Algorithms

Instructor –
Dimitrios Katsaros

Lecture on String Matching

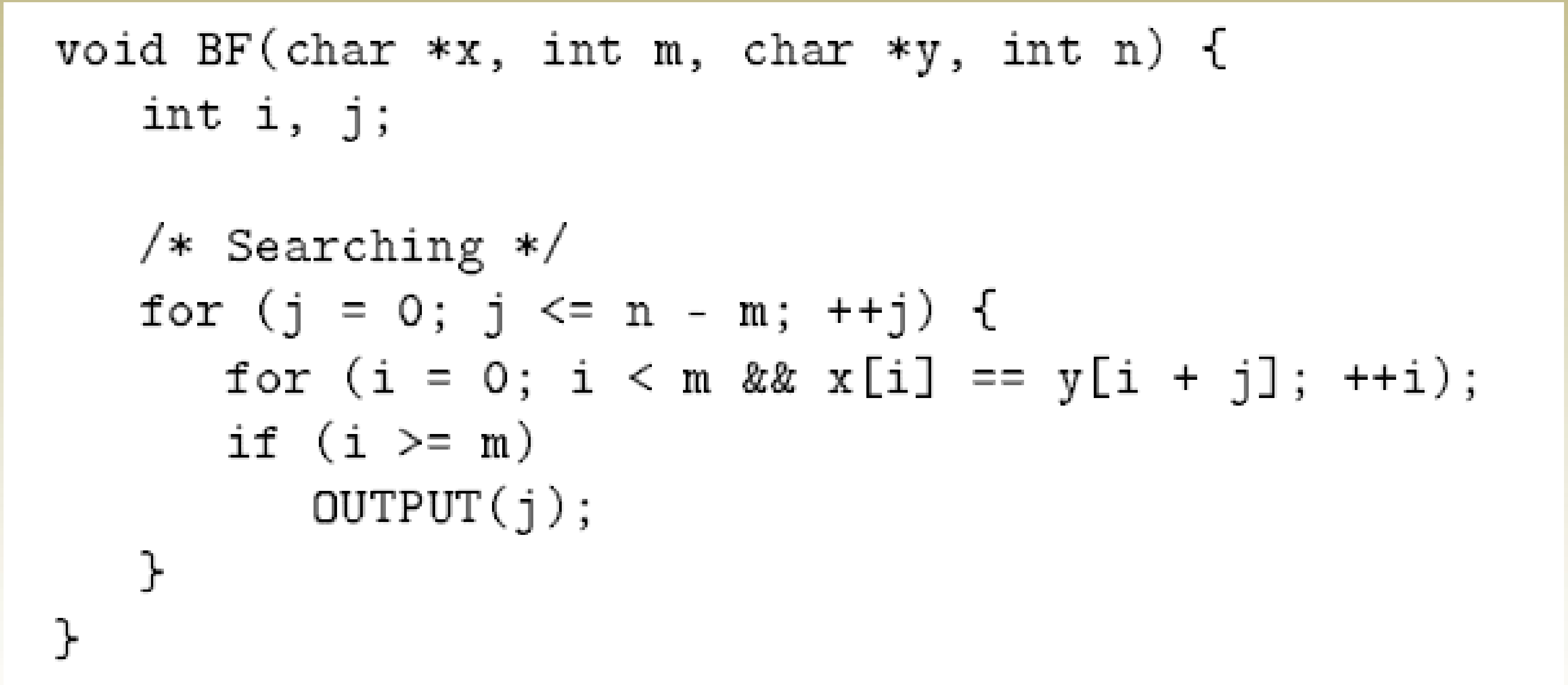
String Matching: Brute Force

- Text y of length: n
- Pattern x of length: m
- Find all occurrences of x in y

- The brute force algorithm consists in checking at all positions in the text between 0 and n-m
- After each attempt it shifts the pattern by exactly one position to the right

