A Fast Pattern Matching Algorithm Based on Middle Characters of Pattern String

Weiqiang Ma*, Fan Yang, Eryue Zhang
Department of Information Engineering, Henan Technology Institute, Zhengzhou 450042, Henan, China
**Corresponding Author.*

Abstract:
String pattern matching is one of the important string operation. At present, the pattern matching algorithm of strings mainly includes BF algorithm, KMP algorithm, and improved KMP algorithm, BM algorithm etc. Drawing from the study of current pattern matching algorithms and practical programming experience, this paper presents a refined pattern matching algorithm that leverages the middle character of the pattern string. The enhancement offered by this pattern matching algorithm lies in its departure from the traditional approach, which typically involves sequential comparison and matching starting from the first character. This paper introduces a fast pattern matching algorithm that focuses on the middle character of the pattern string. By locating this central character within the target string and initiating the comparison and matching process from there, the algorithm significantly decreases the number of character comparisons required, thereby substantially enhancing the efficiency of the matching process. After repeated verification by program practice, The algorithm reduces the quantity of character comparisons while achieving correct matching, improves the matching efficiency, and effectively reduces the time complexity of the algorithm.

Keywords: pattern matching, target string; pattern string, left pattern matching, right pattern matching

INTRODUCTION

Pattern matching [1] is a string localization search. There are two strings, string s and string t. Usually, string s is referred to as the target string and string t is referred to as the pattern string. Finding a substring in the string s that is equal to the string t is called string localization search, also known as pattern matching [2]. Successful pattern matching involves locating a specified pattern string t within a target string s, whereas unsuccessful matching occurs when the pattern string t is not found in the target string s.

At present, the main pattern matching algorithms for strings include Brute Force algorithm [3] (BF algorithm), KMP algorithm [4], improved KMP, BM algorithm, etc. The fundamental principle of the Brute Force (BF) algorithm involves employing an exhaustive search method [5]. This approach begins by comparing the first character of the target string with the first character of the pattern string. If the characters match, the algorithm proceeds to compare subsequent characters. In case of a mismatch, it shifts to the second character of the target string and restarts the comparison with the first character of the pattern string. This process is repeated, moving sequentially through the target string, until either a successful match is found or the matching attempt fails. The KMP algorithm introduces the next [] array [6] on the basis of the BF algorithm. When a comparison fails, the next comparison starts from the next [] position of the target string. The KMP algorithm achieves a reduction in the number of comparisons relative to the BF algorithm. The improved KMP [7] algorithm modifies the next [] array based on the KMP algorithm, introducing the nextval [] array [8], further decreasing the quantity of comparative assessments and improving algorithm efficiency. The BM algorithm is an exact character set matching algorithm that starts from the tail of a pattern string [9].

RESEARCH FOUNDATION

On the basis of research and program practice on existing pattern matching algorithms, this article proposes a pattern matching algorithm based on comparing the middle characters of pattern strings. This algorithm changes people's understanding of traditional pattern matching algorithms starting from the left character and comparing them sequentially. It adopts a method of searching for the middle characters [10] of pattern strings from the target string and comparing them left and right. The algorithm has been repeatedly validated through program practice. Compared with traditional pattern matching algorithms, a new string pattern matching algorithm according to the left and right matching algorithm of middle characters in pattern strings is proposed, and the efficiency of this algorithm is greatly improved compared to traditional pattern matching algorithms.

DESIGN AND IMPLEMENTATION OF PATTERN MATCHING ALGORITHM BASED ON MIDDLE CHARACTERS IN PATTERN STRINGS

Design Ideas for Pattern Matching Algorithm Based on Middle Characters in Pattern Strings

The core concept of the middle character matching algorithm for pattern strings involves initially identifying the middle character of the pattern string t. Subsequently, the algorithm searches for the same character (which could be singular or multiple instances) in the target string s that matches the middle character of the pattern string t, aiming to determine its position [11]. Compare left and right based on this position. Specifically, find the first character in the target string s that matches the middle character of the pattern string t. Upon identifying the position, the algorithm compares the character to the left of the middle character in the pattern string t with the corresponding left character of the initially found middle character in the target string s. If the left characters are all the same, then compare the right character of the middle character of the pattern string t with the right character of the first middle character position found in the target string s. If they are equal, the match is successful. If there is inconsistency in the left or right comparison of the first intermediate character found in the target string s, then search for the second character in the target string s that matches the middle character in the pattern string t. After determining the position, compare the left character of the middle character in the pattern string t with the left character of the second middle character found in the target string s. If it is the same, then compare the character to the right of the middle character of the pattern string t with the character to the right of the second middle character position found in the target string. If it is equal, the match is successful. If there is inconsistency in the second middle character comparison, repeat the above steps until the last character found in the target string s that is the same as the middle character of the t string is located. After determining the position, perform left and right comparisons. If these characters are equal, the matching is deemed successful; if they differ, the matching is considered unsuccessful. In the above comparison process, once any character position in the target string s that matches the middle character of the pattern string t is found, and the left and right comparison results are consistent based on this position, the matching is successful. If a discrepancy arises in aligning the position of the last character in the target string s that matches the middle character of the pattern string t, either to the left or right, this signals a failure in pattern matching, indicating that the matching process has not been successful.

Example Description of Matching Process Based on Pattern String Middle Character Pattern Matching Algorithm

Assuming the target string is s="aaabcdabcaada... a. d." and the pattern string is t="abcdabc", the execution procedure for the pattern matching algorithm that centers around the middle character of the pattern string can be outlined as follows.

Step 1: Determine the middle character of the pattern string t, which contains 7 (n=7) characters t [0], t [1], t [2], t [3], t [4], t [5], t [6] (this algorithm is implemented in C++language, and the array index [12] starts from 0). The middle character is t [n/2], which means t [3] ='d'. (If the pattern string t contains an even number of characters n, the middle character is still t [n/2], which does not affect the algorithm execution result), as shown in Figure 1.

$$\begin{array}{cccccccc} t: & t_0 & t_1 & t_2 & \boxed{t_3} & t_4 & t_5 & t_6 \\ & a & b & c & \boxed{d} & a & b & c \end{array}$$

Figure 1. Find the middle position character t [3] ='d 'in the pattern string t

Step 2: Find the first character in the target string s that is the same as the middle character 'd ' in the pattern string t, and determine its position. s [5] =='d', as shown in Figure 2.

Step 3: The left character of the middle character in the pattern string t matches the first character s [5] found in the target string s that matches the middle character t [3] ='d'. Using s [5] as a benchmark, sequentially compare all characters on the left side of the middle character t [3] in pattern string t with the first corresponding position character on the left side of s [5] found in the target string s that is the same as the middle character t [3] in pattern string t. When comparing, the characters adjacent to the middle character of the pattern string t are sequentially compared to the characters corresponding to the target string s. Because the middle character of the pattern string t in this example is t [3] ='d ', the first character in the target string s that is equal to the middle character t [3] of the pattern string t is s [5] ='d'. The initial comparison is t [2] and s [4]. If it is equal, then compare t [1] and s [3]. If it is equal, then compare t [0] and s [3]. If it is equal, then the middle character and all left characters t [3], t [2], t [1], t [0] in the pattern string t are the same as s [5], s [4], s [3], s [2] in the target string s. This indicates that there are consistent characters in the pattern string t from the first character to the middle position in the target string s, At this point, it can be considered that there is a left half match between the pattern string t and the target string s. As shown in Figure 3.

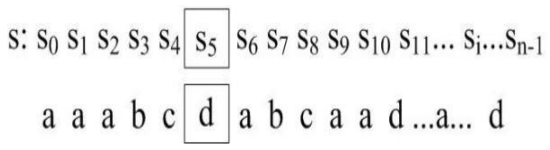

s: s0 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11... si...sn-1
a a a b c d a b c a a d ...a... d

Figure 2. Find the first character s [5] in the target string s that is the same as the middle character 'd ' in the pattern string t, where s [5] ='d'

