ISSN 2347 - 3983
Volume 9. No. 8, August 2021
International Journal of Emerging Trends in Engineering Research
Available Online at http://www.warse.org/IJETER/static/pdf/file/ijeter05982021.pdf
https://doi.org/10.30534/ijeter/2021/05982021

Comparative Analysis of Brute Force and Boyer Moore Algorithms in Word Suggestion Search

Vanessa Ardelia Layustira [1], Wirawan Istiono[2]
[1]Universitas Multimedia Nusantara, Indonesia, vanessa.layustira@student.umn.ac.id
[2]Universitas Multimedia Nusantara, Indonesia, wirawan.istiono@umn.ac.id

ABSTRACT

The development of information that continues to develop causes an explosion of information which certainly has a very complex impact on information storage management. This also impacts on companies that have several data that continues to grow every day. Therefore, there is a needs to have a search engine algorithm that can do a search system quickly with the development of information that continues to increase every day. Search engine applications or search engines in a computer system make it easy for users to find a variety of information. To facilitate its use, search engines add search features or better known as word suggestion, which in designing this application requires string matching algorithms that can be used to solve these problems. Many strings matching algorithms are available and therefore, the need for an analysis of the search algorithm to be able to help determine which search system is appropriate for use in word suggestion search. The result comparing brute force and boyer moore algorithm, it was found that as much as 79.05% showed that the Boyer Moore algorithm has a better time efficiency compared to the Brute Force.

Key words: Word Suggestion, Brute Force Algorithm, Boyer Moore Algorithm, Comparison of Algorithms.

1. INTRODUCTION

The presence of search engines or better known as Search Engines in 1990 has made it easier for computer users to find various information. The American Heritage Dictionary defines the Search Engine as a software program that traces, nets, and displays information from a database. To carry out its functions, this application requires a suggestion search feature closest to the search keyword. This feature is then better known as Word Suggestion [1].
In its design, Word Suggestion needs an algorithm to solve string matching problems. String Matching Algorithm is an algorithm that can match a word with the word you want to find. There are several kinds of String-Matching algorithms that have existed until now. One of them is the Brute Force Algorithm and Boyer Moore's Algorithm [2], [3]. The two algorithms have different ways of working so that when applied in a Word Suggestion search it will produce different search speeds and different efficiency factors as well.
With these differences, it will need an analysis that can compare the two String Matching Algorithms in a Word Suggestion search. This analysis is considered important to determine which algorithm is more effective in doing String Matching and which algorithm is faster, so users can access various information available more efficiently and faster [4].
Some efficiency factors that will be discussed in this journal are the results of each algorithm resulting, time efficiency, feasible solutions, optimal results, ease of implementation, and simplicity of the algorithm being tested [5][6].

2. LITERATURE REVIEW

2.1 Word Suggestion

Word Suggestion comes from English, which means "Suggestion of Words". And if the word is describe according to the Indonesian dictionary, "word" has any meaning that is born with speech; talk; competent. While "Suggestions" are suggestions, influences that can move people's hearts [7], [8]. Word Suggestion is a function of the Auto Text Application that is used to provide suggestions for words that are typed or searched for. By typing a few letters or all the letters, the system will look into the database if there is a word that meets the criteria of the letters entered. If a word that meets the criteria contained in the database, then a list will appear that shows the existing word suggestions. Checking itself is based on matching letters of words entered by users with a list of words in the database [9], [10].

2.2 String Matching Algorithm

String Matching Algorithm is an algorithm to find the search for all possible short strings (patterns) that appear in the text. The pattern is a string of length m characters (m <n). Text is a long string whose length is n characters [11][12]. This string matching algorithm can also be classified into 3 sections according to the search direction, including:

a. From left to right, which is the direction to read, algorithms that fall into this category are the Brute Force

Algorithm and the Knuth-Morris – Pratt Algorithm

b. From right to left, which is the direction that usually produces the best results, the Algorithm that falls into this category is the Boyer Moore Algorithm

c. And the last category is from the direction determined specifically by the Algorithm, this direction produces the best results theoretically, Algorithms included in this category are the Colussi Algorithm and the Crochemore-Perrin Algorithm.

