



```
_FNV1A_Hash_SHIFTless_XORless PROC NEAR
; Line 755
    mov     edx, DWORD PTR _str$[esp-4]
    mov     cl, BYTE PTR [edx]
    test    cl, cl
    mov     eax, -2128831035      ; 811c9dc5H
    je      SHORT $L1582
    npad    1
$L1580:
; Line 756
    movzx   ecx, cl
    xor     ecx, eax
    imul    ecx, 16777619        ; 01000193H
    inc     edx
    mov     eax, ecx
    mov     cl, BYTE PTR [edx]
    test    cl, cl
    jne     SHORT $L1580
$L1582:
; Line 760
    and     eax, 8191            ; 00001fffH
; Line 761
    ret     0
_FNV1A_Hash_SHIFTless_XORless ENDP
```

LEPRECHAUN

AN ENGLISH-WORDLIST RIPPER, REVISION 13++++

Free download at www.sanmayce.com — on Intel Merom-1M 2166 MHz it rips **wikipedia** at 2,860,880++ words per second.

```

0001 /*
0002 This is source of Leprechaun revision 13++++, copyleft Sanmayce, 2010-Aug-27.
0003
0004 Comment/Uncomment accordingly in order to compile:
0005 #define _WIN32_ENVIRONMENT_
0006 //define _POSIX_ENVIRONMENT_
0007
0008 Linux compile(uncomment #include <io.h> line, ignore warnings):
0009 gcc -D_FILE_OFFSET_BITS=64 -m32 -static -O3 -mtune=generic Leprechaun.c -o Leprechaun_r13++++.generic_32bits.exe
0010
0011 Windows compile(uncomment #include <io.h> line, ignore warnings):
0012 For Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 13.10.3077 for 80x86 use:
0013 cl /Ox /Wp64 /TcLeprechaun.c /FaLeprechaun
0014
0015 Windows compile(comment #include <io.h> line, ignore warnings):
0016 For Intel(R) C++ Compiler Professional for applications running on IA-32, Version 11.1 use:
0017 icl /Ox /Wp64 /TcLeprechaun.c /FaLeprechaun /w /QxHOST
0018
0019 Due to my ignorance(calderas in my C knowledge): 64bit code cannot be generated, for now.
0020 Any improvement is welcome.
0021 Enjoy!
0022
0023 Below is the gain in 13++ and 13+++
0024
0025 Words per second performance: 5,974,513w/s
0026 Word count: 4,582,451,898 of them 9,177,221 distinct
0027 Number Of Trees(GREATER THE BETTER): 2855919
0028 Number Of Hash Collisions(Distinct WORDS - Number Of Trees): 6321302
0029
0030 Words per second performance: 6,329,353w/s
0031 Word count: 4,582,451,898 of them 9,177,221 distinct
0032 Number Of Trees(GREATER THE BETTER): 2958681
0033 Number Of Hash Collisions(Distinct WORDS - Number Of Trees): 6218540
0034
0035 The aftermath: 6,321,302 - 6,218,540 = 102,762 less collisions while the speed of hash is not slower for sure - I call this: double trouble avoidance.
0036 Thanks to Fowler/Noll/Vo hash inventors.
0037 */
0038
0039 // To do: must learn how to align, at last.
0040
0041 /*
0042 Enter-the-BESTer or an alchemical clash of pairs of primes.
0043
0044 When an x-bit hash where x < 16 and is not a power of 2 is needed,
0045 here comes 'FNV1A_Hash_4_OCTETS': a slightly tuned FNV1A hash for a huge(22,202,980) wordlist of latin-letters-words.
0046
0047 Two improvements for the generic(base) FNV1A hash:
0048 - first, better speed: by reducing 'imul' instructions when string is 4++ chars
0049 - second, better dispersion: by experimenting(superficially-lite test done, so far) with 'FNV1_32_PRIME'
0050
0051 Or more concretely:
0052 - For FNV1_32_INIT = 2166136261
0053 - Giving to 'FNV1_32_PRIME' all primes between 2 and 11987
0054 - Shifting by 16bits instead of 13bits, when 8192 slots are used
0055
0056 C code:
0057 typedef unsigned char u_int8_t;
0058 typedef unsigned long u_int32_t;
0059
0060 #define FNV1_32_INIT ((u_int32_t)2166136261)
0061 #define FNV1_32_PRIME ((u_int32_t)1607)
0062
0063 #define FNV_32A_OP(hash, octet) \
0064     (((u_int32_t)(hash) ^ (u_int8_t)(octet)) * FNV1_32_PRIME)
0065
0066 #define FNV_32A_OP32(hash, octet) \
0067     (((u_int32_t)(hash) ^ (u_int32_t)(octet)) * FNV1_32_PRIME)
0068
0069 0800 // Invoking: FNV1A_Hash_4_OCTETS(wrd, wrdlen>>) // = 0,1,2,3,4,5,6,7 [1..31]
0070 0801 int FNV1A_Hash_4_OCTETS(char *str, int wrdlen_QUADRUPLETS)
0071 0802 {
0072 0803 u_int32_t hash;
0073 0804 char *p;
0074 0805
0075 0806 hash = FNV1_32_INIT;
0076 0807 p=str;
0077 0808
0078 0809 // The goal of stage #1: to reduce number of 'imul's.
0079 0810
0080 0811 // Stage #1:
0081 0812 for (; wrdlen_QUADRUPLETS != 0; --wrdlen_QUADRUPLETS) {
0082 0813     hash = FNV_32A_OP32(hash, (unsigned long)*(long *)p); // mov edi, DWORD PTR [eax]
0083 0814     p=p+4; // add eax, 4
0084 0815 }
0085 0816
0086 0817 // Stage #2:
0087 0818 for (; *p; ++p) {

```

```

0088 0819     hash = FNV_32A_OP(hash, *p); // mov dl, BYTE PTR [eax]
0089 0820 }
0090 0821
0091 0822 //return ((hash>>13) ^ hash) & 8191; // (((u_int32_t)1<<(x))-1) where x=13
0092 0823     return ((hash>>16) ^ hash) & 8191; // 00..8191 i.e. 2^13=8192
0093 0824 }
0094
0095 Assembler code:
0096 _FNV1A_Hash_4_OCTETS PROC NEAR
0097 ; Line 812
0098 mov edx, DWORD PTR _wrdlen_QUADRUPLETS[esp-4]
0099 test edx, edx
0100 mov eax, DWORD PTR _str$[esp-4]
0101 push esi
0102 mov esi, DWORD PTR _FNV1_32_PRIME
0103 mov ecx, -2128831035
0104 je SHORT $L1612
0105 push edi
0106 npad7
0107 $L1610:
0108 ; Line 813
0109 mov edi, DWORD PTR [eax]
0110 xor edi, ecx
0111 imul edi, esi
0112 ; Line 814
0113 add eax, 4
0114 dec edx
0115 mov ecx, edi
0116 jne SHORT $L1610
0117 pop edi
0118 $L1612:
0119 ; Line 818
0120 mov dl, BYTE PTR [eax]
0121 test dl, dl
0122 je SHORT $L1619
0123 $L1617:
0124 ; Line 819
0125 movzx     edx, dl
0126 xor edx, ecx
0127 imul edx, esi
0128 inc eax
0129 mov ecx, edx
0130 mov dl, BYTE PTR [eax]
0131 test dl, dl
0132 jne SHORT $L1617
0133 $L1619:
0134 ; Line 823
0135 mov eax, ecx
0136 shr eax, 16
0137 xor eax, ecx
0138 and eax, 8191
0139 pop esi
0140 ; Line 824
0141 ret 0
0142 _FNV1A_Hash_4_OCTETS ENDP
0143
0144
0145 So, 'FNV1A_Hash_4_OCTETS' calculates faster and gives better distribution(3549448 for 1607), which is 0.6% better(less collisions), than generic 'FNV1A_Hash' with 3527916.
0146
0147 FNV proves to be great, dealing with 4x8bits(four octets) at once doesn't hurt distribution at all, I was amazed by consistency(stable behaviour) of 'FNV1A_Hash_4_OCTETS'.
0148
0149 I want to make a total clash of all possible pairs 'FNV1_32_INIT' & 'FNV1_32_PRIME' in order to lessen even a few thousand collisions.
0150 This is critical for speed performance e.g. when 30,974,750,142 words, the case of wikipedia-en.html.tar, must be hashed.
0151 The current obstacle is needed-time: each filling (26 slots x 31 sub-slots x 8192 sub-sub-slots) executes in 32-36 seconds for each pair.
0152 Such an easy task, but I can't see how to get done, it is not hard but slow even with 15 times faster testbed.
0153
0154 Between 1..1166136247 there are 58,834,113 primes (inclusive).
0155 Between 1..16777619 there are 1,077,891 primes (inclusive).
0156 or 58834113*1077891 = 63,416,760,895,683 pairs or 2,010,932 years needed at one-pair-per-second rate.
0157
0158 Finding THE best pair in my opinion is a total alchemy, due to the very nature of hashing: which is mainly alchemical and partly scientific.
0159 Since the magnum corpus of words is static-enough, THE pair is worthy to be found.
0160
0161 It doesn't take a think-tank to see the superiority of FNV, Fowler/Noll/Vo did reveal a thing of beauty.
0162
0163 Performance of 'FNV1A_Hash_4_OCTETS': 10236 words/clock or 105 MB/s|3,549,448 used slots (best)
0164
0165 CASE #1: with 'if (strlen(backup[j]) != 0)' before each execution
0166 Performance of 'kuxHash3plus' aka '2in1': 8076 words/clock or 82 MB/s|3,410,463 used slots (worst)
0167 Performance of 'FNV1A_Hash': 8079 words/clock or 83 MB/s|3,527,916 used slots
0168 Performance of 'FNV1A_Hash_SHIFTless_XORless': 8109 words/clock or 83 MB/s|3,540,323 used slots
0169
0170 CASE #2: without 'if (strlen(backup[j]) != 0)' before each execution
0171 Performance of 'kuxHash3plus' aka '2in1': 11673 words/clock or 119 MB/s|3,410,463 used slots (worst)
0172 Performance of 'FNV1A_Hash': 11558 words/clock or 118 MB/s|3,527,916 used slots
0173 Performance of 'FNV1A_Hash_SHIFTless_XORless': 11570 words/clock or 118 MB/s|3,540,323 used slots

```

```

0174
0175 Note:
0176 The 'strlen' overhead(CASE #1) is necessary due to priorly(before hash invocation) needed len-of-string for 'FNV1A_Hash_4_OCTETS'.
0177 Almost always, that is the case, since parsing of incoming text must know length of words/lines/files.
0178 In case of not knowing this length: ((119-105)/105)*100% = 13% degradation is unacceptable.
0179 The 'strlen' is an awful brake.
0180 Also whether the code overhead(one additional cycle) of 'FNV1A_Hash_4_OCTETS' is so successful(as a trade-off) or the testbed is deceiving I
do not know, here I am not so sure regardless of notorious delays caused by 'imul' and 'div' instructions.

0181 */
0182
0183 /*
0184 FNV1_32_PRIME: //?: 16777619
0185
0186 Above Binary-Search-Tree with MaxPEAK = 61 has NODES = 61 and LEAFs = 1
0187 words per second performance: 1,046,822w/s
0188 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
0189 Size of all TEXTual Files: 146,973,879
0190 word count: 12,561,874 of them 12,561,874 distinct
0191 Number Of Trees(GREATER THE BETTER): 2775839
0192
0193 Above Binary-Search-Tree with MaxPEAK = 39 has NODES = 72 and LEAFs = 15
0194 words per second performance: 1,356,588w/s
0195 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
0196 Size of all TEXTual Files: 415,982,896
0197 word count: 35,271,297 of them 22,202,980 distinct
0198 Number Of Trees(GREATER THE BETTER): 3539690
0199
0200 FNV1_32_PRIME: //3549448: 1607
0201
0202 Above Binary-Search-Tree with MaxPEAK = 61 has NODES = 61 and LEAFs = 1
0203 words per second performance: 1,046,822w/s
0204 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
0205 Size of all TEXTual Files: 146,973,879
0206 word count: 12,561,874 of them 12,561,874 distinct
0207 Number Of Trees(GREATER THE BETTER): 2783970
0208
0209 Above Binary-Search-Tree with MaxPEAK = 38 has NODES = 50 and LEAFs = 11
0210 words per second performance: 1,410,851w/s
0211 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
0212 Size of all TEXTual Files: 415,982,896
0213 word count: 35,271,297 of them 22,202,980 distinct
0214 Number Of Trees(GREATER THE BETTER): 3549395
0215
0216 FNV1_32_PRIME: //3550132: 175757909
0217
0218 Above Binary-Search-Tree with MaxPEAK = 60 has NODES = 60 and LEAFs = 1
0219 words per second performance: 966,298w/s
0220 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
0221 Size of all TEXTual Files: 146,973,879
0222 word count: 12,561,874 of them 12,561,874 distinct
0223 Number Of Trees(GREATER THE BETTER): 2784479
0224
0225 Above Binary-Search-Tree with MaxPEAK = 39 has NODES = 64 and LEAFs = 12
0226 words per second performance: 1,410,851w/s
0227 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
0228 Size of all TEXTual Files: 415,982,896
0229 word count: 35,271,297 of them 22,202,980 distinct
0230 Number Of Trees(GREATER THE BETTER): 3550115
0231
0232 FNV1_32_PRIME: //3550687: 201887489
0233
0234 Above Binary-Search-Tree with MaxPEAK = 60 has NODES = 60 and LEAFs = 1
0235 words per second performance: 966,298w/s
0236 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
0237 Size of all TEXTual Files: 146,973,879
0238 word count: 12,561,874 of them 12,561,874 distinct
0239 Number Of Trees(GREATER THE BETTER): 2784377
0240
0241 Above Binary-Search-Tree with MaxPEAK = 40 has NODES = 55 and LEAFs = 11
0242 words per second performance: 1,356,588w/s
0243 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
0244 Size of all TEXTual Files: 415,982,896
0245 word count: 35,271,297 of them 22,202,980 distinct
0246 Number Of Trees(GREATER THE BETTER): 3550528
0247
0248 FNV1_32_PRIME: //3550733: 172783361
0249
0250 Above Binary-Search-Tree with MaxPEAK = 59 has NODES = 59 and LEAFs = 1
0251 words per second performance: 1,046,822w/s
0252 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
0253 Size of all TEXTual Files: 146,973,879
0254 word count: 12,561,874 of them 12,561,874 distinct
0255 Number Of Trees(GREATER THE BETTER): 2786362
0256
0257 Above Binary-Search-Tree with MaxPEAK = 38 has NODES = 70 and LEAFs = 17
0258 words per second performance: 1,410,851w/s
0259 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
0260 Size of all TEXTual Files: 415,982,896

```

```

0261 Word count: 35,271,297 of them 22,202,980 distinct
0262 Number Of Trees(GREATER THE BETTER): 3550746
0263
0264 FNV1_32_PRIME: //3550929: 204312319
0265
0266 Above Binary-Search-Tree with MaxPEAK = 61 has NODES = 61 and LEAFs = 1
0267 words per second performance: 966,298w/s
0268 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
0269 Size of all TEXTual Files: 146,973,879
0270 word count: 12,561,874 of them 12,561,874 distinct
0271 Number Of Trees(GREATER THE BETTER): 2785581
0272
0273 Above Binary-Search-Tree with MaxPEAK = 37 has NODES = 55 and LEAFs = 12
0274 words per second performance: 1,356,588w/s
0275 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
0276 Size of all TEXTual Files: 415,982,896
0277 word count: 35,271,297 of them 22,202,980 distinct
0278 Number Of Trees(GREATER THE BETTER): 3550886
0279
0280 Leprechaun_Microsoft.exe: FNV1_32_PRIME: //3551736: 107712257
0281
0282 Above Binary-Search-Tree with MaxPEAK = 61 has NODES = 61 and LEAFs = 1
0283 words per second performance: 1,046,822w/s
0284 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
0285 Size of all TEXTual Files: 146,973,879
0286 word count: 12,561,874 of them 12,561,874 distinct
0287 Number Of Trees(GREATER THE BETTER): 2786515
0288
0289 Above Binary-Search-Tree with MaxPEAK = 36 has NODES = 64 and LEAFs = 15
0290 words per second performance: 1,356,588w/s
0291 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
0292 Size of all TEXTual Files: 415,982,896
0293 word count: 35,271,297 of them 22,202,980 distinct
0294 Number Of Trees(GREATER THE BETTER): 3551744
0295
0296 Leprechaun_Intel.exe: FNV1_32_PRIME: //3551736: 107712257
0297
0298 Above Binary-Search-Tree with MaxPEAK = 61 has NODES = 61 and LEAFs = 1
0299 words per second performance: 1,256,187w/s
0300 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_en-WORDS.lst
0301 Size of all TEXTual Files: 146,973,879
0302 word count: 12,561,874 of them 12,561,874 distinct
0303 Number Of Trees(GREATER THE BETTER): 2786515
0304
0305 Above Binary-Search-Tree with MaxPEAK = 36 has NODES = 64 and LEAFs = 15
0306 words per second performance: 1,603,240w/s
0307 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
0308 Size of all TEXTual Files: 415,982,896
0309 word count: 35,271,297 of them 22,202,980 distinct
0310 Number Of Trees(GREATER THE BETTER): 3551744
0311
0312 wow: 1,603,240w/s vs 1,356,588w/s respectively Leprechaun_Intel.exe vs Leprechaun_Microsoft.exe, i.e. 18% betterment, no joke!
0313 */
0314
0315 // windows: ~~~~~
0316 // _CRTIMP size_t __cdecl fread(void *, size_t, size_t, FILE *);
0317 // _CRTIMP size_t __cdecl fwrite(const void *, size_t, size_t, FILE *);
0318 // _CRTIMP int __cdecl fgetpos(FILE *, fpos_t *);
0319 // _CRTIMP int __cdecl fsetpos(FILE *, const fpos_t *);
0320
0321 // _CRTIMP _int64 __cdecl _seeki64(int, _int64, int);
0322 // _CRTIMP _int64 __cdecl _telli64(int);
0323 // _CRTIMP _int64 __cdecl _filelengthi64(int);
0324 // above 3 are in 'io.h'
0325
0326 // _CRTIMP int __cdecl fseek(FILE *, long, int);
0327 // _CRTIMP long __cdecl ftell(FILE *);
0328 // _CRTIMP int __cdecl fclose(FILE *);
0329
0330 // #ifndef _SIZE_T_DEFINED
0331 // #ifdef _WIN64
0332 // typedef unsigned _int64 size_t;
0333 // #else
0334 // typedef _w64 unsigned int size_t;
0335 // #endif
0336 // #define _SIZE_T_DEFINED
0337 // #endif
0338
0339 // typedef _int64 fpos_t;
0340
0341 // Linux: ~~~~~
0342 // size_t fread (void *data, size_t size, size_t count, FILE *stream)
0343 // size_t fwrite (const void *data, size_t size, size_t count, FILE *stream)
0344 // int fgetpos (FILE *stream, fpos_t *position)
0345 // int fsetpos (FILE *stream, const fpos_t *position)
0346
0347 // FILE * fopen64 (const char *filename, const char *opentype)
0348 // int fseeko64 (FILE *stream, off64_t offset, int whence)

```

```

0349 // off64_t ftello64 (FILE *stream)
0350 // int fclose (FILE *stream)
0351
0352 // off_t lseek (int filedes, off_t offset, int whence)
0353 // above 1 is in 'unistd.h'
0354
0355
0356 // ===== MUST work both for windows and Linux =====
0357 // #define _WIN32_ENVIRONMENT_
0358 // #define POSIX_ENVIRONMENT_
0359
0360
0361 #ifndef NULL
0362 #ifdef __cplusplus
0363 #define NULL 0
0364 #else
0365 #define NULL ((void*)0)
0366 #endif
0367 #endif
0368
0369 // To do #1: Put this 31 in MAXw1: 'int MAXw1 = 31;'
0370 // To do #2: No need of flushing unsorted words to file: make backup[] array
0371 //           instead of writing. And mostly sort 26 times!
0372 // HEAVY BUG in r.7: unsigned long h11(unsigned long n)
0373 //                 is NOT identical with
0374 //                 unsigned long GRMBLh11[32]; // 00 not used, only 01..31
0375 // BECAUSE DUMBEST DUMB Array GRMBLh11 expects 'int' not
0376 // 'unsigned long' !!!
0377
0378 #include <stdio.h>
0379 #include <ctype.h>
0380 #include <time.h>
0381 #if defined(_WIN32_ENVIRONMENT_)
0382 #include <io.h> // needed for windows' 'lseeki64' and 'telli64'
0383 // Above line must be commented in order to compile with Intel C compiler: an error "can't find io.h" occurs.
0384 #else
0385 #endif /* defined(_WIN32_ENVIRONMENT_) */
0386
0387 typedef unsigned char char_t;
0388 typedef char_t *string;
0389
0390 clock_t clocks1, clocks2;
0391 int Bozan;
0392
0393 typedef unsigned char u_int8_t; //FNV only
0394 typedef unsigned long u_int32_t; //FNV only
0395 typedef unsigned long long u_int64_t; //FNV only
0396
0397 // SINHA fragment[
0398
0399 #define swapKAZE(a, b) { t = *(a); *(a) = *(b); *(b) = t; }
0400
0401 static void InsertSortKAZE(string *a, int n, int d) //void inssort(unsigned char **a, int n, int d)
0402 { string *pi, *pj, s, t; //unsigned char **pi, **pj, *s, *t;
0403   for (pi = a + 1; --n > 0; pi++)
0404     for (pj = pi; pj > a; pj--) {
0405       /* Inline strcmp: break if *(pj-1) <= *pj */
0406       for (s = *(pj-1)+d, t = *pj+d; *s == *t && *s != 0; s++, t++)
0407         ;
0408       if (*s <= *t)
0409         break;
0410       swapKAZE(pj, pj-1);
0411     }
0412 }
0413
0414 //int cmpit(unsigned char **h1, unsigned char **h2)
0415 //{
0416 //  return( scmp(*h1, *h2) );
0417 //}
0418
0419 //int scmp( unsigned char *s1, unsigned char *s2 )
0420 //{
0421 //  while( *s1 != '\0' && *s1 == *s2 )
0422 //    {
0423 //      s1++;
0424 //      s2++;
0425 //    }
0426 //  return( *s1-*s2 );
0427 //}
0428
0429 //static void simplesort(string a[], int n, int b)
0430 //{
0431 //  int i, j;
0432 //  string tmp;
0433 //  for (i = 1; i < n; i++)
0434 //    for (j = i; j > 0 && scmp(a[j-1]+b, a[j]+b) > 0; j--)
0435 //      { tmp = a[j]; a[j] = a[j-1]; a[j-1] = tmp; }
0436 //}

