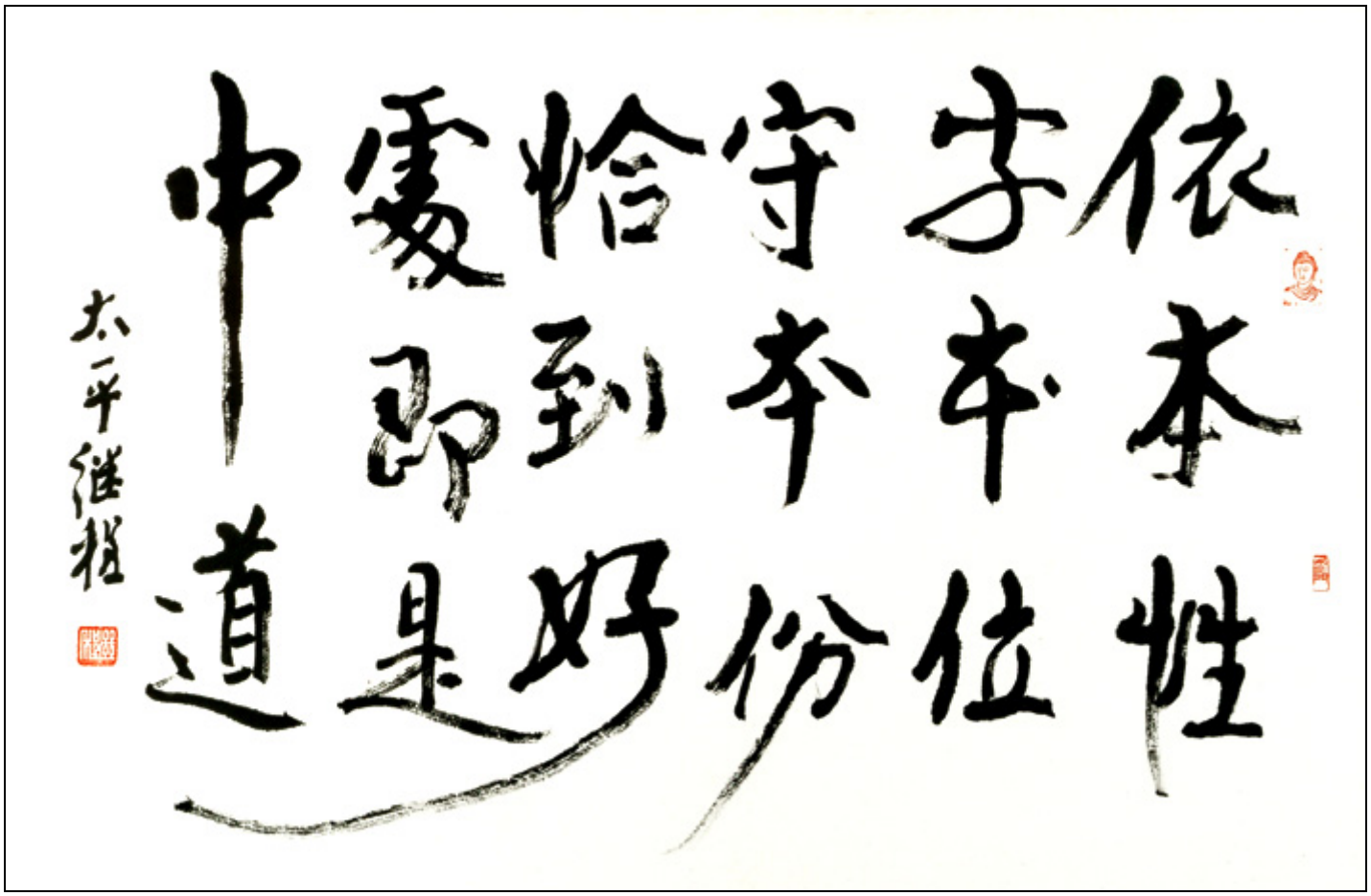
中道

NAKAMICHI




```
0001 // Nakamichi is 100% FREE LZSS FAST decompressor.
0002
0003 // Nakamichi, revision 1-RSSBO++_1GB_7bit, written by Kaze.
0004 // Change #1: Now OFFSET is coded by 7bits i.e. window is 128bytes big, my choice for Min_Match_Length is 8, 16 is tempting since my texts are mainly sorted phraselists i.e. with very similar adjacent lines.
0005 // Nakamichi, revision 1-RSSBO++_1GB, written by Kaze.
0006 // Change #1: Decompression fetches WORD instead of BYTE+BYTE.
0007 // Nakamichi, revision 1-RSSBO+, written by Kaze.
0008 // Change #1: Now, fifteenth bit is used, 'Middle Way' i.e. 8 is replaced by the two extremities - 4 and 16 - the 16bytes long matches are handled by an XMM.
0009 // Change #2: Fixed bug in progress indicator.
0010 // Change #3: Not done yet, the two nasty byte loads to be removed in the decompression part.
0011 // Nakamichi, revision 1-RSSBO, written by Kaze.
0012 // Based on Nobuo Ito's source, thanks Ito.
0013 // The main goal of Nakamichi is to allow supersimple and superfast decoding for English x-grams (mainly) in pure C, or not, heh-heh.
0014 // Natively Nakamichi is targeted as 64bit tool with 16 threads, helping Kazahana to traverse faster when I/O is not superior.
0015 // In short, Nakamichi is intended as x-gram decompressor.
0016
0017 // Eightfold Path ~ the Buddhist path to nirvana, comprising eight aspects in which an aspirant must become practised;
0018 // eightfold way ~ (Physics), the grouping of hadrons into supermultiplets by means of SU(3)); (b) adverb to eight times the number or quantity: OE.
0019
0020 // Revision 1-RSSBO++ notes [
0021 // Note1: On 'ENWIKI', being a XML English text, Nakamichi's compression ratio is among the worsts:
0022 //        021,440,055 enwiki-20140304-pages-articles.7z.001.tangelo
0023 //        041,984,881 enwiki-20140304-pages-articles.7z.001.lzt        -19
0024 //        047,685,453 enwiki-20140304-pages-articles.7z.001.lzt        -11
0025 //        056,365,696 enwiki-20140304-pages-articles.7z.001.YAPPY      32768   uncomp 534.3 MB/s
0026 //        057,497,885 enwiki-20140304-pages-articles.7z.001.YAPPY      16384   uncomp 549.5 MB/s !SUPERIOR DECOMPRESSOR!
0027 //        059,756,354 enwiki-20140304-pages-articles.7z.001.YAPPY      8192    uncomp 549.5 MB/s
0028 //        060,865,688 enwiki-20140304-pages-articles.7z.001.Nakamichi  Decompression at 355 MB/s
0029 //        064,270,963 enwiki-20140304-pages-articles.7z.001.YAPPY      4096    uncomp 574.2 MB/s
0030 //        070,845,249 enwiki-20140304-pages-articles.7z.001.YAPPY      2048    uncomp 631.8 MB/s
0031 //        078,039,768 enwiki-20140304-pages-articles.7z.001.YAPPY      1024    uncomp 680.3 MB/s
```

```
0032 //        104,857,600 enwiki-20140304-pages-articles.7z.001
0033 // Note2: On 'PAGODA', being a highly redundant English text, Nakamichi's compression ratio is among the worsts BUT good enough for my goals:
0034 //        035,834,062 Kazahana_on.PAGODA-order-5.txt.tangelo
0035 //        107,077,360 Kazahana_on.PAGODA-order-5.txt.lzt               -19
0036 //        119,750,869 Kazahana_on.PAGODA-order-5.txt.lzt               -11
0037 //        178,902,966 Kazahana_on.PAGODA-order-5.txt.YAPPY             32768   uncomp 932.0 MB/s
0038 //        180,650,931 Kazahana_on.PAGODA-order-5.txt.YAPPY             16384   uncomp 940.7 MB/s !SUPERIOR DECOMPRESSOR!
0039 //        184,153,244 Kazahana_on.PAGODA-order-5.txt.YAPPY             8192    uncomp 923.5 MB/s
0040 //        191,149,889 Kazahana_on.PAGODA-order-5.txt.YAPPY             4096    uncomp 949.0 MB/s
0041 //        195,522,294 Kazahana_on.PAGODA-order-5.txt.Nakamichi         Decompression at 646 MB/s
0042 //        202,655,189 Kazahana_on.PAGODA-order-5.txt.YAPPY             2048    uncomp 967.2 MB/s
0043 //        219,625,842 Kazahana_on.PAGODA-order-5.txt.YAPPY             1024    uncomp 958.0 MB/s
0044 //        846,351,894 Kazahana_on.PAGODA-order-5.txt
0045 // Note3: On 'OSHO', being a typical English text, Nakamichi's compression ratio is among the worsts:
0046 //        034,419,437 OSHO.TXT.tangelo
0047 //        070,067,665 OSHO.TXT.lzt                                     -19
0048 //        085,054,228 OSHO.TXT.lzt                                     -11
0049 //        097,806,047 OSHO.TXT.YAPPY                                   32768   uncomp 512.5 MB/s
0050 //        099,927,141 OSHO.TXT.YAPPY                                   16384   uncomp 519.7 MB/s !SUPERIOR DECOMPRESSOR!
0051 //        104,158,750 OSHO.TXT.YAPPY                                   8192    uncomp 519.7 MB/s
0052 //        106,283,618 OSHO.TXT.Nakamichi                               Decompression at 406 MB/s
0053 //        112,642,538 OSHO.TXT.YAPPY                                   4096    uncomp 542.1 MB/s
0054 //        124,617,790 OSHO.TXT.YAPPY                                   2048    uncomp 575.3 MB/s
0055 //        137,306,077 OSHO.TXT.YAPPY                                   1024    uncomp 617.1 MB/s
0056 //        206,908,949 OSHO.TXT
0057 // Revision 1-RSSBO++ notes ]
0058
0059 // Revision 1-RSSBO notes [
0060 // Note1: Fifteenth bit is not used, making the window wider by 1bit i.e. 32KB is not tempting, rather I think to use it as a flag: 8bytes/16bytes.
0061 // Note2: English x-grams are as English texts but more redundant, in other words they are phraselists in most cases, sometimes wordlists.
0062 // Note3: On OSHO.TXT, being a typical English text, Nakamichi's compression ratio is among the worsts:
0063 //        206,908,949 OSHO.TXT
0064 //        125,022,859 OSHO.TXT.Nakamichi
0065 //        It struggles with English texts but decomprression speed is quite sweet (Core 2 T7500 2200MHz, 32bit code):
0066 //        Nakamichi, revision 1-, written by Kaze.
0067 //        Decompressing 125022859 bytes ...
0068 //        RAM-to-RAM performance: 477681 KB/s.
0069 // Note4: Also I wanted to see how my 'Railgun_Swampshine_BailOut', being a HEAVYGUN i.e. with big overhead and latency, hits in a real-world application.
