ISSN: 0975-766X
CODEN: IJPTFI
Available Online through *Research Article*
www.ijptonline.com

PATTERN MATCHING USING DATA PREPROCESSING WITH THE HELP OF ONE TIME LOOK INDEXING METHOD

S.Hrushikesava Raju*, Dr. M.Nagabhusana Rao
Professor, Department of CSE, SIETK, NarayanaVanam Road, Puttur, A.P.
Professor, Dept.of CSE, KL University, Vijayawada, A.P.
Email: hkesavaraju@gmail.com

Received on 09-08-2016 *Accepted on 05-09-2016*

Abstract

There are various pattern matching algorithms which take more comparisons in finding a given pattern in the text and are static and restrictive. In order to search pattern or substring of a pattern in the text with less number of comparisons, a general data mining technique is used called data preprocessing which named as D-PM using DP with help of one time look indexing method. The D-PM using DP finds given pattern or substring of given pattern in the text in less time and the time complexity involved is less than existing pattern matching algorithms. The new Pattern Matching Algorithm with data preprocessing (D-PM using DP) proposes Pattern Matching with dynamic search behavior and makes users should have flexibility in searching.

Keywords: Pattern Matching, Data Preprocessing(DP), Time Complexity, Comparisons, Onetime look Indexing.

I. Introduction

Pattern Matching is considered important now days because the operations such as searching for interested patterns in a huge text, in encrypting certain patterns when they are replaced with some unknown data in maintaining secrecy are required. There are variety of applications require these type of operations. These operations are required in defense in order to maintain secrecy, and in searching for some wanted data in a corporate data bases. Pattern Matching means finding whether a pattern is substring of given text or not. If the pattern was found in given text, output just the index of the pattern. Otherwise, display -1 denoting that pattern is not a substring of the text. In Pattern Matching, pattern is preprocessed before searching in the given text. Here, Σ is alphabet consisting of letters used to form pattern and text and $|\Sigma|$ is alphabet size. For some Pattern Matching Algorithms, $|\Sigma|$ is infinite and for other Pattern Matching Algorithms, $|\Sigma|$ is finite. There are various algorithms proposed to find a pattern in given text. All proposed Pattern Matching algorithms are somehow expensive in terms of comparisons. In order to reduce time complexity, a

novel method called D-PM using DP with heuristic one time look indexing is proposed. The first D stands for Dynamic, PM stand for Pattern Matching and DP is a technique called Data Preprocessing with help of one time look indexing method. The advantage of using Data Preprocessing is a technique that removes unnecessary characters such as several spaces instead of a single space and makes it as single space, rectification of unbalancing of symbols in a sentence etc. The dynamism of Pattern Matching is achieved by comparing pattern from its size more than half of its size to the end of pattern. The technique or algorithm used is D-PM using DP with help of one time look indexing works by taking input as an array that consisting indexes of substrings of given pattern. This technique now compares the given pattern in the text from only those indexes which was returned by DP with help of one time look indexing heuristic.

II. Related Work

Based on [1,2,4], there are few pattern Matching Algorithms such as (i) Brute force or linear Pattern matching which compares pattern with every occurrence of text. (ii) Boyer Moore compares pattern with the text using two heuristics such as looking glass and character jump and this somehow reduced the comparison time but still involves overhead.(iii)

KMP is performed using failure function concept and uses knowledge of previously performed comparisons. Although it has less comparisons but still involves overhead. The following lists the difficulties of Pattern Matching Algorithms in which n denote Text size and m denote Pattern size.

Based on [12,13,14], Robin Karp String Pattern Matching Method is proposed by Michael O. Rabin and Richard M. Karp in 1987. The advantage is it can search multiple patterns in the given text using hash function. This work suffers with many issues such as first is more than one pattern having same size can have same hash value and are considered as spurious hits other than exact match, and second is it works only on decimal digits by converting the alphabets of text and pattern into equivalent digits, and third is its time complexity became worst when the pattern is of large size and its average complexity is O(n+m) and worst complexity is O((n+m-1)m).

Based on [3], Although Michael Burrows and David Wheeler produced efficient code in determining pattern matching in a text in 1994, this approach also called block sorting having two drawbacks such as first is it permutes or rotates a block of text at a time, and second is text can be sorted. The time it takes also expensive because it rotates a block of text, then sorts each rotated block, output the pattern by taking only last column from those sorted blocks. The original text can be obtained using inverse BWT approach which is also expensive. The advantage of this

approach is although block is transformed, the characters don't change value and it compress the text using BZIP2, and GZIP2.

