PRACTICAL NO. 5

Name : Anjali Lanjewar
Section: A4-B2
Roll no.-25

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.

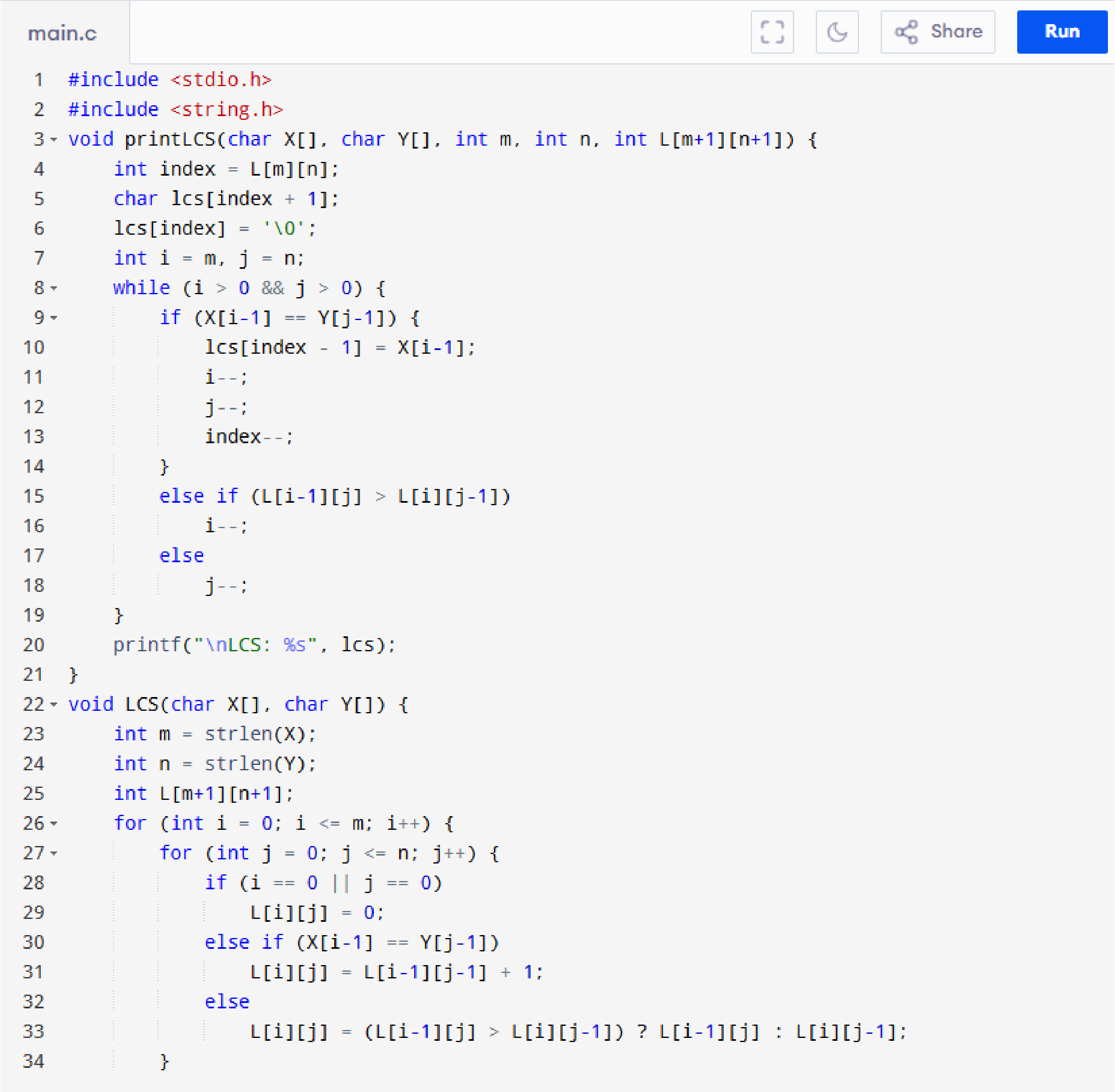
