**COSC 320 – 001**

***Analysis of Algorithms***

2022/2023 Winter Term 2

**Project Topic Number: 2**

**Title of project: Plagiarism Detector**

**Group Members:**

**Divyajot Kaur Dadiala**

**Jusnoor Kaur Sachdeva**

**Robert Barnstead**

**Abstract:**

In this project, we are building a plagiarism detector and comparing three different algorithms KMP, LCS, and RabinCarp fingerprints. All three algorithms can be used to detect plagiarism in a text.

**Project Description**

Building a Plagiarism detector and comparing three different algorithms.

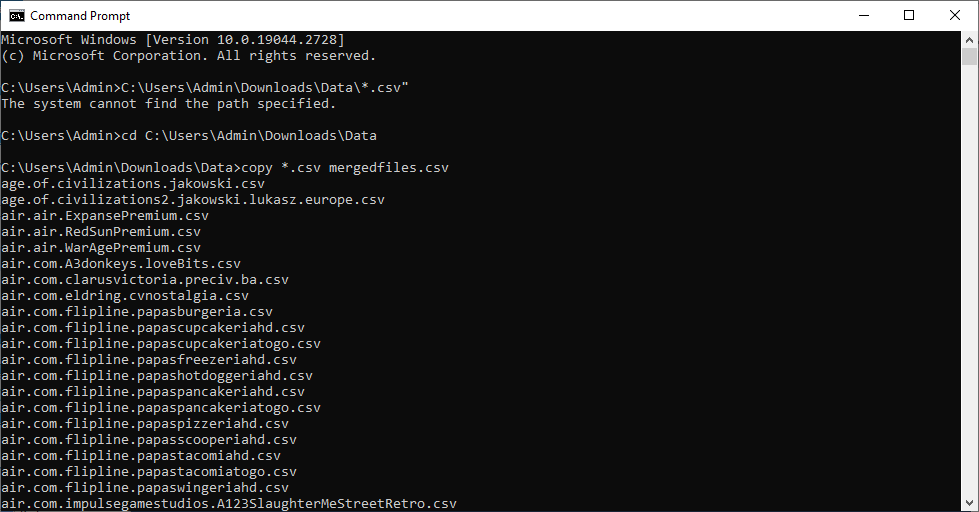
The project is completed in four different milestones – The team wrote the algorithm problem, wrote a pseudo code, provided an analysis, implemented the algorithms, and listed the unexpected cases and difficulties.

**Dataset**

We used the dataset provided by the professor. We merged the csv files into a single csv file using command prompt.

Script to combine csv files

copy “C:\Users\Admin\Downloads\Data\\*.csv” mergedfiles.csv



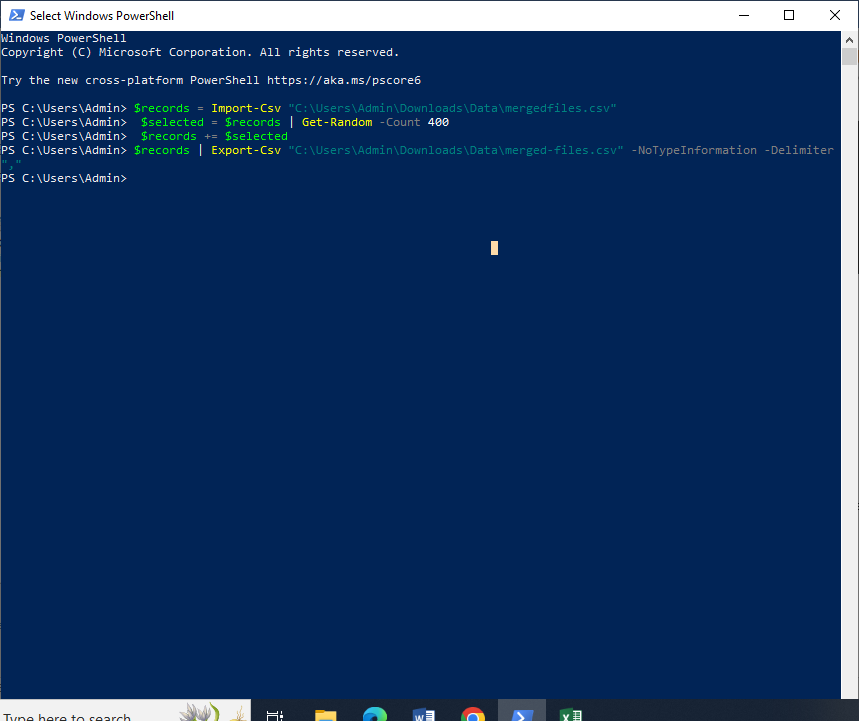
Creating Plagiarized DataSet

$records = Import-Csv "C:\Users\Admin\Downloads\Data\mergedfiles.csv"

