANY	LEPRECHAUNOPHANY
26,151	-OPHILE	DRAGONOPHILE	LEPRECHAUNOPHILE
26,152	-OPHILES	DRAGONOPHILES	LEPRECHAUNOPHILES
26,153	-OPHILIA	DRAGONOPHILIA	LEPRECHAUNOPHILIA
26,154	-OPHILIAC	DRAGONOPHILIAC	LEPRECHAUNOPHILIAC

26,155	-OPHILIC	DRAGONOPHILIC	LEPRECHAUNOPHILIC
26,156	-OPHILISM	DRAGONOPHILISM	LEPRECHAUNOPHILISM
26,157	-OPHILIST	DRAGONOPHILIST	LEPRECHAUNOPHILIST
26,158	-OPHILISTS	DRAGONOPHILISTS	LEPRECHAUNOPHILISTS
26,159	-OPHILOUS	DRAGONOPHILOUS	LEPRECHAUNOPHILOUS
26,160	-OPHILY	DRAGONOPHILY	LEPRECHAUNOPHILY
26,161	-OPHOBE	DRAGONOPHOBE	LEPRECHAUNOPHOBE
26,162	-OPHOBES	DRAGONOPHOBES	LEPRECHAUNOPHOBES
26,163	-OPHOBIA	DRAGONOPHOBIA	LEPRECHAUNOPHOBIA
26,164	-OPHOBI	DRAGONOPHOBI	LEPRECHAUNOPHOBI
26,165	-ophonic	DRAGONophonic	LEPRECHAUNophonic
26,166	-ophony	DRAGONophony	LEPRECHAUNophony
26,167	-OPHORIA	DRAGONOPHORIA	LEPRECHAUNOPHORIA
26,168	-OPHORIANT	DRAGONOPHORIANT	LEPRECHAUNOPHORIANT
26,169	-OPHORIANTS	DRAGONOPHORIANTS	LEPRECHAUNOPHORIANTS
26,170	-OPHORIAS	DRAGONOPHORIAS	LEPRECHAUNOPHORIAS
26,171	-OPHORIC	DRAGONOPHORIC	LEPRECHAUNOPHORIC
26,172	-OPHORICALLY	DRAGONOPHORICALLY	LEPRECHAUNOPHORICALLY
26,173	-OPHORICS	DRAGONOPHORICS	LEPRECHAUNOPHORIC
26,174	-OPLAST	DRAGONOPLAST	LEPRECHAUNOPLAST
26,175	-OPLASTS	DRAGONOPLASTS	LEPRECHAUNOPLASTS
26,176	-OPOLIS	DRAGONOPOLIS	LEPRECHAUNOPOLIS
26,177	-ORAMA	DRAGONORAMA	LEPRECHAUNORAMA
26,178	-oramic	dragonoramic	leprechaunoramic
26,179	-oramical	dragonoramical	leprechaunoramical
26,180	-oramically	dragonoramically	leprechaunoramically
26,181	-OSCOPE	DRAGONOSCOPE	LEPRECHAUNOSCOPE
26,182	-OSCOPES	DRAGONOSCOPES	LEPRECHAUNOSCOPES
26,183	-OSCOPY	DRAGONOSCOPY	LEPRECHAUNOSCOPY
26,184	-OSITIES	DRAGONOSITIES	LEPRECHAUNOSITIES
26,185	-OSITY	DRAGONOSITY	LEPRECHAUNOSITY
26,186	-OSTAS	DRAGONOSTAS	LEPRECHAUNOSTAS
26,187	-OSTASE	DRAGONOSTASE	LEPRECHAUNOSTASE
26,188	-OSTASES	DRAGONOSTASES	LEPRECHAUNOSTASES
26,189	-OSTASIS	DRAGONOSTASIS	LEPRECHAUNOSTASIS
26,190	-othetic	dragonothetic	leprechaunothetic
26,191	-othetical	dragonothetical	leprechaunothetical
26,192	-othetically	dragonothetically	leprechaunothetically
26,193	-OTHYMIA	DRAGONOTHYMIA	LEPRECHAUNOTHYMIA
26,194	-OTHYMIC	DRAGONOTHYMIC	LEPRECHAUNOTHYMIC
26,195	-OTHYMICLY	DRAGONOTHYMICLY	LEPRECHAUNOTHYMICLY
26,196	-otopia	DRAGONotopia	LEPRECHAUNotopia
26,197	-otopian	DRAGONotopian	LEPRECHAUNotopian
26,198	-otopias	DRAGONotopias	LEPRECHAUNotopias
26,199	-otopic	DRAGONotopic	LEPRECHAUNotopic
26,200	-OUS	DRAGONOUS	LEPRECHAUNOUS
26,201	-ovenator	DRAGONovenator	LEPRECHAUNovenator
26,202	-ovenators	DRAGONovenators	LEPRECHAUNovenators
26,203	-path	DRAGONopath	LEPRECHAUNopath
26,204	-pathic	DRAGONopathic	LEPRECHAUNopathic
26,205	-pathically	DRAGONopathically	LEPRECHAUNopathically
26,206	-pathics	DRAGONopathics	LEPRECHAUNopathics
26,207	-pathies	DRAGONopathies	LEPRECHAUNopathies
26,208	-pathist	DRAGONopathist	LEPRECHAUNopathist
26,209	-pathists	DRAGONopathists	LEPRECHAUNopathists
26,210	-paths	DRAGONopaths	LEPRECHAUNopaths
26,211	-pathy	DRAGONopathy	LEPRECHAUNopathy
26,212	-RIES	DRAGONRIES	LEPRECHAUNRIES

```

26,213 | -RY          | DRAGONRY          | LEPRECHAUNRY      |
26,214 | -S           | DRAGONS           | LEPRECHAUNS       |
26,215 | -SHIP        | DRAGONSHIP        | LEPRECHAUNSHIP    |
26,216 | -SMITH       | DRAGONSMITH       | LEPRECHAUNSMITH   |
26,217 | -smithing    | dragonsmithing    | leprechaunsmithing|
26,218 | -SMITHS      | DRAGONSMITHS      | LEPRECHAUNSMITHS  |
26,219 | -smithying   | dragonsmithying   | leprechaunsmithying|
26,220 | -SOME        | DRAGONSOME        | LEPRECHAUNSOME    |
26,221 | -SOMELY      | DRAGONSONELY      | LEPRECHAUNSONELY  |
26,222 | -SOMENESS    | DRAGONSONENESS    | LEPRECHAUNSONENESS|
26,223 | -SOMER       | DRAGONSONER       | LEPRECHAUNSONER   |
26,224 | -SOMEST      | DRAGONSONEST      | LEPRECHAUNSONEST  |
26,225 | -STONE       | DRAGONSTONE       | LEPRECHAUNSTONE   |
26,226 | -STONES      | DRAGONSTONES      | LEPRECHAUNSTONES  |
26,227 | -TASTIC      | DRAGONTASTIC      | LEPRECHAUNTASTIC  |
26,228 | -TASTICLY    | DRAGONTASTICLY    | LEPRECHAUNTASTICLY|
26,229 | -TROPE       | DRAGONOTROPE      | LEPRECHAUNOTROPE  |
26,230 | -TROPIC      | DRAGONOTROPIC     | LEPRECHAUNOTROPIC |
26,231 | -TROPISM     | DRAGONOTROPISM    | LEPRECHAUNOTROPISM|
26,232 | -TROPISMS    | DRAGONOTROPISMS   | LEPRECHAUNOTROPISMS|
26,233 | -TROPIST     | DRAGONOTROPIST    | LEPRECHAUNOTROPIST|
26,234 | -TROPISTS    | DRAGONOTROPISTS   | LEPRECHAUNOTROPISTS|
26,235 | -TROPY       | DRAGONOTROPY      | LEPRECHAUNOTROPY  |
26,236 | -URGIST      | DRAGONURGIST      | LEPRECHAUNURGIST  |
26,237 | -URGISTS     | DRAGONURGISTS     | LEPRECHAUNURGISTS |
26,238 | -URGY        | DRAGONURGY        | LEPRECHAUNURGY    |
26,239 | -WARD        | DRAGONWARD        | LEPRECHAUNWARD    |
26,240 | -WARDS       | DRAGONWARDS       | LEPRECHAUNWARDS   |
26,241 | -WISE        | DRAGONWISE        | LEPRECHAUNWISE    |
26,242 | -WITH        | DRAGONWITH        | LEPRECHAUNWITH    |
26,243 | -WYF         | DRAGONWYF         | LEPRECHAUNWYF     |
26,244 | -Y           | DRAGONY           | LEPRECHAUNY       |