0070 // Revision 1-RSSBO notes ]
0071
0072 // Quick notes on PAGODAs (the padded x-gram lists):
0073 // Every single word in English has its own PAGODA, in example below 'on' PAGODA is given (Kazahana_on.PAGODA-order-5.txt):
0074 // PAGODA order 5 (i.e. with 5 tiers) has 5*(5+1)/2=15 subtiers, they are concatenated and space-padded in order to form the pillar 'on':
0075 /*
0076 D:\_KAZE\Nakamichi_r1-RSSBO>dir \_GW\ka*
0077
0078 04/12/2014  05:07 AM                14 Kazahana_on.1-1.txt
0079 04/12/2014  05:07 AM         1,635,389 Kazahana_on.2-1.txt
0080 04/12/2014  05:07 AM         1,906,734 Kazahana_on.2-2.txt
0081 04/12/2014  05:07 AM        10,891,415 Kazahana_on.3-1.txt
0082 04/12/2014  05:07 AM        15,797,703 Kazahana_on.3-2.txt
0083 04/12/2014  05:07 AM        20,419,280 Kazahana_on.3-3.txt
0084 04/12/2014  05:07 AM        22,141,823 Kazahana_on.4-1.txt
0085 04/12/2014  05:07 AM        36,002,113 Kazahana_on.4-2.txt
0086 04/12/2014  05:07 AM        33,236,772 Kazahana_on.4-3.txt
0087 04/12/2014  05:07 AM        33,902,425 Kazahana_on.4-4.txt
0088 04/12/2014  05:07 AM        24,795,989 Kazahana_on.5-1.txt
0089 04/12/2014  05:07 AM        30,766,220 Kazahana_on.5-2.txt
0090 04/12/2014  05:07 AM        38,982,816 Kazahana_on.5-3.txt
0091 04/12/2014  05:07 AM        38,089,575 Kazahana_on.5-4.txt
0092 04/12/2014  05:07 AM        34,309,057 Kazahana_on.5-5.txt
0093 04/12/2014  05:07 AM       846,351,894 Kazahana_on.PAGODA-order-5.txt
0094
0095 D:\_KAZE\Nakamichi_r1-RSSBO>type \_GW\Kazahana_on.1-1.txt
0096 9,999,999       on
0097
0098 D:\_KAZE\Nakamichi_r1-RSSBO>type \_GW\Kazahana_on.2-1.txt
0099 9,999,999       on_the
0100 1,148,054       on_his
0101 0,559,694       on_her
0102 0,487,856       on_this
0103 0,399,485       on_your
0104 0,381,570       on_my
0105 0,367,282       on_their
0106 ...
0107
0108 D:\_KAZE\Nakamichi_r1-RSSBO>type \_GW\Kazahana_on.2-2.txt
0109 0,545,191       based_on
0110 0,397,408       and_on
0111 0,334,266       go_on
0112 0,329,561       went_on
0113 0,263,035       was_on
0114 0,246,332       it_on
0115 0,229,041       down_on
0116 0,202,151       going_on
0117 ...
0118
0119 D:\_KAZE\Nakamichi_r1-RSSBO>type \_GW\Kazahana_on.5-5.txt
0120 0,083,564       foundation_osho_s_books_on
```

```
0121 0,012,404       medium_it_may_be_on
0122 0,012,354       if_you_received_it_on
0123 0,012,152       medium_they_may_be_on
0124 0,012,144       agree_to_also_provide_on
0125 0,012,139       a_united_states_copyright_on
0126 0,008,067       we_are_constantly_working_on
0127 0,008,067       questions_we_have_received_on
0128 0,006,847       file_was_first_posted_on
0129 0,006,441       of_we_are_already_on
0130 0,006,279       you_received_this_ebook_on
0131 0,005,865       you_received_this_etext_on
0132 0,005,833       to_keep_an_eye_on
0133 ...
0134
0135 D:\_KAZE\Nakamichi_r1-RSSBO>type \_GW\Kazahana_on.PAGODA-order-5.txt
0136 9,999,999                                                        on
0137 9,999,999                                                        on_the
0138 1,148,054                                                        on_his
0139 0,559,694                                                        on_her
0140 0,487,856                                                        on_this
0141 0,399,485                                                        on_your
0142 0,381,570                                                        on_my
0143 0,367,282                                                        on_their
0144 0,356,572                                                        on_to
0145 0,299,453                                                        on_which
0146 0,291,876                                                        on_that
0147 0,276,388                                                        on_it
0148 0,250,541                                                        on_one
0149 0,221,101                                                        on_him
0150 0,206,864                                                        on_an
0151 0,197,541                                                        on_its
0152 0,166,762                                                        on_earth
0153 0,165,157                                                        on_page
0154 0,156,605                                                        on_our
0155 0,155,320                                                        on_all
0156 0,149,930                                                        on_account
0157 ...
0158 0,002,479                                             updated_on
0159 0,002,478                                              signal_on
0160 0,002,477                                               threw_on
0161 0,002,475                                                toll_on
0162 0,002,475                                           predicated_on
0163 0,002,472                                               stains_on
0164 0,002,471                                                 seal_on
0165 ...
0166 0,000,004                                                 aarre_on
0167 0,000,004                                                  aapp_on
0168 0,000,004                                               aanandham_on
0169 0,437,178                                                        on_the_other
0170 0,153,103                                                        on_the_ground
0171 0,146,123                                                        on_account_of
0172 0,140,358                                                        on_the_floor
0173 0,115,525                                                        on_to_the
0174 0,113,868                                                        on_the_table
0175 0,113,560                                                        on_the_contrary
0176 0,111,944                                                        on_the_same
0177 ...
0178 0,000,004                                                        on_a_actual
0179 0,000,004                                                        on_a_abattu
0180 0,181,050                                                 based_on_the
0181 0,151,126                                                   and_on_the
0182 0,109,493                                                   was_on_the
0183 0,108,238                                                  down_on_the
0184 0,089,090                                               depending_on_the
0185 0,083,596                                                   books_on_cd
0186 0,077,772                                                      it_on_the
0187 0,075,412                                                    out_on_the
0188 ...
0189 0,000,004                                                       a_on_campo
0190 0,000,004                                                       a_on_average
0191 0,159,602                                                  and_so_on
0192 0,083,791                                                 s_books_on
0193 0,081,020                                                is_based_on
0194 0,065,126                                                 he_went_on
0195 0,047,856                                               was_going_on
0196 ...
0197 0,000,009                                      optical_transceivers_on
0198 0,000,009                                         oppressive_weight_on
0199 0,000,009                                              opposite_way_on
0200 0,000,009                                            opposite_views_on
0201 ...
0202 0,000,012                                                        on_the_doorstep_were
0203 0,000,012                                                        on_the_doorstep_said
0204 0,000,012                                                        on_the_doorstep_looking
0205 ...
0206 0,000,027                                                  depends_on_several_variables
0207 0,000,027                                                  depends_on_several_things
0208 0,000,027                                                  depends_on_people_s
0209 ...
```

```
0210 0,000,004                                    genetic_influences_on_personality
0211 0,000,004                                    genetic_influences_on_osteoarthritis
0212 0,000,004                                    genetic_influences_on_behavior
0213 ...
0214 0,000,004                                                  on_the_train_to_hell
0215 0,000,004                                                  on_the_train_to_delhi
0216 0,000,004                                                  on_the_train_to_cambridge
0217 ...
0218 0,000,006                                                now_on_the_puppeteers_will
0219 0,000,006                                                now_on_the_open_ground
0220 0,000,006                                                now_on_the_naked_woods
0221 ...
0222 0,000,004                                       leaned_lightly_on_his_shoulder
0223 0,000,004                                       leaned_lightly_on_her_shoulders
0224 0,000,004                                        leaned_lazily_on_the_neck
0225 ...
0226 0,000,039                               american_professional_society_on_the
0227 0,000,039                                         altars_that_were_on_the
0228 0,000,039                                         also_some_verses_on_the
0229 ...
0230 0,000,008                               portrait_of_washington_carved_on
0231 0,000,008                                     portrait_of_a_gentleman_on
0232 0,000,008                                portlets_without_single_sign_on
0233 0,000,008                                    portion_of_this_treatise_on
0234 ...
0235
0236 D:\_KAZE\Nakamichi_r1-RSSBO>dir
0237
0238 04/12/2014  05:07 AM       846,351,894 Kazahana_on.PAGODA-order-5.txt
0239
0240 D:\_KAZE\Nakamichi_r1-RSSBO>Nakamichi.exe Kazahana_on.PAGODA-order-5.txt
0241 Nakamichi, revision 1-RSSBO, written by Kaze.
0242 Compressing 846351894 bytes ...
0243 /; Each rotation means 128KB are encoded; Done 100%
0244 RAM-to-RAM performance: 512 KB/s.
0245
0246 D:\_KAZE\Nakamichi_r1-RSSBO>dir
0247
0248 04/12/2014  05:07 AM       846,351,894 Kazahana_on.PAGODA-order-5.txt
0249 04/15/2014  06:30 PM       293,049,398 Kazahana_on.PAGODA-order-5.txt.Nakamichi
0250
0251 D:\_KAZE\Nakamichi_r1-RSSBO>Nakamichi.exe Kazahana_on.PAGODA-order-5.txt.Nakamichi
0252 Nakamichi, revision 1-RSSBO, written by Kaze.
0253 Decompressing 293049398 bytes ...
0254 RAM-to-RAM performance: 607 MB/s.
0255
0256 D:\_KAZE\Nakamichi_r1-RSSBO>Yappy.exe Kazahana_on.PAGODA-order-5.txt 4096
0257 YAPPY: [b 4K] bytes 846351894 -> 191149889  22.6%  comp  33.8 MB/s  uncomp 875.4 MB/s
0258
0259 D:\_KAZE\Nakamichi_r1-RSSBO>Yappy.exe Kazahana_on.PAGODA-order-5.txt 8192
0260 YAPPY: [b 8K] bytes 846351894 -> 184153244  21.8%  comp  35.0 MB/s  uncomp 898.3 MB/s
0261
0262 D:\_KAZE\Nakamichi_r1-RSSBO>Yappy.exe Kazahana_on.PAGODA-order-5.txt 16384
0263 YAPPY: [b 16K] bytes 846351894 -> 180650931  21.3%  comp  28.8 MB/s  uncomp 906.4 MB/s
0264
0265 D:\_KAZE\Nakamichi_r1-RSSBO>Yappy.exe Kazahana_on.PAGODA-order-5.txt 32768
0266 YAPPY: [b 32K] bytes 846351894 -> 178902966  21.1%  comp  35.0 MB/s  uncomp 906.4 MB/s
0267
0268 D:\_KAZE\Nakamichi_r1-RSSBO>Yappy.exe Kazahana_on.PAGODA-order-5.txt 65536
0269 YAPPY: [b 64K] bytes 846351894 -> 178027899  21.0%  comp  34.5 MB/s  uncomp 914.6 MB/s
0270
0271 D:\_KAZE\Nakamichi_r1-RSSBO>Yappy.exe Kazahana_on.PAGODA-order-5.txt 131072
0272 YAPPY: [b 128K] bytes 846351894 -> 177591807  21.0%  comp  34.9 MB/s  uncomp 906.4 MB/s
0273
0274 D:\_KAZE\Nakamichi_r1-RSSBO>
0275 */
0276
0277 #include <stdio.h>
0278 #include <stdlib.h>
0279 #include <stdint.h> // uint64_t needed
0280 #include <time.h>
0281 #include <string.h>
0282
0283 #include <emmintrin.h> // SSE2 intrinsics
0284 //#include <smmintrin.h> // SSE4.1 intrinsics
0285 //#include <immintrin.h> // AVX intrinsics
0286
0287 void SlowCopy128bit(const char *SOURCE, char *TARGET) { _mm_storeu_si128((__m128i *)(TARGET), _mm_loadu_si128((const __m128i *)(SOURCE))); }
0288
0289 #ifndef NULL
0290 #define NULL ((void*)0)
0291 #endif
0292
0293 // Comment it to see how slower 'BruteForce' is, for Wikipedia 100MB the ratio is 41KB/s versus 197KB/s.
