Knuth-Morris-Pratt Algorithm

Kranthi Kumar Mandumula

Knuth-Morris-Pratt Algorithm

Kranthi Kumar Mandumula

December 18, 2011

outline

- Definition
- History
- Components of KMP
- Algorithm
- Example
- Run-Time Analysis
- Advantages and Disadvantages
- References

Definition:

- Best known for linear time for exact matching.
- Compares from left to right.
- Shifts more than one position.
- Preprocessing approach of Pattern to avoid trivial comparisions.
- Avoids recomputing matches.

History:

Kranthi Kumar Mandumula

 This algorithm was conceived by Donald Knuth and Vaughan Pratt and independently by James H.Morris in 1977.

History:

- Knuth, Morris and Pratt discovered first linear time string-matching algorithm by analysis of the naive algorithm.
- It keeps the information that naive approach wasted gathered during the scan of the text. By avoiding this waste of information, it achieves a running time of O(m + n).
- The implementation of Knuth-Morris-Pratt algorithm is efficient because it minimizes the total number of comparisons of the pattern against the input string.

Components of KMP:

- The prefix-function □ :
 - ★ It preprocesses the pattern to find matches of prefixes of the pattern with the pattern itself.
 - ★ It is defined as the size of the largest prefix of P[0..j-1] that is also a suffix of P[1..j].
 - ★ It also indicates how much of the last comparison can be reused if it fails.
 - ★ It enables avoiding backtracking on the string 'S'.

```
m \leftarrow length[p]
     a[1] \leftarrow 0
     k \leftarrow 0
     for q \leftarrow 2 to m do
        while k > 0 and p[k + 1] \neq p[q] do
           k \leftarrow a[k]
        end while
        if p[k + 1] = p[q] then
           k \leftarrow k + 1
        end if
        a[q] \leftarrow k
     end for
return □
Here a = \Box
```

Knuth-Morris-Pratt Algorithm

Computation of Prefix-function with example:

Kranthi Kumar Mandumula

Let us consider an example of how to compute
 for the pattern 'p'.

Initially:
$$m = length[p] = 7$$

 $\square[1] = 0$
 $k=0$

where m, $\square[1]$, and k are the length of the pattern, prefix function and initial potential value respectively.

Step 1:
$$q = 2$$
, $k = 0$
 $\square[2] = 0$

q	1	2	3	4	5	6	7
р	а	b	а	b	а	С	а
П	0	0					

a y b no se repiten por eso 0

Step 2:
$$q = 3, k = 0$$

ahora a coincide de nuevo, ahora cambia a 1

q	1	2	3	4	5	6	7
р	а	b	а	b	а	С	а
П	0	0	1				

Step 3:
$$q = 4$$
, $k = 1$
 $\lceil [4] = 2 \rceil$

q	1	2	3	4	5	6	7
р	а	b	а	b	а	С	а
П	0	0	1	2			

Step 4:
$$q = 5$$
, $k = 2$
 $\sqcap[5] = 3$

q	1	2	3	4	5	6	7
р	а	b	а	b	а	С	а
П	0	0	1	2	3		

Step 5:
$$q = 6$$
, $k = 3$
 $\sqcap [6] = 1$

q	1	2	3	4	5	6	7
р	а	b	a	b	a	С	а
П	0	0	1	2	3	1	

Se resetea el potencial a 1 porque c no estaba antes

Step 6:
$$q = 7$$
, $k = 1$
 $\Box [7] = 1$

K se mantiene en la memoria

q	1	2	3	4	5	6	7
р	а	b	а	b	а	С	а
П	0	0	1	2	3	1	1

After iterating 6 times, the prefix function computations is complete:

ſ	q	1	2	3	4	5	6	7
Ī	р	а	b	Α	b	а	С	а
Ī	П	0	0	1	2	3	1	1

The running time of the prefix function is O(m).

Step 5: look for the next match.

Kranthi Kumar Mandumula

Step 1: Initialize the input variables:
 n = Length of the Text.
 m = Length of the Pattern.
 □ = Prefix-function of pattern (p).
 q = Number of characters matched.

Step 2: Define the variable:
 q=0, the beginning of the match.

Step 3: Compare the first character of the pattern with first character of text.
 If match is not found, substitute the value of □[q] to q.
 If match is found, then increment the value of q by 1.

Step 4: Check whether all the pattern elements are matched with the text elements.
 If not, repeat the search process.
 If yes, print the number of shifts taken by the pattern.

```
n \leftarrow length[S]
    m \leftarrow length[p]
    a ← Compute Prefix function
    a \leftarrow 0
    for i \leftarrow 1 to n do
       while q > 0 and p[q + 1] \neq S[i] do
          a \leftarrow a[a]
          if p[q + 1] = S[i] then
             q \leftarrow q + 1
          end if
          if q == m then
             q \leftarrow a[q]
          end if
       end while
    end for
Here a = \Box
```

Example of KMP algorithm:

Kranthi Kumar Mandumula

Now let us consider an example so that the algorithm can be clearly understood.

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

```
Initially: n = size 	ext{ of } S = 15;

m = size 	ext{ of } p=7
```

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

String bacbabababacaab

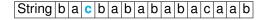
Pattern a b a b a c a

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

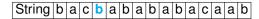
Step 2:
$$i = 2$$
, $q = 0$
comparing $p[1]$ with $S[2]$

String bacbabababacab

Pattern a b a b a c a



Pattern a b a b a c a



String bacbababacaab

Pattern a b a b a c a

Step 6: i = 6, q = 1comparing p[2] with S[6] p[2] matches with S[6]

String b a c b a b a b a b a c a a b

Step 7: i = 7, q = 2comparing p[3] with S[7] p[3] matches with S[7]

String bacbababababab

Pattern a b a b a c a

Step 8: i = 8, q = 3comparing p[4] with S[8] p[4] matches with S[8]

String b a c b a b a b a b a c a a b

Mandumula

String b a c b a b a b a b a c a a b

Pattern a b a b a c a

Step 10: i = 10, q = 5comparing p[6] with S[10] p[6] doesn't matches with S[10]

String b a c b a b a b a c a a b

Pattern a b a b a c a

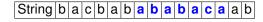
Backtracking on p, comparing p[4] with S[10] because after mismatch $q = \prod [5] = 3$

String bacbababacaab

Pattern a b a b a c a

Step 12: i = 12, q = 5comparing p[6] with S[12] p[6] matches with S[12]

String b a c b a b a b a c a a b



Pattern a b a b a c a

pattern 'p' has been found to completely occur in string 'S'. The total number of shifts that took place for the match to be found are: i - m = 13-7 = 6 shifts.

Run-Time analysis:

- O(m) It is to compute the prefix function values.
- O(n) It is to compare the pattern to the text.
- Total of O(n + m) run time.

Advantages and Disadvantages:

- Advantages:
 - ★ The running time of the KMP algorithm is optimal (O(m + n)), which is very fast.
 - ★ The algorithm never needs to move backwards in the input text T. It makes the algorithm good for processing very large files.

Advantages and Disadvantages:

- Disadvantages:
 - ⋆ Doesn't work so well as the size of the alphabets increases. By which more chances of mismatch occurs.

- Graham A.Stephen, "String Searching Algorithms", year = 1994.
- Donald Knuth, James H. Morris, Jr, Vaughan Pratt, "Fast pattern matching in strings", year = 1977.
- Thomas H.Cormen; Charles E.Leiserson., Introduction to algorithms second edition, "The Knuth-Morris-Pratt Algorithm", year = 2001.