The Analysis of KMP Algorithm and its Optimization

The Analysis of KMP Algorithm and its Optimization

Xiangyu Lu[1]

[1]Department of Computer Science, Harbin Institute of Technology at Weihai, Weihai, Shandong, 264200, China

e-mail: angela@cas-harbour.org

Abstract. Knuth-Morris-Pratt (KMP) algorithm is an essential manifestation of matching algorithms. This paper presents and discusses the KMP algorithm and some of its optimization. Calculating and searching in a letter numbered table and a new data structure called last-identical array propose a new string searching algorithm L-I-KMP based on the KMP algorithm. This new method is faster than KMP in practice of specific situations.

1. Introduction

1.1. The invention of KMP algorithm

Knuth-Morris-Pratt (KMP) algorithm is one of the most efficient string matching algorithms in theory [1]. Along with the alarming rise of information quantity and complexity, human beings need to cope with a more considerable amount of data in the fields of information retrieval, DNA data matching, virus feature matching, data compression, etc. Within this context, string pattern matching has attracted much attention for a long time.

Pattern matching is one of the basic operations of strings. To illustrate, assuming that P is a given substring and T is a much longer string to be looked up, it is required to find all substrings that are the same as P in string T. In this case, string P is called pattern and T is called the target [2-3].

Under normal circumstances, when talking about pattern matching, the first attempt is to match the substring P with string T starting from every single element in T, which uses the brute force method. Although this algorithm conforms to the principle, its workload is enormous and cannot be efficient. By way of illustration, if we use f (T) for the length of T, supposing f (T) =n and f (P) =m, “length” means the number of all the letters in a string, when searching P from T, the number of letter matching operations it will take is to be m*(n-m+1). Then the KMP algorithm, an improved string matching algorithm was discovered by Knuth, Morris, and Pratt at the same time, and by avoiding superfluous comparisons of letters, it becomes relatively fast and popular today [4-5].

1.2. Limitation of KMP

Although this algorithm is relatively outstanding, it still has a considerable margin of improvement. As the pattern slides forward, there are still many matches that are not necessary. In addition, when the pattern first appears in the second part of the text, more comparisons will be needed [6]. In this paper, several existing optimizations of the KMP algorithm will be discussed, and a new method L-I-KMP algorithm is proposed.

2. Related Work

2.1. Original KMP algorithm

As Alzoabi et al. (2013) illustrated in their article, the KMP algorithm has been "considered as the first linear time string-matching algorithm with a serial cost of O(m+n)" [6]. The KMP algorithm traverses the given pattern string from head to tail, trying to find the longest common elements between the prefix and the suffix of each substring of the pattern, and take down the length of the common part in a "Failure Table", and the table should be of the same length to the pattern[7]. Each letter of the pattern in the Failure Table has a corresponding number to be calculated. Then comparison starts from the first letter of P and T, let N1 denotes the number of matched characters and updates with the comparison process, and represents "the corresponding number in the Failure Table of the last matched character" as N2. If the not-match situation starts from one letter of P, then the digits that P needs to move towards the right are: (N1-N2). The array "next" can be used to fulfil the identical functionality.

2.2. Existing optimization of KMP algorithm

The "Failure Table" mentioned above is also called PMT (Partial Match Table). However, for programming convenience, it is not used directly in general. Engineering usually shifts the PMT array one bit to the right and put the number "-1" to the first position. We call the new array "the next array". There are numbers of optimizations that start from the next array; most of them try to figure out the biggest jump that the pattern can do.

Professor Robert S. Boyer and J. Strother Moore presented BM Algorithm in the year 1977. It can be regarded as an improvement of the KMP algorithm on account of adopting part of the idea of the KMP algorithm and being more efficient than KMP algorithm in practice [8]. Some scholars consider it as the most efficient string matching algorithm in common applications [9]. Differentiating with KMP, BM algorithm starts comparing characters from the left of the pattern to the right, although the pattern and text are left alignment at first. When the comparison failed, there are two rules to determine the distance to the right that P needs to slide: "the bad character rule" and "the good suffix rule" [10]. The distance to slide is to be max {distance of "the bad character rule", distance of "the good suffix rule"}. Sunday algorithm is also a fast matching algorithm in practice [11]. In the matching process, it does not focus on the matching direction, but try to skip as many characters as it can when facing mismatch; moreover, it uses the next array as well, and quite like the BM algorithm. The worst computation complexity of Sunday algorithm is O(mn) [12].

The shift-add approach is based on finite automata theory and the limitation of the alphabet, combining ideas of KMP and BM algorithm [13]. This algorithm uses a vector of m in different states, and m is also the length of the pattern. S^j_i ($1 \le i \le m$) means the set of states after comparing the j-th letter of the pattern, and the value of S^j_i is the number of characters that are mismatching in the corresponding positions, if $S^j_m = 0$, then the pattern and text matches. The eKMP algorithm is proposed to enhance the efficiency of the KMP algorithm and its sophistication in detecting the disease DNA sequence, and this is done by continually updating the window size [14].

In 1987, Rabin and Karp presented the idea of hashing a pattern string and comparing its hash value with the text substring [15]. It can be used to detect plagiarism as it can handle multiple pattern matching. Even though this concept is inferior to the BF algorithm theoretically, its complexity can be comparatively low in practice, especially when using an efficient hash function. As KMP algorithm is only suitable for one-dimensional string matching when the Rabin-Karp algorithm is excellent at solving more dimensional matching problem. KMP+Rabin-Karp algorithm was proposed to achieve better efficiency [2]. KMP+Rabin-Karp algorithm combines these two algorithms to solve the two-dimensioned strings matching problem.

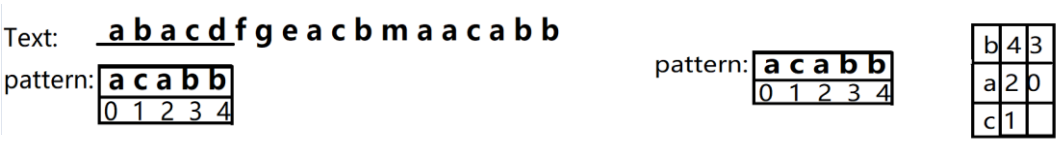
3. Proposed algorithm - L-I-KMP algorithm

3.1. Motivation of L-I-KMP algorithm

After knowing about several categories of optimal KMP algorithm, the author proposes another possible string matching algorithm by using the idea of the KMP algorithm to achieve a big move at one time. This new method refers to Sunday algorithm and makes use of a letter numbered table, and it uses a new data structure called last-identical array to decide the length of the move in some matching occasions. Similar to the KMP algorithm, the process is continued by referring to the data in the table when facing mismatches.

3.2. Preprocessing

Preprocessing is needed for this method. Before the match starts, it is necessary to analyze the data of the pattern, a letter table is being used to take down the letters and their position in the pattern, and the next same letter from the data structure can be directly queried according to the appearing number. The order of the position starts from the tail of the pattern. This idea is being illustrated in more detail in Figure 1. In Figure 1.a), the given pattern is "acabb", and the target text has been listed in the picture. It enables to browse from right to left and record each new character and its position during the process of preprocessing; this process is displayed in Figure 1.b).

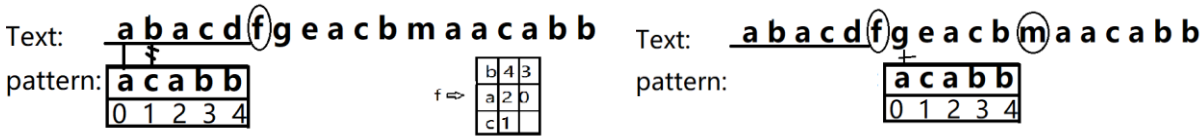


a) a pair of strings given b) make a letter table when preprocessing

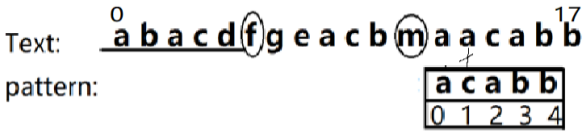
Figure 1. Example of L-I-KMP's preprocessing

