|  |  |
| --- | --- |
| **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;  } | **Step-by-Step DP Table Construction**  **Strings:**  s1 = "cat"  s2 = "cut"  We create a **(m+1) × (n+1) DP table**, where:   * dp[i][j] stores the **length of LCS of s1[i:] and s2[j:]**.   **DP Table Initialization (Bottom-Up)**   | **i\j** | **c** | **u** | **t** | **(empty)** | | --- | --- | --- | --- | --- | | **c** | ? | ? | ? | 0 | | **a** | ? | ? | ? | 0 | | **t** | ? | ? | ? | 0 | | **(empty)** | 0 | 0 | 0 | 0 |   **Filling the Table**  We start from the **bottom-right** and move **backwards**.   1. **Comparing 't' in s1 with 't' in s2:**   s1[2] == s2[2] ('t' == 't')   * + So, dp[2][2] = 1 + dp[3][3] = 1  1. **Comparing 't' in s1 with 'u' in s2:**   s1[2] != s2[1] ('t' ≠ 'u')   * + So, dp[2][1] = max(dp[3][1], dp[2][2]) = max(0, 1) = 1  1. **Comparing 't' in s1 with 'c' in s2:**   s1[2] != s2[0] ('t' ≠ 'c')   * + So, dp[2][0] = max(dp[3][0], dp[2][1]) = max(0, 1) = 1  1. **Comparing 'a' in s1 with 't' in s2:**   s1[1] != s2[2] ('a' ≠ 't')   * + So, dp[1][2] = max(dp[2][2], dp[1][3]) = max(1, 0) = 1  1. **Comparing 'a' in s1 with 'u' in s2:**   s1[1] != s2[1] ('a' ≠ 'u')   * + So, dp[1][1] = max(dp[2][1], dp[1][2]) = max(1, 1) = 1  1. **Comparing 'a' in s1 with 'c' in s2:**   s1[1] != s2[0] ('a' ≠ 'c')   * + So, dp[1][0] = max(dp[2][0], dp[1][1]) = max(1, 1) = 1  1. **Comparing 'c' in s1 with 't' in s2:**   s1[0] != s2[2] ('c' ≠ 't')   * + So, dp[0][2] = max(dp[1][2], dp[0][3]) = max(1, 0) = 1  1. **Comparing 'c' in s1 with 'u' in s2:**   s1[0] != s2[1] ('c' ≠ 'u')   * + So, dp[0][1] = max(dp[1][1], dp[0][2]) = max(1, 1) = 1  1. **Comparing 'c' in s1 with 'c' in s2:**   s1[0] == s2[0] ('c' == 'c')   * + So, dp[0][0] = 1 + dp[1][1] = 2   **Final DP Table**   | **i\j** | **c** | **u** | **t** | **(empty)** | | --- | --- | --- | --- | --- | | **c** | **2** | 1 | 1 | 0 | | **a** | 1 | 1 | 1 | 0 | | **t** | 1 | 1 | **1** | 0 | | **(empty)** | 0 | 0 | 0 | 0 |   **Final Calculation**   * **LCS Length** = dp[0][0] = 2 * **Remaining characters to delete:**   s1: "cat" → Remove 1 character ('a')  s2: "cut" → Remove 1 character ('u')   * **Total Cost:**   Cost = (1 × 1) + (1 × 1) = 1 + 1 = 2 |
| Output:- Length of Longest Common Subsequence: 2  Minimum cost to make strings identical: 2 | |