Min Cost to make strings identical C++

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;

int minCostToMakeIdentical(string s1, string s2, int
c1, int c2) {
    int m = s1.length();
    int n = s2.length();

    // Initialize dp array with size (m+1)x(n+1)
    vector<vector<int>> dp(m + 1, vector<int>(n + 1,
0));

    // Fill dp array
    for (int i = m - 1; i >= 0; i--) {
        for (int j = n - 1; j >= 0; j--) {
            if (s1[i] == s2[j]) {
                dp[i][j] = 1 + dp[i + 1][j + 1];
            } else {
                dp[i][j] = max(dp[i + 1][j], dp[i][j + 1]);
            }
        }
    }

    // Calculate length of LCS
    int lcsLength = dp[0][0];
    cout << "Length of Longest Common Subsequence:
" << lcsLength << endl;

    // Calculate remaining characters in s1 and s2 after
LCS
    int s1Remaining = m - lcsLength;
    int s2Remaining = n - lcsLength;

    // Calculate minimum cost to make strings identical
    int cost = s1Remaining * c1 + s2Remaining * c2;
    return cost;
}

int main() {
    string s1 = "cat";
    string s2 = "cut";
    int c1 = 1;
    int c2 = 1;

    int minCost = minCostToMakeIdentical(s1, s2, c1,
c2);
    cout << "Minimum cost to make strings identical: "
<< minCost << endl;

    return 0;
}
```

Initial Setup:

We have:

- s1 = "cat"
- s2 = "cut"
- c1 = 1 (cost to remove a character from s1)
- c2 = 1 (cost to remove a character from s2)

The goal is to find the **Longest Common Subsequence (LCS)** and then calculate the minimum cost of making the two strings identical by removing characters from them.

Step 1: Initialize DP Table

We initialize the DP table with dimensions (m+1) x (n+1), where m is the length of s1 and n is the length of s2.

- m = 3 (length of s1)
- n = 3 (length of s2)

So, the DP table will be a 4x4 matrix (since we include the 0th index for base cases).

DP Table (Initial):
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]

Step 2: Fill DP Table to Calculate LCS Length

We fill the DP table using the following logic:

- If s1[i] == s2[j], then dp[i][j] = dp[i + 1][j + 1] + 1 (this means the current characters match, and we can extend the LCS).
- If s1[i] != s2[j], then dp[i][j] = max(dp[i + 1][j], dp[i][j + 1]) (this means we have to choose the maximum LCS length from skipping one character from either string).

Now, let's fill the DP table step by step.

1. **For i = 2 and j = 2**:
    - s1[2] = "t" and s2[2] = "t", they match, so dp[2][2] = dp[3][3] + 1 = 0 + 1 = 1.

   DP Table after filling dp[2][2]:
   [0, 0, 0, 0]
   [0, 0, 0, 0]
   [0, 0, 1, 0]
   [0, 0, 0, 0]

2. **For i = 2 and j = 1**:

- s1[2] = "t" and s2[1] = "u", they do not match, so dp[2][1] = max(dp[3][1], dp[2][2]) = max(0, 1) = 1.

DP Table after filling dp[2][1]:
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 1, 1, 0]
[0, 0, 0, 0]

3. **For i = 2 and j = 0**:
   - s1[2] = "t" and s2[0] = "c", they do not match, so dp[2][0] = max(dp[3][0], dp[2][1]) = max(0, 1) = 1.

DP Table after filling dp[2][0]:
[0, 0, 0, 0]
[0, 0, 0, 0]
[1, 1, 1, 0]
[0, 0, 0, 0]

4. **For i = 1 and j = 2**:
   - s1[1] = "a" and s2[2] = "t", they do not match, so dp[1][2] = max(dp[2][2], dp[1][3]) = max(1, 0) = 1.

DP Table after filling dp[1][2]:
[0, 0, 0, 0]
[0, 0, 1, 0]
[1, 1, 1, 0]
[0, 0, 0, 0]

5. **For i = 1 and j = 1**:
   - s1[1] = "a" and s2[1] = "u", they do not match, so dp[1][1] = max(dp[2][1], dp[1][2]) = max(1, 1) = 1.

DP Table after filling dp[1][1]:
[0, 0, 0, 0]
[0, 1, 1, 0]
[1, 1, 1, 0]
[0, 0, 0, 0]

6. **For i = 1 and j = 0**:
   - s1[1] = "a" and s2[0] = "c", they do not match, so dp[1][0] = max(dp[2][0], dp[1][1]) = max(1, 1) = 1.

DP Table after filling dp[1][0]:
[0, 0, 0, 0]
[1, 1, 1, 0]
[1, 1, 1, 0]
[0, 0, 0, 0]

7. **For i = 0 and j = 2**:
   - s1[0] = "c" and s2[2] = "t", they do not match, so dp[0][2] = max(dp[1][2], dp[0][3]) = max(1, 0) = 1.

DP Table after filling dp[0][2]:

	[0, 0, 1, 0] [1, 1, 1, 0] [1, 1, 1, 0] [0, 0, 0, 0] 8. **For i = 0 and j = 1**: ○ s1[0] = "c" and s2[1] = "u", they do not match, so dp[0][1] = max(dp[1][1], dp[0][2]) = max(1, 1) = 1. DP Table after filling dp[0][1]: [0, 1, 1, 0] [1, 1, 1, 0] [1, 1, 1, 0] [0, 0, 0, 0] 9. **For i = 0 and j = 0**: ○ s1[0] = "c" and s2[0] = "c", they match, so dp[0][0] = dp[1][1] + 1 = 1 + 1 = 2. DP Table after filling dp[0][0]: [2, 1, 1, 0] [1, 1, 1, 0] [1, 1, 1, 0] [0, 0, 0, 0] **Step 3: Calculate LCS Length** After filling the DP table, the length of the **Longest Common Subsequence (LCS)** is found at dp[0][0], which is 2. This means the LCS of "cat" and "cut" is of length 2. **Step 4: Calculate the Cost** Now, we calculate the remaining characters in s1 and s2: • Remaining characters in s1: 3 - 2 = 1 (the character "a" needs to be removed). • Remaining characters in s2: 3 - 2 = 1 (the character "u" needs to be removed). The total cost is: $cost=1\times c1+1\times c2=1\times 1+1\times 1=2$
Output:- Length of Longest Common Subsequence: 2 Minimum cost to make strings identical: 2	