X=AGCCCTAAGGGCTACCTAGCTT

Y= GACAGCCTACAAGCGTTAGCTTG

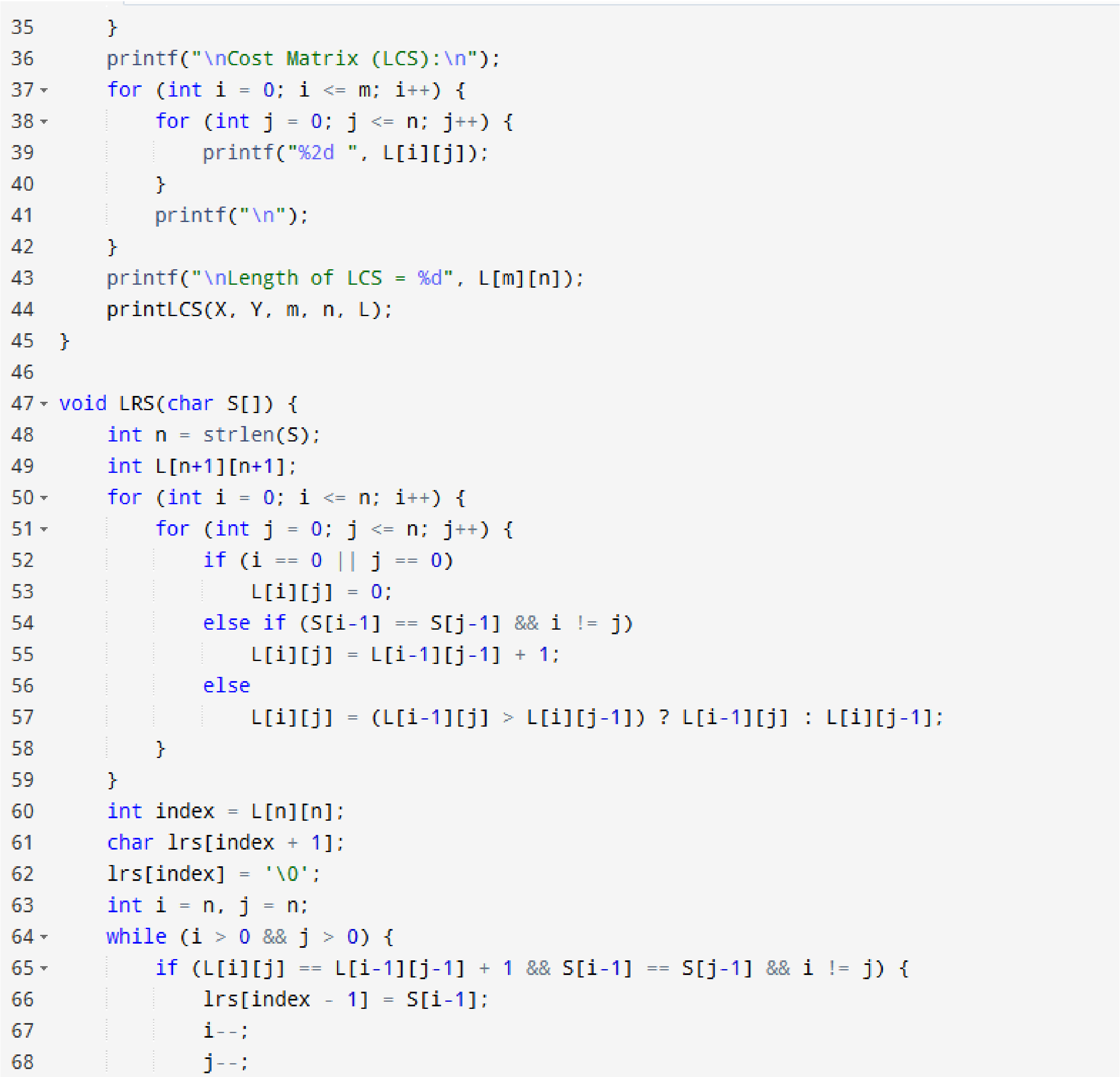
Output: Cost matrix with all costs and direction, final cost of LCS and the LCS.

