GAZİOSMANPAŞA BİLİMSEL ARAŞTIRMA DERGİSİ (GBAD)
Gaziosmanpasa Journal of Scientific Research
ISSN: 2146-8168
http://dergipark.gov.tr/gbad
Araştırma Makalesi (Research Article)

Cilt/Volume : 12
Sayı/Number: 1
Yıl/Year: 2023
Sayfa/Pages: 76-85

***Alınış tarihi (Received)*:** 02.02.2023
***Kabul tarihi (Accepted)*:** 22.05.2023

Applied Comparison of String Matching Algorithms

Zeynep BARUT[1,*] **Volkan ALTUNTAŞ**[2]

[1]*Bursa Teknik Üniversitesi, Mühendislik ve Doğa Bilimleri Fakültesi, Bilgisayar Mühendisliği Bölümü, Bursa*
[1]*Bursa Teknik Üniversitesi, Mühendislik ve Doğa Bilimleri Fakültesi, Bilgisayar Mühendisliği Bölümü, Bursa*
**Sorumlu yazar: zebarut@gmail.com*

ABSTRACT: String matching algorithms are used to find a particular pattern in a string. The aim of this research is to explain the basic ideas and complexity of current string matching algorithms and to perform applied comparison. There are many algorithms used for string matching. In this study, Knuth-Morris-Pratt, Rabin Karp and Boyer Moore Horspool algorithms were compared. It is aimed to increase the accuracy of the study by choosing different algorithms. The basic ideas, potential difficulties and complexities of the algorithms are explained and how these problems can be solved is emphasized. In order to better evaluate the algorithms, three different algorithms were compared in twenty-five different pattern types. As a result of the studies, it has been seen that the Knuth-Morris-Pratt algorithm outperforms other algorithms in most cases. The second best performing algorithm was Boyer Moore Horspool algorithm, and the worst performing algorithm was Rabin Karp algorithm.

Keywords *– String Matching, DNA Sequence, Gene Analysis*

1. Introduction

DNA and protein sequence are long texts found over certain alphabets. These texts appear in problems such as searching for specific sequences, assembling the DNA chain from experimentally obtained fragments, or determining how different two genetic sequences are from each other. However, due to different kinds of errors in experimental measurements, small differences in chains, mutations, evolutionary changes, patterns often do not match the text. Because of all these problems, string matching algorithms are used to find and calculate similarities. String matching is one of the popular areas of research that finds a pattern within a string. Strings formed with certain alphabets form DNA sequences that indicate the genetic code of living things. Through these strings, operations such as searching for certain sequences, determining how similar or different two genetic sequences are, and finding chains that are very similar to the DNA chain sought are performed. In order to perform all these operations, string matching algorithms must be used (Navarro, 2001).

An abnormality is detected by comparing a sequence with a normal sequence. In order to predict the function of a gene, comparisons are made with other similar genes. After the comparison, the new sequence is revealed and matched to the existing sequences. Alignment is performed by finding the same characters in the sequences. Figure 1 shows the alignment of the two sequences (Tripathi ve Pandey, 2016).

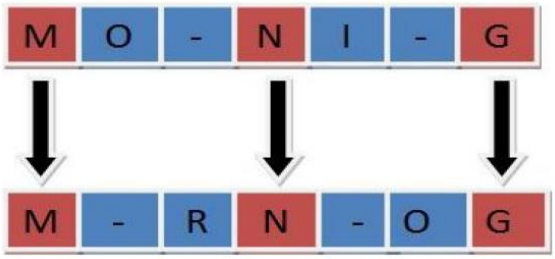

Figure 1. Alignment of two sequences (Tripathi ve Pandey, 2016).

In order for diseases to be detected, repetitions in a particular gene must be found. These repeats may be of different lengths or the repeats may be defective. DNA repeats are sequences found in more than one copy. A definite repeat is a short sequence of nucleotides repeated at least twice contiguously. An approximate repeat is a sequence of nucleotides that is repeatedly repeated with differences between samples. Duplicates that are distant in the genome are called distant or scattered repeats (Pop, 2015).