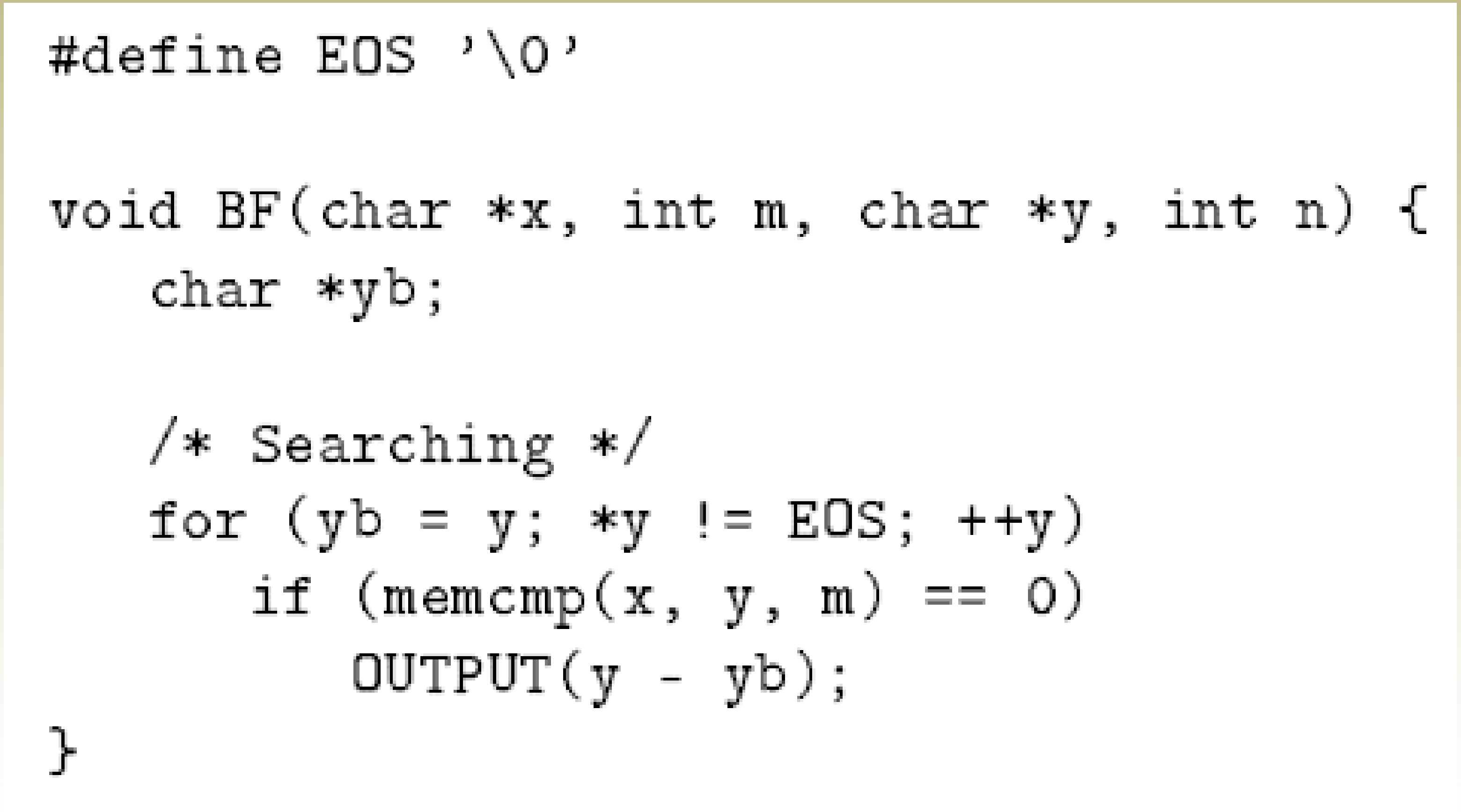
- Time complexity: O(m x n)
- Expected number of character comparisons: 2n

String Matching: Brute Force code-1



```
void BF(char *x, int m, char *y, int n) {
   int i, j;

   /* Searching */
   for (j = 0; j <= n - m; ++j) {
      for (i = 0; i < m && x[i] == y[i + j]; ++i);
      if (i >= m)
         OUTPUT(j);
   }
}
```

String Matching: Brute Force code-2

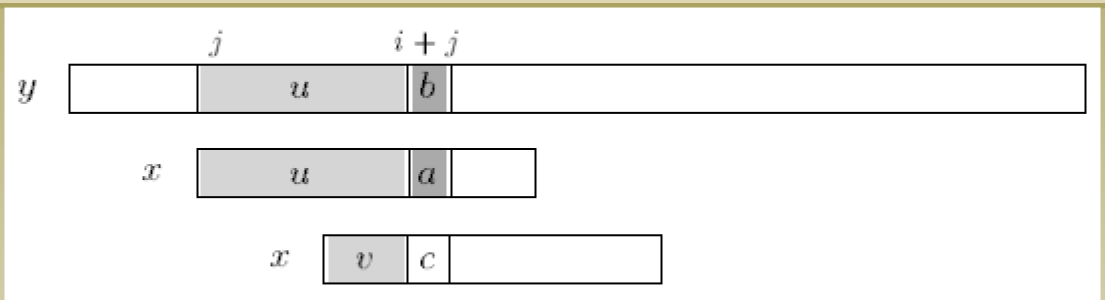


```
#define EOS ’\0’

void BF(char *x, int m, char *y, int n) {
   char *yb;

   /* Searching */
   for (yb = y; *y != EOS; ++y)
      if (memcmp(x, y, m) == 0)
         OUTPUT(y - yb);
}
```

String Matching: Morris-Pratt

- It is possible to improve the length of the shifts and simultaneously remember some portions of text that match the pattern
- Consider an attempt at a left position j on y, that is when the window is positioned on the text factor $y[j \ldots j+m-1]$
- Assume that the first mismatch occurs between $x[i]$ and $y[i+j]$ with $0 < i < m$
- Then, $x[0 \ldots i-1] = y[j \ldots i+j-1] = u$ and $a = x[i] \neq y[j+1]=b$
- When shifting it is reasonable to expect that a prefix v of the pattern matches some suffix of the portion u of text