Length of LCS=16

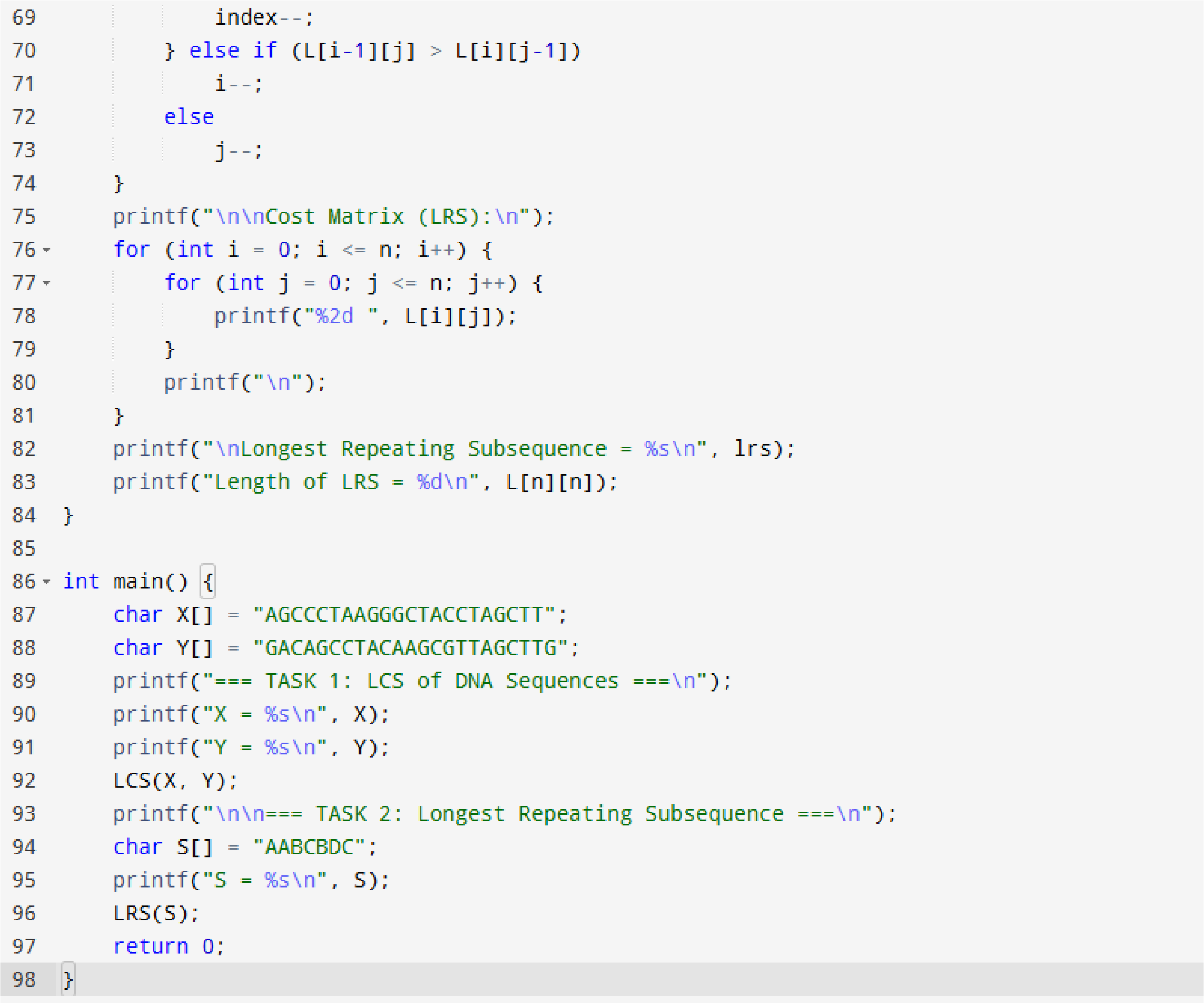
CODE :



```c
#include <stdio.h>
#include <string.h>
void printLCS(char X[], char Y[], int m, int n, int L[m+1][n+1]) {
    int index = L[m][n];
    char lcs[index + 1];
    lcs[index] = '\0';
    int i = m, j = n;
    while (i > 0 && j > 0) {
        if (X[i-1] == Y[j-1]) {
            lcs[index - 1] = X[i-1];
            i--;
            j--;
            index--;
        }
        else if (L[i-1][j] > L[i][j-1])
            i--;
        else
            j--;
    }
    printf("\nLCS: %s", lcs);
}
void LCS(char X[], char Y[]) {
    int m = strlen(X);
    int n = strlen(Y);
    int L[m+1][n+1];
    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            if (i == 0 || j == 0)
                L[i][j] = 0;
            else if (X[i-1] == Y[j-1])
                L[i][j] = L[i-1][j-1] + 1;
            else
                L[i][j] = (L[i-1][j] > L[i][j-1]) ? L[i-1][j] : L[i][j-1];
        }
```

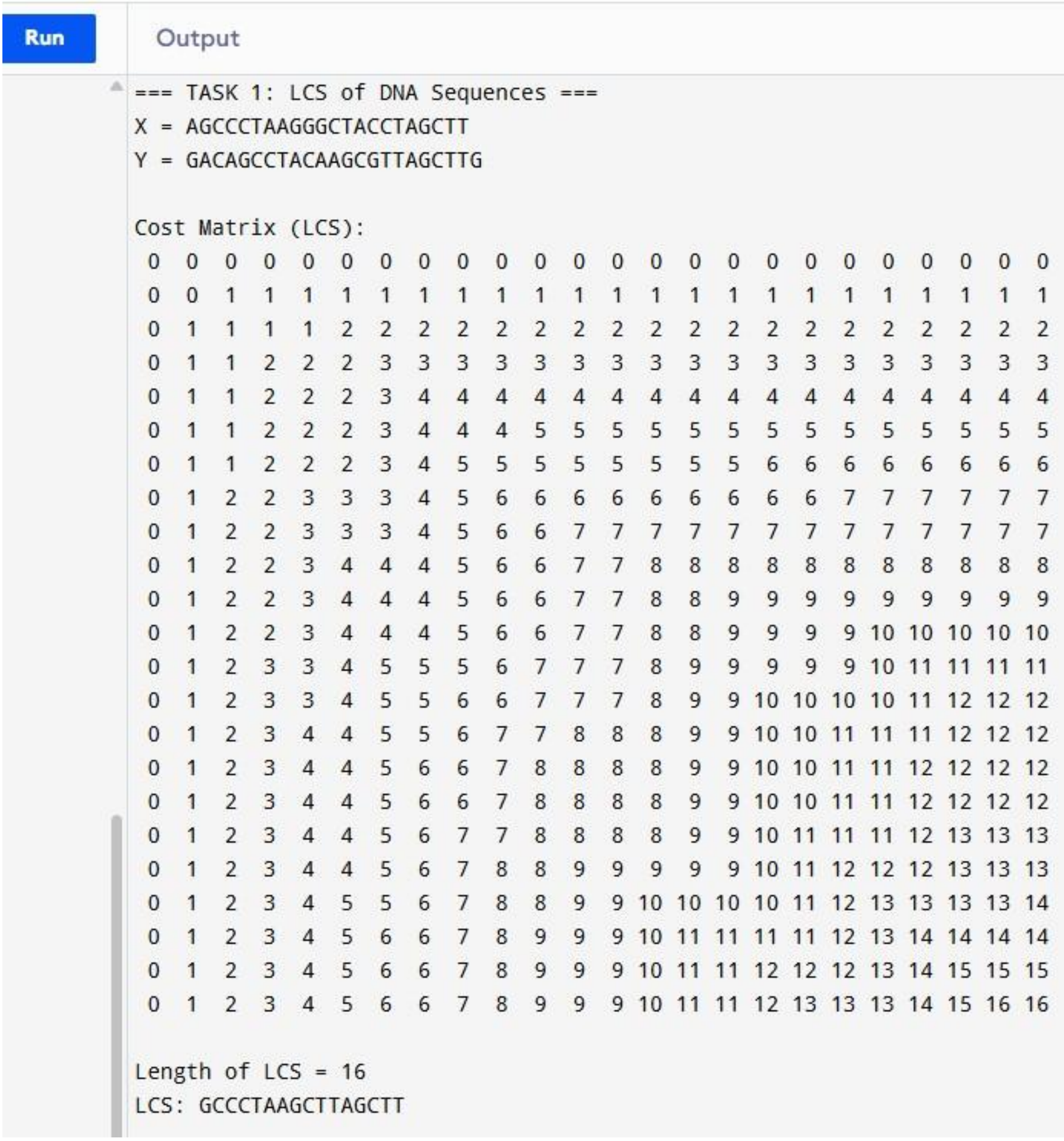


```c
    }
    printf("\nCost Matrix (LCS):\n");
    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            printf("%2d ", L[i][j]);
        }
        printf("\n");
    }
    printf("\nLength of LCS = %d", L[m][n]);
    printLCS(X, Y, m, n, L);
}

void LRS(char S[]) {
    int n = strlen(S);
    int L[n+1][n+1];
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= n; j++) {
            if (i == 0 || j == 0)
                L[i][j] = 0;
            else if (S[i-1] == S[j-1] && i != j)
                L[i][j] = L[i-1][j-1] + 1;
            else
                L[i][j] = (L[i-1][j] > L[i][j-1]) ? L[i-1][j] : L[i][j-1];
        }
    }
    int index = L[n][n];
    char lrs[index + 1];
    lrs[index] = '\0';
    int i = n, j = n;
    while (i > 0 && j > 0) {
        if (L[i][j] == L[i-1][j-1] + 1 && S[i-1] == S[j-1] && i != j) {
            lrs[index - 1] = S[i-1];
            i--;
            j--;
```



```c
            index--;
        } else if (L[i-1][j] > L[i][j-1])
            i--;
        else
            j--;
    }
    printf("\n\nCost Matrix (LRS):\n");
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= n; j++) {
            printf("%2d ", L[i][j]);
        }
        printf("\n");
    }
    printf("\nLongest Repeating Subsequence = %s\n", lrs);
    printf("Length of LRS = %d\n", L[n][n]);
}

int main() {
    char X[] = "AGCCCTAAGGGCTACCTAGCTT";
    char Y[] = "GACAGCCTACAAGCGTTAGCTTG";
    printf("=== TASK 1: LCS of DNA Sequences ===\n");
    printf("X = %s\n", X);
    printf("Y = %s\n", Y);
    LCS(X, Y);
    printf("\n\n=== TASK 2: Longest Repeating Subsequence ===\n");
    char S[] = "AABCBDC";
    printf("S = %s\n", S);
    LRS(S);
    return 0;
}
```

OUTPUT :



```
=== TASK 1: LCS of DNA Sequences ===
X = AGCCCTAAGGGCTACCTAGCTT
Y = GACAGCCTACAAGCGTTAGCTTG

Cost Matrix (LCS):
  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

Length of LCS = 16
LCS: GCCCTAAGCTTAGCTT
```

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

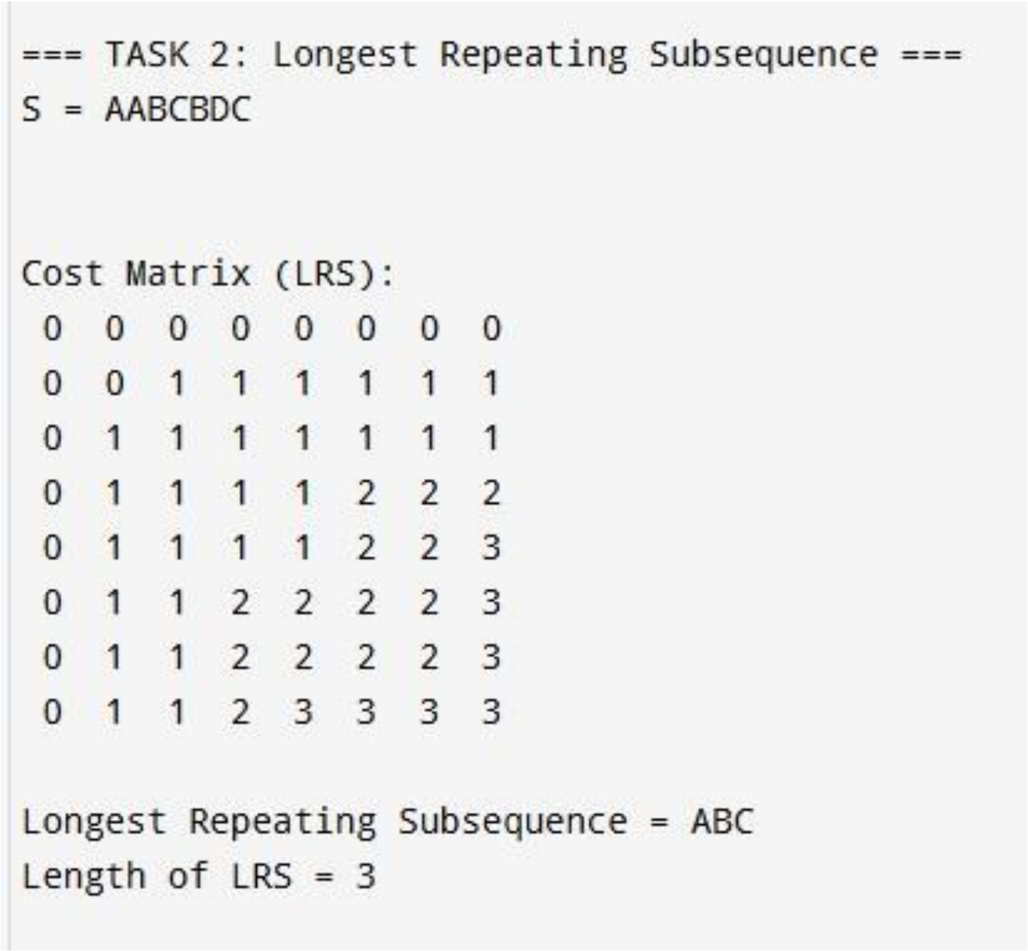
Let the given string be S. You need to find the LRS within S. To use the LCS framework, you

