Practical 5

Name	Shreyash Yadav
Section and Batch	A4 - B3
Roll no.	41

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.

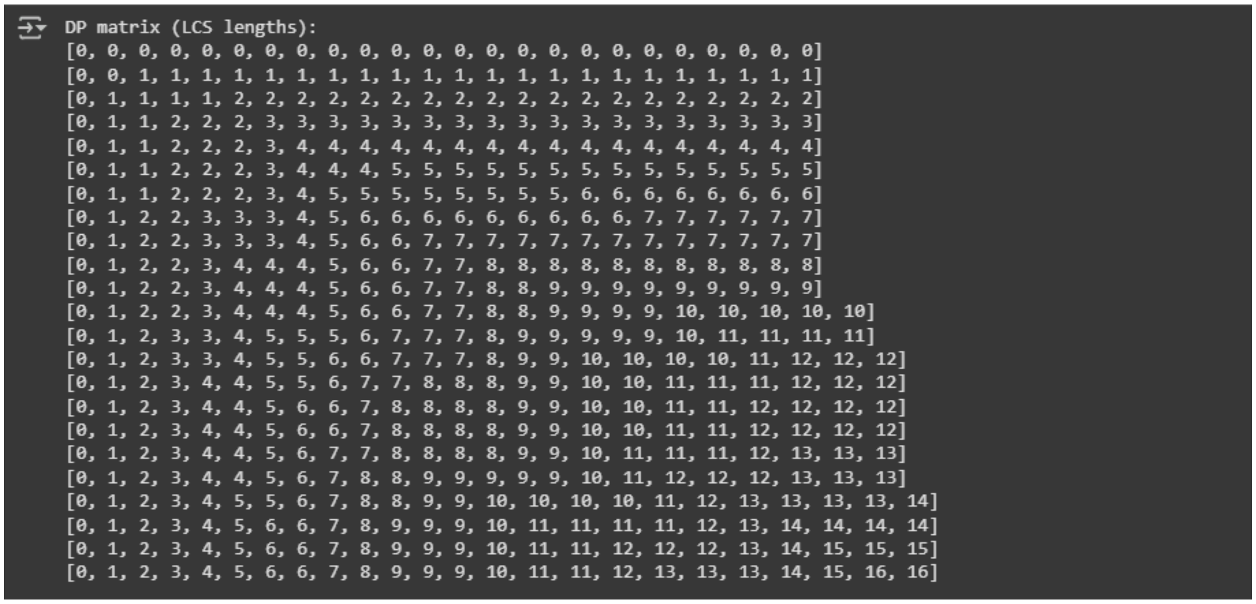
[Note that a subsequence might not include consecutive elements of the original sequence.]

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

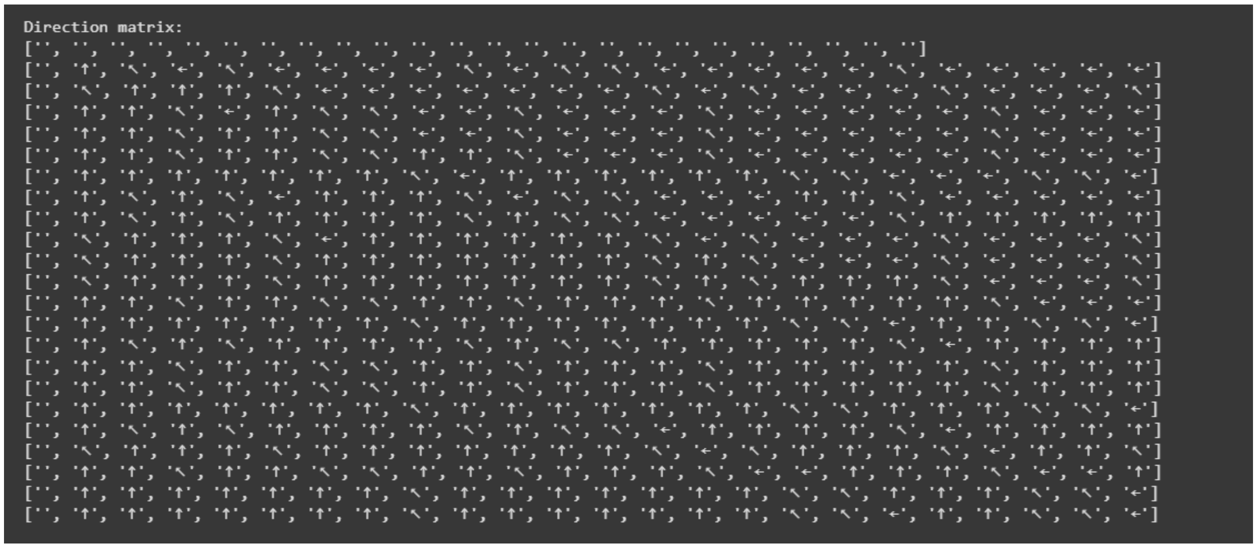
Code:

```python
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] = '↖'
            else:
                if 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_seq = []
    while i > 0 and j > 0:
        if direction[i][j] == '↖':
            lcs_seq.append(X[i-1])
            i -= 1
            j -= 1
        elif direction[i][j] == '↑':
            i -= 1
        else:  # '←'
            j -= 1

    lcs_seq.reverse()

    print("DP matrix (LCS lengths):")
    for row in dp:
        print(row)
    print("\nDirection matrix:")
    for row in direction:
        print(row)

    print(f"\nLength of LCS = {dp[m][n]}")
    print(f"LCS = {''.join(lcs_seq)}")

    return dp[m][n], ''.join(lcs_seq)

X = "AGCCCTAAGGGCTACCTAGCTT"
Y = "GACAGCCTACAAGCGTTAGCTTG"

lcs_length, lcs_sequence = lcs(X, Y)
```

