Comparative Study on Text Pattern Matching for Heterogeneous System

Priya jain
MTech 3rd Sem Computer Science and Engineering
Rungta College Of Engineering & Technology
Bhilai, Chhattisgarh, INDIA
pj520330@gmail.com

Shikha Pandey
Asst. Professor (CSE)
Rungta College Of Engineering & Technology
Bhilai, Chhattisgarh, INDIA
Shikhamtech2008@gmail.com

Abstract— Pattern-matching has been routinely used in various computer applications, for example, in editors, retrieval of information either textual, image, or sound and searching nucleotide or amino acid sequence patterns in genome and protein sequence databases. Pattern-matching algorithm matches the pattern exactly or approximately within the text. This paper presents the comparative analysis of various multiple pattern Text matching algorithms. The highly efficient algorithms like Brute Force algorithm, Knuth Morris Pratt algorithm, Finite Auto Mata algorithm, Bayer Moore algorithm for exact and approximate multi-object and multi-pattern matching on heterogeneous systems. After performing a detailed study on the above mentioned algorithms, the best algorithm having least complexity is chosen. Consequently, the comparison result proves that Bayer Moore Pattern matching algorithm is the most efficient One to apply on heterogeneous system for pattern matching.

Keywords- Single pattern, Multiple pattern, Exact pattern, Matching, Pair, Sequence, Heterogeneous
Introduction (Heading 1)

I. INTRODUCTION

Pattern Matching is the act of checking some sequence of tokens for the presence of the constituents of some pattern. It is a process which takes input as a pattern[0..P-1] of length P and text[0..T-1] of length T, where P is generally very much smaller than T. An exact pattern-matching is finding all the occurrences of a particular pattern (x) x1x2... xm) of m-characters in a text (y) y1 y2 ... yn) of n-characters which are built over a finite set of characters denoted by Σ and the size of this set is equal to σ.

Pattern matching techniques has two categories:

- Single pattern matching technique
- Multiple pattern matching technique

In single pattern matching it is required to find all occurrences of the pattern in the given input text. And if more than one pattern is matched against the given input text simultaneously, then it is known as, multiple pattern matching. The pattern matching algorithms is widely used in network security environments. In network security, the patterns is a string indicating a network intrusion, attack, virus, and snort, spam or dirty network information, etc.

Pattern-matching algorithms scan the text with the help of a window, whose size is equal to the length of the pattern. The first step is to align the left ends of the window and the text and then compare the corresponding characters of the window and the pattern; this procedure is known as attempt. After a match or a mismatch of the pattern, the text window is shifted to the right. The number of characters required to shift the window on the text may vary according to various algorithms. This procedure is repeated until the right end of the window is within the right end of the text.

In this paper we will introduce pattern matching algorithms and the tools, which are used to further implement the tasks in real life. The main objective behind the pattern-matching algorithms is to reduce the total number of character comparisons between the pattern and the text to increase the overall efficiency. The improvement in the efficiency of a search can be achieved by altering the order in which the characters are compared at each attempt and by choosing a shift factor that permits the skipping of a predefined number of characters in the text after each attempt.

The more practical solutions to the real world problems can be generated by the Multiple String Pattern Matching Algorithms. String Matching Algorithms like Brute Force Algorithm, Knuth Morris Pratt Algorithm, Finite Automata Algorithm, Bayer Moore Algorithm etc. are focused in this paper. Each algorithm has certain

merits and demerits. This paper presents the comparative analysis of various multiple string pattern matching algorithms based on different parameters.

II. TEXT PATTERN MATCHING ALGORITHM

A. Brute Force Algorithm

In this algorithm perform linear search. The simplest algorithm for string matching is a brute force algorithm, simply try to match the first character of the pattern with the first character of the text, and if we succeed, try to match the second character, and so on; if hit a failure point, slide the pattern over one character and try again. When we find a match, return its starting location.

Steps:

Step 1:Align pattern at beginning of text

Step 2: Moving from left to right, compare each
Character of pattern to the corresponding
Character in text until

- All character are found to match
- A mismatch is detected

Step 3: While pattern is not found and the text is
Not yet exhausted, realign pattern one
Position to the right and repeat.