Choosing the appropriate string matching algorithm is an important process. A string matching algorithm looks for the pattern in the string without paying attention to the order in the alphabet. These stages require a lot of computation and are time consuming. According to the complexity or application of the problems, the most suitable string matching algorithm should be selected among various algorithms. String matching algorithms are divided into two as Exact string matching algorithms and Approximate string matching algorithms as given in Figure 2. Exact string matching algorithms find exact matches, while Approximate string matching algorithms find similar matches. Which algorithm is appropriate to use should be determined according to the requirements (Hakak ve ark. 2019).

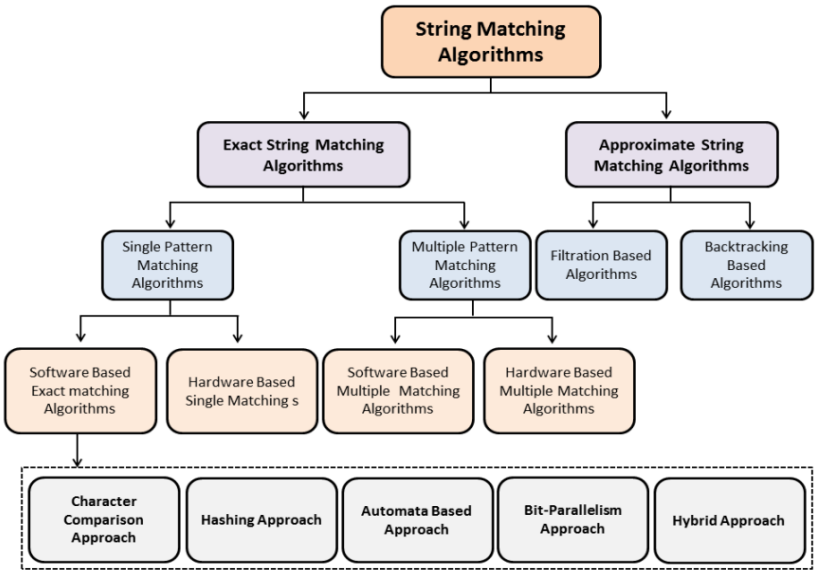

Figure 2. Classification of string matching algorithms (Hakak ve ark. 2019).

Various implementations of many string matching algorithms have been made. By trying different applications with different algorithms, it was tried to find the best performing algorithm for the applications. Thus, it is aimed to reduce the complexity and shorten the computation time. As a result of the studies, it has been seen that the use of Boyer Moore and Knuth Morris Pratt algorithms is more efficient (Singla and Garg, 2012). A new string matching algorithm is proposed. The proposed algorithm is aimed to be used in the field of

network security. As a result of the studies, it has been seen that the new algorithm is more efficient than Boyer Moore and the performance increases as the pattern gets longer. In addition, the overall performance of the new algorithm was found to be very high compared to other algorithms (Alshahrani and Khalil, 2022). A faster algorithm is proposed instead of the Boyer Moore algorithm. As a result of the studies, it has been seen that the proposed algorithm can match efficiently. It also reduced the comparison time (Yuan, 2011). It is aimed to develop the Boyer Moore algorithm using a shift table algorithm. The string is preprocessed through this algorithm. In this way, it has been seen that the algorithm is more successful. The complexity of the created algorithm is O(N) (Kanuga, 2015). Sequences of different lengths were tried on the data set by proposing a new algorithm. As a result of the studies, it has been seen that the algorithm created has high performance. It has also been observed that the algorithm works better as the string length increases (Islam ve Talukder, 2017)

2. Material and Methods

In this study, three algorithms were tested on the Citi bike trip data set in twenty-five different pattern types in order to find pattern matches in a text. By making applied comparisons of the algorithms, the results were examined and the algorithms were compared.

