Modified Suffix Search Algorithm for Multiple String Matching

Jini Raju
Department of Computer Science and Engineering
Sree Buddha College of Engineering
Pattoor,Alappuzha
jiniraju07@gmail.com

Abstract

String Matching is now a prominent field in the area of Computer Science and it has many applications in the real world. A new algorithm for Suffix Search which uses chained hashing is proposed and this works well in matched case and mismatched case. A separate hash function is introduced in this paper. Hash functions can be declared in many ways. In this, radix hashing is used and the need of the shift table used in these algorithms can be avoided. Every pattern matching algorithm consists of mainly two phases. They are the preprocessing phase and the matching phase. Each of these phases has its own time complexity as well as space complexity. The proposed method has very low time complexity in average case.

Keywords: String Matching, Multiple String Pattern Matching, Modified Suffix Search

I. INTRODUCTION

String matching problem can be defined as the searching of all occurrences of patterns in a given text. A text of length n and a pattern of length m, where m<n, is the input of any string matching algorithm. Basically, string matching may be single string matching as well as multiple string matching. Single string matching searches the entire text for a single pattern. Multiple String matching finds all occurrences of pattern set, $P=\{p_1,p_2,\ldots p_n\}$ in the text T. The search procedure, which searches whether a pattern present in the text, must take minimum time and performs with no error. We can say that a pattern P occurs with shift s in text T if $0\leq s \leq n-m$ and T[s+1..s+m]=P[1..m]. If P occurs with shift s in T, we can call it as valid shift; otherwise it is called an invalid shift. The string matching problem is the problem of finding all valid shifts with which a given pattern P occurs in a given text T. Various String matching algorithm are presented in the literature survey. A new approach with hashing with chaining is introduced in this paper to solve some difficulties found in[1]. A hash table is a generalization of the ordinary array. The hash table operations take O (1) time. Hash table is used for minimal searching time, hence it is selected as the data structure for searching. Three algorithms are proposed by Khancome et.al in [1].They are Suffix Search, Suffix Prefix search and Suffix middle prefix search. In suffix search, the suffix of the pattern is taken and searches the text for the remaining prefix of the suffix. By using suffix search, the number of comparisons can be made lesser. These three algorithms work well for large pattern set and gives correct output in matched cases. Several erroneous outputs are produced for inputs that are not valid hits.

II. RELATED WORKS

Multiple string matching is a prominent field in many applications regarding cyber security and virus detectors. Aho-Corasick[2] algorithm is a simple and efficient algorithm that uses Trie structure to find all occurrences of patterns in a string of text. It provides a linear time solution and it constructs a finite state pattern matching machine from all patterns. The major phases of this algorithm are the construction of the finite state machine and application of the matching algorithm. The construction of pattern matching machine takes time proportional to the sum of the lengths of the keywords. The major disadvantage of Aho Corasick algorithm is that it has to search all characters in a text even if it is useful or not. It is possible to actually skip a large portion of text while searching, leading to faster than linear algorithms in average case. This algorithm serves as the basis for UNIX tool fgrep.

BM algorithm [3] is a fastest string searching algorithm which provides a sub linear solution. It scans from right to left and this provides fastest searching algorithm. An algorithm is presented by Boyer and Moore that searches for the location “i” of the first occurrence of a character string, “pat” in another string “string”. During the search operation, the characters of “pat” are matched starting with the last character of “pat”. The information gained by starting the match at the end of the pattern often allows the algorithm to proceed in large jumps through the text being searched. Thus the algorithm has a property that not the entire first i characters of string are inspected and this increases the performance. This algorithm uses bad character heuristics and good suffix and the preprocessing time takes $O(m+\sum)$ and searching phase takes O(mn) time, where m is the length of the pattern and n is the text length.

The Commentz-Walter[4] uses a Trie data structure in the preprocessing phase. It is a multi-pattern string matching algorithm that combines the shifting methods of BM with the Aho Corasick and it is faster than the above two. The first phase include the finite state machine construction and shift calculation whereas second phase includes the matching. Reverse Trie is constructed for FSM construction. The complexity is O(|t|+nocc) plus the additional time for access the trie where |t| is the length of the text and nocc is the occurrence of patterns. But Trie based algorithms takes longer time for searching.

Hash based approaches are very efficient for the pattern matching. The hashing algorithm was presented by Karp and Rabin[5] for single string matching. It takes O(m) for the preprocessing phase and O((n-m+1)m) time for searching phase. In this, all characters are interpreted as radix-d notations. Given a pattern P[1..m] and let p denotes the corresponding decimal value. Also a text T[1..n] is given and let ts denotes the decimal value of the length m substring T[s+1..s+m].If ts==p then T[s+1..s+m]=P[1..m] and s is a valid shift.

