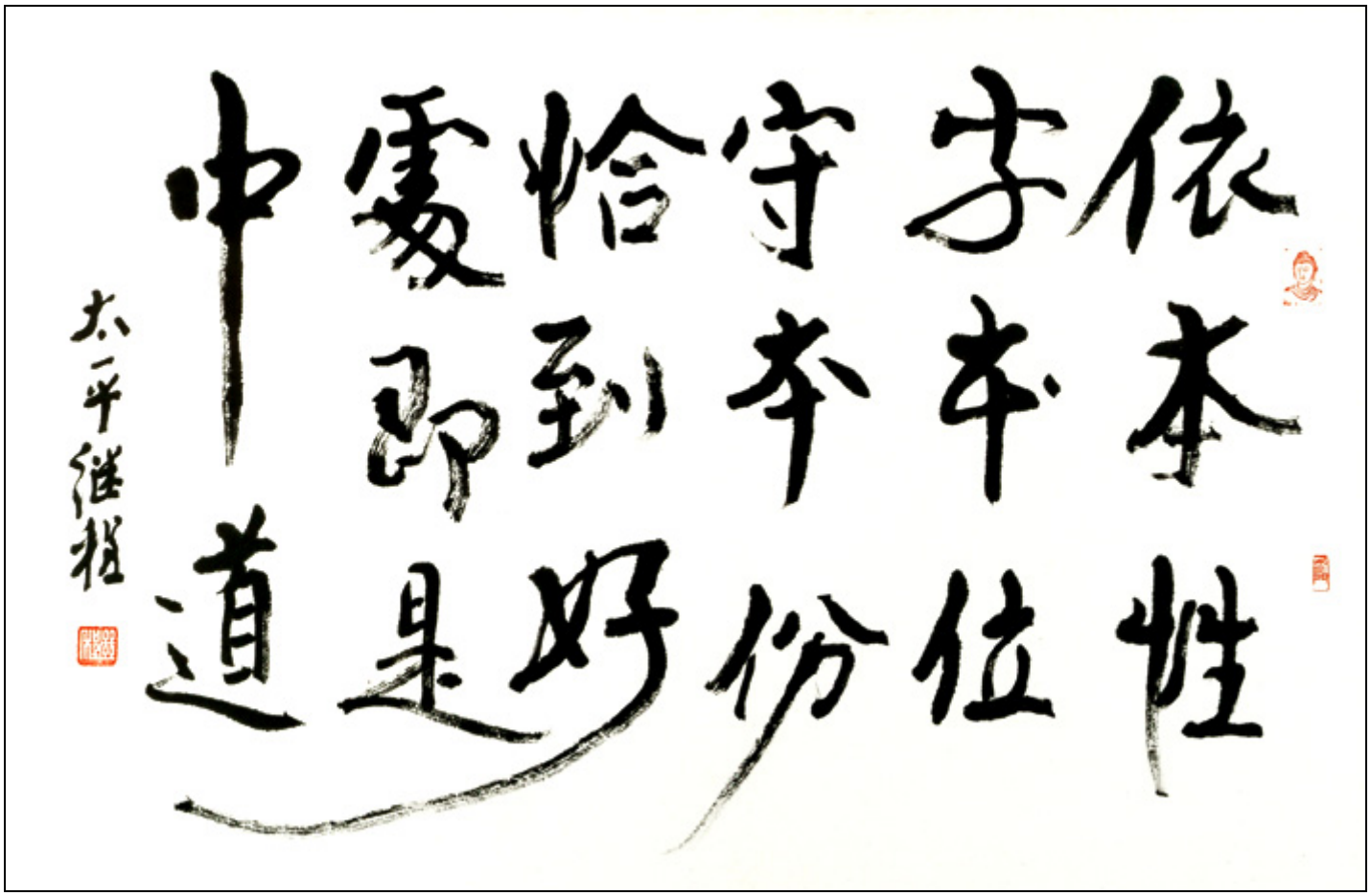
中道

NAKAMICHI




```
0001 // Nakamichi is 100% FREE LZSS FAST decompressor.
0002
0003 // Nakamichi, revision 1-RSSBO++, written by Kaze.
0004 // Change #1: Decompression fetches WORD instead of BYTE+BYTE.
0005 // Nakamichi, revision 1-RSSBO+, written by Kaze.
0006 // 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.
0007 // Change #2: Fixed bug in progress indicator.
0008 // Change #3: Not done yet, the two nasty byte loads to be removed in the decompression part.
0009 // Nakamichi, revision 1-RSSBO, written by Kaze.
0010 // Based on Nobuo Ito's source, thanks Ito.
0011 // The main goal of Nakamichi is to allow supersimple and superfast decoding for English x-grams (mainly) in pure C, or not, heh-heh.
0012 // Natively Nakamichi is targeted as 64bit tool with 16 threads, helping Kazahana to traverse faster when I/O is not superior.
0013 // In short, Nakamichi is intended as x-gram decompressor.
0014
0015 // Eightfold Path ~ the Buddhist path to nirvana, comprising eight aspects in which an aspirant must become practised;
0016 // eightfold way ~ (Physics), the grouping of hadrons into supermultiplets by means of SU(3)); (b) adverb to eight times the number or quantity: OE.
0017
0018 // Revision 1-RSSBO++ notes [
0019 // Note1: On 'ENWIKI', being a XML English text, Nakamichi's compression ratio is among the worsts:
0020 //        021,440,055 enwiki-20140304-pages-articles.7z.001.tangelo
0021 //        041,984,881 enwiki-20140304-pages-articles.7z.001.lzt        -19
0022 //        047,685,453 enwiki-20140304-pages-articles.7z.001.lzt        -11
0023 //        056,365,696 enwiki-20140304-pages-articles.7z.001.YAPPY      32768   uncomp 534.3 MB/s
0024 //        057,497,885 enwiki-20140304-pages-articles.7z.001.YAPPY      16384   uncomp 549.5 MB/s !SUPERIOR DECOMPRESSOR!
0025 //        059,756,354 enwiki-20140304-pages-articles.7z.001.YAPPY      8192    uncomp 549.5 MB/s
0026 //        060,865,688 enwiki-20140304-pages-articles.7z.001.Nakamichi  Decompression at 355 MB/s
0027 //        064,270,963 enwiki-20140304-pages-articles.7z.001.YAPPY      4096    uncomp 574.2 MB/s
0028 //        070,845,249 enwiki-20140304-pages-articles.7z.001.YAPPY      2048    uncomp 631.8 MB/s
0029 //        078,039,768 enwiki-20140304-pages-articles.7z.001.YAPPY      1024    uncomp 680.3 MB/s
0030 //        104,857,600 enwiki-20140304-pages-articles.7z.001
0031 // Note2: On 'PAGODA', being a highly redundant English text, Nakamichi's compression ratio is among the worsts BUT good enough for my goals:
0032 //        035,834,062 Kazahana_on.PAGODA-order-5.txt.tangelo
```

```
0033 //        107,077,360 Kazahana_on.PAGODA-order-5.txt.lzt              -19
0034 //        119,750,869 Kazahana_on.PAGODA-order-5.txt.lzt              -11
0035 //        178,902,966 Kazahana_on.PAGODA-order-5.txt.YAPPY            32768   uncomp 932.0 MB/s
0036 //        180,650,931 Kazahana_on.PAGODA-order-5.txt.YAPPY            16384   uncomp 940.7 MB/s !SUPERIOR DECOMPRESSOR!
0037 //        184,153,244 Kazahana_on.PAGODA-order-5.txt.YAPPY            8192    uncomp 923.5 MB/s
0038 //        191,149,889 Kazahana_on.PAGODA-order-5.txt.YAPPY            4096    uncomp 949.0 MB/s
0039 //        195,522,294 Kazahana_on.PAGODA-order-5.txt.Nakamichi        Decompression at 646 MB/s
0040 //        202,655,189 Kazahana_on.PAGODA-order-5.txt.YAPPY            2048    uncomp 967.2 MB/s
0041 //        219,625,842 Kazahana_on.PAGODA-order-5.txt.YAPPY            1024    uncomp 958.0 MB/s
0042 //        846,351,894 Kazahana_on.PAGODA-order-5.txt
0043 // Note3: On 'OSHO', being a typical English text, Nakamichi's compression ratio is among the worsts:
0044 //        034,419,437 OSHO.TXT.tangelo
0045 //        070,067,665 OSHO.TXT.lzt                                    -19
0046 //        085,054,228 OSHO.TXT.lzt                                    -11
0047 //        097,806,047 OSHO.TXT.YAPPY                                  32768   uncomp 512.5 MB/s
0048 //        099,927,141 OSHO.TXT.YAPPY                                  16384   uncomp 519.7 MB/s !SUPERIOR DECOMPRESSOR!