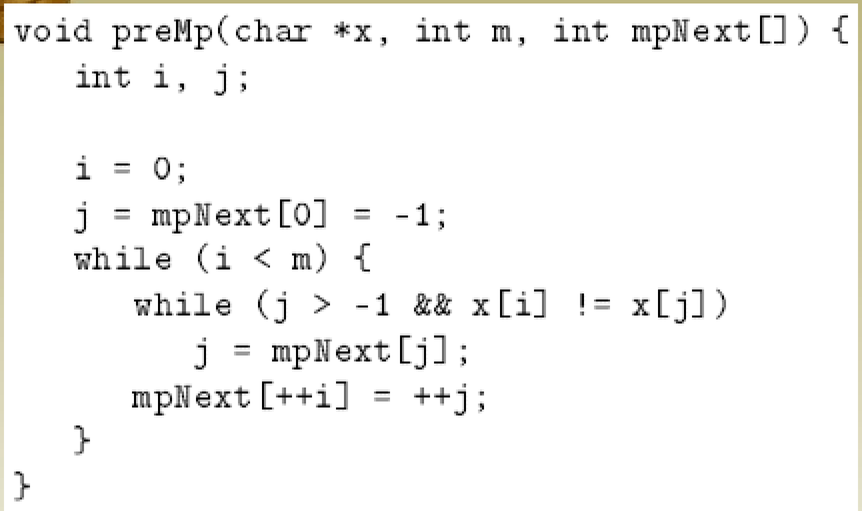
String Matching: Morris-Pratt



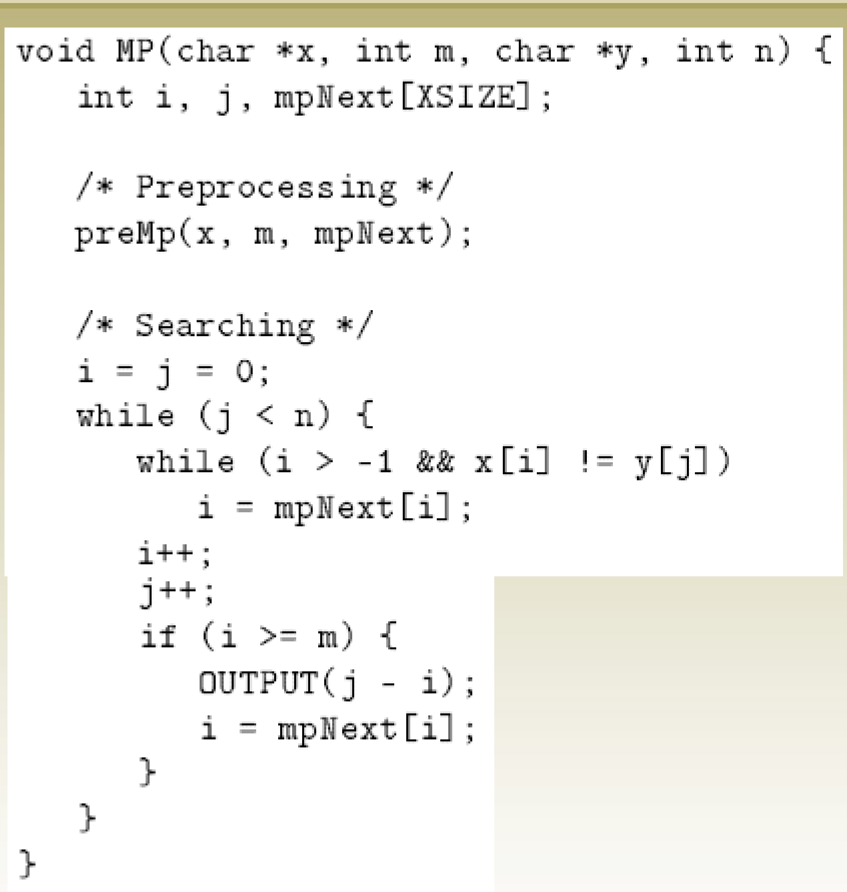
- The longest such prefix v is called the border of u (it occurs at both ends of u)
- This introduces the notation:

  let mpNext[i] be the length of the longest border of x[0…i-1] for 0 < i ≤ c = x[mpNext[i]] and y[i+j] = b
- mpNext[0] = -1
- The table mpNext can be computed in O(m) space and time before searching
- Time complexity: O(m x n)
- At most number of character comparisons: 2n - 1

String Matching: Morris-Pratt code

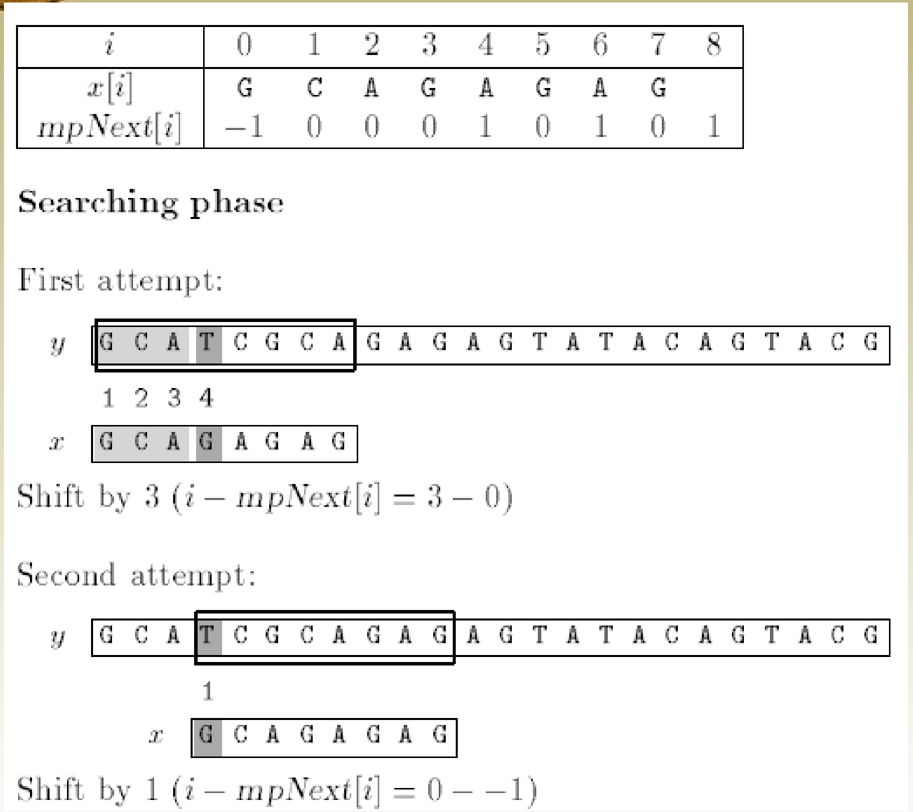


```
void preMp(char *x, int m, int mpNext[]) {
   int i, j;

   i = 0;
   j = mpNext[0] = -1;
   while (i < m) {
      while (j > -1 && x[i] != x[j])
         j = mpNext[j];
      mpNext[++i] = ++j;
   }
}
```



```
void MP(char *x, int m, char *y, int n) {
   int i, j, mpNext[XSIZE];

   /* Preprocessing */
   preMp(x, m, mpNext);

   /* Searching */
   i = j = 0;
   while (j < n) {
      while (i > -1 && x[i] != y[j])
         i = mpNext[i];
      i++;
      j++;
      if (i >= m) {
         OUTPUT(j - i);
         i = mpNext[i];
      }
   }
}
```