Algorithm: Brute force string Match (T[0..n-1], P{0..m-1])

```
//Implements brute force string matching.
// Input: An array T[0..,n-1] of n characters
   Representing a text.
// an array P [0.., m-1] of m characters representing a
  Pattern.
// Output: The index of the first character in the text
  That starts a matching substring or -1 if the
//search is unsuccessful
  for i ← 0 to n-m do
      j ← 0
  while j< m and p[j] =T[i+j] do
        j ← j+1
       if  j=m  return i
       return -1
```

Check each position in the text T to see if the pattern P starts in that position

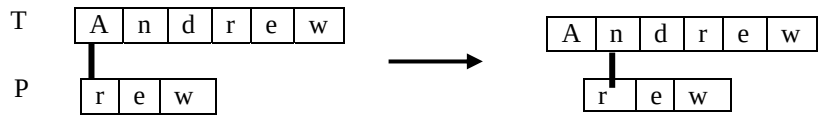


Figure :1. Brute Force Algorithm

Strength of Brute Force Algorithm is wide applicability, simplicity ,yields reasonable algorithms for some important problems (e.g. matrix multiplication, sorting, searching, string matching).and Weaknesses is rarely yields efficient algorithms ,some brute-force algorithms are unacceptably slow ,not as constructive as some other design techniques.

B. Knuth Morris Pratt Algorithm

The algorithm was conceived in 1974 by Donald Knuth and Vaughan Pratt, and independently by James H. Morris. The three published it jointly in 1977. Knuth, Morris and Pratt is a linear time algorithm for the string matching problem. Knuth-Morris-Pratt's algorithm compares the pattern to the text in left-to-right, but shifts the pattern more intelligently. The implementation of Knuth-Morris-Pratt algorithm is efficient because it minimizes

the total number of comparisons of the pattern against the input string.KMP algorithm first calculate PREFIX function and by using the output of prefix function the matching will be started.

Components of KMP algorithm :

The prefix function, Π :

The prefix function,Π for a pattern encapsulates knowledge about how the pattern matches against shifts of itself. This information can be used to avoid useless shifts of the pattern 'p'. In other words, this enables avoiding backtracking on the string 'S'.

Steps:

i. m ← length[p] //'p pattern to be matched
ii. Π[1] ← 0
iii. K ← 0
iv. for q ← 2 to m
v. do while k > 0 and p[k+1] != p[q]
vi. do k ← Π[k]
vii. if p[k+1]=p[q]
viii. then k ←k+1
ix. return Π

The KMP Matcher:

The KMP Matcher, with pattern 'p', string 'S' and prefix function 'Π' as input, finds the occurrence of 'p' in 'S' and returns the number of shifts of 'p' after which occurrence is found.

Steps:

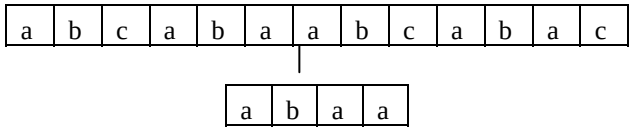
KMP-Matcher(S, p)

i. n ← length[S]
ii. m ← length[p]
iii. Π ← Compute-Prefix-Function(P)
iv. q ← 0 //number of characters matched
v. for i ← 1 to n //scan S from left to right
vi. do while q > 0 and p[q+1] != S[i]
vii. do q ← Π[q] //next character does not match
viii. if P[q+1] = T[i]
ix. then q ← q + 1 //next character matches
x. if q = m //is all of p matched?
xi. then print "Pattern occurs with shift" i – m
xii. q ← Π[q] // look for the next match

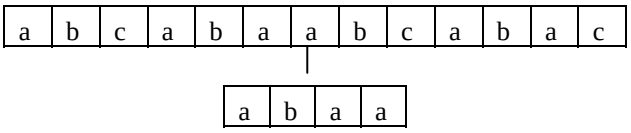
In the above algorithm T and P indicate TEXT and PATTERN. The total number of character present in the text and pattern will be assigned to n and m in step 1 and step 2 respectively. The application of PREFIX TABLE comes to picture in step 3 to select the values from that table. In step 4 q indicate the number of matching character, and the for loop started no 5 to step no 12 calculate that after how many characters the given Pattern is matched with the text. The calculation part will be done by using subtraction of the increment value of i (where q=m) , and value of m (number of character present in pattern).the function of step 12 is used to calculate how many times the pattern is present in a Text. Sometimes there may be more than one patterns present in a given text.