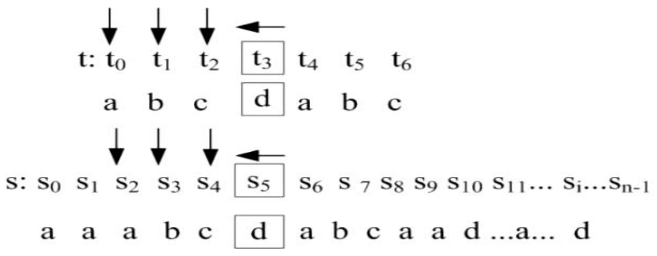

t: t0 t1 t2 t3 t4 t5 t6
a b c d a b c
s: s0 s1 s2 s3 s4 s5 s6 s 7 s8 s9 s10 s11... si...sn-1
a a a b c d a b c a a d ...a... d

Figure 3. Matching the left character of the middle character of pattern string t with the corresponding character of target string s

Step 4: The character to the right of the middle character in the pattern string t matches the corresponding character to the right of the first character found in the target string s that is the same as the middle character t [3]='d'. On the basis of matching the left character in the third step, using s [5] as the benchmark, compare all characters on the right side of the middle character t3=='d ' of the pattern string t with the corresponding position characters on the right side of the target string s [5]. When comparing, the characters adjacent to the middle character of the pattern string t are sequentially compared to the target string s to the right. The initial comparison is t [4] and s [6]. If it is equal, then t [5] and s [7] are compared. If it is equal, then t [6] and s [8] are compared. If it is equal, then the pattern string t starts from the middle position character to the last character and there is a consistent character in the target string s. It can be considered that there is a right half match between the pattern string t and the target string s. At this point, all characters in the pattern string t have been compared, and there are characters in the target string s that match the pattern string t, indicating successful pattern matching. As shown in Figure 4, compare the middle character d of the pattern string to the corresponding character of the target string to the right, and then compare t [4] with s [6], t [5] with s [7], t [6] with s [8].

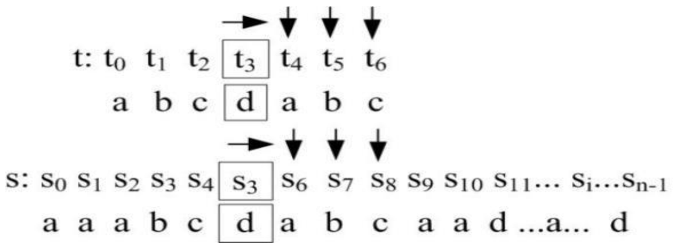

t: t0 t1 t2 t3 t4 t5 t6
a b c d a b c
s: s0 s1 s2 s3 s4 s3 s6 s7 s8 s9 s10 s11... si...sn-1
a a a b c d a b c a a d ...a... d

Figure 4. Matching the right character of the middle character in pattern string t with the corresponding character of the target string s

Step 5: The third and fourth steps above are the matching situation between the left and right characters of the middle character in the pattern string t and the first corresponding character in the target string s. If there is a left or right inconsistency during the execution of the third or fourth step, find the second character in the target string s that is the same as the middle character of the pattern string t, determine its position, and repeat the third and fourth steps above based on that character. If the i-th character found in the target string s is the same as the middle character of the pattern string t, and the corresponding characters in the third and fourth steps are consistent, then the pattern string t successfully matches the target string s. If all the characters in the target string s that match the middle character of the pattern string t do not match the left or right corresponding characters during the third or fourth step, the pattern matching between the pattern string t and the target string s fails, as shown in Figure 5.

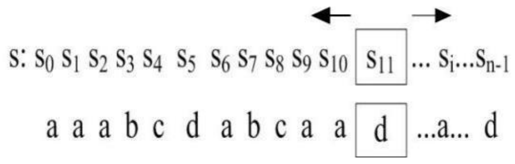

S: S0 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 ... Si...Sn-1
a a a b c d a b c a a d ...a... d

Figure 5. Find the second character in the target string s that is equal to the middle character of the pattern string and determine its position s [11] ='d '

Flowchart of a Fast Pattern Matching Algorithm Based on Middle Characters in Pattern Strings

The design concept and flowchart of a fast pattern matching algorithm based on the middle characters of pattern strings are displayed in Figure 6. Observing Figure 6, it becomes evident that at the beginning, the middle character t [3] of the pattern string t is searched for, with the flag bit bz=0 and the initial variable i=0. Starting from s [0] [12], the character s [i] that is equal to the middle character'd 'of the pattern string t is searched in a loop in the target string s. After finding it, the left character of s [i] is compared with the left character of the middle character of the pattern string t using s [i] as the benchmark (left and right matching need to be implemented in a loop, and the flowchart is simplified for easier understanding). If the left comparison match is successful, perform a right comparison match. If the left and right comparisons are consistent, bz==1, matching is successful, exit the loop, and output matching is successful. If the left or right matching is unsuccessful, then i+1, continue to the next round of matching. If i>StrLength (t) -1, it indicates that all characters in the target string s have been compared and the matching is unsuccessful.

