|  |  |
| --- | --- |
| **Longest Palindromic subseq In C++** | |
| #include <iostream>  #include <string>  #include <vector>  using namespace std;  int LongestPalindromicSubsequence(string str) {      int n = str.length();      //vector<vector<int>> dp(n, vector<int>(n, 0));      int dp[n][n]={0};      for (int g = 0; g < n; g++) {          for (int i = 0, j = g; j < n; i++, j++) {              if (g == 0) {                  dp[i][j] = 1;              } else if (g == 1) {                  dp[i][j] = (str[i] == str[j]) ? 2 : 1;              } else {                  if (str[i] == str[j]) {                      dp[i][j] = 2 + dp[i + 1][j - 1];                  } else {                      dp[i][j] = max(dp[i][j - 1], dp[i + 1][j]);                  }              }          }      }      return dp[0][n - 1];  }  int main() {      string str = "abccba";        int longestPalSubseqLen = LongestPalindromicSubsequence(str);      cout << longestPalSubseqLen << endl;      return 0;  } | **Step-by-Step Dry Run**  Let’s walk through each step of filling the DP table for the input string "abccba".  **Initial Setup**   * Length of string n = 6 * Initialize a 2D DP table dp[6][6] with all zeros.   **Step 1: Base Case for Substrings of Length 1**  When g == 0, each character is a subsequence of length 1.   |  | **a** | **b** | **c** | **c** | **b** | **a** | | --- | --- | --- | --- | --- | --- | --- | | **a** | 1 |  |  |  |  |  | | **b** |  | 1 |  |  |  |  | | **c** |  |  | 1 |  |  |  | | **c** |  |  |  | 1 |  |  | | **b** |  |  |  |  | 1 |  | | **a** |  |  |  |  |  | 1 |   **Step 2: Substrings of Length 2**  When g == 1, we check if adjacent characters match.   |  | **a** | **b** | **c** | **c** | **b** | **a** | | --- | --- | --- | --- | --- | --- | --- | | **a** | 1 | 1 |  |  |  |  | | **b** |  | 1 | 2 |  |  |  | | **c** |  |  | 1 | 2 |  |  | | **c** |  |  |  | 1 | 2 |  | | **b** |  |  |  |  | 1 | 2 | | **a** |  |  |  |  |  | 1 |   **Step 3: Substrings of Length 3 and Beyond**  For substrings of length greater than 2, we follow the general case rules.   | **g (Gap)** | **i** | **j** | **Formula Used** | **dp[i][j]** | | --- | --- | --- | --- | --- | | 2 | 0 | 2 | dp[1][1] + 2 (Match a == a) | 3 | | 2 | 1 | 3 | max(dp[1][2], dp[2][3]) (Max of 1 and 2) | 2 | | 2 | 2 | 4 | dp[3][3] + 2 (Match b == b) | 3 | | 2 | 3 | 5 | max(dp[3][4], dp[4][5]) (Max of 1 and 2) | 2 | | 3 | 0 | 3 | dp[1][2] + 2 (Match a == a) | 3 | | 3 | 1 | 4 | max(dp[1][3], dp[2][4]) (Max of 2 and 3) | 3 | | 3 | 2 | 5 | max(dp[2][4], dp[3][5]) (Max of 3 and 2) | 3 | | 4 | 0 | 4 | dp[1][3] + 2 (Match a == a) | 4 | | 4 | 1 | 5 | max(dp[1][4], dp[2][5]) (Max of 3 and 3) | 4 | | 5 | 0 | 5 | dp[1][4] + 2 (Match a == a) | 6 |   **Final DP Table**   |  | **a** | **b** | **c** | **c** | **b** | **a** | | --- | --- | --- | --- | --- | --- | --- | | **a** | 1 | 1 | 3 | 3 | 4 | 6 | | **b** |  | 1 | 2 | 2 | 3 | 4 | | **c** |  |  | 1 | 2 | 3 | 3 | | **c** |  |  |  | 1 | 2 | 3 | | **b** |  |  |  |  | 1 | 2 | | **a** |  |  |  |  |  | 1 |   **Final Answer**  The length of the **Longest Palindromic Subsequence** is stored in dp[0][n-1] = dp[0][5] = 6.  **Output**:  6 |
| Output:- 6 | |