String Matching: Morris-Pratt execution



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

Searching phase

First attempt:

y G C A T C G C A G A G A G T A T A C A G T A C G

1 2 3 4

x G C A G A G A G

Shift by 3 ($i - mpNext[i] = 3 - 0$)

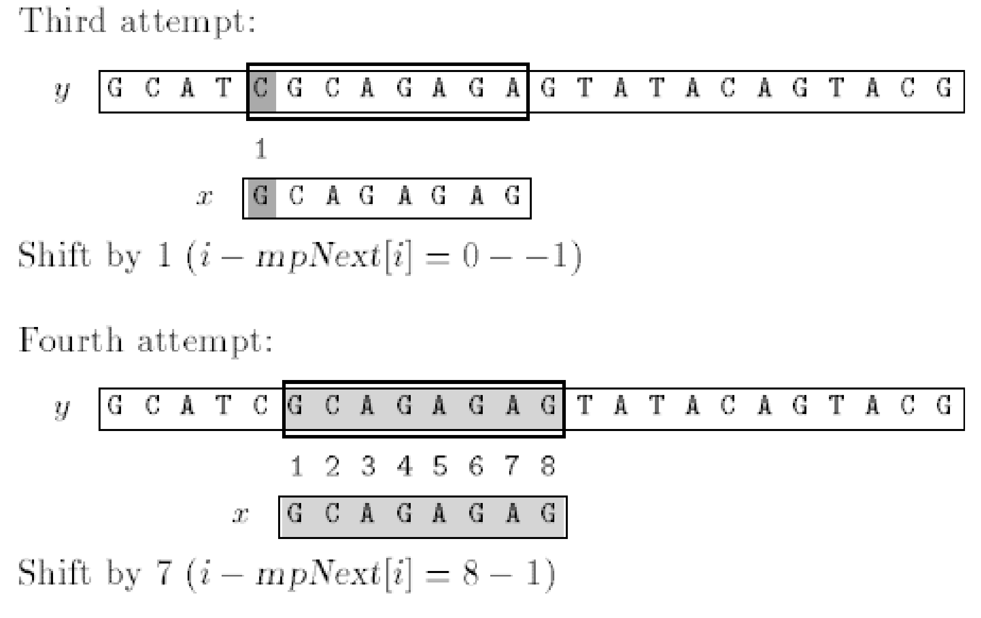
Second attempt:

y G C A T C G C A G A G A G T A T A C A G T A C G

1

x G C A G A G A G

Shift by 1 ($i - mpNext[i] = 0 - -1$)



Third attempt:

y G C A T C G C A G A G A G T A T A C A G T A C G

1

x G C A G A G A G

Shift by 1 ($i - mpNext[i] = 0 - -1$)

Fourth attempt:

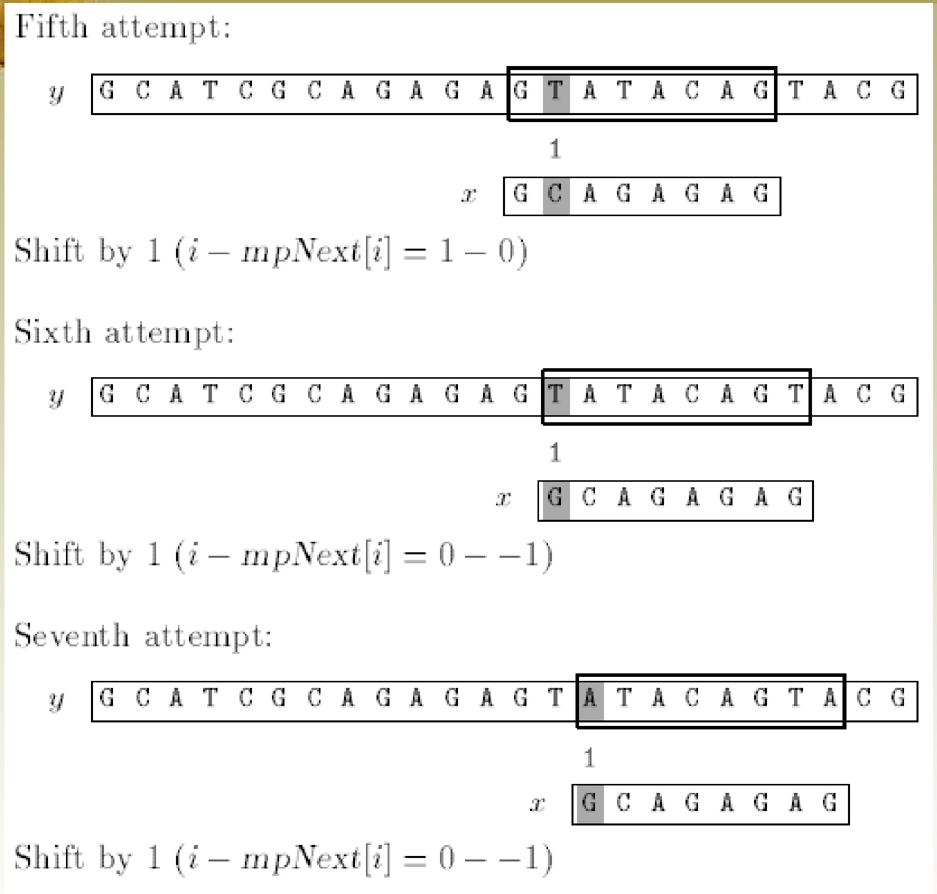
y G C A T C G C A G A G A G T A T A C A G T A C G

1 2 3 4 5 6 7 8

x G C A G A G A G

Shift by 7 ($i - mpNext[i] = 8 - 1$)

String Matching: Morris-Pratt execution



Fifth attempt:

y G C A T C G C A G A G A G T A T A C A G T A C G

1

x G C A G A G A G

Shift by 1 ($i - mpNext[i] = 1 - 0$)

Sixth attempt:

y G C A T C G C A G A G A G T A T A C A G T A C G

1

x G C A G A G A G

Shift by 1 ($i - mpNext[i] = 0 - -1$)

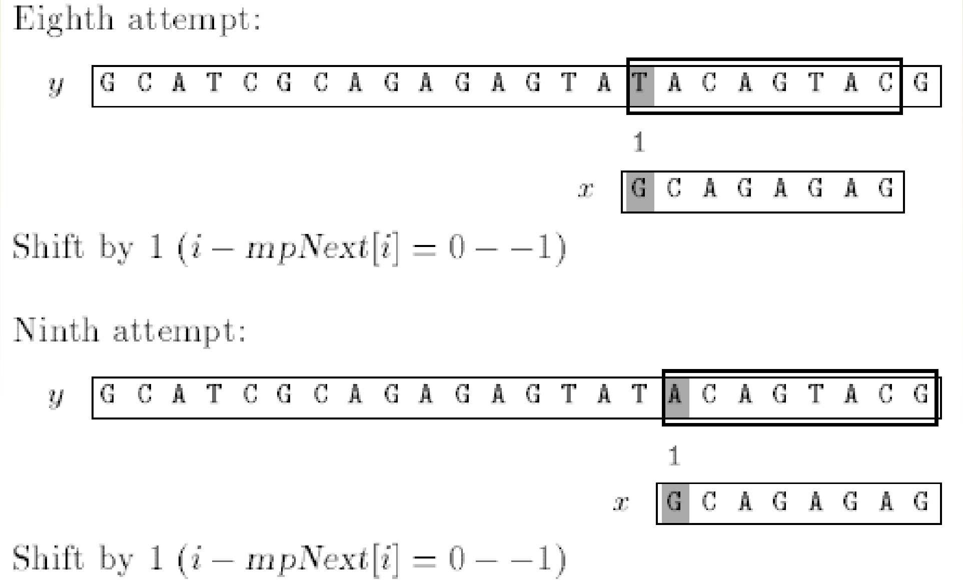
Seventh attempt:

y G C A T C G C A G A G A G T A T A C A G T A C G

1

x G C A G A G A G

Shift by 1 ($i - mpNext[i] = 0 - -1$)



Eighth attempt:

y G C A T C G C A G A G A G T A T A C A G T A C G

1

x G C A G A G A G

Shift by 1 ($i - mpNext[i] = 0 - -1$)

Ninth attempt:

y G C A T C G C A G A G A G T A T A C A G T A C G

1

x G C A G A G A G

Shift by 1 ($i - mpNext[i] = 0 - -1$)

String Matching: Knuth-Morris-Pratt

- Consider an attempt at a left position j, i.e., the window is positioned on the text factor $y[j \ldots j+m-1]$
- Assume that the first mismatch occurs between $x[i]$ and $y[i+j]$ with $0 < i < m$
- Then, $x[0 \ldots i-1] = y[j \ldots i+j-1] = u$ and $a = x[i] \neq y[j+1]=b$
- When shifting it is reasonable to expect that a prefix v of the pattern matches some suffix of the portion u of text

