Scientific Journal Impact Factor (SJIF) : 1.711
International Journal of Modern Trends in Engineering and Research
ISSN (ONLINE) : 2349 - 9745 ISSN (PRINT) : 2393 - 8161

A Composite Algorithm for String Matching

Neha R Dalal[1], Poonam Jadhav[2]
[1] *Lecturer, Department of MCA, KLS's Gogte Institute Of Technology, Belgaum, Affiliated to VTU Belgaum*
[2] *Asst. Professor, Department of MCA, KLS's Gogte Institute Of Technology, Belgaum, Affiliated to VTU Belgaum*

Abstract— String matching algorithms play an important role in the field of Computer Science as they help us extract the relevant data from a huge database. Due to complex nature of technical data stored in huge databases, making use of algorithms like the Knuth-Morris-Pratt Algorithm, Horspool's Algorithm, or the Boyer-Moore algorithm for retrieving a subset of data will be efficient. But it is always preferable to have an advanced variation of any of these algorithms so as to be more time and space efficient. Hence we propose a composite algorithm that will be a variant of Knuth-Morris-Pratt Algorithm, which will prove to be more time and space efficient.

Keywords- Knuth-Morris-Pratt algorithm, string matching algorithm, time efficient, space efficient, Horspool's algorithm, Boyer-Moore algorithm, Composite algorithm

I. INTRODUCTION

In day to day life we come across my instances where we need to search for a particular pattern in a given text. For this there are many string matching algorithms available in computer science. For example we have Knuth-Morris-Pratt Algorithm and the Horspool's Algorithm or the Boyer Moore algorithm which are considerably efficient than their counterparts available.

In Knuth-Morris-Pratt algorithm there is independence from alphabet size and linearity in the pattern length during worst case execution of the algorithm. In comparison, the Horspool's Algorithm or the Boyer Moore execution is faster than the Knuth-Morris-Pratt algorithm and provides optimality in average case and best case execution of the algorithm.

The main objective of any string matching algorithm is to reduce the number of character comparisons done in worst case execution and / or average case execution of the algorithm. Another objective of any string matching algorithm is to be time efficient. In order to achieve the above said objectives we propose a new algorithm called the Composite Algorithm for string matching. This algorithm is a variation of the Knuth-Morris-Pratt Algorithm which will be more time efficient than the Horspool's algorithm or the Boyer Moore algorithm.

II. REVIEW OF RELEVANT LITERATURE

Ever since the publication of Knutt-Moris-Pratt algorithm and the Boyer-Moore algorithm, many research papers have been publishes that cover in the topic of string matching. Most of the string matching algorithms present till date are both time and space efficient. Several comparative surveys have also been conducted on these algorithms. Many algorithms are variants of either the Knutt-Morris-Pratt algorithm or the Boyer-Moore algorithm. The main focus of these algorithms is to reduce the number of character comparison and to reduce the amount of time required in worst/average case.

III. PROPOSED ALGORITHM

In order to search for a pattern p in given text t the algorithm Composite will match p along t by shifting p from left to right. Initially we use two variables i and j, i is initialized to 1... n-m+1in text t and j is initialized to 1of p. Now we have position m of p aligned with position i=i+m-j in t. In Knuth-Morris-Pratt string matching the pattern p is matched with t[i],t[i+1] and so on from left to right until either the entire pattern matches subtext or a mismatch occurs. The mismatch may occur at some position j. If j>1 then a partial match has been found. Both i and j are incremented and the process continues.