.

3. METHODOLOGY

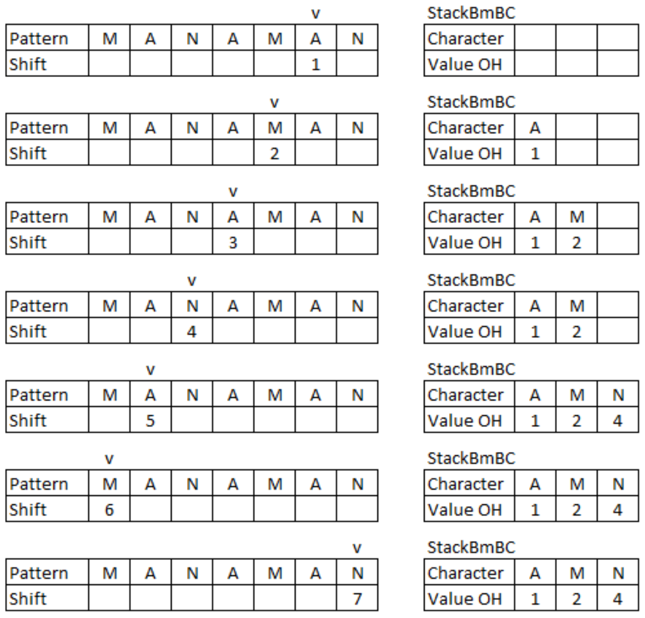
Brute Force Algorithm or better known as Brute-Force Searching Algorithm is a string-matching algorithm written without thinking for improving performance [13]. Brute Force algorithm is an approach to solving a problem with a simple way of thinking, it does not require any special abilities and thought to be able to solve a particular problem. For the reasons above, this algorithm was chosen in solving a simple problem.

In practice, this algorithm is very rarely used but is useful in comparative studies and other studies. Systematically, the steps were taken by the Brute Force Algorithm when matching strings are:

1. The Brute Force Algorithm starts to match the pattern at the beginning of the text.
2. From left to practice, this algorithm is very rarely used but is useful in comparative studies and other studies. This algorithm will match the characters per character pattern with the characters in the corresponding text until one of the following conditions is met:
   a. Characters in the pattern and the text being compared do not match (mismatch).
   b. All characters in the pattern match. Then the algorithm will notify the discovered position.
3. The algorithm then continues to shift the pattern by one to the right and repeats step 2 until the pattern is at the edge of the text.

The main idea of Boyer Moore's Algorithm is to do a matching from the far right of the search string. By using this algorithm, on average the search process will be faster than other search processes. The idea behind this algorithm is that by starting the character matching from the right, and not from the left, more information will be obtained [14]–[16]. Boyer Moore's algorithm uses a string matching method from right to left which is scanning character patterns from right to left starting from the rightmost character. Boyer Moore's algorithm uses two-shift functions namely good-suffix shift and bad-character shift to take the next step after there is a mismatch between character patterns and text characters that are matched to increase search speed [17]–[19]. The workings of the Boyer Moore algorithm it shown in Figure 1, where the step can describe as follow:

1. Run preBmBc and preBmGs procedures to get initialization. And then Run the preBmBc procedure. The function of this procedure is to determine how much shift is needed to reach a particular character in the pattern of the last/right character pattern. The results of the preBmBc procedure are saved in the BmBc table. Case in point: Pattern: MANAMAN



Pattern	M	A	N	A	M	A	N
Shift						1	

StackBmBC

Character			
Value OH			

Pattern	M	A	N	A	M	A	N
Shift					2		

StackBmBC

Character	A		
Value OH	1		

Pattern	M	A	N	A	M	A	N
Shift				3			

StackBmBC

Character	A	M	
Value OH	1	2	

Pattern	M	A	N	A	M	A	N
Shift			4				

StackBmBC

Character	A	M	
Value OH	1	2	

Pattern	M	A	N	A	M	A	N
Shift		5					

StackBmBC

Character	A	M	N
Value OH	1	2	4

Pattern	M	A	N	A	M	A	N
Shift	6						

StackBmBC

Character	A	M	N
Value OH	1	2	4

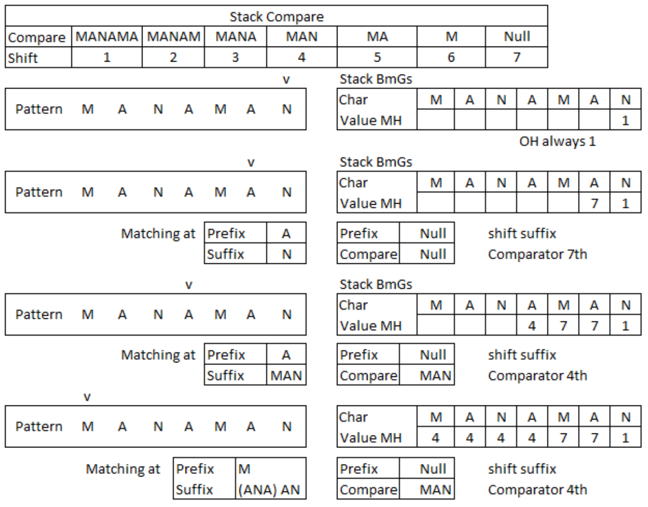
Pattern	M	A	N	A	M	A	N
Shift							7

StackBmBC

Character	A	M	N
Value OH	1	2	4

Figure 1: Trial Phase1 Boyer Moore Algorithm

2. Run the preBmGs procedure. Before running the contents of this procedure, the suffix procedure is run first in the pattern. The function of the suffix procedure is to check the compatibility of several characters starting from the last/right character with several characters starting from each character that is more left than the right character earlier. The results of the suffix procedure are saved in the suff table. So suff[i] records the length of suffix that matches the segment of the pattern ending in the i character. Case in point: Pattern: MANAMAN, the trial phase2 can be seen in Figure 2.



Stack Compare

Compare	MANAMA	MANAM	MANA	MAN	MA	M	Null
Shift	1	2	3	4	5	6	7

Pattern M A N A M A N

Stack BmGs

Char	M	A	N	A	M	A	N
Value MH							1

OH always 1

Pattern M A N A M A N

Stack BmGs

Char	M	A	N	A	M	A	N
Value MH						7	1

Matching at

Prefix	A
Suffix	N

Prefix	Null
Compare	Null

shift suffix Comparator 7th

Pattern M A N A M A N

Stack BmGs

Char	M	A	N	A	M	A	N
Value MH				4	7	7	1

Matching at

Prefix	A
Suffix	MAN

Prefix	Null
Compare	MAN

shift suffix Comparator 4th

Pattern M A N A M A N

Char	M	A	N	A	M	A	N
Value MH	4	4	4	4	7	7	1

Matching at

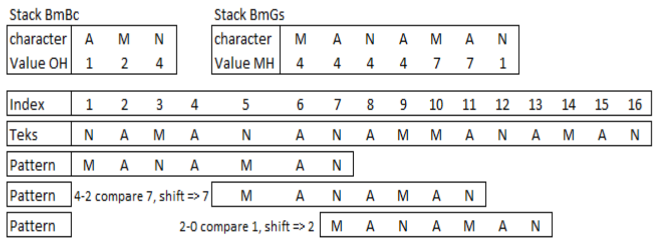
Prefix	M
Suffix	(ANA) AN

Prefix	Null
Compare	MAN

shift suffix Comparator 4th

Figure 2: Trial Phase2 Boyer Moore Algorithm

3. With the preBmGs procedure, it can be seen how many steps in the pattern from a segment to the same other segment are located more left with characters to the left of different segments. The preBmGs procedure uses the suff table to find out all pairs of the same segment. The string search process is performed using the results of the preBmBc and preBmGs procedures, namely the BmBc and BmGs tables. With sample case pattern is MANAMAN and the value sentence sample text is NAMANANAMMANAMAN, the result can be seen in Figure 3.