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

Example:

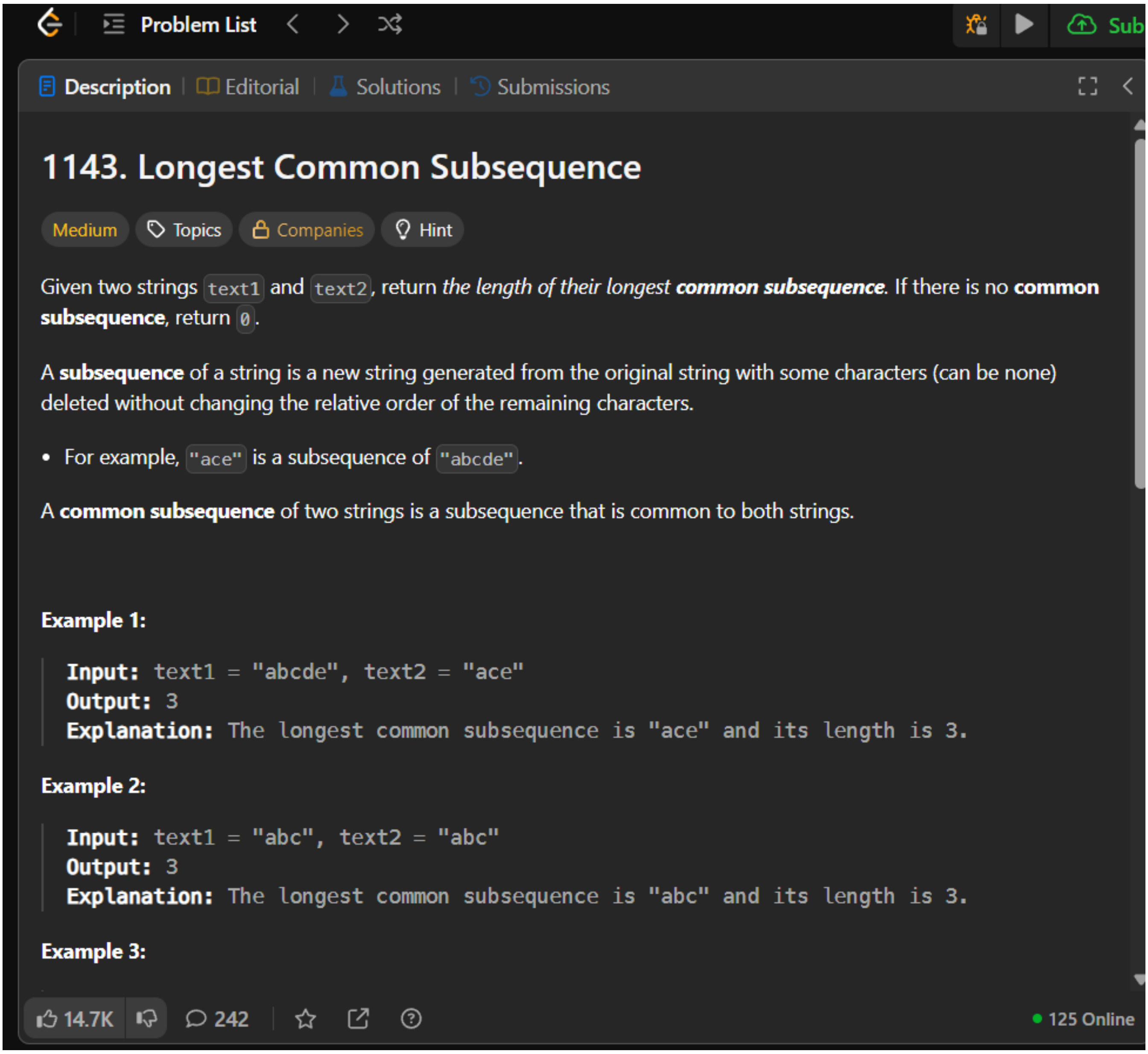
AABCBDC

LRS= ABC or ABD



```
=== TASK 2: Longest Repeating Subsequence ===
S = AABCBDC


Cost Matrix (LRS):
 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

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

LeetCode Assessment :



1143. Longest Common Subsequence