Based on [5,6,7], Although it is easy to search a single character or group in a text using regular expressions which was introduced by Regex Buddy, it suffers with many disadvantages such as first is more effort is required to learn syntax of regular expressions and this varies with the type of language that are using and second is Regex class causes memory overhead at some situations.

Table-I: Drawbacks of Pattern Matching Algorithms.

PM Algorithm	**Computation**	**Time Complexity**
Brute Force	Comparing with every Text index	O(n*m)
Boyer Moore	Last Occurrence table	O(|E|+n*m)
KMP	Failure function	O(n+m)
Rabin Karp Method	Converting pattern and text into decimals	More time complexity depends on the context
BWT Method	Rotating the block of text	
Regular Expressions	Effort to learn the style for each new language	

To avoid this overhead, the Data Mining technique called DATA PREPROCESSIG is used and it searches the pattern in the text from only specified indexes which are in an array returned by one time look indexing method (heuristic) of Data Preprocessing.

III. Proposed Work

The proposed technique is Dynamic Pattern Matching using DP with help of One time look Indexing causes producing results fastly and also results substrings of given pattern whose size more than half of the pattern. The advantage of producing substrings indexes is the substrings can be replaced with some other strings at sender and are substituted at receiver by using those indices. This prevent any third party to know the original text. Yet safety is provided by sending the indices in ASCII. This also makes unknown person in understanding what those values indicate. Yet much more safety is provided by sending octal values of indices ASCII values. So, the receiver who aware of the mechanism, they follow and can only interpret the index array and can allowed to understand the

original text. In proposed, first Data Preprocessing (DP) is applied which removes several white spaces at a place and replace with single space, and also checks the special operators like parenthesis, curely braces and Square Brackets are balanced, if not balanced, it asks the user to enter text with correct balanced symbols. Second, The output of DP is now processed with one time look indexing method which returns separate index array for each substring of the pattern using indexOf() in String class. Later, Dynamic Pattern Matching is Applied.

i) First Step: It uses DP which is a Data Mining technique used to clean the given data. This Data Preprocessing contain many methods in order to get quality data. The methods it have are Data Cleaning which removes noise, Data Integration which merges different files in to one, Data Transformation which produces given data in to required format, and Data reduction which produce given data in reduced size. All these methods are not mandatory and are required based on quality of given data. For our problem, the data cleaning is required which removes noise such as several spaces rather than a single space is enough, validate the balanced symbols in given text. The following is pseudo code which does data Cleaning:

\Procedure DP_Dataclean(String text)

1. read text
2. compare every character with Space,

2.1 count the characters when adjacent characters read are spaces.

2.2 if count>=2, keep a space instead of several spaces.

2.3 if read character is opening symbol, push it into stack.

2.4 If any later read is closing symbol, pop the stack till its matching opening symbol is encountered.

2.5 if end of text is reached and stack found not empty, display unbalanced text is read.

2.6 At end of text is reached and stack found empty, display text entered is valid and is balanced symbols text.

3. output the updated text (clean text).

The code used to work as functionality of DP_Dataclean(String text) is as follows:

```
String DP_DataClean(String text )
{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
int sc=0; // counts spaces count
```

```
int j=0; // new index for cleaned text

String tac; // text after clean string

int s[10];   // stack using array to store symbols

    text=br.readLine();   // text in which pattern to be find

    Stack st;

    for(int i=0;i<text.length;i++)

    {

    if(text[i]==' ')

    {

    ++sc;

    }

    else if(text[i]=='(')

    {

    st.push(text[i]);

    tac[j]=text[i];

    ++j;

    continue;

    }

    else if(text[i]==')')

    {

    tac[j]=text[i];

    ++j;

    char ch=st.pop();

    if(text[i]==ch)

      continue; // stops current loop and executes next iteration

    }

    else

    {
```

```
if(sc>=2)
{
tac[j]=' ';
++j;
continue;
}
tac[j]=text[i];
++j;
} // else closed
} // for loop i  closed
return tac;
} // method DP_DataClean(String text) closed
```

ii) Second: After the text is cleaned, index arrays are returned for the substrings of pattern whose sizes are more than half of the pattern. This step uses one time look index method which in turn involves indexOf(String subp). This One time look indexing method returns index array separately for each substring of the pattern. The following is the pseudo code that returns array of indexes.

