Experimentation with String Search Algorithms

Team - 6
Ezgi Yılmaz - 150150036
Muhammed Burak Buğrul - 150140015

Table of Contents

Introduction

Strings are used in almost every field in computer science and computer engineering. There are some problems that occurred a thousand times in these areas. Some of these problems are palindrome finding, prefix matching, string search(exact match and similarity) etc. In this experiment, our problem of interest is finding the occurrence of a string in another one. In computer science history, several algorithms are implemented in order to solve the problem. They all have advantages and disadvantages against each other. We aim to compare some of

these algorithms and state the advantages and disadvantages of them by using analysis experiment results. Our algorithms of focus are:

- Naive Search Method
- Z Algorithm
- Rabin Karp Algorithm
- Knuth Morris Pratt Algorithm

Background

Problem Statement

Given two strings named ‘text’ and ‘pattern’, the aim of the program is to find occurrences of ‘pattern’ in ‘text’.

Literature Review

In the beginning, it seems like a problem that has a simple trivial solution. But in fact, there are several solutions. While some of them have time linear time complexities in length of ‘text’ and ‘pattern’, some of them have quadratic time complexities.

In the bigger picture, the occurrence situation of this problem may change. ‘pattern’ may change continuously, ‘pattern’ may occur thousands of times in the ‘text’, ‘text’ may change continuously etc. All of the algorithms have different behaviors against different scenarios.

Naive Approach is probably the first thing that came into mind. It is trying to find ‘pattern’ in ‘text’ from starting all indices of ‘text’ separately.

Knuth Morris Pratt Algorithm uses a finite automata-based approach to construct a prefix function[1].

Implementation Rabin - Karp Algorithm requires some knowledge in hashes, especially in string hashes[2].

Z Algorithm runs on a single string for finding prefix matches for every index. But it can be modified in order to find an exact pattern match.

Proposal Summary

We have stated there may be different scenarios for exact string matching problem. We would like to do some measurements on different scenarios by using algorithms above in order to help

to make decisions about which one to use in which scenario. The scenarios(inputs) that we will consider in this experiment are:

	Text Size	Pattern Size	Number of Character of Alphabet Used
Input 1.1	1 000	33	4
Input 1.2	1 000	33	26
Input 2.1	1 000	300	4
Input 2.2	1 000	300	26
Input 3.1	10 000	1 000	4
Input 3.2	10 000	1 000	26
Input 4.1	10 000	3 000	4
Input 4.2	10 000	3 000	26
Input 5.1	100 000	333	4
Input 5.2	100 000	333	26
Input 6.1	100 000	30 000	4
Input 6.2	100 000	30 000	26

All of the algorithms are suitable to run with the inputs above. So in input viewpoint, all algorithms are in the same scope. For each run, we will measure the runtime of each algorithm in order to use the information for comparisons. In addition, we will analyze space and time complexities of algorithms and comment about run time - complexity correlations.

Outcome

We will analyze the run time performance and scenorial behaviors of algorithms in question. The scenarios are in a general scope. Thus, programmers can choose a well-suited algorithm for them to implement according to their possible scenarios.

Value

This project provides a large viewpoint to the popular string algorithms, a lot of app developers and service providers use different algorithms to enable a fast and correct search option in the app or website.

It is very useful research for algorithm selection according to the characteristics of the dataset. Also, we created 4 lettered data set which is more like DNA data and in the microbiology area, these results can be considered.

Methods

We will use Python3.7 programming language for creating the input sets. The most important reason why we choose Python3.7 for this task is its' simple usage for this kind of tasks that not require performance optimizations.

We will use the C++ programming language for implementing all of the algorithms stated above. Our compiler will be C++14.

We will split the implementation parts but we will do pair programming in order to follow the same convention and not making a difference that can affect the performance(eg. Function calls without string reference) and reinforce implementing clean and understandable codes. We will not use any third party libraries for this task.

Schedule and Cost

Research

For the research stage, while Ezgi will learn Naive Approach and Rabin Karp algorithms; Burak will learn Z and KMP algorithms.

Code

Ezgi and Burak will implement the algorithms they learned. The coding methodology will be pair programming. Also, Burak will write a Python3 script in order to generate input scenarios. Burak and Ezgi will test each other's algorithms with small handwritten inputs.

Gantt Chart

	April				May			
	W1	W2	W3	W4	W1	W2	W3	W4
Learning the algorithms	X	X						
Implementing the algorithms		X	X	X				
Generating the test datasets				X	X			
Testing and analyzing the algorithms						X	X	
Visualizing the results							X	X

RACI Matrix

Task /Person	Learning Naive Approach and Rabin Karp Algorithm	Learning KMP and Z Algorithm	Implementing Naive Approach and Rabin Karp Algorithm	Implementing KMP and Z Algorithm	Testing Naive Approach and Rabin Karp Algorithm	Testing KMP and Z Algorithm	Writing the Report
Ezgi	RA	I	RA	CI	I	RA	RACI
Burak	I	RA	CI	RA	RA	I	RACI

Conclusion

In this experiment, we will analyze the performance of some known string pattern search(exact match) algorithms and visualize the results. All algorithms will be tested against datasets that have different characteristics. By using the results of tests, we will analyze state the advantages and disadvantages of the algorithms. Algorithms will be implemented in C++14.

References

[1] Knuth, D. E., Morris, Jr., J. H., & Pratt, V. R. (1977). Fast Pattern Matching in Strings. SIAM Journal on Computing, 6(2), 323–350.

[2] Karp, R. M., & Rabin, M. O. (1987). Efficient randomized pattern-matching algorithms. IBM Journal of Research and Development, 31(2), 249–260.