COSC 320 – 001
Analysis of Algorithms
2020 Winter Term 2

Project Topic Number: #4

String Matching Problem

Group Lead:

Matthew Borle - 42615435

Group Members:

Barry Feng - 12981156

Guy Kaminsky - 65234775

Ross Morrison - 36963411

Abstract

In this first milestone, we got the four of us to meet together and introduce ourselves to each other. We were able to decide to work on project topic four. Next, we were able to complete the headers for the project proposal, and most importantly start to assign tasks to one another. Lastly, we had to familiarize ourselves with what algorithms we need to implement.

Problem Description

String matching is a common problem where strings are compared to detect commonalities. Typically used for plagiarism detection. The problem is one master document of strings must be compared to a set of other documents of strings. Comparing the master to one other document would require looping over each word of each document resulting in $O\left(n^2\right)$ time. If there are n documents to compare, then the time complexity for the entire operation would potentially be $O(n^n)$ time. Often the number of strings we need to compare to is enormous and comparing each one to each other would take exponential time. Often matching strings are not an indicator of plagiarism. For example, two similar programs could have identical variable names, but different implementations. These two would have many matches but are not the same. Subsequent matches can be a helpful indicator for plagiarism, as it is unlikely that entire sentences would match. The more subsequent matches, the more likely the document was copied. Strategies like this can help strengthen the legitimacy of our program's plagiarism claim.

Edge Cases

- Number of strings = 0
  If the number of strings in either document is 0, then no comparisons are needed, and the program completes in $O(1)$.

- Number of strings = 1
  If the number of strings in either document is 1, then only n comparisons are needed, and the program completes in $O(n)$.

- Number of documents to compare to master = 0
  If the number of documents to compare to the master is 0, then nothing needs to be done. The program completes in $O(1)$.

- Number of documents to compare to master = 1

If the number of documents to compare to the master is 1, then only n strings need to be compared to the n strings of the master document. The program completes in $O(n^2)$.

Expected Complexities

The potential max time complexity of $O\left(n^{2^n}\right)$ will likely be a challenge for us to overcome. Shooting for a maximum time complexity of $O\left(n^n\right)$ should be our goal, though we are unsure if it is achievable at this time and with the given knowledge we have.

Dataset Collection

The current plan for collecting a set of data is to start with finding papers related to a certain subject. Once we have accumulated the papers we will create a document based off of those findings. Once we have a working algorithm, we will increase the amount of papers to test against for plagiarism.

Programming Language

We chose to use python for our implementation as it is much quicker to create a functional prototype. Python has multiple string comparison libraries available to make our lives easier, though we may choose to create our own implementation if these restrict our ability. Since python is similar to pseudocode it will be quicker and easier to implement our algorithm.

Task Separation and Responsibilities

We have five tasks to split up among four people to reach the first milestone. We are going to use pairs for the first four tasks. Firstly, Guy Kaminsky and Barry Feng will collect the data. Next, Matthew Borle and Barry Feng will formulate the problem. Further, Guy Kaminsky and Ross Morrison will create the pseudo code. Then, Ross Morrison and Matthew Borle will analyze the algorithm. Finally everyone will meet to talk about their unexpect cases/difficulties.

Unexpected Cases/Difficulties

At this current stage we all encountered difficulties in coming up with ideas on how to design and implement the algorithms. To alleviate this issue, we have set a milestone for all of us to look over KMP, LCSS, Rabin-Karp fingerprint algorithms before the next meeting. This will give us all a better understanding on how to approach this project. As for future difficulties, as mentioned previously,

overcoming the potential max time complexity of $O\left(n^{2^n}\right)$ will be challenging, but likely achievable with the help of the algorithms mentioned.