ISSN:0975-9646
Akhtar Rasool Amrita Tiwari et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 3 (2) , 2012,3394 - 3397

String Matching Methodologies:A Comparative Analysis

Akhtar Rasool,Amrita Tiwari, Gunjan Singla,Nilay Khare
Department of computer Science & Engg.
Maulana Azad National Institute of Technology
Bhopal-462051, India.

Abstract- **String matching is the problem of finding all occurrences of a character pattern in a text. This paper provides an overview of different string matching algorithms and comparative study of these algorithms. In this paper, we have evaluated several algorithms, such as Naive string matching algorithm, Brute Force algorithm, Rabin-Karp algorithm, Boyer-Moore algorithm, Knuth-Morris-Pratt algorithm, Aho-Corasick Algorithm and Commentz Walter algorithm. We analysed the core ideas of these single pattern string matching algorithms and multi-pattern string matching algorithms.We compared the matching efficiencies of these algorithms by searching speed, pre-processing time, matching time and the key ideas used in these algorithms. It is observed that performance of string matching algorithm is based on selection of algorithms used and also on network bandwidth.**

Keyword- **String matching, Naive Search, Rabin Karp, Boyer-Moore, KMP, Exact String Matching, Approximate String Matching, Comparison of String Matching Algorithms.**

I.Introduction

String matching is a technique to find out pattern from given text. Let ∑ be an alphabet. Elements of ∑ are called symbols or characters. For example, if ∑ = {a, b}, then *abab* is a string over ∑. The pattern is denoted by P [1....m]. The text is denoted by T [1...n]. 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 [1]. Figure 1 shows this definition [2].

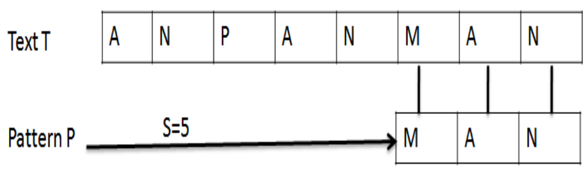

Figure 1: String Matching Example

II.Exact String Matching Algorithms

Exact string matching is used in search of any occurrence of a string *A* in string *B*. These algorithms are applied in biology, and especially in the segment concerning DNA chains [5]. Much of data processing in bioinformatics involves in one way or another recognising certain patterns within DNA, RNA or protein sequences.

A. Single pattern string matching algorithms

1) *Naive string matching algorithm*: It is also known as *Brute Force algorithm***.** It has no pre-processing phase, needs constant extra space. It always shifts the window by exactly one position to the right. It requires 2n expected text characters comparisons. It finds all valid shifts using a loop that checks the condition P[1....m]=T[s+1........s+m] for each of the n-m+1 possible values of s .

Consider the following example.

T=ANPANMAN

P=MAN

ANPANMAN

A brute force method for string matching algorithm is shown in Figure 2:

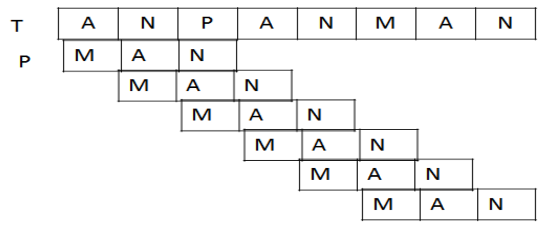

Figure 2: Naive String Matching Example

Naive string matching algorithm takes time O((n-m+1)m), and this bound is tight in the worst case. The worst case running time is thus O((n-m+1)m)[4]. The running time of Naive String Matching algorithm is equal to its matching time, since there is no pre-processing.