```

```

26,245 -----
26,246 Latest revision: 2019-Apr-24
26,247 Total suffixes: 486
26,248 Homethread: https://forum.thefreedictionary.com/postst175564\_Etymology-of--dragon-.aspx
26,249 */
26,250 /*
26,251 /*
26,252 Star_Trek_-_737_TXT-Ebooks.tar (325,071,872 bytes, UTF-8, English/ASCII text mostly), 29bit hashtable:
26,253

```

```

26,254 -----
26,255 | Hasher, Building-Blocks order | Collisions |
26,256 -----
26,257 | iSCSI_CRC32 : All Keys 01 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (171 - 171) = [0] |
26,258 | FNV1A_Pippip: All Keys 01 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (171 - 171) = [0] |
26,259 | WYHASH_v3   : All Keys 01 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (171 - 171) = [0] |
26,260 | XXH3        : All Keys 01 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (171 - 171) = [0] |
26,261 -----
26,262 | iSCSI_CRC32 : All Keys 02 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (7,540 - 7,540) = [0] |
26,263 | FNV1A_Pippip: All Keys 02 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (7,540 - 7,540) = [0] |
26,264 | WYHASH_v3   : All Keys 02 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (7,540 - 7,540) = [0] |
26,265 | XXH3        : All Keys 02 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (7,540 - 7,540) = [0] |
26,266 -----
26,267 | iSCSI_CRC32 : All Keys 03 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (81,269 - 81,269) = [0] |
26,268 | FNV1A_Pippip: All Keys 03 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (81,269 - 81,265) = [4] |
26,269 | WYHASH_v3   : All Keys 03 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (81,269 - 81,263) = [6] |
26,270 | XXH3        : All Keys 03 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (81,269 - 81,262) = [7] |

```

Page 452 of 466

```

26,329 | FNV1A_Pippip: All Keys 03 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (779,296 - 774,956) = [4,340] |
26,330 | WYHASH_v3 : All Keys 03 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (779,296 - 774,722) = [4,574] |
26,331 | XXH3 : All Keys 03 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (779,296 - 774,792) = [4,504] |
26,332 | -----
26,333 | iSCSI_CRC32 : All Keys 04 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (3,486,913 - 3,402,913) = [84,000] |
26,334 | FNV1A_Pippip: All Keys 04 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (3,486,913 - 3,397,884) = [89,029] |
26,335 | WYHASH_v3 : All Keys 04 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (3,486,913 - 3,397,698) = [89,215] |
26,336 | XXH3 : All Keys 04 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (3,486,913 - 3,397,593) = [89,320] |
26,337 | -----
26,338 | iSCSI_CRC32 : All Keys 05 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (6,753,621 - 6,421,931) = [331,690] |
26,339 | FNV1A_Pippip: All Keys 05 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (6,753,621 - 6,424,985) = [328,636] |
26,340 | WYHASH_v3 : All Keys 05 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (6,753,621 - 6,425,636) = [327,985] |
26,341 | XXH3 : All Keys 05 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (6,753,621 - 6,424,949) = [328,672] |
26,342 | -----
26,343 | iSCSI_CRC32 : All Keys 06 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (9,745,315 - 9,069,101) = [676,214] |
26,344 | FNV1A_Pippip: All Keys 06 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (9,745,315 - 9,069,787) = [675,528] |
26,345 | WYHASH_v3 : All Keys 06 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (9,745,315 - 9,071,273) = [674,042] |
26,346 | XXH3 : All Keys 06 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (9,745,315 - 9,070,217) = [675,098] |
26,347 | -----
26,348 | iSCSI_CRC32 : All Keys 07 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (12,146,222 - 11,111,934) = [1,034,288] |
26,349 | FNV1A_Pippip: All Keys 07 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (12,146,222 - 11,109,915) = [1,036,307] |
26,350 | WYHASH_v3 : All Keys 07 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (12,146,222 - 11,110,179) = [1,036,043] |
26,351 | XXH3 : All Keys 07 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (12,146,222 - 11,110,481) = [1,035,741] |
26,352 | -----
26,353 | iSCSI_CRC32 : All Keys 08 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (13,922,118 - 12,573,815) = [1,348,303] |
26,354 | FNV1A_Pippip: All Keys 08 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (13,922,118 - 12,507,408) = [1,414,710] |
26,355 | WYHASH_v3 : All Keys 08 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (13,922,118 - 12,572,063) = [1,350,055] |
26,356 | XXH3 : All Keys 08 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (13,922,118 - 12,572,988) = [1,349,130] |
26,357 | -----
26,358 | iSCSI_CRC32 : All Keys 09 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (15,567,034 - 13,894,829) = [1,672,205] |
26,359 | FNV1A_Pippip: All Keys 09 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (15,567,034 - 13,827,123) = [1,739,911] |
26,360 | WYHASH_v3 : All Keys 09 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (15,567,034 - 13,892,591) = [1,674,443] |
26,361 | XXH3 : All Keys 09 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (15,567,034 - 13,893,522) = [1,673,512] |
26,362 | -----
26,363 | iSCSI_CRC32 : All Keys 10 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (16,911,322 - 14,948,437) = [1,962,885] |
26,364 | FNV1A_Pippip: All Keys 10 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (16,911,322 - 14,882,178) = [2,029,144] |
26,365 | WYHASH_v3 : All Keys 10 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (16,911,322 - 14,949,193) = [1,962,129] |
26,366 | XXH3 : All Keys 10 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (16,911,322 - 14,948,130) = [1,963,192] |
26,367 | -----
26,368 | iSCSI_CRC32 : All Keys 11 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (18,326,389 - 16,037,613) = [2,288,776] |
26,369 | FNV1A_Pippip: All Keys 11 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (18,326,389 - 15,971,799) = [2,354,590] |
26,370 | WYHASH_v3 : All Keys 11 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (18,326,389 - 16,040,122) = [2,286,267] |
26,371 | XXH3 : All Keys 11 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (18,326,389 - 16,037,900) = [2,288,489] |
26,372 | -----
26,373 |
26,374 | enwik9 (1,000,000,000 bytes, UTF-8, XML), 30bit hashtable:
26,375 |
26,376 | -----
26,377 | Hasher, Building-Blocks order | Collisions |
26,378 | -----
26,379 | iSCSI_CRC32 : All Keys 01 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (206 - 206) = [0] |
26,380 | FNV1A_Pippip: All Keys 01 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (206 - 206) = [0] |
26,381 | WYHASH_v3 : All Keys 01 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (206 - 206) = [0] |
26,382 | XXH3 : All Keys 01 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (206 - 206) = [0] |
26,383 | -----
26,384 | iSCSI_CRC32 : All Keys 02 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (22,664 - 22,664) = [0] |
26,385 | FNV1A_Pippip: All Keys 02 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (22,664 - 22,664) = [0] |
26,386 | WYHASH_v3 : All Keys 02 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (22,664 - 22,664) = [0] |