3.3. The algorithm flow

This method is optimized with the Sunday algorithm after the first matching failure but is not precisely the same. In Sunday algorithm, the next step after mismatching is to see whether the next bit of the mismatched character is in the pattern, if not, the whole pattern can skip the character, moving a distance by "the length of matched letters" plus one step. While in L-I-KMP, it is necessary to focus on the next digit of the last bit aligned with the entire pattern string, which is A1 in this context. If A1 is not in the letter table, the amount of movement is the length of itself plus one. Otherwise, looking up the letter table to memorize the first number after A1, and this number, the author name it k, corresponds to the position of one same character in the pattern, call it A2. Align the two characters, and start comparison from the left of the pattern. When mismatches occur, refer to the table again to locate the next number after k, and align the new location of the pattern with A1. The same logic continues until there is no corresponding number for A1. The next bit of the last aligned bit can be found at this point, and thereby to name a new A1. Figure 2 interprets the algorithm flow by using the same example as Figure 1.



a) b)



c)

Figure 2. Example to show the algorithm flow

In Figure 2.a), after preprocessing, the first step matches. However, b ≠ c, so pattern[1] ≠ text[1], the next digit we need to consider is the sixth letter text[5] = f, while f does not exist in the pattern, the pattern can jump over "f", the result is in Figure 2.b). Then it is obvious that, g ≠ a, text[6] ≠ pattern[0], and text[6 + length of pattern] = text[11] = m is checked, and m does not appear in the letter table either. The whole pattern skips m.

In Figure 2.c), the pattern and the text does not match at pattern[1], and the last digit, text[12 + length of pattern] = text[17] = b, of the text is one component of the letter table. The numbers that follow it are 4 and 3 in order. The initiate activity is to drag the pattern and make pattern [4] align with the text [17], and then begin comparing from the left of the pattern. Fortunately, the whole pattern matched. Thus, there is no need to use the number "3" of the letter table any longer.

3.4. Optimization for coding convenience

For coding convenience while matching, refer to the format of the next array in the KMP algorithm and make use of the data from the letter numbered table, a last-identical array is introduced in Figure 3. In the last-identical array, the character in the pattern not only corresponds to its position number but also points to the previous position of the same letter, if this letter appears the first time from left to right, then it points NULL. In Figure 3, given the pattern "acabb" and its position array in order is "01234", its last-identical array is to be "NULL NULL 0 NULL 3".

pattern:

0	1	2	3	4
a	c	a	b	b
null	null	0	null	3

Figure 3. The last-identical array of the given pattern

When comparing from the left to the right, for instance, if pattern [4] = text [i], and I have aligned these two, and a non-match occurs in the previous digit, then it is to locate the next position equals to text [i] by looking up the affiliated number in the last-identical array of the pattern.

4. Performance comparison of algorithms

The times of attempt and the amount of letter's match during the whole matching procedure are two essential elements to analyze the capability of the matching algorithm [16]. The pseudo-code of the KMP algorithm is expressed in Figure 4.

```
KMP-Algorithm (T, P)
n <- length[T]
m <- length[P]
1: pi <- ComputePrefixTable(P)
2: q <- 0
3:  For  i := 0  To  n-1
4:      while q > 0  and  P[q] !=T[i]  do
5:          q <- P[q]
6:      if  P[q] = T[i]
                Then  q ++
```

```
7：    if  q = m
       Then return  i – m + 1
8：return –1
```

Figure 4. KMP Algorithm

When using the KMP algorithm, the first step is to preprocess of the Prefix-Table of P, that is, to calculate the "Failure table" or the "next array". Then step (2) initializes q to be the number of characters that matched, at the beginning of q=0. In step (3), it is time to start scanning the text from left to right. Then in step (4) (5) (6), the algorithm starts matching, if the two characters qualify the equation, the number of matched letters—q, is to plus one. Or if the two does not match, assign p a new value: q=P [q]. The pseudo-code of the new algorithm is displayed in Figure 5.

```
L-I-KMP algorithm（T, P）
N<- length[T]
M<-length[P]
1：Pi <-Compute letter table
2：Li<- Last-identical array
3：q <-  0
4：For  i <-  0  To  n-1
5：   while q > 0  and  P[q] !=T[i]  do
6：     i <-  i - q+ m
7：        q <-  0
8：        if  T[i] ∉ Pi Then
   q <-  0
9：        q <- P[q]
10：                    else if  T[i] ∈ Pi Then
11：              j<- 0
12：           do while Li[j] <> NULL
13：                        if  P[Pi[j]] = T[i] Then
                      compare T[i]~ T[i+m-1] with P[0]~P[m]
14：                      else j++
15：     if  P[q] = T[i]
               Then  q ++
16：     if  q = m
           Then return  i – m + 1
17：return –1
```

Figure 5. Pseudo-code of L-I-KMP algorithm

The table below illustrates the performance comparisons between the new method's pseudo code and KMP algorithm, in matching time.