Example:

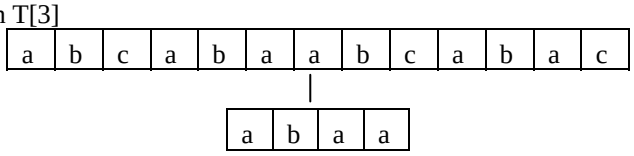
Step 1:



Step 2:



Step 3: compare P[3] with T[3]



Since mismatch is detected, shift 'P' one position to the left and perform steps analogous to those from step 1 to step 3. At position where mismatch is detected, shift 'P' one position to the right and repeat matching procedure.

Step 4:

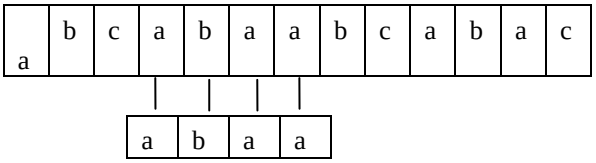


Figure:2. KMP algorithm

Finally, a match would be found after shifting 'P' three times to the right side.

*C. **Finite Automata algorithm***

An automaton with a set of states, and its "control" moves from state to state in response to external "inputs" is called a finite automaton. A finite state machine (FSM, also known as a deterministic finite automaton or DFA) is a way of representing a language (meaning a set of strings; we're interested in representing the set strings matching some pattern). It's explicitly algorithm, represent the language as the set of those strings accepted by some program.

Finite Automata:

A finite automata M is a 5-tuple(Q,q0,A,∑,δ)

- Q is a finite set of states,
- q0 € Q is the start state,
- ACQ is a distinguished set of accepting states,
- ∑ is a finite input alphabet,
- δ is a function from Q*∑ into Q , called the transition function of M.

The finite automata begins in state q0 and reads the character of its input string one at a time .if the automation is in state q and reads input character a, it moves ("make a transition") from state q to state£(q, a).whenever its current state q is a member of A, the machine M is said to have accepted the string read so far .An input that is not accepted is said to be rejected.

FINITE_AUTOMATION MATCHER (T,£,M)

i. n ← length[T]
ii. q ← 0
iii. for i ← 1 to n
iv. do q ← δ (q, T[i])
v. if q = m
vi. then print "Pattern occurs with shift" i-m

for line 4 that present in Finite automation matcher ,need the transition table or transition function.

COMPUTE_TRANSITION _FUNCTION (P,∑)

i. m← length [P]
ii. for q ← 0 to m
iii. do for each character a € ∑
iv. do k← min(m+1, q+2)
v. repeat k ← k-1
vi. until Pk ⊐ Pq0
vii. δ(q, a) ← k
viii. Return δ

Example:

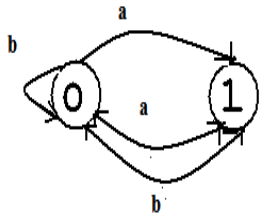


Figure:3. Finite - Automata

state	input	
	a	b
0	1	o
1	0	0

Table :1. Transition Table for Finite Automata

A simple two state finite automaton with state set Q={0,1},start state q0=0, and input alphabet ∑={a,b}.(a) A tabular representation of the transition function δ.(b) An equivalent state- transition representation of the transition diagram. State 1 is the only accepting state. Directed edges represent transition s. for example , the edge from state 1 to state 0 labeled b indicates δ(1,b)=0.

In Finite Automata algorithm a Text and a Pattern will be provided some times with a transition function and with a transition table. By using these table and function to calculate after how many character the given pattern matched with the text.

D. Boyer Moore Algorithm

Boyer-Moore algorithm was developed by Robert S. Boyer and J Strother Moore in 1977 .it is consider the most efficient string-matching algorithm in usual applications, for example, in text editors and commands substitutions. The reason is that it woks the fastest when the alphabet is moderately sized and the pattern is relatively long.