0049 //        104,158,750 OSHO.TXT.YAPPY                                  8192    uncomp 519.7 MB/s
0050 //        106,283,618 OSHO.TXT.Nakamichi                              Decompression at 406 MB/s
0051 //        112,642,538 OSHO.TXT.YAPPY                                  4096    uncomp 542.1 MB/s
0052 //        124,617,790 OSHO.TXT.YAPPY                                  2048    uncomp 575.3 MB/s
0053 //        137,306,077 OSHO.TXT.YAPPY                                  1024    uncomp 617.1 MB/s
0054 //        206,908,949 OSHO.TXT
0055 // Revision 1-RSSBO++ notes ]
0056
0057 // Revision 1-RSSBO notes [
0058 // 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.
0059 // Note2: English x-grams are as English texts but more redundant, in other words they are phraselists in most cases, sometimes wordlists.
0060 // Note3: On OSHO.TXT, being a typical English text, Nakamichi's compression ratio is among the worsts:
0061 //        206,908,949 OSHO.TXT
0062 //        125,022,859 OSHO.TXT.Nakamichi
0063 //        It struggles with English texts but decomprression speed is quite sweet (Core 2 T7500 2200MHz, 32bit code):
0064 //        Nakamichi, revision 1-, written by Kaze.
0065 //        Decompressing 125022859 bytes ...
0066 //        RAM-to-RAM performance: 477681 KB/s.
0067 // 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.
0068 // Revision 1-RSSBO notes ]
0069
0070 // Quick notes on PAGODAs (the padded x-gram lists):
0071 // Every single word in English has its own PAGODA, in example below 'on' PAGODA is given (Kazahana_on.PAGODA-order-5.txt):
0072 // 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':
0073 /*
0074 D:\_KAZE\Nakamichi_r1-RSSBO>dir \_GW\ka*
0075
0076 04/12/2014  05:07 AM                14 Kazahana_on.1-1.txt
0077 04/12/2014  05:07 AM         1,635,389 Kazahana_on.2-1.txt
0078 04/12/2014  05:07 AM         1,906,734 Kazahana_on.2-2.txt
0079 04/12/2014  05:07 AM        10,891,415 Kazahana_on.3-1.txt
0080 04/12/2014  05:07 AM        15,797,703 Kazahana_on.3-2.txt
0081 04/12/2014  05:07 AM        20,419,280 Kazahana_on.3-3.txt
0082 04/12/2014  05:07 AM        22,141,823 Kazahana_on.4-1.txt
0083 04/12/2014  05:07 AM        36,002,113 Kazahana_on.4-2.txt
0084 04/12/2014  05:07 AM        33,236,772 Kazahana_on.4-3.txt
0085 04/12/2014  05:07 AM        33,902,425 Kazahana_on.4-4.txt
0086 04/12/2014  05:07 AM        24,795,989 Kazahana_on.5-1.txt
0087 04/12/2014  05:07 AM        30,766,220 Kazahana_on.5-2.txt
0088 04/12/2014  05:07 AM        38,982,816 Kazahana_on.5-3.txt
0089 04/12/2014  05:07 AM        38,089,575 Kazahana_on.5-4.txt
0090 04/12/2014  05:07 AM        34,309,057 Kazahana_on.5-5.txt
0091 04/12/2014  05:07 AM       846,351,894 Kazahana_on.PAGODA-order-5.txt
0092
0093 D:\_KAZE\Nakamichi_r1-RSSBO>type \_GW\Kazahana_on.1-1.txt
0094 9,999,999       on
0095
0096 D:\_KAZE\Nakamichi_r1-RSSBO>type \_GW\Kazahana_on.2-1.txt
0097 9,999,999       on_the
0098 1,148,054       on_his
0099 0,559,694       on_her
0100 0,487,856       on_this
0101 0,399,485       on_your
0102 0,381,570       on_my
0103 0,367,282       on_their
0104 ...
0105
0106 D:\_KAZE\Nakamichi_r1-RSSBO>type \_GW\Kazahana_on.2-2.txt
0107 0,545,191       based_on
0108 0,397,408       and_on
0109 0,334,266       go_on
0110 0,329,561       went_on
0111 0,263,035       was_on
0112 0,246,332       it_on
0113 0,229,041       down_on
0114 0,202,151       going_on
0115 ...
0116
0117 D:\_KAZE\Nakamichi_r1-RSSBO>type \_GW\Kazahana_on.5-5.txt
0118 0,083,564       foundation_osho_s_books_on
0119 0,012,404       medium_it_may_be_on
0120 0,012,354       if_you_received_it_on
0121 0,012,152       medium_they_may_be_on
```

```
0122 0,012,144       agree_to_also_provide_on
0123 0,012,139       a_united_states_copyright_on
0124 0,008,067       we_are_constantly_working_on
0125 0,008,067       questions_we_have_received_on
0126 0,006,847       file_was_first_posted_on
0127 0,006,441       of_we_are_already_on
0128 0,006,279       you_received_this_ebook_on
0129 0,005,865       you_received_this_etext_on
0130 0,005,833       to_keep_an_eye_on
0131 ...
0132
0133 D:\_KAZE\Nakamichi_r1-RSSBO>type \_GW\Kazahana_on.PAGODA-order-5.txt
0134 9,999,999                                                          on
0135 9,999,999                                                          on_the
0136 1,148,054                                                          on_his
0137 0,559,694                                                          on_her
0138 0,487,856                                                          on_this
0139 0,399,485                                                          on_your
0140 0,381,570                                                          on_my
0141 0,367,282                                                          on_their
0142 0,356,572                                                          on_to
0143 0,299,453                                                          on_which
0144 0,291,876                                                          on_that
0145 0,276,388                                                          on_it
0146 0,250,541                                                          on_one
0147 0,221,101                                                          on_him
0148 0,206,864                                                          on_an
0149 0,197,541                                                          on_its
0150 0,166,762                                                          on_earth
0151 0,165,157                                                          on_page
0152 0,156,605                                                          on_our
0153 0,155,320                                                          on_all
0154 0,149,930                                                          on_account
0155 ...
0156 0,002,479                                                  updated_on
0157 0,002,478                                                   signal_on
0158 0,002,477                                                    threw_on
0159 0,002,475                                                     toll_on
0160 0,002,475                                                predicated_on
0161 0,002,472                                                    stains_on
0162 0,002,471                                                      seal_on
0163 ...
0164 0,000,004                                                     aarre_on
0165 0,000,004                                                      aapp_on
0166 0,000,004                                                  aanandham_on
0167 0,437,178                                                          on_the_other
0168 0,153,103                                                          on_the_ground
0169 0,146,123                                                          on_account_of
0170 0,140,358                                                          on_the_floor
0171 0,115,525                                                          on_to_the
0172 0,113,868                                                          on_the_table
0173 0,113,560                                                          on_the_contrary
0174 0,111,944                                                          on_the_same
0175 ...
0176 0,000,004                                                          on_a_actual
0177 0,000,004                                                          on_a_abattu
0178 0,181,050                                                     based_on_the
0179 0,151,126                                                       and_on_the
0180 0,109,493                                                       was_on_the
0181 0,108,238                                                      down_on_the
0182 0,089,090                                                  depending_on_the
0183 0,083,596                                                     books_on_cd
0184 0,077,772                                                        it_on_the
0185 0,075,412                                                       out_on_the
0186 ...
0187 0,000,004                                                           a_on_campo
0188 0,000,004                                                           a_on_average
0189 0,159,602                                                    and_so_on
0190 0,083,791                                                    s_books_on
0191 0,081,020                                                   is_based_on
0192 0,065,126                                                    he_went_on
0193 0,047,856                                                   was_going_on
0194 ...
0195 0,000,009                                          optical_transceivers_on
0196 0,000,009                                             oppressive_weight_on
0197 0,000,009                                                  opposite_way_on
0198 0,000,009                                               opposite_views_on
0199 ...
0200 0,000,012                                                          on_the_doorstep_were
0201 0,000,012                                                          on_the_doorstep_said
0202 0,000,012                                                          on_the_doorstep_looking
0203 ...
0204 0,000,027                                                    depends_on_several_variables
0205 0,000,027                                                    depends_on_several_things
0206 0,000,027                                                    depends_on_people_s
0207 ...
0208 0,000,004                                           genetic_influences_on_personality
0209 0,000,004                                           genetic_influences_on_osteoarthritis
0210 0,000,004                                           genetic_influences_on_behavior
```

```
0211 ...
0212 0,000,004                                                    on_the_train_to_hell
0213 0,000,004                                                    on_the_train_to_delhi
0214 0,000,004                                                    on_the_train_to_cambridge
0215 ...
0216 0,000,006                                                 now_on_the_puppeteers_will
0217 0,000,006                                                 now_on_the_open_ground
0218 0,000,006                                                 now_on_the_naked_woods
0219 ...
0220 0,000,004                                       leaned_lightly_on_his_shoulder
0221 0,000,004                                       leaned_lightly_on_her_shoulders
0222 0,000,004                                        leaned_lazily_on_the_neck
0223 ...