```

```

0437 //}
0438
0439 // SINHA fragment]
0440
0441 // mkqsort.c BEGIN *****
0442 /*
0443  Multikey quicksort, a radix sort algorithm for arrays of character
0444  strings by Bentley and Sedgewick.
0445
0446  J. Bentley and R. Sedgewick. Fast algorithms for sorting and
0447  searching strings. In Proceedings of 8th Annual ACM-SIAM Symposium
0448  on Discrete Algorithms, 1997.
0449
0450  http://www.CS.Princeton.EDU/~rs/strings/index.html
0451
0452  The code presented in this file has been tested with care but is
0453  not guaranteed for any purpose. The writer does not offer any
0454  warranties nor does he accept any liabilities with respect to
0455  the code.
0456
0457  Ranjan Sinha, 1 jan 2003.
0458
0459  School of Computer Science and Information Technology,
0460  RMIT University, Melbourne, Australia
0461  rsinha@cs.rmit.edu.au
0462
0463 */
0464
0465 // #include "sortstring.h"
0466
0467 /* MULTIKEY QUICKSORT */
0468
0469 #ifndef min
0470 #define min(a, b) ((a) <= (b) ? (a) : (b))
0471 #endif
0472
0473
0474 // ----- BTREE [
0475 #define false -1
0476 #define true 0
0477
0478 struct nodeBTREE {
0479   int data;
0480   struct nodeBTREE* left;
0481   struct nodeBTREE* right;
0482 };
0483
0484 // ----- BTREE ]
0485
0486
0487 /* ssort2 -- Faster Version of Multikey Quicksort */
0488
0489 void vecswap2(unsigned char **a, unsigned char **b, int n)
0490 { while (n-- > 0) {
0491   unsigned char *t = *a;
0492   *a++ = *b;
0493   *b++ = t;
0494 } }
0495
0496
0497 #define swap2(a, b) { t = *(a); *(a) = *(b); *(b) = t; }
0498 #define ptr2char(i) (*(i) + depth)
0499
0500 unsigned char **med3func(unsigned char **a, unsigned char **b, unsigned char **c, int depth)
0501 { int va, vb, vc;
0502   if ((va=ptr2char(a)) == (vb=ptr2char(b)))
0503     return a;
0504   if ((vc=ptr2char(c)) == va || vc == vb)
0505     return c;
0506   return va < vb ?
0507     (vb < vc ? b : (va < vc ? c : a)) :
0508     (vb > vc ? b : (va < vc ? a : c));
0509 }
0510
0511 #define med3(a, b, c) med3func(a, b, c, depth)
0512
0513 void inssort(unsigned char **a, int n, int d)
0514 { unsigned char **pi, **pj, *s, *t;
0515   for (pi = a + 1; --n > 0; pi++)
0516     for (pj = pi; pj > a; pj--) {
0517       /* Inline strcmp: break if *(pj-1) <= *pj */
0518       for (s = *(pj-1)+d, t = *pj+d; *s == *t && *s != 0; s++, t++)
0519         ;
0520       if (*s <= *t)
0521         break;
0522       swap2(pj, pj-1);
0523     }
0524 }

```

```

0525 void mkqsort(unsigned char **a, int n, int depth)
0526 {   int d, r, partval;
0527     unsigned char **pa, **pb, **pc, **pd, **pl, **pm, **pn, **t;
0528     if (n < 20) {
0529         insort(a, n, depth);
0530         return;
0531     }
0532     pl = a;
0533     pm = a + (n/2);
0534     pn = a + (n-1);
0535     if (n > 30) { /* On big arrays, pseudomedian of 9 */
0536         d = (n/8);
0537         pl = med3(pl, pl+d, pl+2*d);
0538         pm = med3(pm-d, pm, pm+d);
0539         pn = med3(pn-2*d, pn-d, pn);
0540     }
0541     pm = med3(pl, pm, pn);
0542     swap2(a, pm);
0543     partval = ptr2char(a);
0544     pa = pb = a + 1;
0545     pc = pd = a + n-1;
0546     for (;;) {
0547         while (pb <= pc && (r = ptr2char(pb)-partval) <= 0) {
0548             if (r == 0) { swap2(pa, pb); pa++; }
0549             pb++;
0550         }
0551         while (pb <= pc && (r = ptr2char(pc)-partval) >= 0) {
0552             if (r == 0) { swap2(pc, pd); pd--; }
0553             pc--;
0554         }
0555         if (pb > pc) break;
0556         swap2(pb, pc);
0557         pb++;
0558         pc--;
0559     }
0560     pn = a + n;
0561     r = min(pa-a, pb-pa); vecswap2(a, pb-r, r);
0562     r = min(pd-pc, pn-pd-1); vecswap2(pb, pn-r, r);
0563     if ((r = pb-pa) > 1)
0564         mkqsort(a, r, depth);
0565     if (ptr2char(a + r) != 0)
0566         mkqsort(a + r, pa-a + pn-pd-1, depth+1);
0567     if ((r = pd-pc) > 1)
0568         mkqsort(a + n-r, r, depth);
0569 }
0570
0571 void mkqsort_main(unsigned char **a, int n) { mkqsort(a, n, 0); }
0572 // mkqsort.c END *****
0573
0574 // why Sinha uses int instead of long??!!
0575 static int readlines(char *file_name, string **lines)
0576 {
0577     int nlines = 0;
0578     size_t size;
0579     FILE *in_file;
0580     string basep, cur, next;
0581     string *ASbackup;
0582
0583     if (!(in_file = fopen(file_name, "rb"))) {
0584         printf("Leprechaun: Can't open file %s \n", file_name);
0585         exit(-1);
0586     }
0587     fseek(in_file, 0, SEEK_END);
0588     size = ftell(in_file);
0589     fseek(in_file, 0, SEEK_SET);
0590     if (!(basep = (string) malloc(size*sizeof(char_t)))) return -1;
0591     printf("Allocated memory for words in MB: %lu\n", ((size*sizeof(char_t))>>20)+1 );
0592     if (fread(basep, 1, size, in_file) < size) {
0593         printf("Leprechaun: Can't read file %s \n", file_name);
0594         exit(-1);
0595     }
0596     fclose(in_file);
0597
0598     // GET nlines:
0599     cur = basep;
0600     while (cur < basep + size) {
0601         next = cur;
0602         while ((next < basep + size) && (*next != '\n')) {next++;}
0603         *--next = '\0'; // This is ala DOS i.e. Windows
0604                        // 1310 not 10(\n=10)
0605         cur = next + 2;
0606         nlines++;
0607     }
0608
0609     // printf("%lu\n", (unsigned long)*lines); -> backup = *lines = 0
0610     ASbackup = (string *)malloc( nlines*sizeof(string) ); // sizeof(string) is 4
0611     if( ASbackup == NULL )
0612     { puts( "Leprechaun: Needed memory allocation denied!\n" ); return( 1 ); }

```

```

0613 printf( "Allocated memory for pointers-to-words in MB: %lu\n", ((nlines*sizeof(string))>>20)+1 );
0614 *lines = ASbackup;
0615 //printf("%lu\n", (unsigned long)*lines); -> backup = *lines = ASbackup = 6946888
0616
0617 // Upload nlines times:
0618     nlines = 0;
0619     cur = basep;
0620     while (cur < basep + size) {
0621         next = cur;
0622         while ((next < basep + size) && (*next != '\n')) {next++;}
0623         *--next = '\0'; // This is ala DOS i.e. Windows
0624                        // 1310 not 10(\n=10)
0625         ASbackup[nlines] = cur;
0626         cur = next + 2;
0627         nlines++;
0628     }
0629     return nlines;
0630 }
0631
0632 void x64toakAZE ( /* stdcall is faster and smaller... Might as well use it for the helper. */
0633     unsigned long long val,
0634     char *buf,
0635     unsigned radix,
0636     int is_neg
0637 )
0638 {
0639     char *p; /* pointer to traverse string */
0640     char *firstdig; /* pointer to first digit */
0641     char temp; /* temp char */
0642     unsigned digval; /* value of digit */
0643
0644     p = buf;
0645
0646     if ( is_neg )
0647     {
0648         *p++ = '-'; /* negative, so output '-' and negate */
0649         val = (unsigned long long)(- (long long)val);
0650     }
0651
0652     firstdig = p; /* save pointer to first digit */
0653
0654     do {
0655         digval = (unsigned) (val % radix);
0656         val /= radix; /* get next digit */
0657
0658         /* convert to ascii and store */
0659         if (digval > 9)
0660             *p++ = (char) (digval - 10 + 'a'); /* a letter */
0661         else
0662             *p++ = (char) (digval + '0'); /* a digit */
0663     } while (val > 0);
0664
0665     /* we now have the digit of the number in the buffer, but in reverse
0666     order. Thus we reverse them now. */
0667
0668     *p-- = '\0'; /* terminate string; p points to last digit */
0669
0670     do {
0671         temp = *p;
0672         *p = *firstdig;
0673         *firstdig = temp; /* swap *p and *firstdig */
0674         --p;
0675         ++firstdig; /* advance to next two digits */
0676     } while (firstdig < p); /* repeat until halfway */
0677 }
0678
0679 /* Actual functions just call conversion helper with neg flag set correctly,
0680 and return pointer to buffer. */
0681
0682 char * _i64toakAZE (
0683     long long val,
0684     char *buf,
0685     int radix
0686 )
0687 {
0688     x64toakAZE((unsigned long long)val, buf, radix, (radix == 10 && val < 0));
0689     return buf;
0690 }
0691
0692 char * _ui64toakAZE (
0693     unsigned long long val,
0694     char *buf,
0695     int radix
0696 )
0697 {
0698     x64toakAZE(val, buf, radix, 0);
0699     return buf;
0700 }

```

```

0701
0702 char * _ui64toaKAZEzerocomma (
0703     unsigned long long val,
0704     char *buf,
0705     int radix
0706 )
0707 {
0708     char *p;
0709     char temp;
0710     int txpman;
0711     int pxnman;
0712     x64toaKAZE(val, buf, radix, 0);
0713     p = buf;
0714     do {
0715     } while ((*++p != '\0'));
0716     p--; // p points to last digit
0717     // buf points to first digit
0718     buf[26] = 0;
0719     txpman = 1;
0720     pxnman = 0;
0721     do
0722     { if (buf <= p)
0723     { temp = *p;
0724       buf[26-txpman] = temp; pxnman++;
0725       p--;
0726       if (pxnman % 3 == 0)
0727       { txpman++;
0728         buf[26-txpman] = (char) ('.','.');
0729       }
0730     }
0731     else
0732     { buf[26-txpman] = (char) ('0'); pxnman++;
0733       if (pxnman % 3 == 0)
0734       { txpman++;
0735         buf[26-txpman] = (char) ('.','.');
0736       }
0737     }
0738     txpman++;
0739     } while (txpman <= 26);
0740     return buf;
0741 }
0742
0743 char * _ui64toaKAZEcomma (
0744     unsigned long long val,
0745     char *buf,
0746     int radix
0747 )
0748 {
0749     char *p;
0750     char temp;
0751     int txpman;
0752     int pxnman;
0753     x64toaKAZE(val, buf, radix, 0);
0754     p = buf;
0755     do {
0756     } while ((*++p != '\0'));
0757     p--; // p points to last digit
0758     // buf points to first digit
0759     buf[26] = 0;
0760     txpman = 1;
0761     pxnman = 0;
0762     while (buf <= p)
0763     { temp = *p;
0764       buf[26-txpman] = temp; pxnman++;
0765       p--;
0766       if (pxnman % 3 == 0 && buf <= p)
0767       { txpman++;
0768         buf[26-txpman] = (char) ('.','.');
0769       }
0770     }
0771     txpman++;
0772     return buf+26-(txpman-1);
0773 }
0774
0775 unsigned char KuxHash(char *str)
0776 { unsigned char h = 0;
0777   int max31 = 0;
0778   //while (*str)
0779   while (str[max31])
0780   { h = h ^ str[max31++];
0781     //h = h ^ *str++; // I am not sure 'str' is returned changed after return?!
0782   }
0783   return h; // 00..255 i.e. 2^8=256
0784 }
0785
0786 int KuxHash2(char *str)
0787 { int h = 0;
0788   unsigned long h2 = 0; // must be long: 31*'z'=31*122

```

```

0789   int max31 = 0;
0790   while (str[max31])
0791   { h = h ^ str[max31]; // 00..255 i.e. 2^8=256
0792     //h2 = h2 + str[max31++]; // [113s]
0793     h2 = h2 + max31 * str[max31++];
0794   }
0795   h=h<<4; // 00..15 i.e. 2^4=16
0796   //h = h|(C str[0] ^ str[max31-1] ); // [111s] a..z: each XOR each gives 00..31
0797   h = h|(C h2%(1<<4)-1) );
0798   return h; // 00..4095 i.e. 2^12=4096
0799 }
0800
0801 //OSHO test - Attempts to Find/Put a WORD into linked list count: 32,011,937
0802 int KuxHash3(char *str)
0803 { int h = 0;
0804   unsigned long h2 = 0; // must be long: 31*'z'=31*122
0805   int max31 = 0;
0806   while (str[max31])
0807   { h = h ^ str[max31]; // 00..255 i.e. 2^8=256
0808     //h2 = h2 + str[max31++]; // [113s]
0809     h2 = h2 + str[max31++] * (max31+1);
0810   }
0811   // Result is: 7bits in 'h' and 32bits in 'h2'.
0812
0813     //printf("%s:\n ",str);
0814     //printf("%d ",h);
0815   h=h<<6; // 00..15 i.e. 00-05+7bits=13bits
0816     //printf("%d ",h);
0817     //printf("%d ",h2);
0818   //h = h|(C str[0] ^ str[max31-1] ); // [111s] a..z: each XOR each gives 00..31
0819   h = h|(C h2%(1<<6)-1) ); // 64-1=63=9*7; 61 is prime
0820     //printf("%d \n",h);
0821   return h; // 00..8191 i.e. 2^13=8192
0822 }
0823
0824 //OSHO test - Attempts to Find/Put a WORD into a linked list count: 31,927,285
0825 int KuxHash3plus(char *str)
0826 { int h = 0;
0827   unsigned long h2 = 0; // must be long: 31*'z'=31*122
0828   int max31 = 0;
0829   while (str[max31])
0830   { h = h ^ str[max31]; // 00..255 i.e. 2^8=256
0831     //h2 = h2 + str[max31++]; // [113s]
0832     h2 = h2 + str[max31++] * (max31+1);
0833   }
0834   // Result is: 7bits in 'h' and 32bits in 'h2'.
0835
0836     //printf("%s:\n ",str);
0837     //printf("%d ",h);
0838   // a in ASCII is 097 = 0110 0001
0839   // z in ASCII is 122 = 0111 1010
0840   // Above two lines show that bits 8-7-6 are always 0-1-1 so need for low 5 bits.
0841   //h=h<<8; // 00..15 i.e. 5bits + 00-07bits=13bits
0842     //printf("%d ",h);
0843     //printf("%d ",h2);
0844   //h = h|(C str[0] ^ str[max31-1] ); // [111s] a..z: each XOR each gives 00..31
0845   h = (( h<<8 )|(C h2%(251) )&8191; // 251 prime
0846     //printf("%d \n",h);
0847   return h; // 00..8191 i.e. 2^13=8192
0848 }
0849
0850 /*
0851 PUBLIC      _KuxHash3plus
0852 ; Function compile flags: /Ogty
0853 _TEXT      SEGMENT
0854 _str$ = 8 ; size = 4
0855 _KuxHash3plus PROC NEAR
0856 ; Line 511
0857 mov ecx, DWORD PTR _str$[esp-4]
0858 mov dl, BYTE PTR [ecx]
0859 push esi
0860 xor esi, esi
0861 xor eax, eax
0862 test dl, dl
0863 je SHORT $L1561
0864 push ebx
0865 push edi
0866 mov edi, 1
0867 sub edi, ecx
0868 npad8
0869 $L1560:
0870 ; Line 512
0871 movsx     edx, BYTE PTR [ecx]
0872 ; Line 514
0873 lea ebx, DWORD PTR [edi+ecx]
0874 imul ebx, edx
0875 xor esi, edx
0876 mov dl, BYTE PTR [ecx+1]

```

```

0877 add eax, ebx
0878 inc ecx
0879 test dl, dl
0880 jne SHORT $L1560
0881 pop edi
0882 pop ebx
0883 $L1561:
0884 ; Line 527
0885 xor edx, edx
0886 mov ecx, 251 ; 000000fbh
0887 div ecx
0888 shl esi, 8
0889 mov eax, edx
0890 ; Line 529
0891 or eax, esi
0892 and eax, 8191 ; 00001ffffh
0893 pop esi
0894 ; Line 530
0895 ret 0
0896 _kuxHash3plus ENDP
0897 _TEXT ENDS
0898 */
0899
0900 //OSHO test - Attempts to Find/Put a WORD into a linked list count: 32,021,975
0901 int KuxHash4(char *str)
0902 {
0903     int h2 = 0;
0904     for (; *str != 0; str++) {
0905         //h2 = (127*h2 + *str) % (8192-1); // 2^13-1 = 8191 is Mersenne prime
0906         h2 = ((h2<<7) + *str) % (8192-1); // 2^13-1 = 8191 is Mersenne prime
0907     }
0908     return h2; // 00..8191 i.e. 2^13=8192
0909 }
0910 }
0911
0912 /*
0913 int hash(char *v, int M)
0914 { int h = 0, a = 127;
0915   for (; *v != 0; v++)
0916       h = (a*h + *v) % M;
0917   return h;
0918 }
0919
0920 int hashU(char *v, int M)
0921 { int h, a = 31415, b = 27183;
0922   for (h = 0; *v != 0; v++, a = a*b % (M-1))
0923       h = (a*h + *v) % M;
0924   return (h < 0) ? (h + M) : h;
0925 }
0926 */
0927
0928 // Kaze: My appreciation of FNV is far beyond C code optimization, it is alchemical, and why not, magical.
0929
0930 /*
0931 FNV hash history
0932 The basis of the FNV hash algorithm was taken from an idea sent as reviewer comments to the IEEE POSIX P1003.2 committee
0933 by Glenn Fowler and Phong Vo back in 1991. In a subsequent ballot round: Landon Curt Noll improved on their algorithm.
0934 Some people tried this hash and found that it worked rather well. In an EMail message to Landon, they named it
0935 the "Fowler/Noll/vo'" or FNV hash.
0936 FNV hashes are designed to be fast while maintaining a low collision rate. The FNV speed allows one to quickly hash
0937 lots of data while maintaining a reasonable collision rate. The high dispersion of the FNV hashes makes them well suited
0938 for hashing nearly identical strings such as URLs, hostnames, filenames, text, IP addresses, etc.
0939 */
0940
0941 /* NOTE: u_int64_t is a 64 bit unsigned type */
0942 /* NOTE: u_int32_t is a 32 bit unsigned type */
0943 /* NOTE: u_int16_t is a 16 bit unsigned type */
0944 /* NOTE: u_int8_t is a 8 bit unsigned type */
0945
0946 //typedef unsigned char u_int8_t; //FNV only
0947 //typedef unsigned long u_int32_t; //FNV only
0948 //typedef unsigned long long u_int64_t; //FNV only
0949
0950 // 32 bit FNV_prime = 2^24 + 2^8 + 0x93 = 16777619
0951 // 64 bit FNV_prime = 2^40 + 2^8 + 0xb3 = 1099511628211
0952
0953 // 32 bit offset_basis = 2166136261
0954 // 64 bit offset_basis = 14695981039346656037
0955
0956 #define FNV1_64_INIT ((u_int64_t)14695981039346656037)
0957 #define FNV1_64_PRIME ((u_int64_t)1099511628211)
0958 #define FNV1_32_INIT ((u_int32_t)2166136261)
0959 #define FNV1_32_PRIME ((u_int32_t)107712257)
0960 // FNV1A_Hash_4_OCTETS gives dispersion as follows:
0961 //3549448: 1607
0962 //3549669: 171072511
0963 //3549702: 124041217
0964 //3549787: 121802497

```

```

0965 //3549866: 175430911
0966 //3549939: 180818431
0967 //3549981: 118699519
0968 //3550132: 175757909
0969 //3550687: 201887489
0970 //3550733: 172783361
0971 //3550929: 204312319
0972 //3551736: 107712257
0973
0974 #define FNV_64A_OP(hash, octet) \
0975     (((u_int64_t)(hash) ^ (u_int8_t)(octet)) * FNV1_64_PRIME)
0976
0977 #define FNV_64A_OP64(hash, octet) \
0978     (((u_int64_t)(hash) ^ (u_int64_t)(octet)) * FNV1_64_PRIME)
0979
0980 #define FNV_32A_OP_GENERIC(hash, octet) \
0981     (((u_int32_t)(hash) ^ (u_int8_t)(octet)) * 16777619)
0982
0983 #define FNV_32A_OP(hash, octet) \
0984     (((u_int32_t)(hash) ^ (u_int8_t)(octet)) * FNV1_32_PRIME)
0985
0986 #define FNV_32A_OP_MULess_core(hash, octet) \
0987     ((u_int32_t)(hash) ^ (u_int8_t)(octet) )
0988
0989 #define FNV_32A_OP_MULess(hash, octet) \
0990     (( FNV_32A_OP_MULess_core(hash, octet)<<5) - FNV_32A_OP_MULess_core(hash, octet) )
0991
0992 #define FNV_32A_OP32(hash, octet) \
0993     (((u_int32_t)(hash) ^ (u_int32_t)(octet)) * FNV1_32_PRIME)
0994
0995 #define FNV_32A_OP64(hash, octet) \
0996     (((u_int64_t)(hash) ^ (u_int64_t)(octet)) * FNV1_32_PRIME)
0997
0998 #define FNV_32A_OP32_MULess_core(hash, octet) \
0999     ((u_int32_t)(hash) ^ (u_int32_t)(octet) )
1000
1001 #define FNV_32A_OP32_MULess(hash, octet) \
1002     (( FNV_32A_OP32_MULess_core(hash, octet)<<5) - FNV_32A_OP32_MULess_core(hash, octet) )
1003
1004
1005 // Invoking: FNV1A_Hash_4_OCTETS_31(wrd, wrdlen>>2) // = 0,1,2,3,4,5,6,7 [1..31]
1006 int FNV1A_Hash_4_OCTETS_31(char *str, int wrdlen_QUADRUPLTS)
1007 {
1008     u_int32_t hash;
1009     char *p;
1010
1011     hash = FNV1_32_INIT;
1012     p=str;
1013
1014     // The goal of stage #1: to reduce number of 'imul's in fact to reduce loops.
1015
1016     // Stage #1:
1017     for (; wrdlen_QUADRUPLTS != 0; --wrdlen_QUADRUPLTS) {
1018         hash = FNV_32A_OP32_MULess(hash, (unsigned long)*(long *p)); // mov edi, DWORD PTR [eax]
1019         p=p+4; // add eax, 4
1020     }
1021
1022     // Stage #2:
1023     for (; *p; ++p) {
1024         hash = FNV_32A_OP_MULess(hash, *p); // mov dl, BYTE PTR [ecx]
1025     }
1026
1027     //return ((hash>>13) ^ hash) & 8191; // (((u_int32_t)1<<(x))-1) where x=13
1028     return ((hash>>16) ^ hash) & 8191; // 00..8191 i.e. 2^13=8192
1029 }
1030
1031
1032 // Invoking: FNV1A_Hash_4_OCTETS(wrd, wrdlen>>2) // = 0,1,2,3,4,5,6,7 [1..31]
1033 int FNV1A_Hash_4_OCTETS(char *str, int wrdlen_QUADRUPLTS)
1034 {
1035     u_int32_t hash;
1036     char *p;
1037
1038     hash = FNV1_32_INIT;
1039     p=str;
1040
1041     // The goal of stage #1: to reduce number of 'imul's.
1042
1043     // Stage #1:
1044     for (; wrdlen_QUADRUPLTS != 0; --wrdlen_QUADRUPLTS) {
1045         hash = FNV_32A_OP32(hash, (unsigned long)*(long *p)); // mov edi, DWORD PTR [eax]
1046         p=p+4; // add eax, 4
1047     }
1048
1049     // Stage #2:
1050     for (; *p; ++p) {
1051         hash = FNV_32A_OP(hash, *p); // mov dl, BYTE PTR [ecx]
1052     }