The efficient algorithm proposed by Wu Manber[6] is used for fast multiple string matching. It includes a shift table to store the values of all valid shifts and hash table to store the prefix of all patterns. The first step is the preprocessing of the set of patterns. Three tables are built in this stage. Shift table is used to determine how many characters in the text can be shifted when the text is scanned. If the shift value is zero, then the hash and prefix tables are used. In this algorithm, minimum length of pattern is taken and call it as m, then the first m characters of each pattern is considered. New hashing based algorithm[1] uses three efficient methods for multiple string pattern matching by imposing a restriction that all characters have equal length. Three algorithms are proposed by them and they are SS,SMPS,SPS. These algorithms use various hash tables in the preprocessing phase and algorithms for hashing is proposed in [1].

III. PROPOSED APPROACH

The existing method that uses the hashing based suffix search uses PPT table and SHIFT table in the preprocessing phase. The major drawback of the approach described in the algorithms of [1] can be illustrated using an example. The idea used in searching phase [1] for suffix search can be modified as follows. Assume that each pattern has fixed length, m. A window is sliding over the text string and gets the last character of the window and hashes it into the SHIFT table. If there exists the that suffix, then the string from first character to$(m-1)^{th}$ character in the search window gets hashes into the PPT table for matching. Thus the searching procedure completes and the window is shifted according to the shift value from the SHIFT table.

Example 1: The pattern set P={aaba, abcb, aadc, zmnd ,qope ,jmqf} and the text string T=aabaaabdabcd.The shift table contains the corresponding shift and PPT table contain the prefix of all patterns. The shift table includes all character in the pattern once and their corresponding shifts.

Table 1: Shifting Value

$P^i[m]$	Shifting Values
a	2
b	1
c	1
d	1
e	1
f	1
m	2
n	4
o	3
p	1
q	3
j	3
z	3
*	4

$p^i[1..m-1]$
aab
abc
aad
zmn
qop
jmq

STEP 1: **a a b a** a a b d a b c d

The suffix of the text is ‘a’ and checks whether the shift table contains a. The SHIFT table contains ‘a’ and then searches for the PPT for the prefix ‘aab’.Thus the pattern matches and shift two positions to the right.

STEP 2: a a **b a a a** b d a b c d .The shift value is retrieved but the prefix is not present in the PPT table and thus the window is shifted two times.

STEP 3: a a b a **a a b d** a b c d

The shift table contains the suffix 'd' and the PPT table contains the prefix 'aab' and reported a match according to the algorithm in [1]. But there exist no pattern 'aabd' in the pattern list. this is the major drawback of the existing algorithm. One more confusion exists in this is the use of hash function, No description of hash function is mentioned in the [1]. Because of these severe drawbacks, we modified the algorithms and techniques trough the introduction of a hash function and the use of a chained hashing.

The proposed system overcomes all the difficulties found in the existing system. The modification is made to the suffix search and this paper proposes the suffix search with chaining method. Also, the use of shift table is avoided and uses a linked list to store the prefix of patterns. A hash function is introduced in this method and the used method is the radix method. The radix-26 is used for each character. Two phases are used for the pattern matching-One is the preprocessing phase and the next is matching phase. The preprocessing phase find the suffix of each pattern in the pattern set and performs a radix hash function on that suffix. The hash(suffix) maps into the hash table that contains the corresponding shift and the head pointer of linked list. Thus the chain contains all patterns with suffix s.

Example 2: *P={aab**a**, abc**b**, aad**c**, zmn**d**,qop**e**, jmq**f** }*

hash(a)=1,hash(b)=2,hash(c)=3,hash(d)=4,hash(e)=5,hash(f)=6,hash(z)=26,hash(m)=13,hash(n)=14,hash(o)=15, hash(p)=16,hash(q)=17,hash(j)=10.Thus, 'a' points to 1st position of hash table, 'b' point to 2nd position and so on…Thus we can make sure that aab is the prefix of which suffix and thus improves its correctness. Another improvement we can made is that in order to reduce the searching time in linked list, move to front heuristics or count heuristics can be made. The algorithm for creating PPT for modified suffix search is in algorithm 2.The matching algorithm is slightly modified for the improvement in the suffix search. The suffix of the pattern is taken and performs the hash function to that character. If the hash table contains the corresponding suffix, then search the linked list for the prefix of that suffix value. If the linked list contains the pattern, then the match is reported. This algorithm can be used efficiently in case of short patterns in the pattern set. This modification algorithm works well in case of success and failure. The modified algorithm is given as algorithm 3.

IV. ALGORITHMS USED IN THIS SYSTEM

The algorithm for SHIFT table calculation is as same as that of [1] and it is described as follows.

Algorithm 1 : Create *ST*

Input: $P=\{p^1, p^2, p^3,...,p^r\}$

Output: Shifting Table (*ST*)

1. Initiate the empty *ST*
2. For *i=1* to *r* Do
3. *ShiftingValue=m*
4. for *j=m-1* down to *1* Do
5. If $p^i[j]$ exists in *ST* Then
6. *ShiftingValue* = the shifting value at $p^i[j]$ in *ST*
7. If $p^i[j] = p^i[m]$ Then
8. *ShiftingValue* = *ShiftingValue*-1
9. If *ShiftingValue* at $p^i[j]$ > *ShiftingValue* Then
   ShiftingValue at $ST[p^i[j]]$= *ShiftingValue*
10. End If
11. End If
12. Else
13. *ST*=$p^i[j]$ at column of ∑
14. add *m* to column of shifting Value
15. End For
16. End If
17. End For
18. Return *ST*

Algorithm2:PPT MODIFIEDSUFFIXSEARCH(P)

1. Find the suffix,s of p^i in the pattern set.
2 .Perform hash(suffix) using radix method.
3. The hash(suffix) maps into the hash table that contains the corresponding shift and the head of linked list.
4. The chain contains all patterns with suffix s.

Algorithm 3: ModifiedSuffixSearch(P)

1.Create PPT(P)
2.Cur=m
3.While Cur<=n
4 If hash[T[Cur]],then
5. Search the linked list for [T[Cur-m+1]…T[Cur-1]]
6. If found then
7. Print pattern matched at position Cur.
8 End If
9.End If
10.NewCur=Shifting Value at hash table
11.Cur=Cur+NewCur
12.End While

The major advantages of the system are that the use of the hash function makes the algorithm more effective and the avoidance of the shift table saves the memory space. The main highlight in this modified version is that it produces correct outputs in matched case and mismatched case and it is very useful in case of short patterns as well as long patterns.

V. CONCLUSION

A modified version of the hash based algorithm is presented here and it produces the correct output in all the cases. It does not produce false negatives of false positives. The usage of linked list may steal some amount of space but the output is 100% accurate. The system that is proposed matches the pattern with the text, but there exists an assumption that all patterns are of equal length. One of the future enhancements is the modification the algorithms presented here for incorporating patterns of different length. The concept of Wu Manber algorithm along with the hashing approach can make the proposed method for different pattern lengths. The patterns are divided into various blocks and these blocks are used to calculate the shift value and hash value. Also, another approach that has to be made in this method is to extend the Suffix Prefix Search and Suffix Middle Prefix Search to provide a faster running time.

References

[1] C. Khancome, and V. Boonjing, "New Hashing-Based Multiple String Pattern matching algorithms", Ninth International Conference on Information Technology,2012,pp.195-200.
[2] A. V. Aho, and M. J. Corasick, "Efficient string matching: An aid to bibliographic search", Comm. ACM, 1975, pp.333-340.
[3] R.S. Boyer, and J.S. Moore, "A fast string searching algorithm", Communications of the ACM, 20(10), 1977, pp.762-772.
[4] B. Commentz-Walter, "A string matching algorithm fast on the average", In Proceedings of the Sixth International Collogium on Automata Languages and Programming, 1979, pp.118-132.
[5] K. M. Karp, and M.O. Rabin, "Efficient randomized patternmatching algorithms" IBM Journal of Research and Development, 31(2), 1987, pp.249-260.
[6] S. Wu, and U. Manber, "A fast algorithm for multi-pattern searching", Report tr-94-17, Department of Computer Science, University of Arizona, Tucson, AZ, 1994.
[7] D.E. Knuth, J.H. Morris, V.R Pratt, "Fast pattern matching in strings", SIAM Journal on Computing 6(1), 1997, pp.323-350.
[8] Y. Hong, D. X. Ke, and C. Yong, "An improved Wu-Manber multiple patterns matching algorithm", Performance, Computing, and Communications Conference, 2006. IPCCC 2006. 25th IEEE International 10-12, 2006, pp.675-680.
[9] Z. A.A. Alqadi, M. Aqel, and I. M.M. El Emary, "Multiple skip Multiple pattern matching algorithm (MSMPMA)", IAENG International Journal of Computer Science, 34:2, IJCS_34_2_03, 2007.
[10] F. C. Botelho, "Near-Optimal Space Perfect Hashing Algorithms", The thesis of PhD. in Computer Science of the Federal University of Minas Gerais, 2008.
[11] R. Pagh, (11, 08, 2009) "Hash and Displace: Efficient Evaluation of Minimal Perfect Hash Functions". [Online]. Available: www.it.-c.dk/people/pagh/papers/hash.pdf.
[12] Wikipedia (10,07,2009)."Hash function". [Online]. Available: en.wikipedia.org/wiki/Hash_fuction.
[13] Simon, I. "String matching and automata", in Results and Trends in Theoetical Computer Science, Graz, Austria, J. Karhumaki, H.Maurer and G. Rozenerg ed., Lecture Notes in Computer Science 814, Springer- Verlag, Berlin. 1994, pp. 386-395.
[14] G. Navarro, "Improved approximate pattern matching on hypertext",Theoretical Computer Science, 2000, 237:pp.455-463.
[15] G. Navarro, and M. Raffinot, "Flexible Pattern Matching in Strings", The press Syndicate of The University of Cambridge. 2002.