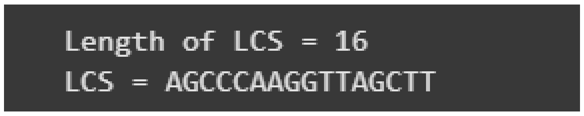
Output:



```
DP matrix (LCS lengths):
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
[0, 1, 1, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]
[0, 1, 1, 2, 2, 2, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]
[0, 1, 1, 2, 2, 2, 3, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5]
[0, 1, 1, 2, 2, 2, 3, 4, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6]
[0, 1, 2, 2, 3, 3, 3, 4, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 7, 7, 7, 7, 7, 7]
[0, 1, 2, 2, 3, 3, 3, 4, 5, 6, 6, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7]
[0, 1, 2, 2, 3, 4, 4, 4, 5, 6, 6, 7, 7, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8]
[0, 1, 2, 2, 3, 4, 4, 4, 5, 6, 6, 7, 7, 8, 8, 9, 9, 9, 9, 9, 9, 9, 9, 9]
[0, 1, 2, 2, 3, 4, 4, 4, 5, 6, 6, 7, 7, 8, 8, 9, 9, 9, 9, 10, 10, 10, 10, 10]
[0, 1, 2, 3, 3, 4, 5, 5, 5, 6, 7, 7, 7, 8, 9, 9, 9, 9, 9, 10, 11, 11, 11, 11]
[0, 1, 2, 3, 3, 4, 5, 5, 6, 6, 7, 7, 7, 8, 9, 9, 10, 10, 10, 10, 11, 12, 12, 12]
[0, 1, 2, 3, 4, 4, 5, 5, 6, 7, 7, 8, 8, 8, 9, 9, 10, 10, 11, 11, 11, 12, 12, 12]
[0, 1, 2, 3, 4, 4, 5, 6, 6, 7, 8, 8, 8, 8, 9, 9, 10, 10, 11, 11, 12, 12, 12, 12]
[0, 1, 2, 3, 4, 4, 5, 6, 6, 7, 8, 8, 8, 8, 9, 9, 10, 10, 11, 11, 12, 12, 12, 12]
[0, 1, 2, 3, 4, 4, 5, 6, 7, 7, 8, 8, 8, 8, 9, 9, 10, 11, 11, 11, 12, 13, 13, 13]
[0, 1, 2, 3, 4, 4, 5, 6, 7, 8, 8, 9, 9, 9, 9, 9, 10, 11, 12, 12, 12, 13, 13, 13]
[0, 1, 2, 3, 4, 5, 5, 6, 7, 8, 8, 9, 9, 10, 10, 10, 10, 11, 12, 13, 13, 13, 13, 14]
[0, 1, 2, 3, 4, 5, 6, 6, 7, 8, 9, 9, 9, 10, 11, 11, 11, 11, 12, 13, 14, 14, 14, 14]
[0, 1, 2, 3, 4, 5, 6, 6, 7, 8, 9, 9, 9, 10, 11, 11, 12, 12, 12, 13, 14, 15, 15, 15]
[0, 1, 2, 3, 4, 5, 6, 6, 7, 8, 9, 9, 9, 10, 11, 11, 12, 13, 13, 13, 14, 15, 16, 16]
```



```
Direction matrix:
['', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '']
['', '↑', '↖', '←', '↖', '←', '←', '←', '←', '↖', '←', '↖', '↖', '←', '←', '←', '←', '←', '↖', '←', '←', '←', '←', '←']
['', '↖', '↑', '↑', '↑', '↖', '←', '←', '←', '←', '←', '←', '←', '↖', '←', '↖', '←', '←', '←', '↖', '←', '←', '←', '↖']
['', '↑', '↑', '↖', '←', '↑', '↖', '↖', '←', '←', '↖', '←', '←', '←', '↖', '←', '←', '←', '←', '←', '↖', '←', '←', '←']
['', '↑', '↑', '↖', '↑', '↑', '↖', '↖', '←', '←', '↖', '←', '←', '←', '↖', '←', '←', '←', '←', '←', '↖', '←', '←', '←']
['', '↑', '↑', '↖', '↑', '↑', '↖', '↖', '↑', '↑', '↖', '←', '←', '←', '↖', '←', '←', '←', '←', '←', '↖', '←', '←', '←']
['', '↑', '↑', '↑', '↑', '↑', '↑', '↑', '↖', '←', '↑', '↑', '↑', '↑', '↑', '↑', '↖', '↖', '←', '←', '←', '↖', '↖', '←']
['', '↑', '↖', '↑', '↖', '←', '↑', '↑', '↑', '↖', '←', '↖', '↖', '←', '←', '←', '↑', '↑', '↖', '←', '←', '←', '←', '←']
['', '↑', '↖', '↑', '↖', '↑', '↑', '↑', '↑', '↖', '↑', '↖', '↖', '←', '←', '←', '←', '←', '↖', '↑', '↑', '↑', '↑', '↑']
['', '↖', '↑', '↑', '↑', '↖', '←', '↑', '↑', '↑', '↑', '↑', '↑', '↖', '←', '↖', '←', '←', '←', '↖', '←', '←', '←', '↖']
['', '↖', '↑', '↑', '↑', '↖', '↑', '↑', '↑', '↑', '↑', '↑', '↑', '↖', '↑', '↖', '←', '←', '←', '↖', '←', '←', '←', '↖']
['', '↖', '↑', '↑', '↑', '↖', '↑', '↑', '↑', '↑', '↑', '↑', '↑', '↖', '↑', '↖', '↑', '↑', '↑', '↖', '←', '←', '←', '↖']
['', '↑', '↑', '↖', '↑', '↑', '↖', '↖', '↑', '↑', '↖', '↑', '↑', '↑', '↖', '↑', '↑', '↑', '↑', '↑', '↖', '←', '←', '←']
['', '↑', '↑', '↑', '↑', '↑', '↑', '↑', '↖', '↑', '↑', '↑', '↑', '↑', '↑', '↑', '↖', '↖', '←', '↑', '↑', '↖', '↖', '←']
['', '↑', '↖', '↑', '↖', '↑', '↑', '↑', '↑', '↖', '↑', '↖', '↖', '↑', '↑', '↑', '↑', '↑', '↖', '←', '↑', '↑', '↑', '↑']
['', '↑', '↑', '↖', '↑', '↑', '↖', '↖', '↑', '↑', '↖', '↑', '↑', '↑', '↖', '↑', '↑', '↑', '↑', '↑', '↖', '↑', '↑', '↑']
['', '↑', '↑', '↖', '↑', '↑', '↖', '↖', '↑', '↑', '↖', '↑', '↑', '↑', '↖', '↑', '↑', '↑', '↑', '↑', '↖', '↑', '↑', '↑']
['', '↑', '↑', '↑', '↑', '↑', '↑', '↑', '↖', '↑', '↑', '↑', '↑', '↑', '↑', '↑', '↖', '↖', '↑', '↑', '↑', '↖', '↖', '←']
['', '↑', '↖', '↑', '↖', '↑', '↑', '↑', '↑', '↖', '↑', '↖', '↖', '←', '↑', '↑', '↑', '↑', '↖', '←', '↑', '↑', '↑', '↑']
['', '↖', '↑', '↑', '↑', '↖', '↑', '↑', '↑', '↑', '↑', '↑', '↑', '↖', '←', '↖', '↑', '↑', '↑', '↖', '←', '↑', '↑', '↖']
['', '↑', '↑', '↖', '↑', '↑', '↖', '↖', '↑', '↑', '↖', '↑', '↑', '↑', '↖', '←', '←', '↑', '↑', '↑', '↖', '←', '←', '↑']
['', '↑', '↑', '↑', '↑', '↑', '↑', '↑', '↖', '↑', '↑', '↑', '↑', '↑', '↑', '↑', '↖', '↖', '↑', '↑', '↑', '↖', '↖', '←']
['', '↑', '↑', '↑', '↑', '↑', '↑', '↑', '↖', '↑', '↑', '↑', '↑', '↑', '↑', '↑', '↖', '↖', '←', '↑', '↑', '↖', '↖', '←']
```