Procedure DP_Onetimelookindexing(String tac):

1. read the pattern and declare two dimensional array and assume array name dsubstringind[10][10].
2. take the sub string of pattern whose size exactly one more than half size of pattern initially.
3. compare this substring with tac string

3.1 if substring is in any portion in the tac, store its index in dsubstringind[0].

3.2 if substring is not anywhere in tac, return -1 which indicate substring is not in tac.

3.3 if the same substring occurs more than once, their indexes are stored in dsubstringind[1 to no. of times occurred].

4. repeat step2 for one character more size than existing substring and increment first dimension index by 1 for each separate substring of pattern until last but one character of pattern is encountered.

The code used to function same as DP_One timelookindexing(String tac) is as follows:

```
int[][] DP_Onetimelookindexing(String tac)
```

```
{
String s; // used to store substring of pattern
String pattern; // which can be searched in text
String orig;
int k=(pattern.length/2);  // point to next index of pattern middle size
if(k%2==0)
k=k/2;   // tac size is even
else
k=k/2+1;  // tac size is odd
int j=0; // j used as dsubstringind[][] first dimension
for(int i=k;i<pattern.length-1;i++)
{
int k=0; // used to locate rest of text portion after substring is found in some portion of text as a pattern
int z=0; // starting index for each separate substring
s=pattern.substring(0,i+1); // s is a substring of text tac
temp=tac;
while(s[s.length()-1]!=pattern[pattern.length()-1])
{
int p=temp.indexOf(s); // returns first occurrence of s in tac
if(p==-1) // s is not a substring of text
{
j++;
break;
}
dsubstringind[j][z]=p;
z++;
if(j==0)   // first dimension is zero  {
temp=tac.substring(p+s.length()-1,tac.length()+1);
```

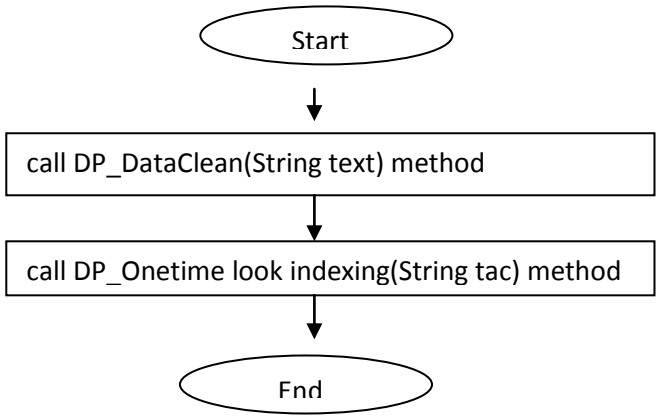
```
}
else if(j>0)
{
if(k==0)
{
temp=tac.substring(0,tac.length());
++k;   }
else
{
temp=tac.substring(dsubstrinind[j][k],tac.length());
++k;
if(dsubstringind[j][k]==’\0’) { k=0; break; }
}
} else if (j >0) is closed
} // while closed

} // for loop i closed
return dsubstringind;
} // close DP_onetimelookindexing
```

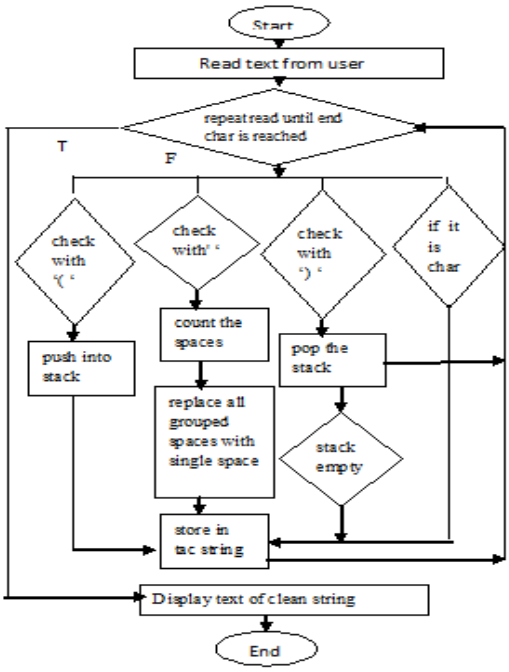
In proposed work, First step produce text in which no noise exists such as several spaces at a time, and no unbalanced symbols. This First step uses data cleaning method of Data Preprocessing. Second step produce index array for each substring of the pattern and that array contain indexes of matching each substring of pattern separately in first dimension. This second step is performed using indexOf(substring of pattern) over a text and that achieves the functionality of One time look Indexing. One time look Indexing method is a method that first searches sub string of a given pattern in the given text whose size is one more than half of the pattern size and if there are any matches found for that substring in the text, those matched starting indexes are stored in one unique first dimension whose index initially is zero. Secondly, next substring of pattern is checked in the given text whose size is one more than first substring size.

