```
/*+1============================================================================*/
/* MODULE                        TBM.C                                        */
/*============================================================================*/
/* FUNCTION     Tuned Boyer-Moore (BM) algorithm for fast string searching; -
 *              This module implements a compact, portable and fast BM
 *              algorithm (ie. a classic BM including a skip-loop).
 *
 *              Search algorithms have a common structure performing a repeated
 *              loop through a sequence of positions in the text being searched
 *              and testing whether the pattern does or does not match the text
 *              at each position :
 *              FOR pos in text DO {        // SEARCH LOOP:
 *                 skiploop                 // 1: skip past immediate mismatch
 *                 match(pattern,text[pos]) // 2: compare pattern against text
 *                 pos += shift(pos, off)   // 3: text[pos+off] != pattern[off]
 *              }
 *
 *              Several choices of algorithm exist for the 3 steps of the
 *              search loop : (1) skip, (2) match and (3) shift.
 *              In the Tuned BM (TBM) we use the following components :
 *              (1) TBM skip  : portable, unrolled variant of search forward
 *                              for last (rightmost) char in pattern; (fast BM)
 *              (2) TBM match : test guard before forward (l->r) linear scan.
 *              (3) TBM shift : "mini Sunday DELTA-2(pattern)".
 *              For a further description of these (and alternative) components
 *              cf. the references given below.
 *
 *   SYSTEM     Standard (ANSI/ISO) C
 *              Tested on PC/MS DOS V.3.3 and UNIX Sys V.3, cf. makefile
 *
 *   SEE ALSO   Modules : TBM.H, MAKEFILE
 *
 *   PROGRAMMER Allan Dystrup (AD)
 *
 *   COPYRIGHT  (c) Allan Dystrup, Kommunedata I/S, March 1992.
 *
 *   VERSION    $Header: d:/cwork/index/tbm/RCS/tbm.c 1.3 92/02/27 15:11:30
 *              Allan_Dystrup Exp Locker: Allan_Dystrup $
 *              $Log:   tbm.c $
 *                      Revision 1.5  92/03/13  13:34:44  Allan_Dystrup
 *                      linted DOS/UX
 *              Revision 1.4  92/03/13  11:15:25  Allan_Dystrup
 *              integrated UCASE
 *              Revision 1.3  92/03/13  08:40:32  Allan_Dystrup
 *              main ucase
 *              Revision 1.2  92/03/11  14:54:40  Allan_Dystrup
 *              hack'ed UCASE
 *                  Revision 1.1  92/03/02  13:40:40  Allan_Dystrup
 *                  Initial revision
 *              Revision 1.5  92/02/28  11:54:16  Allan_Dystrup
 *                  S-L-O-W ucase
 *                  Revision 1.4  92/02/28  10:55:34  Allan_Dystrup
 *                  fbuf & ucase (hack)
 *              Revision 1.3  92/02/27  15:11:30  Allan_Dystrup
 *              *** empty log message ***
 *
```

```
*   REFERENCES The references concentrate on the Boyer-Moore family of string
*              search algorithms, which is superior for single string matching;
*              Other strategies for string search (eg. KMP for single string
*              match, Aho-Corasic for parallel string match and general NFA/DFA
*              for regular expressions) are NOT considered. Please consult an
*              algorithm textbook for a broad introduction to these techniques,
*              (for instance Sedgewick, R. [1983] : "Algorithms", Addison-
*              Wesley, August 1984).
*
*              Boyer, R.S and J.S. Moore [1977] : "A Fast String Searching
*              Algorithm", CACM, Vol.20, No.10, October 1977.
*
*              Schaback, R. [1988] : "On the Expected Sublinearity of the BM
*              Algorithm", SIAM J. COMPUT, Vol.17, No.4, August 1988.
*
*              Rivest, R. [1979] : "On Improving the Worst Case Running Time of
*              the BM String Matching Algorithm", CACM, Vol.22, No.9,Sept.1979
*
*              Sunday, D.M. [1990] : "A Very Fast Substring Search Algorithm",
*              Communications of the ACM, Vol.33, No.8, August 1990.
*
*              Smith, P.D. [1991] : "Experiments with a Very Fast Substring
*              Search Algorithm", Software-Practice and Experience,
*              Vol.21(10), 1065-1074, October 1991
*
*              Hume, A and D. Sunday [1991] : "Fast String Searching",
*              Algorithm", Software-Practice and Experience,
*              Vol.21(11), 1221-1248, November 1991.
*
*   USAGE      Module tbm.c features the following public routines for
*              performing fast string search in text :
*                 vBuildTBM()     // prepare pattern for search function
*                 iRunTBM()       // perform search for pattern in text
*              See headerfile tbm.h and the documentation in this file for
*              a detailed description of the user accessible compile switches,
*              datastructures and functions.
*
*   DOC        Documentation is incorporated into the module and may be
*              selectively extracted (using a utility such as ex.awk) :
*                 Level 1: Module documentation (history, design, testdriver)
*                 Level 2: PUBLIC functions (module program interface, "API")
*                 Level 3: major PRIVATE functions (design)
*                 Level 4: minor PRIVATE functions (support)
*
*   BUGS       The module contains highly optimized code for maximum speed;
*              As a consequence the algorithms may not be immediately compre-
*              hensible. I have tried to counter this effect by an extensive
*              documentation of the program, but if in doubt, please consult
*              the references.
*
*-1=============================================================================*/
```

```c
#include   <stdio.h>
#include   <stdlib.h>
#include   <string.h>
#include   <ctype.h>
#include   "tbm.h"


PRIVATE BYTE
        bDKupper(int ii);



#ifdef MAIN
/*****************************************************************************/
/******************************** MAIN ***************************************/
/*****************************************************************************/
/* Define "signon message" for module tbm.c */
PRIVATE char SIGNON[] =
   "\nKMD TBM Search Functions (Testdriver), Version 0.1\n"
   "MOD[tbm.c] VER[0.1 Exp] DAT[92/02/27] DEV[ad dec]\n"
   "Copyright (c) KommuneData I/S 1992\n\n";


/* Define "usage message" for module tbm.c */
PRIVATE char USAGE[] =
   "\n\aUSAGE:\ttbm  <pattern>  <file>\n"
   "where \t<pattern>  is string to find\n"
   "      \t<file>     is file to search\n\n";


#define MAXLIN 512     /* Max. #chars in one line of text */
#define MAXPAT 256     /* Max. #chars in search pattern */



/*+4 MODULE TBM.C ------------------------------------------------------------*/
/*   NAME                         CHKERR                                     */
/*-- SYNOPSIS ------------------------------------------------------------*/
/* DESCRIPTION
 *   Simple macro for main() testdriver providing error test & diagnostic.
 *-4*/
#define CHKERR(expr, msg) {                                                  \
   if (expr) {                                                               \
       fprintf(stderr,                                                       \
          "\n\aERROR: File[%s]-Line[%d]  Date[%s]-Time[%s]\nCAUSE: %s\n", \
                  __FILE__, __LINE__, __DATE__, __TIME__, msg);          \
          exit(EXIT_FAILURE);                                                \
   }                                                                         \
}
```

```
/*+1 MODULE TBM.C ============================================================*/
/*   NAME   00                    main                                        */
/*== SYNOPSIS ================================================================*/
int main(argc, argv)
   int   argc;         /* Argument count : should be 3, cf. argv[]   */
   char  *argv[];      /* Argument vector: <tbm>, <pattern>, <file> */
{
/* DESCRIPTION
 *   Testdriver for module tbm.c ; -
 *
 *    I: The testdriver exercises the functions in the module, and validates
 *       the functionality through trace-statements (compiled with -DDEBUG).
 *       1: Test reaction to boundary conditions : Empty pattern/textbuffer.
 *
 *   II: The testdriver may be used as a stand alone utility with a simple
 *       commandline interface for searching a file for a textstring :
 *       tbm <pattern> <file>, - (cf. the "USAGE" description above).
