Abstract

The Boyer- Moore algorithm searches for the characters of sequence from right to left beginning with the rightmost one if there occurs any mismatch it uses predefined functions to shift from left to right. If it finds the character in the last place it shifts the previous place and checks the first place of the string.

The Knuth- Morris Pratt searches for occurrences string from left to right and shifts more than one position, to do that it first preprocess the pattern in order to find the position for next matches. When it finds match it stops the search, in case of mismatch it skips unusual comparisons to avoid recomputing matches.

1 Introduction

As in the naive algorithm, the Boyer-Moore algorithm successively aligns \( P(\text{pattern}) \) with \( T(\text{text}) \) and then checks whether \( P \) matches the opposing characters of \( T \). Further, after the check is complete, \( P \) is shifted right relative to \( T \) just as in the naive algorithm. The Boyer-Moore algorithm contains three different steps which are not contained in the naive algorithm—**the right to left scan, the bad character shift rule and the good suffix rule**.

The main feature of boyer moore algorithm is to perform the comparisons from right to left and has the preprocessing in \( O(m + \sigma) \) time and space complexity. It takes \( O(mn) \) time complexity in searching phase. The best case would be \( O(n/m) \).

The Knuth Morris algorithm performs the comparisons from left to right and has a preprocessing phase in \( O(m) \) with space and time complexity. The searching time would be \( O(n+m) \).

1.1 Applications

- The Boyer moore is used in text editors for search and substitute commands.
• It works best when the alphabet is moderately sized and the pattern is relatively long.

• The Knuth morris algorithm is used for binary strings.

2 History

2.1 Boyer-Moore

The Boyer Moore string search algorithm is an efficient string searching algorithm which is the standard benchmark for string search literature. It was developed by Robert Boyer and J Strother Moore in 1977. Sometimes string matching algorithms will have one or several strings within a larger string or text. The maximum number of comparison was showed not more than 6n; in 1980 it was showed to be no more than 4n, until Sept 1991 which was given by Cole’s.

Boyer and Moore also worked on Boyer Moore automated theorem in 1992. It performed the character comparisons in reverse order to the pattern to be searched for and it will not search the entire pattern if it finds any mistake.

2.2 Knuth-Morris Pratt

The Knuth Morris Pratt algorithm was perceived in 1974 by Donald Knuth and Vaughan Pratt, independently by James H. Morris and published it in 1977. There are many algorithms like Brute force string search and Rabin-Karp are not very effective algorithms, because brute force string matching is slow and it was improve by using hash function in Rabin-Karp algorithm. But the complexities of both algorithms are same that is $\mathcal{O}(mn)$. In brute force we look at each character of the text with in the first character of the pattern, if we their is any mismatch we have to go several positions back in the text. In case of a match we simply shift the comparisons between the second character of the string nad the next character of the text.

3 Time complexities

3.1 Boyer-Moore

• Boyer Moore requires more space but takes $2 \times N$ steps and incase of worst case it takes $3 \times N$ steps.

• The Best case performance of the algorithm, for length n and a fixed pattern of length m, is $\mathcal{O}(n/m)$ in the best case, only if one in m characters needs to be checked.

• The Worst case is to find all the possible occurrences in the text that needs approximately $3n$ comparisons. Hence it takes $\mathcal{O}(n + m)$
3.2 Knuth Morris Pratt

- The Knuth Morris Pratt needs time for preprocessing. The preprocess of the pattern can be done in $o(m)$, where $m$ is the length of the pattern and the search needs $\Theta(n + m)$.

4 Working of the algorithms

4.1 Boyer-Moore

- First, the algorithm starts comparing the pattern from leftmost of the text and moves to the right.

- It compares pattern from right to left.

- The main idea of boyer moore is to improve the performance, and to do so the algorithm follows two steps they are:

  - **Good-suffix** and **Bad character shifts**

    - In good suffix shift it looks for character match if any mistach occurs it shifts the pattern to next occurrence of character.