0224 0,000,039                               american_professional_society_on_the
0225 0,000,039                                           altars_that_were_on_the
0226 0,000,039                                           also_some_verses_on_the
0227 ...
0228 0,000,008                               portrait_of_washington_carved_on
0229 0,000,008                                     portrait_of_a_gentleman_on
0230 0,000,008                                portlets_without_single_sign_on
0231 0,000,008                                    portion_of_this_treatise_on
0232 ...
0233
0234 D:\_KAZE\Nakamichi_r1-RSSBO>dir
0235
0236 04/12/2014  05:07 AM       846,351,894 Kazahana_on.PAGODA-order-5.txt
0237
0238 D:\_KAZE\Nakamichi_r1-RSSBO>Nakamichi.exe Kazahana_on.PAGODA-order-5.txt
0239 Nakamichi, revision 1-RSSBO, written by Kaze.
0240 Compressing 846351894 bytes ...
0241 /; Each rotation means 128KB are encoded; Done 100%
0242 RAM-to-RAM performance: 512 KB/s.
0243
0244 D:\_KAZE\Nakamichi_r1-RSSBO>dir
0245
0246 04/12/2014  05:07 AM       846,351,894 Kazahana_on.PAGODA-order-5.txt
0247 04/15/2014  06:30 PM       293,049,398 Kazahana_on.PAGODA-order-5.txt.Nakamichi
0248
0249 D:\_KAZE\Nakamichi_r1-RSSBO>Nakamichi.exe Kazahana_on.PAGODA-order-5.txt.Nakamichi
0250 Nakamichi, revision 1-RSSBO, written by Kaze.
0251 Decompressing 293049398 bytes ...
0252 RAM-to-RAM performance: 607 MB/s.
0253
0254 D:\_KAZE\Nakamichi_r1-RSSBO>Yappy.exe Kazahana_on.PAGODA-order-5.txt 4096
0255 YAPPY: [b 4K] bytes 846351894 -> 191149889  22.6%  comp  33.8 MB/s  uncomp 875.4 MB/s
0256
0257 D:\_KAZE\Nakamichi_r1-RSSBO>Yappy.exe Kazahana_on.PAGODA-order-5.txt 8192
0258 YAPPY: [b 8K] bytes 846351894 -> 184153244  21.8%  comp  35.0 MB/s  uncomp 898.3 MB/s
0259
0260 D:\_KAZE\Nakamichi_r1-RSSBO>Yappy.exe Kazahana_on.PAGODA-order-5.txt 16384
0261 YAPPY: [b 16K] bytes 846351894 -> 180650931  21.3%  comp  28.8 MB/s  uncomp 906.4 MB/s
0262
0263 D:\_KAZE\Nakamichi_r1-RSSBO>Yappy.exe Kazahana_on.PAGODA-order-5.txt 32768
0264 YAPPY: [b 32K] bytes 846351894 -> 178902966  21.1%  comp  35.0 MB/s  uncomp 906.4 MB/s
0265
0266 D:\_KAZE\Nakamichi_r1-RSSBO>Yappy.exe Kazahana_on.PAGODA-order-5.txt 65536
0267 YAPPY: [b 64K] bytes 846351894 -> 178027899  21.0%  comp  34.5 MB/s  uncomp 914.6 MB/s
0268
0269 D:\_KAZE\Nakamichi_r1-RSSBO>Yappy.exe Kazahana_on.PAGODA-order-5.txt 131072
0270 YAPPY: [b 128K] bytes 846351894 -> 177591807  21.0%  comp  34.9 MB/s  uncomp 906.4 MB/s
0271
0272 D:\_KAZE\Nakamichi_r1-RSSBO>
0273 */
0274
0275 #include <stdio.h>
0276 #include <stdlib.h>
0277 #include <stdint.h> // uint64_t needed
0278 #include <time.h>
0279
0280 #include <emmintrin.h> // SSE2 intrinsics
0281 //#include <smmintrin.h> // SSE4.1 intrinsics
0282 //#include <immintrin.h> // AVX intrinsics
0283
0284 void SlowCopy128bit(const char *SOURCE, char *TARGET) { _mm_storeu_si128((__m128i *)(TARGET), _mm_loadu_si128((const __m128i *)(SOURCE))); }
0285
0286 #ifndef NULL
0287 #define NULL ((void*)0)
0288 #endif
0289
0290 void SearchIntoSlidingWindow(unsigned int* retIndex, unsigned int* retMatch, char* refStart,char* refEnd,char* encStart,char* encEnd);
0291 unsigned int SlidingWindowVsLookAheadBuffer(char* refStart, char* refEnd, char* encStart, char* encEnd);
0292 unsigned int Compress(char* ret, char* src, unsigned int srcSize);
0293 unsigned int Decompress(char* ret, char* src, unsigned int srcSize);
0294 char * Railgun_Swampshine_BailOut(char * pbTarget, char * pbPattern, uint32_t cbTarget, uint32_t cbPattern);
0295
0296 // Min_Match_Length=THRESHOLD=4 means 4 and bigger are to be encoded:
0297 #define Min_Match_BAILOUT_Length (4)
0298 #define Min_Match_Length (4)
0299 #define OffsetBITS (14)
```

```
#define LengthBITS (1)

//12bit
//#define REF_SIZE (4095+Min_Match_Length)
#define REF_SIZE ( ((1<<OffsetBITS)-1) + Min_Match_Length )
//3bit
//#define ENC_SIZE (7+Min_Match_Length)
//#define ENC_SIZE ( ((1<<LengthBITS)-1) + Min_Match_Length )
// Caramba, still don't feel the full picture about LookAheadBuffer!
#define ENC_SIZE ( Min_Match_Length+Min_Match_Length*4 )

int main( int argc, char *argv[] ) {
	FILE *fp;
	int SourceSize;
	int TargetSize;
	char* SourceBlock=NULL;
	char* TargetBlock=NULL;
	char* Nakamichi = ".Nakamichi\0";
	char NewFileName[256];
	clock_t clocks1, clocks2;

	printf("Nakamichi, revision 1-RSSBO++, written by Kaze, based on Nobuo Ito's LZSS source.\n");
	if (argc==1) {
		printf("Usage: Nakamichi filename\n"); exit(13);
	}
	if ((fp = fopen(argv[1], "rb")) == NULL) {
		printf("Nakamichi: Can't open '%s' file.\n", argv[1]); exit(13);
	}
	fseek(fp, 0, SEEK_END);
	SourceSize = ftell(fp);
	fseek(fp, 0, SEEK_SET);
	// If filename ends in '.Nakamichi' then mode is decompression otherwise compression.
	if (strcmp(argv[1]+(strlen(argv[1])-strlen(Nakamichi)), Nakamichi) == 0) {
	SourceBlock = (char*)malloc(SourceSize+512);
	TargetBlock = (char*)malloc(5*SourceSize+512);
	fread(SourceBlock, 1, SourceSize, fp);
	fclose(fp);
		printf("Decompressing %d bytes ...\n", SourceSize );
		clocks1 = clock();
		TargetSize = Decompress(TargetBlock, SourceBlock, SourceSize);
		clocks2 = clock();
		printf("RAM-to-RAM performance: %d MB/s.\n", ((TargetSize/(clocks2 - clocks1 + 1))*(long)1000)>>20);
		strcpy(NewFileName, argv[1]);
		*( NewFileName + strlen(argv[1])-strlen(Nakamichi) ) = '\0';
	} else {
	SourceBlock = (char*)malloc(SourceSize+512);
	TargetBlock = (char*)malloc(SourceSize+512);
	fread(SourceBlock, 1, SourceSize, fp);
	fclose(fp);
		printf("Compressing %d bytes ...\n", SourceSize );
		clocks1 = clock();
		TargetSize = Compress(TargetBlock, SourceBlock, SourceSize);
		clocks2 = clock();
		printf("RAM-to-RAM performance: %d KB/s.\n", ((SourceSize/(clocks2 - clocks1 + 1))*(long)1000)>>10);
		strcpy(NewFileName, argv[1]);
		strcat(NewFileName, Nakamichi);
	}
	if ((fp = fopen(NewFileName, "wb")) == NULL) {
		printf("Nakamichi: Can't write '%s' file.\n", NewFileName); exit(13);
	}
	fwrite(TargetBlock, 1, TargetSize, fp);
	fclose(fp);
	free(TargetBlock);
	free(SourceBlock);
	exit(0);
}

void SearchIntoSlidingWindow(unsigned int* retIndex, unsigned int* retMatch, char* refStart,char* refEnd,char* encStart,char* encEnd){
	char* FoundAtPosition;
	unsigned int match=0;
	*retIndex=0;
	*retMatch=0;
// Step #1: First try to find 16bytes long match...