Knuth-Morris-Pratt Algorithm

The Knuth-Morris-Pratt algorithm is used to find patterns in a text. This algorithm looks at the characters one by one and compares them from left to right. In case of mismatch, prior information is used to avoid matching characters that are already known to match based on previous information, as some characters in the next window are known. In case of mismatch, it uses a preprocessed table called a prefix table to skip character comparison during matching. An example of the calculation of the algorithm is given in Figure 3. Most naive string matching algorithms run in O(nm) time, while the Knuth-Morris-Pratt algorithm runs in O(m + n) time. In the algorithm, n represents the length of the string and m the length of the pattern. The time complexity of the algorithm is very fast compared to other string matching algorithms. But it is a complex algorithm to understand. It is widely used in the fields of bioinformatics and DNA sequencing (Mathur, 2022).

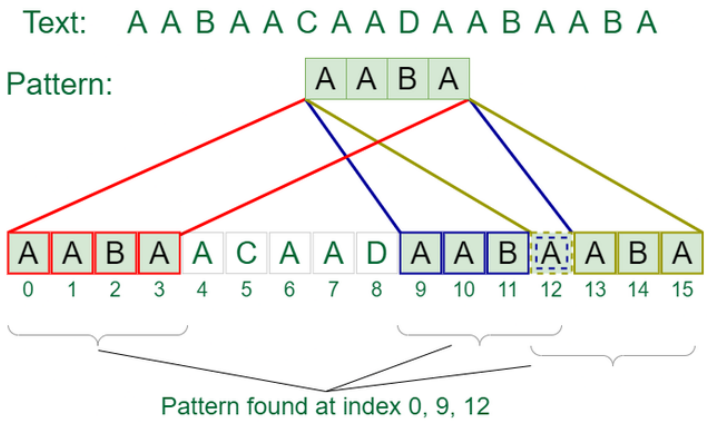

Figure 3. Representation of the Knuth-Morris-Pratt algorithm (Mathur, 2022).

Rabin Karp Algorithm

Rabin Karp algorithm uses hash function to find string matching matches and compares hash values. If the values are the same, it performs left-to-right comparison for each character in the pattern. Like the naive string matching algorithm, it does not look at every character, it separates unmatched characters. Because it uses hash, it calculates quickly and avoids conflicts. The hash values of a window of the pattern and string may match, but the window may not be the actual pattern. This increases the time complexity of the algorithm. An example of calculating the hash values of windows is given in Figure 4. The average and best-case complexity of the algorithm is O(m + n) and the worst-case complexity is O(mn) (Anonymous, 2022).

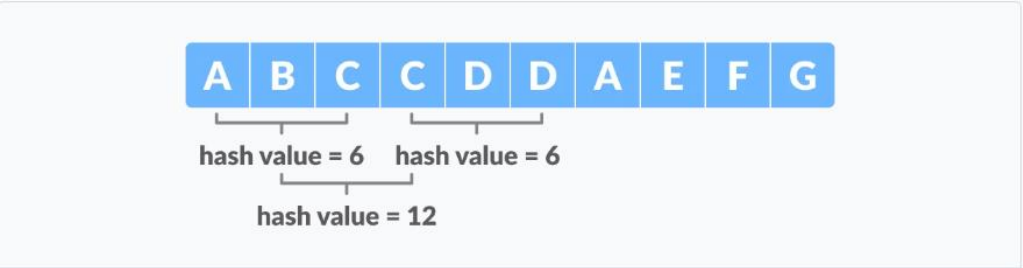

Figure 4. Hash values of scrolled windows (Anonymous, 2022).

