Longest Common Subsequence | DP-4

Difficulty Level : Medium • Last Updated : 06 Jun, 2022

We have discussed Overlapping Subproblems and Optimal Substructure properties in Set 1 and Set 2 respectively. We also discussed one example problem in Set 3. Let us discuss Longest Common Subsequence (LCS) problem as one more example problem that can be solved using Dynamic Programming.

LCS Problem Statement: Given two sequences, find the length of longest subsequence present in both of them. A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous. For example, “abc”, “abg”, “bdf”, “aeg”, ‘”acefg”, .. etc are subsequences of “abcdefg”.

Click here for the Complete Course!

In order to find out the complexity of brute force approach, we need to first know the number of possible different subsequences of a string with length n, i.e., find the number of subsequences with lengths ranging from 1,2,..n-1. Recall from theory of permutation and combination that number of combinations with 1 element are ${}^nC_1$. Number of combinations with 2 elements are ${}^nC_2$ and so forth and so on. We know that ${}^nC_0 + {}^nC_1 + {}^nC_2 + \ldots {}^nC_n = 2^n$. So a string of length n has 2^n-1 different possible subsequences since we do not consider the subsequence with length 0. This implies that the time complexity of the brute force approach will be $O(n * 2^n)$. Note that it takes O(n) time to check if a subsequence is common to both the strings. This time complexity can be improved using dynamic programming.

It is a classic computer science problem, the basis of diff (a file comparison program that outputs the differences between two files), and has applications in bioinformatics.

Examples:
LCS for input Sequences "ABCDGH" and "AEDFHR" is "ADH" of length 3.
LCS for input Sequences "AGGTAB" and "GXTXAYB" is "GTAB" of length 4.

Recommended: Please try your approach on *{IDE}* first, before moving on to the solution.

The naive solution for this problem is to generate all subsequences of both given sequences and find the longest matching subsequence. This solution is exponential in term of time complexity. Let us see how this problem possesses both important properties of a Dynamic Programming (DP) Problem.

1) Optimal Substructure:

Let the input sequences be X[0..m-1] and Y[0..n-1] of lengths m and n respectively. And let L(X[0..m-1], Y[0..n-1]) be the length of LCS of the two sequences X and Y. Following is the recursive definition of L(X[0..m-1], Y[0..n-1]).

If last characters of both sequences match (or X[m-1] == Y[n-1]) then L(X[0..m-1], Y[0..n-1]) = 1 + L(X[0..m-2], Y[0..n-2])

If last characters of both sequences do not match (or X[m-1] != Y[n-1]) then
L(X[0..m-1], Y[0..n-1]) = MAX (L(X[0..m-2], Y[0..n-1]), L(X[0..m-1], Y[0..n-2]))

Examples:

1) Consider the input strings “AGGTAB” and “GXTXAYB”. Last characters match for the strings. So length of LCS can be written as:
L(“AGGTAB”, “GXTXAYB”) = 1 + L(“AGGTA”, “GXTXAY”)

	A	G	G	T	A	B
G	-	-	4	-	-	-
X	-	-	-	-	-	-
T	-	-	-	3	-	-
X	-	-	-	-	-	-
A	-	-	-	-	2	-
Y	-	-	-	-	-	-
B	-	-	-	-	-	1

2) Consider the input strings “ABCDGH” and “AEDFHR. Last characters do not match for the strings. So length of LCS can be written as: L(“ABCDGH”, “AEDFHR”) = MAX (L(“ABCDG”, “AEDFH**R**”), L(“ABCDG**H**”, “AEDFH”))

So the LCS problem has optimal substructure property as the main problem can be solved using solutions to subproblems.

2) Overlapping Subproblems:

Following is simple recursive implementation of the LCS problem. The implementation simply follows the recursive structure mentioned above.

