A Comparison of String matching algorithms- Boyer-Moore algorithm and Brute-Force algorithm

P. P. Borah[1], G. Talukdar[2], Y. Jayanta Singh[3]
[1,2]*Department of Computer Science and Engineering, Royal Group of Institutions, Assam, India.*
(E-mail: pranjalborah777@gmail.com, talukdargitimoni@gmail.com)
[3]*Department of Computer Science and IT, Assam Don Bosco University, India.*
(E-mail: jayanta@dbuniversity.ac.in)

Abstract- String matching is a kind of pattern recognition problem widely used in most of the computer applications, including word processing, text searching and image pattern matching. Based on the Boyer-Moore and Brute-Force algorithm, an analysis of their performance has been presented in this paper. Through a series of experiments, the better performance of Boyer-Moore algorithm is illustrated in comparison to Brute-Force algorithm. It has been found that Boyer Moore algorithm is efficient and has minimum time complexity.

Keywords- Boyer-Moore algorithm, Brute-Force algorithm, Bad-character rule, Good-suffix rule, Preprocessing and String matching algorithms.

I. INTRODUCTION

The exact string matching problem can be defined as follows. For a given pattern P of length m and a text T of length n ($n \geq m$), in which all characters belong to fixed alphabet set $\sum$ and the goal is to find each and any occurrences of the pattern P in the text T. The problem of finding each and any occurrences of a pattern in a given text can be solved through the use of different string matching techniques. In solving some vital tasks, in which string matching has played an important role are signal processing, speech and pattern recognition, library system, web search engine, information retrieval, text-editing, computer security and DNA sequence analysis**.**

Depending upon the size of the text, size of the pattern and the number of the patterns to be matched, different string matching approaches has been put forward to decrease the number of comparison and to avoid large number of mismatch in string matching approach. The simplest way to find whether a given pattern P occurs inside a given text T is to check text T at successive positions from left to right. This approach is called Brute-Force string matching. It is quite effective for short texts and patterns but the efficiency decreases for large data (Azizah Abd Manaf *et al*., 2011). To increase the efficiency of string matching approach some other algorithms are there which invoke preprocessing and different order of comparison. A common exact string matching algorithm Boyer-Moore algorithm uses preprocessing and reverse order search logic that means the algorithm compares the pattern P with the text T within a sliding window in the right-to-left order (R. S. Boyer *et al*., 1977).

The string matching is one of the basic problems in information technology. Furthermore, the most essential time-consuming query in most of the applications is string matching. Better string matching algorithms can improve the applications' efficiencies more significantly. Therefore, fast string matching algorithm is an important area of research.

The paper is organized as follows. Section II includes a small description of some string matching algorithms along with Boyer-Moore and Brute-Force algorithm in detail. Section III highlights the performance comparison of Boyer-Moore and Brute-Force algorithm. Section IV is dedicated to the overview of the essence of string matching algorithm in the current problems in short.

II. STRING MATCHING ALGORITHMS

In the domain of text processing, string-matching acts as an important field. A string-matching algorithm is considered to be an efficient string-matching algorithm if it is complete and can find the pattern P in the text T by making minimal number of comparison within an effective time limit. A couple of algorithms employed to perform String-matching operation are Brute-Force, Karp-Rabin, Morris-Pratt and Boyer-Moore algorithm (Prabhakar Gupta *et al*., 2010).