    - If there exists any pattern which overlaps with another part of the character of same then shift to second portion of first occurrence.

    - Mostly a substring of the pattern may reoccur with the characters.

    - In a bad character shift, if any mismatch occurs it skips the comparison of the character in the text if it does not exists in the pattern

    - Boyer-Moore does good suffix and bad character shifts to improve the searching performance. At mismatch maximum of both is expressed to shift the pattern to the right.
5  **An example of Boyer Moore**

<table>
<thead>
<tr>
<th>Pattern</th>
<th>x</th>
<th>a</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>x</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(x) )</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>-1</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. Try to match

2. **EXAMPLE**

3. **HERE IS A SIMPLE EXAMPLE**.

4. Let us start by looking at \( S \) in the string which is last character under the pattern "HERE IS".

5. It is not a match because \( S \) is not matching with \( E \).

6. Next slide down the pattern by 1 we still do not find a match because \( S \) is not \( L \).

7. Since \( S \) does not occur in the pattern we move to right end of the string \( P \) to match pattern with \( P \).

8. Align the \( P \)’s but focus on the end.

9. \( E \) in the string matches with \( E \) in the pattern.

10. \( L \) is an another match with \( L \) in pattern so does \( P \) and \( M \).

11. Now \( I \) does not match with \( I \) in the string.
Figure 3: Knuth Morris algorithm

12. Shift the MPLE matched text to align un discovered occurrences with its last occurrence in the pattern.

13. Finally look for the P on the end of the string.

14. We found that EXAMPLE is matching with last occurrence of string EXAMPLE.

15. It can find the pattern without looking at all of the characters we looked past.

6 Knuth Morris

- This algorithm skips few comparisons by moving ahead to the next possible position of a match in the pattern.

- The Knuth Morris algorithm overcomes the disadvantages of Brute force string matching algorithm.

- Here, it uses the preprocessing of the pattern in order to avoid searching entire string.

- If the pattern has different characters, in case of a mismatch it starts comparing from first character.

- If any character is repeated in the string and there is a mismatch after that character then the algorithm starts comparing from the next place in given string.

6.1 An example of Knuth Morris Algorithm

- String: bachababab
- Pattern: ababab
1. First it compares P[1] with S[1].
4. At P[1] there is a match with S[2], since there is a match at S[2] with P[1] it does not move it's position.
11. To calculate the total number of shifts we need to subtract total number of string length to number of shifts.

7 Algorithms

7.1 Boyer Moore Algorithm

{Preprocessing stage}
Given the pattern P,
{Search stage}
k := n;
while k <= m do
end while
begin
i := n;
h := k;
while i > 0 and P(i) = T(h) do
begin
i := i - 1;
h := h - 1;
end
end

bacbabababacaab

ababab
end while
if $i = 0$ then
    begin ♦ report an occurrence of $P$ in $T$ ending position at $k$.
end if
else ♦ Shift $P$ (increase $k$) by the maximum amount determined by (extended bad character rule and the good suffix rule.
end;

7.2 Knuth Morris Pratt Algorithm

function $K(M)$
    $m = 0$ ♦ an integer, the beginning of the current match in $S$
    $i = 0$ ♦ the position of the current character in $W$
    $T$ ♦ Table
    while $m + i < \text{length}(S)$ do
        if $W[i]KM = S[m + i]$ then
            end if ♦ $i =$ the $n$ characters and $P$ is the pattern with $m$
        if $i = \text{length}(W) - 1$ then
            return $m$
            let $i = i + 1$
        else
            let $m = m + i - T[i]$
        end if
    end while
    if $T[i] > -1$ then
        let $i = T[i]$
    else
        $i = 0$ ♦ if we reach here, we have searched all of $S$
    end if
end function

8 Figures
Figure 4: Tree representation of boyer moore algorithm

Figure 5: Graphical representation of string searching algorithms