//#ifdef ReplaceBruteForceWithRailgunSwampshineBailOut
	if (refStart < refEnd) {
		FoundAtPosition = Railgun_Swampshine_BailOut(refStart, encStart, (uint32_t)(refEnd-refStart), Min_Match_BAILOUT_Length*4);
		if (FoundAtPosition!=NULL) {
			*retMatch=Min_Match_BAILOUT_Length*4;
			*retIndex=refEnd-FoundAtPosition;
		}
	}
//#else
//	while(refStart < refEnd){
//		match=SlidingWindowVsLookAheadBuffer(refStart,refEnd,encStart,encEnd);
//		if(match > *retMatch){
//			*retMatch=match;
//			*retIndex=refEnd-refStart;
//		}
//		if(*retMatch >= Min_Match_BAILOUT_Length) break;
```

```
0389 //              refStart++;
0390 //    }
0391 //#endif
0392 // Step #2: If no such match then try to find 4bytes long match...
0393 if (*retMatch!=Min_Match_BAILOUT_Length*4) {
0394       if (refStart < refEnd) {
0395                 FoundAtPosition = Railgun_Swampshine_BailOut(refStart, encStart, (uint32_t)(refEnd-refStart), Min_Match_BAILOUT_Length);
0396                 if (FoundAtPosition!=NULL) {
0397                           *retMatch=Min_Match_BAILOUT_Length;
0398                           *retIndex=refEnd-FoundAtPosition;
0399                 }
0400       }
0401 }
0402 }
0403
0404 unsigned int SlidingWindowVsLookAheadBuffer( char* refStart, char* refEnd, char* encStart,char* encEnd){
0405       int ret = 0;
0406       while(refStart[ret] == encStart[ret]){
0407                 if(&refStart[ret] >= refEnd) break;
0408                 if(&encStart[ret] >= encEnd) break;
0409                 ret++;
0410                 if(ret >= Min_Match_BAILOUT_Length) break;
0411       }
0412       return ret;
0413 }
0414
0415 unsigned int Compress(char* ret, char* src, unsigned int srcSize){
0416       unsigned int srcIndex=0;
0417       unsigned int retIndex=0;
0418       unsigned int index=0;
0419       unsigned int match=0;
0420       unsigned int notMatch=0;
0421       unsigned char* notMatchStart=NULL;
0422       char* refStart=NULL;
0423       char* encEnd=NULL;
0424       int Melnitchka=0;
0425       char *Auberge[4] = {"|\0","/\0","-\0","\\\0"};
0426       int ProgressIndicator;
0427
0428       while(srcIndex < srcSize){
0429                 if(srcIndex>=REF_SIZE)
0430                           refStart=&src[srcIndex-REF_SIZE];
0431                 else
0432                           refStart=src;
0433                 if(srcIndex>=srcSize-ENC_SIZE)
0434                           encEnd=&src[srcSize];
0435                 else
0436                           encEnd=&src[srcIndex+ENC_SIZE];
0437
0438                 SearchIntoSlidingwindow(&index,&match,refStart,&src[srcIndex],&src[srcIndex],encEnd);
0439                 //if ( match<Min_Match_Length ) {
0440                 //if ( match<Min_Match_Length || match<8 ) {
0441                 if ( match==0 ) {
0442                           if(notMatch==0){
0443                                     notMatchStart=&ret[retIndex];
0444                                     retIndex++;
0445                           }
0446                           else if (notMatch==127) {
0447                                     *notMatchStart=(unsigned char)((127)<<1);
0448                                     notMatch=0;
0449                                     notMatchStart=&ret[retIndex];
0450                                     retIndex++;
0451                           }
0452                           ret[retIndex]=src[srcIndex];
0453                           retIndex++;
0454                           notMatch++;
0455                           srcIndex++;
0456                           if ((srcIndex-1) % (1<<17) > srcIndex % (1<<17)) {
0457                                     ProgressIndicator = (int)( (srcIndex+1)*(float)100/(srcSize+1) );
0458                                     printf("%s; Each rotation means 128KB are encoded; Done %d%%\r", Auberge[Melnitchka++], ProgressIndicator );
0459                                     Melnitchka = Melnitchka & 3; // 0 1 2 3: 00 01 10 11
0460                           }
0461                 } else {
0462                           if(notMatch > 0){
0463                                     *notMatchStart=(unsigned char)((notMatch)<<1);
0464                                     notMatch=0;
0465                           }
0466 // ---------------------|
0467 //                    \ /
0468
0469                           //ret[retIndex] = 0x80; // Assuming seventh/fifteenth bit is zero i.e. LONG MATCH i.e. Min_Match_BAILOUT_Length*4
0470                           //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
0471 //                     / \
0472 // ---------------------|
0473                           ret[retIndex] = 0x01; // Assuming seventh/fifteenth bit is zero i.e. LONG MATCH i.e. Min_Match_BAILOUT_Length*4
0474                           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
0475                           // 1bit+3bits+12bits:
```

```
0476                            //ret[retIndex] = ret[retIndex] | ((match-Min_Match_Length)<<4);
0477                            //ret[retIndex] = ret[retIndex] | (((index-Min_Match_Length) & 0x0F00)>>8);
0478                            // 1bit+1bit+14bits:
0479                            //ret[retIndex] = ret[retIndex] | ((match-Min_Match_Length)<<(8-(LengthBITS+1))); // No need to set the matchlength
0480 // 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.
0481 // --------------------|
0482 //                    \ /
0483                            //ret[retIndex] = ret[retIndex] | (((index-Min_Match_Length) & 0x3F00)>>8); // 2+4+8=14
0484                            //retIndex++;
0485                            //ret[retIndex] = (char)((index-Min_Match_Length) & 0x00FF);
0486                            //retIndex++;
0487 //                    / \
0488 // --------------------|
0489                            // Now the situation is like LOW:HIGH i.e. FF:3F i.e. 0x3FFF, 16bit&15bit used as flags,
0490                            // should become LOW:HIGH i.e. FC:FF i.e. 0xFFFC, 2bit&1bit used as flags.
0491                            ret[retIndex] = ret[retIndex] | (((index-Min_Match_Length) & 0x00FF)<<2); // 2+4+8=14
0492                            retIndex++;
0493                            ret[retIndex] = (char)(((index-Min_Match_Length) & 0x3FFF)>>6);
0494                            retIndex++;
0495 //                    / \
0496 // --------------------|
0497                            srcIndex+=match;
0498                            if ((srcIndex-match) % (1<<17) > srcIndex % (1<<17)) {
0499                                    ProgressIndicator = (int)( (srcIndex+1)*(float)100/(srcSize+1) );
0500                                    printf("%s; Each rotation means 128KB are encoded; Done %d%%\r", Auberge[Melnitchka++], ProgressIndicator );
0501                                    Melnitchka = Melnitchka & 3; // 0 1 2 3: 00 01 10 11
0502                            }
0503                    }
0504            }
0505            if(notMatch > 0){
0506                    *notMatchStart=(unsigned char)((notMatch)<<1);
0507            }
0508            printf("%s; Each rotation means 128KB are encoded; Done %d%%\n", Auberge[Melnitchka], 100 );
0509            return retIndex;
0510 }
0511
0512 unsigned int Decompress(char* ret, char* src, unsigned int srcSize){
0513            unsigned int srcIndex=0;
0514            unsigned int retIndex=0;
0515            unsigned int index=0;
0516            unsigned int match=0;
0517            unsigned int WORDpair;
0518
0519            while(srcIndex < srcSize){
0520                    //if((unsigned char)src[srcIndex] <= 127){
0521                    WORDpair = *(unsigned short int*)&src[srcIndex];
0522                    if((WORDpair & 0x01) == 0){
0523                            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
0524                            retIndex+=(WORDpair & 0xFF)>>1;
0525                            srcIndex+=(((WORDpair & 0xFF)>>1)+1);
0526                    }
0527                    else{
0528                            // 1bit+3bits+12bits:
0529                            //match = ((src[srcIndex] & 0x7F) >> 4)+Min_Match_Length;
0530                            //index = (src[srcIndex] & 0x0F) << 8;
0531                            // 1bit+1bit+14bits:
0532                            //match = ((src[srcIndex] & 0x4F) >> 4)+Min_Match_Length; // In fact, not needed when eightfoldness is commenced, match is 8.
0533 // The fragment below is outrageously ineffective - it can be done in one WORD operation instead of two BYTE operations.