```

```

1053
1054 //return ((hash>>13) ^ hash) & 8191; // (((u_int32_t)1<<(x))-1) where x=13
1055 return ((hash>>16) ^ hash) & 8191; // 00..8191 i.e. 2^13=8192
1056 }
1057
1058
1059 // Invoking: FNV1A_Hash_8_OCTETS(wrd, wrdlen>>3) // = 0,1,2,3 [1..31]
1060 int FNV1A_Hash_8_OCTETS(char *str, int wrdlen_OCTETS)
1061 {
1062     u_int32_t hash;
1063     char *p;
1064
1065     hash = FNV1_32_INIT;
1066     p=str;
1067
1068     // The goal of stage #1: to reduce number of 'imul's.
1069
1070     // Stage #1:
1071     for (; wrdlen_OCTETS != 0; --wrdlen_OCTETS) {
1072         hash = FNV_32A_OP64(hash, (unsigned long long)*(long *)p); // mov edi, DWORD PTR [eax]
1073         p=p+8; // add eax, 4
1074     }
1075
1076     // Stage #2:
1077     for (; *p; ++p) {
1078         hash = FNV_32A_OP(hash, *p); // mov dl, BYTE PTR [ecx]
1079     }
1080
1081     //return ((hash>>13) ^ hash) & 8191; // (((u_int32_t)1<<(x))-1) where x=13
1082     return ((hash>>16) ^ hash) & 8191; // 00..8191 i.e. 2^13=8192
1083 }
1084
1085
1086 // Invoking: FNV1A_Hash_Granularity(wrd, wrdlen>>0|2|3, 0|2|3)
1087 int FNV1A_Hash_Granularity(char *str, int wrdlen_granulated, int Granularity) // wrdlen>>0=wrdlen
1088 {
1089     u_int32_t hash;
1090     u_int64_t hash64;
1091     char *p;
1092
1093     hash = FNV1_32_INIT;
1094     p=str;
1095
1096     // The goal of stage #1: to reduce number of 'imul's and mainly: the number of loops.
1097
1098     // Stage #1:
1099     if (Granularity == 2) {
1100     for (; wrdlen_granulated != 0; --wrdlen_granulated) {
1101         hash = FNV_32A_OP32(hash, (u_int32_t)*(u_int32_t *)p);
1102         p=p+4; // (1<<Granularity): 1<<0=1, 1<<2=4, 1<<3=8
1103     }
1104     }
1105     if (Granularity == 3) {
1106     hash64 = FNV1_64_INIT;
1107     for (; wrdlen_granulated != 0; --wrdlen_granulated) {
1108         hash64 = FNV_64A_OP64(hash64, (u_int64_t)*(u_int64_t *)p);
1109         p=p+8; // (1<<Granularity): 1<<0=1, 1<<2=4, 1<<3=8
1110     }
1111     for (; *p; ++p) {
1112         hash64 = FNV_64A_OP(hash64, (u_int8_t)*(u_int8_t *)p);
1113     }
1114
1115     //return ((hash>>13) ^ hash) & 8191; // (((u_int32_t)1<<(x))-1) where x=13
1116     return ((hash64>>51) ^ hash64) & 8191; // 00..8191 i.e. 2^13=8192
1117     // probably better shifting is not by 16 bits but ...
1118     //hash64>>16: 3,544,160 just bad
1119     //hash64>>33: 3,547,854
1120     //hash64>>34: 3,547,266
1121     //hash64>>35: 3,547,453
1122     //hash64>>36: 3,547,242
1123     //hash64>>40: 3,548,263
1124     //hash64>>44: 3,548,242
1125     //hash64>>45: 3,549,056
1126     //hash64>>46: 3,549,207
1127     //hash64>>47: 3,549,094
1128     //hash64>>50: 3,549,392
1129     //hash64>>51: 3,549,395 i.e. maximum shift: the 13 most significant bits i.e. (64-13); closest to 3,549,448
1130
1131     // Above results are obtained for following set:
1132     //if (wrdlen<=19) // 4x4+3=19 i.e. last contains 7 clashes
1133     //    Slot = FNV1A_Hash_Granularity(wrd, wrdlen>>2, 2)<<2; //13++++
1134     //else // 2x8+4=20 i.e. first contains 6 clashes
1135     //    Slot = FNV1A_Hash_Granularity(wrd, wrdlen>>3, 3)<<2; //13++++
1136     //}
1137
1138     //if (Granularity != 3) {
1139     // Stage #2:
1140     for (; *p; ++p) {

```

```

1141     hash = FNV_32A_OP(hash, (u_int8_t)*(u_int8_t *)p);
1142 }
1143
1144 //return ((hash>>13) ^ hash) & 8191; // (((u_int32_t)1<<(x))-1) where x=13
1145 return ((hash>>16) ^ hash) & 8191; // 00..8191 i.e. 2^13=8192
1146 //}
1147 }
1148
1149
1150 // char *string; // the string to 64 bit FNV-1a hash */
1151 // u_int64_t hash; // will hold the final value of the hash */
1152 // char *p;
1153 //
1154 // hash = FNV1_64_INIT;
1155 // for (p=string; *p; ++p) {
1156 //     hash = FNV_64A_OP(hash, *p);
1157 // }
1158
1159
1160 // If you need an x-bit hash where x is not a power of 2,
1161 // then we recommend that you compute the FNV hash that is just larger than x-bits and xor-fold the result down to x-bits.
1162 // By xor-folding we mean shift the excess high order bits down and xor them with the lower x-bits.
1163 // For tiny x < 16 bit values, we recommend using a 32 bit FNV-1 hash as follows:
1164
1165 // /* NOTE: for 0 < x < 16 ONLY!!! */
1166 // #define TINY_MASK(x) (((u_int32_t)1<<(x))-1)
1167 // #define FNV1_32_INIT ((u_int32_t)2166136261)
1168 // u_int32_t hash;
1169 // void *data;
1170 // size_t data_len;
1171 //
1172 // hash = fnv_32_buf(data, data_len, FNV1_32_INIT);
1173 // hash = (((hash>>x) ^ hash) & TINY_MASK(x));
1174
1175
1176 int FNV1A_Hash_SHIFTless_XORless(char *str)
1177 {
1178     u_int32_t hash; // will hold the final value of the hash */
1179     char *p;
1180
1181     hash = FNV1_32_INIT;
1182     for (p=str; *p; ++p) {
1183         hash = FNV_32A_OP(hash, *p);
1184     }
1185     //hash = ((hash>>13) ^ hash) & 8191; // (((u_int32_t)1<<(x))-1) where x=13
1186
1187     return hash & 8191; // 00..8191 i.e. 2^13=8192
1188 }
1189
1190 /*
1191 _FNV1A_Hash_SHIFTless_XORless PROC NEAR
1192 ; Line 721
1193 mov edx, DWORD PTR _str$[esp-4]
1194 mov cl, BYTE PTR [edx]
1195 test cl, cl
1196 mov eax, -2128831035 ; 811c9dc5H
1197 je SHORT $L1582
1198 npad 1
1199 $L1580:
1200 ; Line 722
1201 movzx ecx, cl
1202 xor ecx, eax
1203 imul ecx, 16777619 ; 01000193H
1204 inc edx
1205 mov eax, ecx
1206 mov cl, BYTE PTR [edx]
1207 test cl, cl
1208 jne SHORT $L1580
1209 $L1582:
1210 ; Line 726
1211 and eax, 8191 ; 00001ffffH
1212 ; Line 727
1213 ret 0
1214 _FNV1A_Hash_SHIFTless_XORless ENDP
1215 */
1216
1217
1218 int FNV1A_Hash(char *str)
1219 {
1220     u_int32_t hash; // will hold the final value of the hash */
1221     char *p;
1222
1223     hash = FNV1_32_INIT;
1224     for (p=str; *p; ++p) {
1225         hash = FNV_32A_OP(hash, *p);
1226     }
1227     //hash = ((hash>>13) ^ hash) & 8191; // (((u_int32_t)1<<(x))-1) where x=13
1228

```



```

1229 return ((hash>>13) ^ hash) & 8191; // 00..8191 i.e. 2^13=8192
1230 }
1231
1232 /*
1233 _FNV1a_Hash PROC NEAR
1234 ; Line 722
1235 mov edx, DWORD PTR _str$[esp-4]
1236 mov al, BYTE PTR [edx]
1237 test al, al
1238 mov ecx, -2128831035 ; 811c9dc5h
1239 je SHORT $L1582
1240 npad1
1241 $L1580:
1242 ; Line 723
1243 movzx eax, al
1244 xor eax, ecx
1245 imul eax, 16777619 ; 01000193h
1246 inc edx
1247 mov ecx, eax
1248 mov al, BYTE PTR [edx]
1249 test al, al
1250 jne SHORT $L1580
1251 $L1582:
1252 ; Line 727
1253 mov eax, ecx
1254 shr eax, 13 ; 0000000dh
1255 xor eax, ecx
1256 and eax, 8191 ; 00001fffh
1257 ; Line 728
1258 ret 0
1259 _FNV1a_Hash ENDP
1260 */
1261
1262 /*
1263 Wayne Diamond implemented 32-bit FNV algorithm in PowerBASIC inline x86 assembly:
1264
1265 FUNCTION FNV32(BYVAL dwOffset AS DWORD, BYVAL dwLen AS DWORD, BYVAL offset_basis AS DWORD) AS DWORD
1266 #REGISTER NONE
1267 ! mov esi, dwOffset ;esi = ptr to buffer
1268 ! mov ecx, dwLen ;ecx = length of buffer (counter)
1269 ! mov eax, offset_basis ;set to 2166136261 for FNV-1
1270 ! mov edi, &h01000193 ;FNV_32_PRIME = 16777619
1271 ! xor ebx, ebx ;ebx = 0
1272 ! nextbyte:
1273 ! mul edi ;eax = eax * FNV_32_PRIME
1274 ! mov bl, [esi] ;bl = byte from esi
1275 ! xor eax, ebx ;al = al xor bl
1276 ! inc esi ;esi = esi + 1 (buffer pos)
1277 ! dec ecx ;ecx = ecx - 1 (counter)
1278 ! jnz nextbyte ;if ecx is 0, jmp to NextByte
1279 ! mov FUNCTION, eax ;else, function = eax
1280 END FUNCTION
1281
1282 Wayne said:
1283
1284 ''Just thought I should let you know that I've ported the 32-bit FNV algorithm over to inline assembly.
1285 It's actually in PowerBASIC (www.powerbasic.com) format - a compiler I use, but the main function is all assembly.
1286 It could be optimized further in terms of saving a couple of clock cycles,
1287 but it's fairly optimized al ready - only 6 instructions in the main loop, plus 5 setup instructions,
1288 and compiles to just 33 bytes.''
1289
1290 M.S.Schulte sent us these 32-bit FNV-1 and FNV-1a x86 assembler implementations (written in flat assembler),
1291 half of which were optimized for speed, the other half were optimized for size:
1292
1293 small_fnv32: ;FNV1 32bit (size: 31 bytes)
1294 ; Intel Core 2 Duo E6600: 354.20 mb/s
1295
1296 push esi
1297 push edi
1298 mov esi, [esp + 0ch] ;buffer
1299 mov ecx, [esp + 10h] ;length
1300 mov eax, [esp + 14h] ;basis
1301 mov edi, 01000193h ;fnv_32_prime
1302
1303 next:
1304 mul edi
1305 xor al, [esi]
1306 inc esi
1307 loop snext
1308 pop edi
1309 pop esi
1310 retn 0ch
1311
1312 small_fnv32a: ;FNV1a 32bit (size: 31 bytes)
1313 ; Intel Core 2 Duo E6600: 327.68 mb/s
1314
1315 push esi
1316 push edi
1317 mov esi, [esp + 0ch] ;buffer
1318 mov ecx, [esp + 10h] ;length

```

```

1317 mov eax, [esp + 14h] ;basis
1318 mov edi, 01000193h ;fnv_32_prime
1319
1320 nexta:
1321 xor al, [esi]
1322 mul edi
1323 inc esi
1324 loop nexta
1325 pop edi
1326 pop esi
1327 retn 0ch
1328
1329 fast_fnv32: ;FNV1 32bit (size: 36 bytes)
1330 ; Intel Core 2 Duo E6600: 565.12 mb/s
1331
1332 push ebx
1333 push esi
1334 push edi
1335 mov esi, [esp + 10h] ;buffer
1336 mov ecx, [esp + 14h] ;length
1337 mov eax, [esp + 18h] ;basis
1338 mov edi, 01000193h ;fnv_32_prime
1339 xor ebx, ebx
1340
1341 next:
1342 mul edi
1343 mov bl, [esi]
1344 xor eax, ebx
1345 inc esi
1346 dec ecx
1347 jnz next
1348 pop edi
1349 pop esi
1350 pop ebx
1351 retn 0ch
1352
1353 fast_fnv32a: ;FNV1a 32bit (size: 36 bytes)
1354 ; Intel Core 2 Duo E6600: 574.95 mb/s
1355
1356 push ebx
1357 push esi
1358 push edi
1359 mov esi, [esp + 10h] ;buffer
1360 mov ecx, [esp + 14h] ;length
1361 mov eax, [esp + 18h] ;basis
1362 mov edi, 01000193h ;fnv_32_prime
1363 xor ebx, ebx
1364
1365 nexta:
1366 mov bl, [esi]
1367 xor eax, ebx
1368 mul edi
1369 inc esi
1370 dec ecx
1371 jnz nexta
1372 pop edi
1373 pop esi
1374 pop ebx
1375 retn 0ch
1376
1377 /*
1378
1379 [FNV1a 'shift-less-&-xor-less' hash used in Leprechaun r.13+++:]
1380
1381 int FNV1a_Hash_SHIFTless_XORless(char *str)
1382 {
1383 u_int32_t hash;
1384 char *p;
1385
1386 hash = FNV1_32_INIT;
1387 for (p=str; *p; ++p) {
1388 hash = FNV_32A_OP(hash, *p);
1389 }
1390 //hash = ((hash>>13) ^ hash) & 8191; // (((u_int32_t)1<<(x))-1) where x=13
1391
1392 return hash & 8191; // 00..8191 i.e. 2^13=8192
1393 }
1394
1395 1391 Words per second performance: 837,458w/s
1396 Input File with a list of TEXTual Files: wikipedia-en-html.tar.wrd.lst
1397 word count: 12,561,874 of them 12,561,874 distinct
1398 Number of Trees(GREATER THE BETTER): 2772875
1399 Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 41%
1400 Number of Hash Collisions(Distinct WORDS - Number of Trees): 9788999
1401
1402 1397 Words per second performance: 1,007,751w/s
1403 Input File with a list of TEXTual Files: Leprechaun_vs.Wikipedia-LATIN-WORDS.lst
1404 word count: 35,271,297 of them 22,202,980 distinct
1405 Number of Trees(GREATER THE BETTER): 3537061
1406 Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 53%
1407 Number of Hash Collisions(Distinct WORDS - Number of Trees): 18665919
1408

```

```

1405 [My '2in1' hash used in Leprechaun r.13+:+:]
1406
1407 int KuxHash3plus(char *str)
1408 { int h = 0;
1409   unsigned long h2 = 0; // must be long: 31*'z'=31*122
1410   int max31 = 0;
1411   while (str[max31])
1412   { h = h ^ str[max31]; // 00..255 i.e. 2^8=256
1413     //h2 = h2 + str[max31++]; // [113s]
1414     h2 = h2 + str[max31++] * (max31+1);
1415   }
1416   // Result is: 7bits in 'h' and 32bits in 'h2'.
1417
1418   //printf("%s:\n ",str);
1419   //printf("%d ",h);
1420   // a in ASCII is 097 = 0110 0001
1421   // z in ASCII is 122 = 0111 1010
1422   // Above two lines show that bits 8-7-6 are always 0-1-1 so need for low 5 bits.
1423   //h=h<<8; // 00..15 i.e. 5bits + 00-07bits=13bits
1424   //printf("%d ",h);
1425   //printf("%d ",h2);
1426   //h = h[( str[0] ^ str[max31-1] ); // [111s] a..z: each XOR each gives 00..31
1427   h = (( h<<8 )[( h2%(251) )])&8191; // 251 prime
1428   //printf("%d\n",h);
1429   return h; // 00..8191 i.e. 2^13=8192
1430 }
1431
1432 Words per second performance: 785,117w/s
1433 Input File with a list of TEXTual Files: wikipedia-en.html.tar.wrd.lst
1434 Word count: 12,561,874 of them 12,561,874 distinct
1435 Number Of Trees(GREATER THE BETTER): 2663566
1436 Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 40%
1437 Number Of Hash Collisions(Distinct WORDS - Number Of Trees): 9898308
1438
1439 Words per second performance: 979,758w/s
1440 Input File with a list of TEXTual Files: Leprechaun_vs_Wikipedia_LATIN-WORDS.lst
1441 Word count: 35,271,297 of them 22,202,980 distinct
1442 Number Of Trees(GREATER THE BETTER): 3410463
1443 Forest population(Hash Function Quality regarding Collisions i.e. Hash Table Utilization): 51%
1444 Number Of Hash Collisions(Distinct WORDS - Number Of Trees): 18792517
1445
1446 [Last standing for English(en)-wikipedia's wordlist:]
1447 chongo's hash is faster(in total, not the function itself) than Kaze's hash by ((837,458w/s - 785,117w/s)/785,117w/s)*100% = 6.6%
1448 chongo's hash has better distribution than Kaze's hash by ((9898308 - 9788999)/9788999)*100% = 1.1%
1449
1450 [Last standing for LATIN(de,en,es,fr,it,nl,pt,ro)-wikipedia's wordlist:]
1451 chongo's hash is faster(in total, not the function itself) than Kaze's hash by ((1,007,751w/s - 979,758w/s)/979,758w/s)*100% = 2.8%
1452 chongo's hash has better distribution than Kaze's hash by ((18792517 - 18665919)/18665919)*100% = 0.6%
1453
1454 Bottomline is:
1455 Your hash thrash, my hash for trash, he-he.
1456 Thanks a lot, again, Mr. Noll.
1457
1458 Yummy little file: http://www.isthe.com/chongo/src/fnv/fnv-5.0.2.tar.gz
1459 */
1460
1461
1462 // The following example code in the C language computes the binary logarithm (rounding down) of an integer, rounded down. [2] The operator
1463 '>>' represents 'unsigned right shift'. The rounding down form of binary logarithm is identical to computing the position of the most
1464 significant 1 bit.
1465
1466 /*
1467 * Returns the floor form of binary logarithm for a 32 bit integer.
1468 * -1 is returned if n is 0.
1469 */
1470 int FloorLog2(unsigned int n) {
1471   int pos = 0;
1472   if (n >= 1<<16) { n >>= 16; pos += 16; }
1473   if (n >= 1<<8) { n >>= 8; pos += 8; }
1474   if (n >= 1<<4) { n >>= 4; pos += 4; }
1475   if (n >= 1<<2) { n >>= 2; pos += 2; }
1476   if (n >= 1<<1) { n >>= 1; pos += 1; }
1477   return ((n == 0) ? (-1) : pos);
1478 }
1479
1480 int main( argc, argv )
1481 { int argc; char *argv[];
1482   int nLines;
1483   string *backup = NULL;
1484
1485   FILE *fp_in, *fp_out, *fp_outLOG, *fp_inLINE;
1486   int LetterOffset;
1487   unsigned long long FilesLEN;
1488   unsigned long long WORDcount;
1489   unsigned long long WORDcountAttemptsToPut;
1490   int Thunderwith;
1491   unsigned long NumberOfFiles, WORDcountDistinct;