0294 #define ReplaceBruteForceWithRailgunSwampshineBailOut
0295
0296 void SearchIntoSlidingWindow(unsigned int* retIndex, unsigned int* retMatch, char* refStart,char* refEnd,char* encStart,char* encEnd);
0297 unsigned int SlidingWindowVsLookAheadBuffer(char* refStart, char* refEnd, char* encStart, char* encEnd);
0298 unsigned int Compress(char* ret, char* src, unsigned int srcSize);
```

```
0299 unsigned int Decompress(char* ret, char* src, unsigned int srcSize);
0300 char * Railgun_Swampshine_BailOut(char * pbTarget, char * pbPattern, uint32_t cbTarget, uint32_t cbPattern);
0301 char * Railgun_Doublet (char * pbTarget, char * pbPattern, uint32_t cbTarget, uint32_t cbPattern);
0302
0303 // Min_Match_Length=THRESHOLD=4 means 4 and bigger are to be encoded:
0304
0305 //#define Min_Match_BAILOUT_Length (4)
0306 //#define Min_Match_Length (4)
0307 //#define OffsetBITS (7)
0308 //#define LengthBITS (0)
0309
0310 #define Min_Match_BAILOUT_Length (8)
0311 #define Min_Match_Length (8)
0312 #define OffsetBITS (7)
0313 #define LengthBITS (0)
0314
0315 //#define Min_Match_BAILOUT_Length (16)
0316 //#define Min_Match_Length (16)
0317 //#define OffsetBITS (7)
0318 //#define LengthBITS (0)
0319
0320 // When Min_Match_BAILOUT_Length is 4:
0321 /*
0322 D:\_KAZE\_KAZE_GOLD\Nakamichi_project\Nakamichi_r1-RSSBO++7bit>"Nakamichi_r1-RSSBO++_1GB_7bit.exe" Kazahana_on.PAGODA-order-5.txt.Nakamichi
0323 Nakamichi, revision 1-RSSBO++_1GB_7bit, written by Kaze, based on Nobuo Ito's LZSS source.
0324 Decompressing 309409104 bytes ...
0325 RAM-to-RAM performance: 501 MB/s.
0326 */
0327
0328 // When Min_Match_BAILOUT_Length is 8:
0329 /*
0330 D:\_KAZE\_KAZE_GOLD\Nakamichi_project\Nakamichi_r1-RSSBO++7bit>"Nakamichi_r1-RSSBO++_1GB_7bit.exe" Kazahana_on.PAGODA-order-5.txt.Nakamichi
0331 Nakamichi, revision 1-RSSBO++_1GB_7bit, written by Kaze, based on Nobuo Ito's LZSS source.
0332 Decompressing 243155042 bytes ...
0333 RAM-to-RAM performance: 706 MB/s.
0334 */
0335
0336 // When Min_Match_BAILOUT_Length is 16:
0337 /*
0338 D:\_KAZE\_KAZE_GOLD\Nakamichi_project\Nakamichi_r1-RSSBO++7bit>"Nakamichi_r1-RSSBO++_1GB_7bit.exe" Kazahana_on.PAGODA-order-5.txt.Nakamichi
0339 Nakamichi, revision 1-RSSBO++_1GB_7bit, written by Kaze, based on Nobuo Ito's LZSS source.
0340 Decompressing 240128664 bytes ...
0341 RAM-to-RAM performance: 748 MB/s.
0342
0343 D:\_KAZE\_KAZE_GOLD\Nakamichi_project\Nakamichi_r1-RSSBO++7bit>Yappy.exe Kazahana_on.PAGODA-order-5.txt 128
0344 YAPPY: [b 0K] bytes 846351894 -> 394516993  46.6%  comp   6.3 MB/s  uncomp 662.1 MB/s
0345
0346 D:\_KAZE\_KAZE_GOLD\Nakamichi_project\Nakamichi_r1-RSSBO++7bit>Yappy.exe Kazahana_on.PAGODA-order-5.txt 256
0347 YAPPY: [b 0K] bytes 846351894 -> 298614189  35.3%  comp  27.7 MB/s  uncomp 760.0 MB/s
0348
0349 D:\_KAZE\_KAZE_GOLD\Nakamichi_project\Nakamichi_r1-RSSBO++7bit>Yappy.exe Kazahana_on.PAGODA-order-5.txt 512
0350 YAPPY: [b 0K] bytes 846351894 -> 247846152  29.3%  comp  31.3 MB/s  uncomp 875.4 MB/s
0351
0352 D:\_KAZE\_KAZE_GOLD\Nakamichi_project\Nakamichi_r1-RSSBO++7bit>Yappy.exe Kazahana_on.PAGODA-order-5.txt 1024
0353 YAPPY: [b 1K] bytes 846351894 -> 219625842  25.9%  comp  33.9 MB/s  uncomp 898.3 MB/s
0354
0355 D:\_KAZE\_KAZE_GOLD\Nakamichi_project\Nakamichi_r1-RSSBO++7bit>Yappy.exe Kazahana_on.PAGODA-order-5.txt 2048
0356 YAPPY: [b 2K] bytes 846351894 -> 202655189  23.9%  comp  33.9 MB/s  uncomp 875.4 MB/s
0357
0358 D:\_KAZE\_KAZE_GOLD\Nakamichi_project\Nakamichi_r1-RSSBO++7bit>Yappy.exe Kazahana_on.PAGODA-order-5.txt 4096
0359 YAPPY: [b 4K] bytes 846351894 -> 191149889  22.6%  comp  34.1 MB/s  uncomp 890.9 MB/s
0360
0361 D:\_KAZE\_KAZE_GOLD\Nakamichi_project\Nakamichi_r1-RSSBO++7bit>Yappy.exe Kazahana_on.PAGODA-order-5.txt 16384
0362 YAPPY: [b 16K] bytes 846351894 -> 180650931  21.3%  comp  34.6 MB/s  uncomp 914.1 MB/s
0363
0364 D:\_KAZE\_KAZE_GOLD\Nakamichi_project\Nakamichi_r1-RSSBO++7bit>Yappy.exe Kazahana_on.PAGODA-order-5.txt 32768
0365 YAPPY: [b 32K] bytes 846351894 -> 178902966  21.1%  comp  34.6 MB/s  uncomp 914.1 MB/s
0366 */
0367
0368 //12bit
0369 //#define REF_SIZE (4095+Min_Match_Length)
0370 #define REF_SIZE ( ((1<<OffsetBITS)-1) + Min_Match_Length )
0371 //3bit
0372 //#define ENC_SIZE (7+Min_Match_Length)
0373 //#define ENC_SIZE ( ((1<<LengthBITS)-1) + Min_Match_Length )
0374 // Caramba, still don't feel the full picture about LookAheadBuffer!
0375 #define ENC_SIZE ( ((1<<LengthBITS)-1) + Min_Match_Length )
0376
0377 int main( int argc, char *argv[] ) {
0378       FILE *fp;
0379       int SourceSize;
0380       int TargetSize;
0381       char* SourceBlock=NULL;
0382       char* TargetBlock=NULL;
0383       char* Nakamichi = ".Nakamichi\0";
0384       char NewFileName[256];
0385       clock_t clocks1, clocks2;
0386
0387       printf("Nakamichi, revision 1-RSSBO++_1GB_7bit, written by Kaze, based on Nobuo Ito's LZSS source.\n");
```

```
0388        if (argc==1) {
0389                    printf("Usage: Nakamichi filename\n"); exit(13);
0390        }
0391        if ((fp = fopen(argv[1], "rb")) == NULL) {
0392                    printf("Nakamichi: Can't open '%s' file.\n", argv[1]); exit(13);
0393        }
0394        fseek(fp, 0, SEEK_END);
0395        SourceSize = ftell(fp);
0396        fseek(fp, 0, SEEK_SET);
0397        // If filename ends in '.Nakamichi' then mode is decompression otherwise compression.
0398        if (strcmp(argv[1]+(strlen(argv[1])-strlen(Nakamichi)), Nakamichi) == 0) {
0399        SourceBlock = (char*)malloc(SourceSize+512);
0400        //TargetBlock = (char*)malloc(5*SourceSize+512);
0401        TargetBlock = (char*)malloc(1024*1024*1024+512);
0402        fread(SourceBlock, 1, SourceSize, fp);
0403        fclose(fp);
0404                    printf("Decompressing %d bytes ...\n", SourceSize );
0405                    clocks1 = clock();
0406                    TargetSize = Decompress(TargetBlock, SourceBlock, SourceSize);
0407                    clocks2 = clock();
0408                    printf("RAM-to-RAM performance: %d MB/s.\n", ((TargetSize/(clocks2 - clocks1 + 1))*(long)1000)>>20);
0409                    strcpy(NewFileName, argv[1]);
0410                    *( NewFileName + strlen(argv[1])-strlen(Nakamichi) ) = '\0';
0411        } else {
0412        SourceBlock = (char*)malloc(SourceSize+512);
0413        TargetBlock = (char*)malloc(SourceSize+512);
0414        fread(SourceBlock, 1, SourceSize, fp);
0415        fclose(fp);
0416                    printf("Compressing %d bytes ...\n", SourceSize );
0417                    clocks1 = clock();
0418                    TargetSize = Compress(TargetBlock, SourceBlock, SourceSize);
0419                    clocks2 = clock();
0420                    printf("RAM-to-RAM performance: %d KB/s.\n", ((SourceSize/(clocks2 - clocks1 + 1))*(long)1000)>>10);
0421                    strcpy(NewFileName, argv[1]);
0422                    strcat(NewFileName, Nakamichi);
0423        }
0424        if ((fp = fopen(NewFileName, "wb")) == NULL) {
0425                    printf("Nakamichi: Can't write '%s' file.\n", NewFileName); exit(13);
0426        }
0427        fwrite(TargetBlock, 1, TargetSize, fp);
0428        fclose(fp);
0429        free(TargetBlock);
0430        free(SourceBlock);
0431        exit(0);
0432 }
0433
0434 void SearchIntoSlidingWindow(unsigned int* retIndex, unsigned int* retMatch, char* refStart,char* refEnd,char* encStart,char* encEnd){
0435        char* FoundAtPosition;
0436        unsigned int match=0;
0437        *retIndex=0;
0438        *retMatch=0;
0439 #ifdef ReplaceBruteForceWithRailgunSwampshineBailOut
0440        if (refStart < refEnd) {
0441 // With 'Railgun_Swampshine_BailOut':
0442 // D:\_KAZE\_KAZE_GOLD\Nakamichi_project\Nakamichi_r1-RSSBO++7bit>"Nakamichi_r1-RSSBO++_1GB_7bit.exe" Kazahana_on.PAGODA-order-5.txt
0443 // Nakamichi, revision 1-RSSBO++_1GB_7bit, written by Kaze, based on Nobuo Ito's LZSS source.
0444 // Compressing 846351894 bytes ...
0445 // /; Each rotation means 128KB are encoded; Done 100%
0446 // RAM-to-RAM performance: 750 KB/s.
0447                    //FoundAtPosition = Railgun_Swampshine_BailOut(refStart, encStart, (uint32_t)(refEnd-refStart), Min_Match_BAILOUT_Length);
0448 // With 'Railgun_Doublet':
0449 // D:\_KAZE\_KAZE_GOLD\Nakamichi_project\Nakamichi_r1-RSSBO++7bit>"Nakamichi_r1-RSSBO++_1GB_7bit.exe" Kazahana_on.PAGODA-order-5.txt
0450 // Nakamichi, revision 1-RSSBO++_1GB_7bit, written by Kaze, based on Nobuo Ito's LZSS source.
0451 // Compressing 846351894 bytes ...
0452 // /; Each rotation means 128KB are encoded; Done 100%
0453 // RAM-to-RAM performance: 26781 KB/s.