0534 // --------------------|
0535 //                    \ /
0536                            //index = (src[srcIndex] & 0x3F) << 8;
0537                            //srcIndex++;
0538                            //index = (index | (unsigned int)(0x00FF & src[srcIndex])) + Min_Match_Length;
0539                            //srcIndex++;
0540 //                    / \
0541 // --------------------|
0542 // --------------------|
0543 //                    \ /
0544                            index = (WORDpair>>2) + Min_Match_Length;
0545                            srcIndex=srcIndex+2;
0546 //                    / \
0547 // --------------------|
0548                            //if (src[srcIndex-1-1] & 0x40) { // 4 if seventh/fifteenth bit is not zero
0549                            if (WORDpair & 0x02) { // 4 if 2/14 bit is not zero
0550                                    match = Min_Match_BAILOUT_Length;
0551                                    //memcpy(&ret[retIndex],&ret[retIndex-index],match);
0552                                    *(uint32_t*)(ret+retIndex) = *(uint32_t*)(ret+retIndex-index);
0553                            } else {
0554                                    match = Min_Match_BAILOUT_Length*4;
0555                                    //*(uint32_t*)(ret+retIndex) = *(uint32_t*)(ret+retIndex-index);
0556                                    //*(uint32_t*)(ret+retIndex+4) = *(uint32_t*)(ret+retIndex-index+4);
0557                                    //*(uint32_t*)(ret+retIndex+8) = *(uint32_t*)(ret+retIndex-index+8);
0558                                    //*(uint32_t*)(ret+retIndex+12) = *(uint32_t*)(ret+retIndex-index+12);
0559                                    SlowCopy128bit((ret+retIndex-index), (ret+retIndex));
0560                            }
0561                            retIndex+=match;
0562                    }
```

```
0563       }
0564       return retIndex;
0565 }
0566
0567 // Decompression main loop, 93-2a+2=107 bytes long:
0568 /*
0569 ; mark_description "Intel(R) C++ Compiler XE for applications running on IA-32, Version 12.1.1.258 Build 20111011";
0570 ; mark_description "-O3 -FAcs";
0571 .B7.3:
0572
0573 ;;;              //if((unsigned char)src[srcIndex] <= 127){
0574 ;;;              WORDpair = *(unsigned short int*)&src[srcIndex];
0575
0576   0002a 8d 0c 1e         lea ecx, DWORD PTR [esi+ebx]
0577   0002d 0f b7 01         movzx eax, WORD PTR [ecx]
0578
0579 ;;;              if((WORDpair & 0x01) == 0){
0580
0581   00030 a8 01            test al, 1
0582   00032 74 34            je .B7.8
0583
0584 .B7.4:
0585
0586 ;;;                        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
0587 ;;;                        retIndex+=(WORDpair & 0xFF)>>1;
0588 ;;;                        srcIndex+=(((WORDpair & 0xFF)>>1)+1);
0589 ;;;              }
0590 ;;;              else{
0591 ;;;                        // 1bit+3bits+12bits:
0592 ;;;                        //match = ((src[srcIndex] & 0x7F) >> 4)+Min_Match_Length;
0593 ;;;                        //index = (src[srcIndex] & 0x0F) << 8;
0594 ;;;                        // 1bit+1bit+14bits:
0595 ;;;                        //match = ((src[srcIndex] & 0x4F) >> 4)+Min_Match_Length; // In fact, not needed when eightfoldness is commenced, match is 8.
0596 ;;; // The fragment below is outrageously ineffective - it can be done in one WORD operation instead of two BYTE operations.
0597 ;;; // --------------------|
0598 ;;; //                     \ /
0599 ;;;                        //index = (src[srcIndex] & 0x3F) << 8;
0600 ;;;                        //srcIndex++;
0601 ;;;                        //index = (index | (unsigned int)(0x00FF & src[srcIndex])) + Min_Match_Length;
0602 ;;;                        //srcIndex++;
0603 ;;; //                     / \
0604 ;;; // --------------------|
0605 ;;; // --------------------|
0606 ;;; //                     \ /
0607 ;;;                        index = (WORDpair>>2) + Min_Match_Length;
0608 ;;;                        srcIndex=srcIndex+2;
0609 ;;; //                     / \
0610 ;;; // --------------------|
0611 ;;;                        //if (src[srcIndex-1-1] & 0x40) { // 4 if seventh/fifteenth bit is not zero
0612 ;;;                        if (WORDpair & 0x02) { // 4 if 2/14 bit is not zero
0613 ;;;                                  match = Min_Match_BAILOUT_Length;
0614 ;;;                                  //memcpy(&ret[retIndex],&ret[retIndex-index],match);
0615 ;;;                                  *(uint32_t*)(ret+retIndex) = *(uint32_t*)(ret+retIndex-index);
0616
0617   00034 8b 4d 08         mov ecx, DWORD PTR [8+ebp]
0618   00037 83 c3 02         add ebx, 2
0619   0003a 8d 14 39         lea edx, DWORD PTR [ecx+edi]
0620   0003d 8b c8            mov ecx, eax
0621   0003f c1 e9 02         shr ecx, 2
0622   00042 f7 d9            neg ecx
0623   00044 a8 02            test al, 2
0624   00046 74 0d            je .B7.6
0625
0626 .B7.5:
0627   00048 8b 4c 11 fc      mov ecx, DWORD PTR [-4+ecx+edx]
0628   0004c 89 0a            mov DWORD PTR [edx], ecx
0629   0004e b9 04 00 00 00   mov ecx, 4
0630   00053 eb 0f            jmp .B7.7
0631
0632 .B7.6:
0633
0634 ;;;                        } else {
0635 ;;;                                  match = Min_Match_BAILOUT_Length*4;
0636 ;;;                                  //*(uint32_t*)(ret+retIndex) = *(uint32_t*)(ret+retIndex-index);
0637 ;;;                                  //*(uint32_t*)(ret+retIndex+4) = *(uint32_t*)(ret+retIndex-index+4);
0638 ;;;                                  //*(uint32_t*)(ret+retIndex+8) = *(uint32_t*)(ret+retIndex-index+8);
0639 ;;;                                  //*(uint32_t*)(ret+retIndex+12) = *(uint32_t*)(ret+retIndex-index+12);
0640 ;;;                                  SlowCopy128bit((ret+retIndex-index), (ret+retIndex));
0641
0642   00055 f3 0f 6f 44 11
           fc              movdqu xmm0, XMMWORD PTR [-4+ecx+edx]
0644   0005b f3 0f 7f 02      movdqu XMMWORD PTR [edx], xmm0
0645   0005f b9 10 00 00 00   mov ecx, 16
0646
0647 .B7.7:
0648
0649   00064 03 f9            add edi, ecx
0650   00066 eb 28            jmp .B7.10
```

```
0651
0652 .B7.8:
0653   00068 8b 4d 08         mov ecx, DWORD PTR [8+ebp]
0654   0006b 0f b6 c0         movzx eax, al
0655   0006e d1 e8            shr eax, 1
0656   00070 89 44 24 0c      mov DWORD PTR [12+esp], eax
0657   00074 8d 14 39         lea edx, DWORD PTR [ecx+edi]
0658   00077 50               push eax
0659   00078 8d 4c 1e 01      lea ecx, DWORD PTR [1+esi+ebx]
0660   0007c 51               push ecx
0661   0007d 52               push edx
0662   0007e e8 fc ff ff ff   call __intel_fast_memcpy
0663
0664 .B7.15:
0665   00083 83 c4 0c         add esp, 12
0666
0667 .B7.9:
0668   00086 8b 4c 24 0c      mov ecx, DWORD PTR [12+esp]
0669   0008a 03 f9            add edi, ecx
0670   0008c 8d 5c 0b 01      lea ebx, DWORD PTR [1+ebx+ecx]
0671
0672 .B7.10:
0673   00090 3b 5d 10         cmp ebx, DWORD PTR [16+ebp]
0674   00093 72 95            jb .B7.3
0675 */
0676
0677 // The Speed Showdown on my 'Bonboniera' laptop (Core 2 T7500 2200MHz, Windows 7 64bit):
0678 // Grrr, in round #1 Yappy kicked my ass, yet, 4 more rounds remain...