```

```

1491   unsigned long long NumberOfLines; // rev. 12+
1492   unsigned long WHOLELetter_Buffersize;
1493   unsigned long memory_size, LetterBuffer, j, k, LINE10len, wrdlen;
1494   unsigned long long i; // rev. 12+
1495   //unsigned long size_in, size_out, size_inLINE;
1496   unsigned long size_in; // rev. 12+
1497   #if defined(_WIN32_ENVIRONMENT_)
1498   unsigned long long size_inLINESIXFOUR;
1499   #else
1500   size_t size_inLINESIXFOUR;
1501   #endif /* defined(_WIN32_ENVIRONMENT_) */
1502
1503   //unsigned long t1, t2, t3;
1504   time_t t1, t2, t3;
1505
1506   const int NumberOfSlots = 4096*2; // Since r.12+ in rev.12 it was 4096
1507   unsigned long StackPtr;
1508   unsigned long BSTstack [65536*3]; // BST in worst case could become a LL.
1509   unsigned long NumberOfTrees=0, NumberOfHashCollisions=0;
1510   unsigned long iBSTwithMAXpeak, jBSTwithMAXpeak;
1511   unsigned int PEAKiBST;
1512   unsigned long BSTSTotalLEAFs=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur below where 'break'
1513   is ?!
1514   unsigned long BSTwithMAXnode=0, BSTcurrentNode=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur
1515   below where 'break' is ?!
1516   unsigned long BSTcurrentNodeMAXQUANTITY=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of
1517   BSTcurrent occur below where 'break' is ?!
1518   unsigned long BSTwithMAXnodePEAK=1, BSTwithMAXnodeLEAF=1; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent
1519   occur below where 'break' is ?!
1520   unsigned long BSTwithMAXpeak=0, BSTcurrentPeak=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur
1521   below where 'break' is ?!
1522   unsigned long BSTcurrentPeakMAX=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur
1523   below where 'break' is ?!
1524   unsigned long BSTcurrentPeakMAXQUANTITY=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of
1525   BSTcurrent occur below where 'break' is ?!
1526   unsigned long BSTwithMAXpeakNODE=1, BSTwithMAXpeakLEAF=1; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent
1527   occur below where 'break' is ?!
1528   unsigned long BSTwithMAXleaf=0, BSTcurrentLeaf=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent occur
1529   below where 'break' is ?!
1530   unsigned long BSTcurrentLeafMAXQUANTITY=0; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of
1531   BSTcurrent occur below where 'break' is ?!
1532   unsigned long BSTwithMAXleafNODE=1, BSTwithMAXleafPEAK=1; // ?! MADHOUSE: if BSTcurrent is not zeroed here then INSANE values of BSTcurrent
1533   occur below where 'break' is ?!
1534
1535   char *pointerFlush, *BufStart, *Flushing;
1536   unsigned long PseudoLinkedPointer, PseudoLinkedPointerNEW, PseudoLinkedPointerROOT, PseudoLinkedPointerNEWold;
1537   unsigned long PseudoLinkedPointerNEWleft, PseudoLinkedPointerNEWright;
1538   unsigned long PseudoLinkedPointerNEWmiddle;
1539   char *bufend[ 806 ]; // 'a'=0, ... 'z'=25 - 26 letters x 31 lengths
1540   long bufNumberOfWords[ 806 ]; // 'a'=0, ... 'z'=25 - 26 letters x 31 lengths
1541   // long bufNowps[ 806 ][ 8192 ]; // ?! crashes below when an attempt to use it occur
1542   char wrd[32]; // 0..30, 31 = 0
1543   char wrdUP[32]; // 0..30, 31 = 0
1544   char wrdUPold[32]; // 0..30, 31 = 0
1545   char LINE10[257]; // 000..255, 256 = 0
1546   char ZEROS[4]; // 0..3, 0 = 0, 1 = 0, 2 = 0, 3 = 0
1547   char CrdLfA[2]; // 0..1, 0 = 13, 1 = 10
1548   char workbyte;
1549   char workk[1024*96];
1550   long workOffset = -1;
1551   int FoundInLinkedList, Slot;
1552   unsigned long OffsetsInBuffer[31]; // 00..30
1553   unsigned long MAXusedBuffer[32]; // 00 not used, only 01..31
1554   unsigned long GRMBLhill[32]; // 00..31
1555   unsigned long GRMBLPoolAgain[32]; // 00..31
1556   int Melnitcka;
1557   unsigned long MAXusedBufferABS = 0;
1558   unsigned long Utiliza1 = 0;
1559   unsigned long Utiliza2 = 0;
1560   unsigned long TotalWlchars = 0;
1561
1562   /* minimum signed 64 bit value */
1563   #define _I64_MIN (-9223372036854775807i64 - 1)
1564   /* maximum signed 64 bit value */
1565   #define _I64_MAX 9223372036854775807i64
1566   /* maximum unsigned 64 bit value */
1567   #define _UI64_MAX 0xffffffffffffffffui64
1568
1569   /* minimum signed 128 bit value */
1570   #define _i128_MIN (-170141183460469231731687303715884105727i128 - 1)
1571   /* maximum signed 128 bit value */
1572   #define _i128_MAX 170141183460469231731687303715884105727i128
1573   /* maximum unsigned 128 bit value */
1574   #define _ui128_MAX 0xffffffffffffffffffffffffffffffffui128
1575
1576   char l1ToAdigits[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)
1577   // below duplicates are needed because of one_line invoking need different buffers.
1578   char l1ToAdigits2[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(,)

```

```

1568 char l1ToaDigs3[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(.);
1569 char l1ToaDigs4[27]; // 9,223,372,036,854,775,807: 1(sign or carry)+19(digits)+1('\0')+6(.);
1570 unsigned long HEADOffsetFromStartBUKVA = 0;
1571 unsigned long TAILOffsetFromStartBUKVA = 0;
1572 int BStorBtree = 0;
1573 int SplitOccurred;
1574 int POffsetInLEAF;
1575
1576 // INIT INIT INIT INIT INIT INIT INIT INIT INIT INIT INIT INIT INIT INIT INIT
1577 puts( "Leprechaun(Fast Greedy Word-Ripper), revision 13+++++, written by Svalqyatchx." );
1578 puts( "Leprechaun: 'Oh, well, didn't you hear? Bigger is good, but jumbo is dear.' " );
1579 puts( "Kaze: Let's see what a 3-way hash + 6,602,752 Binary-Search-Trees can give us." );
1580 puts( "    also the performance of a 3-way hash + 6,602,752 B-Trees of order 3." );
1581 //puts( "Note1: Compiled with Microsoft C v. 13.10.3077: 'cl /ox /TCLeprechaun.c.' " );
1582 //puts( "Note2: This WORDLISTER makes as output pseudo(unsorted)_wordlist_CRLF_file." );
1583 if( argc != 3 && argc != 4 && argc != 5 && argc != 6 ) // +1 for program name
1584 {
1585     puts( "" );
1586     puts( "The Little Monster' short notes:" );
1587     puts( "Note1: I wish to thank to R.N. Horspool, Ranjan Sinha, Dmitry Shkarin." );
1588     puts( "    Michael Abrash, J. Bentley, R. Sedgewick, Igor Pavlov, Lasse Reinhold." );
1589     puts( "    Landon Noll for sharing their knowledge to public." );
1590     puts( "Note2: Run it without parameters to get usage and short notes." );
1591     puts( "Note3: This simple amateurish(more over I am not versed well neither in C nor );
1592           in mathematics nor in english language, but I am persistent in INDEXING );
1593           GBS of english TEXTS tool is written in ANSI C(at least its source is );
1594           compileable for CL(windows) and GCC(Linux), and its purpose is to );
1595           create a wordList for a group of files(given via fileList)." );
1596     puts( "    Its name comes(according to Heritage Dictionary) from 'low corpus' or );
1597     puts( "    'little body', in fact from amazing movie saga 'Leprechaun 1-2-3-4-5-6' );
1598     puts( "    starring by Warwick Davis." );
1599     puts( "Note4: Only words up to 31 chars are proceeded - the reason is 'DDT'(the" );
1600     puts( "    longest word in Heritage Dictionary 3rd edition) or" );
1601     puts( "    'dichlorodiphenyltrichloroethane' );
1602     puts( "Note5: Cursor hiding in C - mission impossible for me." );
1603     puts( "Note6: By default(third parameter is 1023) allocated memory is 393MB." );
1604     puts( "    Due to 'malloc()' limitation under WINDOWS, maximum value of third" );
1605     puts( "    parameter is 5174 which is 1988MB allocated block." );
1606     puts( "Note7: File Leprechaun.LOG is a log, where new statistics are appended." );
1607     puts( "Note8: Revision 12+ can handle files larger than 4GB." );
1608     puts( "Note9: Revision 12++ has a buffered 'fread()' - therefore I/O READ-BURST SPEED );
1609     puts( "    is the first(worst) bottleneck, as a result r.12++ is much-much faster;" );
1610     puts( "    the second(worse) bottleneck: the linked lists - the b-trees );
1611     puts( "    might be the answer; the third(bad) bottleneck: the amateurish author." );
1612     puts( "NoteA: Revision 12+++ has an improved(2 bits were used dolitshly) main hash" );
1613     puts( "    function - therefore less collisions, for example" );
1614     puts( "    for file 'wikipedia-de-html.tar' 42,291,855,360 bytes with" );
1615     puts( "    5,750,179,678 words of them 7,375,373 distinct attempts to Find/Put" );
1616     puts( "    a WORD into a linked list are 6,117,675,470(r.12++) and 5,845,989,790 );
1617     puts( "    (r.12+++); also two 'if' sections were moved because they were executed );
1618     puts( "    unnecessarily many times." );
1619     puts( "NoteB: Revision 13 uses BSTs instead of LLs, that is Linked-Lists were );
1620     puts( "    replaced by Binary-Search-Trees, as a result for 22,202,980 distinct" );
1621     puts( "    words(out of 35,271,297) r.12+++ needs 225,548,268 total attempts to );
1622     puts( "    Find/Put WORDS into linked lists where r.13 needs 121,674,042 total );
1623     puts( "    attempts to Find/Put WORDS into Binary-Search-Trees. But this is a );
1624     puts( "    significant boost in performance only for wordlists of million words." );
1625     puts( "NoteC: Revision 13+ gives only more statistics. Future revisions could lessen );
1626     puts( "    number of attempts to Find/Put WORDS into Binary-Search-Trees" );
1627     puts( "    furthermore by making them at some point Perfectly-Balanced. But );
1628     puts( "    for huge amount(Multi-(M)billion) of distinct words the b-tree family );
1629     puts( "    must come in, until then this is the leprechaunish niche." );
1630     puts( "NoteD: Revision 13++ has a little fix(2 unnecessary ZEROings, when a new word" );
1631     puts( "    is inserted, were deleted) and a fixed bug(13+ adds stupidly the );
1632     puts( "    highest BST to the wordlist). Also B-Tree of order 3 is added as a );
1633     puts( "    searching method. Main goal of B-Tree is to reduce number of );
1634     puts( "    comparisons but at nasty cost: a precious time wasted to construct it );
1635     puts( "    and twice more memory, i.e. one step forward two backward: this tree is );
1636     puts( "    more effective than BST in cases of 2++ billion/million );
1637     puts( "    different/distinct words." );
1638     puts( "The improvement which comes from using B-Tree of order 3 is about 200%" );
1639     puts( "    much more pleasing than I expected, for wikipedia-en-html.tar.wrd with" );
1640     puts( "    12,561,874 distinct words Total Attempts to Find/Put WORDS into:" );
1641     puts( "    Binary-Search-Trees was 61,895,043 while for" );
1642     puts( "    B-trees order 3 was 19,295,791." );
1643     puts( "NoteE: Revision 13+++ has a faster(not heavily tested yet) and with" );
1644     puts( "    better(0.6% to 1.1%) dispersion Fowler/Noll/vo hash." );
1645     puts( "    so called FNV1a hash. Revision 13++++ boosting: Leprechaun_Intel.exe );
1646     puts( "    gives 1,256,187w/s for wikipedia-en-html.tar.wrd with FNV1_32_PRIME." );
1647     puts( "    107712257 with 3,551,736 dispersion for 'FNV1A_hash_Granularity' );
1648     puts( "NoteF: For old r.12+ a USB connected HDD crippled test:" );
1649     puts( "    for 'H:\>\Leprechaun.exe static.wikipedia.org_downloads_2008-06_en.lst" );
1650     puts( "    wikipedia-en-html.tar.wrd 5400" );
1651     puts( "    where 223,674,511,360 wikipedia-en-html.tar" );
1652     puts( "    on laptop Toshiba Pentium T3400 2166 MHz with" );
1653     puts( "    Motherboard Name: Toshiba Satellite L305" );
1654     puts( "    CPU Type: Mobile DualCore Intel Pentium, 2166 MHz (13 x 167)" );
1655     puts( "    CPU Alias: Merom-1M" );

```

```

1656 puts( "    L1 Code Cache: 32 KB per core" );
1657 puts( "    L1 Data Cache: 32 KB per core" );
1658 puts( "    L2 Cache: 1 MB (On-Die, ECC, ASC, Full-Speed)" );
1659 puts( "    Bus Type: Dual DDR2 SDRAM" );
1660 puts( "    Bus Width: 128-bit" );
1661 puts( "    Real Clock: 333 MHz (DDR)" );
1662 puts( "    Effective Clock: 666 MHz" );
1663 puts( "    EVEREST v5.00.1650 Memory Copy: 3725MB/s with timings 5-5-5-13" );
1664 puts( "    result is logged to 'Leprechaun.LOG'." );
1665 puts( "    Bytes per second performance: 20,658,955B/s" );
1666 puts( "    Words per second performance: 2,860,880W/s" );
1667 puts( "    Input File with a list of TEXTUAL Files:" );
1668 puts( "    static.wikipedia.org_downloads_2008-06_en.lst" );
1669 puts( "    Size of all TEXTUAL Files: 223,674,511,360" );
1670 puts( "    Word count: 30,974,750,142 of them 12,561,874 distinct" );
1671 puts( "    Number of Files: 1" );
1672 puts( "    Number of Lines: 2088618575" );
1673 puts( "    Allocated memory in MB: 1920" );
1674 puts( "    Words with length 01 occupy 0,033KB of 0,349KB given i.e. 09% utilization" );
1675 puts( "    Words with length 02 occupy 0,033KB of 0,349KB given i.e. 09% utilization" );
1676 puts( "    Words with length 03 occupy 0,037KB of 0,697KB given i.e. 05% utilization" );
1677 puts( "    Words with length 04 occupy 0,151KB of 0,871KB given i.e. 17% utilization" );
1678 puts( "    Words with length 05 occupy 0,744KB of 1,568KB given i.e. 47% utilization" );
1679 puts( "    Words with length 06 occupy 1,470KB of 3,136KB given i.e. 46% utilization" );
1680 puts( "    Words with length 07 occupy 2,605KB of 5,923KB given i.e. 43% utilization" );
1681 puts( "    Words with length 08 occupy 3,296KB of 6,968KB given i.e. 47% utilization" );
1682 puts( "    Words with length 09 occupy 3,714KB of 6,968KB given i.e. 53% utilization" );
1683 puts( "    Words with length 10 occupy 3,483KB of 6,968KB given i.e. 49% utilization" );
1684 puts( "    Words with length 11 occupy 3,235KB of 5,923KB given i.e. 54% utilization" );
1685 puts( "    Words with length 12 occupy 2,691KB of 4,181KB given i.e. 64% utilization" );
1686 puts( "    Words with length 13 occupy 2,230KB of 3,484KB given i.e. 64% utilization" );
1687 puts( "    Words with length 14 occupy 1,718KB of 3,484KB given i.e. 49% utilization" );
1688 puts( "    Words with length 15 occupy 1,357KB of 2,613KB given i.e. 51% utilization" );
1689 puts( "    Words with length 16 occupy 1,063KB of 2,613KB given i.e. 40% utilization" );
1690 puts( "    Words with length 17 occupy 0,814KB of 1,742KB given i.e. 46% utilization" );
1691 puts( "    Words with length 18 occupy 0,617KB of 1,742KB given i.e. 35% utilization" );
1692 puts( "    Words with length 19 occupy 0,485KB of 1,742KB given i.e. 27% utilization" );
1693 puts( "    Words with length 20 occupy 0,402KB of 1,742KB given i.e. 23% utilization" );
1694 puts( "    Words with length 21 occupy 0,327KB of 1,742KB given i.e. 18% utilization" );
1695 puts( "    Words with length 22 occupy 0,274KB of 1,742KB given i.e. 15% utilization" );
1696 puts( "    Words with length 23 occupy 0,224KB of 1,394KB given i.e. 16% utilization" );
1697 puts( "    Words with length 24 occupy 0,190KB of 1,394KB given i.e. 13% utilization" );
1698 puts( "    Words with length 25 occupy 0,162KB of 1,220KB given i.e. 11% utilization" );
1699 puts( "    Words with length 26 occupy 0,136KB of 1,220KB given i.e. 11% utilization" );
1700 puts( "    Words with length 27 occupy 0,119KB of 1,046KB given i.e. 11% utilization" );
1701 puts( "    Words with length 28 occupy 0,107KB of 0,871KB given i.e. 12% utilization" );
1702 puts( "    Words with length 29 occupy 0,091KB of 0,697KB given i.e. 13% utilization" );
1703 puts( "    Words with length 30 occupy 0,080KB of 0,523KB given i.e. 15% utilization" );
1704 puts( "    Words with length 31 occupy 0,076KB of 0,523KB given i.e. 14% utilization" );
1705 puts( "    Total pseudo(including hash table) memory utilization: 42%" );
1706 puts( "    Total real(wordlist's words VS allocated block) memory utilization: 60/1000" );
1707 puts( "    Used value for third parameter in KB: 5400" );
1708 puts( "    Use next time as third parameter: 3475" );
1709 puts( "    Time for making unsorted wordlist: 10827 second(s)" );
1710 puts( "    Time for sorting unsorted wordlist: 10 second(s)" );
1711
1712 puts( "" );
1713 puts( "Usage: Leprechaun InFile OutFile [BufferSize] [SortMethod] [TreeMethod]" );
1714 puts( "    <InFile>: Input file with files for Leprechauning, in WINDOWS console" );
1715 puts( "    you can create it by 'E:\KAZEHOME>dir *.txt/s/b-Leprechaun.lst'" );
1716 puts( "    <OutFile>: Output WORDLIST(sorted since r.9, CRLF) file" );
1717 puts( "    <BufferSize>: Optional Dynamic RAM buffer in KB, default(and minimum" );
1718 puts( "    in the same time) is 1023, i.e. omit or specify greater one" );
1719 puts( "    <SortMethod>: Optional Sort Method, default is 'd'," );
1720 puts( "    A - Insertionsort" );
1721 puts( "    B - InsertionX26Sort" );
1722 puts( "    C - MultikeyQuickSortSort by J. Bentley, R. Sedgewick" );
1723 puts( "    D - MultikeyQuickSortX26Sort by J. Bentley, R. Sedgewick" );
1724 puts( "    <TreeMethod>: Optional Tree Method, default is 'x'," );
1725 puts( "    X - Binary-Search-Trees" );
1726 puts( "    Y - B-Trees of order 3" );
1727 puts( "" );
1728 puts( "Have a nice Leprechauning." );
1729 puts( "For contacts: sanmayce@sanmayce.com" );
1730 puts( "Sanmayce Svalqyatchx 'kaze', 2005 Feb 07(rev.13++++: 2010 Aug 27)." );
1731 return( 1 );
1732 }
1733
1734 GRMBLhlll[0]=0;
1735 GRMBLhlll[1]=1;
1736 GRMBLhlll[2]=1;
1737 GRMBLhlll[3]=1;
1738 GRMBLhlll[4]=4;
1739 GRMBLhlll[5]=11;
1740 GRMBLhlll[6]=22;
1741 GRMBLhlll[7]=37;
1742 GRMBLhlll[8]=47;
1743 GRMBLhlll[9]=53;

```

```

1744 GRMBLhi11[10]=50;
1745 GRMBLhi11[11]=46;
1746 GRMBLhi11[12]=38;
1747 GRMBLhi11[13]=32;
1748 GRMBLhi11[14]=25;
1749 GRMBLhi11[15]=20;
1750 GRMBLhi11[16]=18;
1751 GRMBLhi11[17]=14;
1752 GRMBLhi11[18]=10;
1753 GRMBLhi11[19]=8;
1754 GRMBLhi11[20]=7;
1755 GRMBLhi11[21]=6;
1756 GRMBLhi11[22]=5;
1757 GRMBLhi11[23]=4;
1758 GRMBLhi11[24]=3;
1759 GRMBLhi11[25]=3;
1760 GRMBLhi11[26]=2;
1761 GRMBLhi11[27]=2;
1762 GRMBLhi11[28]=2;
1763 GRMBLhi11[29]=2;
1764 GRMBLhi11[30]=1;
1765 GRMBLhi11[31]=1;
1766
1767 if( ( fp_in = fopen( argv[1], "rb" ) ) == NULL )
1768 { printf( "Leprechaun: Can't open file %s \n", argv[1] ); return( 1 ); }
1769
1770 fseek( fp_in, 0L, SEEK_END );
1771 size_in = ftell( fp_in );
1772 fseek( fp_in, 0L, SEEK_SET );
1773 printf( "Size of input file with files for Leprechauning: %lu\n", size_in );
1774
1775 if( ( fp_outLOG = fopen( "Leprechaun.LOG", "a+" ) ) == NULL )
1776 { printf( "Leprechaun: Can't open file Leprechaun.LOG.\n" ); return( 1 ); }
1777
1778 // argc is 4|5|6 due to eventual missing BufferSize
1779 if( argc == 4 ) // not 6 due to eventual missing BufferSize and SortMethod
1780     k = 3;
1781 if( argc == 5 ) // not 6 due to eventual missing BufferSize or SortMethod
1782     k = 4;
1783 if( argc == 6 )
1784     k = 5;
1785 if ( *argv[k] == 'v' || *argv[k] == 'y' ) BStorBtree = 1;
1786
1787 if( argc == 4 || argc == 5 || argc == 6 ) Thunderwith = atoi( argv[3] );
1788 else Thunderwith = 527; // for r.12: 527=17*31 this is minimum because of 4096*14=16KB+ needed for each buffer!
1789 // for r.12+: 1023=33*31 this is minimum because of 4096*24=32KB+ needed for each buffer!
1790 if (Thunderwith < 1023) {Thunderwith = 1023;}
1791 LetterBuffer = Thunderwith * 1024;
1792 WHOLELetter_BufferSize = 0;
1793 for( i = 1; i <= 31; i++ )
1794 { OffsetsInBuffer[i-1] = 0;
1795   for( j = 1; j <= i; j++ )
1796   { OffsetsInBuffer[i-1] = OffsetsInBuffer[i-1] + (GRMBLhi11[(int)(j-1)] * LetterBuffer)/31;
1797   }
1798   WHOLELetter_BufferSize = WHOLELetter_BufferSize + (GRMBLhi11[(int)i] * LetterBuffer)/31; // Make soon 32 in order to shift >>5
1799   GRMBLFoolAgain[(int)i] = (GRMBLhi11[(int)i] * LetterBuffer)/31;
1800 }
1801 memory_size = 26 * WHOLELetter_BufferSize + 1;
1802 pointerFlush = (char *)malloc( memory_size );
1803 if( pointerFlush == NULL )
1804 { puts( "Leprechaun: Needed memory allocation denied!\n" ); return( 1 ); }
1805
1806 fprintf( fp_outLOG, "Leprechaun report:\n" );
1807
1808 printf( "Allocated memory in MB: %lu\n", (memory_size>>20)+1 );
1809
1810 // Check once for ever whether allocated memory is ZEROed? Answer: YES
1811 //for( i = 0; i < memory_size; i++ )
1812 // if ( *(char *) (pointerFlush+i) != 0 ) printf("NON-ZERO encountered, so 'NO'.");
1813
1814 for( i = 0; i < 26; i++ )
1815 { for( k = 1; k <= 31; k++ )
1816   { bufend[i*31+k-1] = pointerFlush + i * WHOLELetter_BufferSize + OffsetsInBuffer[k-1]; // i*31+k-1 must be 0..805
1817     if (i==25) { MAXusedBuffer[k] = (unsigned long)bufend[i*31+k-1]; }
1818     for( j = 0; j < (NumberOfSLOTS+1)*4; j++ ) // ? memset(bufend[i],0,(NumberOfSLOTS+1)*4);
1819     { *bufend[i*31+k-1]++ = 0;
1820       //++bufend[i*31+k-1];
1821     }
1822     if (i==25) { MAXusedBuffer[k] = (unsigned long)bufend[i*31+k-1]-MAXusedBuffer[k]; }
1823     bufNumberOfWords[i*31+k-1]=0;
1824     //for( j = 0; j < NumerofSLOTS; j++ )
1825     //bufNowps[i*31+k-1][j]=0;
1826   }
1827 }
1828
1829 // PROGRAM PROGRAM PROGRAM PROGRAM PROGRAM PROGRAM PROGRAM PROGRAM PROGRAM
1830 (void) time(&t1);
1831 Melnitcka = 0;