Stack BmBc

character	A	M	N
Value OH	1	2	4

Stack BmGs

character	M	A	N	A	M	A	N
Value MH	4	4	4	4	7	7	1

Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Teks	N	A	M	A	N	A	N	A	M	M	A	N	A	M	A	N
Pattern	M	A	N	A	M	A	N									
Pattern	4-2 compare 7, shift => 7				M	A	N	A	M	A	N					
Pattern					2-0 compare 1, shift => 2		M	A	N	A	M	A	N			

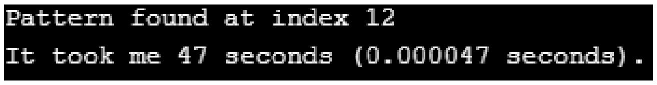
Figure 3: Final Phase and result Boyer Moore Algorithm

In this step, Brute Force and Boyer Moore algorithms will be implemented according to the test parameters that you want to analyze. These parameters include the results generated by the algorithm, algorithm time efficiency, feasible solutions, optimal results, ease of implementation, and simplicity of the algorithm tested to solve string matching problems in searching for word suggestions or information. Algorithm testing is performed using the C ++ programming language on Asus A442U laptops with Intel® Core ™ i5-8250U CPU @ 1.60GHz 1.80 GHz. In testing this algorithm, researchers conduct tests with the same input data and each test is carried out twice as much with four different input inputs. The testing steps for each algorithm are by entering a string of text that you want to test with a different length of text or string.
In Figure 4 shows Brute Force algorithm to test string matching in the first text. The input that was included in this initial test was the string "Vanessa Ardelia" with the keyword searched for "lia".

```
#include <bits/stdc++.h>
#include <time.h>
using namespace std;

void search(char* pat, char* txt) {
    int M = strlen(pat);
    int M = strlen(txt);

    for (int i=0; i<=N-M; i++) {
        int j;

        for(j=0; j<M; j++) {
            if(txt[i+j] != pat[j]) {
                break;
            }
        }
        if(j==M) {
            cout<<"pattern fount at index "<<i<<endl;
        }
    }
}

int main() {
    clock_t t;
    t = clock();
    char txt[] = "Vanessa Ardelia";
    char pat[] = "lia";
    search(pat,txt);
    t = clock() - t;
    printf("It took me %d second (%f
seconds).\n,t,((float)t)/CLOCKS_PER_SEC");
    return 0;
}
```

Figure 4: Brute force algorithm code to test string matching in the first text

From the algorithm above, we get the results for the string "Vanessa Ardelia" with the search keyword "lia", found in the 12th index with the total time needed to execute the Brute Force algorithm by 47 seconds, it shown in Figure 5.



```
Pattern found at index 12
It took me 47 seconds (0.000047 seconds).
```

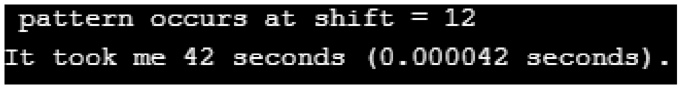
Figure 5: Brute force algorithm code to test string matching in the first text result

In Figure 5 shows Boyer Moore algorithm to test string matching in the first text. The input that was included in this initial test was the string "Vanessa Ardelia" with the keyword searched for "lia".

```
#include <limits.h>
#include <string.h>
#include <stdio.h>
#include <time.h>

# define NO_OF_CHARS 256

int max(int a, int b) { return (a,b)?a:b;}

void badCharHeuristic(char *str, int size, int
badchar(NO_OF_CHARS))
{
    int i;
    for(i=0; i<NO_OF_CHARS; i++) {
        badchar[i] = -1;
    }
    for(i=0; i<size; i++) {
        badchar[(int)str[i]] = i;
    }
}
```

Figure 6: Boyer Moore algorithm code to test string matching in the first text

From the algorithm that shown in Figure 6, the results for the string "Vanessa Ardelia" with the search keyword "lia", found in the 12th index with a total time required for Boyer Moore's algorithm execution by 42 seconds shown in Figure 7.



```
 pattern occurs at shift = 12
It took me 42 seconds (0.000042 seconds).
```

Figure 7: Boyer Moore algorithm code to test string matching in the first text result

4. RESULT AND DISCUSSION

From the methodology and trying phase brute force and Boyer Moore's algorithm that shown in Figure 5 and Figure 7, the next step is to conduct an experiment to ensure the results of the two algorithms, an experiment will be carried out to search for words from several sentences. and here are some sentences and words that will be tested, namely the string "ABCDSNJNNNABDUA" to search for the keyword "NAB", and the string "Ayam goreng kremes" to search for the keyword "Duck" and the string "adadsdad" to search for the keyword "ada", and the last string that will be tried is "ADADAKSNASASASASSASASASSASASASASSAFDK MDLNLNDSNALK" to search for the keyword "NALK". And from all the sentence and pattern that already tried with same code or algorithm that already tried in methodology step, obtained results as shown in Table1.