C++

```
/* A Naive recursive implementation of LCS problem */
#include <bits/stdc++.h>
using namespace std;


/* Returns length of LCS for X[0..m-1], Y[0..n-1] */
int lcs( char *X, char *Y, int m, int n )
{
    if (m == 0 || n == 0)
```

```
        return 0;
    if (X[m-1] == Y[n-1])
        return 1 + lcs(X, Y, m-1, n-1);
    else
        return max(lcs(X, Y, m, n-1), lcs(X, Y, m-1, n));
}



/* Driver code */
int main()
{
    char X[] = "AGGTAB";
    char Y[] = "GXTXAYB";

    int m = strlen(X);
    int n = strlen(Y);

    cout<<"Length of LCS is "<< lcs( X, Y, m, n ) ;

    return 0;
}

// This code is contributed by rathbhupendra
```

C

```
/* A Naive recursive implementation of LCS problem */
#include<bits/stdc++.h>

int max(int a, int b);

/* Returns length of LCS for X[0..m-1], Y[0..n-1] */
int lcs( char *X, char *Y, int m, int n )
{
if (m == 0 || n == 0)
    return 0;
if (X[m-1] == Y[n-1])
    return 1 + lcs(X, Y, m-1, n-1);
else
    return max(lcs(X, Y, m, n-1), lcs(X, Y, m-1, n));
}

/* Utility function to get max of 2 integers */
```

```
int max(int a, int b)
{
    return (a > b)? a : b;
}

/* Driver program to test above function */
int main()
{
char X[] = "AGGTAB";
char Y[] = "GXTXAYB";

int m = strlen(X);
int n = strlen(Y);

printf("Length of LCS is %d", lcs( X, Y, m, n ) );

return 0;
}
```

Java

```
/* A Naive recursive implementation of LCS problem in java*/
public class LongestCommonSubsequence
{

/* Returns length of LCS for X[0..m-1], Y[0..n-1] */
int lcs( char[] X, char[] Y, int m, int n )
{
    if (m == 0 || n == 0)
    return 0;
    if (X[m-1] == Y[n-1])
    return 1 + lcs(X, Y, m-1, n-1);
    else
    return max(lcs(X, Y, m, n-1), lcs(X, Y, m-1, n));
}

/* Utility function to get max of 2 integers */
int max(int a, int b)
{
    return (a > b)? a : b;
}

public static void main(String[] args)
```

```
{
    LongestCommonSubsequence lcs = new LongestCommonSubsequence();
    String s1 = "AGGTAB";
    String s2 = "GXTXAYB";

    char[] X=s1.toCharArray();
    char[] Y=s2.toCharArray();
    int m = X.length;
    int n = Y.length;

    System.out.println("Length of LCS is" + " " +
                                lcs.lcs( X, Y, m, n ) );
}

}

// This Code is Contributed by Saket Kumar
```

Python3

```
# A Naive recursive Python implementation of LCS problem

def lcs(X, Y, m, n):

    if m == 0 or n == 0:
        return 0
    elif X[m-1] == Y[n-1]:
        return 1 + lcs(X, Y, m-1, n-1);
    else:
        return max(lcs(X, Y, m, n-1), lcs(X, Y, m-1, n));


# Driver program to test the above function
X = "AGGTAB"
Y = "GXTXAYB"
print ("Length of LCS is ", lcs(X , Y, len(X), len(Y)) )
```

C#

```
/* C# Naive recursive implementation of LCS problem */
using System;
```

```
class GFG
{


    /* Returns length of LCS for X[0..m-1], Y[0..n-1] */
    static int lcs( char[] X, char[] Y, int m, int n )
    {
        if (m == 0 || n == 0)
        return 0;
        if (X[m - 1] == Y[n - 1])
        return 1 + lcs(X, Y, m - 1, n - 1);
        else
        return max(lcs(X, Y, m, n - 1), lcs(X, Y, m - 1, n));
    }

    /* Utility function to get max of 2 integers */
    static int max(int a, int b)
    {
        return (a > b)? a : b;
    }

    public static void Main()
    {
        String s1 = "AGGTAB";
        String s2 = "GXTXAYB";

        char[] X=s1.ToCharArray();
        char[] Y=s2.ToCharArray();
        int m = X.Length;
        int n = Y.Length;

        Console.Write("Length of LCS is" + " "
                    +lcs( X, Y, m, n ) );
    }
}
// This code is Contributed by Sam007
```