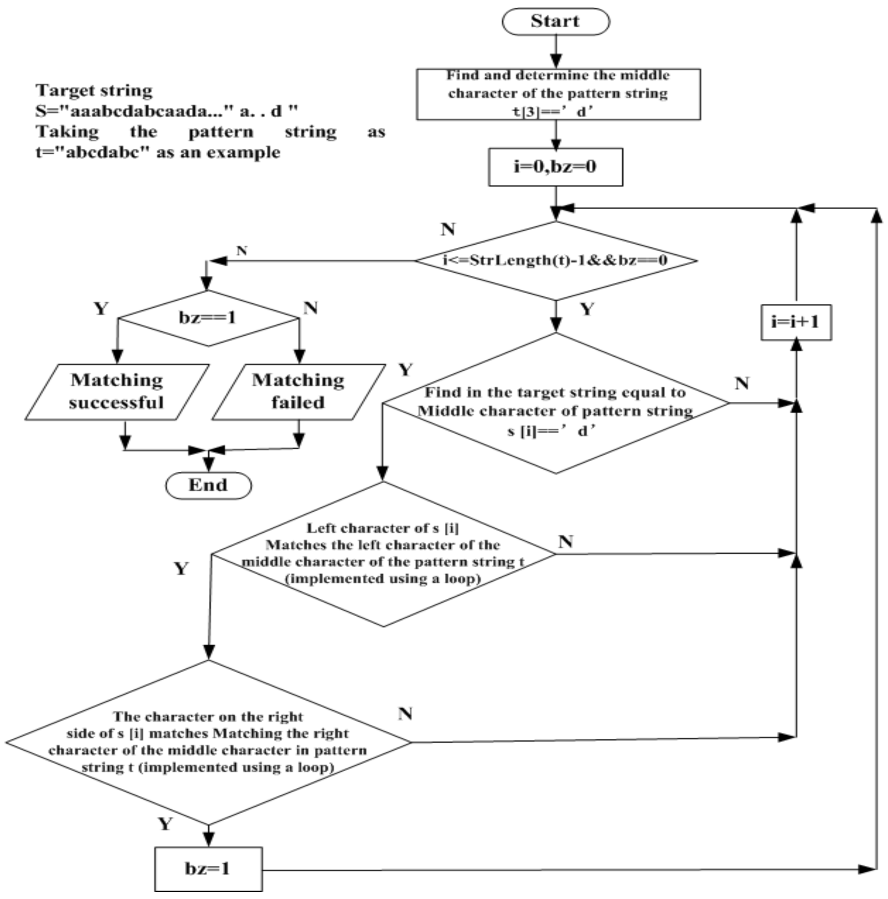
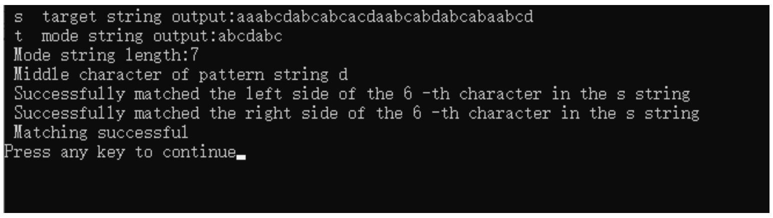

Start
Find and determine the middle character of the pattern string t[3]==' d'
Target string S="aaabcdabcaada..." a. . d "
Taking the pattern string as t="abcdabc" as an example
i=0,bz=0
N
i<=StrLength(t)-1&&bz==0
N
Y
bz==1
N
Y
Matching successful
Matching failed
End
i=i+1
Y
Find in the target string equal to Middle character of pattern string s [i]==' d'
N
Left character of s [i] Matches the left character of the middle character of the pattern string t (implemented using a loop)
N
Y
The character on the right side of s [i] matches Matching the right character of the middle character in pattern string t (implemented using a loop)
N
Y
bz=1