0454                    FoundAtPosition = Railgun_Doublet(refStart, encStart, (uint32_t)(refEnd-refStart), Min_Match_BAILOUT_Length);
0455                    if (FoundAtPosition!=NULL) {
0456                                *retMatch=Min_Match_BAILOUT_Length;
0457                                *retIndex=refEnd-FoundAtPosition;
0458                    }
0459        }
0460 #else
0461        while(refStart < refEnd){
0462                    match=SlidingWindowVsLookAheadBuffer(refStart,refEnd,encStart,encEnd);
0463                    if(match > *retMatch){
0464                                *retMatch=match;
0465                                *retIndex=refEnd-refStart;
0466                    }
0467                    if(*retMatch >= Min_Match_BAILOUT_Length) break;
0468                    refStart++;
0469        }
0470 #endif
0471 }
0472
0473 unsigned int SlidingWindowVsLookAheadBuffer( char* refStart, char* refEnd, char* encStart,char* encEnd){
0474        int ret = 0;
0475        while(refStart[ret] == encStart[ret]){
0476                    if(&refStart[ret] >= refEnd) break;
```

```
0477                    if(&encStart[ret] >= encEnd) break;
0478                    ret++;
0479                    if(ret >= Min_Match_BAILOUT_Length) break;
0480        }
0481        return ret;
0482 }
0483
0484 unsigned int Compress(char* ret, char* src, unsigned int srcSize){
0485        unsigned int srcIndex=0;
0486        unsigned int retIndex=0;
0487        unsigned int index=0;
0488        unsigned int match=0;
0489        unsigned int notMatch=0;
0490        unsigned char* notMatchStart=NULL;
0491        char* refStart=NULL;
0492        char* encEnd=NULL;
0493        int Melnitchka=0;
0494        char *Auberge[4] = {"|\0","/\0","-\0","\\\0"};
0495        int ProgressIndicator;
0496
0497        while(srcIndex < srcSize){
0498                    if(srcIndex>=REF_SIZE)
0499                                refStart=&src[srcIndex-REF_SIZE];
0500                    else
0501                                refStart=src;
0502                    if(srcIndex>=srcSize-ENC_SIZE)
0503                                encEnd=&src[srcSize];
0504                    else
0505                                encEnd=&src[srcIndex+ENC_SIZE];
0506
0507                    SearchIntoSlidingWindow(&index,&match,refStart,&src[srcIndex],&src[srcIndex],encEnd);
0508                    //if ( match<Min_Match_Length ) {
0509                    //if ( match<Min_Match_Length || match<8 ) {
0510                    if ( match==0 ) {
0511                                if(notMatch==0){
0512                                            notMatchStart=&ret[retIndex];
0513                                            retIndex++;
0514                                }
0515                                else if (notMatch==127) {
0516                                            *notMatchStart=(unsigned char)((127)<<1);
0517                                            notMatch=0;
0518                                            notMatchStart=&ret[retIndex];
0519                                            retIndex++;
0520                                }
0521                                ret[retIndex]=src[srcIndex];
0522                                retIndex++;
0523                                notMatch++;
0524                                srcIndex++;
0525                                if ((srcIndex-1) % (1<<17) > srcIndex % (1<<17)) {
0526                                            ProgressIndicator = (int)( (srcIndex+1)*(float)100/(srcSize+1) );
0527                                            printf("%s; Each rotation means 128KB are encoded; Done %d%%\r", Auberge[Melnitchka++], ProgressIndicator );
0528                                            Melnitchka = Melnitchka & 3; // 0 1 2 3: 00 01 10 11
0529                                }
0530                    } else {
0531                                if(notMatch > 0){
0532                                            *notMatchStart=(unsigned char)((notMatch)<<1);
0533                                            notMatch=0;
0534                                }
0535 // ---------------------|
0536 //                     \ /
0537
0538                                //ret[retIndex] = 0x80; // Assuming seventh/fifteenth bit is zero i.e. LONG MATCH i.e. Min_Match_BAILOUT_Length*4
0539                                //if ( match==Min_Match_BAILOUT_Length ) ret[retIndex] = 0xC0; // 8bit&7bit set, SHORT MATCH if seventh/fifteenth bit is not zero i.e.
Min_Match_BAILOUT_Length
0540 //                     / \
0541 // ---------------------|
0542                                ret[retIndex] = 0x01; // Assuming seventh/fifteenth bit is zero i.e. LONG MATCH i.e. Min_Match_BAILOUT_Length*4
0543 //                             if ( match==Min_Match_BAILOUT_Length ) ret[retIndex] = 0x03; // 2bit&1bit set, SHORT MATCH if 2bit is not zero i.e.
Min_Match_BAILOUT_Length
0544                                // 1bit+3bits+12bits:
0545                                //ret[retIndex] = ret[retIndex] | ((match-Min_Match_Length)<<4);
0546                                //ret[retIndex] = ret[retIndex] | (((index-Min_Match_Length) & 0x0F00)>>8);
0547                                // 1bit+1bit+14bits:
0548                                //ret[retIndex] = ret[retIndex] | ((match-Min_Match_Length)<<(8-(LengthBITS+1))); // No need to set the matchlength
0549 // The fragment below is outrageously ineffective - instead of 8bit&7bit I have to use the lower TWO bits i.e. 2bit&1bit as flags, thus in decompressing one WORD
can be fetched instead of two BYTE loads followed by SHR by 2.
0550 // ---------------------|
0551 //                     \ /
0552                                //ret[retIndex] = ret[retIndex] | (((index-Min_Match_Length) & 0x3F00)>>8); // 2+4+8=14
0553                                //retIndex++;
0554                                //ret[retIndex] = (char)((index-Min_Match_Length) & 0x00FF);
0555                                //retIndex++;
0556 //                     / \
0557 // ---------------------|
0558                                // Now the situation is like LOW:HIGH i.e. FF:3F i.e. 0x3FFF, 16bit&15bit used as flags,
0559                                // should become LOW:HIGH i.e. FC:FF i.e. 0xFFFC, 2bit&1bit used as flags.
0560 //                             ret[retIndex] = ret[retIndex] | (((index-Min_Match_Length) & 0x00FF)<<2); // 2+4+8=14
0561 //                             retIndex++;
0562 //                             ret[retIndex] = (char)(((index-Min_Match_Length) & 0x3FFF)>>6);
```

```
0563 //                          retIndex++;
0564                            ret[retIndex] = ret[retIndex] | (((index-Min_Match_Length) & 0x00FF)<<1);
0565                            retIndex++;
0566 //                    / \
0567 // --------------------|
0568                            srcIndex+=match;
0569                            if ((srcIndex-match) % (1<<17) > srcIndex % (1<<17)) {
0570                                    ProgressIndicator = (int)( (srcIndex+1)*(float)100/(srcSize+1) );
0571                                    printf("%s; Each rotation means 128KB are encoded; Done %d%%\r", Auberge[Melnitchka++], ProgressIndicator );
0572                                    Melnitchka = Melnitchka & 3; // 0 1 2 3: 00 01 10 11
0573                            }
0574                    }
0575        }
0576        if(notMatch > 0){
0577                    *notMatchStart=(unsigned char)((notMatch)<<1);
0578        }
0579        printf("%s; Each rotation means 128KB are encoded; Done %d%%\n", Auberge[Melnitchka], 100 );
0580        if (retIndex % 2) {ret[retIndex] = 0x00; retIndex++;} // PADDING: Literal of length ZERO
0581        return retIndex;
0582 }
0583
0584 unsigned int Decompress(char* ret, char* src, unsigned int srcSize){
0585        unsigned int srcIndex=0;
0586        unsigned int retIndex=0;
0587        unsigned int index=0;
0588        unsigned int match=0;
0589        unsigned int WORDpair;
0590
0591        while(srcIndex < srcSize){
0592                    //if((unsigned char)src[srcIndex] <= 127){
0593                    WORDpair = *(unsigned short int*)&src[srcIndex]; // CAUTION: Since r.7bit OFFSET/LENGTH are no longer 2/2 bytes each but 1/2, so padding by MOD 2
is needed i.e. one ASCII 000 char has to be the last byte.
0594                    if((WORDpair & 0x01) == 0){
0595                            memcpy(&ret[retIndex],&src[srcIndex+1],(WORDpair & 0xFF)>>1); // Use padding and replace 'memcpy' with loop of 4 or 4+4 transfers/stores
i.e. *()=DWORD
0596                            retIndex+=(WORDpair & 0xFF)>>1;
0597                            srcIndex+=(((WORDpair & 0xFF)>>1)+1);
0598                    }
0599                    else{
0600                            // 1bit+3bits+12bits:
0601                            //match = ((src[srcIndex] & 0x7F) >> 4)+Min_Match_Length;
0602                            //index = (src[srcIndex] & 0x0F) << 8;
0603                            // 1bit+1bit+14bits:
0604                            //match = ((src[srcIndex] & 0x4F) >> 4)+Min_Match_Length; // In fact, not needed when eightfoldness is commenced, match is 8.
0605 // The fragment below is outrageously ineffective - it can be done in one WORD operation instead of two BYTE operations.
0606 // --------------------|
0607 //                    \ /
0608                            //index = (src[srcIndex] & 0x3F) << 8;
0609                            //srcIndex++;
0610                            //index = (index | (unsigned int)(0x00FF & src[srcIndex])) + Min_Match_Length;
0611                            //srcIndex++;
0612 //                    / \
0613 // --------------------|
0614 // --------------------|
0615 //                    \ /
0616 //                          index = (WORDpair>>2) + Min_Match_Length;
0617 //                          srcIndex=srcIndex+2;
0618                            index = ((WORDpair & 0xFF)>>1) + Min_Match_Length;
0619                            srcIndex=srcIndex+1;
0620 //                    / \
0621 // --------------------|
0622                            //if (src[srcIndex-1-1] & 0x40) { // 4 if seventh/fifteenth bit is not zero
0623 //                          if (WORDpair & 0x02) { // 4 if 2/14 bit is not zero
0624 //                                  match = Min_Match_BAILOUT_Length;
0625 //                                  //memcpy(&ret[retIndex],&ret[retIndex-index],match);
0626 //                                  *(uint32_t*)(ret+retIndex) = *(uint32_t*)(ret+retIndex-index);
0627 //                          } else {
0628 //                                  match = Min_Match_BAILOUT_Length*4;
0629 //                                  //*(uint32_t*)(ret+retIndex) = *(uint32_t*)(ret+retIndex-index);
0630 //                                  //*(uint32_t*)(ret+retIndex+4) = *(uint32_t*)(ret+retIndex-index+4);
0631 //                                  //*(uint32_t*)(ret+retIndex+8) = *(uint32_t*)(ret+retIndex-index+8);
0632 //                                  //*(uint32_t*)(ret+retIndex+12) = *(uint32_t*)(ret+retIndex-index+12);
0633 //                                  SlowCopy128bit((ret+retIndex-index), (ret+retIndex));
0634 //                          }
0635                            match = Min_Match_BAILOUT_Length;
0636                            //*(uint32_t*)(ret+retIndex) = *(uint32_t*)(ret+retIndex-index);
0637                            *(uint64_t*)(ret+retIndex) = *(uint64_t*)(ret+retIndex-index);
0638                            //SlowCopy128bit((ret+retIndex-index), (ret+retIndex));
0639                            retIndex+=match;
0640                    }
0641        }
0642        return retIndex;
0643 }
0644
0645 // Decompression main loop, 71-2a+2=73 bytes long:
0646 /*
0647 ; mark_description "Intel(R) C++ Compiler XE for applications running on IA-32, Version 12.1.1.258 Build 20111011";
0648 ; mark_description "-O3 -FAcs";
0649 .B6.3:
```

```
0650 ;;;              //if((unsigned char)src[srcIndex] <= 127){
0651 ;;;              WORDpair = *(unsigned short int*)&src[srcIndex]; // CAUTION: Since r.7bit OFFSET/LENGTH are no longer 2/2 bytes each but 1/2, so padding by MOD 2
is needed i.e. one ASCII 000 char has to be the last byte.
