



## International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2015

# Comparative Study of SA string Matching Algorithm with other Familiar String Matching Algorithm

Shibdas Bhattacharya, AratrikaSaha

Lecturer, Dept. of Computer Science & Technology , Technique Polytechnic Institute, West Bengal, India

3<sup>rd</sup> year Student , Dept. of Computer Science & Engineering , Kalyani Govt. Engineering College , West Bengal , India

**ABSTRACT:**The purpose of this research is to compare a very efficient String matching algorithm known as SA string matching with other familiar string matching algorithms and show that the complexity is much less than all of these algorithms .

**KEYWORDS:** String Matching , Complexity analysis , Preprocessing time , Matching time

### I. INTRODUCTION

We are first going to discuss about the basic concepts of string and string matching algorithms .

#### A. Definition Of String

In computer programming a string is traditionally a sequence of characters, either as a literal constant or some kind of variable . The latter may allow its elements to be mutated and the length changed , or it may be fixed (after creation) . A string is generally understood as a data type and is often implemented as an array of bytes (or words) that stores a sequence of elements, typically characters, using some character encoding .

#### B. Definition Of String Matching Algorithms

(1) Text-editing programs frequently need to find occurrences of a pattern in the text . Typically , the text is a document being edited , and the pattern searched for is a particular word supplied by the user. Efficient algorithms for this problem-called “String Matching” . It can greatly aid the responsiveness of the text-editing programs . Among their many other applications, string-matching algorithms search for particular patterns in DNA sequences .

#### C. Formal Illustration

We formalize the string matching problem as follows.

We assume that the text is an array  $T[1..n]$  of length  $n$  and that the pattern is an array  $P[1..m]$  of length  $m \leq n$  . We further assume that the elements of  $P$  and  $t$  are drawn from a finite alphabet  $\Sigma$  . We say that pattern  $P$  occurs with shift  $s$  in text  $T$  (or, equivalently , that pattern  $P$  occurs beginning at position  $s+1$  in text  $T$ ) if  $0 \leq s \leq n-m$  and  $T[s+1..s+m] = P[1..m]$  (that is , if  $T[s+j] = P[j]$  , for  $1 \leq j \leq m$ ). If  $P$  occurs with shift  $s$  in  $T$  , then we call  $s$  a valid shift ; otherwise we call  $s$  an invalid shift . The string-matching problem is the problem of finding all valid shifts with which a given pattern  $P$  occurs in a given text  $T$  .

These are the basic concepts of string matching algorithm .Some of the common string matching algorithms that already exist are : Naïve String matching algorithm , Rabin –Karp string search algorithm , Finite State automaton based search , Boyer-Moore string search algorithm , Bitmap algorithm .

In SA String Matching Algorithm we have achieved a complexity which is much less than all the above mentioned algorithm .

It is : **Pre-processing =  $\Theta(n)$  , Matching =  $O(mk)$**



# International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2015

Where  $m$  = Length of the pattern ,  $n$  = Length of the searchable text ,  $K$  = number of. Time there is a hitm.

In the illustration part we will also show that our algorithm is also much efficient than some lower complexity algorithm for many different cases .

## II. SURVEY

Before we go in detailed study of our algorithm and its comparisons with other algorithms we would like to give a brief overview of the already existing algorithms along with their complexity analysis and try to explain why our algorithm performs better than them

In TABLE 1 we can see that all the mentioned algorithms are having complexities much higher than our algorithms' complexity which is Pre-processing =  $\Theta(n)$  , Matching =  $O(mk)$  . In the illustration and comparison part we will show that how it performs better than the algorithms .

TABLE I  
THE ALGORITHMS WITH THEIR RESPECTIVE COMPLEXITY ANALYSIS

ALGORITHM	PREPROCESSING TIME	MATCHING TIME
Naïve String , matching algorithm	[1],0(no preprocessing)	[1] , $O((n-m+1)m)$
Rabin –Karp string search algorithm	[2], $\Theta(m)$	[2], $O(n+m)$
Finite State automaton based search	[3] $\Theta(mk)$	[3] $O(n)$
Boyer-Moore string search algorithm	$\Theta(m+k)$	$O(n)$
Bitmap algorithm	$\Theta(m+k)$	$O(mn)$

Where  $m$  = Length of the pattern ,  $n$  = Length of the searchable text ,  $k$  = particular constant .

### NOTE

Also we will see in the next section that if **the given pattern is not present in the text** at all then it will produce result only after precomputation and wont require the matching part of the algorithm . Thus the **complexity then reduces to only  $O(n)$**  .This provision is not present in any of the above mentioned algorithm thus it performs better .

## III. PROPOSED ALGORITHM

Following assumptions are being made in this algorithm . Let there be an –

Text[ $n$ ] – Searchable text

Pattern[ $m$ ] – The subtext to be matched

$m$  = Length of the pattern

$n$  = Length of the searchable text ,

Hit[ $n$ ] = Array which stores the index number of the characters in the searchable text which matches with the first character of the pattern (i.e. Pattern[1]) .



