IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-5, Issue-2 E-ISSN: 2347-2693

PMT_EDS: Pattern Matching as a Tool for Efficient and Dynamic Search in the Large Files

Hrushikesava Raju S.[1*] and Nagabhushana Rao M.[2]

[1*]PP.CSE.0158, Rayalaseema University, Kurnool (A.P.), India
[2]Department of CSE, K L University, Vijayawada (A.P.), India

[*]*Corresponding Author: hkesavaraju@gmail.com*

Available online at: www.ijcseonline.org

Received: 29/Jan/2017 Revised: 04/Feb/2017 Accepted: 17/Feb/2017 Published: 28/Feb/2017

***Abstract*— There exist many pattern matching approaches, which consume more time and unable to perform operations like recording history, finding number of times a pattern is found along with positions, page numbers etc. and they have limitations in performing operations beyond their usual operations. The pattern matching using indexOf method is proposed to find out a specific pattern or multiple patterns at a time in less time complexity. The additional information is reported by recording history operation, information of where the pattern is located like page number, number of times that pattern is found can be processed by searching operation, and multi-process operation searches multiple patterns and returns their locations, page numbers in less time complexity by using indexOf metod as a thread in achieving better efficiency. To do all these operations, an automated tool is required that asks for operation to perform, required details to be provided in that operation, and results going to be illustrated or reported. Data pre-processing is required when there is any inconsistency present in the dataset.**

Keywords- pattern matching, statistics, searching, multi-process, time complexity, tool, PMT_EDS

I. INTRODUCTION

There were many pattern matching approaches are there for searching a pattern in the small to moderate texts. But there were no tool to search a pattern or multiple different patterns in the huge sized text document or a pdf type document. There are certain pattern matching approaches that are used in the huge texts but they consume more time in the searching process. All these are application programs that ask the input such as a pattern or few patterns, and also a text. In this processing, indices indicating starting index of the pattern in the text are returned as an array. These approaches consume more time in finding position of the pattern. These approaches are specified in the [1,2,3]. But, the present trend expecting new developed apps or apps as tools for this pattern matching. Every approach specified in the [1,2,3]are having drawbacks and one approach used as a proposed methodology namely dynamic pattern matching using indexof() method and data preprocessing. Compared to this, the many pattern matching approaches illustrated takes more time and that can be demonstrated in the chapter results column. The time is measured in terms of number of comparisons and is discussed through examples. Among methods used, the overheads are listed in a table in Introduction chapter. The lagging behind the existing approaches are (i) They are supporting only limited texts and one or few patterns to process (ii) When they are processing, manual text only going to accepted but not large sized files (iii) The time taken to search the pattern(s) is more compared to the tool taken in consideration (iv) All the existing approaches are standalone applications and that requires setup the environment to execute the application successfully (v) The appearance of the output can be although clear but looking makes different feel compared to the output of the tool. Hence, the trend expects migration from standalone applications to the either apps or tools to be developed to do the same. The existing pattern matching approaches produce -1 in case of failure of the pattern in the text or position of the pattern in the text. But, the requirement needed now-a-days is number of times the pattern is occurred in the large text file, page wise statistics such as number of times the pattern occurred in each page along with line number, and history of the (n-1) sized pattern positions in the text document. To do this, some pattern matching is to be taken as a tool(PMT_EDS) which serves the expectations of the user. Hence, The pattern matching with indexof() method can be taken as a tool which finds the pattern or multiple patterns in less time generally O(1) time complexity in case of success or failure also.

II. PROPOSED STUDY

The proposed approach provided in [1],[2],[3] taken in to consideration, transform that into a tool with specific features to perform such as taking the document, taking a pattern or multiple patterns, displaying a report about

number of times those patterns are occurred in the document along with page wise statistics. The approach taken in to account is dynamic pattern matching using indexof() method. The description of transforming this methodology into a tool should be demonstrated in terms of operations such as file submission where either word document or a pdf file can be entered, searching where a file can be inspected for a pattern or multiple patterns given as input, results where the number of times the given pattern or multiple patterns are occurred, and statistics where page-wise pattern or patterns count can be displayed.