0652   0002a 8d 34 1f         lea esi, DWORD PTR [edi+ebx]
0653   0002d 0f b7 06         movzx eax, WORD PTR [esi]
0654 ;;;              if((WORDpair & 0x01) == 0){
0655 ;;;                 memcpy(&ret[retIndex],&src[srcIndex+1],(WORDpair & 0xFF)>>1); // Use padding and replace 'memcpy' with loop of 4 or 4+4 transfers/stores
i.e. *()=DWORD
0656   00030 0f b6 f0         movzx esi, al
0657   00033 d1 ee            shr esi, 1
0658   00035 a8 01            test al, 1
0659   00037 74 18            je .B6.5
0660 .B6.4:
0661 ;;;                    retIndex+=(WORDpair & 0xFF)>>1;
0662 ;;;                    srcIndex+=(((WORDpair & 0xFF)>>1)+1);
0663 ;;;              }
0664 ;;;              else{
0665 ;;;                    // 1bit+3bits+12bits:
0666 ;;;                    //match = ((src[srcIndex] & 0x7F) >> 4)+Min_Match_Length;
0667 ;;;                    //index = (src[srcIndex] & 0x0F) << 8;
0668 ;;;                    // 1bit+1bit+14bits:
0669 ;;;                    //match = ((src[srcIndex] & 0x4F) >> 4)+Min_Match_Length; // In fact, not needed when eightfoldness is commenced, match is 8.
0670 ;;; // The fragment below is outrageously ineffective - it can be done in one WORD operation instead of two BYTE operations.
0671 ;;; // --------------------|
0672 ;;; //                     \ /
0673 ;;;                      //index = (src[srcIndex] & 0x3F) << 8;
0674 ;;;                      //srcIndex++;
0675 ;;;                      //index = (index | (unsigned int)(0x00FF & src[srcIndex])) + Min_Match_Length;
0676 ;;;                      //srcIndex++;
0677 ;;; //                   / \
0678 ;;; // --------------------|
0679 ;;; // --------------------|
0680 ;;; //                     \ /
0681 ;;; //                     index = (WORDpair>>2) + Min_Match_Length;
0682 ;;; //                     srcIndex=srcIndex+2;
0683 ;;;                      index = ((WORDpair & 0xFF)>>1) + Min_Match_Length;
0684 ;;;                      srcIndex=srcIndex+1;
0685 ;;; //                   / \
0686 ;;; // --------------------|
0687 ;;;                      //if (src[srcIndex-1-1] & 0x40) { // 4 if seventh/fifteenth bit is not zero
0688 ;;; //                   if (WORDpair & 0x02) { // 4 if 2/14 bit is not zero
0689 ;;; //                             match = Min_Match_BAILOUT_Length;
0690 ;;; //                             //memcpy(&ret[retIndex],&ret[retIndex-index],match);
0691 ;;; //                             *(uint32_t*)(ret+retIndex) = *(uint32_t*)(ret+retIndex-index);
0692 ;;; //                   } else {
0693 ;;; //                             match = Min_Match_BAILOUT_Length*4;
0694 ;;; //                             //*(uint32_t*)(ret+retIndex) = *(uint32_t*)(ret+retIndex-index);
0695 ;;; //                             //*(uint32_t*)(ret+retIndex+4) = *(uint32_t*)(ret+retIndex-index+4);
0696 ;;; //                             //*(uint32_t*)(ret+retIndex+8) = *(uint32_t*)(ret+retIndex-index+8);
0697 ;;; //                             //*(uint32_t*)(ret+retIndex+12) = *(uint32_t*)(ret+retIndex-index+12);
0698 ;;; //                             SlowCopy128bit((ret+retIndex-index), (ret+retIndex));
0699 ;;; //                   }
0700 ;;;                      match = Min_Match_BAILOUT_Length;
0701 ;;;                      //*(uint32_t*)(ret+retIndex) = *(uint32_t*)(ret+retIndex-index);
0702 ;;;                      *(uint64_t*)(ret+retIndex) = *(uint64_t*)(ret+retIndex-index);
0703   00039 f7 de            neg esi
0704   0003b 8d 14 29         lea edx, DWORD PTR [ecx+ebp]
0705   0003e 03 f2            add esi, edx
0706   00040 43               inc ebx
0707 ;;;                      //SlowCopy128bit((ret+retIndex-index), (ret+retIndex));
0708 ;;;                      retIndex+=match;
0709   00041 83 c5 08         add ebp, 8
0710   00044 8b 46 f8         mov eax, DWORD PTR [-8+esi]
0711   00047 8b 76 fc         mov esi, DWORD PTR [-4+esi]
0712   0004a 89 02            mov DWORD PTR [edx], eax
0713   0004c 89 72 04         mov DWORD PTR [4+edx], esi
0714   0004f eb 1c            jmp .B6.7
0715 .B6.5:
0716   00051 56               push esi
0717   00052 8d 44 1f 01      lea eax, DWORD PTR [1+edi+ebx]
0718   00056 50               push eax
0719   00057 8d 14 29         lea edx, DWORD PTR [ecx+ebp]
0720   0005a 52               push edx
0721   0005b e8 fc ff ff ff   call __intel_fast_memcpy
0722 .B6.12:
0723   00060 8b 4c 24 20      mov ecx, DWORD PTR [32+esp]
0724   00064 83 c4 0c         add esp, 12
0725 .B6.6:
0726   00067 03 ee            add ebp, esi
0727   00069 8d 5c 33 01      lea ebx, DWORD PTR [1+ebx+esi]
0728 .B6.7:
0729   0006d 3b 5c 24 1c      cmp ebx, DWORD PTR [28+esp]
0730   00071 72 b7            jb .B6.3
0731 */
0732
0733 // The Speed Showdown on my 'Bonboniera' laptop (Core 2 T7500 2200MHz, Windows 7 64bit):
0734 // Grrr, in round #1 Yappy kicked my ass, yet, 4 more rounds remain...
0735 /*
0736 D:\_KAZE\Nakamichi_r1-RSSBO>NakamichiVsYappy.bat
```

```
Speed Showdown: Nakamichi VS Yappy, both compiled with Intel v12.1 ...

D:\_KAZE\Nakamichi_r1-RSSBO>Yappy.exe enwiki-20140304-pages-articles.7z.001 1024
YAPPY: [b 1K] bytes 104857600 -> 78039768  74.4%  comp  42.7 MB/s  uncomp 680.3 MB/s

D:\_KAZE\Nakamichi_r1-RSSBO>Yappy.exe enwiki-20140304-pages-articles.7z.001 2048
YAPPY: [b 2K] bytes 104857600 -> 70845249  67.6%  comp  39.8 MB/s  uncomp 623.3 MB/s

D:\_KAZE\Nakamichi_r1-RSSBO>Yappy.exe enwiki-20140304-pages-articles.7z.001 4096
YAPPY: [b 4K] bytes 104857600 -> 64270963  61.3%  comp  36.2 MB/s  uncomp 583.1 MB/s

D:\_KAZE\Nakamichi_r1-RSSBO>Yappy.exe enwiki-20140304-pages-articles.7z.001 8192
YAPPY: [b 8K] bytes 104857600 -> 59756354  57.0%  comp  32.9 MB/s  uncomp 549.5 MB/s

D:\_KAZE\Nakamichi_r1-RSSBO>Yappy.exe enwiki-20140304-pages-articles.7z.001 16384
YAPPY: [b 16K] bytes 104857600 -> 57497885  54.8%  comp  31.4 MB/s  uncomp 541.5 MB/s

D:\_KAZE\Nakamichi_r1-RSSBO>Yappy.exe enwiki-20140304-pages-articles.7z.001 32768
YAPPY: [b 32K] bytes 104857600 -> 56365696  53.8%  comp  31.1 MB/s  uncomp 541.5 MB/s

D:\_KAZE\Nakamichi_r1-RSSBO>Yappy.exe enwiki-20140304-pages-articles.7z.001 65536
YAPPY: [b 64K] bytes 104857600 -> 55799079  53.2%  comp  31.3 MB/s  uncomp 534.3 MB/s

D:\_KAZE\Nakamichi_r1-RSSBO>Yappy.exe enwiki-20140304-pages-articles.7z.001 262144
YAPPY: [b 256K] bytes 104857600 -> 55377127  52.8%  comp  31.4 MB/s  uncomp 534.3 MB/s

D:\_KAZE\Nakamichi_r1-RSSBO>Nakamichi.exe enwiki-20140304-pages-articles.7z.001
Nakamichi, revision 1-RSSBO, written by Kaze.
Compressing 104857600 bytes ...
-; Each rotation means 128KB are encoded; Done 100%
RAM-to-RAM performance: 198 KB/s.

D:\_KAZE\Nakamichi_r1-RSSBO>Nakamichi.exe enwiki-20140304-pages-articles.7z.001.Nakamichi
Nakamichi, revision 1-RSSBO, written by Kaze.
Decompressing 70533827 bytes ...
RAM-to-RAM performance: 531 MB/s.

D:\_KAZE\Nakamichi_r1-RSSBO>
*/

// In my opinion Hamid Buzidi is the best, therefore his lzturbo v1.1 reference results are given below:
/*
D:\_KAZE\Nakamichi_r1-RSSBO>timer32 lzturbo.exe -19 -p0 enwiki-20140304-pages-articles.7z.001 .

Kernel  Time =     0.982 =    0%
User    Time =   152.537 =   99%
Process Time =   153.520 =  100%    Virtual  Memory =    429 MB
Global  Time =   153.519 =  100%    Physical Memory =    407 MB

D:\_KAZE\Nakamichi_r1-RSSBO>timer32.exe lzturbo.exe -d enwiki-20140304-pages-articles.7z.001.lzt .

Kernel  Time =     0.234 =   62%
User    Time =     0.187 =   50%
Process Time =     0.421 =  112%    Virtual  Memory =     98 MB
Global  Time =     0.374 =  100%    Physical Memory =     70 MB

D:\_KAZE\Nakamichi_r1-RSSBO>dir

04/15/2014  08:05 AM       104,857,600 enwiki-20140304-pages-articles.7z.001
04/15/2014  08:04 AM        41,984,881 enwiki-20140304-pages-articles.7z.001.lzt

D:\_KAZE\Nakamichi_r1-RSSBO>timer32 lzturbo.exe -11 -p0 enwiki-20140304-pages-articles.7z.001 .

Kernel  Time =     0.171 =    9%
User    Time =     1.622 =   90%
Process Time =     1.794 =  100%    Virtual  Memory =     58 MB
Global  Time =     1.794 =  100%    Physical Memory =     39 MB

D:\_KAZE\Nakamichi_r1-RSSBO>timer32.exe lzturbo.exe -d enwiki-20140304-pages-articles.7z.001.lzt .

Kernel  Time =     0.249 =   41%
User    Time =     0.140 =   23%
Process Time =     0.390 =   64%    Virtual  Memory =     98 MB
Global  Time =     0.608 =  100%    Physical Memory =     73 MB

D:\_KAZE\Nakamichi_r1-RSSBO>dir

04/15/2014  08:05 AM       104,857,600 enwiki-20140304-pages-articles.7z.001
04/15/2014  08:05 AM        47,685,453 enwiki-20140304-pages-articles.7z.001.lzt

D:\_KAZE\Nakamichi_r1-RSSBO>
*/

// Railgun_Swampshine_BailOut, copyleft 2014-Jan-31, Kaze.
// Caution: For better speed the case 'if (cbPattern==1)' was removed, so Pattern must be longer than 1 char.
