**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:**

For this milestone our team will be describing the second that we will use for our plagiarism detector. We shall be working on the LCSS algorithm. We shall be describing the algorithm problem, writing a pseudo code, providing an analysis and listing the unexpected cases and difficulties.

**Problem Formulation**

**Problem**: For the given problem we have to implement string matching and build a plagiarism detector.

We are given a set of files that we will analyze using the plagiarism detector.

For the 2nd milestone we are working on the **LCSS algorithm**

**Problem Statement for LCSS**: To check plagiarism, two strings are matched using LCSS where, length of the longest subsequence present in both the strings is found.

**What is the LCSS algorithm?**

LCSS 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 this milestone, we analyse the LCSS algorithm that aims to search only those strings where longest common subsequence matches. This leads to checking plagiarism.

**Pseudo-code**

**LCSS algorithm Pseudo Code**

//function to calculate length of Longest Common Sequence

//takes 2 arrays A and B with their lengths a and b

function LCSS(A[], B[], a, b, check[][])

//Once the length of either string is equal to 0 return 0

if a == 0 or b == 0

return 0

//This solves needless recursion such that if a value has already been calculated then a variable other -1 will be at check[a][b] and it will skip the recursion

if check[a][b] != -1

return check[a][b]

//If there is a match between A and B call recursion on A and B at length-1

if A[a-1] == B[b-1]

check[a][b] = 1 + LCSS(A, B, a-1, b-1, check)

return check[a][b]

//If there is no match get the max value between the recursive call of A and B at either a-1 and b or a and b-1

else

check[a][b] = max(LCSS(A, B, a-1, b, check), LCSS(A, B, a, b-1, check))

return check[a][b]

main

string1 = Somestring

string2 = Somestring

string1array[] = CharArray of string1

string2array[] = CharArray of string2

s1 = string1 legnth

s2 = string2 length

checker[s1+1][s2+1]

//create 2d array and store a value < 0 in each location

for i=0 i < checker[].length i++

for j=0 j < checker[][].length j++

checker[i][j] = -1

LCSS(string1array, string2array, s1, s2, checker)

**Algorithm Analysis.**

**In the function:**

Input sequences are string1array[0..s1-1] and string2array[0..s2-1] of lengths s1 and s2 respectively.

LCSS(string1array, string2array, s1, s2, checker) provides the length of the longest common subsequence in s1 and s2.

**Time Complexity:**

**Run time**: **O(s1\*s2)** where s1 and s2 are lengths of strings.

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

**Proof of Correctness**

For proving the proof of correctness, we can use mathematical induction. Let LCSS(A,B,a,b,check) denote the length of the longest common subsequence of A and B of length a and b respectively, and let L denote the length of the longest common subsequence of A and B.

1. Base case: When either the length of string A or B is 0, LCSS(A,B,a,b,check) returns 0. This is correct because an empty string has no common subsequence with any other string, and the length of the longest common subsequence would also be 0.
2. Induction Hypothesis: Assume that LCSS(A,B,i,j,check) is correct for all 0 ≤ i ≤ a and 0 ≤ j ≤ b.
3. Induction Step: We need to show that LCSS(A,B,a,b,check) is also correct. The algorithm uses memoization to avoid redundant computations. If the value of LCSS(A,B,a,b,check) has already been calculated, then it returns the value from the memorization table, which is guaranteed to be correct because of our assumption.

If A[a-1] = B[b-1], then the character A[a-1] is present in the LCS. Therefore, we can include this character in the LCS, and the length of the LCS would be 1 + LCSS(A,B,a-1,b-1,check). This is the same as the length of the LCS of the two strings with the last character removed from both.

If A[a-1] ≠ B[b-1], then we have two possibilities. Either the LCS includes the last character of A or it includes the last character of B. In the former case, we need to find the LCS of A[0...a-2] and B[0...b-1], and in the latter case, we need to find the LCS of A[0...a-1] and B[0...b-2]. Therefore, the length of the LCS would be the maximum of these two values, which is max(LCSS(A,B,a-1,b,check),LCSS(A,B,a,b-1,check)).

As the base case and the induction step are correct, the algorithm is correct. Therefore, the algorithm finds the length of the longest common subsequence of A and B correctly.

**Unexpected Cases/Difficulties.**

**Case 1:** Team communication was a problem, therefore, we created a discord group so that we can discuss the tasks.

**Solution:**For further milestones, we have decided on a zoom meeting or meeting in the lab.

**Case 2:** Task Division: We faced some challenge while distributing tasks quite close to deadline.

**Solution:**Task distribution, we will do much in advance for our next milestone.

**Task Separation and Responsibilities**

Divyajot Kaur Dadiala: Analysis, Problem formation, run time, unexpected cases

Jusnoor Kaur Sachdeva: Proof of Correctness

Robert Barnstead: Pseudo Code