```

```

26,387 | XXH3      : All Keys 02 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (22,664 - 22,664) = [0] |
26,388 -----
26,389 | iSCSI_CRC32 : All Keys 03 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (491,817 - 491,817) = [0] |
26,390 | FNV1A_Pippip: All Keys 03 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (491,817 - 491,738) = [79] |
26,391 | WYHASH_v3   : All Keys 03 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (491,817 - 491,709) = [108] |
26,392 | XXH3      : All Keys 03 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (491,817 - 491,717) = [100] |
26,393 -----
26,394 | iSCSI_CRC32 : All Keys 04 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (3,222,263 - 3,212,764) = [9,499] |
26,395 | FNV1A_Pippip: All Keys 04 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (3,222,263 - 3,217,541) = [4,722] |
26,396 | WYHASH_v3   : All Keys 04 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (3,222,263 - 3,217,456) = [4,807] |
26,397 | XXH3      : All Keys 04 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (3,222,263 - 3,217,511) = [4,752] |
26,398 -----
26,399 | iSCSI_CRC32 : All Keys 05 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (11,375,257 - 11,327,748) = [47,509] |
26,400 | FNV1A_Pippip: All Keys 05 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (11,375,257 - 11,314,689) = [60,568] |
26,401 | WYHASH_v3   : All Keys 05 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (11,375,257 - 11,315,057) = [60,200] |
26,402 | XXH3      : All Keys 05 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (11,375,257 - 11,315,286) = [59,971] |
26,403 -----
26,404 | iSCSI_CRC32 : All Keys 06 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (28,245,411 - 27,879,922) = [365,489] |
26,405 | FNV1A_Pippip: All Keys 06 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (28,245,411 - 27,876,238) = [369,173] |
26,406 | WYHASH_v3   : All Keys 06 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (28,245,411 - 27,878,277) = [367,134] |
26,407 | XXH3      : All Keys 06 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (28,245,411 - 27,877,986) = [367,425] |
26,408 -----
26,409 | iSCSI_CRC32 : All Keys 07 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (53,725,626 - 52,405,388) = [1,320,238] |
26,410 | FNV1A_Pippip: All Keys 07 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (53,725,626 - 52,404,876) = [1,320,750] |
26,411 | WYHASH_v3   : All Keys 07 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (53,725,626 - 52,403,276) = [1,322,350] |
26,412 | XXH3      : All Keys 07 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (53,725,626 - 52,403,054) = [1,322,572] |
26,413 -----
26,414 | iSCSI_CRC32 : All Keys 08 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (87,951,564 - 84,448,947) = [3,502,617] |
26,415 | FNV1A_Pippip: All Keys 08 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (87,951,564 - 84,445,511) = [3,506,053] |
26,416 | WYHASH_v3   : All Keys 08 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (87,951,564 - 84,446,072) = [3,505,492] |
26,417 | XXH3      : All Keys 08 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (87,951,564 - 84,444,323) = [3,507,241] |
26,418 -----
26,419 | iSCSI_CRC32 : All Keys 09 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (130,682,041 - 123,041,596) = [7,640,445] |
26,420 | FNV1A_Pippip: All Keys 09 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (130,682,041 - 123,049,405) = [7,632,636] |
26,421 | WYHASH_v3   : All Keys 09 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (130,682,041 - 123,041,605) = [7,640,436] |
26,422 | XXH3      : All Keys 09 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (130,682,041 - 123,041,777) = [7,640,264] |
26,423 -----
26,424 | iSCSI_CRC32 : All Keys 10 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (180,821,533 - 166,411,474) = [14,410,059] |
26,425 | FNV1A_Pippip: All Keys 10 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (180,821,533 - 166,415,343) = [14,406,190] |
26,426 | WYHASH_v3   : All Keys 10 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (180,821,533 - 166,413,129) = [14,408,404] |
26,427 | XXH3      : All Keys 10 bytes; | (Distinct_Keys - Used_Slots) = [Hash_Collisions]: (180,821,533 - 166,417,790) = [14,403,743] |
26,428 -----
26,429
26,430 FNV1A_Pippip_Yurii:
26,431     mov     rax, QWORD PTR [rdi]
26,432     cmp     rsi, 8
26,433     jbe     .L2
26,434     lea     rax, [rsi-1]
26,435     shr     rax, 4
26,436     lea     rdx, [8+rax*8]
26,437     movabs  rax, -3750763034362895579
26,438     sub     rsi, rdx
26,439     add     rdx, rdi
26,440 .L3:
26,441     xor     rax, QWORD PTR [rdi]
26,442     add     rdi, 8
26,443     imul    rax, rax, 591798841
26,444     xor     rax, QWORD PTR [rdi-8+rsi]

```

```

26,445      imul    rax, rax, 591798841
26,446      cmp     rdi, rdx
26,447      jne     .L3
26,448 .L4:
26,449      mov     rdx, rax
26,450      shr     rdx, 32
26,451      xor     eax, edx
26,452      mov     edx, eax
26,453      shr     edx, 16
26,454      xor     eax, edx
26,455      ret
26,456 .L2:
26,457      movabs   rdx, -3750763034362895579
26,458      mov     ecx, 8
26,459      sub     ecx, esi
26,460      sal     ecx, 3
26,461      sal     rax, cl
26,462      shr     rax, cl
26,463      xor     rax, rdx
26,464      imul    rax, rax, 591798841
26,465      jmp     .L4
26,466
26,467 Source: https://godbolt.org/z/D9MyQQ
26,468
26,469 // Dedicated to Pippip, the main character in the 'Das Totenschiff' roman, actually the B.Traven himself, his real name was Hermann Albert Otto Maksymilian Feige.
26,470 // CAUTION: Add 8 more bytes to the buffer being hashed, usually malloc(...+8) - to prevent out of boundary reads!
26,471 // Many thanks go to Yurii 'Hordi' Hordiienko, he lessened with 3 instructions the original 'Pippip', thus:
26,472 #include <stdlib.h>
26,473 #include <stdint.h>
26,474 #define _PADr_KAZE(x, n) ( ((x) << (n)) >> (n) )
26,475 uint32_t FNV1A_Pippip_Yurii(const char *str, size_t wrdlen) {
26,476     const uint32_t PRIME = 591798841;
26,477     uint32_t hash32; uint64_t hash64 = 14695981039346656037ULL;
26,478     size_t Cycles, NDhead;
26,479     if (wrdlen > 8) {
26,480         Cycles = ((wrdlen - 1) >> 4) + 1; NDhead = wrdlen - (Cycles << 3);
26,481         #pragma nounroll
26,482         for(; Cycles--; str += 8) {
26,483             hash64 = ( hash64 ^ (*(uint64_t *) (str)) ) * PRIME;
26,484             hash64 = ( hash64 ^ (*(uint64_t *) (str+NDhead)) ) * PRIME;
26,485         }
26,486     } else
26,487     hash64 = ( hash64 ^ _PADr_KAZE(*(uint64_t *) (str+0), (8-wrdlen) << 3) ) * PRIME;
26,488     hash32 = (uint32_t) (hash64 ^ (hash64 >> 32)); return hash32 ^ (hash32 >> 16);
26,489 } // Last update: 2019-Oct-30, 14 C lines strong, Kaze.
26,490 */
26,491
26,492 // 2021-Jan-09, Dell i7-1065G7 Ice Lake-U 3900MHz, 64GB DDR4 2133MHz [
26,493 /*
26,494 C:\Double-Deuce_8>dir
26,495
26,496 01/08/2021  05:55                202,240 Nakamichi_DD-192.exe
26,497 01/08/2021  05:55                201,728 Nakamichi_PRV-192.exe
26,498 12/26/2020  19:23          524,794,368 SUPRAPIG_The_Prague_Stringology_Club_(302-PostScript-files).tar
26,499
26,500 C:\Double-Deuce_8>sha1sum "SUPRAPIG_The_Prague_Stringology_Club_(302-PostScript-files).tar"
26,501 79b9c48428daa2c7bd606df7f139423bd2c9c7d2  SUPRAPIG_The_Prague_Stringology_Club_(302-PostScript-files).tar
26,502