Here we have considered the text T is of length n and the pattern P is of length m. The Brute-Force algorithm is a very basic string matching algorithm with $\boldsymbol{O}(mn)$ time complexity (Azizah Abd Manaf *et al*., 2011). Karp-Rabin algorithm uses a hashing function with expected running time $\boldsymbol{O}(n+m)$ whose preprocessing phase is in $\boldsymbol{O}(m)$ time complexity and searching phase is in $\boldsymbol{O}(mn)$ time complexity with constant space. Karp-Rabin algorithm can be used to match against multiple patterns (Graham A. Stephen *et al*., 2000). Due to multiple pattern matching capability, Karp-Rabin algorithm can be used to detect plagiarism even for larger phrases. Morris-Pratt algorithm performs the computation by skipping lots of useless comparisons to increase its efficiency. It performs the comparisons from left to right. Space and time complexity of the preprocessing phase is $\boldsymbol{O}(m)$ and searching phase is in $\boldsymbol{O}(n+m)$ time complexity (Graham A. Stephen *et al*., 2000). Thus Morris-Pratt algorithm gives better performance than Brute-Force algorithm and Karp-Rabin algorithm. On the other hand Boyer-Moore algorithm is faster than that of Morris-Pratt algorithm (Prabhakar Gupta *et al*., 2010).

Though Brute-Force algorithm is not as efficient as Karp-Rabin, Morris-Pratt and Boyer-Moore algorithm, it has some advantages over them. It does not require any pre-processing so it does not require additional space also. It can be effective for short texts and patterns. On the other hand the preprocessing phase time and space complexity of Boyer-Moore algorithm is $\boldsymbol{O}(m+n)$ and it gives better performance while the size of the pattern increases (R. S. Boyer *et al*., 1977). To study the diverging behaviors of Boyer-Moore and Brute-Force algorithm, a comparison between them is represented in this paper. Detailed descriptions of Boyer-Moore and Brute-Force algorithm are as follows.

A. Boyer-Moore algorithm:

Boyer-Moore algorithm uses reverse order search logic that is in the right-to-left order. The movement of sliding window is determined by using bad character rule and good suffix rule (S. S. Sheik *et al*., 2004). The algorithm performs a preprocessing over the pattern P and creates two tables, which are known as Boyer-Moore bad-character (bmBc) and Boyer-Moore good-suffix (bmGs) tables respectively. The Boyer-Moore bad-character table has an entry for each character in the alphabet set which gives the shift value based on the occurrence of the character in the pattern P. On the other hand, the Boyer-Moore good-suffix table gives the matching shift value for each and every character in the pattern P.

Bad Character Rule can be represented as follows. In this paper we have considered the text T be of length n and the pattern P be of length m. We are performing a pair-wise

comparing between the text *T* and the pattern *P* from right to left as shown in Fig.1. Suppose, P_1 is aligned to T_a, where *a* is an arbitrary position in the text *T*. Let us assume an arbitrary position *b* in the pattern *P* such that the first mismatch occurs while comparing T_{a+b-1} with P_b.

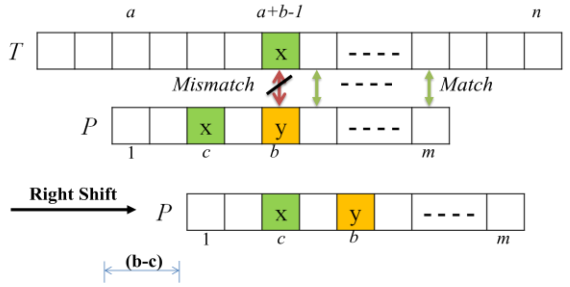

Fig.1: Right shifting of the pattern P using Bad Character Rule in Boyer-Moore algorithm.

Due to mismatch we move the pattern *P* towards right in such way that the largest position *c* in the left of P_b is equal to T_{a+b-1}. In this way, we can shift the pattern (*b-c*) positions towards right as shown in the Fig.1.

Good Suffix Rule can be represented as follows. Considering the text *T* and pattern *P* as shown in Fig.2, if a mismatch occurs while comparing T_{a+b-1} with P_b and there is a match of T_{a+b-1} with $P_{b'-m+b}$, where the value of b' ($m-b+1 \leq b' < m$) is the largest position such that $P_{b+1,m}$ is a suffix of $P_{1,b'}$ and $P_{b'-(m-b)} \neq P_b$.

Therefore we can move the window at least ($m-b'$) position or positions.