PHP

```
<?php
// A Naive recursive PHP
// implementation of LCS problem
function lcs($X, $Y, $m, $n)
{
```

```
    if($m == 0 || $n == 0)
    return 0;
    else if ($X[$m - 1] == $Y[$n - 1])
        return 1 + lcs($X, $Y,
                    $m - 1, $n - 1);
    else
        return max(lcs($X, $Y, $m, $n - 1),
                lcs($X, $Y, $m - 1, $n));
}

// Driver Code
$X = "AGGTAB";
$Y = "GXTXAYB";
echo "Length of LCS is ";
echo lcs($X , $Y, strlen($X),
                strlen($Y));

// This code is contributed
// by Shivi_Aggarwal
?>
```

Javascript

```
<script>
/* A Naive recursive implementation of LCS problem in java*/

/* Returns length of LCS for X[0..m-1], Y[0..n-1] */
function lcs(  X,  Y , m , n )
{
    if (m == 0 || n == 0)
    return 0;
    if (X[m-1] == Y[n-1])
    return 1 + lcs(X, Y, m-1, n-1);
    else
    return max(lcs(X, Y, m, n-1), lcs(X, Y, m-1, n));
}

/* Utility function to get max of 2 integers */
function max(a , b)
{
    return (a > b)? a : b;
}
```

```
    var s1 = "AGGTAB";
    var s2 = "GXTXAYB";

    var X=s1;
    var Y=s2;
    var m = X.length;
    var n = Y.length;

    document.write("Length of LCS is" + " " +
                                lcs( X, Y, m, n ) );

// This code contributed by umadevi9616
</script>
```

Output

```
Length of LCS is 4
```

Time complexity of the above naive recursive approach is O(2^n) in worst case and worst case happens when all characters of X and Y mismatch i.e., length of LCS is 0.

Considering the above implementation, following is a partial recursion tree for input strings "AXYT" and "AYZX"

```
                          lcs("AXYT", "AYZX")
                         /
          lcs("AXY", "AYZX")             lcs("AXYT", "AYZ")
          /                               /
 lcs("AX", "AYZX") lcs("AXY", "AYZ")   lcs("AXY", "AYZ") lcs("AXYT"
```

In the above partial recursion tree, lcs("AXY", "AYZ") is being solved twice. If we draw the complete recursion tree, then we can see that there are many subproblems which are solved again and again. So this problem has Overlapping Substructure property and recomputation of same subproblems can be avoided by either using Memoization or

Tabulation.

Following is a Memoization implementation for the LCS problem.

C++

```cpp
/* A Top-Down DP implementation of LCS problem */
#include <bits/stdc++.h>
using namespace std;

/* Returns length of LCS for X[0..m-1], Y[0..n-1] */
int lcs(char* X, char* Y, int m, int n,
        vector<vector<int> >& dp)
{
    if (m == 0 || n == 0)
        return 0;
    if (X[m - 1] == Y[n - 1])
        return dp[m][n] = 1 + lcs(X, Y, m - 1, n - 1, dp);

    if (dp[m][n] != -1) {
        return dp[m][n];
    }
    return dp[m][n] = max(lcs(X, Y, m, n - 1, dp),
                          lcs(X, Y, m - 1, n, dp));
}

/* Driver code */
int main()
{
    char X[] = "AGGTAB";
    char Y[] = "GXTXAYB";

    int m = strlen(X);
    int n = strlen(Y);
    vector<vector<int> > dp(m + 1, vector<int>(n + 1, -1));
    cout << "Length of LCS is " << lcs(X, Y, m, n, dp);

    return 0;
}
```