```

```
26,503 C:\Double-Deuce_8>timer64 Nakamichi_PRV-192.exe "SUPRAPIG_The_Prague_Stringology_Club_(302-PostScript-files).tar" "SUPRAPIG_The_Prague_Stringology_Club_(302-PostScript-files).tar.Nakamichi" 20 58000 i
26,504 ...
26,505 Current priority class is HIGH_PRIORITY_CLASS.
26,506 This compile uses B-trees only, no memmem() invocations - it compresses worse but much faster.
26,507 Allocating Source-Buffer 500 MB ...
26,508 Allocating Target-Buffer 820 MB ...
26,509 Source-file-Hash(FNV1A_YoshimitsuTRIAD) = 0xa5a8,8e5f
26,510 Leprechaun: Memory pool for B-trees is 58,000 MB.
26,511 Leprechaun: In this revision 8MB 16-way hash is used which results in 16 x 1,048,576 internal B-Trees of order 3.
26,512 Leprechaun: Allocating HASH memory 142,606,401 bytes = 8*(16+1)*2^(20) ... OK
26,513 Leprechaun: Allocating memory for B-trees 58001 MB ... OK
26,514 Leprechaun: Size of input file: 524,794,368
26,515
26,516 Leprechaun: Using (first 24 bytes of) PRV-224 for Matches 24+ long, in order to reduce memory footprint.
26,517 Leprechaun: Inserting keys/BBs of order 004 into B-trees, free RAM in B-tree pool is 00,058,000 MB; Pass #000,001 of 000,001 ...
26,518 Leprechaun: Failure! Increment 'Memory for B-trees'!
26,519 Automatically increasing number of passes in order to fit B-trees into Target Buffer.
26,520 Leprechaun: Inserting keys/BBs of order 004 into B-trees, free RAM in B-tree pool is 00,057,844 MB; Pass #000,002 of 000,002 ... DONE; 00,001,048,208 DEFRAGMENTED B-trees have been rooted so far.
26,521      8,308,154 TotalNonUnique (of length 004) keys have been inserted into DEFRAGMENTED B-trees.
26,522 Leprechaun: Inserting keys/BBs of order 006 into B-trees, free RAM in B-tree pool is 00,057,689 MB; Pass #000,001 of 000,002 ...
26,523 Leprechaun: Failure! Increment 'Memory for B-trees'!
26,524 Automatically increasing number of passes in order to fit B-trees into Target Buffer.
26,525 Leprechaun: Inserting keys/BBs of order 006 into B-trees, free RAM in B-tree pool is 00,057,082 MB; Pass #000,004 of 000,004 ... DONE; 00,002,096,784 DEFRAGMENTED B-trees have been rooted so far.
26,526      20,181,897 TotalNonUnique (of length 006) keys have been inserted into DEFRAGMENTED B-trees.
26,527 Leprechaun: Inserting keys/BBs of order 008 into B-trees, free RAM in B-tree pool is 00,056,879 MB; Pass #000,001 of 000,004 ...
26,528 Leprechaun: Failure! Increment 'Memory for B-trees'!
26,529 Automatically increasing number of passes in order to fit B-trees into Target Buffer.
26,530 Leprechaun: Inserting keys/BBs of order 008 into B-trees, free RAM in B-tree pool is 00,056,052 MB; Pass #000,008 of 000,008 ... DONE; 00,003,145,360 DEFRAGMENTED B-trees have been rooted so far.
26,531      21,209,568 TotalNonUnique (of length 008) keys have been inserted into DEFRAGMENTED B-trees.
26,532 Leprechaun: Inserting keys/BBs of order 010 into B-trees, free RAM in B-tree pool is 00,054,965 MB; Pass #000,008 of 000,008 ... DONE; 00,004,193,936 DEFRAGMENTED B-trees have been rooted so far.
26,533      23,049,939 TotalNonUnique (of length 010) keys have been inserted into DEFRAGMENTED B-trees.
26,534 Leprechaun: Inserting keys/BBs of order 012 into B-trees, free RAM in B-tree pool is 00,053,758 MB; Pass #000,008 of 000,008 ... DONE; 00,005,242,512 DEFRAGMENTED B-trees have been rooted so far.
26,535      23,785,884 TotalNonUnique (of length 012) keys have been inserted into DEFRAGMENTED B-trees.
26,536 Leprechaun: Inserting keys/BBs of order 014 into B-trees, free RAM in B-tree pool is 00,052,456 MB; Pass #000,008 of 000,008 ... DONE; 00,006,291,088 DEFRAGMENTED B-trees have been rooted so far.
26,537      24,025,437 TotalNonUnique (of length 014) keys have been inserted into DEFRAGMENTED B-trees.
26,538 Leprechaun: Inserting keys/BBs of order 016 into B-trees, free RAM in B-tree pool is 00,052,292 MB; Pass #000,001 of 000,008 ...
26,539 Leprechaun: Failure! Increment 'Memory for B-trees'!
26,540 Automatically increasing number of passes in order to fit B-trees into Target Buffer.
26,541 Leprechaun: Inserting keys/BBs of order 016 into B-trees, free RAM in B-tree pool is 00,050,998 MB; Pass #000,016 of 000,016 ... DONE; 00,007,339,664 DEFRAGMENTED B-trees have been rooted so far.
26,542      23,807,316 TotalNonUnique (of length 016) keys have been inserted into DEFRAGMENTED B-trees.
26,543 Leprechaun: Inserting keys/BBs of order 018 into B-trees, free RAM in B-tree pool is 00,049,558 MB; Pass #000,016 of 000,016 ... DONE; 00,008,388,240 DEFRAGMENTED B-trees have been rooted so far.
26,544      23,567,825 TotalNonUnique (of length 018) keys have been inserted into DEFRAGMENTED B-trees.
26,545 Leprechaun: Inserting keys/BBs of order 036 into B-trees, free RAM in B-tree pool is 00,047,867 MB; Pass #000,016 of 000,016 ... DONE; 00,009,436,816 DEFRAGMENTED B-trees have been rooted so far.
26,546      23,699,242 TotalNonUnique (of length 036) keys have been inserted into DEFRAGMENTED B-trees.
26,547 Leprechaun: Inserting keys/BBs of order 064 into B-trees, free RAM in B-tree pool is 00,046,064 MB; Pass #000,016 of 000,016 ... DONE; 00,010,485,392 DEFRAGMENTED B-trees have been rooted so far.
26,548      25,031,430 TotalNonUnique (of length 064) keys have been inserted into DEFRAGMENTED B-trees.
26,549 Leprechaun: Inserting keys/BBs of order 024 into B-trees, free RAM in B-tree pool is 00,044,413 MB; Pass #000,016 of 000,016 ... DONE; 00,011,533,968 DEFRAGMENTED B-
```



```

trees have been rooted so far.
26,550      23,082,831 TotalNonUnique (of length 024) keys have been inserted into DEFRAGMENTED B-trees.
26,551 Leprechaun: Inserting keys/BBs of order 028 into B-trees, free RAM in B-tree pool is 00,042,746 MB; Pass #000,016 of 000,016 ... DONE; 00,012,582,544 DEFRAGMENTED B-
trees have been rooted so far.
26,552      23,219,460 TotalNonUnique (of length 028) keys have been inserted into DEFRAGMENTED B-trees.
26,553 Leprechaun: Inserting keys/BBs of order 048 into B-trees, free RAM in B-tree pool is 00,040,991 MB; Pass #000,016 of 000,016 ... DONE; 00,013,631,120 DEFRAGMENTED B-
trees have been rooted so far.
26,554      24,408,433 TotalNonUnique (of length 048) keys have been inserted into DEFRAGMENTED B-trees.
26,555 Leprechaun: Inserting keys/BBs of order 056 into B-trees, free RAM in B-tree pool is 00,039,203 MB; Pass #000,016 of 000,016 ... DONE; 00,014,679,696 DEFRAGMENTED B-
trees have been rooted so far.
26,556      24,776,643 TotalNonUnique (of length 056) keys have been inserted into DEFRAGMENTED B-trees.
26,557 Leprechaun: Inserting keys/BBs of order 128 into B-trees, free RAM in B-tree pool is 00,037,337 MB; Pass #000,016 of 000,016 ... DONE; 00,015,728,272 DEFRAGMENTED B-
trees have been rooted so far.
26,558      25,764,564 TotalNonUnique (of length 128) keys have been inserted into DEFRAGMENTED B-trees.
26,559 Leprechaun: Inserting keys/BBs of order 256 into B-trees, free RAM in B-tree pool is 00,035,404 MB; Pass #000,016 of 000,016 ... DONE; 00,016,776,848 DEFRAGMENTED B-
trees have been rooted so far.
26,560      26,526,835 TotalNonUnique (of length 256) keys have been inserted into DEFRAGMENTED B-trees.
26,561
26,562 Histogram (rotate it 90 degrees clockwise) of Unique-Building-Blocks appearing 2[+] times (e.g. 'alfalfa-alfa' stream has 1 BB with length 4 appearing 3 times thus
adding 1 to the count for {004}):
26,563 26,526,835{256}
26,564 25,764,564{128}
26,565 25,031,430{064}
26,566 24,776,643{056}
26,567 24,408,433{048}
26,568 23,699,242{036}
26,569 23,219,460{028}
26,570 23,082,831{024}
26,571 23,567,825{018}
26,572 23,807,316{016}
26,573 24,025,437{014}
26,574 23,785,884{012}
26,575 23,049,939{010}
26,576 21,209,568{008}
26,577 20,181,897{006}
26,578 8,308,154{004}
26,579
26,580 Leprechaun: Total Searches-n-Inserts Per Second: 2,578,806 SNIPS = 107,907,588,897 keys in 41,844 seconds
26,581 Leprechaun: RAM needed to house B-trees (relative to the file being ripped): 45N = 22,717MB
26,582 Leprechaun: RAM needed to build B-trees IN ONE PASS: (Target-Buffer = 820 MB) x 16 passes = 13,120MB
26,583 Note: In case of RAM in spades then may use compile option: -DSpeedUpBuilding=13120
26,584
26,585 Compressing 524,794,368 bytes ...
26,586 \; Each rotation means 64KB are encoded; Speed: 0,084,426 B/s; Done 100%; Compression Ratio: 5.16:1; Matches(16/24/48): 823,501/615,166/355,636; 128[+] long matches:
860,790; ETA: 0.00 hours
26,587 NumberOfFullLiterals (lower-the-better): 3599765
26,588 RAM-to-RAM performance: 84426 B/s.
26,589 Compressed to 101,783,592 bytes.
26,590 Source-file-Hash(FNV1A_YoshimitsuTRIAD) = 0xa5a8,8e5f
26,591 Target-file-Hash(FNV1A_YoshimitsuTRIAD) = 0x9c87,2875
26,592
26,593 Kernel Time = 10.187 = 0%
26,594 User Time = 48056.531 = 99%
26,595 Process Time = 48066.718 = 100% Virtual Memory = 59575 MB
26,596 Global Time = 48066.429 = 100% Physical Memory = 59460 MB
26,597
26,598 C:\Double-Deuce_8>sha1sum "SUPRAPIG_The_Prague_Stringology_Club_(302-PostScript-files).tar.NKMCH"
26,599 9408e09e6e6e85cdf05595400355f1230d4fedfd SUPRAPIG_The_Prague_Stringology_Club_(302-PostScript-files).tar.NKMCH