2) *Rabin Karp String Matching Algorithm*: This algorithm uses hashing function. It works in two phases i.e. pre-processing phase (time complexity Θ(m)),matching phase(time complexity average Θ (n+m),worst Θ((n-m+1) m)).[4]

Rabin Karp matcher is used to find a numeric pattern P from a given text T. It firstly divides the pattern with a predefined prime number q to calculate the remainder of pattern P. Then it takes the first m characters from text T at first shift s to compute remainder of m characters from text T. If the remainder of the pattern P and remainder of the text T are equal, only then we compare the text with pattern otherwise there is no need for comparison. We will repeat the process for next set of characters from text for all possible shifts which are from s=0 to n-m. So, according to this, two numbers n1

and n2 can only be equal if REM (n1/q) = REM(n2/q) .[1]
After division, there are three cases:-

TABLE I: CASES

Cases	Condition	Result
Successful hit	REM(n1)=REM(n2)	n1=n2
Spurious hit	REM(n1)=REM(n2)	n1≠n2
Unsuccessful hit	REM(n1)≠REM(n2)	n1≠n2

Ex- For a given text T, pattern P and prime number q
T=2345678997977797976534356678886756456890975 5453434342454547565 5454
P= 667888
q=11
REM(Text) = 234567/11 =3
REM(P) = 667888/11 =1
REM(Text) ≠ REM(P)
Now move on to next set of characters from text and repeat the procedure.

3) *Boyer-Moore String Matching Algorithm*: The Boyer-Moore algorithm (BM) was developed by R.S.Boyer and J.C.Moore in 1977[11].The BM algorithm scans the characters of the pattern from right to left beginning with the rightmost one and performs the comparisons from right to left. 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).It works in two phases: Pre-processing phase in $O(m+|\sum|)$ time complexity, Matching phase in $\Omega(n/m)$, O(n) time complexity[4]. There are 3n text character comparisons in the worst case when searching for a non periodic pattern. [3]
Assume that a mismatch occurs between the character *P[i]=b* of the pattern and the character *T[i+j]=a* of the text during an attempt at position *j*. Then, *P[i+1 .. m-1]=T[i+j+1 .. j+m-1]=u* and *P[i]≠T[i+j]*. The good-suffix shift consists in aligning the segment *T[i+j+1 .. j+m 1]=P[i+1 .. m-1]* with its rightmost occurrence in *P* that is preceded by a character different from *P[i]*. BM algorithm will carry through shift computing as follows:
Good-suffix function: The algorithm looks up string *u* leader character is not *b* in *P* from right to left. If there exists such segment, shift right *P* to get a new attempt window. If there exists no such segment, the shift consists in aligning the longest suffix *v* of *T[i+j+1 .. j+m- 1]* with a matching prefix of *P*.
Bad-char function: The bad-character shift consists in aligning the text character *T[i+j]* with its rightmost occurrence in *P[0 .. m-2]*. If *T[i+j]* does not occur in the pattern *P*, no occurrence of *P* in *T* can include *T[i+j]*, and the left end of the window is aligned with the character immediately after *T[i+j]*, namely *T[i+j+1]*.

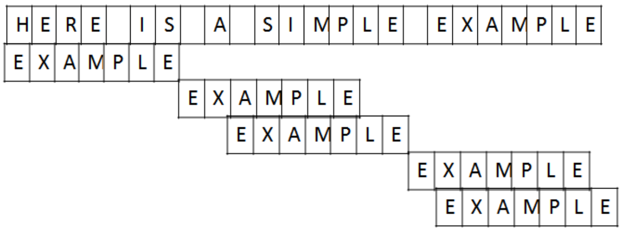

Figure 3: Boyer Moore String Example

BM algorithm uses above-mentioned good-suffix function and bad-char function to calculate the new comparing position shifting rightward *P*. Practice shows that BM Algorithm is fast in the case of larger alphabet. [3]
Scalpel [7] uses the Boyer-Moore single pattern search algorithm. The open-source file carver Scalpel searches for all occurrences of headers and footers from a dictionary of about 40 header- footer pairs in disks that are many gigabytes in size. [8]

4) *Knuth-Morris-Pratt String Matching Algorithm*: The Knuth-Morris-Pratt Algorithm (KMP) was developed by D.Knuth, J.Morris and V.Pratt in 1974. It compares the pattern with the text from left to right. In case of a mismatch or whole match it uses the notion border of the string. It decreases the time of searching compared to the Brute Force algorithm. [11]
KMP algorithm uses automata to find all the occurrences of a pattern in a text. The automata comprises of three parts (Figure 4):
Node: the prefixes of the pattern.
Success Link: link from the prefix node P[0 .. i-1] to the prefix node P[0 .. i]. When matching successfully, we use Success Link linking to the next state.
Failure Link: link from the prefix node P[0 .. i-1] to the prefix node P[0 .. j-1](j<i), which is the max prefix of P[0 .. i-1]. When matching failed, we use Failure Link to backshift proper state and go on. [12]

