# Ramdeobaba University, Nagpur

# 

# 

# PRACTICAL NO. 5

## **Aim: Implement a dynamic algorithm for Longest Common Subsequence (LCS) to find the**

## **length and LCS for DNA sequences.**

## Name: Sujal Singh

## Sec: A4

## Batch: B4

## Roll No:52

## 

**Problem Statement:**

**(i) DNA sequences can be viewed as strings of A, C, G, and T characters, which**

**represent nucleotides. Finding the similarities between two DNA sequences is an**

**important computation performed in bioinformatics.**

**[Note that a subsequence might not include consecutive elements of the original sequence.]**

**TASK 1: Find the similarity between the given X and Y sequences.**

**X=AGCCCTAAGGGCTACCTAGCTT**

**Y= GACAGCCTACAAGCGTTAGCTTG**

**Output: Cost matrix with all costs and direction, final cost of LCS, and the LCS.**

**Length of LCS=16**

**Code:**

**def lcs(X, Y):**

**m, n = len(X), len(Y)**

**dp = [[0] \* (n + 1) for \_ in range(m + 1)]**

**direction = [[""] \* (n + 1) for \_ in range(m + 1)]**

**for i in range(1, m + 1):**

**for j in range(1, n + 1):**

**if X[i - 1] == Y[j - 1]:**

**dp[i][j] = dp[i - 1][j - 1] + 1**

**direction[i][j] = "↖"**

**elif dp[i - 1][j] >= dp[i][j - 1]:**

**dp[i][j] = dp[i - 1][j]**

**direction[i][j] = "↑"**