Table 1: Table result from various string and keyword

Algo rithm	String	Keyword	Length	Time
Brute Force	ABCDSNJN NABDUA	NAB	14	0,000041
	Ayam Goreng Kremes	Bebek	16	0,000002
	adadsdad	ada	8	0,000058
	ADADAKSN ASASASAS SASASASAS SASASASAS SAFDKMDL NLNDSNALK	NALK	56	0,000067
Boyer Moore	ABCDSNJN NABDUA	NAB	14	0,000032
	Ayam Goreng Kremes	Bebek	16	0,000004
	adadsdad	ada	8	0,000049
	ADADAKSN ASASASAS SASASASAS SASASASAS SAFDKMDL NLNDSNALK	NALK	56	0,000050

Based on the test result that shows in Table 1, if the result time complexity both algorithm is compared, it will get result that shown in table 2.

Table 2: Compare time complexity result between brute force and boyer moore algorithm

Brute force Time complexity	Boyer Moore Time complexity	Found keyword	Compare Result
0,000041	0,000032	Found	78,04878049
0,000002	0,000004	Not found	200
0,000058	0,000049	Found	84,48275862
0,000067	0,00005	Found	74,62686567
Average Compare Result			109,2896012
Average found keyword			79,05280159

From the result comparison that shown in Table 2, it was found that as much as 79.05% showed that the Boyer Moore algorithm has a better time efficiency compared to the Brute Force whether the keyword was found or not. Besides, other factors such as feasible solutions, optimal results, ease of implementation, and simplicity of the tested algorithm can be seen in the Table 3.

Table 3: Simplicity of the tested brute force and boyer moore algorithm

	Brute Force	*Boyer Moore*
feasible solutions	√	√
optimal result	√	√
ease of implementation	√	√
simplicity dari algoritma	√	

Based on testing the two algorithms between brute force and boyer moore, can provide feasible solutions for string matching problems in word suggestion, both of these algorithms are also able to provide optimal results, although there are differences in time in the execution of the algorithm. Both the Boyer Moore and Brute Force algorithms are equally easy to implement into coding, the Brute Force algorithm seems simpler than the Boyer Moore algorithm which has its conditions for executing string matching.

5. CONCLUSION

Based on the results and elaboration of the above test, the conclusion is that the Brute Force and Boyer Moore algorithms can overcome the word suggestion problem in searching for keywords using string matching. However, both of these algorithms have advantages and disadvantages of each, where the Brute Force algorithm is more suitable for string matching with keywords and strings that are not too long, while the Boyer Moore algorithm is suitable for all types and conditions of the string and keyword being searched.

ACKNOWLEDGEMENT

Thank you to the Universitas Multimedia Nusantara, Indonesia which has become a place for researchers to develop this journal research. Hopefully, this research can make a major contribution to the advancement of technology in Indonesia.