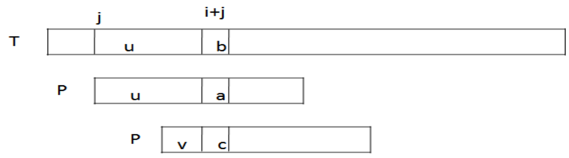

Figure 4: KMP Matching Method

During the searching phase, what happens to i is sort of like a finite automaton. At each step, shifts either to i+1or to i+j (shift j positions forward on occurring a mismatch). The value of j is just a function of i and does not depend on other information. So we can draw something like an automaton with arrows connecting values of j and labelled with matches and mismatches.
Figure 5 shows the working of KMP algorithm:

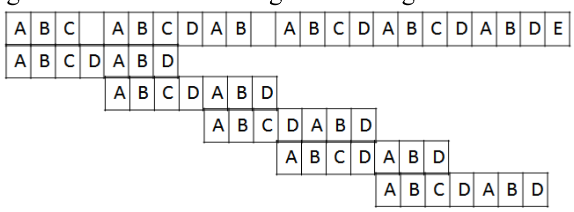

Figure 5: KMP Example

The KMP algorithm works by turning the patterns given into a machine, and then running the machine. It takes *O(m)* space and time complexity in pre-processing phase, and *O(n+m)* time complexity in searching phase (independent of the alphabet size). KMP is a linear time string matching algorithm. [12]

B. Multiple Pattern String Matching Algorithms

Multiple pattern matching is an important problem in text processing and is commonly used to locate all the positions of an input string (the so called "text") where one or more keywords (the so called "patterns") from a finite set of keywords occur. Multi-pattern string matching arises in a number of applications including network intrusion detection, digital forensics, business analytics, and natural language processing. The multiple pattern matching problems can be defined as:

Given an input string T[1...n] of length n and a finite set of r keywords P[p1...pr] where each pi is a string pi = pi1, pi2 . . . pim of length m over a finite character set Σ and the total size of all keywords is denoted as $|P|$, the task is to find all occurrences of any of the keywords in the input string[6]. There are many algorithms used for multi-pattern searching, which varies in speed, measured in terms of time complexity. A few are described below:

1) *Aho-Corasick String Matching Algorithm*: Aho-Corasick algorithm is one of the earliest multi-pattern exact matching algorithms. Aho-Corasick algorithm is a direct extension of the KMP algorithm by combining with the automata. The running time of Aho-Corasick is independent of the number of patterns. The complexity of Aho-Corasick algorithm is *O (nlogn)*. Similar to KMP algorithm, Aho-Corasick algorithm scans the character in text one by one without any jump.

There are two versions : nondeterministic and deterministic of the Aho-Corasick (AC) multi-pattern matching algorithm. The deterministic version makes half as many state transitions as made by the non-deterministic version. In the deterministic version (DFA), each state has a transition pointer for every character in the alphabet as well as a list of matched patterns. Aho and Corasick show how to compute the transition pointers. The number of state transitions made by a DFA when searching for matches in a string of length *n* is *n*. [9]

In pre-processing stage, Aho-Corasick constructs a state machine (Trie) from the strings to be matched. The state machine starts with an empty root node, which is the default non-matching state. Each pattern to be matched adds states to the machine, starting at the root and going to the end of the pattern. The state machine is then traversed and failure pointers are added from each node to the longest prefix of that node which also leads to a valid node in the Trie. [14]

Aho-Corasick works by constructing a state machine from the strings to be matched. The state machine starts with an empty root node which is the default non-matching state. Each pattern to be matched adds states to the machine, starting at the root and going to the end of the pattern. The state machine is then traversed and failure pointers are added from each node to the longest prefix of that node which also leads to a valid node in the trie. We show a single node of the state machine in Figure 6. [10]

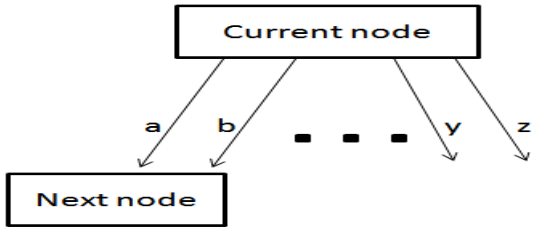

Figure 6 :Aho-Corasick Node Selection

Given a set of patterns = {search, ear, arch, chart}, Figure 7 shows the state machine and goto function. If the text string is "strcmatecadnsearchof" . Aho-Corasick algorithm scans the character in text one by one without any jump.