Boyer Moore Horspool Algorithm

Boyer Moore algorithm uses the good suffix rule whereas Boyer Moore Horspool algorithm is simpler to implement as it uses the bad character rule. Boyer Moore Horspool is used to find one or more patterns in a string. This algorithm aligns and compares each character of the string with the pattern. The mismatch table is created from pattern characters. If there is no match, the search jumps to the next matching position in the pattern based on the value specified in the mismatch table. The running time of the algorithm depends on the size of the searched string. The run time is short compared to most search algorithms. It does not check all the characters because it jumps with the help of the mismatch table. The longer the pattern, the faster the algorithm. Because if the match between the two strings fails, the algorithm uses the mismatch table to extract the positions where the substring cannot match. The worst case complexity is O(nm). An example of the calculation of the algorithm is given in Figure 5 (Saldana, 2021).

Letter	a	b	c	d	*
Value	3	2	1	4	4

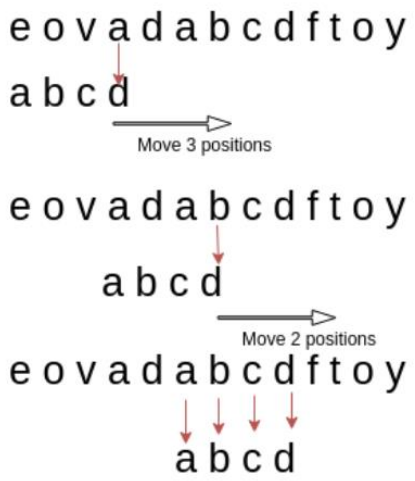

Figure 5. Representation of the Boyer Moore Horspool algorithm (Saldana, 2021).

3. Results and Discussion

Three different algorithms were compared in twenty-five different pattern types. Dev C++ development environment was used to implement the applications. As the data set, June 2014 data was used among the Citi bike trip data. There are 15 features and 936,881 rows of data in the data set (Anonymous, 2023). A view from the data set is given in Figure 6. In the first five cases, various string formations are looked at. Between the Sixth and Tenth cases, the occurrence of the numbers when it is possible to test them is looked at. Searched for a long string between the eleventh and fifteenth states. Searched for a short string between the sixteenth and twentieth states. In the last five cases, a string that is not in the data set is searched. The performance results of the Knuth-Morris-Pratt algorithm are given in Table 1, the performance results of the Rabin-Karp algorithm are given in Table 2, and the performance results of the Boyer-Moore-Horspool algorithm are given in Table 3. It has been observed that the Knuth-Morris-Pratt algorithm gives better results when the searched string is diverse, when the searched string is very short, and when a string is searched that is not in the dataset. It has been observed that Rabin-Karp algorithm gives better results in cases where the searched string is diverse and the searched string is very short. It has been seen that Boyer-Moore-Horspool algorithm gives better results in cases where the searched string is diverse and the searched string is very long and short. In most cases, the Knuth-Morris-Pratt algorithm outperformed other algorithms. In general, Boyer Moore Horspool algorithm gave better results than Knuth-Morris-Pratt algorithm in very long strings where time occurrences are searched. The second best performing algorithm was Boyer Moore Horspool algorithm, and the worst performing algorithm was Rabin Karp algorithm. The Boyer-Moore-Horspool algorithm skips when there is a mismatch. Rabin Karp algorithm works slower as it does not skip any characters (Biçer ve Zhang, 2019).