The basic idea behind the Composite algorithm is that if a match of pattern p is not found in the text t then find the next position i at which t[i] = p[m]. Knuth Morris Pratt algorithm is then applied on p[1..m-1] and x[i-m+1..i-1]. If a partial match of pattern p is found in the text t then Knuth Morris Pratt algorithm is continued on p[1...m]. Thus the Knuth Morris Pratt algorithm needs to compare position I of x with position j of p, where j may be the extension of a partial match with p[1…j-1] or the result of an assignment $j=p^1(j)$. The variant of Knuth-Morris-Pratt algorithms is as follows:

```
   Composite (m; i,j)
while j < m and t[i]=p[j] do
        i ← i+1; j ← j+1
If j=m then
        i ← i+1; j ← j+1; output i – m
```

IV. IMPLEMENTATION

In order to show that the composite algorithm is more time efficient than the existing string matching algorithms, an experiment was conducted by implementing the composite algorithm using PHP language. The pseudo-code is as follows:

```
function print(  $pos )
{
        global $matches;
        $matches++;
 }
function makebetap( &$p, $m )
{
global $betap;
$i = 0;
$j = $betap[0] = -1;
while( $i < $m )
        {
                while( ($j > -1) && ($p[$i] != $p[$j]) )
                {
                   $j = $betap[$j];
                }
                if( $p[$i] == $p[++$j] )
                   $betap[$i] = $betap[$j];
                else
                   $betap[$i] = $j;
                $i++;
   }
```



```
}
function makeDelta( &$p,$m )
{
        global $Delta;
   for( $i = 0; $i < 10000; ++$i )
                $Delta[$i] = $m + 1;
   for( $i = 0; $i < $m; ++$i )
                $Delta[ ord($p[$i]) ] = $m - $i;
}

function Composite( &$p, $m, &$x, $n )
{
        global $Delta;
        global $betap;
        global $matches;

   $i = 0;
        $j = 0;
        $mp = $m-1;
        $ip = $mp;
   if( $m < 1 )
                return;

   makebetap( $p, $m );
   makeDelta( $p, $m );

   while( $ip < $n ) {
   $j = 0;
        if( $j <= 0 ) {
        while( $p[ $mp ] != $x[ $ip ] ) {
          $a = isset($x[ $ip+1 ]) ? $x[ $ip+1 ] : 0;
                $ip += $Delta[ ord($a) ];
                 if( $ip >= $n )
                        return;
                 }
                 $j = 0;
                 $i = $ip - $mp;
                 while( ($j < $mp) && ($x[$i] == $p[$j])
++$i;
++$j;
                 }
                 if( $j == $mp ) {
                output( $i-$mp );
                        ++$i;
                        ++$j;
                 }
                 if( $j <= 0 )
```

```
                ++$i;
            else
                $j = $betap[$j-1];
        } else {
            while( ($j < $m) && ($x[$i] == $p[$j]) )
                {
                    ++$i;
                    ++$j;
                }

            if( $j == $m )
                {
                    print( $i-$m );
                }
            $j = $betap[$j];
        }
        $ip = $i + $mp - $j;
    }
 return $matches;
 }
?>
```

Fig 4.1 shows how the page looks after the user has browsed and uploaded a file. The pattern that the user wishes to search in the uploaded file is 'the'.

Fig. 4.1 Home Page

Fig 4.2 displays the time efficiency of Composite Algorithm. It also show comparison with Horspool's Algorithm.

Fig 4.2 Time efficiency of Composite algorithm

Fig 4.3 displays the file uploaded by the user. Also highlights all the matches of the pattern found in file.