0679 /*
0680 D:\_KAZE\Nakamichi_r1-RSSBO>NakamichiVsYappy.bat
0681 Speed Showdown: Nakamichi VS Yappy, both compiled with Intel v12.1 ...
0682
0683 D:\_KAZE\Nakamichi_r1-RSSBO>Yappy.exe enwiki-20140304-pages-articles.7z.001 1024
0684 YAPPY: [b 1K] bytes 104857600 -> 78039768  74.4%  comp  42.7 MB/s  uncomp 680.3 MB/s
0685
0686 D:\_KAZE\Nakamichi_r1-RSSBO>Yappy.exe enwiki-20140304-pages-articles.7z.001 2048
0687 YAPPY: [b 2K] bytes 104857600 -> 70845249  67.6%  comp  39.8 MB/s  uncomp 623.3 MB/s
0688
0689 D:\_KAZE\Nakamichi_r1-RSSBO>Yappy.exe enwiki-20140304-pages-articles.7z.001 4096
0690 YAPPY: [b 4K] bytes 104857600 -> 64270963  61.3%  comp  36.2 MB/s  uncomp 583.1 MB/s
0691
0692 D:\_KAZE\Nakamichi_r1-RSSBO>Yappy.exe enwiki-20140304-pages-articles.7z.001 8192
0693 YAPPY: [b 8K] bytes 104857600 -> 59756354  57.0%  comp  32.9 MB/s  uncomp 549.5 MB/s
0694
0695 D:\_KAZE\Nakamichi_r1-RSSBO>Yappy.exe enwiki-20140304-pages-articles.7z.001 16384
0696 YAPPY: [b 16K] bytes 104857600 -> 57497885  54.8%  comp  31.4 MB/s  uncomp 541.5 MB/s
0697
0698 D:\_KAZE\Nakamichi_r1-RSSBO>Yappy.exe enwiki-20140304-pages-articles.7z.001 32768
0699 YAPPY: [b 32K] bytes 104857600 -> 56365696  53.8%  comp  31.1 MB/s  uncomp 541.5 MB/s
0700
0701 D:\_KAZE\Nakamichi_r1-RSSBO>Yappy.exe enwiki-20140304-pages-articles.7z.001 65536
0702 YAPPY: [b 64K] bytes 104857600 -> 55799079  53.2%  comp  31.3 MB/s  uncomp 534.3 MB/s
0703
0704 D:\_KAZE\Nakamichi_r1-RSSBO>Yappy.exe enwiki-20140304-pages-articles.7z.001 262144
0705 YAPPY: [b 256K] bytes 104857600 -> 55377127  52.8%  comp  31.4 MB/s  uncomp 534.3 MB/s
0706
0707 D:\_KAZE\Nakamichi_r1-RSSBO>Nakamichi.exe enwiki-20140304-pages-articles.7z.001
0708 Nakamichi, revision 1-RSSBO, written by Kaze.
0709 Compressing 104857600 bytes ...
0710 -; Each rotation means 128KB are encoded; Done 100%
0711 RAM-to-RAM performance: 198 KB/s.
0712
0713 D:\_KAZE\Nakamichi_r1-RSSBO>Nakamichi.exe enwiki-20140304-pages-articles.7z.001.Nakamichi
0714 Nakamichi, revision 1-RSSBO, written by Kaze.
0715 Decompressing 70533827 bytes ...
0716 RAM-to-RAM performance: 531 MB/s.
0717
0718 D:\_KAZE\Nakamichi_r1-RSSBO>
0719 */
0720
0721 // In my opinion Hamid Buzidi is the best, therefore his lzturbo v1.1 reference results are given below:
0722 /*
0723 D:\_KAZE\Nakamichi_r1-RSSBO>timer32 lzturbo.exe -19 -p0 enwiki-20140304-pages-articles.7z.001 .
0724
0725 Kernel  Time =     0.982 =    0%
0726 User    Time =   152.537 =   99%
0727 Process Time =   153.520 =  100%    Virtual  Memory =    429 MB
0728 Global  Time =   153.519 =  100%    Physical Memory =    407 MB
0729
0730 D:\_KAZE\Nakamichi_r1-RSSBO>timer32.exe lzturbo.exe -d enwiki-20140304-pages-articles.7z.001.lzt .
0731
0732 Kernel  Time =     0.234 =   62%
0733 User    Time =     0.187 =   50%
0734 Process Time =     0.421 =  112%    Virtual  Memory =     98 MB
0735 Global  Time =     0.374 =  100%    Physical Memory =     70 MB
0736
0737 D:\_KAZE\Nakamichi_r1-RSSBO>dir
0738
0739 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;
	register uint32_t ulHashPattern;
	signed long count;

	unsigned char bm_Horspool_Order2[256*256]; // Bitwise soon...
	uint32_t i, Gulliver;

	uint32_t PRIMALposition, PRIMALpositionCANDIDATE;
	uint32_t PRIMALlength, PRIMALlengthCANDIDATE;
	uint32_t j, FoundAtPosition;

	if (cbPattern > cbTarget) return(NULL);

	if ( cbPattern<4 ) {
		// SSE2 i.e. 128bit Assembly rules here:
		// ...
		pbTarget = pbTarget+cbPattern;
		ulHashPattern = ( (*(char *)(pbPattern))<<8 ) + *(pbPattern+(cbPattern-1));
		if ( cbPattern==3 ) {
			for ( ;; ) {
				if ( ulHashPattern == ( (*(char *)(pbTarget-3))<<8 ) + *(pbTarget-1) ) {
					if ( *(char *)(pbPattern+1) == *(char *)(pbTarget-2) ) return((pbTarget-3));
				}
				if ( (char)(ulHashPattern>>8) != *(pbTarget-2) ) {
					pbTarget++;
					if ( (char)(ulHashPattern>>8) != *(pbTarget-2) ) pbTarget++;
				}
				pbTarget++;
				if (pbTarget > pbTargetMax) return(NULL);
			}
		} else {
		}
		for ( ;; ) {
			if ( ulHashPattern == ( (*(char *)(pbTarget-2))<<8 ) + *(pbTarget-1) ) return((pbTarget-2));
			if ( (char)(ulHashPattern>>8) != *(pbTarget-1) ) pbTarget++;
			pbTarget++;
			if (pbTarget > pbTargetMax) return(NULL);
		}
	} else { //if ( cbPattern<4 )
		if ( cbPattern<=NeedleThreshold2vs4swampLITE ) {
			// BMH order 2, needle should be >=4:
			ulHashPattern = *(uint32_t *)(pbPattern); // First four bytes
			for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]=0;}
			for (i=0; i < cbPattern-1; i++) bm_Horspool_Order2[*(unsigned short *)(pbPattern+i)]=1;
			i=0;
			while (i <= cbTarget-cbPattern) {
				Gulliver = 1; // 'Gulliver' is the skip
				if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+cbPattern-1-1]] != 0 ) {
					if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+cbPattern-1-1-2]] == 0 ) Gulliver = cbPattern-(2-1)-2; else
{
						if ( *(uint32_t *)&pbTarget[i] == ulHashPattern) { // This fast check ensures not missing a match (for
remainder) when going under 0 in loop below:
							count = cbPattern-4+1;
							while ( count > 0 && *(uint32_t *)(pbPattern+count-1) == *(uint32_t *)(&pbTarget[i]+(count-
1)) )
								count = count-4;
							if ( count <= 0 ) return(pbTarget+i);
						}
					}
				} else Gulliver = cbPattern-(2-1);
```

```
0826                                    i = i + Gulliver;
0827                                    //GlobalI++; // Comment it, it is only for stats.