**else:**

**dp[i][j] = dp[i][j - 1]**

**direction[i][j] = "←"**

**i, j = m, n**

**lcs\_str = ""**

**while i > 0 and j > 0:**

**if direction[i][j] == "↖":**

**lcs\_str = X[i - 1] + lcs\_str**

**i -= 1**

**j -= 1**

**elif direction[i][j] == "↑":**

**i -= 1**

**else:**

**j -= 1**

**print("===== COST MATRIX (DP Table) =====")**

**for row in dp:**

**print(row)**

**print("\n===== DIRECTION MATRIX =====")**

**for row in direction:**

**print(row)**

**print("\nLength of LCS:", dp[m][n])**

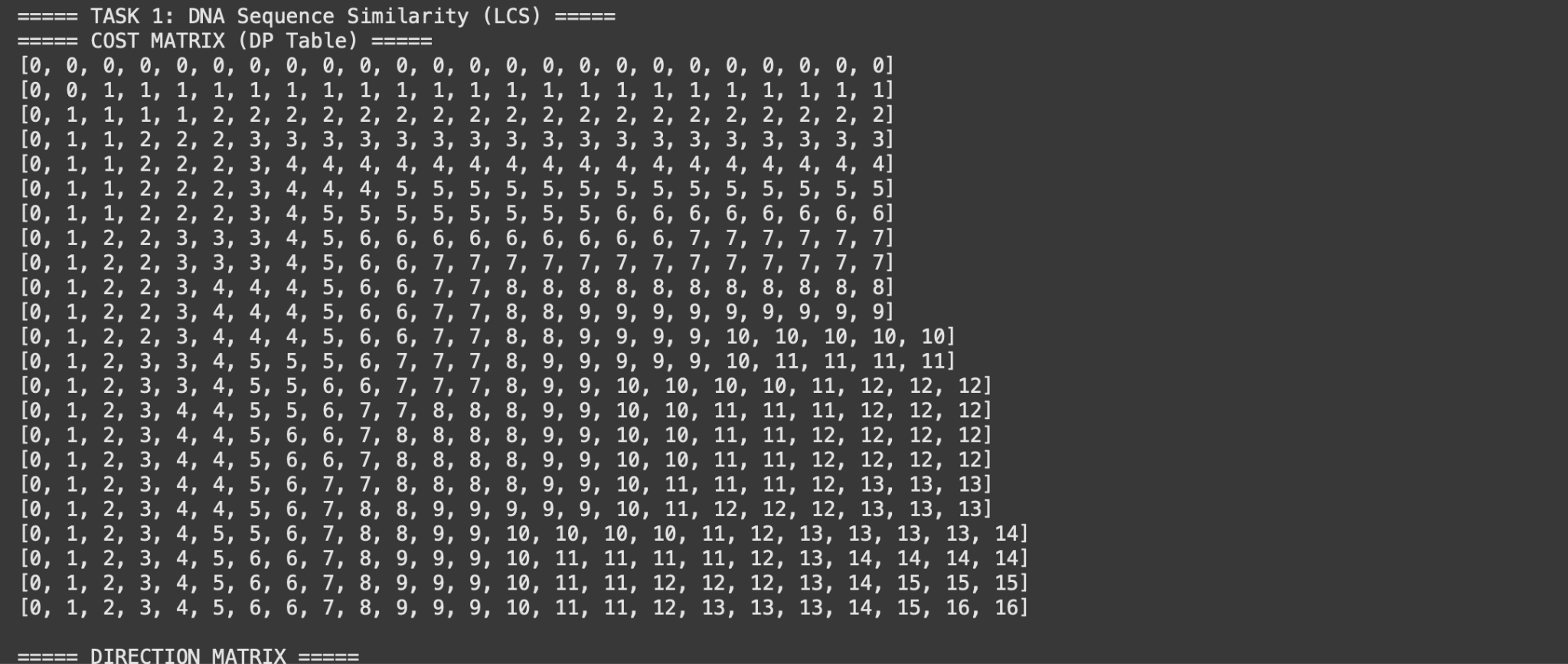
**print("LCS:", lcs\_str)**

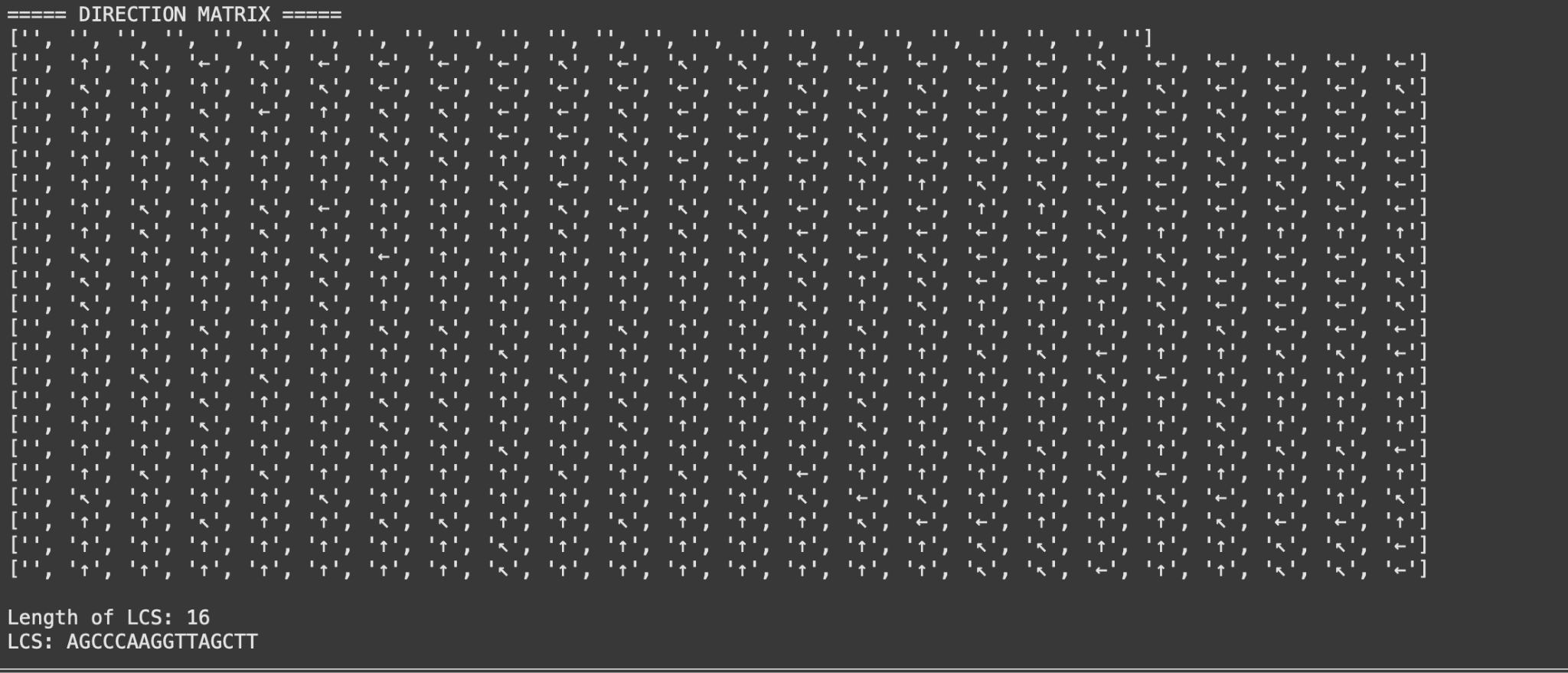
**X = "AGCCCTAAGGGCTACCTAGCTT"**

**Y = "GACAGCCTACAAGCGTTAGCTTG"**

**print("===== TASK 1: DNA Sequence Similarity (LCS) =====")**

**lcs(X, Y)**

Output:

****

**TASK-2: Find the longest repeating subsequence (LRS). Consider it as a variation of the**

**longest common subsequence (LCS) problem.**

**Let the given string be S. You need to find the LRS within S. To use the LCS framework, you**

**effectively compare S with itself. So, consider string1 = S and string2 = S.