Java

```java
/*package whatever //do not write package name here */
import java.io.*;

class GFG
{

  // A Top-Down DP implementation of LCS problem

  // Returns length of LCS for X[0..m-1], Y[0..n-1]
  static int lcs(String X,String Y,int m,int n,int[][] dp){

    if (m == 0 || n == 0)
      return 0;

    if (dp[m][n] != -1)
      return dp[m][n];

    if(X.charAt(m - 1) == Y.charAt(n - 1)){
      dp[m][n] = 1 + lcs(X, Y, m - 1, n - 1, dp);
      return dp[m][n];
    }

    dp[m][n] = Math.max(lcs(X, Y, m, n - 1, dp),lcs(X, Y, m - 1, n, dp));
    return dp[m][n];
  }

  // Drivers code
  public static void main(String args[]){

    String X = "AGGTAB";
    String Y = "GXTXAYB";

    int m = X.length();
    int n = Y.length();
    int[][] dp = new int[m + 1][n + 1];
    for(int i=0;i<m + 1;i++){
      for(int j=0;j<n + 1;j++){
        dp[i][j] = -1;
      }
    }

    System.out.println("Length of LCS is "+lcs(X, Y, m, n, dp));

  }
}
```

```
// This code is contributed by shinjanpatra
```

Python3

```
# A Top-Down DP implementation of LCS problem

# Returns length of LCS for X[0..m-1], Y[0..n-1]
def lcs(X, Y, m, n, dp):

    if (m == 0 or n == 0):
        return 0

    if (dp[m][n] != -1):
        return dp[m][n]

    if X[m - 1] == Y[n - 1]:
        dp[m][n] = 1 + lcs(X, Y, m - 1, n - 1, dp)
        return dp[m][n]

    dp[m][n] = max(lcs(X, Y, m, n - 1, dp),lcs(X, Y, m - 1, n, dp))
    return dp[m][n]

# Driver code

X = "AGGTAB"
Y = "GXTXAYB"

m = len(X)
n = len(Y)
dp = [[-1 for i in range(n + 1)]for j in range(m + 1)]

print(f"Length of LCS is {lcs(X, Y, m, n, dp)}")

# This code is contributed by shinjanpatra
```

C#

```
/* C# Naive recursive implementation of LCS problem */
using System;

class GFG {
```

```
    /* Returns length of LCS for X[0..m-1], Y[0..n-1] */
    static int lcs(char[] X, char[] Y, int m, int n, int [,]L) {
        if (m == 0 || n == 0)
            return 0;

        if (L[m, n] != -1)
            return L[m, n];

        if (X[m - 1] == Y[n - 1]) {
            L[m, n] = 1 + lcs(X, Y, m - 1, n - 1, L);
            return L[m, n];
        }

        L[m, n] = max(lcs(X, Y, m, n - 1, L), lcs(X, Y, m - 1, n, L));
        return L[m, n];
    }

    /* Utility function to get max of 2 integers */
    static int max(int a, int b) {
        return (a > b) ? a : b;
    }

    public static void Main() {
        String s1 = "AGGTAB";
        String s2 = "GXTXAYB";

        char[] X = s1.ToCharArray();
        char[] Y = s2.ToCharArray();
        int m = X.Length;
        int n = Y.Length;
        int[,]L = new int[m + 1, n + 1];
        for (int i = 0; i <= m; i++)
        {
            for (int j = 0; j <= n; j++)
            {
                L[i, j] = -1;
            }
        }
        Console.Write("Length of LCS is" + " "
            + lcs(X, Y, m, n, L));
    }
}
```

```
// This code is contrributed by akshitsaxenaa09
```