0828                            }
0829                            return(NULL);
0830                    } else { // if ( cbPattern<=NeedleThreshold2vs4swampLITE )
0831
0832 // Swampwalker_BAILOUT heuristic order 4 (Needle should be bigger than 4) [
0833 // Needle: 1234567890qwertyuiopasdfghjklzxcv           PRIMALposition=01 PRIMALength=33  '1234567890qwertyuiopasdfghjklzxcv'
0834 // Needle: vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv           PRIMALposition=29 PRIMALength=04  'vvvv'
0835 // Needle: vvvvvvvvvvBOOMSHAKALAKAvvvvvvvvvv          PRIMALposition=08 PRIMALength=20  'vvvBOOMSHAKALAKAvvvv'
0836 // Needle: Trollland                                  PRIMALposition=01 PRIMALength=09  'Trollland'
0837 // Needle: Swampwalker                                PRIMALposition=01 PRIMALength=11  'Swampwalker'
0838 // Needle: licenselessness                            PRIMALposition=01 PRIMALength=15  'licenselessness'
0839 // Needle: alfalfa                                    PRIMALposition=02 PRIMALength=06  'lfalfa'
0840 // Needle: Sandokan                                   PRIMALposition=01 PRIMALength=08  'Sandokan'
0841 // Needle: shazamish                                  PRIMALposition=01 PRIMALength=09  'shazamish'
0842 // Needle: Simplicius Simplicissimus                  PRIMALposition=06 PRIMALength=20  'icius Simplicissimus'
0843 // Needle: domilliaquadringenquattuorquinquagintillion  PRIMALposition=01 PRIMALength=32  'domilliaquadringenquattuorquinqu'
0844 // Needle: boom-boom                                  PRIMALposition=02 PRIMALength=08  'oom-boom'
0845 // Needle: vvvvv                                      PRIMALposition=01 PRIMALength=04  'vvvv'
0846 // Needle: 12345                                      PRIMALposition=01 PRIMALength=05  '12345'
0847 // Needle: likey-likey                                PRIMALposition=03 PRIMALength=09  'key-likey'
0848 // Needle: BOOOOOM                                    PRIMALposition=03 PRIMALength=05  'OOOOM'
0849 // Needle: aaaaaBOOOOOM                               PRIMALposition=02 PRIMALength=09  'aaaaBOOOO'
0850 // Needle: BOOOOOMaaaaa                               PRIMALposition=03 PRIMALength=09  'OOOOMaaaa'
0851 PRIMALength=0;
0852 for (i=0+(1); i < cbPattern-((4)-1)+(1)-(1); i++) { // -(1) because the last BB order 4 has no counterpart(s)
0853       FoundAtPosition = cbPattern - ((4)-1) + 1;
0854       PRIMALpositionCANDIDATE=i;
0855       while ( PRIMALpositionCANDIDATE <= (FoundAtPosition-1) ) {
0856                 j = PRIMALpositionCANDIDATE + 1;
0857                 while ( j <= (FoundAtPosition-1) ) {
0858                           if ( *(uint32_t *)(pbPattern+PRIMALpositionCANDIDATE-(1)) == *(uint32_t *)(pbPattern+j-(1)) ) FoundAtPosition = j;
0859                           j++;
0860                 }
0861                 PRIMALpositionCANDIDATE++;
0862       }
0863       PRIMALengthCANDIDATE = (FoundAtPosition-1)-i+1 +((4)-1);
0864       if (PRIMALengthCANDIDATE >= PRIMALength) {PRIMALposition=i; PRIMALength = PRIMALengthCANDIDATE;}
0865       if (cbPattern-i+1 <= PRIMALength) break;
0866       if (PRIMALength > 128) break; // Bail Out for 129[+]
0867 }
0868 // Swampwalker_BAILOUT heuristic order 4 (Needle should be bigger than 4) ]
0869
0870 // Here we have 4 or bigger NewNeedle, apply order 2 for pbPattern[i+(PRIMALposition-1)] with length 'PRIMALength' and compare the pbPattern[i] with length
'cbPattern':
0871 PRIMALengthCANDIDATE = cbPattern;
0872 cbPattern = PRIMALength;
0873 pbPattern = pbPattern + (PRIMALposition-1);
0874
0875 // Revision 2 commented section [
0876 /*
0877 if (cbPattern-1 <= 255) {
0878 // BMH Order 2 [
0879                           ulHashPattern = *(uint32_t *)(pbPattern); // First four bytes
0880                           for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]= cbPattern-1;} // cbPattern-(Order-1) for Horspool; 'memset' if not optimized
0881                           for (i=0; i < cbPattern-1; i++) bm_Horspool_Order2[*(unsigned short *)(pbPattern+i)]=i; // Rightmost appearance/position is needed
0882                           i=0;
0883                           while (i <= cbTarget-cbPattern) {
0884                                     Gulliver = bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+cbPattern-1-1]];
0885                                     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
0886                                     if ( Gulliver == cbPattern-2 ) { // CASE #1: means the pair (char order 2) is found
0887                                               if ( *(uint32_t *)&pbTarget[i] == ulHashPattern) {
0888                                                         count = cbPattern-4+1;
0889                                                         while ( count > 0 && *(uint32_t *)(pbPattern+count-1) == *(uint32_t *)(&pbTarget[i]+(count-1)) )
0890                                                                   count = count-4;
0891 // If we miss to hit then no need to compare the original: Needle
0892 if ( count <= 0 ) {
0893 // I have to add out-of-range checks...
0894 // i-(PRIMALposition-1) >= 0
0895 // &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
0896 // i-(PRIMALposition-1)+(count-1) >= 0
0897 // &pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4
0898       if ( (i-(PRIMALposition-1) >= 0) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) && (&pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4) ) {
0899                 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:
0900                           count = PRIMALengthCANDIDATE-4+1;
0901                           while ( count > 0 && *(uint32_t *)(pbPattern-(PRIMALposition-1)+count-1) == *(uint32_t *)(&pbTarget[i-(PRIMALposition-1)]+(count-1)) )
0902                                     count = count-4;
0903                           if ( count <= 0 ) return(pbTarget+i-(PRIMALposition-1));
0904                 }
0905       }
0906 }
0907                                               }
0908                                               Gulliver = 1;
0909                                     } else
0910                                               Gulliver = cbPattern - Gulliver - 2; // CASE #3: the pair is found and not as suffix i.e. rightmost position
0911                                     }
```

```
0912                                        i = i + Gulliver;
0913                                        //GlobalI++; // Comment it, it is only for stats.
0914                                }
0915                                return(NULL);
0916 // BMH Order 2 ]
0917 } else {
0918                                // BMH order 2, needle should be >=4:
0919                                ulHashPattern = *(uint32_t *)(pbPattern); // First four bytes
0920                                for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]=0;}
0921                                for (i=0; i < cbPattern-1; i++) bm_Horspool_Order2[*(unsigned short *)(pbPattern+i)]=1;
0922                                i=0;
0923                                while (i <= cbTarget-cbPattern) {
0924                                        Gulliver = 1; // 'Gulliver' is the skip
0925                                        if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+cbPattern-1-1]] != 0 ) {
0926                                                if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+cbPattern-1-1-2]] == 0 ) Gulliver = cbPattern-(2-1)-2; else
{
0927                                                        if ( *(uint32_t *)&pbTarget[i] == ulHashPattern) { // This fast check ensures not missing a match (for
remainder) when going under 0 in loop below:
0928                                                                count = cbPattern-4+1;
0929                                                                while ( count > 0 && *(uint32_t *)(pbPattern+count-1) == *(uint32_t *)(&pbTarget[i]+(count-
1)) )
0930                                                                        count = count-4;
0931 // If we miss to hit then no need to compare the original: Needle
0932 if ( count <= 0 ) {
0933 // I have to add out-of-range checks...
0934 // i-(PRIMALposition-1) >= 0
0935 // &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
0936 // i-(PRIMALposition-1)+(count-1) >= 0
0937 // &pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4
0938      if ( (i-(PRIMALposition-1) >= 0) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) && (&pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4) ) {
0939                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:
0940                                count = PRIMALlengthCANDIDATE-4+1;
0941                                while ( count > 0 && *(uint32_t *)(pbPattern-(PRIMALposition-1)+count-1) == *(uint32_t *)(&pbTarget[i-(PRIMALposition-1)]+(count-1)) )
0942                                        count = count-4;
0943                                if ( count <= 0 ) return(pbTarget+i-(PRIMALposition-1));
0944                }
0945      }
0946 }
0947                                                        }
0948                                                }
0949                                        } else Gulliver = cbPattern-(2-1);
0950                                        i = i + Gulliver;
0951                                        //GlobalI++; // Comment it, it is only for stats.
0952                                }
0953                                return(NULL);
0954 }
0955 */
0956 // Revision 2 commented section ]
0957
0958                if ( cbPattern<=NeedleThreshold2vs4swampLITE ) {
0959
0960                                // BMH order 2, needle should be >=4:
0961                                ulHashPattern = *(uint32_t *)(pbPattern); // First four bytes
0962                                for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]=0;}
0963                                for (i=0; i < cbPattern-1; i++) bm_Horspool_Order2[*(unsigned short *)(pbPattern+i)]=1;
0964                                i=0;
0965                                while (i <= cbTarget-cbPattern) {
0966                                        Gulliver = 1; // 'Gulliver' is the skip
0967                                        if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+cbPattern-1-1]] != 0 ) {
0968                                                if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+cbPattern-1-1-2]] == 0 ) Gulliver = cbPattern-(2-1)-2; else
{
0969                                                        if ( *(uint32_t *)&pbTarget[i] == ulHashPattern) { // This fast check ensures not missing a match (for
remainder) when going under 0 in loop below:
0970                                                                count = cbPattern-4+1;
0971                                                                while ( count > 0 && *(uint32_t *)(pbPattern+count-1) == *(uint32_t *)(&pbTarget[i]+(count-
1)) )
0972                                                                        count = count-4;
0973 // If we miss to hit then no need to compare the original: Needle
0974 if ( count <= 0 ) {
0975 // I have to add out-of-range checks...
0976 // i-(PRIMALposition-1) >= 0
0977 // &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
0978 // i-(PRIMALposition-1)+(count-1) >= 0
0979 // &pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4
0980      if ( (i-(PRIMALposition-1) >= 0) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) && (&pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4) ) {
0981                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:
0982                                count = PRIMALlengthCANDIDATE-4+1;
0983                                while ( count > 0 && *(uint32_t *)(pbPattern-(PRIMALposition-1)+count-1) == *(uint32_t *)(&pbTarget[i-(PRIMALposition-1)]+(count-1)) )
0984                                        count = count-4;
0985                                if ( count <= 0 ) return(pbTarget+i-(PRIMALposition-1));
0986                }
0987      }
0988 }
0989                                                        }
0990                                                }
0991                                        } else Gulliver = cbPattern-(2-1);
0992                                        i = i + Gulliver;
```

```
0993                                        //GlobalI++; // Comment it, it is only for stats.
