**PRACTICAL 5**

**Aim :**

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

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**TASK 1 :**

**CODE :**

import time

start = time.time() # Starting Time

A = list("AGCCCTAAGGGCTACCTAGCTT")

B = list("GACAGCCTACAAGCGTTAGCTTG")

m = len(A)

n = len(B)

# Initialization :

# Setting two Matrices for values and direction

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

direc = [['H'] \* (n+1) for \_ in range(m+1)] # 'H' as default direction

# Conditions to Check

for i in range(m+1):

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

if A[i-1] == B[j-1]: # Match Condition

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

direc[i][j] = "D" # Diagonal

elif val[i-1][j] >= val[i][j-1]: # Unmatch Condition

val[i][j] = val[i-1][j]

direc[i][j] = "U" # Up

else:

val[i][j] = val[i][j-1]

direc[i][j] = "S" # Side

# Backtracking to get LCS

i, j = m, n

lcs = []

while i > 0 and j > 0:

if direc[i][j] == "D":

lcs.append(A[i-1])

i -= 1

j -= 1

elif direc[i][j] == "U":

i -= 1

else:

j -= 1

lcs = "".join(reversed(lcs))

# Results

print("\nCost + Direction Matrix :\n\n")

for i in range(m+1):

row = []

for j in range(n+1):

cell = f"{val[i][j]}/{direc[i][j]}"

row.append(cell.rjust(5)) # pad each cell to width 5 for proper alignation

print("".join(row))

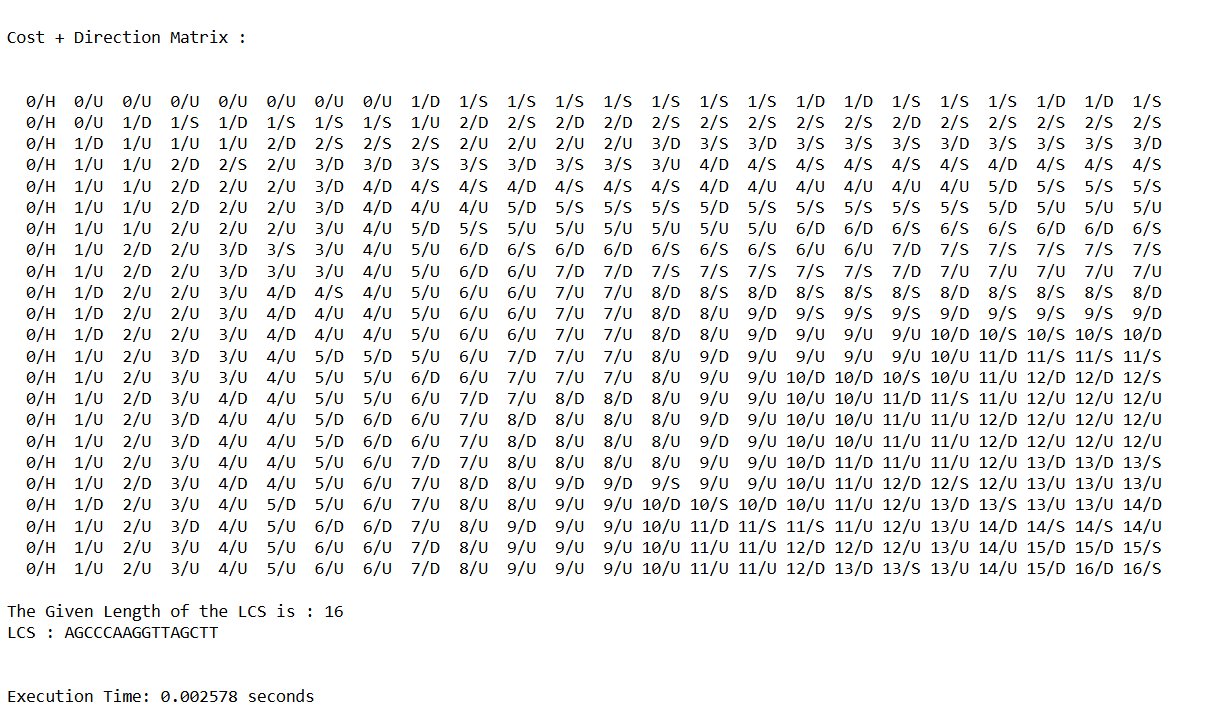
end = time.time() # Ending Time

print("\nThe Given Length of the LCS is :", val[m][n])

print("LCS :", lcs)