tripduration,"starttime","stoptime","start station id","start station name","start station latitude","start station longitude","end station id","end station name","end station latitude","end station longitude","bikeid","usertype","birth year","gender"
520,"2014-06-01 00:00:02","2014-06-01 00:08:42","358","Christopher St & Greenwich St","40.73291553","-74.00711384","426","West St & Chambers St","40.71754834","-74.01322069","18840","Subscriber","1979","1"
520,"2014-06-01 00:00:27","2014-06-01 00:09:07","335","Washington Pl & Broadway","40.72903917","-73.99404649","265","Stanton St & Chrystie St","40.72229346","-73.99147535","17442","Customer",\N,"0"
414,"2014-06-01 00:00:32","2014-06-01 00:07:26","439","E 4 St & 2 Ave","40.7262807","-73.98978041","368","Carmine St & 6 Ave","40.73038599","-74.00214988","16447","Subscriber","1980","1"
310,"2014-06-01 00:00:34","2014-06-01 00:05:44","463","9 Ave & W 16 St","40.74206539","-74.00443172","380","W 4 St & 7 Ave S","40.73401143","-74.00293877","18218","Subscriber","1984","1"
457,"2014-06-01 00:00:35","2014-06-01 00:08:12","352","W 56 St & 6 Ave","40.76340613","-73.97722479","305","E 58 St & 3 Ave","40.76095756","-73.96724467","18115","Subscriber","1969","1"

Figure 6. A view of the dataset

Table 1. Performance results of the Knuth-Morris-Pratt algorithm

Knuth-Morris-Pratt Algorithm			
Case	Pattern Type	Pattern	Time
1	Many Patterns	“Shevchenko Pl & E 7 St”	1985469 microsecond
2	Many Patterns	“Broadway & W 29 St”	1847674 microsecond
3	Many Patterns	“Cadman Plaza West & Montague St”	1834609 microsecond
4	Many Patterns	“E 47 St & 1 Ave”	11431 microsecond
5	Many Patterns	“Lexington Ave & E 26 St”	426877 microsecond
6	Timestamp - Digits	“2014-07-01 00:05:06”	2299193 microsecond
7	Timestamp - Digits	“2014-06-01 00:16:12”	2303608 microsecond
8	Timestamp - Digits	“2014-06-28 13:11:12”	596743 microsecond
9	Timestamp - Digits	“2014-06-01 00:08:42”	563297 microsecond
10	Timestamp - Digits	“2014-07-01 00:00:57”	532006 microsecond
11	Very Long	“Elizabeth St & Hester St"","40.71729","-73.996375","250","Lafayette St & Jersey St”	1869288 microsecond
12	Very Long	“Clark St & Henry St"","40.69760127","-73.99344559","274","Lafayette Ave & Fort Greene Pl”	1835850 microsecond
13	Very Long	“Broadway & E 14 St"","40.73454567","-73.99074142","459","W 20 St & 11 Ave","40.746745”	438279 microsecond
14	Very Long	“Howard St & Centre St"","40.71910537","-73.99973337","507","E 25 St & 2 Ave”	421778 microsecond

15	Very Long	“Watts St & Greenwich St””,””40.72405549””,””-74.00965965””,””358””,””Christopher St & Greenwich St”	463464 microsecond
16	Very Short	“ab”	1882694 microsecond
17	Very Short	“ca”	1873651 microsecond
18	Very Short	“ad”	432266 microsecond
19	Very Short	“sc”	437420 microsecond
20	Very Short	“West”	15616 microsecond
21	Not Exists	“bbb”	1877225 microsecond
22	Not Exists	“bab”	1876326 microsecond
23	Not Exists	“dca”	437515 microsecond
24	Not Exists	“East”	421779 microsecond
25	Not Exists	“dme”	421775 microsecond