The tool PMT_EDS can be discussed as follows:

A) File Submission: Here, the file can be taken and enable for further operations.

Pseudo_procedure file_submission()

Step1: Create a button named upload

Step2: select a file from the system directory and click on open

Step3: The file can be saved and upload and take that file for further opearations

B) Searching: Here, the pattern(s) can be searched and indices are returned.

Pseudo_procedure searching(file,p[]):

Step1: Take the each pattern from p array using sub-script (index)

Step2: develop the logic for searching

pc=page count=1

lc=100=number of characters in the line

pl=lines in a page = 25

beg=0; // indicating the starting index of each line

end=100; // end index of each line

k=0; // Navigation of each pattern

k1=0; // page loop variable

index_i1,index_i2,index_i3,….,index_IN-1=0; // maintains the pagewise count

declare arrays pi1[20],pi2[20],pi3[20],. . . .,piN[20] // page-wise count of patterns

int indices[2000]; // maximum length of indices and also keeping track of pattern occurred final count

int twodimen[][]=new int[p.lenth][2000]; //two dimensional array where first dimension navigates from pattern1 to pattern and second dimension stores step-wise details of occurred indices of patterns

fp=fopen(file,"r")

if(fp!=EOF) {

for(pi=1;pi<=pc;pi++) //pages navigation,where pi is index to read the characters, pc is number of pages

{ line=read(fp,beg,end); // reading each page

for(pli=0;pli<=25;pli++) // where each line is verified in each page {

while(fp!='\n') {

indices[0]=indices[0].arraycopy(line.indexOf(p[0])); // storing p[0] occurred instances(subscripts) in indices[0] array of a page // first pattern

indices[1]= indices[1].arraycopy(line.indexOf(p[1])); // storing p[1] occurred instances in indices[1] array of a page // second pattern

indices[2]= indices[2].arraycopy(line.indexOf(p[2])); // storing p[2] occurred indices in indices[2] array of a page // third pattern

….

….

indices[N-1]=indices[N-1].arraycopy(line. indexOf(p[N-1])); // storing p[N-1] occurred indices in indices[2] array of a page // N-1 pattern

} // while counts the no. of times a pattern occurred in same line

index_i1+=indices[0].length(); // finding no. of entries for pattern1 in the array of a particular page

pi1[k1]=pi1.arraycopy(index_i1); // storing locations of the first pattern in the pi1 array

index_i2+=indices[1].length(); // finding no. of entries for pattern2 in the array of a particular page

pi2[k1]=pi2.arraycopy(index_i2); // storing locations of the second pattern in the pi2 array

index_i3+=indices[2].length(); // finding no. of entries for pattern3 in the array of a particular page

pi3[k1]=pi3.arraycopy(index_i3); // storing locations of the third pattern in the pi3 array

……

……

index_IN-1+=indices[N-1].length();//finding no. of entries for patterN-1 in the array of a particular page

piN-1[k1]=piN-1.arraycopy(index_IN-1); //storing locations of the last pattern in the piN-1 array

} // for iteration

k1++; // page loop variable to be incremented for further pages

twodimen[][]=twodimen.arraycopy(*pi1); // storing first pattern indices in the final array

// updating twodimensional array with page-wise pattern count

twodimen[][]=twodimen.arraycopy(*pi2); // storing second pattern indices in the final array

// updating twodimensional array with page-wise pattern count
twodimen[][]=twodimen.arraycopy(*pi3); // storing third pattern indices in the final array
// updating twodimensional array with page-wise pattern count
…
…
twodimen[][]=twodimen.arraycopy(*PiN-1); // storing last pattern indices in the final array
//updating twodimensional array with page-wise pattern count
pli=0; // for second page onwards, this index became 0 for each new page
beg+=100; // transferring into next lines
end+=100; // pointing page end line size
pc++; // tracking of number of pages by assuming 100 columns size and 100 lines in a page
} // end of the page count
} // this is repeated until there are no more characters to read from the file.

