Logical Matching Strategy: An Exact String Matching Algorithm

Sanil Shanker KP
Department of Computer Science,
Farook College, Kozhikode, India.
Email: sanil@farookcollege.ac.in

Abstract

String matching is a method of finding a pattern with some properties within a given sequence of symbols. This paper explains the characteristics of an exact string matching algorithm, the Logical Matching Strategy.

Keywords: Exact String Matching, Logical Matching Strategy.

I. INTRODUCTION

Much progress has been made over the last few decades in the applications of string matching especially in pattern matching. Exact string matching is the technique of finding the occurrence of a particular string, called a pattern, in another string called the text. A string \( x = x_1x_2......x_q \) is a substring of another string \( y = y_1y_2......y_n \) if and only if there exists an \( i, 0 < i \leq n \) so that \( y_{i+j-1} = x_j \) for \( j = 1, 2 ......q \). For example, the string 'THEM' is a substring of a string 'MATHEMATICS'. The substring of a string can be formed by deleting zero or more characters from the starting or the ending position of the string. Logical Matching Strategy is an exact string matching algorithm [1]. This paper explores the characteristics of Logical Matching Strategy.

II. REVIEW OF IMPORTANT STRING MATCHING ALGORITHMS

Characteristics of important string matching algorithms were studied and tabulated in Table- 1. The algorithms include, Boyer-Moore algorithm [2], Horspool algorithm [3], Knuth- Morris-Pratt algorithm [4], Karp-Rabin algorithm [5] and Quick-Search algorithm [6]. Here \( m, n \) and \( k \) are the length of the pattern, length of the text and number of alphabets respectively.

III. CHARACTERISTICS OF LOGICAL MATCHING STRATEGY

Logical Matching Strategy has two phases. The characters in the sequence pattern are pre-processed in the Phase I to generate the indices. That is, shift both the text and pattern either from right to left or from left to right so that each element in the pattern coincides with its corresponding index in its respective column. The information from pre-processing phase is used in the Phase II to match the indices of pattern with those of the text.

That is, let text \( T = t_1t_2......t_n \) and pattern \( P = p_1p_2......p_m \) be two strings of lengths \( n \) and \( m \) respectively from the same finite alphabet \( \Sigma \) such that \( m < n \).

Phase I. Generate indices of text and pattern

Phase II. Match the indices of pattern with those of text.

The method provides a way to find the exact locations of the pattern in the text. In Phase I of the method the computational time is linear and in Phase II the computational time depends on the length of the text.

IV. APPLICATIONS OF LOGICAL MATCHING STRATEGY

In 2011, Sanil et al demonstrated the method by locating the exact positions of the repeating DNA sequence pattern in the text sequence and also demonstrates the possibilities of alignment- free comparison of sequential pattern using Logical Match [7]. The paper entitled Sequential Data Mining using Correlation Matrix Memory explains a unique way of search for sequential pattern using correlation matrix memory [8]. Here, the concept of logical match is used to locate the pattern in the text.

V. SUMMARY

This paper explains the characteristics of Logical Matching Strategy in detail. The method can possibly be implemented to develop a way of research in sequential data mining.
Table 1: Characteristics of different string matching algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Comparison order</th>
<th>Pre-processing phase</th>
<th>Time (Pre-processing phase)</th>
<th>Time (Searching phase)</th>
<th>Main Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boyer-Moore</td>
<td>From right to left</td>
<td>Yes</td>
<td>O(m+k)</td>
<td>O(mn)</td>
<td>Use both good suffix shift and bad character shift. It is not very efficient for small alphabets. Some of the characters in the text can be skipped completely without comparing them with the pattern as it can be shown that they can never contribute to an occurrence of the pattern in the text.</td>
</tr>
<tr>
<td>KMP</td>
<td>From left to right</td>
<td>Yes</td>
<td>O(m)</td>
<td>O(n+m)</td>
<td>Independent from alphabet size. KMP avoids back tracking on the text when a mismatch occurs, by exploiting the knowledge of the matched sub-string in the text prior to the mismatch</td>
</tr>
<tr>
<td>Horspool</td>
<td>Is not relevant</td>
<td>Yes</td>
<td>O(m+k)</td>
<td>O(mn)</td>
<td>The algorithm is effective when the alphabet size is large and the length of the pattern is small as the shift value is computed in the pre-processing stage for all the characters in the alphabet set.</td>
</tr>
<tr>
<td>Karp-Rabin</td>
<td>From left to right</td>
<td>Yes</td>
<td>O(m)</td>
<td>O(mn)</td>
<td>Using hashing function. Effective for multiple pattern matching in one dimensional string matching.</td>
</tr>
<tr>
<td>Quick search algorithm</td>
<td>Is not relevant</td>
<td>Yes</td>
<td>O(m+k)</td>
<td>O(mn)</td>
<td>It uses only the bad character shift. It is very fast for short pattern.</td>
</tr>
</tbody>
</table>

REFERENCES


