Central Washington University

Advanced Algorithms

Winter 2019

Project 5 Report

Author:
Hermann Yepdjio

Professor:
Dr. Razvan Andonie

March 7, 2019

Contents

1 Introduction **2**

2 implementation **2**

2.1 Naive String Matching Approach 2

2.2 Aho-Corasick String Matching Approach 2

3 experimentation process **3**

4 Results **3**

4.1 Rlots for running time 3

4.2 Observation 5

5 Conclusion **5**

1 Introduction

The Aho-Corasick string matching algorithm is a fast and memory efficient algorithm invented by Alfred V. Aho and Margaret J. Corasick. Given a list of patterns to find in a given input text, the algorithm will locate all the occurrences of the patterns in the input text by matching them all at once instead of one at the time. The goal of this project was to implement in python both the Aho-Corasick algorithm and a naive approach (which matches the patterns one at the time using the python's string.find() function) and compare both implementations by running some experiments.

2 implementation

We implemented both algorithms so that they will find only the last occurrence of the patterns instead of finding all the occurrences. When a pattern is found, a message containing the pattern itself and information about its position in the text input, is printed on the screen.

2.1 Naive String Matching Approach

- The algorithm takes as input the name of the input text file and the list of patterns to look for,
- the text file is open and its whole content is read as a single string,
- the algorithm loops over the list of patterns calling the string.rfind() function on the text input with the current pattern as parameter at each iteration. The string.rfind(x) function is another version of string.find(x) which looks for the last occurrence of a in string instead of the first occurrence.

2.2 Aho-Corasick String Matching Approach

This algorithm uses the pyahocorasick package.

- The algorithm takes as input the name of the input text file and the list of patterns to look for,
- the text file is open and its whole content is read as a single string,

- an empty automaton is created using the pyahochorasick package and populated with every single word in the input text
- the algorithm loops over the list of patterns calling the automaton.get(pattern) function at each iteration, in order to find the last occurrence of the current pattern in the text input.

3 experimentation process

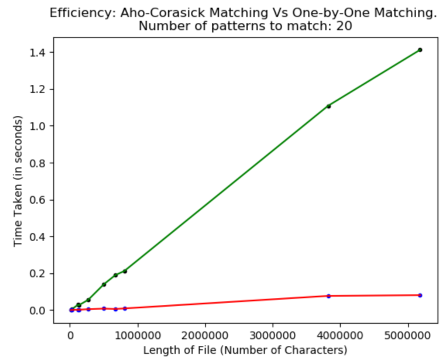
For the experimentation, we ran the algorithm using 10 files (for input text) of different sizes (ranging from 12324 to 5182632 words) and 7 other files (for patterns) of different sizes (ranging from 20 to 4941 words). We recorded the time taken for each algorithm to string match each pattern list to all the input text and the results are shown and discussed in the next section.

4 Results

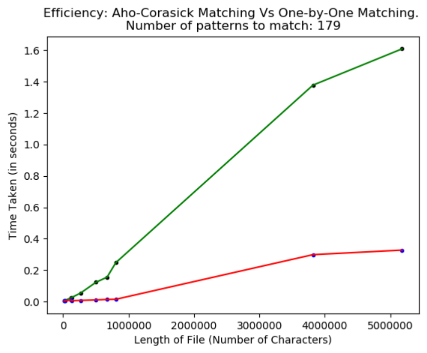
4.1 Rlots for running time

The figures below show the results we got during the experimentation process.

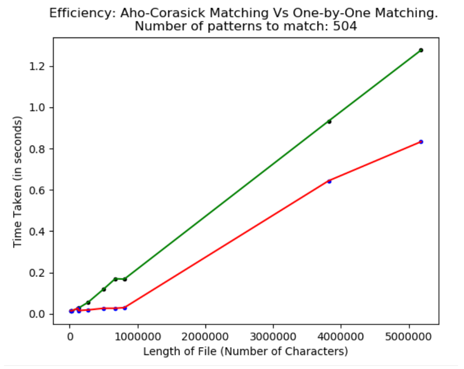
- The green curve represents the Aho-Corasick implementation
- The red curve represents the naive string matching implementation.



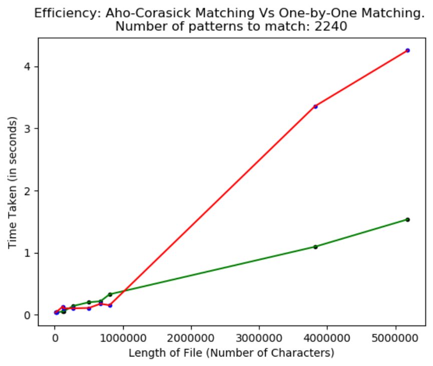
(a) number of pattern: 20



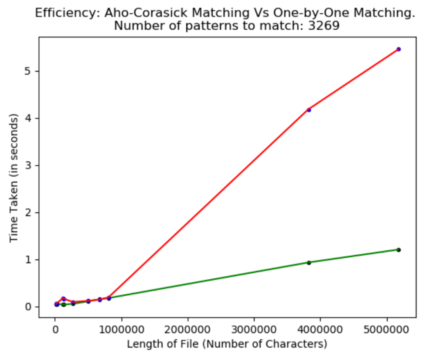
(b) number of patterns: 179



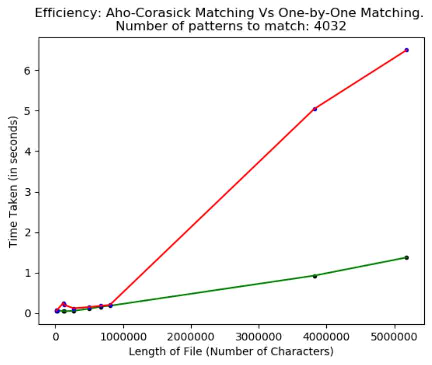
(c) number of patterns: 504



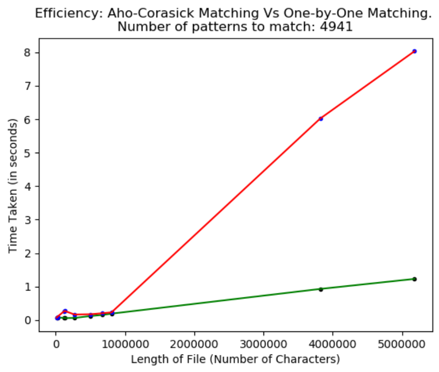
(d) number of patterns: 2240



(e) number of patterns: 3269



(f) number of patterns: 4032



(g) number of patterns: 4941

4.2 Observation

As we can see from figures a, b and c above, the naive approach (red curve) is faster than Aho-Corasick (green curve) for small numbers of patterns ($n \leq 54$ in our case). figures d through g show that as the number of patterns increases the Aho-Corasick algorithm becomes way faster than the naive approach.

5 Conclusion

After experimenting with the naive string matching approach and the Aho-Corasick implementation, we can conclude that when the number of patterns to be matched is relatively small, it is better to use the naive string matching implementation and when the number of patterns to be matched is relatively big, then it is better to use the Aho-Corasick implementation.