// A Space optimized Dynamic Programming based C++

// program for LPS problem

#include <bits/stdc++.h>

using namespace std;

// Returns the length of the longest palindromic

// subsequence in str

int lps(string &s)

{

    int n = s.length();

    // a[i] is going to store length of longest

    // palindromic subsequence of substring s[0..i]

    int a[n];

    // Pick starting point

    for (int i = n - 1; i >= 0; i--) {

        int back\_up = 0;

        // Pick ending points and see if s[i]

        // increases length of longest common

        // subsequence ending with s[j].

        for (int j = i; j < n; j++) {

            // similar to 2D array L[i][j] == 1

            // i.e., handling substrings of length

            // one.

            if (j == i)

                a[j] = 1;

            // Similar to 2D array L[i][j] = L[i+1][j-1]+2

            // i.e., handling case when corner characters

            // are same.

            else if (s[i] == s[j])

            {

                // value a[j] is depend upon previous

                // unupdated value of a[j-1] but in

                // previous loop value of a[j-1] is

                // changed. To store the unupdated

                // value of a[j-1] back\_up variable

                // is used.

                int temp = a[j];

                a[j] = back\_up + 2;

                back\_up = temp;

            }

            // similar to 2D array L[i][j] = max(L[i][j-1],

            // a[i+1][j])

            else

            {

                back\_up = a[j];

                a[j] = max(a[j - 1], a[j]);

            }

        }

    }

    return a[n - 1];