Longest Common substring In C++

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

int LongestCommonSubstring(string s1, string
s2) {
    int m = s1.length();
    int n = s2.length();
    vector<vector<int>> dp(m + 1, vector<int>(n +
1, 0));
    //int dp[m+1][n+1]={0};
    int maxLen = 0;

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

    return maxLen;
}

int main() {
    string s1 = "xyzabcp";
    string s2 = "pqabcxy";

    cout << LongestCommonSubstring(s1, s2) <<
endl;

    return 0;
}
```

Input:

- s1 = "xyzabcp"
- s2 = "pqabcxy"

Initial Setup:

- m = s1.length() = 7
- n = s2.length() = 7
- dp is a (m+1) x (n+1) matrix initialized to 0. (i.e., dp[8][8])
- maxLen = 0

Table Format for dp:

The rows represent s1 (0 to m) and the columns represent s2 (0 to n).

Step 1: Initialize the dp Matrix

The dp matrix is initialized to all zeros:

```
dp = [
  [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, 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, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0]
]
```

Step 2: Iterative Calculation

We iterate over i (1 to m) and j (1 to n), and compute dp[i][j] based on the characters s1[i-1] and s2[j-1].

Key Rule:

- If s1[i-1] == s2[j-1]: dp[i][j] = dp[i-1][j-1] + 1
- Otherwise: dp[i][j] = 0
- Update maxLen to track the largest value of dp[i][j].

Fill the Table:

i = 1, s1[0] = 'x':

- Compare 'x' with each character of s2:

  dp[1][1] = 0 ('x' != 'p')
  dp[1][2] = 0 ('x' != 'q')
  dp[1][3] = 0 ('x' != 'a')
  dp[1][4] = 0 ('x' != 'b')
  dp[1][5] = 0 ('x' != 'c')
  dp[1][6] = 1 ('x' == 'x')
  dp[1][7] = 0 ('x' != 'y')

  Updated dp:

  dp[1] = [0, 0, 0, 0, 0, 0, 1, 0]

i = 2, s1[1] = 'y':

- Compare 'y' with each character of s2:

  dp[2][1] = 0 ('y' != 'p')
  dp[2][2] = 0 ('y' != 'q')
  dp[2][3] = 0 ('y' != 'a')
  dp[2][4] = 0 ('y' != 'b')
  dp[2][5] = 0 ('y' != 'c')
  dp[2][6] = 0 ('y' != 'x')
  dp[2][7] = 2 ('y' == 'y', dp[1][6] + 1)

  Updated dp:

  dp[2] = [0, 0, 0, 0, 0, 0, 0, 2]

i = 3, s1[2] = 'z':

- Compare 'z' with each character of s2:

  dp[3][1] = 0 ('z' != 'p')
  dp[3][2] = 0 ('z' != 'q')
  dp[3][3] = 0 ('z' != 'a')
  dp[3][4] = 0 ('z' != 'b')
  dp[3][5] = 0 ('z' != 'c')
  dp[3][6] = 0 ('z' != 'x')
  dp[3][7] = 0 ('z' != 'y')

  Updated dp:

dp[3] = [0, 0, 0, 0, 0, 0, 0, 0]

i = 4, s1[3] = 'a':

- Compare 'a' with each character of s2:

```
dp[4][1] = 0   ('a' != 'p')
dp[4][2] = 0   ('a' != 'q')
dp[4][3] = 1   ('a' == 'a', dp[3][2] + 1)
dp[4][4] = 0   ('a' != 'b')
dp[4][5] = 0   ('a' != 'c')
dp[4][6] = 0   ('a' != 'x')
dp[4][7] = 0   ('a' != 'y')
```

Updated dp:

dp[4] = [0, 0, 1, 0, 0, 0, 0, 0]

i = 5, s1[4] = 'b':

- Compare 'b' with each character of s2:

less
Copy code

```
dp[5][1] = 0   ('b' != 'p')
dp[5][2] = 0   ('b' != 'q')
dp[5][3] = 0   ('b' != 'a')
dp[5][4] = 2   ('b' == 'b', dp[4][3] + 1)
dp[5][5] = 0   ('b' != 'c')
dp[5][6] = 0   ('b' != 'x')
dp[5][7] = 0   ('b' != 'y')
```

Updated dp:

dp[5] = [0, 0, 0, 2, 0, 0, 0, 0]

i = 6, s1[5] = 'c':

- Compare 'c' with each character of s2:

```
dp[6][1] = 0   ('c' != 'p')
dp[6][2] = 0   ('c' != 'q')
dp[6][3] = 0   ('c' != 'a')
dp[6][4] = 0   ('c' != 'b')
dp[6][5] = 3   ('c' == 'c', dp[5][4] + 1)
dp[6][6] = 0   ('c' != 'x')
dp[6][7] = 0   ('c' != 'y')
```

Updated dp:

dp[6] = [0, 0, 0, 0, 3, 0, 0, 0]

i = 7, s1[6] = 'p':

- Compare 'p' with each character of s2:

  dp[7][1] = 1   ('p' == 'p', dp[6][0] + 1)
  dp[7][2] = 0   ('p' != 'q')
  dp[7][3] = 0   ('p' != 'a')
  dp[7][4] = 0   ('p' != 'b')
  dp[7][5] = 0   ('p' != 'c')
  dp[7][6] = 0   ('p' != 'x')
  dp[7][7] = 0   ('p' != 'y')

  Updated dp:

  dp[7] = [1, 0, 0, 0, 0, 0, 0, 0]

Final Result:

- maxLen = 3, which corresponds to the substring "abc".

Output:-
3