Javascript

```
<script>

/* A Top-Down DP implementation of LCS problem */

/* Returns length of LCS for X[0..m-1], Y[0..n-1] */
function lcs(X, Y, m, n, dp)
{
    if (m == 0 || n == 0)
        return 0;
    if (X[m - 1] == Y[n - 1])
        return dp[m][n] = 1 + lcs(X, Y, m - 1, n - 1, dp);

    if (dp[m][n] != -1) {
        return dp[m][n];
    }
    return dp[m][n] = Math.max(lcs(X, Y, m, n - 1, dp),
                           lcs(X, Y, m - 1, n, dp));
}

/* Driver code */

let X = "AGGTAB";
let Y = "GXTXAYB";

let m = X.length;
let n = Y.length;
let dp = new Array(m + 1);
for(let i = 0; i < m + 1; i++)
{
    dp[i] = new Array(n + 1).fill(-1);
}
document.write("Length of LCS is " + lcs(X, Y, m, n, dp));

// This code is contributed by shinjanpatra

</script>
```

Output

```
Length of LCS is 4
```

Time Complexity : O(mn) ignoring recursion stack space

Following is a tabulated implementation for the LCS problem.

Python3

```python
def lcs(s1 , s2):
    m, n = len(s1), len(s2)
    prev, cur = [0]*(n+1), [0]*(n+1)
    for i in range(1, m+1):
        for j in range(1, n+1):
            if s1[i-1] == s2[j-1]:
                cur[j] = 1 + prev[j-1]
            else:
                if cur[j-1] > prev[j]:
                    cur[j] = cur[j-1]
                else:
                    cur[j] = prev[j]
        cur, prev = prev, cur
    return prev[n]

#end of function lcs


# Driver program to test the above function
s1 = "AGGTAB"
s2 = "GXTXAYB"
print("Length of LCS is ", lcs(s1, s2))
# BY PRASHANT SHEKHAR MISHRA
```

C++

```cpp
/* Dynamic Programming C implementation of LCS problem */
#include <bits/stdc++.h>
using namespace std;

/* Returns length of LCS for X[0..m-1], Y[0..n-1] */
int lcs(char *X, char *Y, int m, int n)
{
```

```
    // intitalizing a matrix of size (m+1)*(n+1)
    int L[m + 1][n + 1];

    /* Following steps build L[m+1][n+1] in bottom up fashion. Note
        that L[i][j] contains length of LCS of X[0..i-1] and Y[0..j-1] */
    for (int i = 0; i <= m; i++)
    {
        for (int j = 0; j <= n; j++)
        {
            if (i == 0 || j == 0)
                L[i][j] = 0;

            else if (X[i - 1] == Y[j - 1])
                L[i][j] = L[i - 1][j - 1] + 1;

            else
                L[i][j] = max(L[i - 1][j], L[i][j - 1]);
        }
    }

    // L[m][n] contains length of LCS for X[0..n-1] and Y[0..m-1]
    return L[m][n];
}

// Driver program to test above function
int main()
{

    char X[] = "AGGTAB";
    char Y[] = "GXTXAYB";

    int m = strlen(X);
    int n = strlen(Y);

    cout << "Length of LCS is: " << lcs(X, Y, m, n);

    return 0;
}
// code submitted by Aditya Yadav (adityayadav012552)
```

Java

```
/* Dynamic Programming Java implementation of LCS problem */
public class LongestCommonSubsequence
```

```
{

/* Returns length of LCS for X[0..m-1], Y[0..n-1] */
int lcs( char[] X, char[] Y, int m, int n )
{
    int L[][] = new int[m+1][n+1];

    /* Following steps build L[m+1][n+1] in bottom up fashion. Note
        that L[i][j] contains length of LCS of X[0..i-1] and Y[0..j-1] */
    for (int i=0; i<=m; i++)
    {
    for (int j=0; j<=n; j++)
    {
        if (i == 0 || j == 0)
            L[i][j] = 0;
        else if (X[i-1] == Y[j-1])
            L[i][j] = L[i-1][j-1] + 1;
        else
            L[i][j] = max(L[i-1][j], L[i][j-1]);
    }
    }
return L[m][n];
}

/* Utility function to get max of 2 integers */
int max(int a, int b)
{
    return (a > b)? a : b;
}

public static void main(String[] args)
{
    LongestCommonSubsequence lcs = new LongestCommonSubsequence();
    String s1 = "AGGTAB";
    String s2 = "GXTXAYB";

    char[] X=s1.toCharArray();
    char[] Y=s2.toCharArray();
    int m = X.length;
    int n = Y.length;

    System.out.println("Length of LCS is" + " " +
                                lcs.lcs( X, Y, m, n ) );
}
```

```
}

// This Code is Contributed by Saket Kumar
```