Figure 6. Flowchart of fast pattern matching algorithm based on middle characters in pattern strings

APPLICATION AND EXPERIMENTAL VERIFICATION OF ALGORITHMS

Algorithm Data Validation

According to the above flowchart, write the program code (implemented in C++language) and save it as *. cpp [13]. Execute it in the C++compilation environment [14] (VC++6.0 or VS, etc.). During the experiment, the target string data s="aaabcdabcabcabdacdacabaababcd" and the pattern string t="abcdabc". The execution result is provided in Figure 7. Observing Figure 7, it is evident that the program runs correctly.

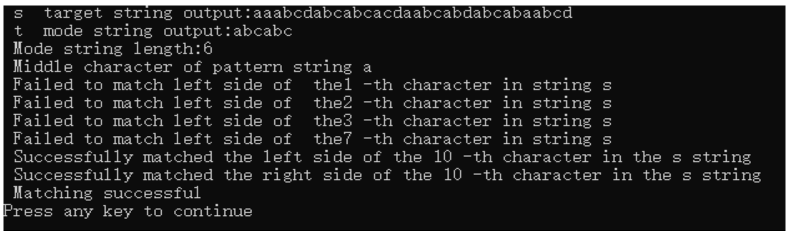


```
 s  target string output:aaabcdabcabcacdaabcabdabcabaabcd
 t  mode string output:abcdabc
 Mode string length:7
 Middle character of pattern string d
 Successfully matched the left side of the 6 -th character in the s string
 Successfully matched the right side of the 6 -th character in the s string
 Matching successful
Press any key to continue
```

Figure 7. Verification of program running results

After the program execution is completed, the pattern string t is outputted as a string length, and the middle character of the pattern string t is also outputted, indicating a successful match from the sixth character of the target string. Finally, the result of the successful match is outputted. The above data verifies that the pattern string t="abcdabc" has an odd number of 7 characters, and the middle character of the pattern string is the fourth character t [3] ='d ', which is easy to determine. How to determine if the pattern string is an even number of middle characters?

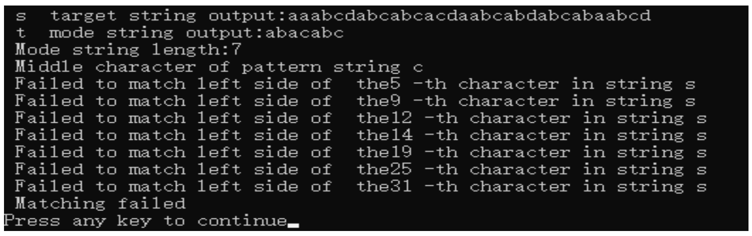
When writing code, the middle character of the pattern string is t [StrLength (t)/2]. The value of the middle position is half of the string length StrLength (t)/2 [15]; If the pattern string t="abcdabc" has an odd length of 7, with the middle character t= [7/2]=t [3]='d '(the result of 7/2 in C++language is 3 because the index of the C++language array starts from 0, and t [3] corresponds to the fourth character of the pattern string), and the pattern string t="abcabc" has an even number of 6 characters, the program running result is illustrated in Figure 8. Based on the execution result displayed in the figure, it is evident that the middle character of the pattern string t="abcabc" is t [6/2]=t [3]='a'. Find the character 'a' in the target string. Start comparing left and right when found. As depicted in Figure 8, the first, second, third, and seventh characters of the target string s are all 'a'. Once found, start comparing left and right. In the execution result, it can be seen that the left matching of the first, second, third, and seventh characters of the target string s has failed, The target string s is successfully matched around the tenth character, and the output matching is successful. The execution result is correct.



```
 s  target string output:aaabcdabcabcacdaabcabdabcabaabcd
 t  mode string output:abcabc
 Mode string length:6
 Middle character of pattern string a
 Failed to match left side of  the1 -th character in string s
 Failed to match left side of  the2 -th character in string s
 Failed to match left side of  the3 -th character in string s
 Failed to match left side of  the7 -th character in string s
 Successfully matched the left side of the 10 -th character in the s string
 Successfully matched the right side of the 10 -th character in the s string
 Matching successful
Press any key to continue
```