$selected = $records | Get-Random -Count 400

$records += $selected

$records | Export-Csv "C:\Users\Admin\Downloads\Data\mergedfilesplagiariseddata.csv" -NoTypeInformation -Delimiter ","



**Implementation**

**GitHub Link**: https://github.com/rbarnstead/cosc320-milestone3

**The KMP Algorithm Implementation**

KMP Algorithm has been designed to take two strings a text string and a pattern string and determine whether the pattern appears in the text. To implement this, we took the dataset provided and read the CSV file using openCSV in Java to create one giant string of text. To define our pattern we randomly took one of the entries from the text that was found while reading the CSV file and stored it as a string. Then we called the search function using both the text and pattern. In the search function we call the LPS function in order to create the LPS array that assists us in finding the pattern in the string. This returns us the Char range where the pattern is found, or in our case where plagiarism has been found. A major issue that we ran into was the time it takes to create the string.

**LCSS Implementation**

The LCS Algorithm takes two strings A which is the text and B which is the pattern and checks for the longest common sub sequence. This is done using a 2d array of size [len(B)+1][len(A)+1] called check that gets filled retroactively.

This is done by initializing each value to 0 then checking whether the value at A[i-1] and B[j-1] match, if they do we set check[i][j] equal to the value at check[i-1][j-1]+1. This increases the longest common subsequence by 1. If they are not a match we set the longest common subsequence equal to the max between A[i-1][j] or B[i][j-1]. Finally, we return the value at check[a][b] or the last value in the array which is our longest common subsequence value.

**Rabin-Karp Fingerprints:**

Rabin-Karp fingerprinting is a string-matching algorithm which looks

for a pattern within a text(using the process of hashing). In this

algorithm, a fingerprint is created which works as a hash value for the

given pattern, and then it slides a window of the same size over the

text, eventually creating fingerprints for each window. Now, if the

fingerprint of the current window matches the fingerprint of the

pattern then the algorithm is used to check if the window and the

pattern match character by character. This algorithm is applied to a

number of areas such as plagiarism detection, data deduplication and

intrusion detection. It is also used for searching files for a specific

pattern.

It has an average case complexity of O(n+m) , where n is the length of

the text and m is the length of the pattern and this is one of the

advantages of the algorithm since it makes the algorithm efficient

**Results and Analysis**

**KMP Algorithm**

For analyzing the performance of the KMP algorithm, we plotted Input Size and Time

**Input Size and Time:**

Time taken to read the input file increases exponentially as the number of lines in the input file increases. The time to read a single line is constant but as the number of line increases, the time taken to read the file increases exponentially.

KMP algorithm has a worst-case time complexity of O(n)(where n is the length of the text). The length of the text increases as the number of lines read from CSV file increases and the time taken to perform string matching increases linearly.

The time complexity of the KMP algorithm is O(n+m), where n is the length of the text and m is the length of the pattern. As ‘n’(i.e., the length of the text) grows, the running time of the algorithm will keep increasing. KMP algorithm has a linear time complexity, so the graph of the running time will also be a linear graph. Therefore, the running time graph is straight line passing through the origin.

KMP algorithm uses an array to store the longest proper prefix that is also a suffix of each substring of the pattern. Depending on the implementation of this array, the constant factors may differ. The KMP algorithm also uses a function to compute the LPS array, which is used for the string matching, implementation of this function can affect the constant factors in the running time of the algorithm.

**LCS Algorithm Result and Analysis**

**Input Size and Run Time**

LCS is an algorithm that finds longest common subsequence. Elements of the sequences may not accupy the consecutive positions. String based comparisons are accurate for plagiarism detection.

In our code for LCS, first 100 rows are read and concatenates as single string.

Next, we defined a function LCS. This function takes four arguments, Strings A and B and their lengths a, b. Then, it calculates and returns length of the longest common subsequence between the two strings A and B.

