**Name-Abhishek Jagam**

**Class - A5-B1-13**

**Topic - LCS & LRS**

**Aim: Implement a dynamic algorithm for Longest Common Subsequence (LCS) to find the length and LCS for DNA sequences.**

**Code For LCS :**

def lcs\_full(X: str, Y: str) -> tuple[int, str]:

m = len(X)

n = len(Y)

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

# Fill the C table

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

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

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

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

else:

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

# Backtrack

lcs\_string = ""

i, j = m, n

while i > 0 and j > 0:

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

lcs\_string = X[i - 1] + lcs\_string

i -= 1

j -= 1

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

i -= 1

else:

j -= 1

lcs\_length = C[m][n]

return lcs\_length, lcs\_string

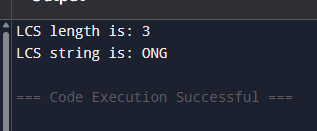
string1 = "STRONG"

string2 = "LONGEST"

length, lcs = lcs\_full(string1, string2)

print(f"LCS length is: {length}")

print(f"LCS string is: {lcs}")



**Code For LRS :**

def lrs\_full(S: str) -> tuple[int, str]:

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):

# Key LRS condition: match AND different indices (i != j)

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])

# Backtrack to reconstruct the LRS string

lrs\_string = ""

i, j = n, n

while i > 0 and j > 0:

# Match/Diagonal move condition (must satisfy LRS constraint)

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

lrs\_string = S[i - 1] + lrs\_string

i -= 1

j -= 1

# Move up

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

i -= 1

# Move left

else:

j -= 1

lrs\_length = dp[n][n]

return lrs\_length, lrs\_string

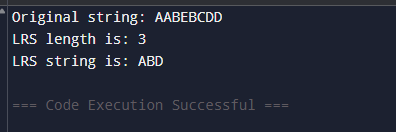
test\_string = "AABEBCDD"

length, lrs = lrs\_full(test\_string)

print(f"Original string: {test\_string}")

print(f"LRS length is: {length}")

print(f"LRS string is: {lrs}")



**Leetcode:**

**Q-1143**