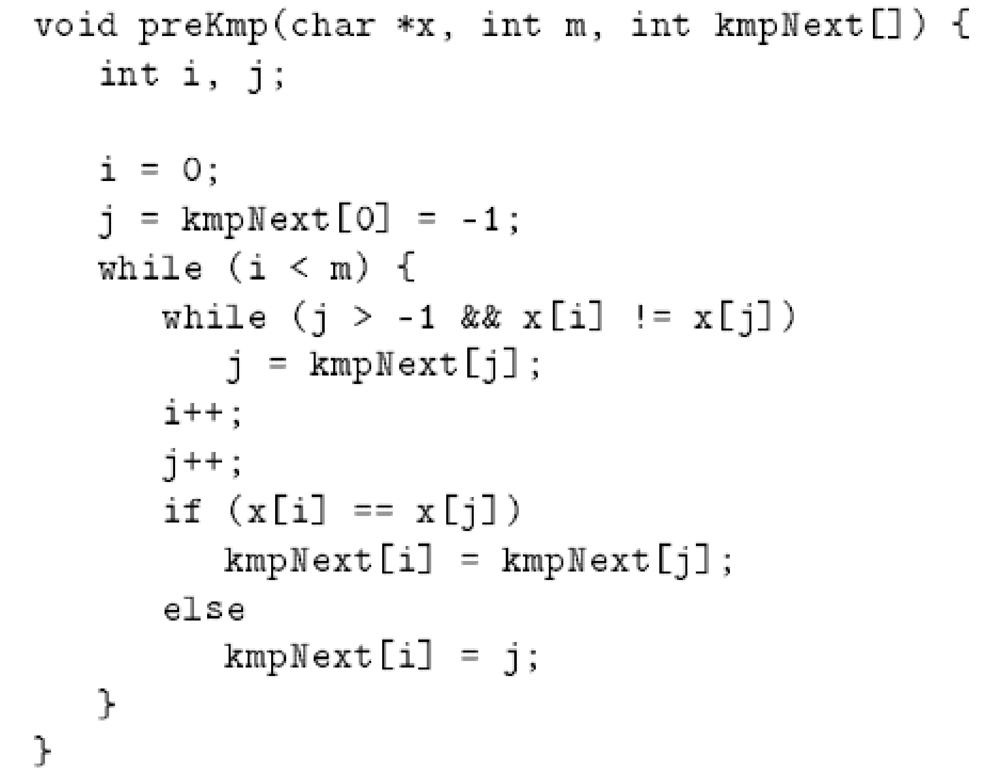
- Moreover, if we want to avoid another immediate mismatch, the character following the prefix v in the pattern must be different from a
- The longest such prefix v is called the tagged (or strong) border of u (it occurs at both ends of u followed by different characters in x)

String Matching: Knuth-Morris-Pratt

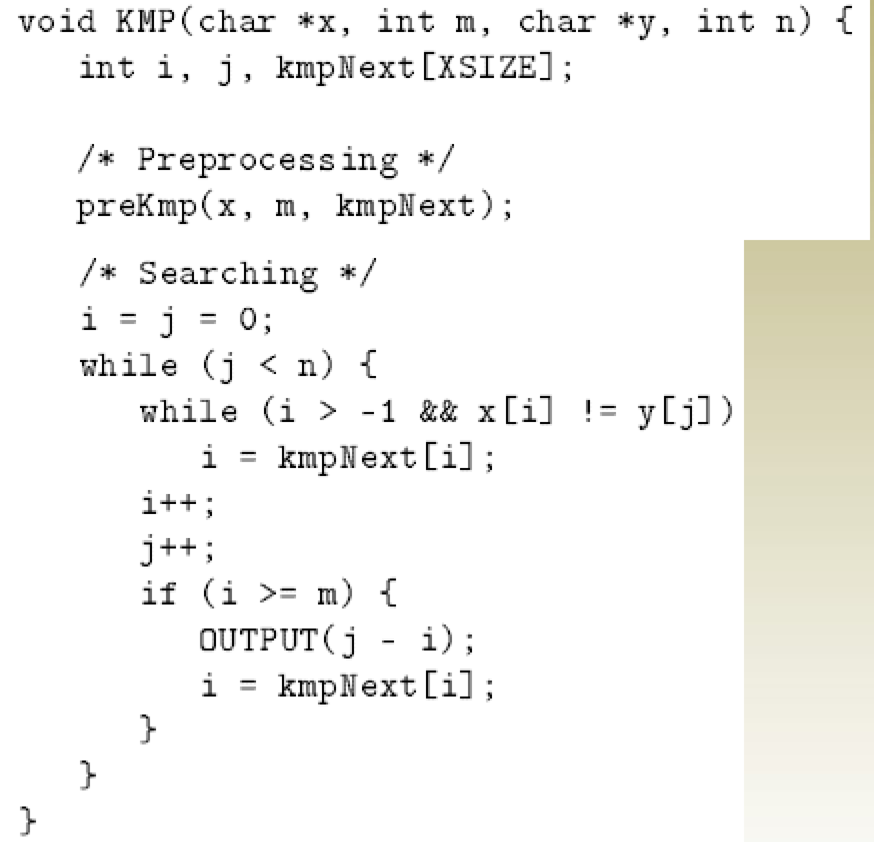
- This introduces the notation:

  Let kmpNext[i] be the length of the longest border of x[0…i-1] followed by a character c different from x[i] and -1 if no such tagged border exists, $0 < i \leq m$
- Then after a shift the comparisons can resume between characters = x[kmpNext[i]] and y[i+j] without missing any occurrences of x in y and avoiding a backtrack on the text
- The table kmpNext can be computed in O(m) space and time before searching
- Time complexity: O(m x n)
- At most number of character comparisons: 2n - 1