#define NeedleThreshold2vs4swampLITE 9+10 // Should be bigger than 9. BMH2 works up to this value (inclusive), if bigger then BMH4 takes over.
char * Railgun_Swampshine_BailOut (char * pbTarget, char * pbPattern, uint32_t cbTarget, uint32_t cbPattern)
{
      char * pbTargetMax = pbTarget + cbTarget;
```

```
0826	register uint32_t ulHashPattern;
0827	signed long count;
0828
0829	unsigned char bm_Horspool_Order2[256*256]; // Bitwise soon...
0830	uint32_t i, Gulliver;
0831
0832	uint32_t PRIMALposition, PRIMALpositionCANDIDATE;
0833	uint32_t PRIMALength, PRIMALengthCANDIDATE;
0834	uint32_t j, FoundAtPosition;
0835
0836	if (cbPattern > cbTarget) return(NULL);
0837
0838	if ( cbPattern<4 ) {
0839		// SSE2 i.e. 128bit Assembly rules here:
0840		// ...
0841		pbTarget = pbTarget+cbPattern;
0842		ulHashPattern = ( (*(char *)(pbPattern))<<8 ) + *(pbPattern+(cbPattern-1));
0843		if ( cbPattern==3 ) {
0844			for ( ;; ) {
0845				if ( ulHashPattern == ( (*(char *)(pbTarget-3))<<8 ) + *(pbTarget-1) ) {
0846					if ( *(char *)(pbPattern+1) == *(char *)(pbTarget-2) ) return((pbTarget-3));
0847				}
0848				if ( (char)(ulHashPattern>>8) != *(pbTarget-2) ) {
0849					pbTarget++;
0850					if ( (char)(ulHashPattern>>8) != *(pbTarget-2) ) pbTarget++;
0851				}
0852				pbTarget++;
0853				if (pbTarget > pbTargetMax) return(NULL);
0854			}
0855		} else {
0856		}
0857		for ( ;; ) {
0858			if ( ulHashPattern == ( (*(char *)(pbTarget-2))<<8 ) + *(pbTarget-1) ) return((pbTarget-2));
0859			if ( (char)(ulHashPattern>>8) != *(pbTarget-1) ) pbTarget++;
0860			pbTarget++;
0861			if (pbTarget > pbTargetMax) return(NULL);
0862		}
0863	} else { //if ( cbPattern<4 )
0864		if ( cbPattern<=NeedleThreshold2vs4swampLITE ) {
0865			// BMH order 2, needle should be >=4:
0866			ulHashPattern = *(uint32_t *)(pbPattern); // First four bytes
0867			for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]=0;}
0868			for (i=0; i < cbPattern-1; i++) bm_Horspool_Order2[*(unsigned short *)(pbPattern+i)]=1;
0869			i=0;
0870			while (i <= cbTarget-cbPattern) {
0871				Gulliver = 1; // 'Gulliver' is the skip
0872				if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+cbPattern-1-1]] != 0 ) {
0873					if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+cbPattern-1-1-2]] == 0 ) Gulliver = cbPattern-(2-1)-2; else
{
0874						if ( *(uint32_t *)&pbTarget[i] == ulHashPattern) { // This fast check ensures not missing a match (for
remainder) when going under 0 in loop below:
0875							count = cbPattern-4+1;
0876							while ( count > 0 && *(uint32_t *)(pbPattern+count-1) == *(uint32_t *)(&pbTarget[i]+(count-
1)) )
0877								count = count-4;
0878							if ( count <= 0 ) return(pbTarget+i);
0879						}
0880					}
0881				} else Gulliver = cbPattern-(2-1);
0882				i = i + Gulliver;
0883				//GlobalI++; // Comment it, it is only for stats.
0884			}
0885			return(NULL);
0886		} else { // if ( cbPattern<=NeedleThreshold2vs4swampLITE )
0887
0888 // Swampwalker_BAILOUT heuristic order 4 (Needle should be bigger than 4) [
0889 // Needle: 1234567890qwertyuiopasdfghjklzxcv           PRIMALposition=01 PRIMALength=33  '1234567890qwertyuiopasdfghjklzxcv'
0890 // Needle: vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv           PRIMALposition=29 PRIMALength=04  'vvvv'
0891 // Needle: vvvvvvvvvvBOOMSHAKALAKAvvvvvvvvvv          PRIMALposition=08 PRIMALength=20  'vvvBOOMSHAKALAKAvvvv'
0892 // Needle: Trollland                                  PRIMALposition=01 PRIMALength=09  'Trollland'
0893 // Needle: Swampwalker                                PRIMALposition=01 PRIMALength=11  'Swampwalker'
0894 // Needle: licenselessness                            PRIMALposition=01 PRIMALength=15  'licenselessness'
0895 // Needle: alfalfa                                    PRIMALposition=02 PRIMALength=06  'lfalfa'
0896 // Needle: Sandokan                                   PRIMALposition=01 PRIMALength=08  'Sandokan'
0897 // Needle: shazamish                                  PRIMALposition=01 PRIMALength=09  'shazamish'
0898 // Needle: Simplicius Simplicissimus                  PRIMALposition=06 PRIMALength=20  'icius Simplicissimus'
0899 // Needle: domilliaquadringenquattuorquinquagintillion  PRIMALposition=01 PRIMALength=32  'domilliaquadringenquattuorquinqu'
0900 // Needle: boom-boom                                  PRIMALposition=02 PRIMALength=08  'oom-boom'
0901 // Needle: vvvvv                                      PRIMALposition=01 PRIMALength=04  'vvvv'
0902 // Needle: 12345                                      PRIMALposition=01 PRIMALength=05  '12345'
0903 // Needle: likey-likey                                PRIMALposition=03 PRIMALength=09  'key-likey'
0904 // Needle: BOOOOOM                                    PRIMALposition=03 PRIMALength=05  'OOOOM'
0905 // Needle: aaaaaBOOOOOM                               PRIMALposition=02 PRIMALength=09  'aaaaBOOOO'
0906 // Needle: BOOOOOMaaaaa                               PRIMALposition=03 PRIMALength=09  'OOOOMaaaa'
0907 PRIMALength=0;
0908 for (i=0+(1); i < cbPattern-((4)-1)+(1)-(1); i++) { // -(1) because the last BB order 4 has no counterpart(s)
0909	FoundAtPosition = cbPattern - ((4)-1) + 1;
0910	PRIMALpositionCANDIDATE=i;
0911	while ( PRIMALpositionCANDIDATE <= (FoundAtPosition-1) ) {
```

```
0912                  j = PRIMALpositionCANDIDATE + 1;
0913                  while ( j <= (FoundAtPosition-1) ) {
0914                            if ( *(uint32_t *)(pbPattern+PRIMALpositionCANDIDATE-(1)) == *(uint32_t *)(pbPattern+j-(1)) ) FoundAtPosition = j;
0915                            j++;
0916                  }
0917                  PRIMALpositionCANDIDATE++;
0918        }
0919        PRIMALlengthCANDIDATE = (FoundAtPosition-1)-i+1 +((4)-1);
0920        if (PRIMALlengthCANDIDATE >= PRIMALlength) {PRIMALposition=i; PRIMALlength = PRIMALlengthCANDIDATE;}
0921        if (cbPattern-i+1 <= PRIMALlength) break;
0922        if (PRIMALlength > 128) break; // Bail Out for 129[+]
0923 }
0924 // Swampwalker_BAILOUT heuristic order 4 (Needle should be bigger than 4) ]
0925
0926 // Here we have 4 or bigger NewNeedle, apply order 2 for pbPattern[i+(PRIMALposition-1)] with length 'PRIMALlength' and compare the pbPattern[i] with length
'cbPattern':
0927 PRIMALlengthCANDIDATE = cbPattern;
0928 cbPattern = PRIMALlength;
0929 pbPattern = pbPattern + (PRIMALposition-1);
0930
0931 // Revision 2 commented section [
0932 /*
0933 if (cbPattern-1 <= 255) {
0934 // BMH Order 2 [
0935                           ulHashPattern = *(uint32_t *)(pbPattern); // First four bytes
0936                           for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]= cbPattern-1;} // cbPattern-(Order-1) for Horspool; 'memset' if not optimized
0937                           for (i=0; i < cbPattern-1; i++) bm_Horspool_Order2[*(unsigned short *)(pbPattern+i)]=i; // Rightmost appearance/position is needed
0938                           i=0;
0939                           while (i <= cbTarget-cbPattern) {
0940                                     Gulliver = bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+cbPattern-1-1]];
0941                                     if ( Gulliver != cbPattern-1 ) { // CASE #2: if equal means the pair (char order 2) is not found i.e. Gulliver remains
intact, skip the whole pattern and fall back (Order-1) chars i.e. one char for Order 2
0942                                     if ( Gulliver == cbPattern-2 ) { // CASE #1: means the pair (char order 2) is found
0943                                               if ( *(uint32_t *)&pbTarget[i] == ulHashPattern) {
0944                                                         count = cbPattern-4+1;
0945                                                         while ( count > 0 && *(uint32_t *)(pbPattern+count-1) == *(uint32_t *)(&pbTarget[i]+(count-1)) )
0946                                                                   count = count-4;
0947 // If we miss to hit then no need to compare the original: Needle
0948 if ( count <= 0 ) {
0949 // I have to add out-of-range checks...
0950 // i-(PRIMALposition-1) >= 0
0951 // &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
0952 // i-(PRIMALposition-1)+(count-1) >= 0
0953 // &pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4
0954       if ( (i-(PRIMALposition-1) >= 0) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) && (&pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4) ) {
0955                   if ( *(uint32_t *)&pbTarget[i-(PRIMALposition-1)] == *(uint32_t *)(pbPattern-(PRIMALposition-1))) { // This fast check ensures not missing a match
(for remainder) when going under 0 in loop below:
0956                           count = PRIMALlengthCANDIDATE-4+1;
0957                           while ( count > 0 && *(uint32_t *)(pbPattern-(PRIMALposition-1)+count-1) == *(uint32_t *)(&pbTarget[i-(PRIMALposition-1)]+(count-1)) )
0958                                     count = count-4;
0959                           if ( count <= 0 ) return(pbTarget+i-(PRIMALposition-1));
0960                  }
0961       }
0962 }
0963                                               }
0964                                               Gulliver = 1;
0965                                     } else
0966                                               Gulliver = cbPattern - Gulliver - 2; // CASE #3: the pair is found and not as suffix i.e. rightmost position
0967                                     }
0968                                     i = i + Gulliver;
0969                                     //GlobalI++; // Comment it, it is only for stats.
0970                           }
0971                           return(NULL);
0972 // BMH Order 2 ]
0973 } else {
0974                           // BMH order 2, needle should be >=4:
0975                           ulHashPattern = *(uint32_t *)(pbPattern); // First four bytes
0976                           for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]=0;}
0977                           for (i=0; i < cbPattern-1; i++) bm_Horspool_Order2[*(unsigned short *)(pbPattern+i)]=1;
0978                           i=0;
0979                           while (i <= cbTarget-cbPattern) {
0980                                     Gulliver = 1; // 'Gulliver' is the skip
0981                                     if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+cbPattern-1-1]] != 0 ) {
0982                                               if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+cbPattern-1-1-2]] == 0 ) Gulliver = cbPattern-(2-1)-2; else
{
0983                                                         if ( *(uint32_t *)&pbTarget[i] == ulHashPattern) { // This fast check ensures not missing a match (for
remainder) when going under 0 in loop below:
0984                                                                   count = cbPattern-4+1;
0985                                                                   while ( count > 0 && *(uint32_t *)(pbPattern+count-1) == *(uint32_t *)(&pbTarget[i]+(count-
1)) )
0986                                                                             count = count-4;
0987 // If we miss to hit then no need to compare the original: Needle
0988 if ( count <= 0 ) {
0989 // I have to add out-of-range checks...