```

```

26,600
26,601 C:\Double-Deuce_8>timer64 Nakamichi_DD-192.exe "SUPRAPHIG_The_Prague_Stringology_Club_(302-PostScript-files).tar" "SUPRAPHIG_The_Prague_Stringology_Club_(302-PostScript-
files).tar.Nakamichi" 20 58000 i
26,602 ...
26,603 Current priority class is HIGH_PRIORITY_CLASS.
26,604 This compile uses B-trees only, no memmem() invocations - it compresses worse but much faster.
26,605 Allocating Source-Buffer 500 MB ...
26,606 Allocating Target-Buffer 820 MB ...
26,607 Source-file-Hash(FNV1A_YoshimitsuTRIAD) = 0xa5a8,8e5f
26,608 Leprechaun: Memory pool for B-trees is 58,000 MB.
26,609 Leprechaun: In this revision 8MB 16-way hash is used which results in 16 x 1,048,576 internal B-Trees of order 3.
26,610 Leprechaun: Allocating HASH memory 142,606,401 bytes = 8*(16+1)*2^(20) ... OK
26,611 Leprechaun: Allocating memory for B-trees 58001 MB ... OK
26,612 Leprechaun: Size of input file: 524,794,368
26,613
26,614 Leprechaun: Using (first 24 bytes of) DoubleDeuce (a.k.a. 3x2x4B=24B: 2xCastagnoli, 2xKoopman, 2xKoopman2) for Matches 24+ long, in order to reduce memory footprint.
26,615 Leprechaun: Inserting keys/BBs of order 004 into B-trees, free RAM in B-tree pool is 00,058,000 MB; Pass #000,001 of 000,001 ...
26,616 Leprechaun: Failure! Increment 'Memory for B-trees'!
26,617 Automatically increasing number of passes in order to fit B-trees into Target Buffer.
26,618 Leprechaun: Inserting keys/BBs of order 004 into B-trees, free RAM in B-tree pool is 00,057,844 MB; Pass #000,002 of 000,002 ... DONE; 00,001,048,208 DEFRAGMENTED B-
trees have been rooted so far.
26,619      8,308,154 TotalNonUnique (of length 004) keys have been inserted into DEFRAGMENTED B-trees.
26,620 Leprechaun: Inserting keys/BBs of order 006 into B-trees, free RAM in B-tree pool is 00,057,689 MB; Pass #000,001 of 000,002 ...
26,621 Leprechaun: Failure! Increment 'Memory for B-trees'!
26,622 Automatically increasing number of passes in order to fit B-trees into Target Buffer.
26,623 Leprechaun: Inserting keys/BBs of order 006 into B-trees, free RAM in B-tree pool is 00,057,082 MB; Pass #000,004 of 000,004 ... DONE; 00,002,096,784 DEFRAGMENTED B-
trees have been rooted so far.
26,624      20,181,897 TotalNonUnique (of length 006) keys have been inserted into DEFRAGMENTED B-trees.
26,625 Leprechaun: Inserting keys/BBs of order 008 into B-trees, free RAM in B-tree pool is 00,056,879 MB; Pass #000,001 of 000,004 ...
26,626 Leprechaun: Failure! Increment 'Memory for B-trees'!
26,627 Automatically increasing number of passes in order to fit B-trees into Target Buffer.
26,628 Leprechaun: Inserting keys/BBs of order 008 into B-trees, free RAM in B-tree pool is 00,056,052 MB; Pass #000,008 of 000,008 ... DONE; 00,003,145,360 DEFRAGMENTED B-
trees have been rooted so far.
26,629      21,209,568 TotalNonUnique (of length 008) keys have been inserted into DEFRAGMENTED B-trees.
26,630 Leprechaun: Inserting keys/BBs of order 010 into B-trees, free RAM in B-tree pool is 00,054,965 MB; Pass #000,008 of 000,008 ... DONE; 00,004,193,936 DEFRAGMENTED B-
trees have been rooted so far.
26,631      23,049,939 TotalNonUnique (of length 010) keys have been inserted into DEFRAGMENTED B-trees.
26,632 Leprechaun: Inserting keys/BBs of order 012 into B-trees, free RAM in B-tree pool is 00,053,758 MB; Pass #000,008 of 000,008 ... DONE; 00,005,242,512 DEFRAGMENTED B-
trees have been rooted so far.
26,633      23,785,884 TotalNonUnique (of length 012) keys have been inserted into DEFRAGMENTED B-trees.
26,634 Leprechaun: Inserting keys/BBs of order 014 into B-trees, free RAM in B-tree pool is 00,052,456 MB; Pass #000,008 of 000,008 ... DONE; 00,006,291,088 DEFRAGMENTED B-
trees have been rooted so far.
26,635      24,025,437 TotalNonUnique (of length 014) keys have been inserted into DEFRAGMENTED B-trees.
26,636 Leprechaun: Inserting keys/BBs of order 016 into B-trees, free RAM in B-tree pool is 00,052,292 MB; Pass #000,001 of 000,008 ...
26,637 Leprechaun: Failure! Increment 'Memory for B-trees'!
26,638 Automatically increasing number of passes in order to fit B-trees into Target Buffer.
26,639 Leprechaun: Inserting keys/BBs of order 016 into B-trees, free RAM in B-tree pool is 00,050,998 MB; Pass #000,016 of 000,016 ... DONE; 00,007,339,664 DEFRAGMENTED B-
trees have been rooted so far.
26,640      23,807,316 TotalNonUnique (of length 016) keys have been inserted into DEFRAGMENTED B-trees.
26,641 Leprechaun: Inserting keys/BBs of order 018 into B-trees, free RAM in B-tree pool is 00,049,558 MB; Pass #000,016 of 000,016 ... DONE; 00,008,388,240 DEFRAGMENTED B-
trees have been rooted so far.
26,642      23,567,825 TotalNonUnique (of length 018) keys have been inserted into DEFRAGMENTED B-trees.
26,643 Leprechaun: Inserting keys/BBs of order 036 into B-trees, free RAM in B-tree pool is 00,047,867 MB; Pass #000,016 of 000,016 ... DONE; 00,009,436,816 DEFRAGMENTED B-
trees have been rooted so far.
26,644      23,699,242 TotalNonUnique (of length 036) keys have been inserted into DEFRAGMENTED B-trees.
26,645 Leprechaun: Inserting keys/BBs of order 064 into B-trees, free RAM in B-tree pool is 00,046,064 MB; Pass #000,016 of 000,016 ... DONE; 00,010,485,392 DEFRAGMENTED B-
trees have been rooted so far.
26,646      25,031,430 TotalNonUnique (of length 064) keys have been inserted into DEFRAGMENTED B-trees.