Table 2. Performance results of the Rabin-Karp algorithm

Rabin-Karp Algorithm			
Case	Pattern Type	Pattern	Time
1	Many Patterns	“Shevchenko Pl & E 7 St”	10779624 microsecond
2	Many Patterns	“Broadway & W 29 St”	10864922 microsecond
3	Many Patterns	“Cadman Plaza West & Montague St”	10777935 microsecond
4	Many Patterns	“E 47 St & 1 Ave”	15696 microsecond
5	Many Patterns	“Lexington Ave & E 26 St”	2506316 microsecond
6	Timestamp - Digits	“2014-07-01 00:05:06”	10791227 microsecond
7	Timestamp - Digits	“2014-06-01 00:16:12”	10795988 microsecond
8	Timestamp - Digits	“2014-06-28 13:11:12”	2589611 microsecond
9	Timestamp - Digits	“2014-06-01 00:08:42”	2542778 microsecond
10	Timestamp - Digits	“2014-07-01 00:00:57”	2524888 microsecond
11	Very Long	“Elizabeth St & Hester St””,””40.71729””,””-73.996375””,””250””,””Lafayette St & Jersey St”	10801142 microsecond
12	Very Long	“Clark St & Henry St””,””40.69760127””,””-73.99344559””,””274””,””Lafayette Ave & Fort Greene Pl”	10811950 microsecond
13	Very Long	“Broadway & E 14 St””,””40.73454567””,””-73.99074142””,””459””,””W 20 St & 11 Ave””,””40.746745”	2511660 microsecond

14	Very Long	“Howard St & Centre St””,””40.71910537””,””-73.99973337””,””507””,””E 25 St & 2 Ave”	2516286 microsecond
15	Very Long	“Watts St & Greenwich St””,””40.72405549””,””-74.00965965””,””358””,””Christopher St & Greenwich St”	2752474 microsecond
16	Very Short	“ab”	10657736 microsecond
17	Very Short	“ca”	10747151 microsecond
18	Very Short	“ad”	2476132 microsecond
19	Very Short	“sc”	2502266 microsecond
20	Very Short	“West”	31244 microsecond
21	Not Exists	“bbb”	10686658 microsecond
22	Not Exists	“bab”	10691513 microsecond
23	Not Exists	“dca”	2512482 microsecond
24	Not Exists	“East”	2502403 microsecond
25	Not Exists	“dme”	2580395 microsecond

Table 3. Performance results of the Boyer-Moore- Horspool algorithm

Boyer-Moore- Horspool Algorithm			
Case	Pattern Type	Pattern	Time
1	Many Patterns	“Shevchenko Pl & E 7 St”	2680581 microsecond
2	Many Patterns	“Broadway & W 29 St”	2700258 microsecond
3	Many Patterns	“Cadman Plaza West & Montague St”	682834 microsecond
4	Many Patterns	“E 47 St & 1 Ave”	31208 microsecond
5	Many Patterns	“Lexington Ave & E 26 St”	589579 microsecond
6	Timestamp - Digits	“2014-07-01 00:05:06”	1575196 microsecond
7	Timestamp - Digits	“2014-06-01 00:16:12”	1831472 microsecond
8	Timestamp - Digits	“2014-06-28 13:11:12”	441392 microsecond
9	Timestamp - Digits	“2014-06-01 00:08:42”	490741 microsecond
10	Timestamp - Digits	“2014-07-01 00:00:57”	384700 microsecond
11	Very Long	“Elizabeth St & Hester St””,””40.71729””,””-73.996375””,””250””,””Lafayette St & Jersey St”	230807 microsecond
12	Very Long	“Clark St & Henry St””,””40.69760127””,””-73.99344559””,””274””,””Lafayette Ave & Fort Greene Pl”	220547 microsecond

13	Very Long	“Broadway & E 14 St””,””40.73454567””,””-73.99074142””,””459””,””W 20 St & 11 Ave””,””40.746745”	62485 microsecond
14	Very Long	“Howard St & Centre St””,””40.71910537””,””-73.99973337””,””507””,””E 25 St & 2 Ave”	78790 microsecond
15	Very Long	“Watts St & Greenwich St””,””40.72405549””,””-74.00965965””,””358””,””Christopher St & Greenwich St”	62495 microsecond
16	Very Short	“ab”	5185651 microsecond
17	Very Short	“ca”	5141319 microsecond
18	Very Short	“ad”	1213935 microsecond
19	Very Short	“sc”	1196737 microsecond
20	Very Short	“West”	15615 microsecond
21	Not Exists	“bbb”	5163754 microsecond
22	Not Exists	“bab”	5199936 microsecond
23	Not Exists	“dca”	1224822 microsecond
24	Not Exists	“East”	1246675 microsecond
25	Not Exists	“dme”	1219397 microsecond