 *       1: Signon & check args
 *       2: Open textfile to search
 *       3: Prepare pattern for TBM search
 *       4: Perform TBM search on textfile, one line at a time, until EOF
 *       5: Clean up & Terminate
 *-1*/

    FILE     *fdTxt;                  /* File handle for textfile */
    BYTE      cBuf[MAXLIN+MAXPAT];    /* Linebuffer for textfile  */
    DWORD     dwCount = 0L;           /* Linecount for textlines  */
    int       iRet;                   /* Integer return code      */


#ifdef DEBUG
    /*---------------------------------------------------------------------*/
    /*  I: INTERNAL TEST PART OF MAIN() TESTDRIVER                         */

    /* 1: Test reaction to boundary conditions : Empty pattern/textbuffer.   */

    strcpy((char *)cBuf, "");                          /* 1.1: Empty pattern */
    vBuildTBM(cBuf, strlen(cBuf));

    strcpy((char *)cBuf, "Dummy\n");
    if ( iRunTBM(cBuf, strlen((char*)cBuf)) )
       printf("[%05lu]\t%s", dwCount, cBuf);


    strcpy((char *)cBuf, "Dummy");                 /* 1.2: Empty textbuffer */
    vBuildTBM(cBuf, strlen(cBuf));

    strcpy((char *)cBuf, "");
    if ( iRunTBM(cBuf, strlen((char*)cBuf)) )
       printf("[%05lu]\t%s", dwCount, cBuf);
#endif /* DEBUG */


    /*---------------------------------------------------------------------*/
    /* II: APPLICATION PART OF MAIN() TESTDRIVER                          */

    /* 1: Signon & check args */
    fputs(SIGNON, stdout);

    if (argc != 3) {
       fputs(USAGE, stdout);
       exit(EXIT_FAILURE);
    }
```

```
    /* 2: Open textfile to search */
    fdTxt = fopen(argv[2], "r");
    CHKERR(fdTxt == NULL, "Function fopen()")
    iRet = setvbuf(fdTxt, NULL, _IOFBF, 4*1024);
    CHKERR(iRet != 0, "Function setvbuf()")


    /* 3: Prepare pattern for TBM search */
    vBuildTBM( (BYTE*) argv[1], strlen(argv[1])); /* len. of pattern (EXCL \0) */


    /* 4: Perform TBM search on textfile, one line at a time, until EOF */
    while (1) {

       /* Read next line from file to buffer; - Break at EOF */
       if( fgets((char*)cBuf, MAXLIN, fdTxt) == NULL || feof(fdTxt) )
           break;
       dwCount++;

       /* Run TBM search on text in buffer;  - echo text if match */
            if ( iRunTBM(cBuf, strlen((char*)cBuf)) )
               printf("[%05lu]\t%s", dwCount, cBuf);
    }


    /* 5: Clean up & Terminate */
    free(fdTxt);
    return(EXIT_SUCCESS);
}
#endif      /* MAIN */
```

```
/*****************************************************************************/
/****************************** TBM ******************************************/
/*****************************************************************************/

/* Structure defining search pattern and TBM parameters */
PRIVATE struct {
    BYTE      pat[1024];       /* PATTERN Buffer holding pattern to match   */
    int       patlen;          /* ------- Sizeof(pattern) pat[] (incl. \0)  */
    int       delta[ASIZE];    /* SKIP..: BM skip table for pattern endchar */
    BYTE      rarec;           /* MATCH.: Guard, "rarest" char in pattern   */
    int       rareoff;         /* ------- Position in pattern of guard-char */
    int       md2;             /* SHIFT.: Mini-sd2 shift value              */
} pat;



/*+2 MODULE TBM.C ===============================================================*/
/*   NAME   01                  vBuildTBM                                       */
/*== SYNOPSIS ===================================================================*/
PUBLIC void
vBuildTBM(pb, len)
    BYTE     *pb;            /* Ptr begin of (\0-terminated) pattern string */
    int       len;           /* Length of pattern-string (#byte, EXCL \0)   */
{
/* DESCRIPTION
 *    Init. datastructures nessecary for running a Tuned Boyer-Moore search :
 *    fill in delta table for skip-loop, find rarest pattern-char (guard) for
 *    match-loop, and get "endcharacter reoccurence" (md2) for shift-function.