Python3

```
# Dynamic Programming implementation of LCS problem

def lcs(X , Y):
    # find the length of the strings
    m = len(X)
    n = len(Y)

    # declaring the array for storing the dp values
    L = [[None]*(n+1) for i in range(m+1)]

    """Following steps build L[m+1][n+1] in bottom up fashion
    Note: L[i][j] contains length of LCS of X[0..i-1]
    and Y[0..j-1]"""
    for i in range(m+1):
        for j in range(n+1):
            if i == 0 or j == 0 :
                L[i][j] = 0
            elif X[i-1] == Y[j-1]:
                L[i][j] = L[i-1][j-1]+1
            else:
                L[i][j] = max(L[i-1][j] , L[i][j-1])

    # L[m][n] contains the length of LCS of X[0..n-1] & Y[0..m-1]
    return L[m][n]
#end of function lcs


# Driver program to test the above function
X = "AGGTAB"
Y = "GXTXAYB"
print ("Length of LCS is ", lcs(X, Y) )

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

C#

```
// Dynamic Programming C# implementation
// of LCS problem
using System;

class GFG
{

    /* Returns length of LCS for X[0..m-1], Y[0..n-1] */
    static int lcs( char[] X, char[] Y, int m, int n )
    {
        int [,]L = new int[m+1,n+1];

        /* Following steps build L[m+1][n+1]
        in bottom up fashion. Note
        that L[i][j] contains length of
        LCS of X[0..i-1] and Y[0..j-1] */
        for (int i = 0; i <= m; i++)
        {
            for (int j = 0; j <= n; j++)
            {
                if (i == 0 || j == 0)
                    L[i, j] = 0;
                else if (X[i - 1] == Y[j - 1])
                    L[i, j] = L[i - 1, j - 1] + 1;
                else
                    L[i, j] = max(L[i - 1, j], L[i, j - 1]);
            }
        }
        return L[m, n];
    }

    /* Utility function to get max of 2 integers */
    static int max(int a, int b)
    {
        return (a > b)? a : b;
    }

    // Driver code
    public static void Main()
    {

        String s1 = "AGGTAB";
        String s2 = "GXTXAYB";

        char[] X=s1.ToCharArray();
```

```
        char[] Y=s2.ToCharArray();
        int m = X.Length;
        int n = Y.Length;

        Console.Write("Length of LCS is" + " " +lcs( X, Y, m, n ) );
    }
}
// This Code is Contributed by Sam007
```

PHP

```
<?php
// Dynamic Programming C#
// implementation of LCS problem
function lcs($X , $Y)
{
// find the length of the strings
$m = strlen($X);
$n = strlen($Y) ;

// declaring the array for
// storing the dp values

/*Following steps build L[m+1][n+1]
in bottom up fashion .
Note: L[i][j] contains length of
LCS of X[0..i-1] and Y[0..j-1] */
for ($i = 0; $i <= $m; $i++)
{
for ($j = 0; $j <= $n; $j++)
{
    if ($i == 0 || $j == 0)
    $L[$i][$j] = 0;

    else if ($X[$i - 1] == $Y[$j - 1])
    $L[$i][$j] = $L[$i - 1][$j - 1] + 1;

    else
    $L[$i][$j] = max($L[$i - 1][$j],
                     $L[$i][$j - 1]);
}
}

