NAME:NEER AWASTHI

CLASS:A4 B2 20

**PRACTICAL 5**

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

length and LCS for DNA sequences.

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 are an

important computation performed in bioinformatics.

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

X=AGCCCTAAGGGCTACCTAGCTT

Y= GACAGCCTACAAGCGTTAGCTTG

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

Length of LCS=16

SOURCE CODE:

# ------------------- TASK 1: Longest Common Subsequence -------------------

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)]

# Fill cost matrix

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] = "←"

# Backtrack to find LCS string

i, j = m, n

lcs\_str = []

while i > 0 and j > 0:

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

lcs\_str.append(X[i - 1])

i -= 1

j -= 1

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

i -= 1

else:

j -= 1

lcs\_str.reverse()

print("\n------------------- TASK 1: LCS -------------------")

print("\nCOST MATRIX (with directions):\n")

print(" ", end="")

for ch in " " + Y:

print(f"{ch:>3}", end="")

print()

for i in range(m + 1):

if i == 0:

print(" ", end="")

else:

print(f"{X[i - 1]:>3}", end="")

for j in range(n + 1):

print(f"{dp[i][j]:>2}{direction[i][j]:1}", end=" ")

print()

print(f"\nFinal Cost (Length of LCS) = {dp[m][n]}")

print(f"LCS = {''.join(lcs\_str)}")

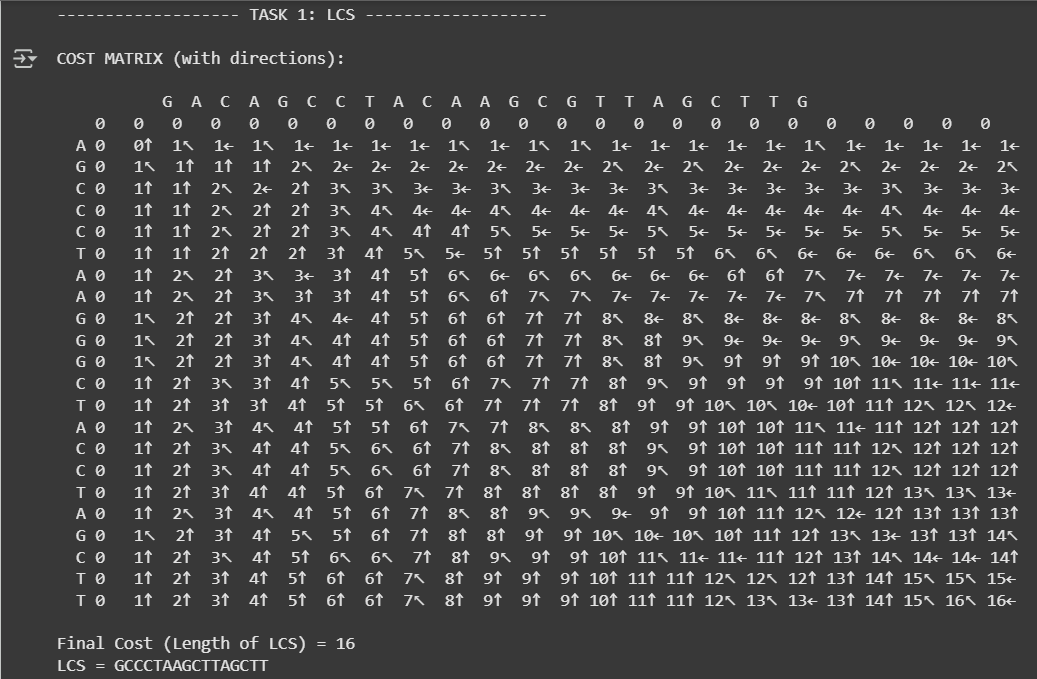
# Given DNA sequences

X = "AGCCCTAAGGGCTACCTAGCTT"

Y = "GACAGCCTACAAGCGTTAGCTTG"

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

SOURCE CODE:

# ------------------- TASK 2: Longest Repeating Subsequence -------------------

def lrs(S):

n = len(S)

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

direction = [[""] \* (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]

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] = "←"

# Backtrack to get one possible LRS

i, j = n, n

lrs\_str = []

while i > 0 and j > 0:

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

lrs\_str.append(S[i - 1])

i -= 1

j -= 1

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

i -= 1

else:

j -= 1

lrs\_str.reverse()

print("\n------------------- TASK 2: LRS -------------------")

print("\nCOST MATRIX (with directions):\n")

print(" ", end="")

for ch in " " + S:

print(f"{ch:>3}", end="")

print()

for i in range(n + 1):

if i == 0:

print(" ", end="")

else:

print(f"{S[i - 1]:>3}", end="")

for j in range(n + 1):

print(f"{dp[i][j]:>2}{direction[i][j]:1}", end=" ")

print()

print(f"\nFinal Cost (Length of LRS) = {dp[n][n]}")

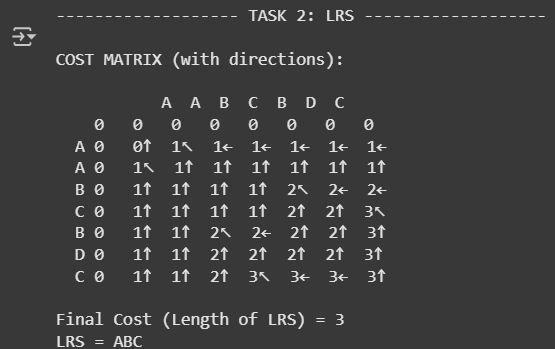
print(f"LRS = {''.join(lrs\_str)}")

# Example input from assignment

S = "AABCBDC"

lrs(S)

OUTPUT:



LeetCode Assesment:

