Speeding up motif Finding : KMP Algo

Natte Sai Bharath Department of Computer Science and Engineering Amrita Vishwa Vidyapeetham Amritapuri, Kerala, India
amenu4aie20052@am.students.amri ta.edu

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. This research was conducted to answer the problems of researchers for finding publication or papers that suitable with their topics. In this research was developed software for searchingabstract keyword using Knuth Morris Pratt algorithm to answer the problems of researchers. Waterfall model used to develop abstract keywords searching software as tools development that has five phases, namely communication, planning, modeling, construction and deployment. The software was developed can display search results effectively and efficiently according to the enter search keywords, it can see while search results are shown is in case sensitive. The software is also tested; the testing process is conducted by functional observation with black box testing approach, Observation results while testing is conducted show the software running suitable with expected or 100% same with entered keyword, so worthy to be used as one of the tools of researchers searching articles.**

I. Introduction (*Heading 1*)

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.

II. Literature Review

1) Firstly, the kmp algorithm solves the problem of string matching. For example, given two strings S="s1s2...sn" and T="t1t2...tm", the process of finding the substring T in the main string S is called string matching. Pattern matching

2) If you want to solve the string matching, it is actually an easy to solve problem, but as the scale increases, the time scale also becomes larger

Why is it easy to solve problems, such as the BF algorithm, that is, constant matching: the naive pattern matching algorithm

- Starting from the first character, it is compared with the first character of the pattern T. If they are equal, the subsequent characters of the two will be compared. If they are not equal, Then start from the second of the main string and compare with the first character of the pattern T ;
- BF is simple, that is simple and easy to understand, but there are many more steps, for example, if the index of the main string fails, it needs to be matched in order with the opening of the substring.

3)The emergence of KMP reduces the movement of the index. Its core idea is to keep the index i of the main string unchanged and move the index of the mode string; therefore, the time complexity increase relative to BF is huge.

III. Related Work

2.1. Original KMP algorithm

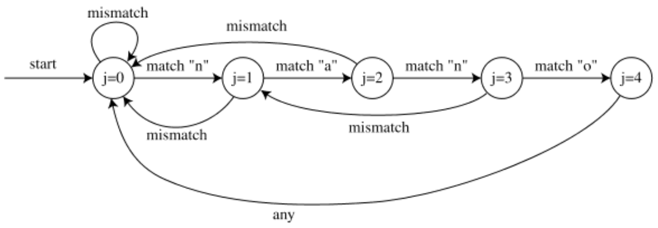
As Alzoabi et al. (2013) illustrated in their article, the KMP algorithm has been "considered as the first linear time string-matching algorithm with a serial cost of O(m+n)" [6]. The KMP algorithm traverses the given pattern string from head to tail, trying to find the longest common elements between 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≤i≤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 twodimensioned strings matching problem.

A. KMP and finite automata

If we look just at what happens to j during the algorithm above, it's sort of like a finite automaton. At each step j is set either to j+1 (in the inner loop, after a match) or to the overlap o (after a mismatch). At each step the value of o is just a function of j and doesn't depend on other information like the characters in T[]. So we can draw something like an automaton, with arrows connecting values of j and labeled with matches and mismatches.



The difference between this and the automata we are used to is that it has only two arrows out of each circle, instead of one per character. But we can still simulate it just like any other automaton, by placing a marker on the start state (j=0) and moving it around the arrows. Whenever we get a matching character in T[] we move on to the next character of the text. But whenever we get a mismatch, we look at the same character in the next step, except for the case of a mismatch in the state j=0.

IV. PERFORMANCE 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.For papers with more than six authors:

```
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].

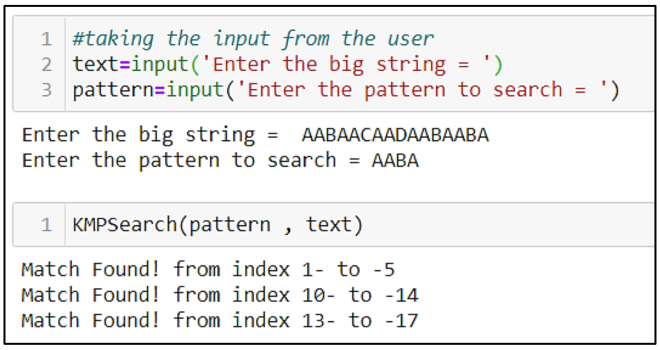
The table below illustrates the performance comparisons between the new method's pseudo code and KMP algorithm, in matching time.

Table 1. Performance comparison

Data scale	Height of letter numbered table		KMP algorithm	
N=10, M=2	1	2	0.17	0.14
N=500, M=5	2	5	0.42	0.42
N=500, M=50	5	26	5.26	7.22

V. RESULT AND DISCUSSION

Search software is software that can be used specifically to search publication either journal, proceedings or other research result. This software was developed to answer the problems that occur in researchers in search of publications as a research reference.
The software has the main features of publishing searches either journals, proceedings or other research based on titles, abstracts and abstract keywords. The software has been developed using waterfall model with five (5) development phase i.e. communication, planning, modeling, construction and deployment. Searching process performed on the software the Knuth Morris Pratt algorithm is called to perform the search process. The steps that the Knuth-Morris-Pratt algorithm



```
#taking the input from the user
text=input('Enter the big string = ')
pattern=input('Enter the pattern to search = ')
```

```
Enter the big string =  AABAACAADAABAABA
Enter the pattern to search = AABA
```

```
KMPSearch(pattern , text)
```

```
Match Found! from index 1- to -5
Match Found! from index 10- to -14
Match Found! from index 13- to -17
```

performs when matching the process as follows:
a. The Knuth-Morris-Pratt algorithm begins to match the pattern at the beginning of the text. From left to right, this algorithm will match characters per character pattern with characters in the corresponding text, until one of the following conditions is met:
i). The characters in the pattern and in the text that are mismatch.
ii). All characters in the pattern match. Then the algorithm will notify the invention in this position.
b. The algorithm then shifts the pattern by string, then repeats step 2 until the pattern is at the end of the text.
The KMP algorithm finds all occurrences of a pattern of length n in the text of length m with the time complexity of O (m + n). KMP algorithm requires only O (n) space of internal memory if text is read from external file. All O quantities do not depend on the size of the alphabet space.

Advantages

- The most obvious advantage of KMP Algorithm – data is that it's guaranteed worst-case efficiency as discussed.
- The pre-processing and the always-on time is pre-defined.
- There are no worst-case or accidental inputs.
- Preferable where the search string in a larger space is easier and more efficiently searched due to it being a time linear algorithm.
- The algorithm needs to move backward in the input text. This is particularly favorable in processing large files.

Disadvantages

- One of the glaring disadvantages of KMP Algorithm – data is that it doesn't work well when the size of the alphabets increases. Due to this more and more error occurs.
- For processing very large files it also requires resources in the form of processors and that could be a problem for smaller organizations to adopt KMP Algorithm – data

VI. Conclusion

In order to become efficient in learning search and Data Science broadly, it becomes extremely important to be familiar with various kinds of search algorithms. KMP Algorithm is the most widely used and most advantageous for organizations around the world due to its capacity of being the worst-case fault resistant algorithm. It's beneficial for anyone wanting to make a career in Data Science in general and Search in particular.The KMP algorithm is also a matter of dynamic programming. Our public account article directory has a series of articles that specialize in dynamic programming, and all are based on a set of frameworks. It is nothing more than describing the logic of the problem, clarifying the meaning of the dp array and defining the base case. That's a shit. I hope this article will give you a deeper understanding of dynamic programming.

VII. References

1. "9.1 Knuth-Morris-Pratt KMP String Matching Algorithm." *YouTube*, YouTube, 25 Mar. 2018, www.youtube.com/watch?v=V5-7GzOfADQ&ab_channel=AbdulBari.
2. Bird, Richard. "The Knuth–Morris–Pratt Algorithm." *Pearls of Functional Algorithm*

Design, pp. 127–135., doi:10.1017/cbo9780511763199.018.

3. Bird, Richard. “The Knuth–Morris–Pratt Algorithm.” *Pearls of Functional Algorithm Design*, pp. 127–135., doi:10.1017/cbo9780511763199.018.
4. “KMP Algorithm In Detail.” *Algo-En*, labuladong.gitbook.io/algo-en/i.-dynamic-programming/kmpcharactermatchingalgorithmindynamicprogramming#:~:text=Conclusion,) = O(M).
5. “KMP Algorithm: Complete Guide to KMP Algorithm with Examples.” *EDUCBA*, 1 Mar. 2021, www.educba.com/kmp-algorithm/.
6. “Knuth–Morris–Pratt Algorithm.” *Wikipedia*, Wikimedia Foundation, 29 June 2021, en.wikipedia.org/wiki/Knuth–Morris–Pratt_algorithm.
7. Lefarov, Max. “Pattern Search with the Knuth-Morris-Pratt (KMP) Algorithm.” *Medium*, Towards Data Science, 23 Dec. 2020, towardsdatascience.com/pattern-search-with-the-knuth-morris-pratt-kmp-algorithm-8562407dba5b.
8. “Python Program for KMP Algorithm for Pattern Searching.” *GeeksforGeeks*, 30 Dec. 2020, www.geeksforgeeks.org/python-program-for-kmp-algorithm-for-pattern-searching-2/#:~:text=Given a text txt[0,may assume that n > m.

VIII. Appendix

```
def KMPSearch(pat, txt):
    M = len(pat)
    N = len(txt)
    # create lps[] that will hold the longest prefix suffix
    # values for pattern
    lps = [0]*M
    j = 0 # index for pat[]
    # Preprocess the pattern (calculate lps[] array)
    computeLPSArray(pat, M, lps)
    i = 0 # index for txt[]
    while i < N:
        if pat[j] == txt[i]:
            i += 1
            j += 1

        if j == M:
            print ("Match Found! from index " + str(i-j)+"- to -"+str(i))
            j = lps[j-1]

        # mismatch after j matches
        elif i < N and pat[j] != txt[i]:
            # Do not match lps[0..lps[j-1]] characters,
            # they will match anyway
            if j != 0:
                j = lps[j-1]
            else:
                i += 1
def computeLPSArray(pat, M, lps):
    len = 0 # length of the previous longest prefix suffix

    lps[0] # lps[0] is always 0
    i = 1

    # the loop calculates lps[i] for i = 1 to M-1
    while i < M:
        if pat[i]== pat[len]:
            len += 1
            lps[i] = len
            i += 1
        else:
            if len != 0:
                len = lps[len-1]

                # Also, note that we do not increment i here
            else:
                lps[i] = 0
                i += 1
```