# International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2015

## STEP 1 (Precomputation) :

```
int t = 0 ;
For(i=0 ; i<n ; i++)
{
    If(Pattern[0] == Text[i]&& Pattern[m-1] == Text[i+m-1]&& Pattern[m*0.5] == Text[(i+m-1)*0.5])
    {
        While(t < n)
        {
            Hit[t] = i ;
            t++ ;
        }
    }
    If(Hit[t]+m < n)
    Break ;
}
if(t==0)
Pattern does not exist in the Searchable string
```

## STEP 2 (Matching) :

```
For(i=0 ; i<t ; t++)
{
    For(u = m-2 ; u > 0 ; u--)
    {
        if(Text[Hit[t] + u] == Pattern[u])
        match ;
    }
}
```

We can see that in the pre computation part we have to do maximum 'n' number of comparisons , thus it gives a complexity of  $O(n)$  .Also in the precomputation part only we are checking with first,last and the middle character of the pattern , thereby determining that only which part in the searchable string/Text we need to revisit during the matching time .This reduces the number of pattern with which we are going to match This herby reduces the complexity of the matching algorithm as we only revisit the pre-computed parts in the given searchable string. So it reduces to  $O(km)$ , where k is some constant .

Also in our algorithm we can keep track of the number of the comparisons that has to be made in the matching part of the algorithm. It will always be equal to the number of hit that occurs.

# International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2015

## IV. ILLUSTRATION WITH COMPARATIVE STUDY

**Example 1 :**

Searchable string/Text –

a	b	b	d	a	c	a	a	b	c
i=0	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8	i=9

Pattern –

a	b	c
i=0	i=1	i=2

Searchable string length (n) = 10

Pattern length (m) = 3

Hit[n] = [7]

From the above mentioned algorithm we can clearly say that after precomputation the Hit[n] have only one value since the 1<sup>st</sup>, middle and the last of the pattern matched with only a single instance in the Searchable String [i.e. when Text[7] == Pattern[1] && Text[7+3-1 = 9] == Pattern[3-1 = 2] ]. Now the control goes to the matching algorithm and it checks only once with [Text[8] == pattern[1]], which is true in this case, so it generates the result “Match”

Here since In preprocessing total comparisons is 10 (i.e. O(n))

In matching total comparisons is 1 (i.e. O(1\*m) = O(m))

Since in this case there is no longest substring so KMP algorithm works as a naïve algorithm here, thus giving a complexity of preprocessing = 0, matching = O((n-m+1)m), which is much more than the complexity given by SA algorithm. Thus we can say SA algorithm performs better.

**Example 2 :**

Searchable string/Text –

a	b	b	d	a	c	a	a	b	c
i=0	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8	i=9

Pattern –

x	y	z
i=0	i=1	i=2

Searchable string length (n) = 10

Pattern length (m) = 3

Hit[n] = 0(no hit)

In this example we can see that none of the characters present in the pattern match with any character of the given searchable string. Thus there will be no Hit. From there only we can say that it does not match. So time complexity for this particular example is O(n), cause it does not require the matching part of the algorithm, it can generate result from only the precomputation part.

For this case KMP does not generate any result as in KMP algorithm there is no provision for this case where the string is not present at all. But still while working since this example does not have any longest substring so it behaves as naïve. So here also we can see that SA algorithm performs much better than KMP.



ISSN(Online): 2320-9801  
ISSN (Print): 2320-9798

# International Journal of Innovative Research in Computer and Communication Engineering

*(An ISO 3297: 2007 Certified Organization)*

**Vol. 3, Issue 11, November 2015**

## REFERENCES

- [1] Thomas H. Cormen , Charles E. Leiserson , Ronald L.Rivest , Clifford Stein, "Introduction to Algorithms" , Thrid edition , PHI publication.
- [2] International journal of advanced research in Computer Scienceand Software engineering , Rabin-Karp Algorithm with hashing a string matchingtool.Vol4,Issue3,Ms Sunita,MsRitu Malik,Ms Mamta Gulia
- [3]<http://www.iitg.ac.in/rinkulu/algorithms/slides/str-dfa.pdf>

## BIOGRAPHY

Shibdas Bhattacharya is a Lecturer in the Computer Science & Technology Department, Technique Polytechnic Institute, West Bengal , India. He Received Bachelor of Technology (B.Tech) in Computer Science & Engineering Degree in 2012 from Jis, West Bengal, India. His Research Interests are Algorithms, Theory Of Computing.

Aratrikasahais a 3rd Year Student in the Computer Science & Engineering Department, Kalyani Govt. Engineering College, West Bengal , India. She Received Diplomain Computer Science & Engineering Degree in 2014 from Women's Polytechnic Chandannagar, West Bengal, India. Her Research Interests are Algorithms, Image Processing.