If any matches are found for second substring, those matched starting indexes are stored in another unique dimension whose value is 1. This procedure is repeated by picking the substrings from pattern and search continue until substring of pattern whose size is one less than pattern size. The advantage of using second step is the substring of pattern whose size is one less than pattern size are stored in last index of dsubstringind[] first dimension. Using only that last index array, pattern is easily found in the text and produces the index array that contain indexes of matching pattern in the text. The benefit of having other index arrays is user can know indexes of substrings of text and also sender can replace a particular substring with unknown pattern which the third party can't identify. The text can be understood only by the authorized receiver by interpreting index as ASCII value or Octal Value, and size of substring pattern.

The flow chart for DP with one time look indexing method is constructed using DP_Data Clean(String text) and DP_onetime look indexing(String tac) and is pictured as follows:



FLOWCHART I: DP using Onetime look Indexing Method.

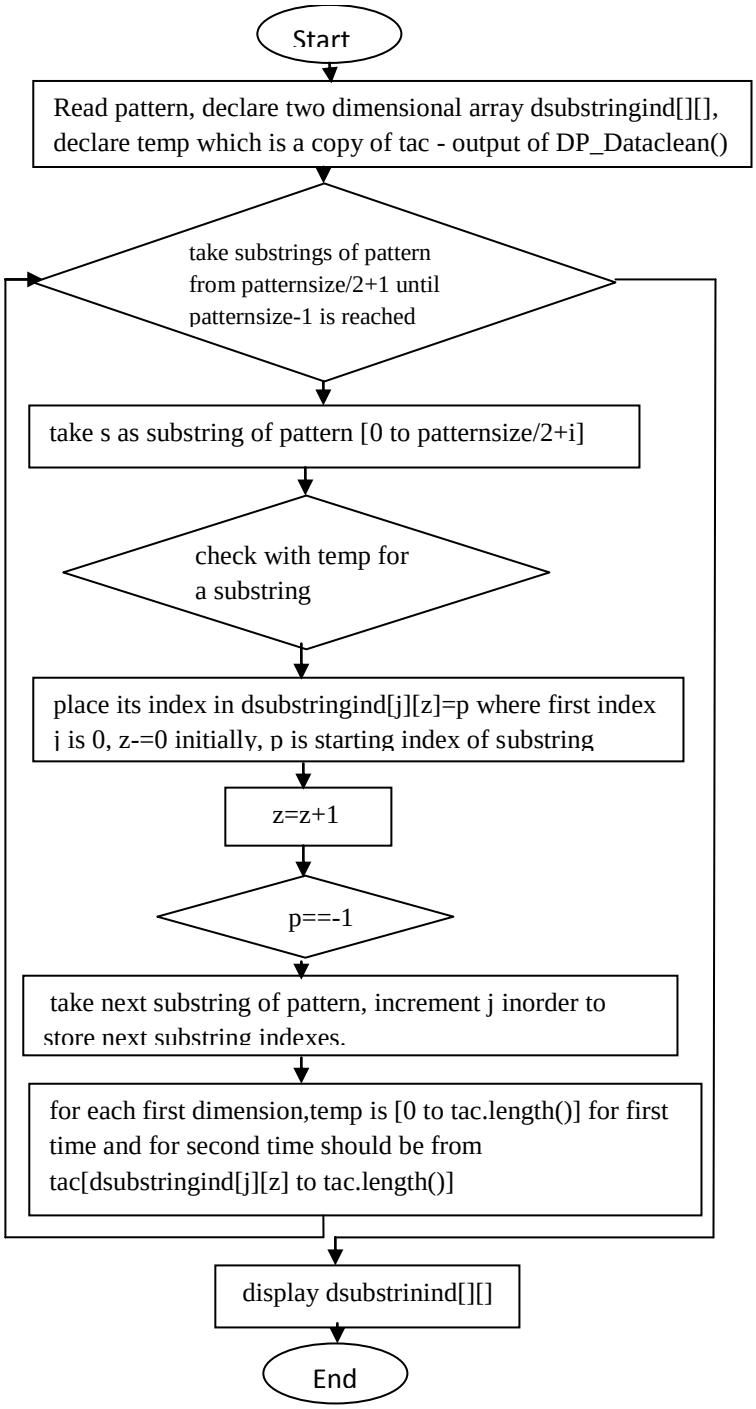


Flowchart II: DP_DataClean(String text) Method.

DP_DataClean(String text) of Flowchart I is depicted in Flowchart II: The second step also specified in Flowchart III which includes steps like reading pattern, reading two dimensional array, repeating loop from substring of pattern of size initially one more than half of pattern size till last but one character of pattern of reached, storing indexes of each separate substring in first dimension of two dimensional array dsubstringind[][].

The pictorial diagram is as follows:

Flowchart III: DP_OnetimelookIndexing(String