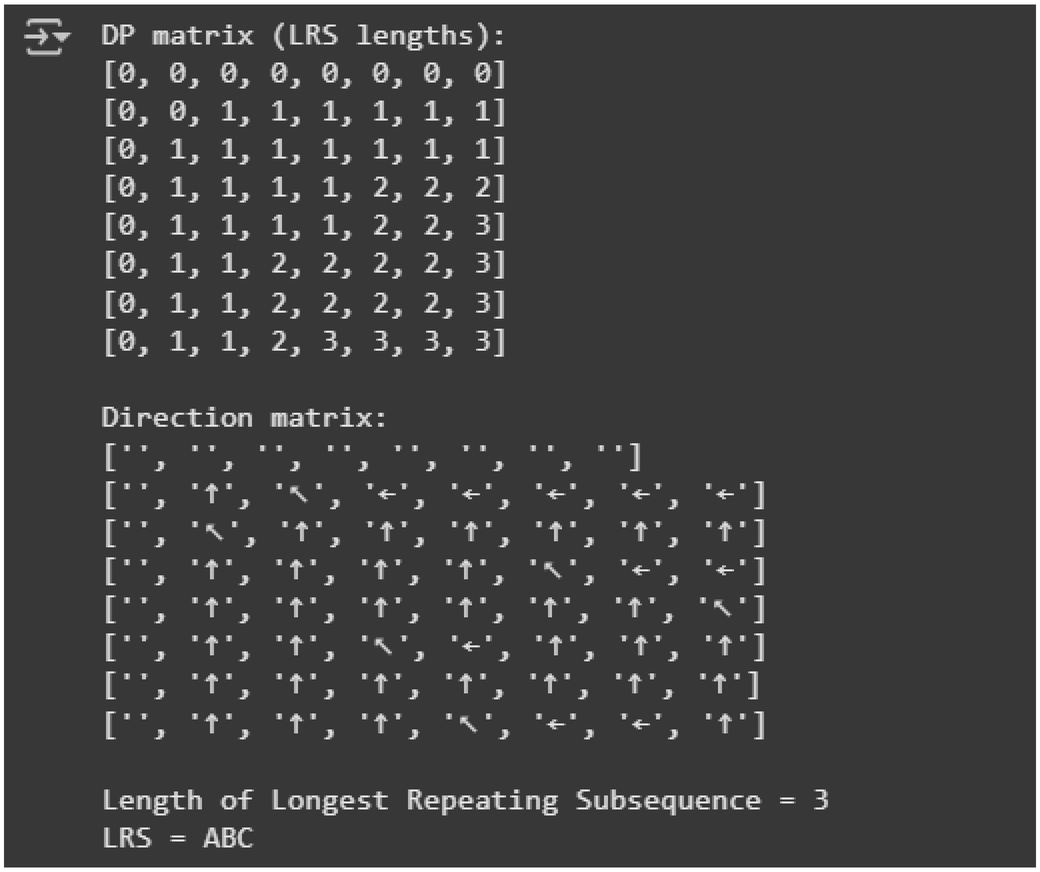
```
Length of LCS = 16
LCS = AGCCCAAGGTTAGCTT
```