 *
 *    1: Initialize fields of struct "pat" from function arguments.
 *
 *    2: For SKIP-LOOP: Define the BM (Boyer-Moore) delta skip-table d[];
 *       d[] is the "delta" array precomputed for each char A in the alphabet
 *       to hold the #chars from the last char in the pattern to the rightmost
 *       occurence of A in the pattern (0 for the last char, patlen for all
 *       chars not in the pattern); -
 *       d[] is used in the skip loop to match the last pattern-char P with
 *       the corresponding text-char T : If ((k = d[T]) > 0), k holds the
 *       "delta-value" to shift the pattern right for aligning the last T
 *       in the pattern with T in the text (or for shifting the pattern past
 *       T in the text, when k = patlen).
 *
 *    3: For MATCH-LOOP: Define guard as the rarest character of the pattern;
 *       In the match-loop we will test the guard against the text before
 *       doing a full match test.
 *
 *    4: For SHIFT-FUNC.: Find "mini-sd2" shift (sd2: Sunday's general DELTA2);
 *       Define md2 as the #chars to shift the pattern to the first leftward
 *       reoccurence of the skip-loop char (here: the LAST char in pattern), -
 *       if the last char doesn't reoccur in the pattern, md2 = patternlength.
 *       This shift-value is applied to the "endpointer", whenever a mismatch
 *       occurs (cf. the exec function).
 *
 * RETURN
 *       Func.return value: None (void); - Resources are statically allocated
 *       for struct pat, so dynamic error conditions can not occur.
 *-2*/
```

```
    register BYTE *pe;          /* Ptr to pattern end                    */
    register BYTE *p;           /* Ptr for reverse scan of pattern       */
    register int j;             /* Index into delta array                */
    register int r;             /* Offset in pattern of "rare" character */
    register int *d;            /* Alias for delta array                 */
    BYTE     *bp;               /* Scan ptr. for pattern                 */


    /* 1: Initialize structure "pat" from the function arguments */
    pat.patlen = len;                       /* #chars in pat (EXCL \0)   */
    assert(pat.patlen < sizeof(pat.pat));   /* must fit into buffer      */
    memcpy(pat.pat, pb, pat.patlen);
    for (bp = pat.pat; bp < pat.pat+pat.patlen; bp++)
       *bp = UCASE(*bp);                    /* Possibly conv. to uppercase */
    D(fprintf(stdout, "Pattern[%s], Length[%d]\n", pat.pat, pat.patlen);)


    /* 2: For SKIP-LOOP: define the BM skip delta table d[] */
    for (j = 0, d = pat.delta; j < ASIZE; j++)
            d[j] = pat.patlen;                  /* Init all = pat.patlen */

    for (pb = pat.pat, pe = pb+pat.patlen-1; pb <= pe; pb++) {
            d[*pb] = pe - pb;                   /* Reset for pat.chars   */
        D(fprintf(stdout, "\tSkip :\t[%c]=%d\n", *pb, pe - pb);)
    }


    #ifdef TBM_FQ
    /* 3: For MATCH-LOOP: define guard as rarest char of pattern (cf. freq[]) */
    r = 0;                                  /* rare pos. (#chr from pat.start) */
    for (pb = pat.pat, pe = pb+pat.patlen-1; pb < pe; pb++)
            if (freq[*pb] < freq[pat.pat[r]])
                r = pb - pat.pat;
    pat.rarec = pat.pat[r];                 /* guard char */
    pat.rareoff = r - (pat.patlen - 1);     /* negative guard pos (from end) */
    D(fprintf(stdout, "\tMatch:\tGuard[%c], Offset[%d]\n", pat.rarec, pat.rareoff);)
    #endif /*TBM_FQ*/


    /* 4: For SHIFT-FUNC.: define md2 = shift to left reoccur. of term. char */
    for (pe = pat.pat+pat.patlen-1, p = pe-1; p >= pat.pat; p--)
            if (*p == *pe)
                break;
    /* now *p is first leftward reoccurrence in pattern of the term. char: *pe */
    pat.md2 = pe - p;
    D(fprintf(stdout, "\tShift:\tMD2[%d]\n", pat.md2);)

} /* END function vBuildTBM() */
```

```
/*+2 MODULE TBM.C ============================================================*/
/*   NAME   02                  iRunTBM                                        */
/*== SYNOPSIS ================================================================*/
PUBLIC int
iRunTBM(base, n)
    BYTE     *base;             /* Ptr to base of textblock for TBM search */
    int       n;                /* Length of textblock base[]              */
{
/* DESCRIPTION
 *    Perform a TBM search, applying pattern in struct pat to textbuffer base.