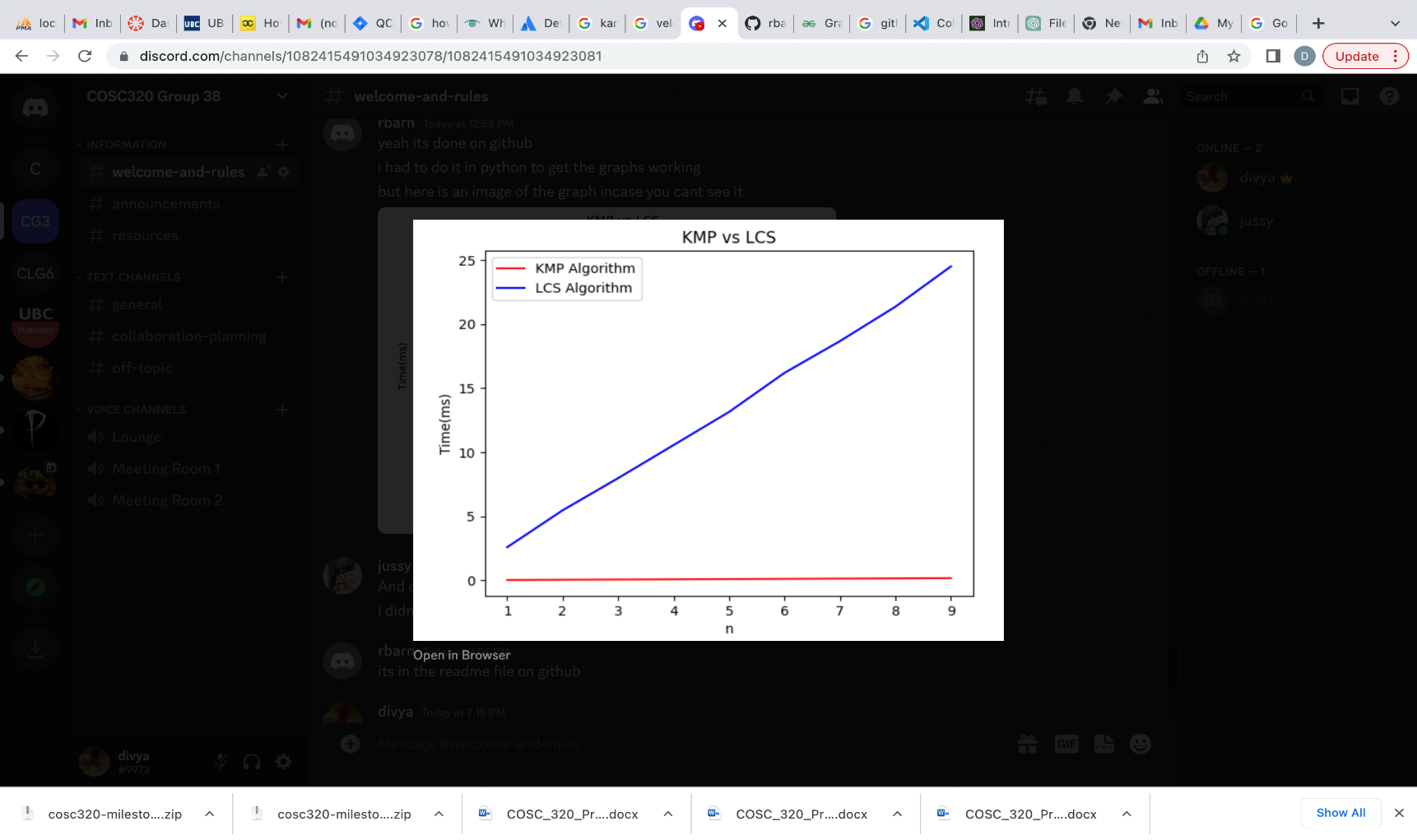
Therefore, the **Time complexity** of LCS algorithm is **O(ab)** where a and b is the length of the strings A and B.

Since the algorithm processes each character only once, the actual run time depends on lengths of strings A and B.

**Worst Case Scenario: If there no match in the strings, O (2^n).**

**Input Size**: The input size of the LCS algorithm increases with run time. The time complexity is O(ab). If length of either of the strings is increased the runtime increases linearly.

**Graphs/ Plot**



**The graph above illustrates the run time analysis for the LCS and KMP algorithms for increasing input overtime.**

The input length is shown on the x-axis and execution time is shown on the y axis. We have used matplot lib for plotting our graph in Python, numpy for computations and timeit for measuring the code’s execution time. We set the start time and defined a range of random integers from 1 to 10. We used the list dt to store the execution times for each input length. We used the for loop to go through the integers. Then we called the search function inside the loop for implementing the KMP algorithm on input strings A and B. The result of the execution time is added to dt.

We repeated the same procedure for LCS algorithm storing the execution time in dt2 list.

**Unexpected Cases/Difficulties**

**Large File Size** – We faced difficulty in uploading and downloading the Dataset files. It could be due to Internet connectivity.

**Creating plagiarized Dataset and Implementation:** We wrote a script. We should have taken TA’s help to have more clarity on implementation.

**Team Coordination** – We think that working together in a Lab would have helped and we could have done the task faster.

**Time management**: it was tough to schedule a group meeting keeping in mind the timings given by all the team members so there are tasks assigned to all the members.

**Solution**: Scheduled a zoom meeting so that there is a discussion keeping in mind the comfort level of all the team members

**Responsibilities**

**Algorithm Analysis and Run time** – Divyajot/ Jusnoor

**Coding/ Github** - RBarn

**Video Creation** – Divyajot and Jusnoor

---------------------------------------------------------------------------------------