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.

Code:

```python
def longest_repeating_subsequence(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] = dp[i-1][j-1] + 1
                direction[i][j] = '↖'
            else:
                if 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 = n, n
    lrs_seq = []
    while i > 0 and j > 0:
        if direction[i][j] == '↖':
            lrs_seq.append(S[i-1])
            i -= 1
            j -= 1
        elif direction[i][j] == '↑':
            i -= 1
        else:
            j -= 1

    lrs_seq.reverse()

    print("DP matrix (LRS lengths):")
    for row in dp:
        print(row)
    print("\nDirection matrix:")
    for row in direction:
        print(row)

    print(f"\nLength of Longest Repeating Subsequence = {dp[n][n]}")
    print(f"LRS = {''.join(lrs_seq)}")

    return dp[n][n], ''.join(lrs_seq)

S = "AABCBDC"
lrs_length, lrs_sequence = longest_repeating_subsequence(S)
```

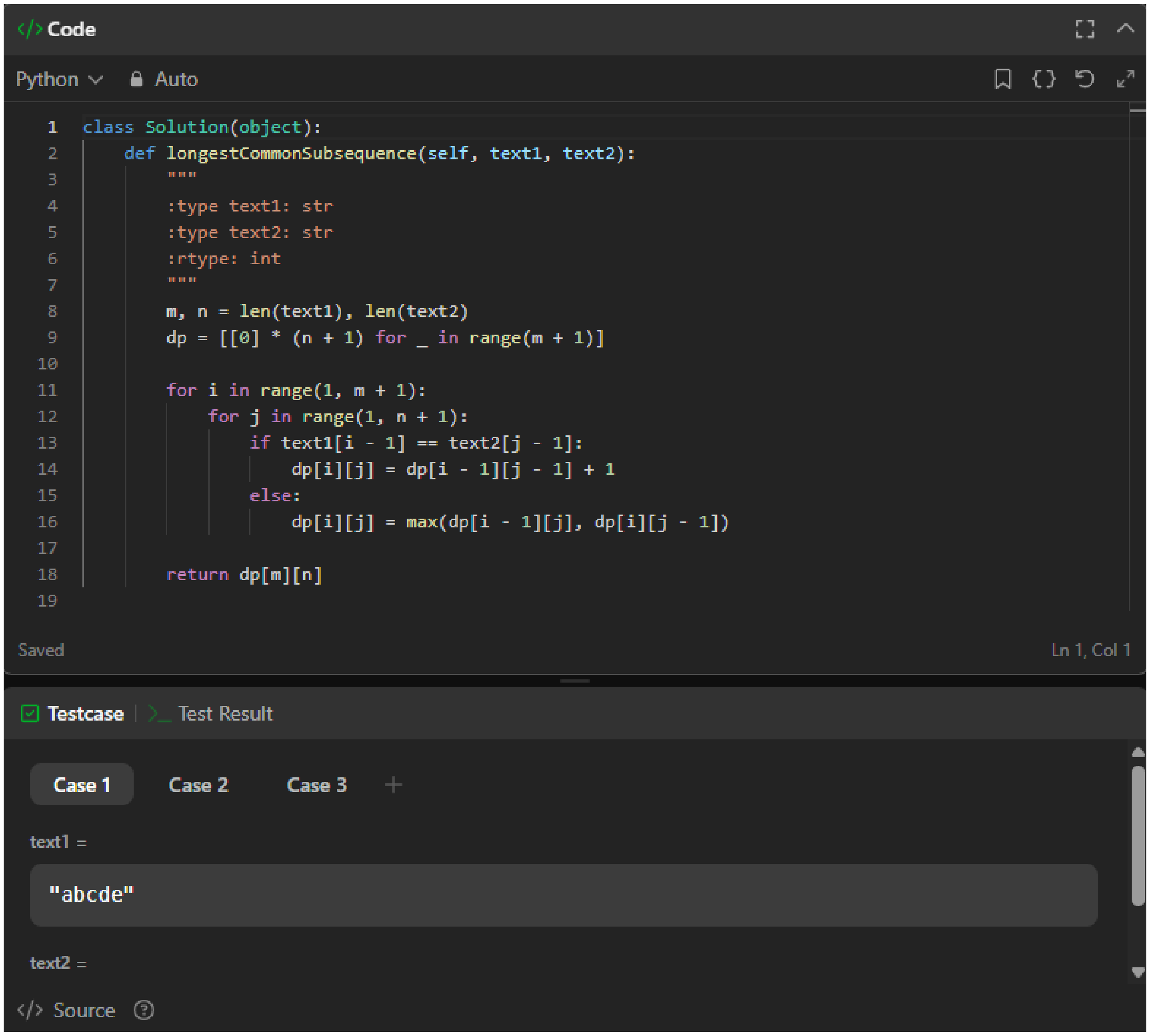
Output:



```
DP matrix (LRS lengths):
[0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 1, 1, 1, 1, 1]
[0, 1, 1, 1, 1, 1, 1, 1]
[0, 1, 1, 1, 1, 2, 2, 2]
[0, 1, 1, 1, 1, 2, 2, 3]
[0, 1, 1, 2, 2, 2, 2, 3]
[0, 1, 1, 2, 2, 2, 2, 3]
[0, 1, 1, 2, 3, 3, 3, 3]

Direction matrix:
['', '', '', '', '', '', '', '']
['', '↑', '↖', '←', '←', '←', '←', '←']
['', '↖', '↑', '↑', '↑', '↑', '↑', '↑']
['', '↑', '↑', '↑', '↑', '↖', '←', '←']
['', '↑', '↑', '↑', '↑', '↑', '↑', '↖']
['', '↑', '↑', '↖', '←', '↑', '↑', '↑']
['', '↑', '↑', '↑', '↑', '↑', '↑', '↑']
['', '↑', '↑', '↑', '↖', '←', '←', '↑']

Length of Longest Repeating Subsequence = 3
LRS = ABC
```

LeetCode Assessment:
https://leetcode.com/problems/longest-common-subsequence/description/

Execution:

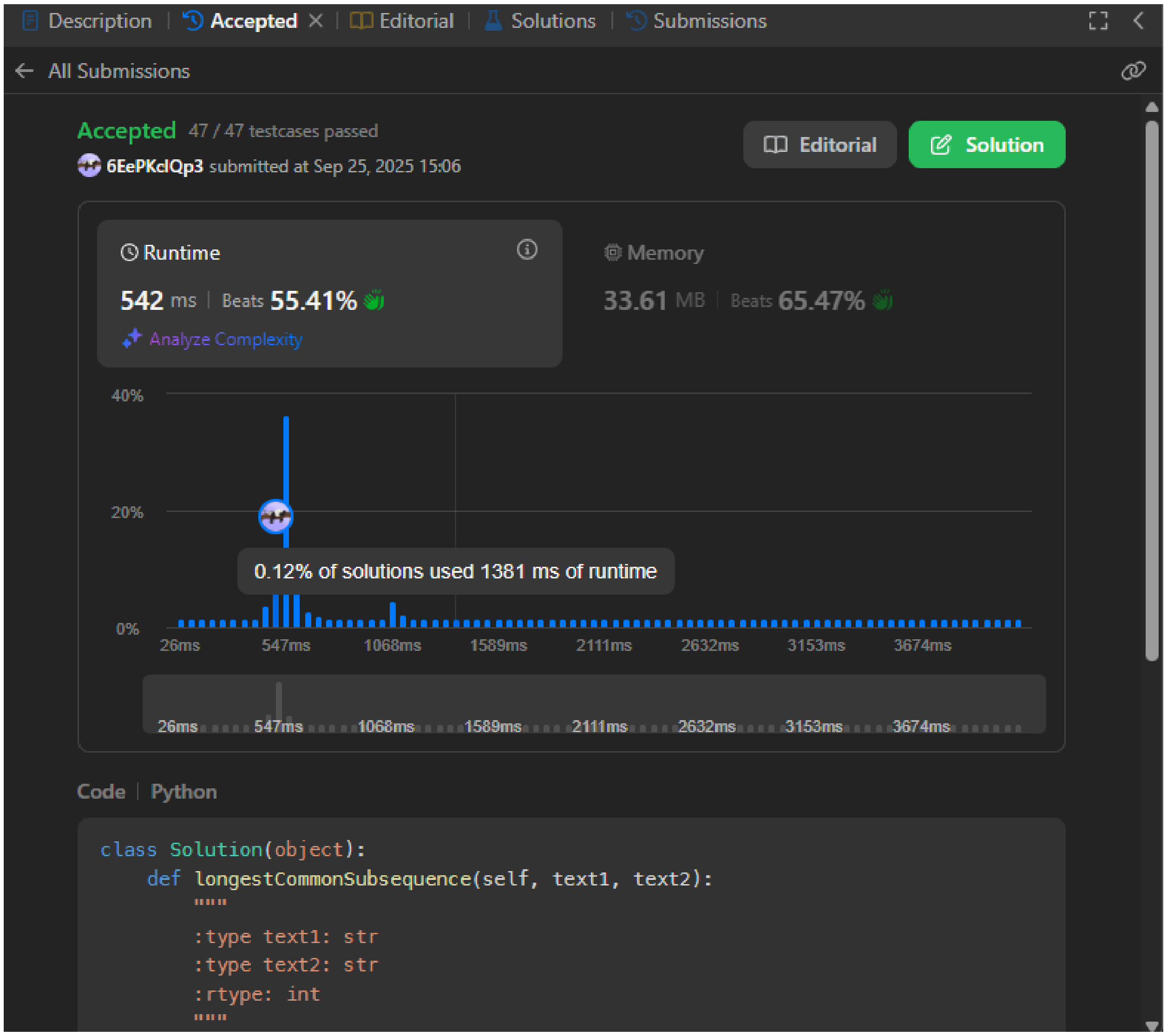


```python
class Solution(object):
    def longestCommonSubsequence(self, text1, text2):
        """
        :type text1: str
        :type text2: str
        :rtype: int
        """
        m, n = len(text1), len(text2)
        dp = [[0] * (n + 1) for _ in range(m + 1)]

        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]

```

text1 =

"abcde"

text2 =



Accepted 47 / 47 testcases passed

6EePKclQp3 submitted at Sep 25, 2025 15:06

Runtime

542 ms | Beats 55.41%

Memory

33.61 MB | Beats 65.47%

0.12% of solutions used 1381 ms of runtime

Code | Python

```
class Solution(object):
    def longestCommonSubsequence(self, text1, text2):
        """
        :type text1: str
        :type text2: str
        :rtype: int
        """
```