Faculty of Engineering and Natural Sciences

Department of Computer Engineering

CMP3005 - Analysis of Algorithms Term Project Report

Onur Güzel

August 14, 2020

Abstract

In the field of computer science, more specifically about algorithms, there are different approaches for dividing algorithms into groups. The main criteria is dividing it by the computational problem to be solved. One of the most useful groups of algorithms are the string matching algorithms. Those are algorithms for searching similar or equal strings of characters for different purposes. String matching algorithms are widely used in computer science for text correction, plagiarism detection etc. This report gives details of an algorithm developed with C++ using Brute-Force string matching technique.

1 Introduction

A basic method for matching substrings is to check every position of the pattern in the given text. This means checking every position in the text at which the pattern might have a positive match. However, it is not necessary that a position is a match, so the algorithm returns one of the two possible logical values: true or false. In our approach if there is a positive match at a position, it returns "1" alongside with the location of the matching string in the vector container; otherwise, it returns "0" and proceeds to the next character in the text.

2 Project Description

Two separate vectors are initialized for the statements and the text file. Before passing the input files to the vector container, "splitSentences" and "splitStatements" functions are called to place them word by word consecutively. Once the operations are completed a for loop is initialized to find the matching statements in the text and printing the the whole sentence including the missing part. This is done using "isEqual" and "NoOfMatches" functions. Basically algorithms takes the statement from the input file and goes through the text word by word until a match is found,which is defined by "isEqual" function. Upon completion of this operation "printAnswer" function is called to print the statement. This operation is performed until all of the statements has been processed. At the end elapsed time is printed to measure the efficieny.

```
23
24 int main(int argc, char const *argv[]) {
25
26
27     auto start = chrono::system_clock::now();
28     vector<string> statements = splitStatements("statements.txt");
29     vector<string> sentences = splitSentences("the_truman_show_script.txt");
30
31
32     int index;
33
34
35
36     for (int i = 0; i < statements.size(); i++) {
37
38         cout << statements[i] << endl;
39         index = NoOfMatches(statements[i], sentences);
40         printAnswer(statements[i], sentences[index]);
41     }
42
43     auto end = chrono::system_clock::now();
44
45     chrono::duration<double> elapsedSeconds = end  - start;
46     cout << "Run Time: "<< elapsedSeconds.count() <<" seconds"<< endl;
47
48     return 0;
49 }
50
```

Figure 1: Main Function

2.1 String Matching Method

Brute-Force string matching method is implemented by using the functions in the figures below;

```
195
196 bool isEqual(string& s1, string& s2) {
197     string shortString = s1.length() > s2.length() ? s2 : s1;
198     string longString = s1.length() > s2.length() ? s1 : s2;
199     for (int i = 0; i < shortString.length(); i++) {
200         if (shortString[i] == longString[i]) {
201             continue;
202         }
203         return 0;
204     }
205     return 1;
206 }
207
```

Figure 2: "isEqual" Function

```
100
101  int numberOfMatches(string& s1, string& s2) {d:
102
103      vector<string> s1Words = splitWords(s1);
104      vector<string> s2Words = splitWords(s2);
105
106      int matchCount = 0;
107
108      for (int i = 0; i < s1Words.size(); i++) {
109          for (int j = 0; j < s2Words.size(); j++) {
110              if (isEqual(s1Words[i], s2Words[j])) {
111                  s2Words.erase(s2Words.begin() + j);
112                  matchCount++;
113                  break;
114              }
115          }
116      }
117
118
119      return matchCount;
120
121  }
```

Figure 3: "numberOfMatches" Function

2.2 Time Complexity

Best Case: O(n)
Average Case: O(m+n)
Worst Case: O(m*n)

Where m is pattern's length and n is text's length

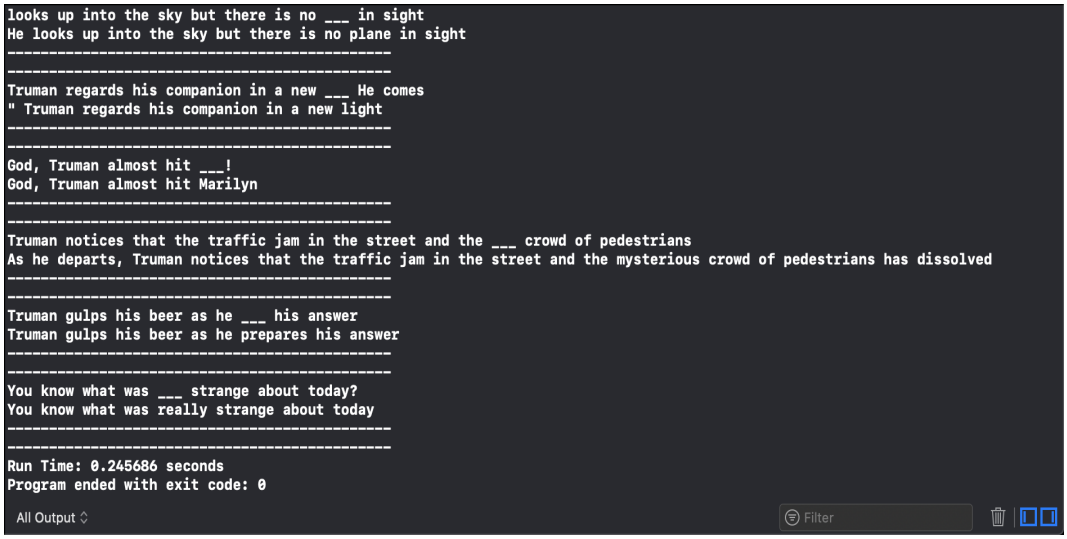

Figure 4: Run Time

3 Conclusion

Even though Brute-Force string matching algorithm is not the most efficient string matching algorithm to use in most of cases including this one, ease of implementation makes it well-suited for the basic computational problems. Basically, it returns good results when the text is short, the problem arises when the text is extremely long. An optimized version of it can provide better results depending on the problem to be solved. In worst case scenarios, the time complexity can rise to O(m*n).