String Matching: Knuth-Morris-Pratt code

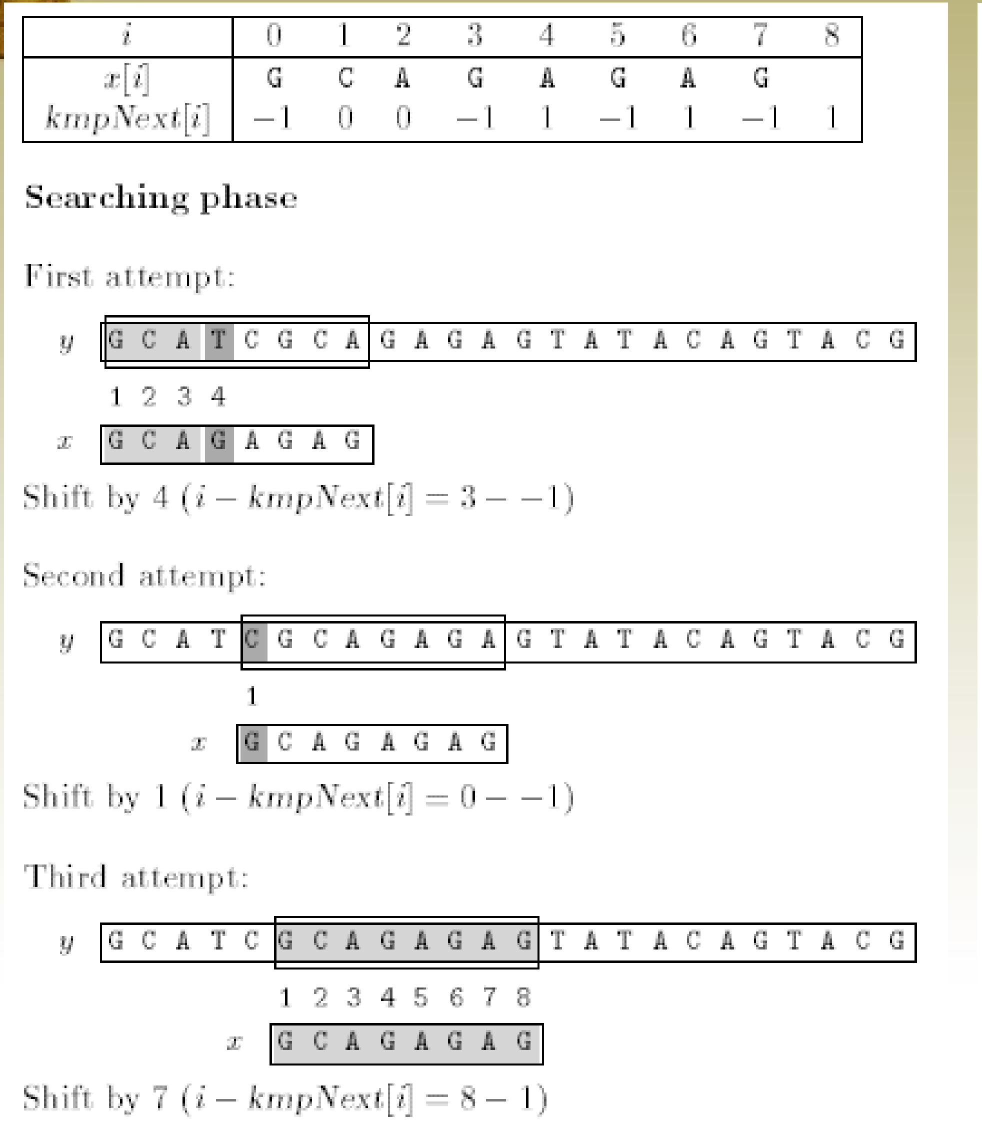


```
void preKmp(char *x, int m, int kmpNext[]) {
   int i, j;

   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;
   }
}
```



```
void KMP(char *x, int m, char *y, int n) {
   int i, j, kmpNext[XSIZE];

   /* Preprocessing */
   preKmp(x, m, kmpNext);

   /* Searching */
   i = j = 0;
   while (j < n) {
      while (i > -1 && x[i] != y[j])
         i = kmpNext[i];
      i++;
      j++;
      if (i >= m) {
         OUTPUT(j - i);
         i = kmpNext[i];
      }
   }
}
```

String Matching: Knuth-Morris-Pratt exec



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

Searching phase

First attempt:

y G C A T C G C A G A G A G T A T A C A G T A C G

1 2 3 4

x G C A G A G A G

Shift by 4 ($i - kmpNext[i] = 3 - -1$)

Second attempt:

y G C A T C G C A G A G A G T A T A C A G T A C G

1

x G C A G A G A G

Shift by 1 ($i - kmpNext[i] = 0 - -1$)