```

```
26,647 Leprechaun: Inserting keys/BBs of order 024 into B-trees, free RAM in B-tree pool is 00,044,413 MB; Pass #000,016 of 000,016 ... DONE; 00,011,533,968 DEFRAGMENTED B-
trees have been rooted so far.
26,648          23,082,831 TotalNonUnique (of length 024) keys have been inserted into DEFRAGMENTED B-trees.
26,649 Leprechaun: Inserting keys/BBs of order 028 into B-trees, free RAM in B-tree pool is 00,042,746 MB; Pass #000,016 of 000,016 ... DONE; 00,012,582,544 DEFRAGMENTED B-
trees have been rooted so far.
26,650          23,219,460 TotalNonUnique (of length 028) keys have been inserted into DEFRAGMENTED B-trees.
26,651 Leprechaun: Inserting keys/BBs of order 048 into B-trees, free RAM in B-tree pool is 00,040,991 MB; Pass #000,016 of 000,016 ... DONE; 00,013,631,120 DEFRAGMENTED B-
trees have been rooted so far.
26,652          24,408,433 TotalNonUnique (of length 048) keys have been inserted into DEFRAGMENTED B-trees.
26,653 Leprechaun: Inserting keys/BBs of order 056 into B-trees, free RAM in B-tree pool is 00,039,204 MB; Pass #000,016 of 000,016 ... DONE; 00,014,679,696 DEFRAGMENTED B-
trees have been rooted so far.
26,654          24,776,643 TotalNonUnique (of length 056) keys have been inserted into DEFRAGMENTED B-trees.
26,655 Leprechaun: Inserting keys/BBs of order 128 into B-trees, free RAM in B-tree pool is 00,037,337 MB; Pass #000,016 of 000,016 ... DONE; 00,015,728,272 DEFRAGMENTED B-
trees have been rooted so far.
26,656          25,764,564 TotalNonUnique (of length 128) keys have been inserted into DEFRAGMENTED B-trees.
26,657 Leprechaun: Inserting keys/BBs of order 256 into B-trees, free RAM in B-tree pool is 00,035,404 MB; Pass #000,016 of 000,016 ... DONE; 00,016,776,848 DEFRAGMENTED B-
trees have been rooted so far.
26,658          26,526,835 TotalNonUnique (of length 256) keys have been inserted into DEFRAGMENTED B-trees.
26,659
26,660 Histogram (rotate it 90 degrees clockwise) of Unique-Building-Blocks appearing 2[+] times (e.g. 'alfalfa-alfa' stream has 1 BB with length 4 appearing 3 times thus
adding 1 to the count for {004}):
26,661 26,526,835{256]
26,662 25,764,564{128]
26,663 25,031,430{064]
26,664 24,776,643{056]
26,665 24,408,433{048]
26,666 23,699,242{036]
26,667 23,219,460{028]
26,668 23,082,831{024]
26,669 23,567,825{018]
26,670 23,807,316{016]
26,671 24,025,437{014]
26,672 23,785,884{012]
26,673 23,049,939{010]
26,674 21,209,568{008]
26,675 20,181,897{006]
26,676 8,308,154{004]
26,677
26,678 Leprechaun: Total Searches-n-Inserts Per Second: 4,065,234 SNIPS = 107,907,588,897 keys in 26,544 seconds
26,679 Leprechaun: RAM needed to house B-trees (relative to the file being ripped): 45N = 22,717MB
26,680 Leprechaun: RAM needed to build B-trees IN ONE PASS: (Target-Buffer = 820 MB) x 16 passes = 13,120MB
26,681 Note: In case of RAM in spades then may use compile option: -DSpeedUpBuilding=13120
26,682
26,683 Compressing 524,794,368 bytes ...
26,684 \; Each rotation means 64KB are encoded; Speed: 0,102,840 B/s; Done 100%; Compression Ratio: 5.16:1; Matches(16/24/48): 823,501/615,166/355,636; 128[+] long matches:
860,790; ETA: 0.00 hours
26,685 NumberOfFullLiterals (lower-the-better): 3599765
26,686 RAM-to-RAM performance: 102840 B/s.
26,687 Compressed to 101,783,592 bytes.
26,688 Source-file-Hash(FNV1A_YoshimitsuTRIAD) = 0xa5a8,8e5f
26,689 Target-file-Hash(FNV1A_YoshimitsuTRIAD) = 0x9c87,2875
26,690
26,691 Kernel Time = 16.500 = 0%
26,692 User Time = 31633.468 = 99%
26,693 Process Time = 31649.968 = 99% Virtual Memory = 59575 MB
26,694 Global Time = 31650.988 = 100% Physical Memory = 59461 MB
26,695
26,696 C:\Double-Deuce_8>shalsum "SUPRAPIG_The_Prague_Stringology_Club_(302-PostScript-files).tar.NKMCH"
```

```
26,697 9408e09e6e6e85cdf05595400355f1230d4fedfd SUPRAPIG_The_Prague_Stringology_Club_(302-PostScript-files).tar.NKMCH
26,698
26,699 C:\Double-Deuce_8>Nakamichi_DD-192.exe "SUPRAPIG_The_Prague_Stringology_Club_(302-PostScript-files).tar.Nakamichi" /bench
26,700 ...
26,701 Decompressing 101,783,592 bytes (being the compressed stream) ...
26,702 Warming up ...
26,703 RAM-to-RAM performance:
26,704 2499 MB/s; 2502 MB/s; 2502 MB/s; 2502 MB/s; 2496 MB/s; 2496 MB/s; 2496 MB/s; 2496 MB/s; 2496 MB/s; 2505 MB/s; 2502 MB/s; 2505 MB/s; 2505 MB/s; 2502 MB/s; 2502 MB/s;
2502 MB/s
26,705 Enforcing 17 seconds idling to avoid throttling ...
26,706 2496 MB/s; 2496 MB/s; 2496 MB/s; 2496 MB/s; 2496 MB/s; 2496 MB/s; 2493 MB/s; 2496 MB/s; 2496 MB/s; 2496 MB/s; 2496 MB/s; 2496 MB/s; 2496 MB/s; 2496 MB/s; 2496 MB/s;
2496 MB/s
26,707 Enforcing 17 seconds idling to avoid throttling ...
26,708 2496 MB/s; 2496 MB/s; 2496 MB/s; 2496 MB/s; 2496 MB/s; 2496 MB/s; 2496 MB/s; 2493 MB/s; 2496 MB/s; 2496 MB/s; 2496 MB/s; 2496 MB/s; 2496 MB/s; 2496 MB/s; 2502 MB/s; 2502 MB/s;
2502 MB/s
26,709 Enforcing 17 seconds idling to avoid throttling ...
26,710 This CPU seems to be working at 2,894 MHz, or more due to ensleeping.
26,711 RAM-to-RAM (peak) performance: 2505 MB/s.
26,712 Source-file-Hash(FNV1A_YoshimitsuTRIAD) = 0x9c87,2875
26,713 Target-file-Hash(FNV1A_YoshimitsuTRIAD) = 0x2ca9,d1a7
26,714
26,715 01/09/2021 18:04 101,783,592 SUPRAPIG_The_Prague_Stringology_Club_(302-PostScript-files).tar.Nakamichi
26,716 01/09/2021 18:04 524,794,368 SUPRAPIG_The_Prague_Stringology_Club_(302-PostScript-files).tar.NKMCH
26,717 12/26/2020 19:23 524,794,368 SUPRAPIG_The_Prague_Stringology_Club_(302-PostScript-files).tar
26,718
26,719 E:\_KAZE_Smxt_Benchmarks\Nakamichi_Double-Deuce_C_source_binaries>fc "SUPRAPIG_The_Prague_Stringology_Club_(302-PostScript-files).tar"
"SUPRAPIG_The_Prague_Stringology_Club_(302-PostScript-files).tar.NKMCH" /b
26,720 Comparing files SUPRAPIG_The_Prague_Stringology_Club_(302-PostScript-files).tar and SUPRAPIG_THE_PRAGUE_STRINGOLOGY_CLUB_(302-POSTSCRIPT-FILES).TAR.NKMCH
26,721 0026C021: 46 30
26,722 0026C022: 46 39
26,723 0026C023: 46 30
26,724 0026C024: 46 32
26,725 0026C025: 46 36
26,726 0026C028: 46 45
26,727 0026C029: 46 30
26,728 0026C02A: 46 30
26,729 0026C02B: 46 31
26,730 0026C02C: 46 45
26,731 0026C02D: 46 42
26,732 0026C02F: 46 38
26,733 0026C030: 46 33
26,734 0026C032: 46 34
26,735 0026C033: 46 38
26,736 0026C034: 46 34
26,737 0026C035: 46 39
26,738 0026C036: 46 36
26,739 0026C037: 46 43
26,740 0026C038: 46 31
26,741 0026C039: 46 33
26,742 0026C03A: 46 45
26,743 0026C03B: 46 30
26,744 0026C03C: 46 34
26,745 0026C03D: 46 38
26,746 0026C03E: 46 34
26,747 0026C03F: 46 39
26,748 0026C040: 46 39
26,749 0026C041: 46 30
26,750 0026C042: 46 33
```