Figure 8. Validation of execution results when mode string t="abcabc"

The above verified data are all matched successfully, take t="abacadc", run the program execution result as shown in Figure 9, and the matching fails.



```
 s  target string output:aaabcdabcabcacdaabcabdabcabaabcd
 t  mode string output:abacabc
 Mode string length:7
 Middle character of pattern string c
 Failed to match left side of  the5 -th character in string s
 Failed to match left side of  the9 -th character in string s
 Failed to match left side of  the12 -th character in string s
 Failed to match left side of  the14 -th character in string s
 Failed to match left side of  the19 -th character in string s
 Failed to match left side of  the25 -th character in string s
 Failed to match left side of  the31 -th character in string s
 Matching failed
Press any key to continue
```

Figure 9. Verification of execution results for pattern string t="abacadc" matching failure

From Figure 9, it can be seen that the middle character of the pattern string t is' c '. Find all the characters' c' in the target string, which are the 5th, 9th, 12th, 14th, 19th, 25th, and 31st position characters of the target string. These characters are matched left and right, and the matching fails. Finally, the pattern matching fails. The algorithm execution result is correct.

Verification Program Description

The above data validation program is implemented in C++language, and the target string s and pattern string t in the algorithm are stored in sequence. The algorithm uses the generating string function StrAssign (SqString&s, char cstr []) [16], the string length function StrLength (SqString s) [17], and the string output function DispStr (SqString s) [18], which can be found in reference books. The above experiment is implemented using multiple functions in the program. Among them, there are mainly left side matching functions, right side matching functions, and pattern matching functions. The left matching function and the right matching function are similar to the string equality function [17]. The pattern matching function finds the middle characters of the target string s and the pattern string t in a loop and compares them left and right. The key to this algorithm is to save the position information when a character found in the target string s that matches the middle character of the pattern string t is unsuccessful in comparison. In the algorithm, variable i is used to store positional information, and the introduction of variable i enables the search of subsequent intermediate characters from unsuccessful matching positions for continued comparison. Both the left and right matching functions only use one loop, while the pattern matching function is a double loop. During program execution, the main function main () calls the pattern matching function, which in turn calls the left and right matching functions to complete the pattern matching. The algorithm uses multiple functions to facilitate program implementation, debugging, and reduce the workload of writing the main function main (). Interested readers can contact the author to obtain the algorithm source code. In addition, this algorithm has conducted extensive data testing on the pattern string t and the target string s during runtime validation, including the case where the pattern string t contains odd or even characters [19]. Both successful and unsuccessful matching can obtain correct results.

ANALYSIS AND ALGORITHM ADVANTAGES OF FAST PATTERN MATCHING ALGORITHM BASED ON MIDDLE CHARACTERS IN PATTERN STRINGS

A new pattern matching algorithm based on the middle character of pattern string is proposed, which first searches for characters in the target string S that are the same as the middle character of pattern string t, and then compares them left and right. The algorithm is innovative. Although this algorithm has an additional step of searching for intermediate characters, compared with traditional pattern matching algorithms, only some characters in the target string s participate in the assessment, rather than all characters participate in the comparison. This effectively reduces the number of loops and character comparisons, greatly improving the the algorithm's execution efficiency and reducing its time complexity.

Assuming the target string s contains n characters and the pattern string t contains m characters, the optimal time complexity of this algorithm is O (m), with an average time complexity of O (n×m/2), compared to traditional pattern matching algorithms, the efficiency has been improved.