Given two strings `text1` and `text2`, return *the length of their longest **common subsequence***. If there is no **common subsequence**, return `0`.

A **subsequence** of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of the remaining characters.

- For example, `"ace"` is a subsequence of `"abcde"`.

A **common subsequence** of two strings is a subsequence that is common to both strings.

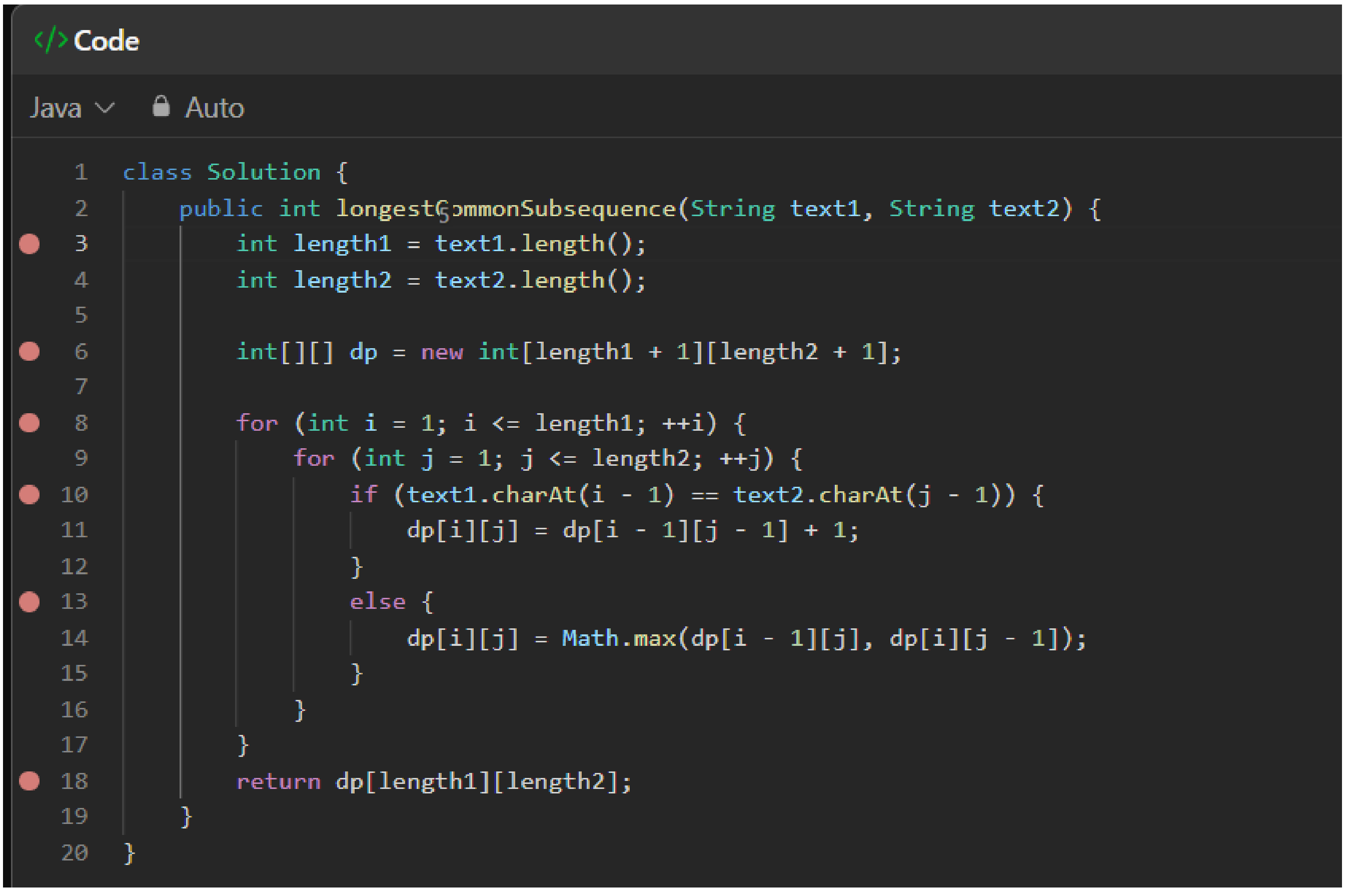
Example 1:

```
Input: text1 = "abcde", text2 = "ace"
Output: 3
Explanation: The longest common subsequence is "ace" and its length is 3.
```

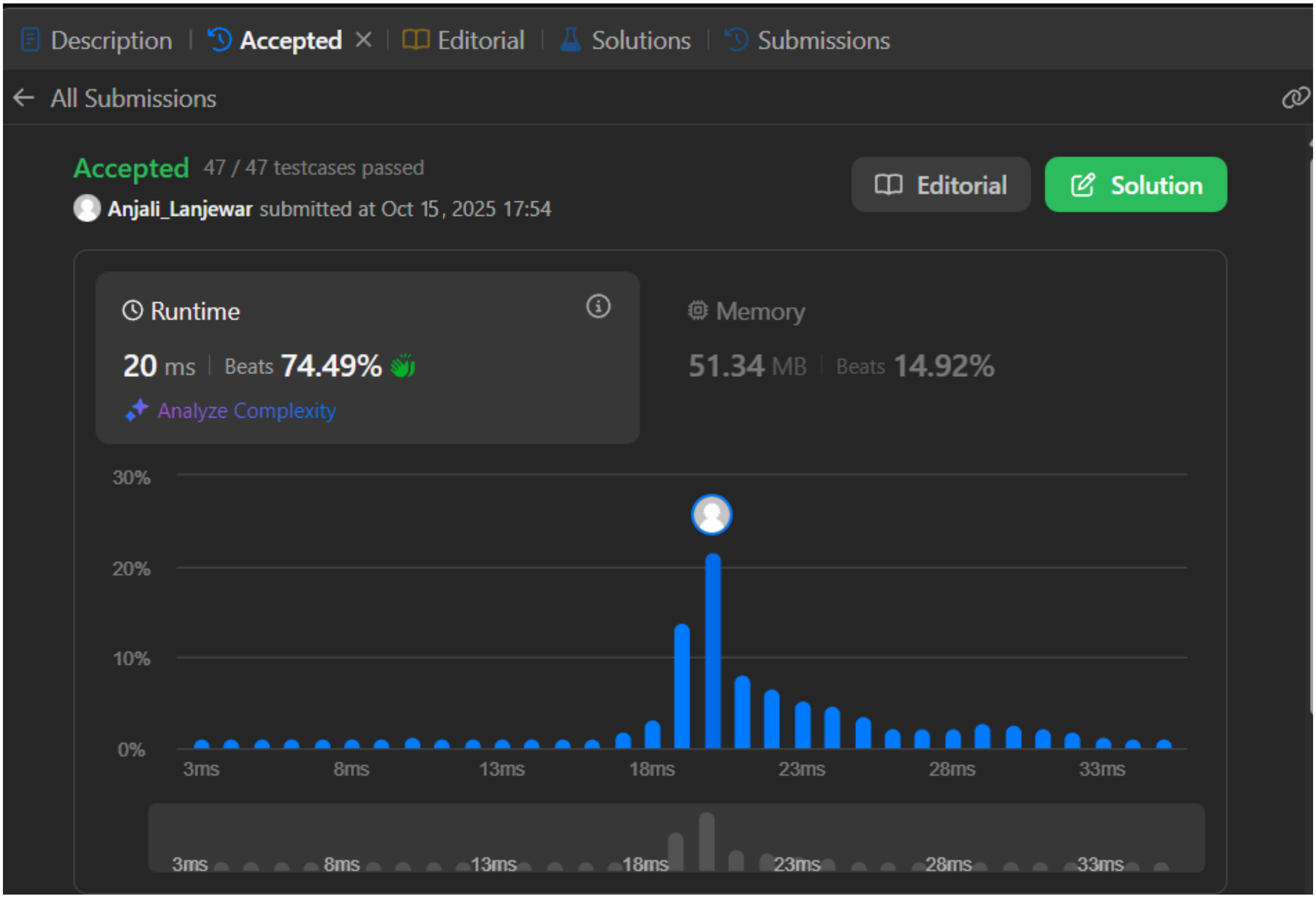
Example 2:

```
Input: text1 = "abc", text2 = "abc"
Output: 3
Explanation: The longest common subsequence is "abc" and its length is 3.
```

Example 3:



```java
class Solution {
    public int longestCommonSubsequence(String text1, String text2) {
        int length1 = text1.length();
        int length2 = text2.length();

        int[][] dp = new int[length1 + 1][length2 + 1];

        for (int i = 1; i <= length1; ++i) {
            for (int j = 1; j <= length2; ++j) {
                if (text1.charAt(i - 1) == text2.charAt(j - 1)) {
                    dp[i][j] = dp[i - 1][j - 1] + 1;
                }
                else {
                    dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
                }
            }
        }
        return dp[length1][length2];
    }
}
```



Accepted 47 / 47 testcases passed

Anjali_Lanjewar submitted at Oct 15, 2025 17:54

Runtime

20 ms | Beats 74.49%

Memory

51.34 MB | Beats 14.92%