```

```

1832 WORDcount = 0; // Total word count i.e. for all files!
1833 WORDcountDistinct = 0;
1834 NumberOfFiles = 0;
1835 NumberOfLines = 0;
1836 FilesLEN = 0;
1837 LINE10len = 0;
1838
1839 for( k = 0; k < size_in; k++ )
1840 {
1841     fread( &workbyte, 1, 1, fp_in );
1842     if( workbyte != 10 )
1843     { if( workbyte != 13 ) // NON UNIX
1844       { if( LINE10len < 255 ) { LINE10[ LINE10len ] = workbyte; }
1845         LINE10len++;
1846       }
1847     }
1848     else
1849     {
1850     }
1851     else
1852     { if( 1 <= LINE10len && LINE10len <= 255 )
1853       { LINE10[ LINE10len ] = 0;
1854         if( ( fp_inLINE = fopen( LINE10, "rb" ) ) == NULL )
1855         { printf( "Leprechaun: Can't open file %s \n", LINE10 ); return( 1 ); }
1856
1857         //fseek( fp_inLINE, 0L, SEEK_END ); //Rev. 12
1858         //size_inLINE = ftell( fp_inLINE ); //Rev. 12
1859         //fseek( fp_inLINE, 0L, SEEK_SET ); //Rev. 12
1860
1861         #if defined(_WIN32_ENVIRONMENT_)
1862         // 64bit:
1863         _lseeki64( fileno(fp_inLINE), 0L, SEEK_END );
1864         size_inLINESIXFOUR = _telli64( fileno(fp_inLINE) );
1865         _lseeki64( fileno(fp_inLINE), 0L, SEEK_SET );
1866         #else
1867         // 64bit:
1868         fseeko( fp_inLINE, 0L, SEEK_END );
1869         size_inLINESIXFOUR = ftello( fp_inLINE );
1870         fseeko( fp_inLINE, 0L, SEEK_SET );
1871         #endif /* defined(_WIN32_ENVIRONMENT_) */
1872
1873         printf( "Size of Input TEXTual file: %s\n", _ui64toaKAZEcomma(size_inLINESIXFOUR, l1ToaDigits, 10) );
1874         FilesLEN = FilesLEN + size_inLINESIXFOUR;
1875         NumberOfFiles++;
1876
1877         //~~~~~
1878         wrdlen = 0;
1879         for( i = 0; i < size_inLINESIXFOUR; i++ )
1880         {
1881             // ~~~~~ Buffering fread [
1882             if( workKoffset == -1 ) {
1883                 if ( i + 1024*96 < size_inLINESIXFOUR ) {
1884                     fread( &workk[0], 1, 1024*96, fp_inLINE );
1885                     workKoffset = 0;
1886                     workbyte = workk[workKoffset];
1887                 }
1888             }
1889             else {
1890                 fread( &workbyte, 1, 1, fp_inLINE );
1891             }
1892             else {
1893                 workKoffset++;
1894                 workbyte = workk[workKoffset];
1895                 if (workKoffset == 1024*96 - 1) workKoffset = -1;
1896             }
1897             // ~~~~~ Buffering fread ]
1898
1899             if( isalpha( workbyte ) )
1900             {
1901                 if( wrdlen < 31 )
1902                 { wrd[ wrdlen ] = tolower( workbyte ); }
1903                 wrdlen++;
1904             }
1905             if ( workbyte < 'A' ) // Most characters are uneder alphabet - only one if
1906             {
1907                 // This fragment is MIRRORed: #1 copy [
1908                 if( workbyte == 10 ) {NumberOfLines++;}
1909                 if( 1 <= wrdlen && wrdlen <= 31 )
1910                 {
1911                     wrd[ wrdlen ] = 0;
1912                     // OTKACHAM: 1<<17-1 gives 65536 i.e. '-' have had high priority than '<<'
1913                     //Next line gives error due to mix of '&' and 'double'
1914                     if ( ( ++WORDcount & ((1<<20)-1) ) == 0 )
1915                     { // _ui64toaKAZEzerocomma(WORDcount, l1ToaDigits, 10);
1916                       //printf( "word count: %s(%lu/128 done)\r", l1ToaDigits, ((long long)i*100) / size_inLINESIXFOUR );
1917                     }
1918                     ++Melnitcka;
1919                     Melnitcka = Melnitcka % 4;
1920                     if (Melnitcka == 0) { printf( "|; word count: %s of them %s distinct; Done: %lu/64\r", _ui64toaKAZEcomma(WORDcount, l1ToaDigits, 10),

```

```

        _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, l1ToaDigits2, 10), ((long long)i<<6) / size_inLINESIXFOUR ); }
1920 if (MeInitchka == 1) { printf( "\n"; word count: %s of them %s distinct; Done: %lu/64r", _ui64toaKAZEcomma(WORDcount, l1ToaDigits, 10),
        _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, l1ToaDigits2, 10), ((long long)i<<6) / size_inLINESIXFOUR ); }
1921 if (MeInitchka == 2) { printf( "\n"; word count: %s of them %s distinct; Done: %lu/64r", _ui64toaKAZEcomma(WORDcount, l1ToaDigits, 10),
        _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, l1ToaDigits2, 10), ((long long)i<<6) / size_inLINESIXFOUR ); }
1922 if (MeInitchka == 3) { printf( "\n"; word count: %s of them %s distinct; Done: %lu/64r", _ui64toaKAZEcomma(WORDcount, l1ToaDigits, 10),
        _ui64toaKAZEcomma((unsigned long long)WORDcountDistinct, l1ToaDigits2, 10), ((long long)i<<6) / size_inLINESIXFOUR ); }
1923 }
1924 LetterOffset = (int)( wrd[0] - 'a' ) * 31 + (wrdlen-1); // 0..805
1925 //BufStart = pointerflush + LetterOffset * LetterBuffer; // OLD
1926
1927 BufStart = pointerflush + (int)( wrd[0] - 'a' ) * WHOLELetter_BufferSize + OffsetsInBuffer[wrdlen-1];
1928 // Above line and Below line are equal
1929 //BufStart = pointerflush + (LetterOffset / 31) * WHOLELetter_BufferSize + OffsetsInBuffer[LetterOffset % 31];
1930
1931 //Slot = KuxHash3plus(wrd)<<2; //13++
1932 //Slot = FNV1A_Hash_SHIFTless_XORless(wrd)<<2; //13+++
1933 //Slot = FNV1A_Hash_4_OCTETS(wrd, wrdlen>>2)<<2; //13++++
1934 //Slot = FNV1A_Hash_4_OCTETS_31(wrd, wrdlen>>2)<<2; //13+++++
1935 /*
1936 if (wrdlen<=19) // 4x4+3=19
1937     Slot = FNV1A_Hash_4_OCTETS(wrd, wrdlen>>2)<<2; //13++++
1938 else
1939     Slot = 2x8+4=20 i.e. first contains 5 clashes
1940
1941     Slot = FNV1A_Hash_8_OCTETS(wrd, wrdlen>>3)<<2; //13+++++
1942 */
1943 if (wrdlen<=19) // 4x4+3=19 i.e. last contains 7 clashes
1944     Slot = FNV1A_Hash_Granularity(wrd, wrdlen>>2, 2)<<2; //13+++++
1945 else
1946     Slot = 2x8+4=20 i.e. first contains 6 clashes
1947
1948     Slot = FNV1A_Hash_Granularity(wrd, wrdlen>>3, 3)<<2; //13+++++
1949
1950 memcpy( &PseudolinkedPointer, BufStart+Slot, 4 );
1951
1952 //; Line 917
1953 // mov     edx, DWORD PTR [eax+ebp]
1954 // add     esp, 4
1955 // ?? DANGEROUS: above and below lines are(must:long must be 4bytes) identical
1956 //PseudolinkedPointer = (unsigned long)*(long *)(&BufStart+Slot);
1957
1958 //; Line 919
1959 // mov     edx, DWORD PTR [eax+ebp]
1960 // add     esp, 4
1961 // while (count--) {
1962 //     *(char *)dst = *(char *)src;
1963 //     dst = (char *)dst + 1;
1964 //     src = (char *)src + 1;
1965 // }
1966 if (BStor8tree != 1)
1967 {
1968     if (PseudolinkedPointer == 0) // means EMPTY-SLOT
1969     {
1970         //if( (unsigned long)(bufend[LetterOffset] - BufStart) + wrdlen + 4 < (GRMBLh11[(int)wrdlen] * LetterBuffer)/31 ) // OLD slower
1971         //if( (unsigned long)(bufend[LetterOffset] - BufStart) + wrdlen + 4 + 4 < GRMBLFoolAgain[(int)wrdlen] ) // +4 more for BST instead
1972         of LL
1973         {
1974             memcpy( BufStart+Slot, &bufend[LetterOffset], 4 );
1975         }
1976     }
1977     //; Line 932
1978     // mov     DWORD PTR [eax+ebp], esi
1979     // ?? DANGEROUS: above and below lines are(must:long must be 4bytes) identical
1980     //*(long *)(&BufStart+Slot) = *(long *)&bufend[LetterOffset];
1981
1982     //; Line 936
1983     // mov     DWORD PTR [eax+ebp], esi
1984     // Below 3 lines are commented due to experiment below malloc which shows that allocated memory is ZEROed.
1985     //memcpy( bufend[LetterOffset], &BufStart[NumberOfSLOTS*4], 4 ); // means next exists not: Means PseudolinkedPointerL =
1986     0
1987     //; Line 940
1988     // mov     ecx, DWORD PTR [ebp+32768]
1989     // ?? DANGEROUS: above and below lines are(must:long must be 4bytes) identical
1990     //*(long *)bufend[LetterOffset] = *(long *)&BufStart[NumberOfSLOTS*4];
1991
1992     //; Line 944
1993     // mov     ecx, DWORD PTR [ebp+32768]
1994     //bufend[LetterOffset] = bufend[LetterOffset] + 4;
1995     //memcpy( bufend[LetterOffset], &BufStart[NumberOfSLOTS*4], 4 ); // means next exists not: Means PseudolinkedPointerR =
1996     0
1997     bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4; // + 4 due to above commenting
1998     memcpy( bufend[LetterOffset], wrd, wrdlen ); WORDcountDistinct++; bufNumberOfWords[LetterOffset]++;
1999     //bufNowps[LetterOffset][Slot]++; // ?? crashes
2000     bufend[LetterOffset] = bufend[LetterOffset] + wrdlen;
2001     if (MAXusedBuffer[wrdlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrdlen] = (unsigned
2002     long)(bufend[LetterOffset] - BufStart);}
2003 }
2004 else
2005 { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n" );
2006 }
2007 printf( fp_outLOG, "Input File with a list of TEXTual Files: %s\n", argv[1] );
2008 printf( fp_outLOG, "Size of all TEXTual Files: %s\n", _ui64toaKAZEcomma(FilesLEN, l1ToaDigits, 10) );
2009 printf( fp_outLOG, "word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, l1ToaDigits, 10), _ui64toaKAZEcomma((unsigned long
2010 long)WORDcountDistinct, l1ToaDigits2, 10) );
2011 printf( fp_outLOG, "Number of Files: %lu\n", NumberOfFiles );
2012 printf( fp_outLOG, "Number of Lines: %lu\n", NumberOfLines );
2013 printf( fp_outLOG, "Allocated memory in MB: %lu\n", (unsigned long)(memory_size>>20)+1 );
2014 printf( fp_outLOG, "Total Attempts to Find/Put WORDS into Binary-Search-Trees: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, l1ToaDigits,
2015 10) );
2016 for( k = 1; k < 32; k++ )
2017 { printf( fp_outLOG, "Words with length %s occupy %sKB of %sKB given i.e. %s% utilization\n", _ui64toaKAZEzerocomma(k, l1ToaDigits, 10)+(26-
2018 2), _ui64toaKAZEzerocomma((MAXusedBuffer[k]>>10)+1, l1ToaDigits2, 10)+(26-5), _ui64toaKAZEzerocomma(((GRMBLh11[(int)k] *
2019 LetterBuffer)/31)>>10)+1, l1ToaDigits3, 10)+(26-5), _ui64toaKAZEzerocomma((unsigned long long)(MAXusedBuffer[k]*100)/((GRMBLh11[(int)k] *
2020 LetterBuffer)/31), l1ToaDigits4, 10)+(26-2), "%0" ); // 26 are all 26-DESIRED=24
2021 }
2022 printf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
2023 printf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n" );
2024 return( 1 );
2025 }
2026 }
2027 else // means USED-SLOT
2028 { FoundInLinkedList = 0;
2029 while (PseudolinkedPointer != 0 && FoundInLinkedList == 0)
2030 {
2031     if (memcmp(PseudolinkedPointer+44,wrd,wrdlen) == 0)
2032     while ( --count && *(char *)buf1 == *(char *)buf2 ) {
2033         buf1 = (char *)buf1 + 1;
2034         buf2 = (char *)buf2 + 1;
2035     }
2036     return( *((unsigned char *)buf1) - *((unsigned char *)buf2) );
2037 }
2038 FoundInLinkedList = 1;
2039 else // i.e. < or >
2040 {
2041     if (memcmp(PseudolinkedPointer+44,wrd,wrdlen) > 0)
2042     memcpy( &PseudolinkedPointerNEW, PseudolinkedPointer, 4 );
2043     else
2044     {
2045         PseudolinkedPointer = PseudolinkedPointer + 4;
2046         memcpy( &PseudolinkedPointerNEW, PseudolinkedPointer, 4 );
2047     }
2048     if (PseudolinkedPointerNEW == 0)
2049     {
2050         //if( (unsigned long)(bufend[LetterOffset] - BufStart) + wrdlen + 4 < (GRMBLh11[(int)wrdlen] * LetterBuffer)/31 ) // OLD slower
2051         if( (unsigned long)(bufend[LetterOffset] - BufStart) + wrdlen + 4 + 4 < GRMBLFoolAgain[(int)wrdlen] )
2052         { memcpy( PseudolinkedPointer, &bufend[LetterOffset], 4 );
2053         // Below 3 lines are commented due to experiment below malloc which shows that allocated memory is ZEROed.
2054         //memcpy( bufend[LetterOffset], &BufStart[NumberOfSLOTS*4], 4 ); // means next exists not
2055         //bufend[LetterOffset] = bufend[LetterOffset] + 4;
2056         //memcpy( bufend[LetterOffset], &BufStart[NumberOfSLOTS*4], 4 ); // means next exists not
2057         bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4; // + 4 due to above commenting
2058         memcpy( bufend[LetterOffset], wrd, wrdlen ); WORDcountDistinct++; bufNumberOfWords[LetterOffset]++;
2059         //bufNowps[LetterOffset][Slot]++; // ?? crashes
2060         bufend[LetterOffset] = bufend[LetterOffset] + wrdlen;
2061         if (MAXusedBuffer[wrdlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrdlen] = (unsigned
2062         long)(bufend[LetterOffset] - BufStart);}
2063     }
2064     else
2065     { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n" );
2066     }
2067 }
2068 printf( fp_outLOG, "Input File with a list of TEXTual Files: %s\n", argv[1] );
2069 printf( fp_outLOG, "Size of all TEXTual Files: %s\n", _ui64toaKAZEcomma(FilesLEN, l1ToaDigits, 10) );
2070 printf( fp_outLOG, "word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, l1ToaDigits, 10), _ui64toaKAZEcomma((unsigned long
2071 long)WORDcountDistinct, l1ToaDigits2, 10) );
2072 printf( fp_outLOG, "Number of Files: %lu\n", NumberOfFiles );
2073 printf( fp_outLOG, "Number of Lines: %lu\n", NumberOfLines );
2074 printf( fp_outLOG, "Allocated memory in MB: %lu\n", (unsigned long)(memory_size>>20)+1 );
2075 printf( fp_outLOG, "Total Attempts to Find/Put WORDS into Binary-Search-Trees: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, l1ToaDigits,
2076 10) );
2077 for( k = 1; k < 32; k++ )
2078 { printf( fp_outLOG, "Words with length %s occupy %sKB of %sKB given i.e. %s% utilization\n", _ui64toaKAZEzerocomma(k, l1ToaDigits, 10)+(26-
2079 2), _ui64toaKAZEzerocomma((MAXusedBuffer[k]>>10)+1, l1ToaDigits2, 10)+(26-5), _ui64toaKAZEzerocomma(((GRMBLh11[(int)k] *
2080 LetterBuffer)/31)>>10)+1, l1ToaDigits3, 10)+(26-5), _ui64toaKAZEzerocomma((unsigned long long)(MAXusedBuffer[k]*100)/((GRMBLh11[(int)k] *
2081 LetterBuffer)/31), l1ToaDigits4, 10)+(26-2), "%0" ); // 26 are all 26-DESIRED=24
2082 }
2083 printf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
2084 printf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n" );
2085 return( 1 );
2086 }
2087 }
2088 PseudolinkedPointer = PseudolinkedPointerNEW;
2089 }
2090 WORDcountAttemptsToPut++;
2091 } // while
2092 }
2093 // BST fragment ]
2094 }
2095 // B-tree order 3 fragment [
2096 // LEAF structure: [LeftPointer][MiddlePointer][RightPointer][Leftword][Rightword]
2097 // 4bytes 4bytes 4bytes wrdlen wrdlen
2098 // ALL B-tree order 3 fragment consists of 3 sub-Fragments:
2099 }

```

```

1999 printf( fp_outLOG, "Allocated memory in MB: %lu\n", (unsigned long)(memory_size>>20)+1 );
2000 printf( fp_outLOG, "Total Attempts to Find/Put WORDS into Binary-Search-Trees: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, l1ToaDigits,
2001 10) );
2002 for( k = 1; k < 32; k++ )
2003 { printf( fp_outLOG, "Words with length %s occupy %sKB of %sKB given i.e. %s% utilization\n", _ui64toaKAZEzerocomma(k, l1ToaDigits, 10)+(26-
2004 2), _ui64toaKAZEzerocomma((MAXusedBuffer[k]>>10)+1, l1ToaDigits2, 10)+(26-5), _ui64toaKAZEzerocomma(((GRMBLh11[(int)k] *
2005 LetterBuffer)/31)>>10)+1, l1ToaDigits3, 10)+(26-5), _ui64toaKAZEzerocomma((unsigned long long)(MAXusedBuffer[k]*100)/((GRMBLh11[(int)k] *
2006 LetterBuffer)/31), l1ToaDigits4, 10)+(26-2), "%0" ); // 26 are all 26-DESIRED=24
2007 }
2008 printf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
2009 printf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n" );
2010 return( 1 );
2011 }
2012 }
2013 else // means USED-SLOT
2014 { FoundInLinkedList = 0;
2015 while (PseudolinkedPointer != 0 && FoundInLinkedList == 0)
2016 {
2017     if (memcmp(PseudolinkedPointer+44,wrd,wrdlen) == 0)
2018     while ( --count && *(char *)buf1 == *(char *)buf2 ) {
2019         buf1 = (char *)buf1 + 1;
2020         buf2 = (char *)buf2 + 1;
2021     }
2022     return( *((unsigned char *)buf1) - *((unsigned char *)buf2) );
2023 }
2024 FoundInLinkedList = 1;
2025 else // i.e. < or >
2026 {
2027     if (memcmp(PseudolinkedPointer+44,wrd,wrdlen) > 0)
2028     memcpy( &PseudolinkedPointerNEW, PseudolinkedPointer, 4 );
2029     else
2030     {
2031         PseudolinkedPointer = PseudolinkedPointer + 4;
2032         memcpy( &PseudolinkedPointerNEW, PseudolinkedPointer, 4 );
2033     }
2034     if (PseudolinkedPointerNEW == 0)
2035     {
2036         //if( (unsigned long)(bufend[LetterOffset] - BufStart) + wrdlen + 4 < (GRMBLh11[(int)wrdlen] * LetterBuffer)/31 ) // OLD slower
2037         if( (unsigned long)(bufend[LetterOffset] - BufStart) + wrdlen + 4 + 4 < GRMBLFoolAgain[(int)wrdlen] )
2038         { memcpy( PseudolinkedPointer, &bufend[LetterOffset], 4 );
2039         // Below 3 lines are commented due to experiment below malloc which shows that allocated memory is ZEROed.
2040         //memcpy( bufend[LetterOffset], &BufStart[NumberOfSLOTS*4], 4 ); // means next exists not
2041         //bufend[LetterOffset] = bufend[LetterOffset] + 4;
2042         //memcpy( bufend[LetterOffset], &BufStart[NumberOfSLOTS*4], 4 ); // means next exists not
2043         bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4; // + 4 due to above commenting
2044         memcpy( bufend[LetterOffset], wrd, wrdlen ); WORDcountDistinct++; bufNumberOfWords[LetterOffset]++;
2045         //bufNowps[LetterOffset][Slot]++; // ?? crashes
2046         bufend[LetterOffset] = bufend[LetterOffset] + wrdlen;
2047         if (MAXusedBuffer[wrdlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrdlen] = (unsigned
2048         long)(bufend[LetterOffset] - BufStart);}
2049     }
2050     else
2051     { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n" );
2052     }
2053 }
2054 printf( fp_outLOG, "Input File with a list of TEXTual Files: %s\n", argv[1] );
2055 printf( fp_outLOG, "Size of all TEXTual Files: %s\n", _ui64toaKAZEcomma(FilesLEN, l1ToaDigits, 10) );
2056 printf( fp_outLOG, "word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, l1ToaDigits, 10), _ui64toaKAZEcomma((unsigned long
2057 long)WORDcountDistinct, l1ToaDigits2, 10) );
2058 printf( fp_outLOG, "Number of Files: %lu\n", NumberOfFiles );
2059 printf( fp_outLOG, "Number of Lines: %lu\n", NumberOfLines );
2060 printf( fp_outLOG, "Allocated memory in MB: %lu\n", (unsigned long)(memory_size>>20)+1 );
2061 printf( fp_outLOG, "Total Attempts to Find/Put WORDS into Binary-Search-Trees: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, l1ToaDigits,
2062 10) );
2063 for( k = 1; k < 32; k++ )
2064 { printf( fp_outLOG, "Words with length %s occupy %sKB of %sKB given i.e. %s% utilization\n", _ui64toaKAZEzerocomma(k, l1ToaDigits, 10)+(26-
2065 2), _ui64toaKAZEzerocomma((MAXusedBuffer[k]>>10)+1, l1ToaDigits2, 10)+(26-5), _ui64toaKAZEzerocomma(((GRMBLh11[(int)k] *
2066 LetterBuffer)/31)>>10)+1, l1ToaDigits3, 10)+(26-5), _ui64toaKAZEzerocomma((unsigned long long)(MAXusedBuffer[k]*100)/((GRMBLh11[(int)k] *
2067 LetterBuffer)/31), l1ToaDigits4, 10)+(26-2), "%0" ); // 26 are all 26-DESIRED=24
2068 }
2069 printf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
2070 printf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n" );
2071 return( 1 );
2072 }
2073 }
2074 PseudolinkedPointer = PseudolinkedPointerNEW;
2075 }
2076 WORDcountAttemptsToPut++;
2077 } // while
2078 }
2079 // BST fragment ]
2080 }
2081 // B-tree order 3 fragment [
2082 // LEAF structure: [LeftPointer][MiddlePointer][RightPointer][Leftword][Rightword]
2083 // 4bytes 4bytes 4bytes wrdlen wrdlen
2084 // ALL B-tree order 3 fragment consists of 3 sub-Fragments:
2085 }