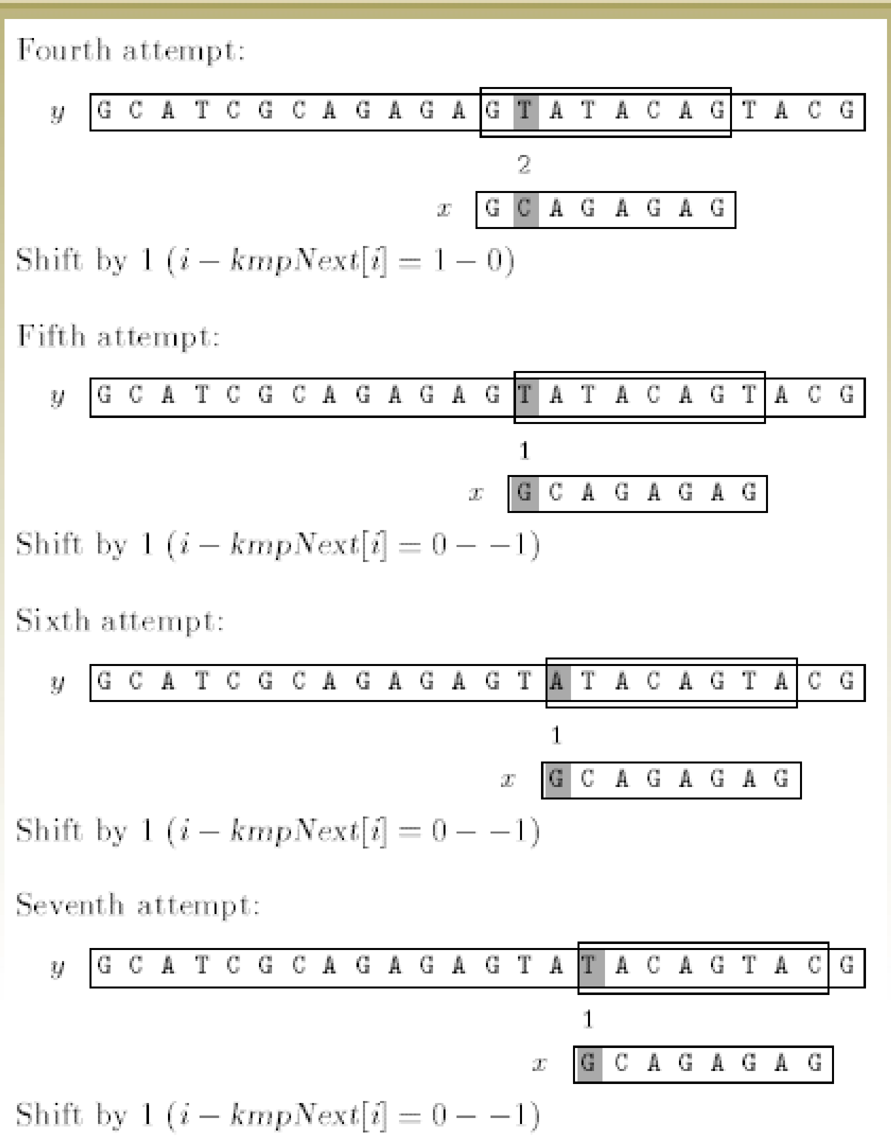
Third attempt:

y G C A T C G C A G A G A G T A T A C A G T A C G

1 2 3 4 5 6 7 8

x G C A G A G A G

Shift by 7 ($i - kmpNext[i] = 8 - 1$)



Fourth attempt:

y G C A T C G C A G A G A G T A T A C A G T A C G

2

x G C A G A G A G

Shift by 1 ($i - kmpNext[i] = 1 - 0$)

Fifth attempt:

y G C A T C G C A G A G A G T A T A C A G T A C G

1

x G C A G A G A G

Shift by 1 ($i - kmpNext[i] = 0 - -1$)

Sixth attempt:

y G C A T C G C A G A G A G T A T A C A G T A C G

1

x G C A G A G A G

Shift by 1 ($i - kmpNext[i] = 0 - -1$)

Seventh attempt:

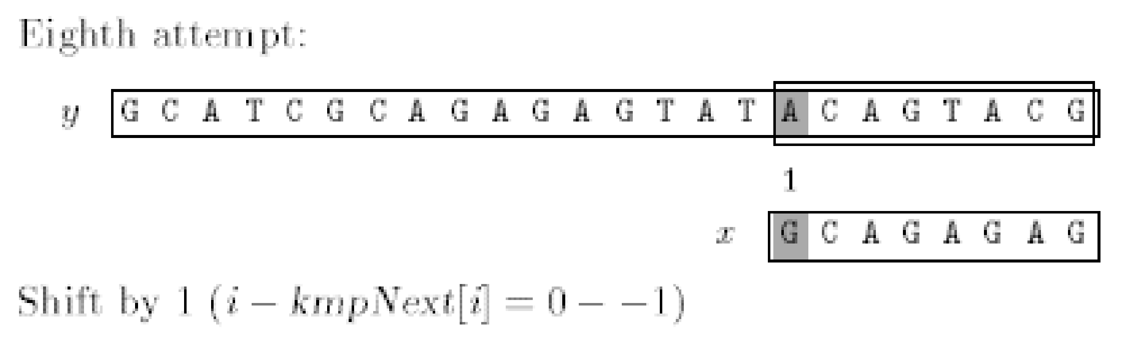
y G C A T C G C A G A G A G T A T A C A G T A C G

1

x G C A G A G A G

Shift by 1 ($i - kmpNext[i] = 0 - -1$)

String Matching: Knuth-Morris-Pratt exec



Eighth attempt:

y G C A T C G C A G A G A G T A T A C A G T A C G

1

x G C A G A G A G

Shift by 1 ($i - kmpNext[i] = 0 - -1$)

- KMP performed 18 character comparisons
- MP performed 19 character comparisons