Kranthi Kumar Mandumula

Knuth-Morris-Pratt Algorithm

Kranthi Kumar Mandumula

December 18, 2011

Kranthi Kumar Mandumula Knuth-Morris-Pratt Algorithm

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

- 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.

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

- 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.

• 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] ← 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 = \square
```

Kranthi Kumar Mandumula

Kranthi Kumar Mandumula 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'.

Pattern	а	b	a	b	a	С	a

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

Kranthi Kumar Mandumula

Step 1:
$$q = 2, k = 0$$

 $\Box[2] = 0$

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

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

 $\Box[3] = 1$

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

Kranthi Kumar Mandumula

・ロト・日ト・ヨト・ヨト ヨー のへで ula Knuth-Morris-Pratt Algorithm

Kranthi Kumar Mandumula

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

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

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

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

Kranthi Kumar Mandumula

・ロト・日ト・ヨト・ヨト ヨー のへで ula Knuth-Morris-Pratt Algorithm

Kranthi Kumar Mandumula

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

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

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

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

Kranthi Kumar Mandumula

・ロト・日ト・ヨト・ヨト ヨー のへで ula Knuth-Morris-Pratt Algorithm

Kranthi Kumar Mandumula

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

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

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

Step 1: Initialize the input variables: n = Length of the Text. m = Length of the Pattern. \square = Prefix-function of pattern (p). g = Number of characters matched. Step 2: Define the variable: a=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 $\sqcap[q]$ to q. If match is found, then increment the value of g 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. Step 5: look for the next match.

```
n \leftarrow length[S]
    m \leftarrow length[p]
    a \leftarrow 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
```

Kranthi Kumar Mandumula

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

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

Pattern a b a b a c a

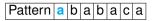
Let us execute the KMP algorithm to find whether 'p' occurs in 'S'.

◆□ ▶ ◆□ ▶ ◆ □ ▶ ◆ □ ▶ ◆ □ ▶

Kranthi Kumar Mandumula

```
Initially: n = size of S = 15;
m = size of p=7
Step 1: i = 1, q = 0
comparing p[1] with S[1]
```

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



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

```
Pattern a b a b a c a
```

Knuth-Morris-Pratt Algorithm

Kranthi Kumar Mandumula

Step 3: i = 3, q = 1
 comparing p[2] with S[3] p[2] does not match with S[3]

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

Backtracking on p, comparing p[1] and S[3] Step 4: i = 4, q = 0comparing p[1] with S[4] p[1] does not match with S[4]

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

Pattern a b a b a c a

Kranthi Kumar Mandumula Knuth-Morris-Pratt Algorithm

Knuth-Morris-Pratt Algorithm

Kranthi Kumar Mandumula

Kranthi Kumar Mandumula

Step 5: i = 5, q = 0comparing p[1] with S[5]

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 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

Pattern a b a b a c a

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

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

Step 8: i = 8, q = 3 comparing 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

Pattern a b a c a

Kranthi Kumar Mandumula

<□▶ < □▶ < □▶ < □▶ < □▶ < □ > ○ < ○

Step 9: i = 9, q = 4comparing p[5] with S[9] p[5] matches with S[9]

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 = 5

comparing p[6] with S[10] p[6] doesn't matches with S[10]

String b a c b a 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 $q = \Box[5] = 3$

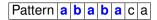
Kranthi Kumar Mandumula Knuth-Morris-Pratt Algorithm Knuth-Morris-Pratt Algorithm

Kranthi Kumar Mandumula

Kranthi Kumar Mandumula

Step 11: i = 11, q = 4 comparing p[5] with S[11]

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



Step 12: i = 12, g = 5 comparing 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

Kranthi Kumar Mandumula

Step 13: i = 13, q = 6 comparing p[7] with S[13] p[7] matches with S[13]

String b a c b a 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.

- 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:

★ 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.

• Disadvantages:

★ Doesn't work so well as the size of the alphabets increases. By which more chances of mismatch occurs.

Kranthi Kumar Mandumula

- 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.