```

```

2077 // 1] Search 2] if Search failed Trasirascht(pushing in stack PseudolinkedPointer(visited LEAFs)) Search 3] Insert Iterative
2078
2079 // 1] Search [ _____1407 line in C - see below: whole Search in assembler_____
2080         { if (PseudolinkedPointer == 0) // means EMPTY-SLOT
2081         {
2082             if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wrklen + 4 + 4 + 4 < GRMBLFoolAgain[(int)wrklen] ) // +4 more for BST
instead of LL; + more(see LEAF)
2083             {
2084                 memcpy( BufStart+Slot, &bufend[LetterOffset], 4 );
2085                 bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
2086                 memcpy( bufend[LetterOffset], wrd, wrklen ); WORDcountDistinct++; bufNumberOfWords[LetterOffset]++;
2087                 bufend[LetterOffset] = bufend[LetterOffset] + 2*wrklen;
2088                 if (MAXusedBuffer[wrklen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wrklen] = (unsigned
long)(bufend[LetterOffset] - BufStart);}
2089             }
2090             else
2091             { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n" );
2092             fprintf( fp_outLOG, "Input File with a list of TEXTual Files: %s\n", argv[1] );
2093             fprintf( fp_outLOG, "Size of all TEXTual Files: %s\n", _ui64toaKAZEcomma(FilesLEN, 11toADigits, 10) );
2094             fprintf( fp_outLOG, "word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, 11toADigits, 10), _ui64toaKAZEcomma((unsigned long
long)WORDcountDistinct, 11toADigits2, 10) );
2095             fprintf( fp_outLOG, "Number of Files: %lu\n", NumberOfFiles );
2096             fprintf( fp_outLOG, "Number of Lines: %lu\n", NumberOfLines );
2097             fprintf( fp_outLOG, "Allocated memory in MB: %lu\n", (unsigned long)(memory_size>20)+1 );
2098             fprintf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11toADigits, 10)
);
2099             for( k = 1; k < 32; k++ )
2100             { fprintf( fp_outLOG, "words with length %s occupy %sKb of %sKb given i.e. %s% utilization\n", _ui64toaKAZEzerocomma(k, 11toADigits, 10)+(26-
2), _ui64toaKAZEzerocomma((MAXusedBuffer[k]>10)+1, 11toADigits2, 10)+(26-5), _ui64toaKAZEzerocomma(((GRMBLhill[(int)k] *
LetterBuffer)/31)>10)+1, 11toADigits3, 10)+(26-5), _ui64toaKAZEzerocomma((unsigned long long)(MAXusedBuffer[k]*100)/((GRMBLhill[(int)k] *
LetterBuffer)/31), 11toADigits4, 10)+(26-2), "%0" ); // 26 are all 26-DESIRED=24
2101             }
2102             fprintf( fp_outLOG, "used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
2103             fprintf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n");
2104             return( 1 );
2105         }
2106         FoundInLinkedList = 1;
2107     }
2108     else // means USED-SLOT
2109     { FoundInLinkedList = 0;
2110       while (PseudolinkedPointer != 0 && FoundInLinkedList == 0)
2111       {
2112           // ***** 'P W P' section [
2113           // LW: existence check if ( *(char *) (PseudolinkedPointer+4+4+4) != 0 )
2114           // RW: existence check if ( *(char *) (PseudolinkedPointer+4+4+4+wrklen) != 0 )
2115           // here ALWAYS LW exists: no need for existence check - line below
2116           // if ( *(char *) (PseudolinkedPointer+4+4+4) != 0 )
2117           if (memcmp(PseudolinkedPointer+4+4+4, wrd, wrklen) > 0) // go LP
2118           { memcpy( &PseudolinkedPointerNEW, PseudolinkedPointer + 0, 4 ); //LP
2119             PseudolinkedPointer = PseudolinkedPointerNEW;
2120           }
2121           else if (memcmp(PseudolinkedPointer+4+4+4, wrd, wrklen) < 0) // go RP or MP
2122           { // RW existence check - line below:
2123             if ( *(char *) (PseudolinkedPointer+4+4+4+wrklen) != 0 ) // RW exists
2124             { // Here all 'P W P' section is repeated; the way of handling case when dynamic number of words in leaf
2125             // ++++++
2126             // ***** 'P W P' section 2 [
2127             // LW: existence check if ( *(char *) (PseudolinkedPointer+4+4+4) != 0 )
2128             // RW: existence check if ( *(char *) (PseudolinkedPointer+4+4+4+wrklen) != 0 )
2129             // here ALWAYS RW exists: no need for existence check - line below
2130             // if ( *(char *) (PseudolinkedPointer+4+4+4+wrklen) != 0 )
2131             if (memcmp(PseudolinkedPointer+4+4+4+wrklen, wrd, wrklen) > 0) // go MP
2132             { memcpy( &PseudolinkedPointerNEW, PseudolinkedPointer + 4, 4 ); //MP
2133               PseudolinkedPointer = PseudolinkedPointerNEW;
2134             }
2135             else if (memcmp(PseudolinkedPointer+4+4+4+wrklen, wrd, wrklen) < 0) // go RP
2136             { // No 2w after RW - go RP
2137               memcpy( &PseudolinkedPointerNEW, PseudolinkedPointer + 4 + 4, 4 ); //RP
2138               PseudolinkedPointer = PseudolinkedPointerNEW;
2139             }
2140             else FoundInLinkedList = 1; // wrd is RW
2141             WORDcountAttemptsToPut++;
2142             // ***** 'P W P' section 2 ]
2143             // ++++++
2144             }
2145             else // RW empty - go MP
2146             { memcpy( &PseudolinkedPointerNEW, PseudolinkedPointer + 4, 4 ); //MP
2147               PseudolinkedPointer = PseudolinkedPointerNEW;
2148             }
2149             }
2150             else FoundInLinkedList = 1; // wrd is LW
2151             WORDcountAttemptsToPut++;
2152             // ***** 'P W P' section ]
2153             } // while
2154             WORDcountAttemptsToPut--; // - 1 due to BST way of counting i.e. direct hash hit is not counted only successors
2155         }
2156         // 1] Search ] _____1484 line in C - see below: whole Search in assembler_____
2157

```

```

2158 /*
2159 ; Line 1397
2160 jmp $L2139
2161 $L2042:
2162 ; Line 1408
2163 test edx, edx
2164 jne SHORT $L2110
2165 ; Line 1410
2166 mov ecx, DWORD PTR _bufends[esp+esi*4+892340]
2167 mov edi, DWORD PTR _GRMBLFoolAgain[esp+ebx*4+892340]
2168 lea edx, DWORD PTR _bufends[esp+esi*4+892340]
2169 lea esi, DWORD PTR [ebx+ebx+12]
2170 sub esi, ebp
2171 add esi, ecx
2172 cmp esi, edi
2173 mov DWORD PTR tv4122[esp+892340], edx
2174 jae $L2113
2175 ; Line 1412
2176 mov DWORD PTR [eax+ebp], ecx
2177 ; Line 1413
2178 lea eax, DWORD PTR [ecx+12]
2179 ; Line 1436
2180 jmp $L2749
2181 $L2110:
2182 ; Line 1437
2183 mov DWORD PTR _FoundInLinkedLists[esp+892340], 0
2184 npad11
2185 $L2141:
2186 ; Line 1438
2187 mov eax, DWORD PTR _FoundInLinkedLists[esp+892340]
2188 test eax, eax
2189 jne $L2142
2190 ; Line 1445
2191 lea ebp, DWORD PTR [edx+12]
2192 mov ecx, ebx
2193 lea edi, DWORD PTR _wrd$[esp+892340]
2194 mov esi, ebp
2195 xor eax, eax
2196 repe cmpsb
2197 je SHORT $L2682
2198 sbb eax, eax
2199 sbb eax, -1
2200 $L2682:
2201 test eax, eax
2202 jle SHORT $L2143
2203 ; Line 1447
2204 mov edx, DWORD PTR [edx]
2205 ; Line 1449
2206 jmp $L2153
2207 $L2143:
2208 mov ecx, ebx
2209 lea edi, DWORD PTR _wrd$[esp+892340]
2210 mov esi, ebp
2211 xor eax, eax
2212 repe cmpsb
2213 je SHORT $L2640
2214 sbb eax, eax
2215 sbb eax, -1
2216 $L2640:
2217 test eax, eax
2218 jge SHORT $L2145
2219 ; Line 1451
2220 mov cl, BYTE PTR [edx+ebx+12]
2221 test cl, cl
2222 lea eax, DWORD PTR [edx+ebx+12]
2223 je SHORT $L2147
2224 ; Line 1459
2225 mov ecx, ebx
2226 lea edi, DWORD PTR _wrd$[esp+892340]
2227 mov esi, eax
2228 xor ebp, ebp
2229 repe cmpsb
2230 je SHORT $L2695
2231 sbb ebp, ebp
2232 sbb ebp, -1
2233 $L2695:
2234 test ebp, ebp
2235 jle SHORT $L2148
2236 ; Line 1461
2237 mov edx, DWORD PTR [edx+4]
2238 ; Line 1463
2239 jmp SHORT $L2151
2240 $L2148:
2241 mov esi, eax
2242 mov ecx, ebx
2243 lea edi, DWORD PTR _wrd$[esp+892340]
2244 xor eax, eax
2245 repe cmpsb

```

```

2246 je SHORT $L2642
2247 sbb eax, eax
2248 sbb eax, -1
2249 $L2642:
2250 test eax, eax
2251 jge SHORT $L2150
2252 ; Line 1466
2253 mov edx, DWORD PTR [edx+8]
2254 ; Line 1468
2255 jmp SHORT $L2151
2256 $L2150:
2257 mov DWORD PTR _FoundInLinkedList$[esp+892340], 1
2258 $L2151:
2259 ; Line 1469
2260 mov ecx, DWORD PTR _WORDcountAttemptsToPut$[esp+892340]
2261 mov eax, DWORD PTR _WORDcountAttemptsToPut$[esp+892344]
2262 add ecx, 1
2263 adc eax, 0
2264 mov DWORD PTR _WORDcountAttemptsToPut$[esp+892340], ecx
2265 mov DWORD PTR _WORDcountAttemptsToPut$[esp+892344], eax
2266 ; Line 1473
2267 jmp SHORT $L2153
2268 $L2147:
2269 ; Line 1475
2270 mov edx, DWORD PTR [edx+4]
2271 ; Line 1478
2272 jmp SHORT $L2153
2273 $L2145:
2274 mov DWORD PTR _FoundInLinkedList$[esp+892340], 1
2275 $L2153:
2276 ; Line 1479
2277 mov esi, DWORD PTR _WORDcountAttemptsToPut$[esp+892340]
2278 mov ecx, DWORD PTR _WORDcountAttemptsToPut$[esp+892344]
2279 add esi, 1
2280 adc ecx, 0
2281 testdx, edx
2282 mov DWORD PTR _WORDcountAttemptsToPut$[esp+892340], esi
2283 mov DWORD PTR _WORDcountAttemptsToPut$[esp+892344], ecx
2284 jne $L2141
2285 $L2142:
2286 ; Line 1482
2287 mov edx, DWORD PTR _WORDcountAttemptsToPut$[esp+892340]
2288 mov ecx, DWORD PTR _WORDcountAttemptsToPut$[esp+892344]
2289 or eax, -1
2290 add edx, eax
2291 adc ecx, eax
2292 mov DWORD PTR _WORDcountAttemptsToPut$[esp+892344], ecx
2293 mov DWORD PTR _WORDcountAttemptsToPut$[esp+892340], edx
2294 $L2139:
2295 */
2296
2297 if (FoundInLinkedList == 0)
2298 {
2299 // 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search [
2300 // 'TracingSearch' is the same as 'Search' except that adds the trail in my simulated stack,
2301 // the goal is not to waste time in 'Search' by dealing with no needed trail in case of not 'Insert'.
2302 // simulated stack contains pairs of 'Address of ParentLEAF' + 'Offset of ParentPointer in ParentLEAF i.e. 0 for LP, 4 for MP, 8 for RP'.
2303 // 'Offset ...' saves unnecessary comparisons of NEWword which after splitting goes up.
2304 memcpy( &PseudoLinkedPointer, BufStart+slot, 4 );
2305 StackPtr = 0;
2306 while (PseudoLinkedPointer != 0)
2307 {
2308 // ***** 'p w p' section [
2309 // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
2310 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
2311 // here ALWAYS LW exists: no need for existence check - line below
2312 // if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
2313 if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wrdlen) > 0) // go LP
2314 { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 0, 4 ); // LP
2315 if (StackPtr > 65536*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 ); }
2316 BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
2317 BSTstack[StackPtr] = 0; ++StackPtr; //LPoffset=0;MPoffset=4;RPoffset=8;
2318 PseudoLinkedPointer = PseudoLinkedPointerNEW;
2319 }
2320 else if (memcmp(PseudoLinkedPointer+4+4+4, wrd, wrdlen) < 0) // go RP or MP
2321 { // RW existence check - line below:
2322 if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 ) // RW exists
2323 { // Here all 'P W P' section is repeated; the way of handling case when dynamic number of words in leaf
2324 // ***** 'p w p' section [
2325 // ***** 'p w p' section 2 [
2326 // LW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4) != 0 )
2327 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
2328 // here ALWAYS RW exists: no need for existence check - line below
2329 // if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
2330 if (memcmp(PseudoLinkedPointer+4+4+4+wordlen, wrd, wrdlen) > 0) // go MP
2331 { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
2332 if (StackPtr > 65536*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 ); }
2333 BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf

```

```

2334 BSTstack[StackPtr] = 4; ++StackPtr; //LPoffset=0;MPoffset=4;RPoffset=8;
2335 PseudoLinkedPointer = PseudoLinkedPointerNEW;
2336 }
2337 else if (memcmp(PseudoLinkedPointer+4+4+4+wordlen, wrd, wrdlen) < 0) // go RP
2338 { // No ?W after RW - go RP
2339 memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4 + 4, 4 ); //RP
2340 if (StackPtr > 65536*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 ); }
2341 BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
2342 BSTstack[StackPtr] = 8; ++StackPtr; //LPoffset=0;MPoffset=4;RPoffset=8;
2343 PseudoLinkedPointer = PseudoLinkedPointerNEW;
2344 }
2345 else FoundInLinkedList = 1; // wrd is RW
2346 // ***** 'p w p' section 2 ]
2347 // ***** 'p w p' section 2 ]
2348 // ***** 'p w p' section 2 ]
2349 }
2350 else // RW empty - go MP
2351 { memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer + 4, 4 ); //MP
2352 if (StackPtr > 65536*3-1-1) { printf( "\nLeprechaun: Failure! 'B-tree order 3' simulated stack overflow!\n" ); return( 13 ); }
2353 BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //pt to visited leaf
2354 BSTstack[StackPtr] = 4; ++StackPtr; //LPoffset=0;MPoffset=4;RPoffset=8;
2355 PseudoLinkedPointer = PseudoLinkedPointerNEW;
2356 }
2357 }
2358 else FoundInLinkedList = 1; // wrd is LW
2359 // ***** 'p w p' section ]
2360 // ***** 'p w p' section ]
2361 // ***** 'p w p' section ]
2362 // 2] if Search failed Trasirascht(pushing in stack PseudoLinkedPointer(visited LEAFs)) Search [
2363 // 3] Insert Iterative [
2364 // There are total 4 situations:
2365 // Case #1: Outer NODE(including ROOT) [ ][ ][ ][LW][ ] Split Occurs ---- 'wrdUP' (wrdlen bytes)
2366 // Case #2: Outer NODE(including ROOT) [ ][ ][ ][LW][RW] Split Occurs ---- & 'PseudoLinkedPointerNEW' (ptr to NEW LEAF)
2367 // Case #3: ROOT [LP][MP][ ][LW][ ] Split Occurs ---- ARE GOING UP
2368 // Case #4: Inner NODE(including ROOT) [LP][MP][RP][LW][RW] Split Occurs ----
2369 // There are total 2 situations for PARENT LEAF: <-----
2370 // Case #3: [LP][MP][ ][LW][ ]
2371 // Case #4: [LP][MP][RP][LW][RW] Split Occurs
2372
2373 // ~ First deal alonely with the OUTER NODE(LEAF) where Search stopped i.e Case #1 & Case #2:
2374 PoffsetInLEAF = BSTstack[--StackPtr];
2375 PseudoLinkedPointer = BSTstack[--StackPtr];
2376 // NOTE: ONE LEAF IS FULL ONLY WHEN LAST CELL FOR KEY(here RW) EXISTS!
2377 // RW: existence check if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 )
2378 if ( *(char *) (PseudoLinkedPointer+4+4+4+wordlen) != 0 ) // If LEAF is full: Case #2
2379 { SplitOccurred = 1; WORDcountDistinct++; BufNumberOfWords[LetterOffset]++;
2380 // ALlocate NEW LEAF:
2381 if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wordlen + 4 + 4 + 4 < GRMBLFoolAgain[(int)wordlen] ) // +4 more for BST
2382 instead of LL; + more(see LEAF)
2383 { memcpy( &PseudoLinkedPointerNEW, &bufend[LetterOffset], 4 );
2384 bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
2385 bufend[LetterOffset] = bufend[LetterOffset] + 2*wordlen;
2386 if (MAXusedBuffer[wordlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wordlen] = (unsigned
2387 long)(bufend[LetterOffset] - BufStart);}
2388 }
2389 else
2390 { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n" );
2391 printf( fp_outLOG, "Input File with a list of TEXTual Files: %s\n", argv[1] );
2392 printf( fp_outLOG, "Size of all TEXTual Files: %s\n", _ui64toaKAZEcomma(FilesLEM, l1ToaDigits, 10) );
2393 printf( fp_outLOG, "word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, l1ToaDigits, 10), _ui64toaKAZEcomma((unsigned long
2394 long)WORDcountDistinct, l1ToaDigits2, 10) );
2395 printf( fp_outLOG, "Number of Files: %lu\n", NumberOfFiles );
2396 printf( fp_outLOG, "Number of Lines: %lu\n", NumberOfLines );
2397 printf( fp_outLOG, "Allocated memory in MB: %lu\n", (unsigned long)(memory_size>20)+1 );
2398 printf( fp_outLOG, "Total Attempts to Find/Put WORDS into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, l1ToaDigits, 10) );
2399 }
2400 for( k = 1; k < 32; k++)
2401 { printf( fp_outLOG, "words with length %s occupy %sKB of %sKB given i.e. %s% utilization\n", _ui64toaKAZEzerocomma(k, l1ToaDigits, 10)+(26-
2402 2), _ui64toaKAZEzerocomma(MAXusedBuffer[k]>>10)+1, l1ToaDigits2, 10)+(26-5), _ui64toaKAZEzerocomma(((GRMBLh1l1[(int)k] *
2403 LetterBuffer)/31)>>10)+1, l1ToaDigits3, 10)+(26-5), _ui64toaKAZEzerocomma((unsigned long long)(MAXusedBuffer[k]*100)/((GRMBLh1l1[(int)k] *
2404 LetterBuffer)/31), l1ToaDigits4, 10)+(26-2), "%0" ); // 26 are all 26-DESIRED=24
2405 }
2406 printf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)Thunderwith );
2407 printf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n" );
2408 return( 1 );
2409 }
2410 if (PoffsetInLEAF == 0) // wrd < LW
2411 {
2412 memcpy( wrdUP, PseudoLinkedPointer+4+4+4, wrdlen ); // LW up
2413 memcpy( PseudoLinkedPointer+4+4+4, wrd, wrdlen ); // wrd go to OLD LEAF
2414 memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wordlen, wrdlen ); // RW go to NEW LEAF
2415 * (char *) (PseudoLinkedPointer+4+4+4+wordlen) = 0; // RW mark unused in OLD LEAF
2416 }
2417 if (PoffsetInLEAF == 4) // LW < wrd < RW
2418 {
2419 memcpy( wrdUP, wrd, wrdlen ); // wrd up
2420 memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4+wordlen, wrdlen ); // RW go to NEW LEAF