0990 // i-(PRIMALposition-1) >= 0
0991 // &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
0992 // i-(PRIMALposition-1)+(count-1) >= 0
0993 // &pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4
0994       if ( (i-(PRIMALposition-1) >= 0) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) && (&pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4) ) {
```

```
0995                if ( *(uint32_t *)&pbTarget[i-(PRIMALposition-1)] == *(uint32_t *)(pbPattern-(PRIMALposition-1))) { // This fast check ensures not missing a match
(for remainder) when going under 0 in loop below:
0996                        count = PRIMALlengthCANDIDATE-4+1;
0997                        while ( count > 0 && *(uint32_t *)(pbPattern-(PRIMALposition-1)+count-1) == *(uint32_t *)(&pbTarget[i-(PRIMALposition-1)]+(count-1)) )
0998                                count = count-4;
0999                        if ( count <= 0 ) return(pbTarget+i-(PRIMALposition-1));
1000                }
1001        }
1002 }
1003                                                        }
1004                                                }
1005                        } else Gulliver = cbPattern-(2-1);
1006                        i = i + Gulliver;
1007                        //GlobalI++; // Comment it, it is only for stats.
1008                }
1009                return(NULL);
1010 }
1011 */
1012 // Revision 2 commented section ]
1013
1014                if ( cbPattern<=NeedleThreshold2vs4swampLITE ) {
1015
1016                        // BMH order 2, needle should be >=4:
1017                        ulHashPattern = *(uint32_t *)(pbPattern); // First four bytes
1018                        for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]=0;}
1019                        for (i=0; i < cbPattern-1; i++) bm_Horspool_Order2[*(unsigned short *)(pbPattern+i)]=1;
1020                        i=0;
1021                        while (i <= cbTarget-cbPattern) {
1022                                Gulliver = 1; // 'Gulliver' is the skip
1023                                if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+cbPattern-1-1]] != 0 ) {
1024                                        if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+cbPattern-1-1-2]] == 0 ) Gulliver = cbPattern-(2-1)-2; else
{
1025                                                if ( *(uint32_t *)&pbTarget[i] == ulHashPattern) { // This fast check ensures not missing a match (for
remainder) when going under 0 in loop below:
1026                                                        count = cbPattern-4+1;
1027                                                        while ( count > 0 && *(uint32_t *)(pbPattern+count-1) == *(uint32_t *)(&pbTarget[i]+(count-
1)) )
1028                                                                count = count-4;
1029 // If we miss to hit then no need to compare the original: Needle
1030 if ( count <= 0 ) {
1031 // I have to add out-of-range checks...
1032 // i-(PRIMALposition-1) >= 0
1033 // &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
1034 // i-(PRIMALposition-1)+(count-1) >= 0
1035 // &pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4
1036        if ( (i-(PRIMALposition-1) >= 0) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) && (&pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4) ) {
1037                if ( *(uint32_t *)&pbTarget[i-(PRIMALposition-1)] == *(uint32_t *)(pbPattern-(PRIMALposition-1))) { // This fast check ensures not missing a match
(for remainder) when going under 0 in loop below:
1038                        count = PRIMALlengthCANDIDATE-4+1;
1039                        while ( count > 0 && *(uint32_t *)(pbPattern-(PRIMALposition-1)+count-1) == *(uint32_t *)(&pbTarget[i-(PRIMALposition-1)]+(count-1)) )
1040                                count = count-4;
1041                        if ( count <= 0 ) return(pbTarget+i-(PRIMALposition-1));
1042                }
1043        }
1044 }
1045                                                        }
1046                                                }
1047                        } else Gulliver = cbPattern-(2-1);
1048                        i = i + Gulliver;
1049                        //GlobalI++; // Comment it, it is only for stats.
1050                }
1051                return(NULL);
1052
1053                } else { // if ( cbPattern<=NeedleThreshold2vs4swampLITE )
1054
1055                        // BMH pseudo-order 4, needle should be >=8+2:
1056                        ulHashPattern = *(uint32_t *)(pbPattern); // First four bytes
1057                        for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]=0;}
1058                        // In line below we "hash" 4bytes to 2bytes i.e. 16bit table, how to compute TOTAL number of BBs, 'cbPattern - Order + 1' is the number
of BBs for text 'cbPattern' bytes long, for example, for cbPattern=11 'fastest fox' and Order=4 we have BBs = 11-4+1=8:
1059                        //"fast"
1060                        //"aste"
1061                        //"stes"
1062                        //"test"
1063                        //"est "
1064                        //"st f"
1065                        //"t fo"
1066                        //" fox"
1067                        //for (i=0; i < cbPattern-4+1; i++) bm_Horspool_Order2[( *(unsigned short *)(pbPattern+i+0) + *(unsigned short *)(pbPattern+i+2) ) & (
(1<<16)-1 )]=1;
1068                        //for (i=0; i < cbPattern-4+1; i++) bm_Horspool_Order2[( (*(uint32_t *)(pbPattern+i+0)>>16)+(*(uint32_t *)(pbPattern+i+0)&0xFFFF) ) & (
(1<<16)-1 )]=1;
1069                        // Above line is replaced by next one with better hashing:
1070                        for (i=0; i < cbPattern-4+1; i++) bm_Horspool_Order2[( (*(uint32_t *)(pbPattern+i+0)>>(16-1))+(*(uint32_t *)(pbPattern+i+0)&0xFFFF) ) &
( (1<<16)-1 )]=1;
1071                        i=0;
1072                        while (i <= cbTarget-cbPattern) {
1073                                Gulliver = 1;
1074                                //if ( bm_Horspool_Order2[( (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2]>>16)+(*(uint32_t *)&pbTarget[i+cbPattern-1-1-
```

```
2]&0xFFFF) ) & ( (1<<16)-1 )] != 0 ) { // DWORD #1
                                        // Above line is replaced by next one with better hashing:
                                        if ( bm_Horspool_Order2[( (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2]>>(16-1))+(*(uint32_t *)&pbTarget[i+cbPattern-1-1-
2]&0xFFFF) ) & ( (1<<16)-1 )] != 0 ) { // DWORD #1
                                                //if ( bm_Horspool_Order2[( (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]>>16)+(*(uint32_t *)&pbTarget[i+cbPattern-
1-1-2-4]&0xFFFF) ) & ( (1<<16)-1 )] == 0 ) Gulliver = cbPattern-(2-1)-2-4; else {
                                                // Above line is replaced in order to strengthen the skip by checking the middle DWORD,if the two DWORDs are 'ab'
and 'cd' i.e. [2x][2a][2b][2c][2d] then the middle DWORD is 'bc'.
                                                // The respective offsets (backwards) are: -10/-8/-6/-4 for 'xa'/'ab'/'bc'/'cd'.
                                                //if ( ( bm_Horspool_Order2[( (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-6]>>16)+(*(uint32_t
*)&pbTarget[i+cbPattern-1-1-2-6]&0xFFFF) ) & ( (1<<16)-1 )] ) + ( bm_Horspool_Order2[( (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]>>16)+(*(uint32_t
*)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF) ) & ( (1<<16)-1 )] ) + ( bm_Horspool_Order2[( (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]>>16)+(*(uint32_t
*)&pbTarget[i+cbPattern-1-1-2-2]&0xFFFF) ) & ( (1<<16)-1 )] ) < 3 ) Gulliver = cbPattern-(2-1)-2-4-2; else {
                                                // Above line is replaced by next one with better hashing:
                                                // When using (16-1) right shifting instead of 16 we will have two different pairs (if they are equal), the
highest bit being lost do the job especialy for ASCII texts with no symbols in range 128-255.
                                                // Example for genomesque pair TT+TT being shifted by (16-1):
                                                // T            = 01010100
                                                // TT           = 01010100 01010100
                                                // TTTT         = 01010100 01010100 01010100 01010100
                                                // TTTT>>16     = 00000000 00000000 01010100 01010100
                                                // TTTT>>(16-1) = 00000000 00000000 10101000 10101000 <--- Due to the left shift by 1, the 8th bits of 1st and 2nd
bytes are populated - usually they are 0 for English texts & 'ACGT' data.
                                                //if ( ( bm_Horspool_Order2[( (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-6]>>(16-1))+(*(uint32_t
*)&pbTarget[i+cbPattern-1-1-2-6]&0xFFFF) ) & ( (1<<16)-1 )] ) + ( bm_Horspool_Order2[( (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]>>(16-1))+(*(uint32_t
*)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF) ) & ( (1<<16)-1 )] ) + ( bm_Horspool_Order2[( (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]>>(16-1))+(*(uint32_t
*)&pbTarget[i+cbPattern-1-1-2-2]&0xFFFF) ) & ( (1<<16)-1 )] ) < 3 ) Gulliver = cbPattern-(2-1)-2-4-2; else {
                                                // 'Maximus' uses branched 'if', again.
                                                if ( \
                                                ( bm_Horspool_Order2[( (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-6 +1]>>(16-1))+(*(uint32_t
*)&pbTarget[i+cbPattern-1-1-2-6 +1]&0xFFFF) ) & ( (1<<16)-1 )] ) == 0 \
                                                || ( bm_Horspool_Order2[( (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4 +1]>>(16-1))+(*(uint32_t
*)&pbTarget[i+cbPattern-1-1-2-4 +1]&0xFFFF) ) & ( (1<<16)-1 )] ) == 0 \
                                                ) Gulliver = cbPattern-(2-1)-2-4-2 +1; else {
                                                // Above line is not optimized (several a SHR are used), we have 5 non-overlapping WORDs, or 3 overlapping WORDs,
within 4 overlapping DWORDs so:
// [2x][2a][2b][2c][2d]
// DWORD #4
// [2a] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-6]>>16) =     !SHR to be avoided! <--
// [2x] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-6]&0xFFFF) =                      |
//     DWORD #3                                                                    |
// [2b] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]>>16) =     !SHR to be avoided!   |<--
// [2a] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF) = ------------------------  |
//         DWORD #2                                                                    |
// [2c] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]>>16) =     !SHR to be avoided!     |<--
// [2b] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]&0xFFFF) = ---------------------------  |
//             DWORD #1                                                                    |
// [2d] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-0]>>16) =                                |
// [2c] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-0]&0xFFFF) = -------------------------------
//
// So in order to remove 3 SHR instructions the equal extractions are:
// DWORD #4
// [2a] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF) =  !SHR to be avoided! <--
// [2x] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-6]&0xFFFF) =                      |
//     DWORD #3                                                                    |
// [2b] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]&0xFFFF) =  !SHR to be avoided!   |<--
// [2a] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF) = ------------------------  |
//         DWORD #2                                                                    |
// [2c] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-0]&0xFFFF) =  !SHR to be avoided!     |<--
// [2b] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]&0xFFFF) = ---------------------------  |
//             DWORD #1                                                                    |
// [2d] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-0]>>16) =                                |
// [2c] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-0]&0xFFFF) = -------------------------------
                                                //if ( ( bm_Horspool_Order2[( (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF)+(*(uint32_t
*)&pbTarget[i+cbPattern-1-1-2-6]&0xFFFF) ) & ( (1<<16)-1 )] ) + ( bm_Horspool_Order2[( (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]&0xFFFF)+(*(uint32_t
*)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF) ) & ( (1<<16)-1 )] ) + ( bm_Horspool_Order2[( (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-0]&0xFFFF)+(*(uint32_t
*)&pbTarget[i+cbPattern-1-1-2-2]&0xFFFF) ) & ( (1<<16)-1 )] ) < 3 ) Gulliver = cbPattern-(2-1)-2-6; else {
// Since the above Decumanus mumbo-jumbo (3 overlapping lookups vs 2 non-overlapping lookups) is not fast enough we go DuoDecumanus or 3x4:
// [2y][2x][2a][2b][2c][2d]
// DWORD #3
//         DWORD #2
//                 DWORD #1
                                                //if ( ( bm_Horspool_Order2[( (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]>>16)+(*(uint32_t
*)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF) ) & ( (1<<16)-1 )] ) + ( bm_Horspool_Order2[( (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-8]>>16)+(*(uint32_t
*)&pbTarget[i+cbPattern-1-1-2-8]&0xFFFF) ) & ( (1<<16)-1 )] ) < 2 ) Gulliver = cbPattern-(2-1)-2-8; else {
                                                        if ( *(uint32_t *)&pbTarget[i] == ulHashPattern) {
                                                                // Order 4 [
                                                        // Let's try something "outrageous" like comparing with[out] overlap BBs 4bytes long instead of 1 byte
back-to-back:
                                                        // Inhere we are using order 4, 'cbPattern - Order + 1' is the number of BBs for text 'cbPattern' bytes
long, for example, for cbPattern=11 'fastest fox' and Order=4 we have BBs = 11-4+1=8:
                                                        //0:"fast" if the comparison failed here, 'count' is 1; 'Gulliver' is cbPattern-(4-1)-7
                                                        //1:"aste" if the comparison failed here, 'count' is 2; 'Gulliver' is cbPattern-(4-1)-6
                                                        //2:"stes" if the comparison failed here, 'count' is 3; 'Gulliver' is cbPattern-(4-1)-5
                                                        //3:"test" if the comparison failed here, 'count' is 4; 'Gulliver' is cbPattern-(4-1)-4
                                                        //4:"est " if the comparison failed here, 'count' is 5; 'Gulliver' is cbPattern-(4-1)-3
                                                        //5:"st f" if the comparison failed here, 'count' is 6; 'Gulliver' is cbPattern-(4-1)-2
                                                        //6:"t fo" if the comparison failed here, 'count' is 7; 'Gulliver' is cbPattern-(4-1)-1
                                                        //7:" fox" if the comparison failed here, 'count' is 8; 'Gulliver' is cbPattern-(4-1)
```

```
1142                                                                    count = cbPattern-4+1;
1143                                                                    // Below comparison is UNIdirectional:
1144                                                                    while ( count > 0 && *(uint32_t *)(pbPattern+count-1) == *(uint32_t *)(&pbTarget[i]+(count-
1)) )
1145                                                                            count = count-4;
1146 // count = cbPattern-4+1 = 23-4+1 = 20
1147 // boomshakalakaZZZZZZ[ZZZZ] 20
1148 // boomshakalakaZZ[ZZZZ]ZZZZ 20-4
1149 // boomshakala[kaZZ]ZZZZZZZZ 20-8 = 12
1150 // boomsha[kala]kaZZZZZZZZZZ 20-12 = 8
1151 // boo[msha]kalakaZZZZZZZZZZ 20-16 = 4
1152
1153 // If we miss to hit then no need to compare the original: Needle
1154 if ( count <= 0 ) {
1155 // I have to add out-of-range checks...