Table 1. Performance comparison.

Data scale	Height of letter numbered table		KMP algorithm		L-I-KMP algorithm	
N=10, M=2	1	2	0.17	0.14	0.13	0.19
N=500, M=5	2	5	0.42	0.42	0.67	0.33
N=500, M=50	5	26	5.26	7.22	5.32	5.24

The matching information displayed in Table 1 obtained from repeated running with the same data generated randomly. According to the figures in the table, it is noticeable that L-IKMP algorithm performs well in practice when the types of letters are only a few in the pattern, and the same letters

are further apart. If the pattern is not so long, it can achieve a good speed similar to the original KMP algorithm. Therefore, this is an appropriate algorithm.

5. Conclusion

Through the experimental demonstration of string matching processing, it is not uneasy to discover that the efficiency of KMP and L-I KMP algorithms are almost equal, assuming that the amount of data is small. However, when the data set is large, and the number of types of the pattern is relatively big, or the number is small but unevenly distributed, the L-I KMP algorithm is superior to the KMP algorithm. This study confirms that string matching problems can be solved in practical applications. Additionally, the new algorithm can be improved in many aspects by optimizing the last-identical array or completing the comparing process with the letter numbered table.

References

[1] Knuth, D.E., Morris, J.H., Pratt, V.R. (1977) Fast pattern matching in strings. SIAM Journal on Computing, 6(2): 323-350.
[2] Chang, C., Wang H., (2012). Comparison of two-dimensional string matching algorithms. In: 2012 International Conference on Computer Science and Electronics Engineering. IEEE. pp. 608-611.
[3] Song Y.U., Zheng J., Hu W.X. (2009) Improved KMP algorithm, Journal of East China Normal University, 32(4): 92-97.
[4] Morris Jr, J.H., Pratt, V.R. (1970) A linear pattern-matching algorithm, University of California, Berkeley.
[5] Knuth, D.E., Pratt, V.R. Automata theory can be useful, unpublished manuscript.
[6] Alzoabi, U.S., Alosaimi, N.M., Bedaiwi, A.S., Alabdullatif, A.M. (2013) Parallelization of KMP string matching algorithm. In: 2013 World Congress on Computer and Information Technology (WCCIT). IEEE. pp. 1-3.
[7] Park, S., Kim, D., Park, N., Lee, M. (2018) High performance parallel KMP algorithm on a heterogeneous architecture. In: 2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS* W). IEEE. pp. 65-71.
[8] Boyer, R. S., Moore, J. S. (1977) A fast string searching algorithm. Communications of ACM, 20(10): 762-772.
[9] Yang, T., Wang, S.S., Qiao, X.D., Chen, Q. (2009) Analyze and improvement of BM algorithm. In: 2009 5th International Conference on Wireless Communications, Networking and Mobile Computing. IEEE. pp. 1-4.
[10] Hou, X.F., Yan, Y.B., Lu, X. (2010) Hybrid pattern-matching algorithm based on BM-KMP algorithm. In: 2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE). IEEE. pp. V5-310-V5-313.
[11] Sunday, D.M. (1990) A very fast substring search algorithm. Communications of ACM, 33(8): 132-142.
[12] Pan, G.H., Zhang, X.Z. (2012) Study on efficiency of Sunday algorithm. Journal of Computer Applications, 32(11): 3082-3084, 3088.
[13] Baeza-Yates, R.A., Gonnet, G.H. (1992) A new approach to text searching. Communications of ACM, 35(10): 74-82.
[14] Kalita, N., Sharma, R., Borah, S. (2015) eKMP: A Proposed Enhancement of KMP Algorithm. In: JAIN, Lakhmi C., et al., Computational Intelligence in Data Mining-Volume 3. Springer, New Delhi. pp. 479-487.
[15] Karp, R.M., Rabin, M.O., (1987). Efficient randomized pattern-matching algorithms. IBM Journal of Research and Development, 31(2): 249-260.