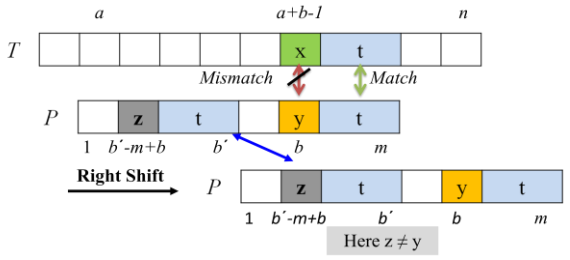

Fig.2: Right shifting of the pattern P using Good-Suffix Rule in Boyer-Moore algorithm.

The execution time of Boyer-Moore algorithm is sub-linear only because of these Bad-Character Rule and Good-Suffix Rule (Prabhakar Gupta *et al*., 2010). Using these two rules Boyer-Moore algorithm can make right shifts (if mismatch occur) which is greater than or equal to one. So the algorithm does not need to check each character of the text *T* which leads to the sub-linearity of the algorithm. Moreover the algorithm makes ***O***(*m*) comparisons when the pattern *P* is not present in the text *T*. The best performance of Boyer-Moore algorithm is ***O***(*n/m*) (Azizah Abd Manaf *et al*., 2011).

B. Brute-Force algorithm:

Brute force string matching is a quite simple approach. The algorithm attempts to match the pattern P with a sub-string of the text T at successive positions from left to right . Then after each mismatch, the algorithm shifts the pattern by exactly one position to the right as shown in Fig.3. Thus the algorithm checks at all positions in the text *T* from 1st to $(n-m)^{th}$, whether an occurrence of the pattern starts there or not (Azizah Abd Manaf *et al*., 2011).

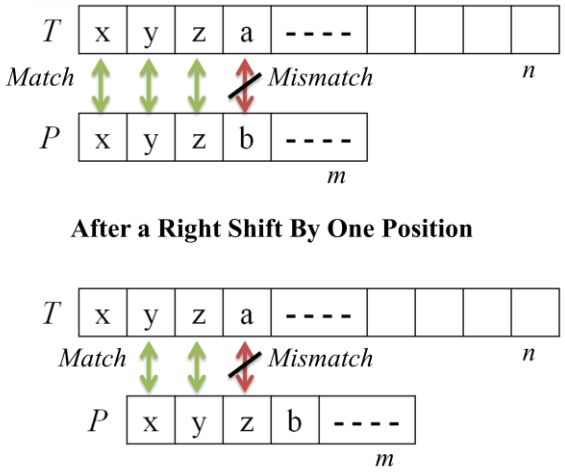

Fig.3: Brute force algorithm is checking for the pattern *P* in the text *T*. When a mismatch is occurred, the pattern *P* is shifted towards right exactly by one position.

The brute force algorithm requires no pre-processing phase. As the pattern length increases, the number of comparisons increases respectively. Time complexity of Brute-Force algorithm is ***O***(*mn*) (Azizah Abd Manaf *et al*., 2011).

The notion of such study is given in studies of Simone (Simone Faroy *et al*., 2012). Some of related studies are performed in literatures of following studies (Stefan Kurtz, 1996; Petteri Jokinen *et al*., 1988; Abdulwahab Ali Al-mazroi *et al*., 2011; M. Crochemore *et al*., 1994; Petri Kalsi *et al*., 2008; Nimisha Singla *et al*., 2012; Simone Faro *et al*., 2011; William W. Cohen *et al*., 2003)

III. PERFORMANCE COMPARISON

The study of R. S. Boyer *et al*., (1977) investigated that the Boyer-Moore algorithm is sub-linear as the number of references per character decreases as the patterns get longer. When the alphabet set is sufficiently large and the pattern is sufficiently long, the algorithm executes less than one instruction per character pass (R. S. Boyer *et al*., 1977). While on the other hand Brute-Force algorithm shows linear time complexity. The behaviors of Boyer-Moore and Brute-Force algorithm are shown in Table1.

Table1: Behaviors of Boyer-Moore and Brute-Force algorithm.