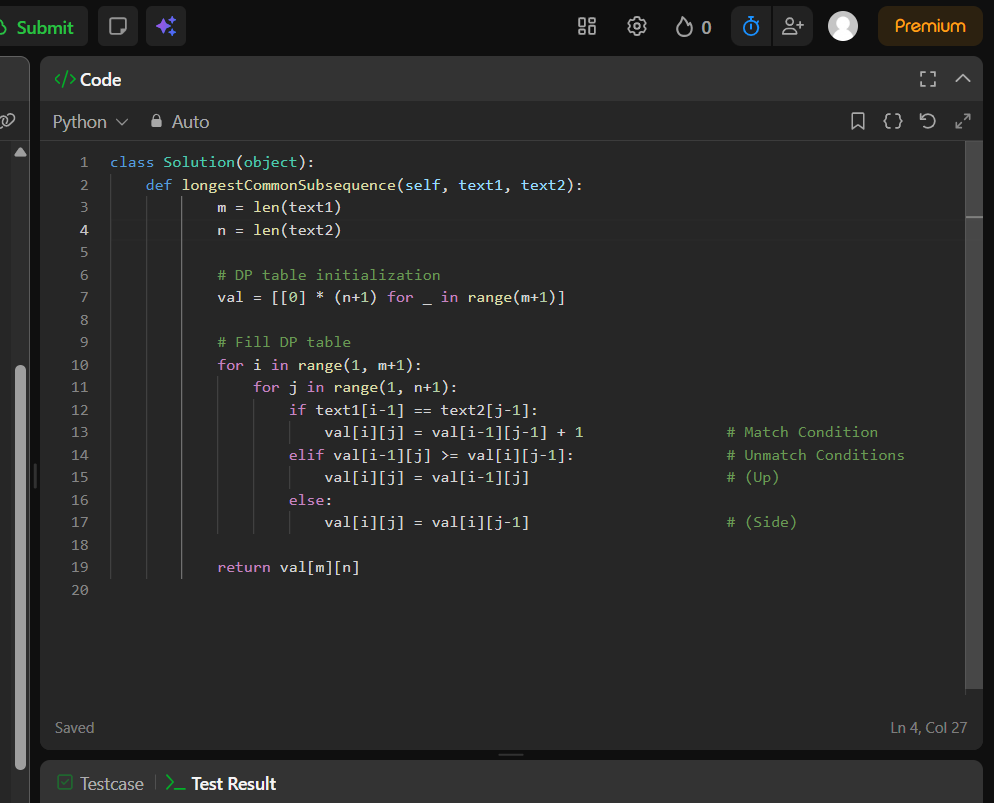
print("\n\nExecution Time: {:.6f} seconds".format(end - start))