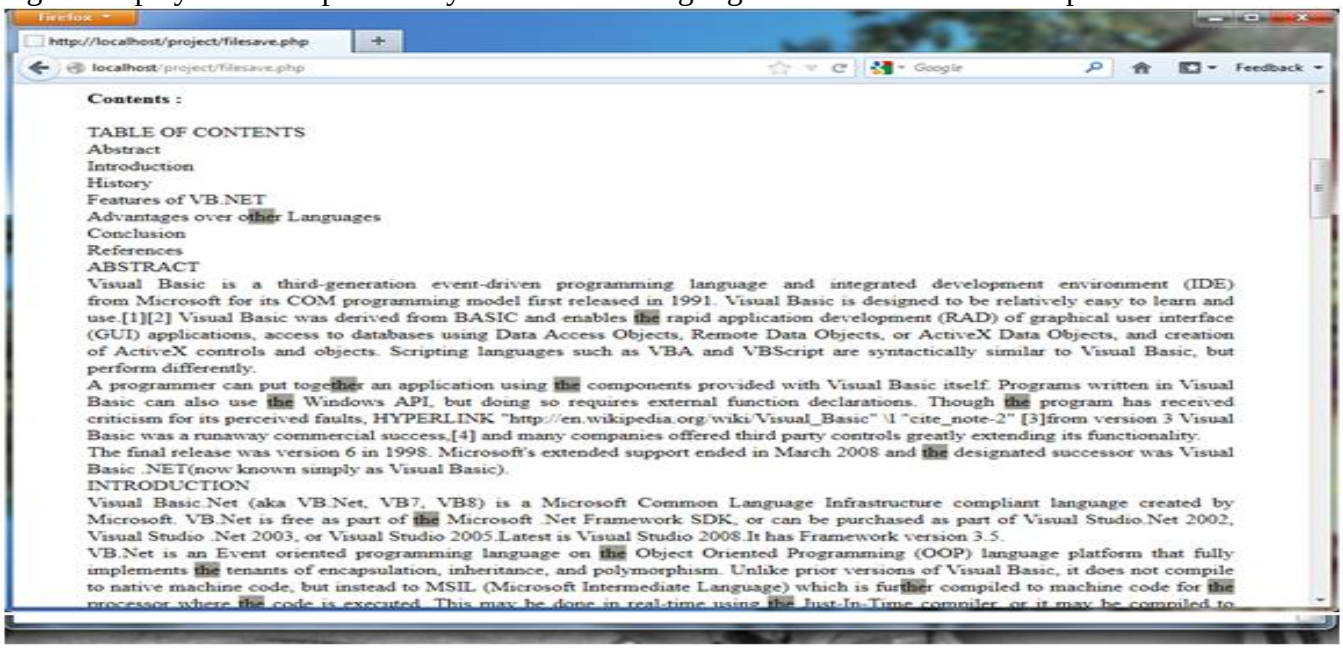

Fig. 4.3 Result Page

V. CONCLUSION

A number of experiments were conducted to compare the Composite algorithm with other time efficient algorithms like Knuth-Morris-Pratt algorithm and the Horspool's algorithm. The horspool's algorithm was converted to php code to implement and compare it with the Composite algorithm. Minimum 20 repetitions for a given pattern and text were conducted and the fastest time was taken as the result. The fastest time is closest to the minimum time that the run would have taken independent of interrupts, cache misses, or other external effects. All timings include preprocessing as well as the string search algorithm itself.

VI.ACKNOWELEDGMENTS

Researchers wishes to express a sense of gratitude and love to beloved parents for their manual support, strength, help and everything.

REFERENCES

[1] R.A. Baeza-Yates, G. Gonnet, A new approach to text searching Commun. Assoc. Comput. Mach. 35 (1992)74–82.
[2] R.S. Boyer, J.S. Moore, A fast string searching algorithm, Commun Assoc. Comput. Mach. 20 (1977) 762-772.
[3] C. Charras, T. Lecroq, Exact string matching algorithms Laboratoire d'Informatique, Université de Rouen, 1997, http://www-igm.univmlv.fr/lecroq/string/index.html. F.Franek, C.G. Jennings, W.F. Smyth, A simple fast hybrid pattern-matching algorithm, in: A. Apostolico, M.Crochemore, K. Park (Eds.),16th Annual Symp. Combinatorial Pattern Matching, in: Lecture Notes in Computer Science, vol. 3537, Springer-Verlag, 2005, pp. 288–297.
[4] L. Colussi, Correctness and efficiency of the pattern matching algorithms, Information and Computation 95(1991) 225–251.
[5] L. Colussi, Fastest pattern matching in strings, J Algorithms 16 (1994) 163–189.

