Knuth-Morris-Pratt Algorithm

The problem of String Matching

Given a string 'S', the problem of string matching deals with finding whether a pattern 'p' occurs in 'S' and if 'p' does occur then returning position in 'S' where 'p' occurs.

The Knuth-Morris-Pratt Algorithm

Knuth, Morris and Pratt proposed a linear time algorithm for the string matching problem.

A matching time of O(n) is achieved by avoiding comparisons with elements of 'S' that have previously been involved in comparison with some element of the pattern 'p' to be matched. i.e., backtracking on the string 'S' never occurs

Components of KMP algorithm

The prefix function, Π

The prefix function, IT for a pattern encapsulates knowledge about how the pattern matches against shifts of itself. This information can be used to avoid useless shifts of the pattern 'p'. In other words, this enables avoiding backtracking on the string 'S'.

The KMP Matcher

With string 'S', pattern 'p' and prefix function 'Π' as inputs, finds the occurrence of 'p' in 'S' and returns the number of shifts of 'p' after which occurrence is found.

The prefix function, Π

```
Following algorithm the prefix function, \Pi:
```


П 0 0 1

Example: compute Π for the pattern 'p' below:

```
Step 4: q = 5, k =2
                        q 1 2 3 4 5 6 7
       \Pi[5] = 3
                        р
                                а
                                   b
                                      а
                                         c a
                        П
                          0
                                   2
                        q 1 2 3 4 5 6 7
Step 5: q = 6, k = 3
                        p a b a b a c a
       \Pi[6] = 1
                        П 0 0 1 2 3 1
                        q 1 2 3 4 5 6 7
Step 6: q = 7, k = 1
                        p a b a b a c a
       \Pi[7] = 1
                        Π 0 0 1
                                   2
                                      3
                        q 1 2 3 4 5 6 7
After iterating 6 times, the prefix
 function computation is
                        p a b A b a c a
  complete:
                        П 0 0 1 2 3 1
```

```
The KMP Matcher
The KMP Matcher, with pattern 'p', string 'S' and prefix function '\Pi' as input, finds a match of p in S.
Following pseudocode computes the matching component of KMP algorithm:
 KMP-Matcher(S,p)
 n = length[S]
 m = length[p]
 B \Pi = Compute-Prefix-Function(p)
 l q = 0
                                                 //number of characters matched
5 \text{ for } i = 1 \text{ to n}
                                                //scan S from left to right
     while q > 0 and p[q+1] != S[i]
             q = \Pi[q]
                                            //next character does not match
     if p[q+1] = S[i]
            q = q + 1
                                            //next character matches
           == m //is all of p matched?
print "Pattern occurs with shift" i – m
     if a == m
                                            // look for the next match
Note: KMP finds every occurrence of a 'p' in 'S'. That is why KMP does not terminate in step 12, rather it searches remainder of 'S' for any more occurrences of 'p'.
```

```
Illustration: given a String 'S' and pattern 'p' as follows:

S bacbababababacaca

P abababaca

Let us execute the KMP algorithm to find whether 'p' occurs in 'S'.

For 'p' the prefix function, \( \Pi \) was computed previously and is as follows:

\[
q 1 2 3 4 5 6 7 \]

\[
p a b A b a c a
\]

\[
\Pi 0 0 1 2 3 1 1
\]
```

```
Initially: n = size of S = 15;
    m = size of p = 7

Step 1: i = 1, q = 0
    comparing p[1] with S[1]

S    b    a    c    b    a    b    a    b    a    c    a    a    b

p    a    b    a    b    a    c    a
    P[1] does not match with S[1]. 'p' will be shifted one position to the right.

Step 2: i = 2, q = 0
    comparing p[1] with S[2]

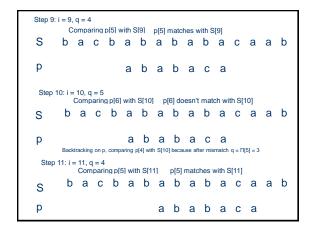
S    b    a    c    b    a    b    a    b    a    c    a    a    b

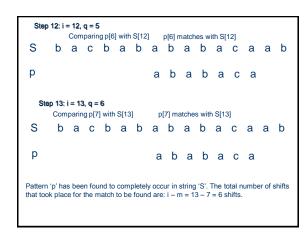
p    a    b    a    b    a    b    a    b    a    b    a    c    a    a    b

p    a    b    a    b    a    b    a    b    a    b    a    c    a    a    b
```

```
Step 6: i = 6, q = 1

Comparing p[2] with S[6] p[2] matches with S[6]
S
    bacbababacaab
              ababaca
р
Step 7; i = 7, q = 2
      Comparing p[3] with S[7] p[3] matches with S[7]
    bacbababacaab
               ababaca
p
 Step 8: i = 8, q = 3
       Comparing p[4] with S[8] p[4] matches with S[8]
    bacbababacaab
S
р
               ababaca
```





##