The algorithm scans the characters of the pattern from right to left beginning with the rightmost one. In case of a mismatch (or a complete match of the whole pattern) it uses two pre-computed functions to shift the window to the right. These two shift functions are called the *good-suffix shift* (also called matching shift and the *bad-character shift* (also called the occurrence shift).

Assume that a mismatch occurs between the character $x[i]=a$ of the pattern and the character $y[i+j]=b$ of the text during an attempt at position j.Then, $x[i+1……..m-1]=y[i+j+1…....j+m-1]$=u and $x[i] \neq y[i+j]$. The good-suffix shift consists in aligning the segment $y[i+j+1.... j+m-1]=x[i+1.... m-1]$ with its rightmost occurrence in x that is preceded by a character different from $x[i]$.

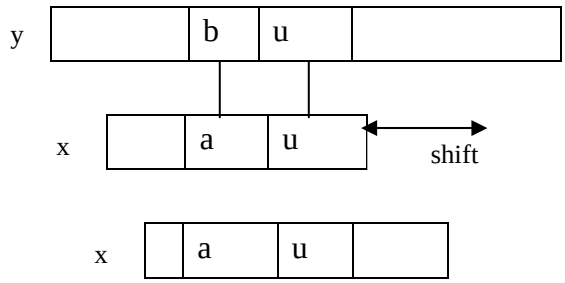


Figure .4. The good-suffix shift, *u* re-occurs preceded by a character *c* different from *a*.

The bad-character shift consists in aligning the text character $y[i+j]$ with its rightmost occurrence in $x[0 .. m-2]$.

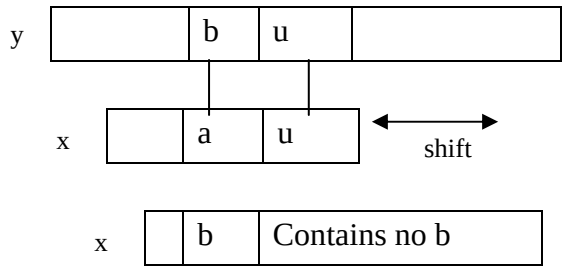


Figure.5. The bad-character shift, *a* occurs in *x*.

If $y[i+j]$ does not occur in the pattern x, no occurrence of x in y can include $y[i+j]$, and the left end of the window is aligned with the character immediately after $y[i+j]$, namely $y[i+j+1]$

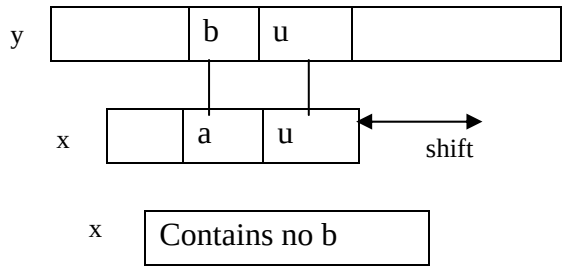


Figure.6. The bad-character shift, *b* does not occur in *x*.

Last Function:

Define a function last(c) that takes a character c from the alphabet and specifies how far may shift the pattern P if a character equal to c is found in the text that does not match the pattern

$$last(c) = \begin{cases} \text{Index of last occurrence} & \text{if c is in pattern p} \\ -1 & \text{other wise} \end{cases}$$

BOYER_MOORE_MATCHER (T, P)

Input: Text with n characters and Pattern with m characters

Output: Index of the first substring of T matching P

i. Compute function last
ii. i ← m-1
iii. j ← m-1
iv. Repeat
v. if P[j] = T[i] then
vi. if j=0 then
vii. return i // we have a match
viii. else
ix. i ← i -1
x. j ← j -1
xi. else
xii. i ← i + m - Min(j, 1 + last[T[i]])
xiii. j ← m -1
xiv. until i > n -1
xv. Return "no match"

The computation of the last function takes O(m+|∑|) time and actual search takes O(mn) time. Therefore the worst case running time of Boyer-Moore algorithm is O(nm + |∑|)

The Boyer-Moore algorithm uses two different heuristics for determining the maximum possible shift distance in case of a mismatch: the "bad character" and the "good suffix" heuristics. Both heuristics can lead to a shift distance of *m*. For the bad character heuristics this is the case, if the first comparison causes a mismatch and the corresponding text symbol does not occur in the pattern at all. For the good suffix heuristics this is the case, if only the first comparison was a match, but that symbol does not occur elsewhere in the pattern.

