IMAGE PROCESSING:PATTERN RECOGNITION

SUJANA GURRAM
Indiana State University
Terre Haute IN, USA

December 14, 2011

Abstract

This project is to find pattern in a text file using BOYER-MOORE ALGORITHM(String Searching Algorithm).

Image Processing describes the qualities and measurements of images of objects. Pattern Recognition is concerned with description and classification of entities.

1 Introduction

In String matching pattern recognition the original problem is so easy to state that even a child could undersatnd it. Two strings of characters are given: one is called Text and the other is Pattern. Here the basic problem is to find all occurances of the pattern in the Text. And usually the Text is longer than Pattern.

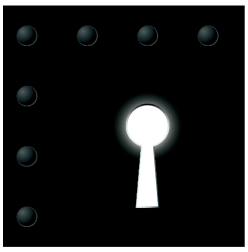

Figure 1: Search

Searching for a simple string(or)pattern in a text file is one of the standard problem in the old days of computing. Boyer-Moore Algorithm promulgated in 1977 was the major algorithmic breakthrough which is a new exact pattern matching algorithm that had excellent, distinguished logic to existing versions.This Boyer-Moore Algorithm performs character comparision in reverse order to the Pattern being searched for in the Text file. And if any mismatch occurs or found it had a method that doesn't require the full Pattern to search.

However Boyer-Moore algorithm contains three inventive ideas which or not contained in algorithms like for example Naive String Searching Algorithm:

1. Right to Left shift scan,
2. Bad Character Shift rule,
3. Good Suffix shift rule.

And we do have a Two-Dimensional pattern matching algorithm independently designed by Bird and Baker for a two-dimensional pattern matching problem that combines the use of the Aho-Corasick algorithm and KMP algorithm.

2 History

The distinct way to solve the string matching problem is to slide pattern along the text and comparing it with corresponding part of the text. Which is called Brute-Force Algorithm, which take O(nm) time. where n is the text and m is length of pattern. And by speeding up the Brute-Force Algorithm steps faster algorithms are obtained:

1. Comparision step which leads to Karp-Rabin Algorithm.
2. Sliding step by pre-processing the pattern, such as Knuth-morris-Pratt Algorithm.
3. Linear time versions which leads to Boyer-Moore Algorithm.

In the early 1956 string pattern recognition problems were formulated in terms of finite automata for long time. Solutions provided by applying these techniques are complicated and had high running time(designed by Kleene). Even Ukkonen's 1995-Algorithm for constructing suffix trees used finite automata.

However Gusfield showed, many of those algorithms may be reformulated without finite automate terminology. In 1970 Morris and Pratt came up with the first linear time algorithm to solve the problem. It skipped comparisions by

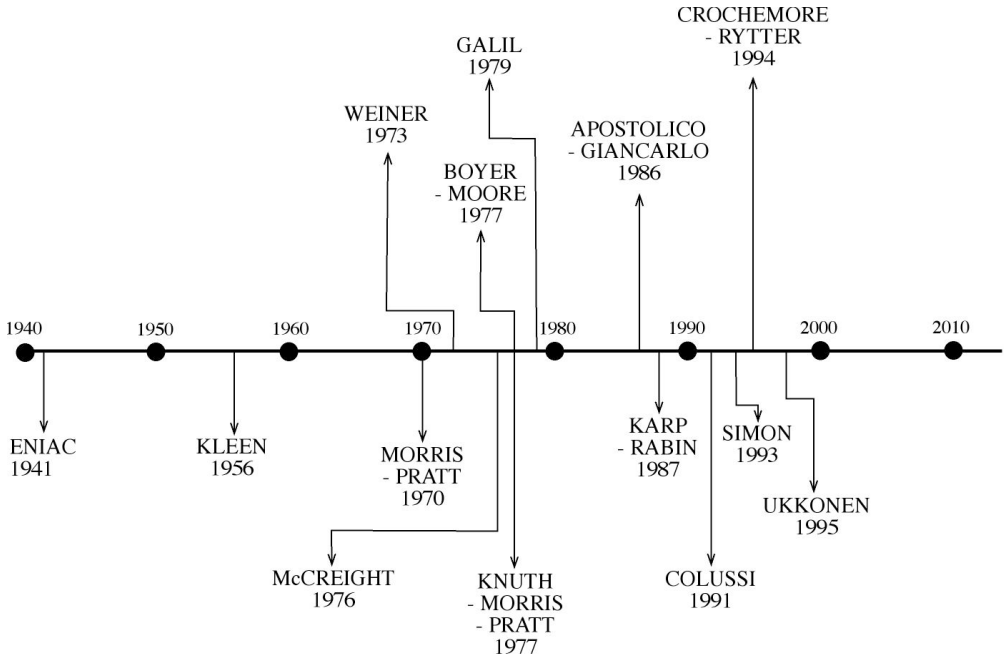

Figure 2: Timeline of the history of some string matching algorithms

studying internal structure of pattern. Seven years later, in 1977, Knuth, Morris and Pratt enhanced that algorithm. Although they achieved the same time complexity, their algorithm works much better in practice. A few years later, in 1991, Colussi pyt forward a enhancement of the Knuth-Morris-pratt algorithm that performs fewer comparisions in the worst case even. In 1977 Boyer and Moore proposed Quadratic Algorithm which is very much used in practice because of its good performnace. And even several enhancements followed Boyer-Moore Algorithm. The Galil Algorithm is an enhancement for periodic patterns that runs in linear time.

In 1987 Karp and Rabin published an algorithm that performs the comparision step by computing fingerprints of the pattern and the text, and if the set up is good its average case is like linear. In 1993 simon presented an economical implementation of finite automata running in linear time and he observed that only few edges are relevant in the automation. And a year later Crochemore and Rytter came up with a linear-time algorithm with a better performance thanthat of the simon algorithm. And for processing the text instead of the pattern, the first algorithm was designed by Weiner in 1973. Three years later McCreight gave a different algorithm, which reduced the amount of space used. And then Ukkonen gave a nice algorithm to build suffix trees.

3 Why Boyer-Moore and Not Other String Search Algorithms?

String matching problem idea is to find an occurance of a pattern in the text or to decide that none exists. For this problem of pattern matching the two best known algorithms are Knuth-Morris-Pratt Algorithm and Boyer-Moore Algorithm. Firstly to discuss about Knuth-Morris-Pratt Algorithm, KMP scans "words" in the forward direction and if we are scanning forward with KMP, and spotted the non-word charater, then we can only skip forward as far as we matched. So it is mostly and often considered impractical.If we consider Boyer-Moore Algorithm, BM scans in the backward direction. scanning backward with BM is faster because if we spot a character that isn't in the word then we can skip many spaces forward. Best-case, we may only need to compare the last character every time, and skip forward the full length word. This makes Boyer-Moore Algorithm as a choice in practical applications.

BM can be much faster than KMP, but only on certain inputs that allow many caharcters to be skipped. So BM can be faster or slower than KMP depending on the inp ut, while KMP is perfectly reliable O(n).

4 Two Dimensional Pattern Matching

The first worst-case linear time for 2D searching is due to Bird and, independently, to baker. It uses Aho-Corasick's multi-string searching algorithm as the main component, achieving O(n^2+m^2) searching time worst and average case. This algorithm needs O(n+m^2) extra space. And the basic idea is to run a finite automaton that searches for the m strings that form the pattern in every row, while at same time running the KMP string searching algorithm on every column, searching for the row indices of the pattern. And the probabilistic 2D pattern matching algorithms are proposed by Karp and rabin. However, since the constant in the running time O(n^2) is too large, these algorithms cannot be used in practice. In this case extra storage is either O(n) or O(m^2).

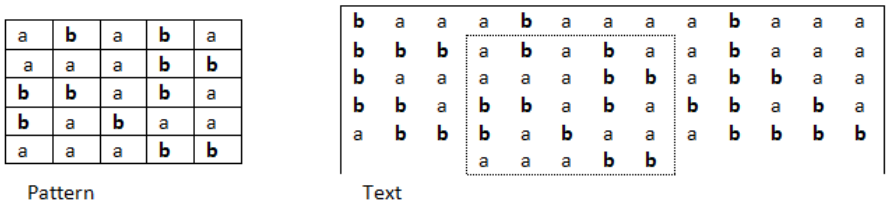

Figure 3: Simple Two Dimensional Searching example

5 Problem Statement

Boyer-Moore String Searching Algorithm avoids lot of needless comparisions by significantly shifting pattern relative to text. And Boyer-Moore Algorithm works fast when the alphabet(or)string is moderatley sized and the pattern is relatively long. And in all the algorithms we have seen so far preprecesses the textfile in which we had to find the pattern, because of which the process of being able to match the textfile with pattern became moderately slow resulting into inefficient running time. To overcome this we tried to implement Boyer-Moore String Search Algorithm in which it preprocesses the pattern itself avoiding excessive comparisions by acquiring the facts from previous fruitless attempts which yeilds efficient running time.