 *    Requires basic info. on search pattern previously set up by iBuildTBM();
 *    Now iRunTBM() may be called repeatedly to search text-buffers for pat.:
 *
 *    1: Initialize variables
 *       1.1: Set up shorthand register var's for struct. "pat" fields.
 *       1.2: Initialize pointers to text and pattern
 *       1.3: Initialize scratch variables
 *       1.4: Catch boundary condition (Immediate ret on empty pattern/text)
 *
 *    2: Insert sentinel string after the text block : put patlen copies of
 *       the terminal pattern char pat[patlen-1] after text[textlen]; -
 *       This trick removes a test of End-Of-Text (s < e) from the skip loop.
 *
 *    3: SEARCH LOOP ... :
 *       3.1 SKIP LOOP
 *           d0[] is the "delta array" index'ed by the current text-char T at
 *           the position of the last pattern-char P to give the #chars for
 *           shifting the pattern right in case of T/P-mismatch, thus aligning
 *           the last occurence of T-in-the-pattern with T-in-the-text.
 *           A shift value of 0 indicates T/P-match, thus acting as a sentinel
 *           breaking the skip loop.
 *
 *       3.2 MATCH LOOP
 *           First test Guard (rarest char in pattern) against text
 *           If guard match : do a complete frwd scan of pattern against text
 *           If total match : increment match-count & continue with shift
 *                            or break and just return [T]
 *
 *       3.3 SHIFT FUNCTION
 *           Shift the endpointer "s" by md2 chars to the right : s always
 *           points to the position in the text for matching the LAST char
 *           of the pattern; - thus by shifting s to the right, we "drag"
 *           along the pattern to the new position for the next match attempt.
 *
 * RETURN
 *    Side effects .....: Textbuffer partly thrashed by terminal sentinel.
 *                        NB: buffer must have room for MAXPATTERN after text.
