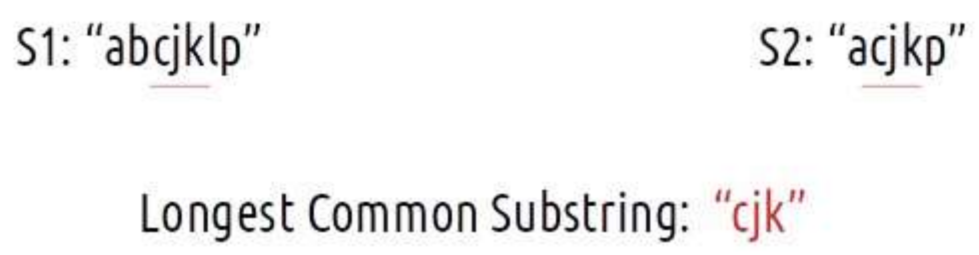
Longest Common Substring | (DP – 27)

A substring of a string is a subsequence in which all the characters are **consecutive**. Given two strings, we need to find the longest common substring.

Example:



We need to print the **length** of the longest common substring.

Problem Link: Longest Common Substring

Solution :

Pre-req: Longest Common Subsequence, Print Longest Common Subsequence

Approach:

We can modify the approach we used in the article Longest Common Subsequence, in order to find the longest common substring. The main distinguishing factor between the two is the consecutiveness of the characters.

While finding the longest common subsequence, we were using two pointers (ind1 and ind2) to map the characters of the two strings. We will again have the same set of conditions for finding the longest common substring, with slight modifications to what we do when the condition becomes true.

We will try to form a solution in the bottom-up (tabulation) approach. We will set two nested loops with loop variables i and j.

Thinking in terms of consecutiveness of characters

We have two conditions:

- if(S1[i-1] != S2[j-1]), the characters don't match, therefore the consecutiveness of characters is broken. So we set the cell value (dp[i][j]) as 0.
- if(S1[i-1] == S2[j-1]), then the characters match and we simply set its value to 1+dp[i-1][j-1]. We have done so because dp[i-1][j-1] gives us the longest common substring till the last cell character (current strings -{matching character}). As the current cell's character is matching we are adding 1 to the consecutive chain.

Note: dp[n][m] will **not** give us the answer; rather the maximum value in the entire dp array will give us the length of the longest common substring. This is because there is no restriction that the longest common substring is present at the end of both the strings.

Code:

- C++ Code
- Java Code

```
#include <bits/stdc++.h>

using namespace std;

int lcs(string &s1, string &s2){

    int n = s1.size();
    int m = s2.size();
    vector<vector<int>> dp(n+1,vector<int>(m+1,0));

    int ans = 0;

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

    return ans;

}

int main() {

  string s1= "abcjklp";
  string s2= "acjkp";

  cout<<"The Length of Longest Common Substring is "<<lcs(s1,s2);
}
```

Output:

The Length of Longest Common Substring is 3

Time Complexity: O(N*M)

Reason: There are two nested loops

Space Complexity: O(N*M)

Reason: We are using an external array of size ‘N*M)’. Stack Space is eliminated.

Space Optimization

If we look closely, we need values from the previous row: dp[**ind-1**][ ]

So we are not required to contain an entire array, we can simply have two rows prev and cur where prev corresponds to dp[ind-1] and cur to dp[ind].

After declaring prev and cur, replace dp[ind-1] to prev and dp[ind] with cur and after the inner loop executes, we will set prev = cur, so that the cur row can serve as prev for the next index.

Code:

- C++ Code
- Java Code

```
#include <bits/stdc++.h>


using namespace std;



int lcs(string &s1, string &s2){
        //      Write your code here.

    int n = s1.size();
    int m = s2.size();

    vector<int> prev(m+1,0), cur(m+1,0);

    int ans = 0;

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

    return ans;

}

int main() {

  string s1= "abcjklp";
  string s2= "acjkp";

  cout<<"The Length of Longest Common Substring is "<<lcs(s1,s2);
}
```

Output:

The Length of Longest Common Substring is 3

Time Complexity: O(N*M)

Reason: There are two nested loops.

Space Complexity: O(M)

Reason: We are using an external array of size ‘M+1’ to store only two rows.