4. Conclusion

In this study, an applied comparison of algorithms was carried out to explain the basic ideas and complexity of the algorithms. Knuth-Morris-Pratt, Rabin Karp and Boyer Moore Horspool algorithms were used for the study. Rabin Karp algorithm gave results close to Boyer Moore Horspool algorithm, but in no case better than Boyer Moore Horspool algorithm, only when the searched string is diverse, when the string is very short and when a string is searched that is not in the data set. The Rabin Karp algorithm is the slowest matching algorithm in all cases because it does not use a mismatch table. Boyer Moore Horspool was second in most cases as he used a mismatch table when there was any mismatch. The fastest algorithm was the Knuth-Morris-Pratt algorithm. Here, it is aimed to increase the accuracy by choosing algorithms with different structures. The algorithms used were examined and compared on the same data set. In the future, an up-to-date and more comprehensive review will be made by using new algorithms and data sets apart from the existing algorithms used in the study. In addition, by developing a faster and more efficient new algorithm for high number of string comparisons, significant progress can be achieved in many areas. As the string length increases, the memory savings become significant and the system slows down. It would be more efficient to consider a different algorithm with lower time complexity.

5. References

Alshahrani, A.M., Khalil, M.I., 2022. Exact and Like String Matching Algorithm for Web and Network Security. 2013 World Congress on Computer and Information Technology (WCCIT).

Anonymous. Index of bucket tripdata, https://s3.amazonaws.com/tripdata/index.html. Accessed: 29 January 2023.

Anonymous. Rabin - Karp Algorithm, https://www.programiz.com/dsa/rabin-karp-algorithm. Accessed: 29 January 2023.

Biçer, M., Zhang, X., 2019. An Efficient, Hybrid, Double-Hash String Matching Algorithm. 2019 IEEE Long Island Systems, Applications and Technology Conference (LISAT).

Hakak, S.I., Kamsın, A., Shıvakumara, P., Gılkar, G.A., Khan, W.Z., Imran, M., 2019. Exact String Matching Algorithms: Survey, Issues, and Future Research Directions. Special Section On New Trends in Brain Signal Processing and Analysis, 7, 69614-69637.

Islam, T., Talukder, K.H., 2017. An Improved Algorithm for String Matching using Index Based Shifting Approach. 20th International Conference of Computer and Information Technology (ICCIT), 22-24 December.

Kanuga, P., 2015. New Shift table Algorithm For Multiple Variable Length String Pattern Matching. 2015 International Conference on Circuit, Power and Computing Technologies [ICCPCT].

Mathur, T., 2022. KMP Algorithm, https://www.scaler.com/topics/data-structures/kmp-algorithm/. Accessed: 29 January 2023.

Navarro, G., 2001. A Guided Tour to Approximate String Matching. ACM Computer Surveys, 33(1), 31-88.

Pop, P. G., 2015. DNA Repeats Detection Using a Dedicated Dot-Plot Analysis. 2015 38th International Conference on Telecommunications and Signal Processing (TSP).

Saldana, F., 2021. The Boyer-Moore-Horspool Algorithm, https://www.encora.com/insights/the-boyer-moore-horspool-algorithm. Accessed: 29 January 2023.

Singla, N., Garg, D., 2012. String Matching Algorithms and their Applicability in various Applications. International Journal of Soft Computing and Engineering (IJSCE), 1(6), 218-222.

Tripathi, S., Pandey, A. K., 2016. Identifying DNA Sequence by using Stream Matching Techniques. 5th International Conference on System Modeling & Advancement in Research Trends, 25th_27'h November, 334-338.

Yuan, L., 2011. An Improved Algorithm for Boyer-Moore String Matching in Chinese Information Processing. 2011 International Conference on Computer Science and Service System (CSSS), 182-184.