```

```

2415         *(char *)(&PseudoLinkedPointer+4+4+4*wordlen) = 0; // RW mark unused in OLD LEAF
2416     }
2417     if (PoffsetInLEAF == 8) // wrd > RW
2418     {
2419         memcpy( wrdUP, PseudoLinkedPointer+4+4+4*wordlen, wrdlen ); // RW up
2420         *(char *)(&PseudoLinkedPointer+4+4+4*wordlen) = 0; // RW mark unused in OLD LEAF
2421         memcpy( PseudoLinkedPointerNEW+4+4+4, wrd, wrdlen ); // wrd go to NEW LEAF
2422     }
2423 }
2424 else // If LEAF is not full: Case #1
2425 { SplitOccured = 0; WORDcountDistinct++; bufNumberOfWords[LetterOffset]++;
2426   if (PoffsetInLEAF == 0) // wrd < [LW] so [LW] -> [] [LW] -> [wrd][LW]
2427   {
2428       memcpy( PseudoLinkedPointer+4+4+4*wordlen, PseudoLinkedPointer+4+4+4, wrdlen );
2429       memcpy( PseudoLinkedPointer+4+4+4, wrd, wrdlen );
2430   }
2431   if (PoffsetInLEAF == 4) // wrd > [LW] so [LW] -> [LW][wrd]
2432   {
2433       memcpy( PseudoLinkedPointer+4+4+4*wordlen, wrd, wrdlen );
2434   }
2435 }
2436 }
2437 if (SplitOccured != 0)
2438 {
2439     // ~ Second deal with the INNER NODE(S) i.e Case #3 & Case #4:
2440     while (StackPtr != 0 || SplitOccured != 0)
2441     {
2442         // 'PseudoLinkedPointerNEW' is new LEAF to be inserted
2443         // 'wrdUP' is NEW word to be inserted
2444         if (StackPtr != 0)
2445         {
2446             PoffsetInLEAF = BSTstack[--StackPtr];
2447             PseudoLinkedPointer = BSTstack[--StackPtr];
2448             if ( *(char *)(&PseudoLinkedPointer+4+4+4*wordlen) != 0 ) // If LEAF is full: Case #4
2449             { SplitOccured = 1;
2450               memcpy( wrdUPold, wrdUP, wrdlen ); // LW up
2451               PseudoLinkedPointerNEWold = PseudoLinkedPointerNEW;
2452               // ALlocate NEW LEAF:
2453               if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wordlen + 4 + 4 + 4 < GRMBLFOolAgain((int)wordlen) ) // +4 more for BST
2454               instead of LL; + more(see LEAF)
2455               {
2456                   memcpy( &PseudoLinkedPointerNEW, &bufend[LetterOffset], 4 );
2457                   bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
2458                   bufend[LetterOffset] = bufend[LetterOffset] + 2*wordlen;
2459                   if (MAXusedBuffer[wordlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wordlen] = (unsigned
2460 long)(bufend[LetterOffset] - BufStart);}
2461               }
2462               else
2463               { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n" );
2464                 printf( fp_outLOG, "Input File with a list of TEXTual Files: %s\n", argv[1] );
2465                 printf( fp_outLOG, "Size of all TEXTual Files: %s\n", _ui64toaKAZEcomma(FilesLEM, l1ToaDigits, 10) );
2466                 printf( fp_outLOG, "word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, l1ToaDigits, 10), _ui64toaKAZEcomma((unsigned long
2467 long)WORDcountDistinct, l1ToaDigits2, 10) );
2468                 printf( fp_outLOG, "Number of Files: %lu\n", NumberOfFiles );
2469                 printf( fp_outLOG, "Number of Lines: %lu\n", NumberOfLines );
2470                 printf( fp_outLOG, "Allocated memory in MB: %lu\n", (unsigned long)(memory_size>20)+1 );
2471                 printf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, l1ToaDigits, 10)
2472 );
2473                 for( k = 1; k < 32; k++)
2474                 { printf( fp_outLOG, "words with length %s occupy %sKB of %sKB given i.e. %s% utilization\n", _ui64toaKAZEzerocomma(k, l1ToaDigits, 10)+(26-
2475 2), _ui64toaKAZEzerocomma((MAXusedBuffer[k]>10)+1, l1ToaDigits2, 10)+(26-5), _ui64toaKAZEzerocomma(((GRMBLh1l[(int)k] *
2476 letterBuffer)/31)>10)+1, l1ToaDigits3, 10)+(26-5), _ui64toaKAZEzerocomma((unsigned long long)(MAXusedBuffer[k]*100)/((GRMBLh1l[(int)k] *
2477 letterBuffer)/31), l1ToaDigits4, 10)+(26-2), "%0" ); // 26 are all 26-DESIRED=24
2478 }
2479                 printf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)thunderwith );
2480                 printf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n" );
2481                 return( 1 );
2482             }
2483             if (PoffsetInLEAF == 0) // wrdUPold < LW
2484             {
2485                 memcpy( wrdUP, PseudoLinkedPointer+4+4+4, wrdlen ); // LW up
2486                 memcpy( PseudoLinkedPointer+4+4+4, wrdUPold, wrdlen ); // wrdUPold go to OLD LEAF
2487                 memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4*wordlen, wrdlen ); // RW go to NEW LEAF
2488                 *(char *)(&PseudoLinkedPointer+4+4+4*wordlen) = 0; // RW mark unused in OLD LEAF
2489                 // [LP](PseudoLinkedPointerNEWold)[MP][RP](wrdUPold)[LW][RW] -----
2490                 // pair [LW] PseudoLinkedPointerNEW goes up
2491                 // PseudoLinkedPointer: PseudoLinkedPointerNEW:
2492                 // [LP](PseudoLinkedPointerNEWold)[] (wrdUPold) [MP][RP] [RW] <---
2493                 // no need to put zero in RP because logic is based on words existence:
2494                 memcpy( PseudoLinkedPointerNEW+0, PseudoLinkedPointer+4, 4 );
2495                 memcpy( PseudoLinkedPointerNEW+4, PseudoLinkedPointer+8, 4 );
2496                 memcpy( PseudoLinkedPointer+4, &PseudoLinkedPointerNEWold, 4 );
2497             }
2498             if (PoffsetInLEAF == 4) // LW < wrdUPold < RW
2499             {
2500                 memcpy( wrdUP, wrdUPold, wrdlen ); // wrdUPold up
2501                 memcpy( PseudoLinkedPointerNEW+4+4+4, PseudoLinkedPointer+4+4+4*wordlen, wrdlen ); // RW go to NEW LEAF
2502                 *(char *)(&PseudoLinkedPointer+4+4+4*wordlen) = 0; // RW mark unused in OLD LEAF

```

```

2496         // [LP][MP](PseudoLinkedPointerNEWold)[RP][LW](wrdUPold)[RW] -----
2497         // pair [wrdUPold] PseudoLinkedPointerNEW goes up
2498         // PseudoLinkedPointer: PseudoLinkedPointerNEW:
2499         // [LP][MP] [LW] (PseudoLinkedPointerNEWold)[RP] [RW] <---
2500         // no need to put zero in RP because logic is based on words existence:
2501         memcpy( PseudoLinkedPointerNEW+0, &PseudoLinkedPointerNEWold, 4 );
2502         memcpy( PseudoLinkedPointerNEW+4, PseudoLinkedPointer+8, 4 );
2503     }
2504     if (PoffsetInLEAF == 8) // wrdUPold > RW
2505     {
2506         memcpy( wrdUP, PseudoLinkedPointer+4+4+4*wordlen, wrdlen ); // RW up
2507         *(char *)(&PseudoLinkedPointer+4+4+4*wordlen) = 0; // RW mark unused in OLD LEAF
2508         memcpy( PseudoLinkedPointerNEW+4+4+4, wrdUPold, wrdlen ); // wrdUPold go to NEW LEAF
2509         // [LP][MP][RP](PseudoLinkedPointerNEWold)[LW][RW](wrdUPold) -----
2510         // pair [RW] PseudoLinkedPointerNEW goes up
2511         // PseudoLinkedPointer: PseudoLinkedPointerNEW:
2512         // [LP][MP] [LW] [RP](PseudoLinkedPointerNEWold)[] (wrdUPold) <---
2513         // no need to put zero in RP because logic is based on words existence:
2514         memcpy( PseudoLinkedPointerNEW+0, PseudoLinkedPointer+8, 4 );
2515         memcpy( PseudoLinkedPointerNEW+4, &PseudoLinkedPointerNEWold, 4 );
2516     }
2517 }
2518 else // If LEAF is not full: Case #3
2519 { SplitOccured = 0;
2520   if (PoffsetInLEAF == 0) // wrdUP < [LW] so [LW] -> [] [LW] -> [wrdUP][LW]
2521   {
2522       memcpy( PseudoLinkedPointer+4+4+4*wordlen, PseudoLinkedPointer+4+4+4, wrdlen );
2523       memcpy( PseudoLinkedPointer+4+4+4, wrdUP, wrdlen );
2524       // [LP][MP] [] -> [LP] [MP] -> [LP][MP][MP]
2525       memcpy( PseudoLinkedPointer+8, PseudoLinkedPointer+4, 4 );
2526       memcpy( PseudoLinkedPointer+4, &PseudoLinkedPointerNEW, 4 );
2527   }
2528   if (PoffsetInLEAF == 4) // wrdUP > [LW] so [LW] -> [LW][wrdUP]
2529   {
2530       memcpy( PseudoLinkedPointer+4+4+4*wordlen, wrdUP, wrdlen );
2531       // [LP][MP] [] -> [LP][MP][MP]
2532       memcpy( PseudoLinkedPointer+8, &PseudoLinkedPointerNEW, 4 );
2533   }
2534   break;
2535 }
2536 }
2537 else // Empty stack means ROOT and more over ROOT is already splitted(Case #4 is off)
2538 {
2539     // If LEAF is not full: Case #3
2540     // THIS IS WHERE A NEW(SECOND) LEAF 'PseudoLinkedPointerROOT' must be allocated:
2541     if( (unsigned long)(bufend[LetterOffset] - BufStart) + 2*wordlen + 4 + 4 + 4 < GRMBLFOolAgain((int)wordlen) ) // +4 more for BST
2542     instead of LL; + more(see LEAF)
2543     {
2544         memcpy( &PseudoLinkedPointerROOT, &bufend[LetterOffset], 4 );
2545         bufend[LetterOffset] = bufend[LetterOffset] + 4 + 4 + 4; // + 4 due to above commenting
2546         memcpy( bufend[LetterOffset], wrdUP, wrdlen );
2547         bufend[LetterOffset] = bufend[LetterOffset] + 2*wordlen;
2548         if (MAXusedBuffer[wordlen] < (unsigned long)(bufend[LetterOffset] - BufStart)) {MAXusedBuffer[wordlen] = (unsigned
2549 long)(bufend[LetterOffset] - BufStart);}
2550     }
2551     else
2552     { printf( "\nLeprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n" );
2553       printf( fp_outLOG, "Input File with a list of TEXTual Files: %s\n", argv[1] );
2554       printf( fp_outLOG, "Size of all TEXTual Files: %s\n", _ui64toaKAZEcomma(FilesLEM, l1ToaDigits, 10) );
2555       printf( fp_outLOG, "word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, l1ToaDigits, 10), _ui64toaKAZEcomma((unsigned long
2556 long)WORDcountDistinct, l1ToaDigits2, 10) );
2557       printf( fp_outLOG, "Number of Files: %lu\n", NumberOfFiles );
2558       printf( fp_outLOG, "Number of Lines: %lu\n", NumberOfLines );
2559       printf( fp_outLOG, "Allocated memory in MB: %lu\n", (unsigned long)(memory_size>20)+1 );
2560       printf( fp_outLOG, "Total Attempts to Find/Put WORDs into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, l1ToaDigits, 10)
2561 );
2562       for( k = 1; k < 32; k++)
2563       { printf( fp_outLOG, "words with length %s occupy %sKB of %sKB given i.e. %s% utilization\n", _ui64toaKAZEzerocomma(k, l1ToaDigits, 10)+(26-
2564 2), _ui64toaKAZEzerocomma((MAXusedBuffer[k]>10)+1, l1ToaDigits2, 10)+(26-5), _ui64toaKAZEzerocomma(((GRMBLh1l[(int)k] *
2565 letterBuffer)/31)>10)+1, l1ToaDigits3, 10)+(26-5), _ui64toaKAZEzerocomma((unsigned long long)(MAXusedBuffer[k]*100)/((GRMBLh1l[(int)k] *
2566 letterBuffer)/31), l1ToaDigits4, 10)+(26-2), "%0" ); // 26 are all 26-DESIRED=24
2567 }
2568       printf( fp_outLOG, "Used value for third parameter in KB: %lu\n", (unsigned long)thunderwith );
2569       printf( fp_outLOG, "Leprechaun: Failure! Increment 'Memory for each Letter' parameter(third one)!\n\n" );
2570       return( 1 );
2571     }
2572     // Here -- 'PseudoLinkedPointerROOT' --
2573     // (wrdUP)
2574     // 'PseudoLinkedPointer' <-- --> 'PseudoLinkedPointerNEW'
2575     // (LW) (RW)
2576     memcpy( PseudoLinkedPointerROOT, &PseudoLinkedPointer, 4 ); // LP
2577     memcpy( PseudoLinkedPointerROOT+4, &PseudoLinkedPointerNEW, 4 ); // MP
2578     // Here must NEW ROOT be updated i.e. HASH table(SLOT) must point it:
2579     memcpy( BufStart+Slot, &PseudoLinkedPointerROOT, 4 );
2580     break; //because it is ROOT without split
2581 }
2582 } // while
2583 } //if (SplitOccured != 0)

```



```

2577 // 3] Insert Iterative ]
2578 } //if (FoundInLinkedList == 0)
2579 // ##### B-tree order 3 ]
2580 }
2581         } // if( 1 <= wrdlen && wrdlen <= 31 )
2582         wrdlen = 0;
2583         // This fragment is MIRROREd: #1 copy ]
2584         }
2585         //else if( workbyte >= 'A' && workbyte <= 'Z' )
2586         else if( workbyte <= 'z' )
2587         {
2588             if( wrdlen < 31 )
2589             { wrd[ wrdlen ] = workbyte + 32 ; }
2590             wrdlen++;
2591         }
2592         else if( workbyte >= 'a' && workbyte <= 'z' )
2593         {
2594             if( wrdlen < 31 )
2595             { wrd[ wrdlen ] = workbyte; }
2596             wrdlen++;
2597         }
2598         else
2599         {
2600             // This fragment is MIRROREd: #2 copy [
2601             goto Elstupid;
2602             // This fragment is MIRROREd: #2 copy ]
2603         }
2604     } // i 'for'
2605     //~~~~~
2606 ++Melnitckka;
2607 Melnitckka = Melnitckka % 4;
2608 if (Melnitckka == 0){ printf( "]; word count: %s of them %s distinct; Done: %lu/64\n", _ui64toaKAZEcomma(WORDcount, 11ToaDigits, 10),
2609 _ui64toaKAZEcomma(unsigned long long)WORDcountDistinct, 11ToaDigits2, 10, 64 ); }
2609 if (Melnitckka == 1){ printf( "]; word count: %s of them %s distinct; Done: %lu/64\n", _ui64toaKAZEcomma(WORDcount, 11ToaDigits, 10),
2610 _ui64toaKAZEcomma(unsigned long long)WORDcountDistinct, 11ToaDigits2, 10, 64 ); }
2610 if (Melnitckka == 2){ printf( "-; word count: %s of them %s distinct; Done: %lu/64\n", _ui64toaKAZEcomma(WORDcount, 11ToaDigits, 10),
2611 _ui64toaKAZEcomma(unsigned long long)WORDcountDistinct, 11ToaDigits2, 10, 64 ); }
2611 if (Melnitckka == 3){ printf( "\\; word count: %s of them %s distinct; Done: %lu/64\n", _ui64toaKAZEcomma(WORDcount, 11ToaDigits, 10),
2612 _ui64toaKAZEcomma(unsigned long long)WORDcountDistinct, 11ToaDigits2, 10, 64 ); }
2612
2613         LINE10len = 0;
2614         LINE10[ LINE10len ] = 0;
2615         fclose( fp_inLINE );
2616     }
2617     } // k 'for'
2618
2619
2620 (void) time(&t3);
2621 if (t3 <= t1) {t3 = t1; t3++;}
2622 printf( "words per second performance: %sw/s\n", _ui64toaKAZEcomma(WORDcount/((int) t3-t1), 11ToaDigits, 10) ); // Rev. 12+
2623
2624     // FLUSH FLUSH FLUSH FLUSH FLUSH FLUSH FLUSH FLUSH FLUSH FLUSH
2625     printf("Flushing unsorted words...\\n");
2626     if( ( fp_out = fopen( argv[2], "wb+" ) ) == NULL )
2627     { printf( "Leprechaun: Can't create file %s \\n", argv[2] ); return( 1 ); }
2628     ZEROS[0] = 0; ZEROS[1] = 0; ZEROS[2] = 0; ZEROS[3] = 0;
2629     CRdLFa[0] = 13; CRdLFa[1] = 10;
2630
2631     for( i = 0; i < 806; i++ )
2632     { //BufStart = pointerFlush + i * LetterBuffer; // OLD
2633         BufStart = pointerFlush + (i / 31) * WHOLELetter_Buffersize + OffsetsInBuffer[i % 31];
2634         // for( j = 0; j < NumberOfSLOTS; j++ )
2635         {
2636             {
2637                 Slot = j<<2;
2638                 memcpy( &PseudoLinkedPointer, BufStart+Slot, 4 );
2639                 while (PseudoLinkedPointer != 0)
2640                 {
2641                     memcpy( &PseudoLinkedPointerNEW, PseudoLinkedPointer, 4 );
2642                     memcpy( PseudoLinkedPointer, ZEROS, 4 );
2643                     PseudoLinkedPointer = PseudoLinkedPointerNEW;
2644                 }
2645             }
2646             // Start of COUPLES [OFFSET: 4byte(ZEROS)][WORD:up to 31bytes]
2647             //fwrite(BufStart+(NumberOfSLOTS+1)*4, bufend[i] - (BufStart+(NumberOfSLOTS+1)*4), 1, fp_out );
2648             // * Follows STATE of UGLINESS: *
2649             // Flushing = BufStart+(NumberOfSLOTS+1)*4 + 4; // '4' in order to skip first 4 zeros
2650             //in case of current buffer not have been used then NOT entering in this cycle
2651             while( Flushing < bufend[i] )
2652             { if (Flushing != 0) {fwrite(Flushing, 1, 1, fp_out ); TotalWLchars++;}
2653               // below 'Flushing-1' works due to skipped first 4 zeros!
2654               if (Flushing-1 != 0 && Flushing == 0) {fwrite(CRdLFa, 2, 1, fp_out);}
2655               //last word must be suffixed with 1310 too
2656               if (Flushing == bufend[i]-1) {fwrite(CRdLFa, 2, 1, fp_out );}
2657               Flushing++;
2658             }
2659             for( j = 0; j < NumberOfSLOTS; j++ )
2660             {
2661                 Slot = j<<2;

```

```

2661         memcpy( &PseudoLinkedPointer, BufStart+Slot, 4 );
2662         if (PseudoLinkedPointer != 0)
2663         {
2664             NumberOfTrees++;
2665             if (BSTorBtree != 1)
2666             {
2667                 // ===== BST traverse [
2668                 // DONE JOB:
2669                 // Must be written BST traverse ! with simulated stack i.e. non-recursive.
2670                 // ...
2671                 // /*
2672                 // Given a binary search tree, print out
2673                 // its data elements in increasing
2674                 // sorted order.
2675                 // */
2676                 // void printTree(struct node* node) {
2677                 // if (node == NULL) return;
2678                 // printTree(node->left);
2679                 // printf("%d ", node->data);
2680                 // printTree(node->right);
2681                 // }
2682
2683                 // FUTURE JOB:
2684                 // I need functions:
2685                 // BST_LeafNumber() // greater the better
2686                 // BST_NodeNumber() // 'BSTcurrent' below
2687                 // BST_Peak() // i.e. Levels, root has height = 1
2688                 // BST_PeakIB() // IBBST(Ideal Balanced BST) has 1 + lgNodeNumber height
2689                 // I need 'Ideal Balancing BST FRAGMENT' with simulated stack:
2690                 // I need 'Ideal Balancing BST FRAGMENT' to be executed when Peak() >= PeakIB()<<1:
2691
2692                 // ----- [
2693                 BSTcurrentNode = 0; BSTcurrentPeak = 0; BSTcurrentLeaf = 0;
2694                 BSTcurrentPeakMAX = 0; // Height of current BST
2695
2696                 StackPtr = 0;
2697                 while ( 2==2 ) {
2698                     while (PseudoLinkedPointer != 0)
2699                     {
2700                         if (StackPtr > 65536*3-1-3) { printf( "\\nLeprechaun: Failure! BST simulated stack overflow, too high BST!\\n" ); return( 13 );}
2701                         memcpy( &PseudoLinkedPointerNEWleft, PseudoLinkedPointer, 4 );
2702                         PseudoLinkedPointer = PseudoLinkedPointer + 4;
2703                         memcpy( &PseudoLinkedPointerNEWright, PseudoLinkedPointer, 4 );
2704                         BSTstack[StackPtr] = PseudoLinkedPointer + 4; ++StackPtr; //ptr to wrd
2705                         BSTstack[StackPtr] = PseudoLinkedPointerNEWright; ++StackPtr;
2706                         BSTstack[StackPtr] = PseudoLinkedPointerNEWleft; ++StackPtr; //needed for stats not for recursion
2707                     }
2708                     // BST stats [
2709                     if (PseudoLinkedPointerNEWleft == 0 && PseudoLinkedPointerNEWright == 0) {BSTcurrentLeaf++; BSTsTotalLEAFs++;}
2710                     BSTcurrentPeak++;
2711                     if (BSTcurrentPeakMAX < BSTcurrentPeak) BSTcurrentPeakMAX = BSTcurrentPeak;
2712                     BSTstack[StackPtr] = BSTcurrentPeak; ++StackPtr; //needed for stats not for recursion
2713                     // BST stats ]
2714                     PseudoLinkedPointer = PseudoLinkedPointerNEWright; // choose right instead of 'PseudoLinkedPointerNEWleft' because of stats
2715                 }
2716                 print
2717             }
2718             if (StackPtr == 0) break;
2719             BSTcurrentPeak = BSTstack[--StackPtr]; // level of the node(1 is root) needed only for stats(print)
2720             PseudoLinkedPointerNEWleft = BSTstack[--StackPtr]; // left pointer needed only for stats(print)
2721             PseudoLinkedPointerNEWright = BSTstack[--StackPtr]; // right pointer
2722             memcpy( wrd, BSTstack[--StackPtr], i%31+1 );
2723             fwrite(wrd, i%31+1, 1, fp_out); TotalWLchars = TotalWLchars + i%31+1;
2724             fwrite(CRdLFa, 2, 1, fp_out);
2725             BSTcurrentNode++;
2726             PseudoLinkedPointer = PseudoLinkedPointerNEWleft;
2727         }
2728     }
2729     // ----- ]
2730
2731     // BST stats [
2732     if (BSTwithMAXnode < BSTcurrentNode) {
2733         BSTwithMAXnode = BSTcurrentNode;
2734         BSTwithMAXnodePEAK = BSTcurrentPeakMAX;
2735         BSTwithMAXnodeLEAF = BSTcurrentLeaf;
2736         BSTcurrentNodeMAXQUANTITY = 0;
2737     }
2738     if (BSTwithMAXnode == BSTcurrentNode) BSTcurrentNodeMAXQUANTITY++;
2739     if (BSTwithMAXpeak < BSTcurrentPeakMAX) {
2740         BSTwithMAXpeak = BSTcurrentPeakMAX;
2741         BSTwithMAXpeakNODE = BSTcurrentNode;
2742         BSTwithMAXpeakLEAF = BSTcurrentLeaf;
2743         BSTcurrentPeakMAXQUANTITY = 0;
2744         iBSTwithMAXpeak=i; jBSTwithMAXpeak=j;
2745     }
2746     if (BSTwithMAXpeak == BSTcurrentPeakMAX) BSTcurrentPeakMAXQUANTITY++;
2747     if (BSTwithMAXleaf < BSTcurrentLeaf) {
2748         BSTwithMAXleaf = BSTcurrentLeaf;
2749         BSTwithMAXleafNODE = BSTcurrentNode;
2750         BSTwithMAXleafPEAK = BSTcurrentPeakMAX;
2751         BSTcurrentLeafMAXQUANTITY = 0;
2752     }
2753     if (BSTwithMAXleaf == BSTcurrentLeaf) BSTcurrentLeafMAXQUANTITY++;
2754     // BST stats ]

