International Journal of Latest Research in Science and Technology **ISSN (Online):2278-5299**
Volume 4, Issue 2: Page No.23-25, March-April 2015
https://www.mnkpublication.com/journal/ijlrst/index.php

ANALYSIS OF NUMBER OF LOOPS EXECUTED IN KMP ALGORITHM

Arif Khan , Li Chen
Department of Computer Science & Information Technology
University of the District of Columbia
Washington DC, USA 20008

Abstract:- We have explained difficulties in understanding the KMP algorithm, and have analyzed the number of executions of the loops in pattern matching phase of the KMP method required to improve the time complexity.

Keyword: KMP Algorithm

1. INTRODUCTION

String matching is a popular method for many applications. It is also an important topic in algorithm analysis courses at both undergraduate and graduate levels of education. Students generally found difficulties to understand KMP algorithm. This article is specifically written to present a simple, clear explanation of the KMP algorithm. In addition, it provides an analysis of the method and a modification of the error related to the execution of the loops during pattern matching described in some text-books.

KMP algorithm was designed [1] by Donald Kunth, James H Morris, and Vaughan Pratt in 1977. Using this technique we can solve the problem of finding occurrence of a pattern of string within another string. The former is known as pattern string and the later is called text string. This method has two phases. In the first phase, we find the partial match within the pattern string. The results are used in the second phase to find the matching of the pattern string in the text. The method has overall time complexity O(m + n), where m and n are the number of characters in pattern string (P) and text string (T) respectively. The essence of KMP algorithm has been extended to generalize the pattern matching problem for two dimensional sub-array matching [2 – 5].

The text editing service frequently uses this type of string matching solution in various situations. String matching solutions also have many other important applications in the fields of search engine, Network Intrusion Detection System, DNA sequencing, etc. Several different algorithms have been proposed to solve the string matching problems [6]. Some of them are Brute Force, Boyer Moore, approximate string matching, KMP, etc. All, other than KMP, have time complexity O(mn).

2. KMP Algorithm and its Complexity Analysis.

In this section, we explain the main features of the KMP algorithm and analyze the number of loops executed in KMP algorithm. The number of execution of loop is important because it directly relates to time complexity.

We first discuss and explain three terminologies, namely "Prefix", "Suffix", and "failure function" which are central to the KMP algorithm [7].

2.1 Prefix, Suffix and Failure Function

Prefix: A string Y is a prefix of a string X if X = YZ for some string Z. In other words we can define prefix that has all the characters in a string with one or more cut-offs, all at the end. Example: "G", "Gr", "Gro", and "Grou" are all prefixes of the string "Group".

Suffix: A string Y is a suffix of a string X if X = ZY for some string Z. In other words, we can define suffix that has all the characters in a string with one or more cut-offs at the beginning. Example: "roup", "oup", "up", and "p" are all suffixes of string "Group." An example of the text string and the pattern string with characters, which are numbered as below:

Text string	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	a	b	a	a	b	b	a	b	a	a	b	a	a	b	a	b

Pattern string	0	1	2	3	4	5	6
	a	b	a	a	b	a	b

In the above example we have a value of n = 16 and a value of m = 7.

Failure function: It is also called the prefix function. Apparently the KMP algorithm is similar to the brute-force algorithm without failure function, which considers shifts in order from 1 to $n - m$, and finds out whether the pattern matches at that shift. The distinction between these two algorithms is that the KMP algorithm uses information generated from the partial matches of the pattern and text, and skips the shifts that are guaranteed not to result in a match.

The said partial matches are generated by prefix function or failure function. To determine the failure function we exploit the concept of prefix and suffix as discussed earlier.

Publication History
Manuscript Received : *6 April 2015*
Manuscript Accepted : *11 April 2015*
Revision Received : *26 April 2015*
Manuscript Published : *30 April 2015*

With the help of failure function, we build a table known as partial match table. We demonstrate below how to get the failure function, and how to construct from it the partial match table. Let us consider a pattern given as: abaabab. For this pattern we have prefix and suffix of part of the pattern with j = 0, 1, 2, 3, 4, 5, and 6 as shown in Table I:

Table I

j	0	1	2	3	4	5	6
Part of Pattern with j [P′(j)]	a	ab	aba	abaa	abaab	abaaba	abaabab
Prefixes of [P′(j)]	null	a	a, ab	a, ab, aba	a, ab, aba, abaa	a, ab, aba, abaa, abaab	a, ab, aba, abaa, abaab, abaabab
Suffixes of [P′(j)]	null	b	ba, a	baa, ba, a	baab, aab, ab, b	baaba, aaba, aba, ba, a	baabab, aabab, abab, bab, ab, b

Now we compute the failure function which is obtained from the partial matches of the string. To get the partial match we use the concept of prefixes and suffixes and their pattern matches. If the pattern of the prefixes and suffixes of the part of the given pattern string matches for any value j, then the number of character in the matching prefix and suffix string will be the value of the failure function for that j. If we call the part of the given pattern as subpattern, then the failure function can be defined as the length of the longest prefix in the subpattern that matches a suffix in the same subpattern. If it does not match, then the value of the failure function will be 0. In Table I we have shown the matching pair of prefix and suffix by underline for various value of j.

We also note that for j = 0 and 1, there is no matching. So the failure function, F(j) for j = 0 and 1 is 0. If we count the number of characters of the matching string, then we get the value of F(j) for any value of j. It should be remembered that the value of $F(j)$ is obtained only from the partial matching of the pattern P. So using Table I, we can build the partial match table for the pattern P which is shown in Table II.