Listing: Nakamichi_Ryuugan-ditto-1TB_btree.c; Last version: 2021-Aug-30; Font: MxPlus ToshibaTxL2 8x16.ttf; Downloadable at: www.sanmayce.com/Nakamichi/Kaidanji.zip

Page 462 of 466

Listing: Nakamichi Ryuugan-ditto-1TB btree.c; Last version: 2021-Aug-30; Font: MxPlus ToshibaTxL2 8x16.ttf; Downloadable at: www.sanmayce.com/Nakamichi/Kaidanji.zip

```

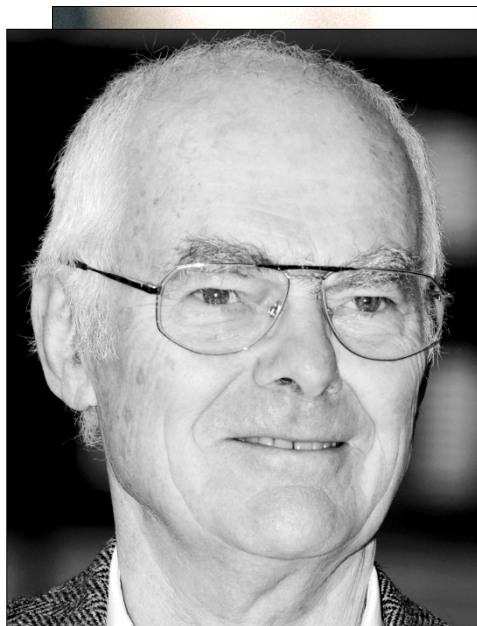
26,925 | 155,243,093 | 54,632,574 | 3,526,925,660 | branches:u |
26,926 | 4,887,072 | 6,490,361 | 127,177,075 | branch-misses:u |
26,927 | 380,228,246 | 35,730,747 | 4,344,466,576 | L1-dcache-loads:u |
26,928 | 19,137,410 | 7,682,673 | 187,276,863 | L1-dcache-load-misses:u |
26,929 | 1,628,108 | 3,365,779 | 79,885,744 | LLC-loads:u |
26,930 | 286,959 | 829,414 | 9,377,915 | LLC-load-misses:u |
26,931 +-----+
26,932 */
26,933
26,934 // Last change: 2021-Aug-28, Gumbotron replaced with the fastest Gumbotron_YMM.
26,935 // Last change: 2021-Aug-04, Added '-D_NaquaHash' for speed up and digging into 16bytes key2hash fun.
26,936 // Last change: 2021-Jul-05, Added '-DKanshiketsu' (not fully deployed) as a booster for building trees (by watching wether the previous order has presence for each position in a vector of size N)...
26,937 // Last change: 2021-Jun-23, In Kaidanji section, changed 6 to 'MatchLensNUM-1', also raising the 5GB to 12GB for maximum size of file decompression malloc().
26,938 // Last change: 2021-Mar-12, once again, added 22:, 30: and 24: codepaths, to boost compression a little with atavistic memmem(), also added (displacing 6:4 within 16MB window) 512:4=128 within 16MB window, also added fflush( stdout ); - to make the console output Windows-like.
26,939 // Last change: 2021-Jan-13, NastyBugFixed, for a long time this nasty bug remained uncrushed, namely not nullifying the 8bytes for 'CounterOccurencies' now serving also as LastSeenOffset, simply forgot to zeroing it :(
26,940 // Last change: 2021-Jan-12; A nasty bug exists, still uncrushed, namely when the key is not found in B-trees the corresponding 'LastSeenOffset_PseudoPointer[??]' should be 0/ZERO, but for some reason they are RANDOM?!... see/search for '2021-Jan-12 [' sting.
26,941 // Last change: 2021-Jan-07; Added 2 more CRC32 rounds to the clusterhash/DoubleDeuce, because SUPRAPIG_The_Prague_Stringology_Club_(302-PostScript-files).tar was problematic - decompressed file was not as original?!
26,942 // Last change: 2021-Jan-02; Changed (due to collisions with 'SUPRAPIG_glibc_watcom_grep.tar') CRC32C+CRC32K+CRC32K2 i.e. 12B to 16B with OnWard+HalfWard+InWard+OffWard traversing.
26,943 // Last change: 2020-Dec-31; Changed CRC64+CRC32 with CRC32C+CRC32K+CRC32K2, also the ordering of the polynomial
26,944 // Last change: 2020-Dec-22; Added ala Kaidanji 256+ matches (without B-tree-ing them)
26,945 // Last change: 2020-Dec-18; Minor changes in Decompress function: The outside 'if' sections were replaced by 'switch'.
26,946 // Last change: 2020-Dec-17; Minor changes in Decompress function: 1] Moved "MatchLen = (DWORDtrio&0x0C);" where it is used 2] Replaced all XMM copies with YMM ones - to avoid mix of XMM and YMM, experimental.
26,947 // Last change: 2020-Dec-09; The Affinity (locking to Core 1) was commented since it crashes-n-restarts new (2020-H2 Windows 10)... when decompressing in /bench mode in real-time priority only?!
26,948 // Last change: 2020-Nov-20; Added another RISKY alternative to SHA3 - CRC64 + CRC32C, instead of -D_NSHA3 add to command/compile line -D_NDD
26,949 // Last change: 2020-Nov-12; Added alternative to SHA3 - PRVhash, instead of -D_NSHA3 add to command/compile line -D_NPRV
26,950 // Last change: 2020-Jul-27; Back to supersimplisticism - 'Kaidanji' returns refined. Simply, ripping only orders 4..16 and then quickly trying to expand the match in the found position.
26,951 // Last change: 2020-Jun-28; Added "levels", simply put, search for "2020-Jun-28" to see the changes. Just reduced the matchlens for searching.
26,952 // Last change: 2020-Jun-21; The console messages (during start) were updated in order to autogenerate README.TXT like stuff (via Nakamichi.exe>README.TXT).
26,953 // Last change: 2020-Jun-17; Fixing the reporting stats bug (in 'B_tree_Non_Unique_Only_DEFRAGMENTED' function) by refreshing the master/real BUGGYoffset.
26,954 // Last change: 2020-Jun-14; Added 256 as matchlen by using SHA3, since the "MAX" is 255 (1..255 - one byte encodes the matchlen within the LEAF).
26,955 // Last change: 2020-Jun-09; Using 2N instead of 3N. Writing to .LOGs is engulfed with -DLITE i.e. if LITE preprocessing is present then no .LOGs, also no decompression (thus no verification and benchmark) after compression thus GETTING RID OF the third N which is 'SourceBlockREVERSED' block/malloc. Also, more stats - printing the number of REPETITIVE keys (per BB order) inserted into B-trees.