Dynamic Pattern Matching: It start search the pattern from only indices that exist in last value of first dimension of two dimensional array dsubstringind[][]. The array avoids comparing pattern again with text from scratch when a mismatch occurs and it uses knowledge of previously performed comparisons that exists in the dsubstringind[][]. The

name dynamic is given because of the reason that searching doesn't begin from starting of the text but from indices in first dimension last value only. If the pattern matched in many places in the text, the starting indexes of all those matched regions in the text.

The following is the pseudo procedure of D-PM(String pattern, String tac, int dsubstringind[][]):

Procedure D-PM(String pattern, String tac,int dsubstringind[][])

1. take dsubstringind[][] first dimension last value.

2. find first dimension last value size i.e dsubstrinind[lastvalue].length

3. compare every index in first dimension plus pattern size -1 location element with pattern last element

3.1 if it is a match, assign dsubstringind[lastvalue][i] to int array index[] where I from first index to last index.

3.2 if not match, return -1

3.3 repeat step3 until last index in first dimension last value is encountered.

4. display index[]

The time consumed to do this type of pattern Matching is O(1) for each time pattern was found in text.

The pseudo code that accomplish job of D-PM is as follows:

```
int[] D-PM(String pattern, String tac, int[][] dsustrinind)
{
int r=dsubstringind.length; // to know first dimension
last value
int c=dsubstrinind[r-1].length;
if(c==0)
 return -1;
int j=0,k=0;  // j is dsubstringind[][] second dimension and k is index array variable index
int len=pattern.length();
int index[]=new int[c];  // index array contains only matched indexes in text
while( j< c)
{
if(tac[dsubstringind[r-1][j]+len-1]!=pattern[len-1])
{
++j;
```

