1 Knuth-Morris-Pratt algorithm (入力待ちアボート機能つき)

```
#include <stdio.h>
#include<fcntl.h>
#include<sys/time.h>
#include<sys/types.h>
#include<unistd.h>
#define XSIZE 1000
#define N 1000
#define M 100

int isready(int fd, int timeout)
{
  int rc;
  fd_set fds;
  struct timeval tv;

  FD_ZERO(&fds);
  FD_SET(fd,&fds);
  tv.tv_sec = timeout;
  tv.tv_usec = 0;

  rc = select(fd+1, &fds, NULL, NULL, &tv);
  if (rc < 0)
    return -1;

  return FD_ISSET(fd,&fds) ? 1 : 0;
}

main()
{
  char x[N], y[N];
  char text1[100] = "A STRING SEARCHING EXAMPLE CONSISTING OF SIMPLE TEXT",
       patn1[50] = "STRING";
  char text2[100] = "GCATCGCAGAGAGTATACAGTACG",
       patn2[50] = "GCAGAGAG";
  int m, n;
  int fd, s;

  fd = STDIN_FILENO;

  printf("\nPlease input text: ");

  /* scanf("%s", &y) ; */
  s = isready(fd, 5);  /* stdin の fileno */
  if (s < 0) {
     exit(1); /* EOF etc ... */
  } else if (0 < s) {  /* normal input ready */
     scanf("%s", &y);
  } else /* s == 0 */ {
     strcpy(y, text2);
     printf("%s", y);
  }

  n = strlen(y);

  printf("\nPlease input pattern: ");

  /* scanf("%s", &x) ; */
  s = isready(fd, 5);  /* stdin の fileno */
  if (s < 0) {
     exit(1); /* EOF etc ... */
  } else if (0 < s) {  /* normal input ready */
     scanf("%s", &x);
  } else /* s == 0 */ {
     strcpy(x, patn2);
     printf("%s", x);
  }

  m = strlen(x);
```

```
  printf("\n\nText is %s, %d, and pattern is %s, %d\n\n", y, n, x, m);

  KMP(x, m, y, n);

}

void KMP(char *x, int m, char *y, int n) {
  int i, j, kmpNext[1000];
  int c_cmp, trial, c_old;

  /* preprocessing */
  preKMP(x, m, kmpNext);

  c_cmp = 0;
  trial = 0;

  /* Searching */
  i = j = 0;
  while (j <= n-m) {
    while (i > -1 && c_cmp++ && x[i] != y[j]) {
      printf("    mismatch: y[%d]=%c, x[%d]=%c\n", j, y[j], i, x[i]);
      trial++;
      printf("%d 試行, %d 文字比較回数\n\n", trial, c_cmp-c_old);
      c_old = c_cmp;
      i = kmpNext[i];
    }
    i++;
    j++;
    if (i >= m) {
      trial++;
      if (x[i-1] == y[j-1])
         printf("   Matched at y[%d]=%c and x[%d]=%c!!!\n%d attempt, %d 文字比較
回数\n", j-1, y[j-i], i-1, x[i-1], trial, c_cmp-c_old);
      else
        printf("    mismatch: y[%d]=%c, x[%d]=%c\n", j-1, y[j-1], i-1, x[i-1]);
      c_old = c_cmp;
      i = kmpNext[i];
    }
  }
 printf("\n合計 %d 試行, %d 回文字比較\n", trial, c_cmp);
}

void preKMP(char *x, int m, int kmpNext[]) {
  int i, j;

  i = 0;
  j = kmpNext[0] = -1;
  while (i < m) {
    while (j > -1 && x[i] != x[j])
      j = kmpNext[j];
    i++;
    j++;
    if (x[i] == x[j])
      kmpNext[i] = kmpNext[j];
    else
      kmpNext[i] = j;
  }
 printf("\nkmpNext Table\n\n ");
 for(i=0;  i<=m; i++) printf("%d: %c %d, ", i, x[i], kmpNext[i]);
 printf("\n\n");
}
```