Behaviors	**Boyer-Moore**	**Brute-Force**
Order of comparison	Right to left	Left to right
Shifting rules	Both bad character rule and good suffix rule	One by one character shift
Preprocessing time complexity	***O****(m+n)*	No pre-processing

Both the Boyer-Moore and Brute-Force algorithm are implemented using java. The performance comparison is done with respect to the matching times (in Nanoseconds) for different input scales. In Table2 the performance comparison of Boyer-Moore and Brute-Force algorithms are given, where n is the length of the text *T*, m is the length of the pattern *P* and the matching times are given in Nanoseconds.

Table2: Performance comparison of Boyer-Moore and Brute-Force algorithm.

Input Scale	**Boyer-Moore**	**Brute-Force**
n=10, m=2	3,636,899	10,692,715
n=100, m= 2	4,074,790	14,627,957
n=100, m= 10	3,718,222	15,314,766
n=100, *m*= 50	3,591,666	15,824,537
n=1000, *m*= 50	5,121,397	32,428,455
n=1000,m=200	4,566,272	32,886,075

The Table2 summarizes that the Boyer-Moore algorithm is more efficient than that of Brute-Force algorithm. In such a way different algorithms can be implemented and compared.

IV. CONCLUSION

In the present arena to find the exact content in minimum time is the most essential factor. String algorithms play a crucial role in this regard. A lot of research is going on at the level of software and hardware to make pattern searching faster. By the application of various algorithms in diverse fields the approximate best algorithm can be ascertained. It has been found that most applications uses Boyer Moore algorithm for its desirable and efficient work and also most of other applications uses the basics of this algorithm as it has minimum time complexity.

REFERENCE

[1] Abdulwahab Ali Al-mazroi and Nur'Aini Abdul Rashid (2011), A Fast Hybrid Algorithm for the Exact String Matching Problem, *American J. of Engineering and Applied Sciences.* 4 (1): 102-107.

[2] Azizah Abd Manaf, Akram Zeki, Mazdak Zamani, Suriayati Chuprat, Eyas El-Qawa- smeh (2011), Informatics Engineering and Information Science, Part II, Springer Heidelberg Dordrecht, London, New York.

[3] Graham A. Stephen (2000), String Searching Algorithms, World Scientific Publishing Co., Singapore.

[4] M. Crochemore, A. Czumaj, L. Gasieniec, S. Jarominek, T. Lecroq, W. Plandowski, and W. Rytter (1994), Speeding Up Two String-Matching Algorithms, *Algorithmica*.

[5] Nimisha Singla, Deepak Garg (2012), String Matching Algorithms and their Applicability in various Applications, *International Journal of Soft Computing and Engineering*,Vol-I, 6.

[6] Petri Kalsi, Hannu Peltola, and Jorma Tarhio (2008), Comparison of Exact String Matching Algorithms for Biological Sequences, BIRD.

[7] Petteri Jokinen, Jorma Tarhio, and Esko Ukkonen (1988), A comparison of approximate string matching algorithms, Software: Practice and Experience, vol. 1(1), 1–4 January.

[8] Prabhakar Gupta, Vineet Agarwal, Manish Varshney (2010), Design and analysis of algorithms, PHI Learning Pvt. Ltd., New Delhi.

[9] R. S. Boyer and J. S. Moore (1977), A fast string searching algorithm, *Communications of the ACM*, vol. 20, no. 10, pp. 762–772.

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

[11] Simone Faroy and Thierry Lecroq (2012), A Multiple Sliding Windows Approach to Speed Up String Matching Algorithm, *Symposium on Experimental and Efficient Algorithms.*

[12] Simone Faro and Thierry Lecroq (2011), 2001–2010: Ten Years of Exact String Matching Algorithms, *Proceedings of PSC.*

[13] Stefan Kurtz (1996), Approximate String Searching under Weighted Edit Distance, *In Proceedings of Third South American Workshop on String Processing*, Brazil.

[14] William W. Cohen, Pradeep Ravikumar, Stephen E. Fienberg (2003), A Comparison of String Metrics for Matching Names and Records, *American Association for Artificial Intelligence.*