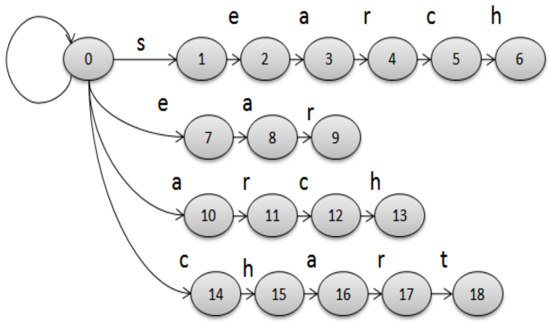

Figure 7:Aho-Corasick Example

2) *Commentz – Walter String Matching Algorithm:* Commentz-Walter algorithm combines the Boyer-Moore technique with the Aho-Corasick algorithm. In pre-processing stage, differing from Aho-Corasick algorithm, Commentz-Walter algorithm constructs a converse state machine from the patterns to be matched. Each pattern to be matched adds states to the machine, starting from right side and going to the first character of the pattern, and combining the same node.

In searching stage, Commentz-Walter algorithm uses the idea of BM algorithm. The length of matching window is the minimum pattern length. In matching window, Commentz-Walter 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 a pre-computed shift table to shift the window to the right.[14]

For pattern set { search, ear, arch, chart }, Figure 8 shows the Commentz-Walter state machine and the goto function for the text string "strcmatecadnsearchof".

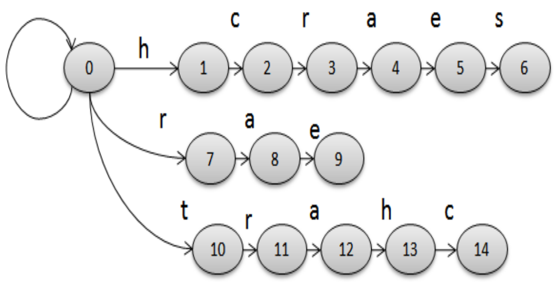

Figure 8: Commentz Walter Example

TABLE II: A COMAPARATIVE ANALYSIS

Algorithms	Time Complexity	Search Type	Multiple String	Key Ideas	Approach
Brute Force	O((n-m+1)m)	Prefix	No	Searching with all alphabets	linear searching
Rabin Karp	Θ(m),Θ (n+m)	Prefix	No	Compare the text and the patterns from their hash functions	hashing based
Boyer-Moore	O(m+|∑|) ,O(n)	Suffix	No	Bad-character and good-suffix heuristics to determine the shift distance.	heuristics based
Knuth-Morris-Pratt	O(m) ,O(n+m)	Prefix	No	constructs an automaton from the pattern	heuristics based
Aho Corasick	O(n),O(m+z);z= number of matches	Prefix	Yes	Finite automaton that tracks the partial prefix match.	automaton based
Commentz Walter		suffix	yes	constructs a converse state machine from the patterns	automaton based

III.APPROXIMATE STRING MATCHING ALGORITHMS

Approximate String matching is a problem in computer science which is applied in text searching, pattern recognition and signal processing applications. For a text T[1..n] and pattern P[1...m], we are supposed to find all the occurrences of pattern in the text whose edit distance to the pattern is at most K. The edit distance between two strings is defined as minimum number of character insertion, deletion and replacements needed to make them equal.

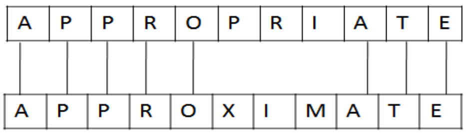

Figure 9: Approximate String Matching Example

Here K(T,P) = 3.

Approximate string matching problem is solved with the help of dynamic programming.