CONCLUSION

Compared with traditional pattern matching algorithms such as BF and KMP, the middle character pattern matching algorithm has the following advantages: Firstly, this algorithm proposes a new pattern matching algorithm idea, which first searches for characters in the target string S that are the same as the middle character of the pattern string t, and then compares them left and right. Changed the traditional pattern matching algorithm's approach of comparing each character from left to right, and proposed a new algorithmic solution for pattern matching algorithms. The second is the fast pattern matching algorithm based on the middle character of the pattern string, which only compares the left and right characters in the same position as the middle character of the pattern string t. For the target string s, only some characters participate in the comparison, not all characters in the target string S participate in the comparison, greatly reducing the comparison characters and reducing the time complexity of the algorithm. Thirdly, in the process of comparing the middle characters of the pattern string t with the target string s, if there is inconsistency in the left comparison, the current comparison will be exited directly, and the right comparison will not be performed. Instead, the next character position in the target string s that is the same as the middle character of the pattern string t will be directly searched for comparison, which effectively reduces the number of loops and character comparisons, thus significantly enhancing the algorithm's execution efficiency and reducing its time complexity. In summary, the three major features greatly accelerate the efficiency of pattern matching.

REFERENCES

[1] Li Chunbao, Jiang Jingjue, Yu Dandan, et al. Data Structure Tutorial (Sixth Edition). Tsinghua University Press, 2022, PP 126-134.

[2] Dehghani H, Mhaskar N, Smyth W F. Practical KMP/BM Style Pattern-Matching on Indeterminate Strings. arXiv e-prints, 2022.

[3] Jiao Wenhuan, Feng Xingjie. Research on an Improved String Matching Model. Computer Simulation, 2022, PP 039-003.

[4] Yao Xiuqing. Discussion on the Improvement of KMP Algorithm. Digital Technology and Applications, 2020, 38 (4): PP 2.

[5] Kuljinder Singh Bumrah, Himani Sivaraman, Sandeep Budhani. A refined Algorithm for using Pattern with Variable Length by using the existing pattern matching. European Journal of Molecular & Clinical Medicine, 2021, PP 3.

[6] Yu Fei. Analysis and Research on Pattern Matching Algorithm. Computer Knowledge and Technology, 2018, PP 251-252.

[7] Yang Lingxue, Tian Hongbing. Discussion on the Explanation of KMP Pattern Matching Algorithm in "Data Structure". Science and Technology Consulting, 2019, 19, PP 196-197.

[8] Shao Lan, Tang Yongqun, Kong Lingshun. Research and Implementation of a String Matching Algorithm Based on the KMP Algorithm. Network Security Technology and Application, 2018, 12, PP 61-62.

[9] Wu Xihong. Performance Analysis of Improved QS Pattern Matching Algorithm. Computer Engineering and Applications. 2014, 50 (2).

[10] Zhao Xiao, He Lifeng, Wang Xin, et al. An Efficient Pattern String Matching Algorithm. Journal of Shaanxi University of Science and Technology, 2017, PP 184-185.

[11] Xingliang Yuan, Huayi Duan, Cong Wang, et al. Assuring String Pattern Matching in Outsourced Middleboxes. IEEE / ACM Transactions on Networking Volume 26, Issue 3. 2018. PP 1362-1375.

[12] Stephen Prata. C++ Primer Plus (Sixth Edition). Addison-Wesly, january 2012, PP 380-187.

[13] Tan Haoqiang. C++ Object Oriented Programming (3rd Edition). Tsinghua University Press, 2023, PP 242-245.

[14] Rick Mercer. Computing Fundamentals with C++ (3rd Edition). Franklin, Beedle & Associates, Inc 10 2017 PP 125-128.

[15] Mark Allen. Data Structures and Algorithm. Analysis in C++ (Fourth Edition). Addison-Wesley, 2014, PP 47-49.

[16] Jon Kleinberg, Eva Tardos Algorithm. Design lst Edition. Addison-Wesley 2005, PP 27-35.

[17] Anany Levitin. Introduction to the Design and Analysis of Algorithms (3rd edition). Published by Pearson, July 14, 2021.

[18] AbdulJabbar Safa S., Farhan Alaa K., Abdelhamid Abdelaziz A, et al. Razy: A String Matching Algorithm for Automatic Analysis of Pathological Reports. Axioms, Volume 11, Issue 10. 2022. PP 547.

[19] Krishnaveni Bommidi, Sridhar Sundaramurthy. A compressed string matching algorithm for face recognition with partial occlusion. Multimedia Systems. Volume 27, Issue 2. 2021. PP 1-13.