REFERENCES

1. A. M. Thike, S. Lupin, and Y. Vagapov, “Implementation of brute force algorithm for topology optimisation of wireless networks,” 2016 International Conference for Students on Applied Engineering, ICSAE 2016, no. October 2017, pp. 264–268, 2017.
2. P. Pangestu and S. E. Wahyuningrum, “Word search using boyer-moore algorithm,” Proxies : Jurnal Informatika, vol. 2, no. 1, pp. 6–11, 2018.
3. D. Budianta, “Brute Force Algorithm Implementation on Knowledge Management System Overcoming Heavy Metal of Pb and Cd in Soil At Palm Oil Plantation,” International Journal of Latest Trends in Engineering and Technology, vol. 8, no. 2, pp. 297–301, 2017.
4. C. KOMALASARI and W. ISTIONO, “A Comparative Study of Cocktail Sort and Insertion Sort,” Journal of Applied Computer Science & Mathematics, vol. 15, no. 1, pp. 21–25, 2021.
5. S. Smale, “On the efficiency of algorithms of analysis,” Bulletin of the American Mathematical Society, vol. 13, no. 2, pp. 87–121, 1985.
6. W. Philips, W. Istiono, and U. M. Nusantara, “Analysis of MinFinder Algorithm on Large Data Amounts,” International Journal of Emerging Trends in Engineering Research, vol. 9, no. 6, pp. 627–632, 2021.
7. M. M. Yulianto, R. Arifudin, and A. Alamsyah, “Autocomplete and Spell Checking Levenshtein Distance Algorithm To Getting Text Suggest Error Data Searching In Library,” Scientific Journal of Informatics, vol. 5, no. 1, p. 75, 2018.
8. R. Muntazari, M. . Arini, and M. K. Hendra Bayu Suseno, “Application of the Boyer Moore Method in the Application Dictionary of Web-Based Information Technology Terms,” INTEGRATED (Information Tecknology and Vocational Education), vol. 1, no. 2, pp. 1–8, 2019.
9. F. T. Waruwu, “Application Of Boyer Moore Algorithm for Text Searching,” International Journal of Informatics and Computer Science (The IJICS), vol. 1, no. 1, pp. 18–22, 2017.
10. A. Kumar, “Available Online at www.jgrcs.info STRING MATCHING RULES USED BY VARIANTS OF BOYER-MOORE,” Journal of Global Research in Computer Science, vol. 5, no. 1, pp. 8–11, 2014.
11. B. Lakshmi and B. Navyasri, “of Emerging Trends Energy Efficient Routing Mechanism for Harsh Environment,” International Journal of Emerging Trends in Engineering Research, vol. 7, no. 9, pp. 7–11, 2019.
12. S. I. Hakak, A. Kamsin, P. Shivakumara, G. A. Gilkar, W. Z. Khan, and M. Imran, “Exact String Matching Algorithms: Survey, Issues, and Future Research Directions,” IEEE Access, vol. 7, pp. 69614–69637, 2019.
13. K. A. F. A. Samah, N. Sabri, R. Hamzah, R. Roslan, N. A. Mangshor, and A. A. M. Asri, “Brute force algorithm implementation for traveljoy travelling recommendation system,” Indonesian Journal of Electrical Engineering and Computer Science, vol. 16, no. 2, pp. 1042–1049, 2019.
14. R. Rahim, A. S. Ahmar, A. P. Ardyanti, and D. Nofriansyah, “Visual Approach of Searching Process using Boyer-Moore Algorithm,” Journal of Physics: Conference Series, vol. 930, no. 1, 2017.
15. S. Supatmi and I. D. Sumitra, “Fingerprint Identification using Bozorth and Boyer-Moore Algorithm,” IOP Conference Series: Materials Science and Engineering, vol. 662, no. 2, 2019.
16. M. J. and . D. N. K., “An Enhanced Boyer-Moore Algorithm for WorstCase Running Time,” GSTF Journal on Computing (JoC), vol. 2, no. 1, pp. 152–157, 2014.
17. N. Ben Nsira, T. Lecroq, and M. Elloumi, “A fast Boyer-Moore type pattern matching algorithm for highly similar sequences,” International Journal of Data Mining and Bioinformatics, vol. 13, no. 3, pp. 266–288, 2015.
18. P. P. Borah, G. Talukdar, and Y. J. Singh, “A Comparison of String Matching algorithms-Boyer Moore Algorithm and Brute Force algorithm,” National Conference on New Approaches of Basic Sciences towards the Development of Engineering and TechnologyAt: Assam Don Bosco University, Guwahati, no. March 2013, pp. 1–6, 2013.
19. R. Fitriyanto, A. Yudhana, and S. Sunardi, “Boyer-Moore String Matching Algorithm and SHA512 Implementation for Jpeg/exif File Fingerprint Compilation in DSA,” JUITA: Jurnal Informatika, vol. 8, no. 1, p. 1, 2020.