IV.COMPARATIVE ANALYSIS

This work categorizes the algorithms into various categories to emphasize the data structure that drives the matching. These categories are automaton-based, heuristics-based and hashing-based.

An *automaton-based* algorithm builds a finite state automaton from the patterns in the pre-processing stage and tracks the partial match of the pattern prefixes in the text by state transition in the automaton.

A *heuristics-based* algorithm allows skipping some characters to accelerate the search according to certain heuristics. Some algorithms require a verification algorithm following a possible match to verify if a true match occurs.

A *hashing based* algorithm compares the hash values of characters in the text segment by segment with those of the characters in the patterns. If both hash values are equal, a possible match may occur. The characters in the text and those in the patterns are then compared to verify if a true match occurs. [13]

Based on all the data represented in the paper, a comparative analysis of all the algorithms is:presented in Table II

IV. CONCLUSION

This research reviews and profiles some typical string matching algorithms to observe their performance under various conditions and gives an insight into choosing the efficient algorithms. By analyzing these string matching algorithms, it can be concluded that Boyer-Moore, Aho-Corasick and KMP string matching algorithms are efficient. Practice shows that BM Algorithm is fast in the case of larger alphabet. KMP decreases the time of searching compared to the Brute Force algorithm. Exact and approximate string matching algorithms makes various problems in the solvable state. Innovation and creativity in string matching can play an immense role for getting time efficient performance in various domains of computer science.

REFERENCES

[1] Rajender Singh Chillar, Barjesh Kochar “RB-Matcher: String Matching Technique”, World Academy of Science, Engineering and Technology 42 2008,pp.132-135.

[2]http://en.wikipedia.org/wiki/Boyer%E2%80%93Moore_string_search_algorithm.

[3] Jingbo Yuan, Jisen Zheng, Shunli Ding “An Improved Pattern Matching Algorithm”, Third International Symposium on Intelligent Information Technology and Security Informatics, 2010 ,pp.599-603.

[4] http://en.wikipedia.org/wiki/String_matching.

[5] Predrag Bro anac, Leo Budin, and Domagoj Jakobovi “Parallelized Rabin-Karp Method for Exact String Matching” *Int. Conf. on Information Technology Interfaces,* June 27-30, 2011, Cavtat, Croatia,pp. 585-590.

[6] Charalampos S. Kouzinopoulos and Konstantinos G. Margaritis “A Performance Evaluation of the Preprocessing Phase of Multiple Keyword Matching Algorithms” Panhellenic Conference on Informatics,2011,pp.85-89.

[7] http://www.digitalforensicssolutions.com/Scalpel.

[8] R. Boyer and J. Moore “ A fast string searching algorithm”, *CACM*, 20,10, 1977,pp.262-272.

[9] Xinyan Zha and Sartaj Sahni “Multipattern String Matching On A GPU”,IEEE,2011,pp. 277-282.

[10] Nathan Tuck, Timothy Sherwood, Brad Calder, George Varghese “Deterministic Memory-Efficient String Matching Algorithms for Intrusion Detection” IEEE INFOCOM 2004.

[11] Prasad J C#1, Dr.K.S.M.Panicker “Single Pattern Search Implementations in a Cluster Computing Environment”, on Digital Ecosystems and Technologies 2010,pp.391-396.

[12] A- Ning Du, Bin- Xingfang, Xiao-Chun Yun, Ming-Zenghu, Xiu-R ong Zheng,” Comparision of String Matching Algorithms: An Aid To Information Content Security” Proceedings of the Second International Conference on Mache Learmng and Cybernetics, xi", *2-5* November,2003,pp. 2996-3001.

[13] P0-Chinglin, Zhi-Xiang Li, Ying-Darlin, Yuan-Chang Lai, Frank C. Lin “ Profiling and Accelerating String Matching Algorithms In Three Network Content Security Applications”, IEEE communications surveys The Electronic Magazine of Original Peer-Reviewed Survey Articles, 2ND QUARTER 2006, VOLUME 8, NO. 2, pp. 24-36.

[14] Yang Dong hong, Xu Ke, Cui Yong “An Improved Wu-Manber Multiple Patterns Matching Algorithm” by the National Natural Science Foundation of China under Grant No.60473082 and the National Grand Fundamental Research 973 Program of China pp. 675-680.