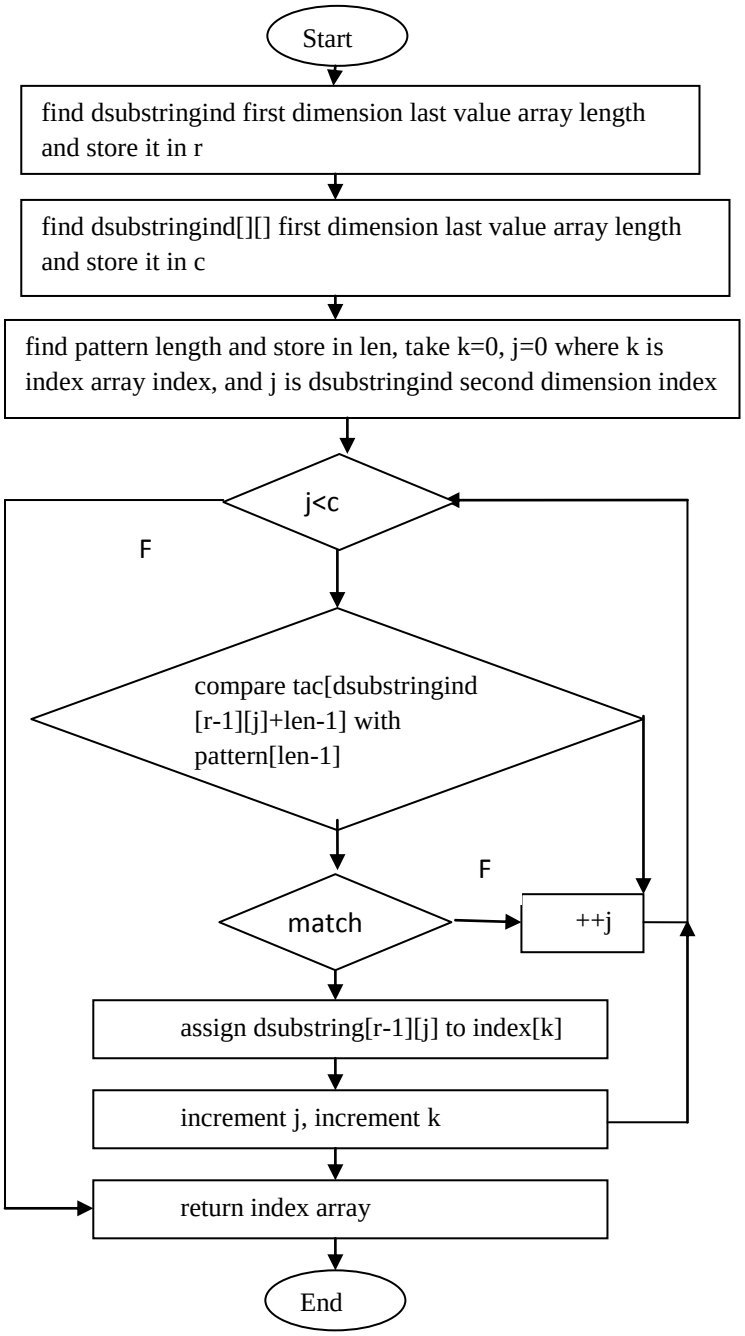
```
continue;  // stop executing rest of instructions and next loop executes

}
else
{
index[k]=dsubstringind[r-1][j];

++j;
++k;
}
} // while closed
return index;   // matched indices of the pattern in the text are returned as output

} // close D-PM method
```

The work of D-PM is depicted in and is as follows:



FLOWCHART IV: D- PM Flowchart

IV. Results with Examples

In this, different types of text can be assumed and a pattern is read. Then, examine the number of comparisons that different pattern Matching algorithms took. Also, examine the proposed D-PM incurred comparisons.

Type I: The text contains characters in normal way in which pattern appears in one or more places.

Assume text having pattern in only one place and

text= abacaabaccabacabaabb, and pattern read is abacab. The following table tells the number of comparisons that the existing different Pattern Matching algorithms consist:

Type II: Text has partial patterns at many places and has pattern at only two or three places.

Pattern Matching Algorithm	**No. of comparisons**
Brute force	28
Boyer Moore	13
KMP	19
Robin Karp	20
D-PM	1

Text= abacaabacababacababac , and pattern read is abacab. The following table tells the number of comparisons that the existing different Pattern Matching algorithms consist:

Pattern Matching Algorithm	**No. of comparisons for first time**	**No. of comparisons for Second time**
Brute force	17	Unable to process for second time
Boyer Moore	12	
KMP	13	
Robin Karp	21	21
D-PM	1	1

Type III: Text have characters along with several places at some portions and have balanced symbols.

Assume text=”hello how are you. What are you doing(reply) “ and pattern= “you”. The following table tells the number of comparisons that the existing different Pattern Matching algorithms consist:

Pattern Matching Algorithm	**No. of comparisons for first time**	**No. of comparisons for Second time**

Brute force	21	Unable to process for second time
Boyer Moore	10	
KMP	20	
Robin Karp	21	21
D-PM	1	1

The Brute Force, Boyer Moore, KMP are static and are restricted to search for a pattern in text once and failed to search the same pattern again for further time. The Robin Karp takes given text and pattern in decimal digits and perform comparisons using modular Arithmetic and find multiple patterns in the text. Although it is efficient using Horners Rule, It won't return starting index of pattern in less time.

Type IV: Text have several spaces at some portions and have symbols which are unbalanced.