0994                                }
0995                                return(NULL);
0996
0997                        } else { // if ( cbPattern<=NeedleThreshold2vs4swampLITE )
0998
0999                                // BMH pseudo-order 4, needle should be >=8+2:
1000                                ulHashPattern = *(uint32_t *)(pbPattern); // First four bytes
1001                                for (i=0; i < 256*256; i++) {bm_Horspool_Order2[i]=0;}
1002                                // 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:
1003                                //"fast"
1004                                //"aste"
1005                                //"stes"
1006                                //"test"
1007                                //"est "
1008                                //"st f"
1009                                //"t fo"
1010                                //" fox"
1011                                //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;
1012                                //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;
1013                                // Above line is replaced by next one with better hashing:
1014                                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;
1015                                i=0;
1016                                while (i <= cbTarget-cbPattern) {
1017                                        Gulliver = 1;
1018                                        //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
1019                                        // Above line is replaced by next one with better hashing:
1020                                        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
1021                                                //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 {
1022                                                // 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'.
1023                                                // The respective offsets (backwards) are: -10/-8/-6/-4 for 'xa'/'ab'/'bc'/'cd'.
1024                                                //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 {
1025                                                // Above line is replaced by next one with better hashing:
1026                                                // 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.
1027                                                // Example for genomesque pair TT+TT being shifted by (16-1):
1028                                                // T            = 01010100
1029                                                // TT           = 01010100 01010100
1030                                                // TTTT         = 01010100 01010100 01010100 01010100
1031                                                // TTTT>>16     = 00000000 00000000 01010100 01010100
1032                                                // 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.
1033                                                //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 {
1034                                                // 'Maximus' uses branched 'if', again.
1035                                                if ( \
1036                                                ( 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 \
1037                                                || ( 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 \
1038                                                ) Gulliver = cbPattern-(2-1)-2-4-2 +1; else {
1039                                                // 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:
1040 // [2x][2a][2b][2c][2d]
1041 // DWORD #4
1042 // [2a] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-6]>>16) =     !SHR to be avoided! <--
1043 // [2x] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-6]&0xFFFF) =                       |
1044 //     DWORD #3                                                                    |
1045 // [2b] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]>>16) =     !SHR to be avoided!   |<--
1046 // [2a] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF) = ------------------------  |
1047 //         DWORD #2                                                                    |
1048 // [2c] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]>>16) =     !SHR to be avoided!     |<--
1049 // [2b] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]&0xFFFF) = --------------------------  |
1050 //             DWORD #1                                                                    |
1051 // [2d] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-0]>>16) =                                |
1052 // [2c] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-0]&0xFFFF) = ------------------------------
1053 //
1054 // So in order to remove 3 SHR instructions the equal extractions are:
1055 // DWORD #4
1056 // [2a] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF) =  !SHR to be avoided! <--
1057 // [2x] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-6]&0xFFFF) =                       |
1058 //     DWORD #3                                                                    |
1059 // [2b] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]&0xFFFF) =  !SHR to be avoided!   |<--
1060 // [2a] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-4]&0xFFFF) = ------------------------  |
1061 //         DWORD #2                                                                    |
1062 // [2c] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-0]&0xFFFF) =  !SHR to be avoided!     |<--
```

```
1063 // [2b] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-2]&0xFFFF) = ----------------------------  |
1064 //            DWORD #1                                                                        |
1065 // [2d] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-0]>>16) =                                    |
1066 // [2c] (*(uint32_t *)&pbTarget[i+cbPattern-1-1-2-0]&0xFFFF) = ------------------------------
1067                                                 //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 {
1068 // Since the above Decumanus mumbo-jumbo (3 overlapping lookups vs 2 non-overlapping lookups) is not fast enough we go DuoDecumanus or 3x4:
1069 // [2y][2x][2a][2b][2c][2d]
1070 // DWORD #3
1071 //         DWORD #2
1072 //                 DWORD #1
1073                                                 //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 {
1074                                                 if ( *(uint32_t *)&pbTarget[i] == ulHashPattern) {
1075                                                         // Order 4 [
1076                                                 // Let's try something "outrageous" like comparing with[out] overlap BBs 4bytes long instead of 1 byte
back-to-back:
1077                                                 // 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:
1078                                                 //0:"fast" if the comparison failed here, 'count' is 1; 'Gulliver' is cbPattern-(4-1)-7
1079                                                 //1:"aste" if the comparison failed here, 'count' is 2; 'Gulliver' is cbPattern-(4-1)-6
1080                                                 //2:"stes" if the comparison failed here, 'count' is 3; 'Gulliver' is cbPattern-(4-1)-5
1081                                                 //3:"test" if the comparison failed here, 'count' is 4; 'Gulliver' is cbPattern-(4-1)-4
1082                                                 //4:"est " if the comparison failed here, 'count' is 5; 'Gulliver' is cbPattern-(4-1)-3
1083                                                 //5:"st f" if the comparison failed here, 'count' is 6; 'Gulliver' is cbPattern-(4-1)-2
1084                                                 //6:"t fo" if the comparison failed here, 'count' is 7; 'Gulliver' is cbPattern-(4-1)-1
1085                                                 //7:" fox" if the comparison failed here, 'count' is 8; 'Gulliver' is cbPattern-(4-1)
1086                                                         count = cbPattern-4+1;
1087                                                         // Below comparison is UNIdirectional:
1088                                                         while ( count > 0 && *(uint32_t *)(pbPattern+count-1) == *(uint32_t *)(&pbTarget[i]+(count-
1)) )
1089                                                                 count = count-4;
1090 // count = cbPattern-4+1 = 23-4+1 = 20
1091 // boomshakalakaZZZZZZ[ZZZZ] 20
1092 // boomshakalakaZZ[ZZZZ]ZZZZ 20-4
1093 // boomshakala[kaZZ]ZZZZZZZZ 20-8 = 12
1094 // boomsha[kala]kaZZZZZZZZZZ 20-12 = 8
1095 // boo[msha]kalakaZZZZZZZZZZ 20-16 = 4
1096
1097 // If we miss to hit then no need to compare the original: Needle
1098 if ( count <= 0 ) {
1099 // I have to add out-of-range checks...
1100 // i-(PRIMALposition-1) >= 0
1101 // &pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4
1102 // i-(PRIMALposition-1)+(count-1) >= 0
1103 // &pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4
1104       if ( (i-(PRIMALposition-1) >= 0) && (&pbTarget[i-(PRIMALposition-1)] <= pbTargetMax - 4) && (&pbTarget[i-(PRIMALposition-1)+(count-1)] <= pbTargetMax - 4) ) {
1105                 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:
1106                         count = PRIMALlengthCANDIDATE-4+1;
1107                         while ( count > 0 && *(uint32_t *)(pbPattern-(PRIMALposition-1)+count-1) == *(uint32_t *)(&pbTarget[i-(PRIMALposition-1)]+(count-1)) )
1108                                 count = count-4;
1109                         if ( count <= 0 ) return(pbTarget+i-(PRIMALposition-1));
1110                 }
1111       }
1112 }
1113
1114                                                         // 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.
1115                                                         // Below comparison is BIdirectional. It pays off when needle is 8+++ long:
1116                                                         for (count = cbPattern-4+1; count > 0; count = count-4) {
1117 //                                                              if ( *(uint32_t *)(pbPattern+count-1) != *(uint32_t *)(&pbTarget[i]+(count-1)) )
{break;};
1118 //                                                              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)
1119 //                                                              }
1120 //                                                              if ( count <= 0 ) return(pbTarget+i);
1121                                                                 // Checking the order 2 pairs in mismatched DWORD, all the 3:
1122                                                                 //if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+count-1]] == 0 )
Gulliver = count; // 1 or bigger, as it should
1123                                                                 //if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+count-1+1]] == 0 )
Gulliver = count+1; // 1 or bigger, as it should
1124                                                                 //if ( bm_Horspool_Order2[*(unsigned short *)&pbTarget[i+count-1+1+1]] == 0 )
Gulliver = count+1+1; // 1 or bigger, as it should
1125                                                         //      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)
1126                                                                 // Above compound 'if' guarantees not that Gulliver > 1, an example:
1127                                                                 // Needle:    fastest tax
1128                                                                 // Window: ...fastast tax...
1129                                                                 // After matching ' tax' vs ' tax' and 'fast' vs 'fast' the mismathced DWORD is
'test' vs 'tast':
1130                                                                 // '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.
1131                                                                 // 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'.
                                                                                //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
                                                                                // Above line is replaced by next one with better hashing:
// //                                                                           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
                                                                                // Order 4 ]
                                                                }
                                                        }
                                        } else Gulliver = cbPattern-(2-1)-2; // -2 because we check the 4 rightmost bytes not 2.
                                        i = i + Gulliver;
                                        //GlobalI++; // Comment it, it is only for stats.
                                }
                                return(NULL);

                } // if ( cbPattern<=NeedleThreshold2vs4swampLITE )
                } // if ( cbPattern<=NeedleThreshold2vs4swampLITE )
        } //if ( cbPattern<4 )
}

// Last change: 2014-Apr-17
// If you want to help me to improve it, email me at: sanmayce@sanmayce.com
// 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/