1156 // i-(PRIMALposition-1) >= 0
1157 // &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
1158 // i-(PRIMALposition-1)+(count-1) >= 0
1159 // &pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4
1160       if ( (i-(PRIMALposition-1) >= 0) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) && (&pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4) ) {
1161                   if ( *(uint32_t *)&pbTarget[i-(PRIMALposition-1)] == *(uint32_t *)(pbPattern-(PRIMALposition-1))) { // This fast check ensures not missing a match
(for remainder) when going under 0 in loop below:
1162                           count = PRIMALlengthCANDIDATE-4+1;
1163                           while ( count > 0 && *(uint32_t *)(pbPattern-(PRIMALposition-1)+count-1) == *(uint32_t *)(&pbTarget[i-(PRIMALposition-1)]+(count-1)) )
1164                                   count = count-4;
1165                           if ( count <= 0 ) return(pbTarget+i-(PRIMALposition-1));
1166                   }
1167       }
1168 }
1169
1170                                                                    // In order to avoid only-left or only-right WCS the memcmp should be done as left-to-right
and right-to-left AT THE SAME TIME.
1171                                                                    // Below comparison is BIdirectional. It pays off when needle is 8+++ long:
1172 //                                                                 for (count = cbPattern-4+1; count > 0; count = count-4) {
1173 //                                                                         if ( *(uint32_t *)(pbPattern+count-1) != *(uint32_t *)(&pbTarget[i]+(count-1)) )
{break;};
1174 //                                                                          if ( *(uint32_t *)(pbPattern+(cbPattern-4+1)-count) != *(uint32_t
*)(&pbTarget[i]+(cbPattern-4+1)-count) ) {count = (cbPattern-4+1)-count +(1); break;} // +(1) because two lookups are implemented as one, also no danger of 'count' being
0 because of the fast check outwith the 'while': if ( *(uint32_t *)&pbTarget[i] == ulHashPattern)
1175 //                                                                 }
1176 //                                                                 if ( count <= 0 ) return(pbTarget+i);
1177                                                                         // Checking the order 2 pairs in mismatched DWORD, all the 3:
1178                                                                         //if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+count-1]] == 0 )
Gulliver = count; // 1 or bigger, as it should
1179                                                                         //if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+count-1+1]] == 0 )
Gulliver = count+1; // 1 or bigger, as it should
1180                                                                         //if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+count-1+1+1]] == 0 )
Gulliver = count+1+1; // 1 or bigger, as it should
1181                                                                    //      if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+count-1]] +
bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+count-1+1]] + bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+count-1+1+1]] < 3 ) Gulliver = count; // 1 or bigger,
as it should, THE MIN(count,count+1,count+1+1)
1182                                                                        // Above compound 'if' guarantees not that Gulliver > 1, an example:
1183                                                                        // Needle:    fastest tax
1184                                                                        // Window: ...fastast tax...
1185                                                                        // After matching ' tax' vs ' tax' and 'fast' vs 'fast' the mismathced DWORD is
'test' vs 'tast':
1186                                                                        // 'tast' when factorized down to order 2 yields: 'ta','as','st' - all the three
when summed give 1+1+1=3 i.e. Gulliver remains 1.
1187                                                                        // Roughly speaking, this attempt maybe has its place in worst-case scenarios but
not in English text and even not in ACGT data, that's why I commented it in original 'Shockeroo'.
1188                                                                        //if ( bm_Horspool_Order2[( (*(uint32_t *)&pbTarget[i+count-1]>>16)+(*(uint32_t
*)&pbTarget[i+count-1]&0xFFFF) ) & ( (1<<16)-1 )] == 0 ) Gulliver = count; // 1 or bigger, as it should
1189                                                                        // Above line is replaced by next one with better hashing:
1190 //                                                                     if ( bm_Horspool_Order2[( (*(uint32_t *)&pbTarget[i+count-1]>>(16-1))+(*(uint32_t
*)&pbTarget[i+count-1]&0xFFFF) ) & ( (1<<16)-1 )] == 0 ) Gulliver = count; // 1 or bigger, as it should
1191                                                                    // Order 4 ]
1192                                                                }
1193                                                        }
1194                                        } else Gulliver = cbPattern-(2-1)-2; // -2 because we check the 4 rightmost bytes not 2.
1195                                        i = i + Gulliver;
1196                                        //GlobalI++; // Comment it, it is only for stats.
1197                                }
1198                                return(NULL);
1199
1200                   } // if ( cbPattern<=NeedleThreshold2vs4swampLITE )
1201                   } // if ( cbPattern<=NeedleThreshold2vs4swampLITE )
1202       } //if ( cbPattern<4 )
1203 }
1204
1205 // Fixed version from 2012-Feb-27.
1206 // Caution: For better speed the case 'if (cbPattern==1)' was removed, so Pattern must be longer than 1 char.
1207 char * Railgun_Doublet (char * pbTarget, char * pbPattern, uint32_t cbTarget, uint32_t cbPattern)
1208 {
1209       char * pbTargetMax = pbTarget + cbTarget;
1210       register uint32_t ulHashPattern;
1211       uint32_t ulHashTarget, count, countSTATIC;
1212
1213       if (cbPattern > cbTarget) return(NULL);
1214
```

```
1215        countSTATIC = cbPattern-2;
1216
1217        pbTarget = pbTarget+cbPattern;
1218        ulHashPattern = (*(uint16_t *)(pbPattern));
1219
1220        for ( ;; ) {
1221                    if ( ulHashPattern == (*(uint16_t *)(pbTarget-cbPattern)) ) {
1222                                count = countSTATIC;
1223                                while ( count && *(char *)(pbPattern+2+(countSTATIC-count)) == *(char *)(pbTarget-cbPattern+2+(countSTATIC-count)) ) {
1224                                            count--;
1225                                }
1226                                if ( count == 0 ) return((pbTarget-cbPattern));
1227                    }
1228                    pbTarget++;
1229                    if (pbTarget > pbTargetMax) return(NULL);
1230        }
1231 }
1232
1233 // Last change: 2014-Apr-18
1234 // If you want to help me to improve it, email me at: sanmayce@sanmayce.com
1235 // Enfun!
```

The extraordinary eight-fold path.
This unique way avoids the two extremes: self-mortification that weakens one's body and self-indulgence that retards one's mind.
It consists of the following eight factors:
1) Harmonious perspective (Sammà Diññhi)
2) Harmonious feeling (Sammà Saükappa)
3) Harmonious speech (Sammà Vàcà)
4) Harmonious action (Sammà Kammanta)
5) Harmonious living (Sammà Ajãva)
6) Harmonious practice (Sammà Vàyàma)
7) Harmonious introspection (Sammà Sati)
8) Harmonious equilibrium (Sammà Samàdhi)

/'Treasury of Truth', Illustrated Dhammapada, Author: Ven. Weragoda Sarada Maha Thero/

1. The best of paths is the Eightfold Path. The best of truths are the four Sayings. Non-attachment is the best of states. The best of bipeds is the Seeing One.
2. This is the only Way. There is none other for the purity of vision. Do you follow this path. This is the bewilderment of Māra.
3. Entering upon that path, you will make an end of pain. Having learnt the removal of thorns, have I taught you the path.
4. Striving should be done by yourselves; the Tathāgatas are only teachers. The meditative ones, who enter the way, are delivered from the bonds of Māra.
5. "All conditions are impermanent:" when one sees this with wisdom, one is disenchanted with suffering; this is the path to purity.
6. "All conditions are unsatisfactory:" when one sees this with wisdom, one is disenchanted with suffering; this is the path to purity.
7. "All phenomena are not-self:" when one sees this with wisdom, one is disenchanted with suffering; this is the path to purity.
8. The inactive idler who strives not when he should strive, who, though young and strong, is slothful, with (good) thoughts depressed, does not by wisdom realise the Path.
9. Watchful of speech, well restrained in mind, let him do nought unskillful through his body. Let him purify these three ways of action and win the path realised by the sages.
10. From meditation arises wisdom. Without meditation wisdom wanes. Knowing this twofold path of gain and loss, let one so conduct oneself so that wisdom increases.
11. Cut down the entire forest, not just a single tree. From the forest springs fear. Cutting down both forest and brushwood, be passionless, O monks.
12. For as long as the slightest passion of man towards women is not cut down, so long is his mind in bondage, like the calf to its mother.
13. Cut off your affection, as though it were an autumn lily, with the hand. Cultivate this path of peace. Nibbāna has been expounded by the Auspicious One.
14. Here will I live in the rainy season, here in the autumn and in the summer: thus muses the fool. He realises not the danger (of death).
15. The doting man with mind set on children and herds, death seizes and carries away, as a great flood (sweeps away) a slumbering village.
16. There are no sons for one's protection, neither father nor even kinsmen; for one who is overcome by death no protection is to be found among kinsmen.
17. Realising this fact, let the virtuous and wise person swiftly clear the way that leads to nibbāna.

/'The Dhammapada', 20 – Magga Vagga, edited by Bhikkhu Pesala/