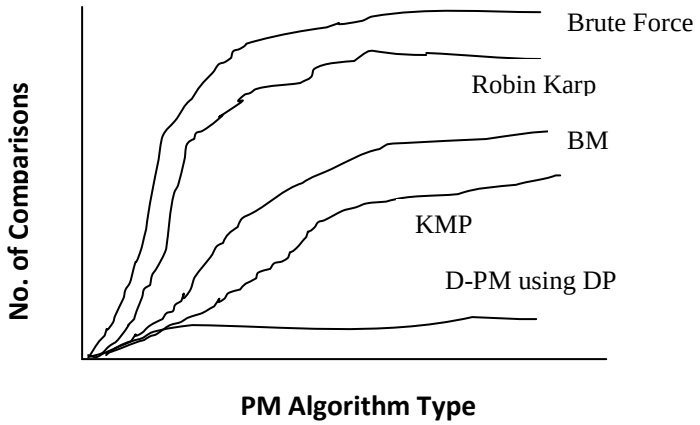
Assume text="hello how are you. what are you doing (dude (reply)" and pattern = "you".

The following table tells the number of comparisons that the existing different Pattern Matching algorithms consist:

PM Algorithm Type	**No. of comparisons**	**difficulty / advantage**
Brute force	21	Unable to preprocess text
BM	10	
KMP	20	
Robin Karp	21	Search multiple patterns and it search in all the text.
D-PM	1	Asks the user with balanced symbols text. Then, Preprocess the given text and dynamically finds pattern.

By observing several examples, Dynamic Pattern Matching (D-PM) using DP with the help of One time look Indexing method always take O(1) time for each time pattern is found in the cleaned text of given text.

The following graph illustrate the efficiency of the D-PM using DP with the help of One time Look Indexing method when compared with others. Sometimes, the efficiency of Pattern Matching algorithms look more or less depending on pattern and text types. Most of cases, the following efficiency works.



Graph I: Efficiency of PM algorithms over No. of Comparisons.

V. Conclusion

The proposed D-PM using DP with the help of One time look Indexing search the pattern at different portions in the text dynamically using output of DP_Onetimelook Indexing method such as dsubstringind[][] first dimension last value array. The D-PM start searching the pattern in text from only the indexes exists in dsubstringind [lastvalue] array and this avoids unnecessary comparisons between pattern and text and produces starting indexes of the pattern that matched in the text in less no. of comparisons. The data Mining technique called Data preprocessing removes any noise exists in given text and applies Onetime look Indexing method on cleaned text. This also supports multiple patterns to search in a text but D-PM is called every time new pattern to search in the text. The future extension may take all the patterns at a time which are required to search and return all patterns starting indexes although they occurred in text multiple times.

References:

1. Data Structures and Algorithms in java, Michael T.Good Rich and Roberto Tamassia.
2. Data Structures and Algorithms using C++ by Akepogu Ananda Rao.
3. The Burrows Wheeler Transform by Donald Adjeroh, Timothy Bell and Amar Mukharjee.
4. Data Structures and Algorithms using Visual Basic.NET by Machael McMillan.
5. Introduction to String Matching and modification in R using Regular expressions, Svetlana, Eden,march,2007.
6. Mastering Regular Expression by Jeffrey.E.F.Fredl, 3rd Edition by O,reilly publications.
7. Regular expressions and Matching in Modern Perl 2011-12 edition.
8. A FAST Pattern Matching Algorithm by S. S. Sheik,Sumit K. Aggarwal,Anindya Poddar, N. Balakrishnan,and K. Sekar, J. Chem. Inf. Comput. Sci. 2004, 44, 1251-1256.
9. Data Mining Concepts and Techniques,Second Edtion, Micheline Kamber and Jiawei Han.

10. Data preparation for Data Mining by Dorian Pyle.

11. Introduction to Data Mining by Pang-Ning Tan, Vipin Kumar, Michael Steenbach.

12. people.cedarville.edu/Employee/...web/.../rabin_karp_matching.ppt

13. csnotes.upm.edu.my/...nsf/.../StringMatching%20-%20Part%201.ppt

14. cs.smith.edu/~streinu/Teaching/Courses/252/...af/CSC_252.PPT

15. gauss.ececs.uc.edu/Courses/c472/lectures/.../21PatternMatching.pdf, pp.No 1- 15 and pp.no 26-pp.no-36.

16. www.cosc.canterbury.ac.nz /tad.takaoka /cosc229 / patgeo.pdf, pp.no 48 – pp.no 54.

Corresponding Author:

S.Hrushikesava Raju*,

Email: *hkesavaraju@gmail.com*