2 Boyer-Moore Algorithm

```
#include <math.h>
#include <stdio.h>
#define XSIZE 1000
#define ASIZE 128
#define N 1000
#define M 100
```

```
#define MAX(a,b) ((a)> (b)) ?  (a): (b)

main()
{
  char x[N], y[N];
  int m, n;

  printf("Please input text: ");
  scanf("%s", &y) ;
  n = strlen(y);

  printf("Please input pattern: ");
  scanf("%s", &x) ;
  m = strlen(x);

  printf("\n %s, %d, %s, %d\n", y, n, x, m);

  BM(x, m, y, n);

}

void BM(char *x, int m, char *y, int n) {
  int i, j, bmGs[XSIZE], bmBc[ASIZE];
  int c_cmp, trial, c_old;

  /* preprocessing */
  preBmGs(x, m, bmGs);
  preBmBc(x, m, bmBc);

  c_cmp = 0;
  trial = 0;

  /* Searching */
  j = 0;
  while (j <= n - m) {
    for (i = m - 1; i >= 0 && c_cmp++ && x[i] == y[i + j]; --i);
    trial++;
    if (i < 0) {
      printf("   Matched at y[%d]=%c and x[%d]=%c!!!\n%d attempt, %d char comparisons\n\n"

      j += bmGs[0];
    }
    else {
      printf("    mismatch: y[%d]=%c, x[%d]=%c\n", i+j, y[i+j], i, x[i]);
      printf("%d attempt, %d char comparisons\n\n", trial, c_cmp-c_old);
      j += MAX(bmGs[i], bmBc[y[i + j]] - m + 1 + i);
    }
    c_old = c_cmp;
  }
 printf("\n合計 %d 試行, %d 回文字比較\n", trial, c_cmp);
}

void preBmBc(char *x, int m, int bmBc[]){
  int i;

  for (i = 0; i < ASIZE; ++i)
    bmBc[i] = m;
  for (i = 0; i < m - 1; ++i)
    bmBc[x[i]] = m - i - 1;
}

void preBmGs(char *x, int m, int bmGs[]) {
  int i, j, suff[XSIZE];

  suffixes(x, m, suff);

  for (i = 0; i < m; ++i)
    bmGs[i] = m;
  j = 0;
  for (i = m - 1; i >= -1; --i)
    if (i == -1 || suff[i] == i + 1)
      for (; j < m - 1 - i; ++j)
```

```
        if (bmGs[j] == m)
          bmGs[j] = m - 1 - i;
  for (i = 0; i <= m - 2; ++i)
    bmGs[m - 1 - suff[i]] = m - 1 - i;
}

void suffixes(char *x, int m, int *suff) {
  int f, g, i;

  suff[m - 1] = m;
  g = m - 1;
  for (i = m - 2; i >= 0; --i) {
    if (i > g && suff[i + m - 1 - f] < i - g)
      suff[i] = suff[i + m - 1 - f];
    else {
      if (i < g)
        g = i;
      f = i;
      while (g >= 0 && x[g] == x[g + m - 1 - f])
        --g;
      suff[i] = f - g;
    }
  }
}
```

3 実行例

```
text: GCATCGCAGAGAGTATACAGTACG
pattern: GCAGAGAG
```

c	A	C	G	T
bmBC[c]	1	6	2	8

i	0	1	2	3	4	5	6	7
x[i]	G	C	A	G	A	G	A	G
suff[i]	1	0	0	2	0	4	0	8
bmGs[i]	7	7	7	2	7	4	7	1

```
    mismatch: y[7]=A, x[7]=G
1 試行, 1 回文字比較

    mismatch: y[6]=C, x[5]=G
2 試行, 3 回文字比較

   Matched at y[5]=G and x[0]=G!!!
3 試行, 8 回文字比較

    mismatch: y[17]=C, x[5]=G
4 試行, 3 回文字比較

    mismatch: y[22]=C, x[6]=A
5 試行, 2 回文字比較
```

合計 5 試行, 17 回 文字比較

i	0	1	2	3	4	5	6	7	8
x[i]	G	C	A	G	A	G	A	G	
kmpNext[i]	-1	0	0	-1	1	-1	1	-1	1

```
    mismatch: y[3]=T, x[3]=G
1 試行, 4 回文字比較

    mismatch: y[4]=C, x[0]=G
2 試行, 1 回文字比較

   Matched at y[12]=G and x[7]=G!!!
3 試行, 8 回文字比較

    mismatch: y[13]=T, x[1]=C
4 試行, 1 回文字比較

    mismatch: y[13]=T, x[0]=G
5 試行, 1 回文字比較

    mismatch: y[14]=A, x[0]=G
6 試行, 1 回文字比較

    mismatch: y[15]=T, x[0]=G
7 試行, 1 回文字比較

    mismatch: y[16]=A, x[0]=G
8 試行, 1 回文字比較
```

合計 8 試行, 18 回文字比較

4 必修課題 — 8/1 締切

必修 **1 冊の本の中に出現する単語の頻度 (1-gram, 2-gram, 3-gram) を求めよ。本は Project Gutenberg (http://promo.net/pg/) から選ぶこと。ただし、他の人と違う本を使用すること。また、ダウンロードしたテキストには通常最初の部分に Project の説明があるので、その部分は落すこと。 (8/1 締切)**

随意 1 **配列の要素に 1 回目にアクセスされた時に初期値を始めて初期値を入れる方法を考えよ。 2 回目以降のアクセスも、通常の配列のアクセスと比べると遅くなってもよいが、初期化を遅延したい。 (8/15 締切, 遅れても良い)**

随意 2 **自分が選んだアルゴリズムの可視化を行え。 (8/15 締切, 遅れても良い)**