26,956 // Last change: 2020-Jun-08; To avoid anomalies (preventively) printing the loaded into RAM checksum of file being compressed, also dumping the decompressed content in one go (if less than 3800MB), otherwise in 1MB chunks.
26,957 // Last change: 2020-May-09; Added info report how many RAM is needed to build trees in one pass.
26,958 // Last change: 2020-Apr-22; Added LastSeen...[] for 128:, 56:.
26,959 // Last change: 2020-Apr-12; Grmb! Ugh! The stupid (done in a hurry) change from 2020-Feb-26 had a bug - the last 'i' has to be 'i64'.
26,960 // Last change: 2020-Feb-26; Grmb! It crashes with 2.7 and 3.3 GB DNA files, don't know the exact reason - the bug maybe is the writing in variables areas due to using 'int i' for building REVERSED memory block - maybe it becomes after 2GB negative...
26,961 // Last change: 2020-Feb-19; Added LastSeen...[] for 12:1, 8:1, 4:1.
26,962 // Last change: 2020-Feb-14; The same as 2020-Feb-02 but added -DLITE compile switch - it forces all memmem() i.e. Railgun invocations to immediately retutn NULL i.e. NOT FOUND - thus this 'LITE' compile uses only B-trees.
26,963 // Last change: 2020-Feb-02; 1] Do not write to file if in bench mode. 2] Enabled SHA3 at compile time, optional, that is. 3] Commented the UPDATING at 'EncStart' - it used to confuse the parser, see the '/// debugg' tag
26,964 // Last change: 2020-Jan-29; Added 48 as LastSeen. Also, fixed the compression rate slowdown blunder with 'Skip=NULL;' init - unnecessary memmem() for 128 needles.
26,965 // Last change: 2020-Jan-25; Changed the updating of LastSeenOffsets (not going past Sliding Window, but up to LookAheadBuffer thus fallback to memmem() is avoided).
26,966 // Last change: 2020-Jan-13; In order to enter SCB, decompression happens on 'stdout'.
26,967 // Last change: 2020-Jan-11; Fixed 5 overlooked '(signed int)' casts in Railgun, should be 4+GB long i.e. '(signed long long)'

```



```
26,968 // Last change: 2020-Jan-10; Fixed an overlooked haystack size of uint32_t in Railgun, should be 4+GB long, so, char * Railgun_Trollldom(char * pbTarget, char *
pbPattern, uint64_t cbTarget, uint32_t cbPattern);.
26,969 // Last change: 2020-Jan-04; Fixed an overlooked heuristic for 18:(2+1) causing different ratios.
26,970 // Last change: 2020-Jan-01; // Some more BtreeHEURISTIC were added, investigate why 2019-Dec-26 gives better ration with 'Sherlock'...?! Surely the answer lies in the
three 24: sections and four 28: sections.
26,971 // Last change: 2019-Dec-26;
26,972 // Last change: 2019-Dec-19;
26,973 // Last change: 2019-Dec-07;
26,974 // Last change: 2019-Dec-04;
26,975 // Last change: 2019-Sep-28;
26,976 // Last change: 2019-Jul-01;
26,977 // Last change: 2019-Apr-24;
26,978 // Last change: 2019-Feb-13;
26,979 // Last change: 2018-Nov-12;
26,980 // Last change: 2017-Oct-22;
26,981 // Last change: 2016-Aug-22; fixed Railgun_Swampshine, after that changed with Trollldom.
26,982 // Last change: 2016-Apr-08; small fix in encoding (avoiding possible negative offset if Sliding Window is biggy) from 2015-Mar-03.
26,983 // If you want to help me to improve it, email me at: sanmayce@sanmayce.com
26,984 // Enfun!
```

Nakamichi-Dragoneye-Kaidanji=The_Zennish_Microdeduplicator



Leprechaun returns ... the teraripper (from Greek *teras* 'monster', thus, **monstrous ripper**) is back to shamrock' your searches — Prof. Rudolf Bayer's B-tree order 3 implemented by machinely yours Sanmayce is to replace Railgun 'Trolldom' i.e. memmem() approach in Nakamichi parser.

All matches — 4,6,8 bytes long, for a start — are to be findable without any hashing, just in a few RAM fetches. Simply, all the matches (put as keys in the leaves) in the B-tree are to be paired with their last seen offset, hence the immediateness.

¹ shamrock is a 3-leaf clover – a symbol of Ireland



Nakamichi 'Dragoneye-Kaidanji' highlights:

- SCALABLE! Gets faster when more Physical or/and External RAM is available;
- Compileable under Windows and Linux, most suitable to run on 3TB machines;
- It targets superfast decompression of huge/tera corpora of textual data, while being the fastest (compressionwise) variant;
- The Zennish LZSS Microdeduplicator, a texttoy (file-to-file [de]compressor) written in plain C, 100% FREE - licenseless it is;
- CHUNKABLE! Since there are no headers/metadata in .Nakamichi files, arbitrary chunks/blocks can be compressed-n-concatenated;
- The Leprechaunesque (Internal/External) B-trees order 3 (2 keys MAX) are brutally-optimized;
- DEFRAGMENTED B-trees are in use during compression stage, their leaves contain only BBs appearing 2[+] times;
- Ability to deduplicate (as little as) 64 bytes long chunks 1TB backwards;
- AUTO-RESETTING PASSES (when TargetBuffer is filled then automatically restart building by doubling the passes);
- Author: Georgi 'Sanmayce' Marinov, e-mail: sanmayce@sanmayce.com, Twitter: <https://twitter.com/Sanmayce>

Nakamichi