```

```

2748 // ===== BST traverse ]
2749 } else
2750 {
2751 // $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ B-tree order 3 traverse [
2752 // DONE JOB:
2753 // Must be written B-tree traverse ! with simulated stack i.e. non-recursive.
2754 // ...
2755 StackPtr = 0;
2756 while ( 2==2 ) {
2757     while (PseudoLinkedPointer != 0)
2758     {
2759         if (StackPtr > 65536*3-1) { printf( "\nLeprechaun: Failure! B-tree simulated stack overflow, too high B-tree!\n" ); return( 13
2760 );}
2761 BSTstack[StackPtr] = PseudoLinkedPointer; ++StackPtr; //ptr to RwrD
2762 if ( (*char *) (PseudoLinkedPointer + 4 + 4 + 4 + i%31+1) == 0 ) { memcpy( PseudoLinkedPointer + 4 + 4, &BufStart[NumberOfSLOTS*4], 4 ); }
2763 memcpy( &PseudoLinkedPointerNEWleft, PseudoLinkedPointer, 4 );
2764 memcpy( &PseudoLinkedPointerNEWMiddle, PseudoLinkedPointer + 4, 4 );
2765 memcpy( &PseudoLinkedPointerNEWRright, PseudoLinkedPointer + 4 + 4, 4 );
2766 // Give first from right to left non-zero PTR
2767 if (PseudoLinkedPointerNEWRright != 0 )
2768 { memcpy( PseudoLinkedPointer + 4 + 4, &BufStart[NumberOfSLOTS*4], 4 );
2769 PseudoLinkedPointer = PseudoLinkedPointerNEWRright;
2770 }
2771 else if (PseudoLinkedPointerNEWMiddle != 0 )
2772 { memcpy( PseudoLinkedPointer + 4, &BufStart[NumberOfSLOTS*4], 4 );
2773 PseudoLinkedPointer = PseudoLinkedPointerNEWMiddle;
2774 }
2775 else if (PseudoLinkedPointerNEWleft != 0 )
2776 { memcpy( PseudoLinkedPointer, &BufStart[NumberOfSLOTS*4], 4 );
2777 PseudoLinkedPointer = PseudoLinkedPointerNEWleft;
2778 }
2779 else
2780 {
2781 PseudoLinkedPointer = 0;
2782 }
2783 if (StackPtr == 0) break;
2784 PseudoLinkedPointer = BSTstack[--StackPtr];
2785 memcpy( &PseudoLinkedPointerNEWleft, PseudoLinkedPointer, 4 );
2786 memcpy( &PseudoLinkedPointerNEWMiddle, PseudoLinkedPointer + 4, 4 );
2787 memcpy( &PseudoLinkedPointerNEWRright, PseudoLinkedPointer + 4 + 4, 4 );
2788 if (PseudoLinkedPointerNEWleft+PseudoLinkedPointerNEWMiddle+PseudoLinkedPointerNEWRright == 0) // One LEAF is PRINTED when LP=0 MP=0
2789 RP=0
2790 {
2791     memcpy( wrd, PseudoLinkedPointer + 4 + 4 + 4, i%31+1 );
2792     fwrite(wrd, i%31+1, 1, fp_out); TotalWLchars = TotalWLchars + i%31+1;
2793     fwrite(CrdLfA, 2, 1, fp_out);
2794     if ( (*char *) (PseudoLinkedPointer + 4 + 4 + 4 + i%31+1) != 0 )
2795     { memcpy( wrd, PseudoLinkedPointer + 4 + 4 + 4 + i%31+1, i%31+1 );
2796         fwrite(wrd, i%31+1, 1, fp_out); TotalWLchars = TotalWLchars + i%31+1;
2797         fwrite(CrdLfA, 2, 1, fp_out);
2798     }
2799     PseudoLinkedPointer = 0;
2800 }
2801 // $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ B-tree order 3 traverse ]
2802 }
2803 }
2804 } // j
2805
2806 } // i
2807
2808 if (BSTorBtree != 1)
2809 {
2810 // ~~~~~ Longest path ~~~~~ [
2811 i=BSTwithMAXpeak; j=BSTwithMAXpeak;
2812 BufStart = pointerfflush + (i / 31) * WHOLEletter_Buffersize + OffsetsInBuffer[i % 31];
2813 Slot = j<2;
2814 memcpy( &PseudoLinkedPointer, BufStart+Slot, 4 );
2815 if (PseudoLinkedPointer != 0)
2816 {
2817 // ----- [
2818 BSTcurrentNode = 0; BSTcurrentPeak = 0; BSTcurrentLeaf = 0;
2819 BSTcurrentPeakMAX = 0; // Height of current BST
2820 StackPtr = 0;
2821 // BST print [
2822     printf( fp_outLOG, "A(not always THE) Binary-Search-Tree with the longest path(height, PEAK, number of levels):\n" );
2823 // BST print ]
2824 while ( 2==2 ) {
2825     while (PseudoLinkedPointer != 0)
2826     {
2827         if (StackPtr > 65536*3-1-3) { printf( "\nLeprechaun: Failure! BST simulated stack overflow, too high BST!\n" ); return( 13 );}
2828         memcpy( &PseudoLinkedPointerNEWleft, PseudoLinkedPointer, 4 );
2829         PseudoLinkedPointer = PseudoLinkedPointer + 4;
2830         memcpy( &PseudoLinkedPointerNEWRright, PseudoLinkedPointer, 4 );
2831         BSTstack[StackPtr] = PseudoLinkedPointer + 4; ++StackPtr; //ptr to wrd
2832         BSTstack[StackPtr] = PseudoLinkedPointerNEWRright; ++StackPtr;
2833         BSTstack[StackPtr] = PseudoLinkedPointerNEWleft; ++StackPtr; //needed for stats not for recursion

```

```

2834 // BST stats [
2835     if (PseudoLinkedPointerNEWleft == 0 && PseudoLinkedPointerNEWright == 0) {BSTcurrentLeaf++;} //BSTstotalLEAFs++;} // REMOVED
2836 to avoid mess in TOTAL stats
2837     BSTcurrentPeak++;
2838     if (BSTcurrentPeakMAX < BSTcurrentPeak) BSTcurrentPeakMAX = BSTcurrentPeak;
2839     BSTstack[StackPtr] = BSTcurrentPeak; ++StackPtr; //needed for stats not for recursion
2840 // BST stats ]
2841     PseudoLinkedPointer = PseudoLinkedPointerNEWRright; // choose right instead of 'PseudoLinkedPointerNEWleft' because of stats
2842 print
2843     }
2844     if (StackPtr == 0) break;
2845     BSTcurrentPeak = BSTstack[--StackPtr]; // level of the node(1 is root) needed only for stats(print)
2846     PseudoLinkedPointerNEWleft = BSTstack[--StackPtr]; // left pointer needed only for stats(print)
2847     PseudoLinkedPointerNEWRright = BSTstack[--StackPtr]; // right pointer
2848     memcpy( wrd, BSTstack[--StackPtr], i%31+1 );
2849     //fwrite(wrd, i%31+1, 1, fp_out); TotalWLchars = TotalWLchars + i%31+1;
2850     //fwrite(CrdLfA, 2, 1, fp_out);
2851     BSTcurrentNode++;
2852     PseudoLinkedPointer = PseudoLinkedPointerNEWleft;
2853     // BST print [
2854     for (k = 0; k < BSTcurrentPeak; k++) fprintf( fp_outLOG, "%c", ' ' );
2855     if (PseudoLinkedPointerNEWleft == 0) fprintf( fp_outLOG, "[ " ); else fprintf( fp_outLOG, "]" );
2856     for (k = 0; k < i%31+1; k++) fprintf( fp_outLOG, "%c", *(char *) (wrd+k) );
2857     if (PseudoLinkedPointerNEWRright == 0) fprintf( fp_outLOG, "]" ); else fprintf( fp_outLOG, "[ " );
2858     if (BSTcurrentPeak == 1) fprintf( fp_outLOG, " ROOT" );
2859     fprintf( fp_outLOG, "\n" );
2860 // BST print ]
2861 }
2862 // ----- ]
2863 fprintf( fp_outLOG, "Above Binary-Search-Tree with MaxPEAK = %s has NODES = %s and LEAFs = %s\n", _ui64toaKAZEcomma(BSTwithMAXpeak,
2864 11ToaDigits2, 10), _ui64toaKAZEcomma(BSTwithMAXpeakNODE, 11ToaDigits3, 10), _ui64toaKAZEcomma(BSTwithMAXpeakLEAF, 11ToaDigits, 10));
2865 fprintf( fp_outLOG, "Legend:\n" );
2866 fprintf( fp_outLOG, "At left side of the word - '[' means no left successor\n" );
2867 fprintf( fp_outLOG, "At left side of the word - '[' means left successor exists\n" );
2868 fprintf( fp_outLOG, "At right side of the word - '[' means no right successor\n" );
2869 fprintf( fp_outLOG, "At right side of the word - '[' means right successor exists\n" );
2870 // ~~~~~ Longest path ~~~~~ ]
2871
2872 // BST stats [
2873 PEAKibBST=1+floorLog2(BSTwithMAXnode);
2874 //PEAKibBST=1;
2875 //while (BSTwithMAXnode>PEAKibBST) PEAKibBST++;
2876 // BST stats ]
2877
2878 (void) time(&t2);
2879 if (t2 <= t1) {t2 = t1; t2++;}
2880 printf("Time for making unsorted wordlist: %d second(s)\n", (int) t2-t1);
2881 printf( fp_outLOG, "Bytes per second performance: %sB/s\n", _ui64toaKAZEcomma(FilesLEN/((int) t3-t1), 11ToaDigits, 10) ); // Rev. 12+
2882 printf( fp_outLOG, "Words per second performance: %sW/s\n", _ui64toaKAZEcomma(WORDcount/((int) t3-t1), 11ToaDigits, 10) ); // Rev. 12+
2883 printf( fp_outLOG, "Input File with a list of TEXTUAL Files: %s\n", argv[1] );
2884 printf( fp_outLOG, "Size of all TEXTUAL Files: %s\n", _ui64toaKAZEcomma(FilesLEN, 11ToaDigits, 10) );
2885 printf( fp_outLOG, "Word count: %s of them %s distinct\n", _ui64toaKAZEcomma(WORDcount, 11ToaDigits, 10), _ui64toaKAZEcomma((unsigned long)
2886 WORDcountDistinct, 11ToaDigits2, 10) );
2887 printf( fp_outLOG, "Number of Files: %lu\n", NumberOfFiles );
2888 printf( fp_outLOG, "Number of Lines: %lu\n", NumberOfLines );
2889 printf( fp_outLOG, "Allocated memory in MB: %lu\n", (unsigned long)(memory_size>20)+1 );
2890 printf( fp_outLOG, "Forest population(Hash Function Quality regarding collisions i.e. Hash Table Utilization): %lu%s\n",
2891 (NumberOfTrees*100)/(26*31*8192), "%0" );
2892 printf( fp_outLOG, "Number of Hash collisions(Distinct WORDS - Number of Trees): %lu\n", NumberOfHashCollisions );
2893
2894 if (BSTorBtree != 1)
2895 {
2896     printf( fp_outLOG, "Maximum Attempts to Find/Put a WORD into a Binary-Search-Tree: %s'\n", _ui64toaKAZEcomma(BSTwithMAXpeak, 11ToaDigits,
2897 10) );
2898     printf( fp_outLOG, "Total Attempts to Find/Put WORDS into Binary-Search-Trees: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11ToaDigits,
2899 10) );
2900     printf( fp_outLOG, "Total Number of LEAFs in Binary-Search-Trees(GREATER THE BETTER): %s\n", _ui64toaKAZEcomma(BSTstotalLEAFs, 11ToaDigits,
2901 10) );
2902     printf( fp_outLOG, "Perfectly-Balanced-Binary-Search-Tree for MaxNODES = %s must have PEAK = %s = rounding down of integer (1+lb(%s))\n",
2903     _ui64toaKAZEcomma(BSTwithMAXnode, 11ToaDigits, 10), _ui64toaKAZEcomma(PEAKibBST, 11ToaDigits2, 10), _ui64toaKAZEcomma(BSTwithMAXnode,
2904 11ToaDigits3, 10));
2905     printf( fp_outLOG, "Binary-Search-Tree(1st out of %s) with MaxNODES = %s has PEAK = %s and LEAFs = %s\n",
2906     _ui64toaKAZEcomma(BSTcurrentNodeMAXQUANTITY, 11ToaDigits4, 10), _ui64toaKAZEcomma(BSTwithMAXnode, 11ToaDigits2, 10),
2907     _ui64toaKAZEcomma(BSTwithMAXnodePEAK, 11ToaDigits3, 10), _ui64toaKAZEcomma(BSTwithMAXnodeLEAF, 11ToaDigits, 10));
2908     printf( fp_outLOG, "Binary-Search-Tree(1st out of %s) with MaxPEAK = %s has NODES = %s and LEAFs = %s\n",
2909     _ui64toaKAZEcomma(BSTcurrentPeakMAXQUANTITY, 11ToaDigits4, 10), _ui64toaKAZEcomma(BSTwithMAXpeak, 11ToaDigits2, 10),
2910     _ui64toaKAZEcomma(BSTwithMAXpeakNODE, 11ToaDigits3, 10), _ui64toaKAZEcomma(BSTwithMAXpeakLEAF, 11ToaDigits, 10));
2911     printf( fp_outLOG, "Binary-Search-Tree(1st out of %s) with MaxLEAFs = %s has NODES = %s and PEAK = %s\n",
2912     _ui64toaKAZEcomma(BSTcurrentLeafMAXQUANTITY, 11ToaDigits4, 10), _ui64toaKAZEcomma(BSTwithMAXleaf, 11ToaDigits2, 10),
2913     _ui64toaKAZEcomma(BSTwithMAXleafNODE, 11ToaDigits3, 10), _ui64toaKAZEcomma(BSTwithMAXleafPEAK, 11ToaDigits, 10));
2914     }
2915     else
2916     {
2917         printf( fp_outLOG, "Total Attempts to Find/Put WORDS into B-trees order 3: %s\n", _ui64toaKAZEcomma(WORDcountAttemptsToPut, 11ToaDigits, 10)
2918 );
2919     }

```

Leprechaun(Fast Greedy Word-Ripper), revision 13+++++; B-trees order 3 &(not versus) Binary Search Trees

```

3068         slot = FNV1A_Hash_4_OCTETS_31(backup[j], (strlen(backup[j])>>2)); //13++
3069     }
3070 }
3071 clocks2 = clock();
3072 printf( "Performance of 'FNV1A_Hash_4_OCTETS_31': %lu words/clock or %lu MB/s\n", (nlines/((long)(clocks2 - clocks1 + 1)>>4)) ,
((TotalWChars>>10)/((long)(clocks2 - clocks1 + 1)>>4)) );
3073 // 2]
3074
3075 // 4[
3076 clocks1 = clock();
3077 for (Bozan=0; Bozan < (1<<4); Bozan++) // 16 times, at end >>4
3078 {
3079     for( j = 0; j < nlines; j++ )
3080     { //slot = kuxHash3plus(backup[j]);
3081         //fprintf(fp_out, "Hashcode: %s - ", _ui64toaKAZEzerocomma(slot, 11TOadigits, 10)+(26-5));
3082         // To make it EVEN !!!
3083         //wordlen = strlen(backup[j]);
3084         //if (strlen(backup[j]) != 0)
3085             slot = kuxHash3plus(backup[j]); //13++
3086     }
3087 }
3088 clocks2 = clock();
3089 printf( "Performance of 'kuxHash3plus': %lu words/clock or %lu MB/s\n", (nlines/((long)(clocks2 - clocks1 + 1)>>4)) ,
((TotalWChars>>10)/((long)(clocks2 - clocks1 + 1)>>4)) );
3090 // 4]
3091
3092 // 6[
3093 clocks1 = clock();
3094 for (Bozan=0; Bozan < (1<<4); Bozan++) // 16 times, at end >>4
3095 {
3096     for( j = 0; j < nlines; j++ )
3097     { //slot = kuxHash3plus(backup[j]);
3098         //fprintf(fp_out, "Hashcode: %s - ", _ui64toaKAZEzerocomma(slot, 11TOadigits, 10)+(26-5));
3099         wrdlen = strlen(backup[j]);
3100         if (wrdlen<=19) // 4x4+3=19 i.e. last contains 7 clashes
3101             slot = FNV1A_Hash_Grularity(wrd, wrdlen>>2, 2); //13++++
3102         else // 2x8+4=20 i.e. first contains 6 clashes
3103             slot = FNV1A_Hash_Grularity(wrd, wrdlen>>3, 3); //13++++
3104     } // Conclusion: two functions > 64 bytes lead to horrible slowness, so unite them in one: fit in the cache line.
3105 }
3106 clocks2 = clock();
3107 printf( "Performance of 'FNV1A_Hash_Grularity': %lu words/clock or %lu MB/s\n", (nlines/((long)(clocks2 - clocks1 + 1)>>4)) ,
((TotalWChars>>10)/((long)(clocks2 - clocks1 + 1)>>4)) );
3108 // 6]
3109
3110 // Hash benchmarking ----- ]
3111 */
3112
3113
3114 if( ( fp_outLOG = fopen( "Leprechaun.LOG", "a+" ) ) == NULL )
3115 { printf( "Leprechaun: Can't open file Leprechaun.LOG.\n" ); return( 1 ); }
3116 fprintf( fp_outLOG, "Time for sorting unsorted wordlist: %d second(s)\n\n", (int) t3-t2);
3117 printf( "Leprechaun: Done.\n" );
3118     return 0;
3119 }
3120 else
3121 { printf("Leprechaun: Input file too large, wordlist remains unsorted!\n");
3122     return 1;
3123 }
3124 // SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT SORT
3125 }
3126
3127 /*
3128 TO BE DONE: Ideal Balancing BST [
3129
3130 link rotR(link h)
3131 { link x = h->l; h->l = x->r; x->r = h;
3132     return x; }
3133
3134 link rotL(link h)
3135 { link x = h->r; h->r = x->l; x->l = h;
3136     return x; }
3137
3138 link parR(link h, int k)
3139 { int t = h->l->N;
3140     if (t > k )
3141     { h->l = parR(h->l, k); h = rotR(h); }
3142     if (t < k )
3143     { h->r = parR(h->r, k-t-1); h = rotL(h); }
3144     return h;
3145 }
3146
3147 link balancer(link h)
3148 {
3149     if (h->N < 2) return h;
3150     h = parR(h, h->N/2);
3151     h->l = balancer(h->l);
3152     h->r = balancer(h->r);

```

```

3153     return h;
3154 }
3155
3156 TO BE DONE: Ideal Balancing BST ]
3157 */
3158
3159 /*
3160 #include <stdlib.h>
3161 #include "Item.h"
3162 typedef struct STnode* link;
3163 struct STnode { Item item; link l, r; int N };
3164 static link head, z;
3165 link NEW(Item item, link l, link r, int N)
3166 { link x = malloc(sizeof *x);
3167     x->item = item; x->l = l; x->r = r; x->N = N;
3168     return x;
3169 }
3170 void STinit()
3171 { head = (z = NEW(NULLitem, 0, 0, 0)); }
3172 int STcount() { return head->N; }
3173 Item searchR(link h, Key v)
3174 { Key t = key(h->item);
3175     if (h == z) return NULLitem;
3176     if (eq(v, t) return h->item;
3177     if (less(v, t) return searchR(h->l, v);
3178         else return searchR(h->r, v);
3179 }
3180 Item STsearch(Key v)
3181 { return searchR(head, v); }
3182 link insertR(link h, Item item)
3183 { Key v = key(item), t = key(h->item);
3184     if (h == z) return NEW(item, z, z, 1);
3185     if (less(v, t)
3186         h->l = insertR(h->l, item);
3187     else h->r = insertR(h->r, item);
3188     (h->N)++; return h;
3189 }
3190 void STinsert(Item item)
3191 { head = insertR(head, item); }
3192 */
3193
3194 /*
3195 int count(link h)
3196 {
3197     if (h == NULL) return 0;
3198     return count(h->l) + count(h->r) + 1;
3199 }
3200
3201 int height(link h)
3202 { int u, v;
3203     if (h == NULL) return -1;
3204     u = height(h->l); v = height(h->r);
3205     if (u > v) return u+1; else return v+1;
3206 }
3207
3208 void printnode(char c, int h)
3209 { int i;
3210     for (i = 0; i < h; i++) printf(" ");
3211     printf("%c\n", c);
3212 }
3213
3214 void show(link x, int h)
3215 {
3216     if (x == NULL) { printnode("x", h); return; }
3217     show(x->r, h+1);
3218     printnode(x->item, h);
3219     show(x->l, h+1);
3220 }
3221 */

```