 *    Func.return value :  0  if no match of pattern in text,
 *                        >0  if match, - value depending on #define TBM_TF :
 *                            1  if TBM_TB   defined (ie. return [True|False] )
 *                            c  if TBM_TB undefined (where c=count of matches)
 *-2*/

    /* 1.1: Set up shorthand register var's for struct. "pat" fields     */
    register int n1  = pat.patlen - 1; /* #chars in pattern (excl. \0)   */
    register int *d0 = pat.delta;      /* addr. of delta shift array     */
    register int md2 = pat.md2;        /* value of mini-sd2 shift        */
    register BYTE rc = pat.rarec;      /* rarest char of pattern         */
    register int ro  = pat.rareoff;    /* pos. from end-of-pat of rc     */

    /* 1.2: Initialize pointers to text and pattern  */
    register BYTE *s = base + n1;      /* startchar in text for match    */
```

```
register BYTE *e = base + n;        /* endchar of text for match */
register BYTE *ep = pat.pat + n1;   /* startchar in pat for match */

/* 1.3: Initialize scratch variables */
register BYTE *p, *q;
register int k;
register int nmatch = 0;


/* 1.4: Return "no match" on boundary condition (empty pattern/text) */
if (pat.patlen <= 0 || n <= 0) {
   printf("Empty %s\a\n\n", (pat.patlen == 0 ? "pattern" : "text"));
   return(0);
}


/* 2: Set up sentinel for skip loop (thus removing test for End-Of-Text) */
memset(e, pat.pat[pat.patlen - 1], pat.patlen);


/* 3: Enter SEARCH LOOP ... */
while (s < e) {

   /* ------------------------------------------------------------------ */
   /* 3.1: SKIP LOOP : ufast                                             */

   /* Skip on the last character in pattern, using delta shift table d0 */
        k = d0[UCASE(*s)];            /* get shift to rightmost *s in pattern    */
        while (k) {                   /* while k > 0, ie no match (pos. shift)   */
            k = d0[UCASE(*(s += k))]; /* use 3-fold loop unrolling for max. speed */
            k = d0[UCASE(*(s += k))]; /* if match, k=0 (so unrolling is harmless) */
            k = d0[UCASE(*(s += k))];
        }
        if (s >= e)                   /* sentinel assures match at End-Of-Text   */
            break;


   /* ------------------------------------------------------------------ */
   /* 3.2: MATCH LOOP : guard test & fwd scan                            */

   #ifdef TBM_FQ
        /* First test Guard (rarest char in pattern) against text */
        if (UCASE(s[ro]) != rc)        /* obs: ro is a negative offset */
            goto mismatch;
   #endif /*TBM_FQ*/

        /* Guard-match : do a complete forward scan of pattern against text */
        for (p = pat.pat, q = s - n1; p < ep;) {
            if (UCASE(*q++) != *p++)
                        goto mismatch;
        }

        /* Total-match : increment match-count or break w. answer [T|F] */
        nmatch++;

   #ifdef TBM_TF
        break;              /* break at first match */
   #endif /*TBM_TF*/
```

```
mismatch:
       /* ------------------------------------------------------------------ */
       /* 3.3: SHIFT : md2                                                   */

       /* Mismatch (or complete match); - shift "endptr" & continue search  */
       s += md2;

    } /* End search-loop : while (s<e) */


    /* 4: Return the verdict : #match (or [1|0] if just [T|F] requested)   */
    *e = '\0';         /* Restore string termination after skip sentinel   */
    return (nmatch);

} /* END function iRunTBM() */



/*+3 MODULE TBM.C ----------------------------------------------------------*/
/*   NAME   03                    bDKupper                                   */
/*-- SYNOPSIS ----------------------------------------------------------------*/
PRIVATE BYTE
bDKupper(ii)                         /* Input char to convert to uppercase */
   int ii;                         /* NB: char as param. promotes to int */
{
/* DESCRIPTION
 *   Convert character <(BYTE)ii> from lower- to uppercase, INCL DANISH CHARS.
 * RETURN
 *   IF (std. routine toupper(c) DOES return a converted char)
 *      return(std. converted char)
 *   ELSE IF (DKconvert DOES return a converted char)
 *      return(DK. converted char)
 *   ELSE
 *      return(unconverted char)
 *-3*/

   static const BYTE bDKlow[] = {(BYTE)'`',(BYTE)'›',(BYTE)'†'};
   static const BYTE bDKupp[] = {(BYTE)'’',(BYTE)'□',(BYTE)'□'};
   register BYTE  bi = (BYTE) ii;    /* Input char from int param   */
   BYTE  bo;                         /* Output char from conversion */
   BYTE *bp;                         /* Character pointer into cDKx */


   /* Perform conversion and return char */
   return( ((bo = (BYTE) toupper(ii)) != bi) ? bo :
              ((bp = (BYTE*) strchr((char*)bDKlow, bi)) != NULL) ? bDKupp[bp - bDKlow] :
                 bi );

} /* END function bDKupper */



/* End of module TBM.C                                                      */
/*==========================================================================*/
```