6 Major Reason For Boyer-Moore To Be Quick And Efficient

One of the main reasons for the high efficiency of the fast pattern matching algorithm of Boyer and Moore is PREPROCESSING. The Boyer-Moore's pattern matching algorithm uses two preprocessing algorithms:one based on string and the other on single characters only. In 1981 correctness of the single-character preprocessing algorithm has been established and mechanically verified by Boyer and Moore. String-preprocessing algorithm is more complicated and some errors were found by different authors.In this algorithm a minor error was found in 1999 and was corrected. And to make sure that corrected algorithm doesn't lead to any new errors a correctness proof for the corrected algorithm was developed.

Boyer-Moore on an avg is close to O(n/m) where pattern is of size m and text is of size n and KMP is O(n). This is faster because it compares words from right to left , and as soon as it finds a letter in text which is not present in pattern , search moves m places ahead . so for large files , it should be expected that a lot of such moves would be there so a lot of characters of text wont be accessed even once unlike KMP where each letter is accessed atleast once.

7 Time Complexity

Boyer-Moore Algorithm has a Time Complexity O(n/m) heedless whether the text file contains the perfect match or not. Where 'n' is length of text 'm'is length of pattern.

8 Algorithm

1. Boyer-Moore Algorithm is extremely fast on large Text File which are relative to length of pattern.
2. Boyer-Moore Algorithm perform comparisions from right to left or scans characters from right to left begining with right most character of string.
3. In case of mismatch it uses two pre-computed functions to shift the pattern characters to right.

   Two functions are:

. Bad Character Shift.

. Good Suffix Shift.

8.1 Bad Character Shift

In this method the pattern characters are compared with the text string from right to left. Now find for the text character that causes a mismatch with the pattern. Then find for the occurance of that text character somewhere else in the pattern. Then the pattern is shifted that is aligned to the mismatched text character. If the mismatched text character doesn't occur in the pattern shift the pattern to the position (m+1) where m is the length of the pattern.

Example: Indiana State University - Text File

Find the **University** - Pattern in the text file

Process to find the pattern(university) in the Text File(Indiana state university:

Step 1:

1. **Indiana State university** - Text File

   university - Pattern

(Now we have to compare rightmost last character in university with the text file, here 't'(in text file) and 'y'(in pattern) if mismatch we have to search correct match for 't' in the pattern if we find it we have to shift to right to match the text file character).

Step 2:

2. **Indiana State university**

   university

(Now again the same process repeats compare last rightmost charater 'y'(in pattern) with 'a' (in text file) and if the match is not found shift the whole pattern to right).

Step 3:

3. **Indiana State university**

   university

(repeat the same process. Here if we compare last characters from right 's' in (text file) and 'y' in (pattern) doesn't match so we have to search for letter 's' in pattern to match the 's' in text file. once we get the correct match we have to shift the pattern to right, so the resultant will be in the below form)

Step 4:

4. **Indiana State university**

   university

(resultant is pattern matched with the text file).

8.2 Good Suffix Shift

In Good Suffix Method the pattern characters are compared with the text file from right to left. And find the text character that causes mismatch with the pattern. Now consider the suffix characters in the pattern which matches with the text string for further process. Then find for the occurance of that matched suffix character sequence in the pattern. The pattern can be shifted untill the occurance of matched suffix character sequence is aligned to the text character which matched with the pattern.

Example: x y z x y x y - Text File

z x y x y - pattern

Proving that Pattern matches the Text File:

So in the above exmaple the suffix characters in the pattern matches with the text file. so if we consider the third charcter 'y' in pattern and 'z' in text file mismatch. Then we have to find for the matched suffix charcter sequence in the pattern. And we have to continue the process till we find the exact match for Pattern in the Text File. so now the resultant answer is displayed below:

x y z x y x y

z x y x y

References

[1] Dan Gusfield *Algorithms on Strings, Trees, Sequences* 1997.

[2] Graham A.Stephen *String Searching algorithms*