**OUTPUT : ( UPDATED )**

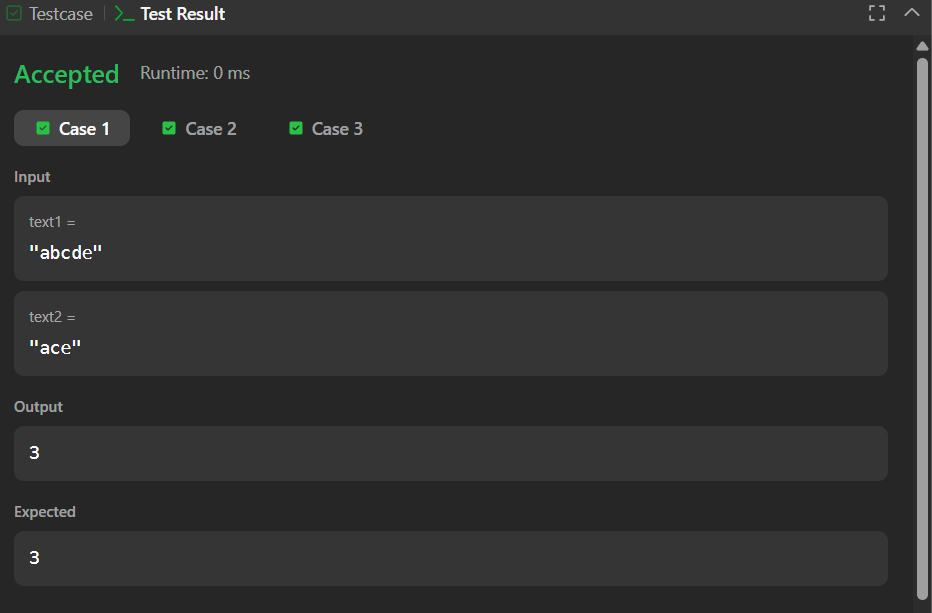
****

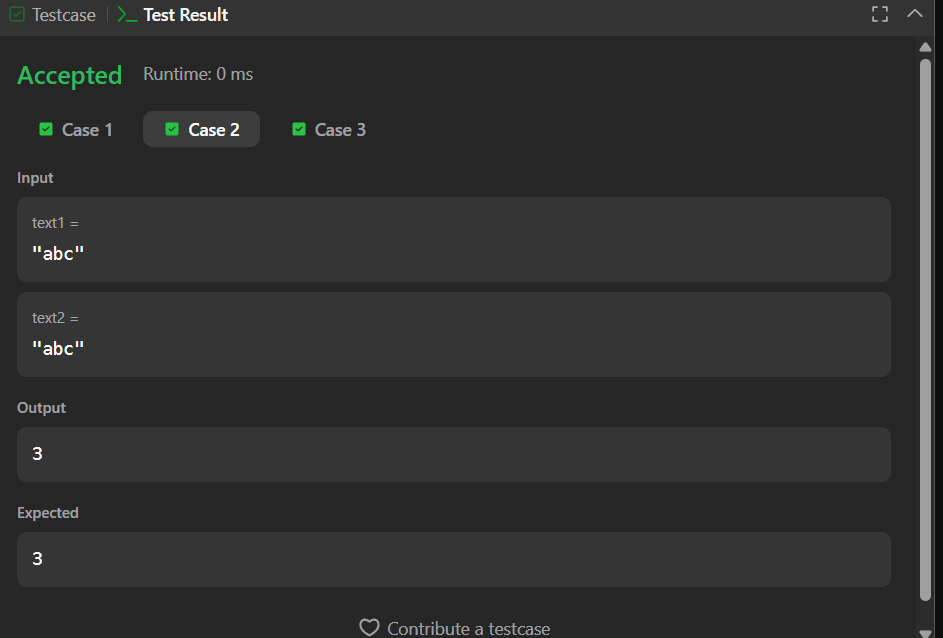
**LEETCODE :**

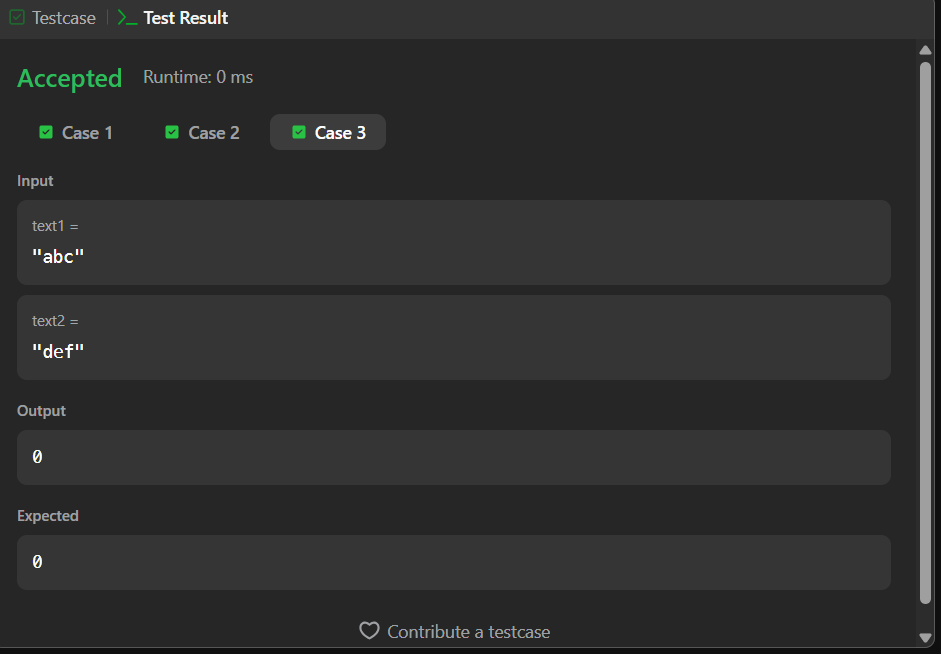
**CODE :**

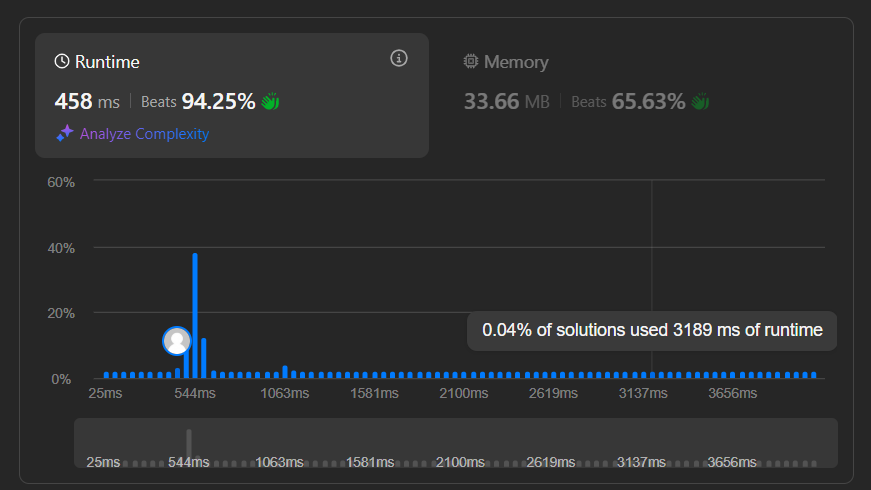
****

**RESULTS :**

****

****

****

****

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**TASK 2 :**

**CODE :**

import time

def LRS(s):

n = len(s)

# Initialisation Cost Matrix :

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

# Conditions :

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

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

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

c[i][j] = 1 + c[i-1][j-1] # Match at different index → diagonal + 1

else:

c[i][j] = max(c[i-1][j], c[i][j-1]) # Mismatch or same index → take max of top and left.

# Backtracking for LRS :

i, j = n, n

lrs = []

while i > 0 and j > 0:

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

lrs.append(s[i-1])

i -= 1

j -= 1

elif c[i-1][j] >= c[i][j-1]:

i -= 1

else:

j -= 1

lrs.reverse() # Backtracking builds reversed string therfore Reversing the String.

return c[n][n], "".join(lrs)

start = time.time() # Starting Time

s = "AABEBCDD"

length, Lrs = LRS(s)

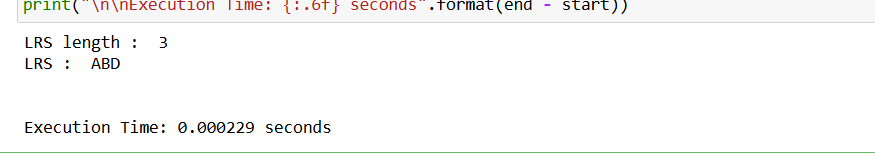
end = time.time() # Ending Time

print("LRS length : ", length)

print("LRS : ", Lrs)

print("\n\nExecution Time: {:.6f} seconds".format(end - start))

**OUTPUT :**



**GITHUB :**

(Link Pasted)