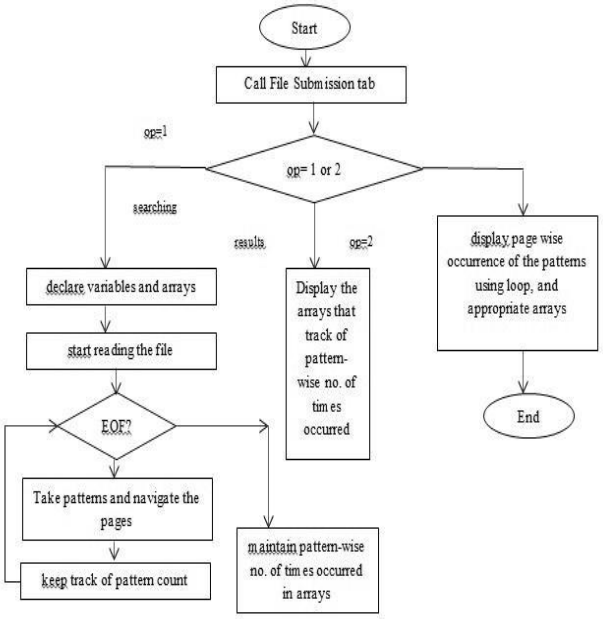
The above code involves navigating from first page to last page, in each page number of times the first pattern is found can be noted. Although it is noting, the final number of times the pattern occurred can be returned and displayed. The advantage of this process is all patterns given for searching can be identified simultaneously in when searching in each page. This searching module is crucial and is important in doing the remaining operations.

C) Results: In this, the number of times, each pattern is occurred can be displayed.
Pseudo_procedure PMT_EDS_results(twodimen[][])
Step1: take the loop (while) for first dimension index to last index i.e as long as condition is true
Step1.1: Display from first index dimension value to last dimensional value
Step 1.2: increment loop variable
The following is the pseudo procedure that evaluates the what the result to get.
Pseudo_procedure results_PMT_EDS(p[],indices[]):
i=0 // loop variable from first pattern to last pattern in the pattern array p[]
while(i<p.length) {
Display p[i] is occurred indices[i] times in the file
// use output function depending on the language used. For instance, assume c language, the function used is printf()
} // closing of while
This module displays the each pattern and their count.

D) Statistics: Here, pattern occurrence is displayed based on concerned page.
Pseudo_procedure PMT_EDS_statistics(pi1,pi2,pi3,. . . , piN-1):
Step1: take a loop called while with condition
Step2: Call arrays pi1,pi2,pi3, ... , piN-1, use the printing(output) method
This method displays pattern1 in each page-wise occurrence, pattern2 in each page-wise occurrence, and so on until patternN-1 in each page-wise occurrence.

The following is the pseudo procedure to display statistics such as each pattern page-wise occurrence details:

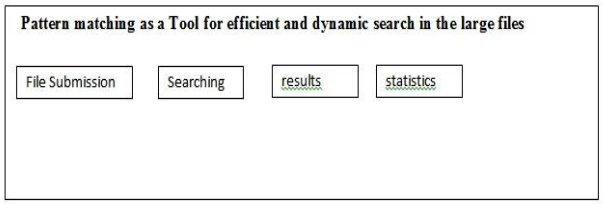
Pseudo_procedure statistics_PMT_EDS(pi1,pi2,pi3,…….,piN-1) where pi1,pi2,pi3,……..,piN-1 are patterns count:
Step1: Take arrays indices[0], indices[1], indices[3],…..,indices[N-1] as pattern occurred indices overally.
for(i=0;i<=N-1;i++) {
display indices[i] array elements
display each pattern occurred indices from pi[i]
} // iteration
This second phase involves developing a tool named PMT_EDS(Pattern Matching as a Tool for efficient and dynamic Searching for large files).
Now, the following flowchart displays the working of the tool called PMT_EDS:



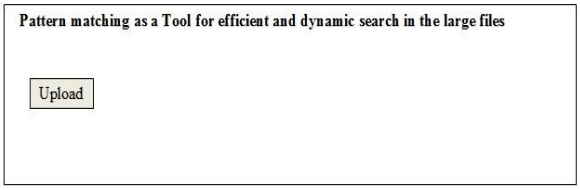
The above tool should be required to know about the pattern matching statistics in the given file uploaded. As of now, there are only few methods are used for knowing the indices of the patter(s). There is no approach available to know the statistics about the pattern(s) occurrence.

III. Results

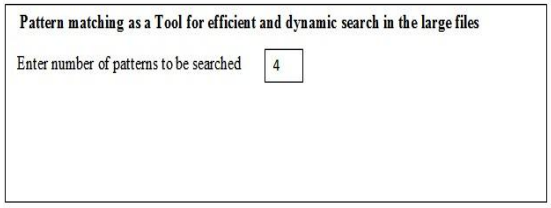
This tool appears in terms of screen shots where each screen shot denotes the functionality of the operation that is to be performed. The screen shots are File Submission tab in which any file can be uploaded with limitation in size, searching tab screen shot asks for pattern(s) to be find out, results tab screen shot displays the number of times, each pattern is occurred throughout the whole document, and statistics tab screen shot displays page-wise information about each pattern occurrence in the whole document.
The first screen looks like as follows:



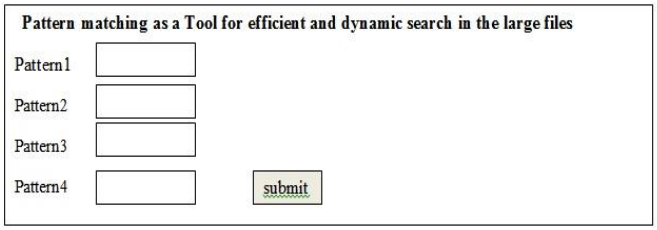
The second screen shot is uploading the file using File Submission tab:



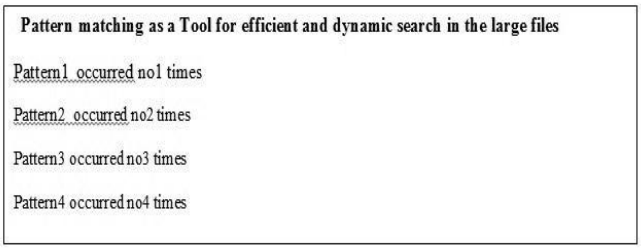
After the file is selected, file taken in one file pointer in read mode. The searching operation screen shot(third window) to be displayed as follows:



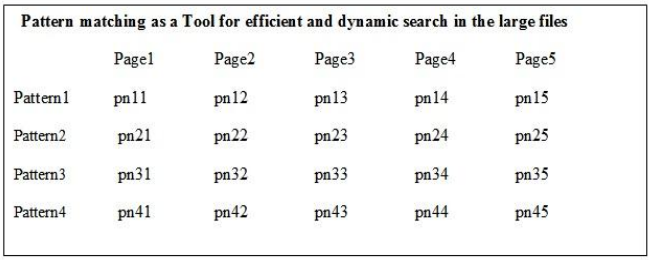
In searching, the screen shot(fourth) that asks for each pattern value so that they can be inspected.
It is as displayed as follows:



In searching, the screen shot (fifth) that displays each pattern the number of times they occurred throughout the entire document:

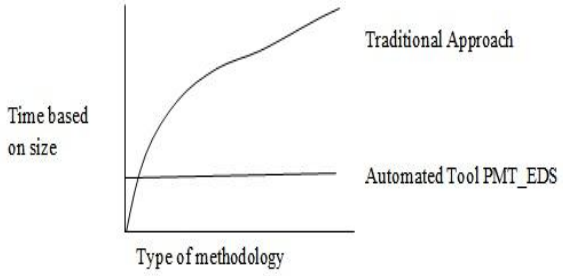


Assume there are only 5 pages. The Next operation to perform is statistics tab. The appearance of the statistics screen shot (sixth) is as follows:



The time to search each pattern in the large text file takes O(1) every time is O(1). It is the best time ever no other pattern matching approach is consuming. This tool can search all the patterns in less time so it is efficient. This tool also searching all the patterns in dynamically in the sense spontaneously.

The following shows time when the comparison taken between the tool and traditional methods based on size:



In above graph, The automated tool takes only O(1) for each pattern search in the large file.

IV. Conclusion

There were many drawbacks of using most of traditional methods for finding a pattern(s) in the text such as more time consuming in searching, lot of preprocessing is taken before searching, unable to process files, unable to simultaneously processing pattern(s). Hence, To overcome

all the pitfalls, the result considered is a tool which finds pattern(s) in less time, no preprocessing is required i.e. in single shot or stretch the pattern(s) bibliography can be known, support searching over files, and support processing multiple patterns at a time (simultaneous processing). All operations are included in the tool in order to process the pattern. Hence, pattern matching is made as a tool and is user friendly. The efficiency and dynamism is better compared to traditional approaches. In future, any extra operations than existing operations (searching, results, and statistics) that this tool going to be supported are provided.

REFERENCES:

[1]. Hrushikesava Raju S., Nagabhushana Rao M., "Improvement of Time Complexity on Pattern Matching using One -Time Look Indexing and Data Preprocessing", IJCSE, Vol.4(11),PP.100-106,2016 E-ISSN:2347-2693.

[2]. Hrushikesava Raju S. ,Swarna Latha T.,"Dynamic Pattern Matching: Efficient Pattern Matching using Data Preprocessing with help of One time look indexing method", IJARCET,Vol.2(2),pp.592-599, 2013,ISSN:2278-1323.

[3]. Hrushikesava Raju S., Nagabhushana Rao M.," "Pattern Matching Using Data Preproc-Essing With The Help Of One Time Look Indexing Method", IJPT, Vol.8(3),pp.14749-14756, ISSN:0975-766X.

[4]. Michael Good Rich T. and Roberto Tamassia, "Data Structures and Algorithms in java", Fifth Edition, January,2010.

[5]. Akepogu Ananda Rao and Radhika Raju polagiri, "Data Structures and Algorithms using C++", Kindle Edition,Pearson, July,2010.

[6]. Donald Adjeroh, Timothy Bell and Amar Mukharjee,"The Burrows Wheeler Transform", Springer, July,2008.

[7]. Machael McMillan,"Data Structures and Algorithms using Visual Basic.NET", Cambridge Edition, March,2005.

[8]. Svetlana, Eden, "Introduction to String Matching and modification in R using Regular expressions", March,2007.

[9]. Jeffrey.E.F.Fredl,"Mastering Regular Expression", 3rd Edition, O,reilly publications,December,1998.

[10]. Regular expressions and Matching in Modern Perl 2011-12 edition,ISBN-10: 1680500880,ISBN-13: 978-1680500882,October,2015.

[11]. S. S. Sheik,Sumit K. Aggarwal,Anindya Poddar, N. Balakrishnan,and K. Sekar ,"A FAST Pattern Matching Algorithm", J. Chem. Inf. Comput. Sci. 2004, 44, 1251-1256.

[12]. Micheline Kamber and Jiawei Han, "Data Mining Concepts and Techniques", Second Edition,March,2006.

Author's Profile

Mr. S. HrushiKesava Raju, working as a Professor in the Dept. of CSE, Siddharth Institute of Engineering and Technology(SIETK), Narayanavanam Road, Puttur. He is pursuing Ph.D from Rayalaseema University in the stream of CSE. His areas of interest are Data Mining, Data Structures, and Networks.

Dr. M.Nagabhushana Rao, working as Professor in the Dept. of CSE, K L University, Vijayawada, A.P. He had completed Ph.D from S.V. University in the area of Data mining. He is presently guiding many scholars in various disciplines.