III. COMPARATIVE ANALYSIS OF SELECTED MLTIPLE PATTERN STRING MATCHING ALGORITHM

In this paper, we analyzed selected multiple pattern string matching algorithms on the basis of time complexity, search type, key ideas and approach parameters. Each algorithm has certain advantages and disadvantages.

The main advantage of Brute force algorithm is that wide applicability, simplicity, reasonable algorithms for some important problems (e.g., matrix multiplication, sorting, searching, string matching).Weakness of the brute-force algorithms are unacceptably slow. It take much time as it search linearly.

KMP runs in optimal time: O(m+n). The algorithm never nseeds to move backwards in the input text, T. KMP doesn't work so well as the size of the alphabet increases, more chance of a mismatch (more possible mismatches).mismatches tend to occur early in the pattern, but KMP is faster when the mismatches occur later.

Finite state machine not suited to all problem domains, should only be used when a systems behavior can be decomposed into separate states with well defined conditions for state transitions. This means that all states, transitions and conditions need to be known up front and be well defined. Larger systems implemented using a FSM can be difficult to manage and maintain without a well thought out design

Boyer-Moore pattern matching algorithm ,which achieves sub-linear running time by skipping characters in the input text according to the ―bad character and ―good suffix heuristics. Boyer-Moore algorithm is extremely fast on large alphabet (relative to the length of the pattern).

IV. CONCLUSION

This paper is based on multiple pattern string matching algorithms. There are various scenarios where we can use a Particular type of algorithm. Comparison result proves that Bayer Moore Pattern matching algorithm is the most efficient One to apply on heterogeneous system for pattern matching. The Bayer Moore algorithm is to be very efficient as well as fast and gives the accurate results .

REFERENCES

[1] Aho, Alfred V,Margaret J.Corasick "Efficient string Matching: An aid to bibliographic search," Communication of the ACM 18(6) : 333-340, June 1975.

[2] Zeeshan Ahmed Khan1, R.K Pateriya2 "Multiple Pattern String Matching Methodologies: A Comparative Analysis". International Journal of Scientific and Research Publications, Volume 2, Issue 7, July 2012, ISSN 2250-3153.

[3] Devaki-Paul, "Novel Devaki-Paul Algorithm for Multiple Pattern Matching" International Journal of Computer Applications (0975 – 8887) Vol 13– No.3, January 2011.

[4] Raju Bhukya, DVLN Somayajulu, "Exact Multiple Pattern Matching Algorithm using DNA Sequence and Pattern Pair". International Journal of Computer Applications (0975 – 8887) Volume 17– No.8, March 2011.

[5] Ziad A.A Alqadi, Musbah Aqel & Ibrahiem M.M.EI Emary, "Multiple Skip Multiple Pattern Matching algorithms". IAENG International Vol 34(2) 2007.

[6] Rami H. Mansi, and Jehad Q. Odeh, "On Improving the Naive String Matching Algorithm," Asian Journal of Information Technology, Vol.8, No. I, ISS N 1682-3915,2009, pp. 14-23.

[7] S.Viswanadha Raju, S R Mantena, A.Vinaya Babu, G V S Raju. "Efficient Parallel Pattern Matching using Partition Method". Proceedings of the Seventh International Conference on Parallel and Distributed Computing,Applications and Technologies (PDCAT'06) 0-7695-2736-1/06 $20.00 © 2006 IEEE.

[8] R.A.Wagner, M.J.Fischer, "The String to String Correction Problem", Journal of the ACM, vol.21, pp. 168-173,1974.

[9] Vlastimil Kosar, Jan Korenek. "Reduction of FPGA Resources for RegularExpression Matching by Relation Similarity". 978-1-4244-9756-0/11/$26.00 ©2011 IEEE.

[10] Robert S. Boyer and J. Strother Moore."A fast string searching algorithm. Communications" of the ACM, 20(10):762–772, 1977.