Table II

j	0	1	2	3	4	5	6
Characters of pattern [P(j)]	a	B	a	a	b	a	b
Failure function [F(j)]	0	0	1	1	2	3	2

We can use these failure functions and the partial matching table to check the pattern matching. The algorithm used for this purpose has been widely studied [7]. While doing this, we basically encounter two different pseudo-codes. As mentioned earlier in the present article, our objective is also to analyze the pseudo code of the KMP string matching algorithm for the loop that also has another loop (e.g., one loop inside another loop; thus two loops in total), as stated in some text books. This type of structure increases time complexity.

3. Analysis on Number of Loops executed in KMP Method

We overcome the above problem by modifying the pseudo code while preserving the matching algorithm single for loop (one loop). We also determine the number of total loops executed to find the pattern match resulting from both pseudo codes. We also compare our results to show the effectiveness of the algorithm mentioned here.

3.1 Pseudo code with for loop and while loop:

Check_Pattern Method (T, P) // It returns Boolean value – whether match found or not.

```
n = size(T)
F[i] : Obtained from Failure_Function method
k = 0
m = size(P)
for (j from 0 to n) //Scan the text from left to right
  {
     while (k ≠ 0 and P[k] ≠ T[j])  //To check the mismatch and reset the pattern string
        {
          k = F[k]  //Iterative process
        }
    if (P[k] == T[j])  //To check the matching
        {
          k = k + 1
          if (k == m) //Check all locations before the last one have been matched
             return true //Match found
        }
  } //All pattern element compared
 return false // All text string scanned but no pattern found
```

As stated earlier, we find that the algorithm stated above has one loop that also contains another loop. So it contains two loops. Now we can consider the loop and modify the pseudo code as below

3.2 Modified Pseudo code with while loop only:

Check_Pattern Method (T, P) // It returns Boolean value – whether match found or not

```
n = size(T)
m = size(P)
F[i] : Obtained from Failure_Function method
k = 0
j = 0
while (j<n)
{
  If P[k] == T[j] // If one pattern element matches with the string element
    {
      j = j + 1
      k = k + 1
      if(k==m)  //Check all locations before the last one have been matched
        {
          return true // Match found
        }
    }
  else //Match not found
```

```
    {
      k = F[k]
      j = j + 1
      }
    }//End of "while loop"
  return false //pattern not found after scanning all text string
```

Based on the above two pseudo codes we have performed the pattern matching and have presented our result as given in Table III. This result demonstrates that the pseudo code only with the "while loop" corresponds to appreciably lower number of execution of loops compared to the pseudo code with "for loop" containing another "while (e.g., two loops)". In addition to that, we can conclude from our results that during pattern matching phase the time complexity with one loop (while only) is strictly O(n), n is the number of elements of the text string. But with two loops (for and while together), the time complexity is at best O(n + m), where m is the number of element of pattern string. As the time complexity during this phase depends only on the text string, the method with one loop is more efficient.

Table III

Summary of analysis of KMP algorithm (Check_Pattern Method)

Text Pattern (T): abaabbabab [Number of elements in string T (n): 10]
Search Pattern 1(P1): abaabab [Number of elements in string P1 (m): 7]
Search Pattern 2 (P2): aabba [Number of elements in string P2 (m): 5]
Search Pattern 3 (P3): abababca [Number of elements in string P3 (m): 8]
Search Pattern 4 (P4): abbaba [Number of elements in string P4 (m): 6]

Pattern Found and Algorithm Methods	Time Complexity	Number of execution of loops needed for getting matches of the pattern			
		P1	P2	P3	P4
Pattern found?	---	No	Yes	No	Yes
Check_Pattern Method (T, P) with for loop that contains while loop	O(n + m)	13	8	13	11
Check_Pattern Method (T, P) with while loop only	O(n)	10	7	10	9

4. Conclusions

In this article we have explained prefix, suffix and failure function to understand the KMP algorithm without any difficulties. We have analyzed the method with introduction of a new loop pattern for achieving better time complexity.

REFERENCES

[1] Donald Kunth, James H Morris Jr., and Pratt Vaughan, "Fast pattern matching in string", SIAM Journal of computing, Vol 6, No 2 pp 323 – 350 (1977)

[2] Alfred V Aho and Margaret J Corasick, "Efficient string matching: An aid to bibliographic search", Communications to ACM, Vol. 18, No 6, pp 333 – 340 (1975).

[3] Rui Feng Zhu and Tadao Takaoka, "A Technique for two Dimensional Pattern Matching", Communications of ACM, Vol. 32, pp 1110 – 1120 (1989)

[4] Baeza-Yates Recardo and Régnier Mireille, "Fast two dimensional pattern matching", Information Processing Letters, Vol. 45 pp 51 – 57, (1993)

[5] Saima Hasib, Mahak Motwani, and Amit Saxena, "Importance of Aho-Corasick String Matching Algorithm in Real World Applications", Int. J. Computer Science and Information Technologies, Vol. 4, pp 467 – 469 (2013)

[6] Nimisha Singla and Deepak Garg, "String Matching Algorithm and their Applicability in Various Applications", Int. J. Soft Computing and Engineering, Vol. 1 pp 218 – 222 (2012).

[7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifferd Stein, "Introduction to Algorithms", Third Edition, MIT Press and PHI Learning Private Limited, pp. 1002 – 1013 (2014).