// L[m][n] contains the length of
```

```php
// LCS of X[0..n-1] & Y[0..m-1]

return $L[$m][$n];
}

// Driver Code
$X = "AGGTAB";
$Y = "GXTXAYB";
echo "Length of LCS is ";
echo lcs($X, $Y);

// This code is contributed
// by Shivi_Aggarwal
?>
```

Javascript

```javascript
<script>

// Dynamic Programming Java implementation of LCS problem

// Utility function to get max of 2 integers
function max(a, b)
{
    if (a > b)
        return a;
    else
        return b;
}

// Returns length of LCS for X[0..m-1], Y[0..n-1]
function lcs(X, Y, m, n)
{
    var L = new Array(m + 1);
    for(var i = 0; i < L.length; i++)
    {
        L[i] = new Array(n + 1);
    }
    var i, j;

    /* Following steps build L[m+1][n+1] in
    bottom up fashion. Note that L[i][j]
    contains length of LCS of X[0..i-1]
```

```
    and Y[0..j-1] */
    for(i = 0; i <= m; i++)
    {
        for(j = 0; j <= n; j++)
        {
            if (i == 0 || j == 0)
                L[i][j] = 0;
            else if (X[i - 1] == Y[j - 1])
                L[i][j] = L[i - 1][j - 1] + 1;
            else
                L[i][j] = max(L[i - 1][j], L[i][j - 1]);
        }
    }

    /* L[m][n] contains length of LCS
    for X[0..n-1] and Y[0..m-1] */
    return L[m][n];
}

// Driver code
var x = "AGGTAB";
var y = "GXTXAYB";

var m = x.length;
var n = y.length;

document.write("Length of LCS is " + lcs(x, y, m, n));

// This code is contributed by akshitsaxenaa09

</script>
```

Output

```
 Length of LCS is  4
```

Time Complexity of the above implementation is O(mn) which is much better than the worst-case time complexity of Naive Recursive implementation.

The above algorithm/code returns only length of LCS. Please see the following post for printing the LCS.

Printing Longest Common Subsequence

You can also check the space optimized version of LCS at

Space Optimized Solution of LCS

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Recent Articles based on LCS!

References:

http://www.youtube.com/watch?v=V5hZoJ6uK-s

http://www.algorithmist.com/index.php/Longest_Common_Subsequence

http://www.ics.uci.edu/~eppstein/161/960229.html

http://en.wikipedia.org/wiki/Longest_common_subsequence_problem

Previous

Painting Fence Algorithm

Next

Count all subsequences having product less than K

RECOMMENDED ARTICLES

01 **Longest Increasing Subsequence using Longest Common Subsequence Algorithm**
14, Oct 19

02 **Maximum length subsequence such that adjacent elements in the subsequence have a common factor**
11, Feb 19

03 **C++ Program for Longest Common Subsequence**
14, Jun 11

04 **Java Program for Longest Common Subsequence**
14, Jun 11

05 **Longest Common Subsequence with at most k changes allowed**
05, Oct 17

06 **Minimum cost to make Longest Common Subsequence of length k**
05, Dec 17

07 **Longest Common Anagram Subsequence**
27, Dec 17

08 **Longest Common Subsequence | DP using Memoization**
03, Aug 18

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Medium

Easy | Normal | Medium | Hard | Expert

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org

Improved By : sirrobot, Shivi_Aggarwal, VinayShrma9, mayankagarwal44442, rathbhupendra, jethawahimanshu, Codextor, akshitsaxenaa09, prashantsmishra024, umadevi9616, akhilkashyap, amartyaghoshgfg, prasanna1995, shinjanpatra, adityayadav012552

Article Tags : Amazon, FactSet, Hike, LCS, subsequence, Dynamic Programming, Strings

Practice Tags : Amazon, FactSet, Hike, Strings, Dynamic Programming, LCS

NodeJS