**

**Example:**

**AABCBDC**

**LRS= ABC or ABD**

**Code:def longest\_repeating\_subsequence(S):**

**n = len(S)**

**dp = [[0] \* (n + 1) for \_ in range(n + 1)]**

**for i in range(1, n + 1):**

**for j in range(1, n + 1):**

**if S[i - 1] == S[j - 1] and i != j:**

**dp[i][j] = 1 + dp[i - 1][j - 1]**

**else:**

**dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])**

**i, j = n, n**

**lrs = ""**

**while i > 0 and j > 0:**

**if dp[i][j] == dp[i - 1][j - 1] + 1 and S[i - 1] == S[j - 1] and i != j:**

**lrs = S[i - 1] + lrs**

**i -= 1**

**j -= 1**

**elif dp[i - 1][j] >= dp[i][j - 1]:**

**i -= 1**

**else:**

**j -= 1**

**print("===== COST MATRIX (DP Table) =====")**

**for row in dp:**

**print(row)**

**print("\nLongest Repeating Subsequence (LRS):", lrs)**

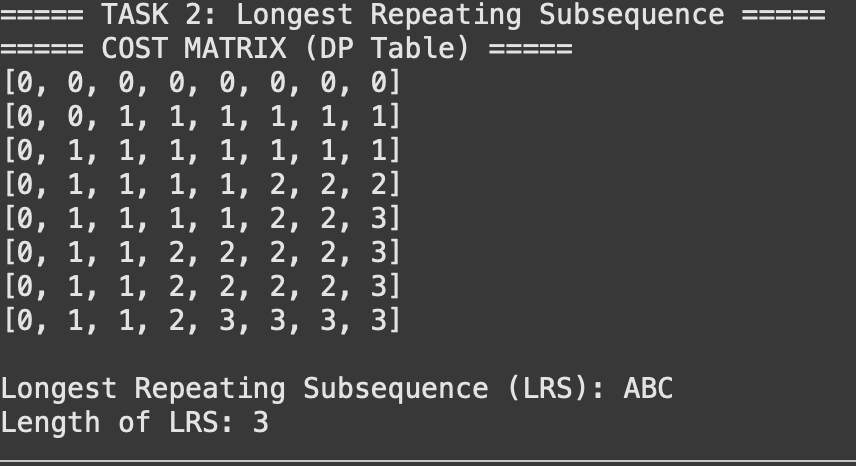
**print("Length of LRS:", dp[n][n])**

**S = "AABCBDC"**

**print("===== TASK 2: Longest Repeating Subsequence =====")**

**longest\_repeating\_subsequence(S)**

**Output:**

****

# Leet Code Submission

## Code:

class Solution:

def longestCommonSubsequence(self, text1: str, text2: str) -> int:

m = len(text1)

n = len(text2)

dp = [[0] \* (n + 1) for \_ in range(m + 1)]

for i in range(1, m + 1):

for j in range(1, n + 1):

if text1[i - 1] == text2[j - 1]:

dp[i][j] = dp[i - 1][j - 1] + 1

else:

dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])

return dp[m][n]

Output/Screenshot:

