Longest Common Subsequence | DP-4

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 30 Sep, 2019

We have discussed Overlapping Subproblems and Optimal Substructure properties in [Set 1](https://www.geeksforgeeks.org/dynamic-programming-set-1/) and [Set 2](https://www.geeksforgeeks.org/dynamic-programming-set-2-optimal-substructure-property/) respectively. We also discussed one example problem in [Set 3](https://www.geeksforgeeks.org/longest-increasing-subsequence/). 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”.

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 nC1. Number of combinations with 2 elements are nC2 and so forth and so on. We know that nC0 + nC1 + nC2 + … nCn = 2n. So a string of length n has 2n-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 \* 2n). 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](http://en.wikipedia.org/wiki/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.

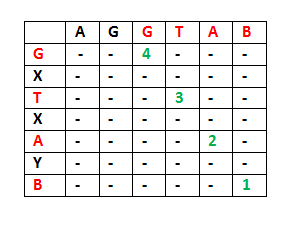
[Recommended: Please solve it on “***PRACTICE***” first, before moving on to the solution.](https://practice.geeksforgeeks.org/problems/longest-common-subsequence/0)

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”)  
  
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;    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 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 |

**Python**

|  |
| --- |
| # 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  ?> |

**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", "AY")

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 tabulated implementation for the LCS problem.

**C++**

|  |
| --- |
| /\* Dynamic Programming C++ implementation of LCS problem \*/  #include<bits/stdc++.h>  using namespace std;    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 )  {      int L[m + 1][n + 1];      int 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];  }    /\* Utility function to get max of 2 integers \*/  int max(int a, int b)  {      return (a > b)? a : b;  }    // 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**

|  |
| --- |
| /\* Dynamic Programming C 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 )  {     int L[m+1][n+1];     int 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];  }    /\* 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**

|  |
| --- |
| /\* 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 |

**Python**

|  |
| --- |
| # 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 xrange(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  // 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  ?> |

**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](https://www.geeksforgeeks.org/printing-longest-common-subsequence/)

You can also check the space optimized version of LCS at  
[Space Optimized Solution of LCS](https://www.geeksforgeeks.org/space-optimized-solution-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!](https://www.geeksforgeeks.org/tag/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>

Attention reader! Don’t stop learning now. Get hold of all the important DSA concepts with the [**DSA Self Paced Course**](https://practice.geeksforgeeks.org/courses/dsa-self-paced?utm_source=geeksforgeeks&utm_medium=article&utm_campaign=gfg_article_dsa_content_bottom) at a student-friendly price and become industry ready.  To complete your preparation from learning a language to DS Algo and many more,  please refer [**Complete Interview Preparation Course**](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=GeeksforGeeks&utm_medium=Text&utm_campaign=GFG_Article_Bottom_Text_CIP)**.**

In case you wish to attend live classes with industry experts, please refer [**DSA Live Classes**](https://practice.geeksforgeeks.org/courses/geeks-classes-live?utm_source=geeksforgeeks&utm_medium=article&utm_campaign=gfg_article_dsa_content_bottom)

Longest Increasing Subsequence using Longest Common Subsequence Algorithm

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 10 Jun, 2021

Given an array **arr[]** of **N** integers, the task is to find and print the Longest Increasing Subsequence.  
**Examples:** 

***Input:****arr[] = {12, 34, 1, 5, 40, 80}****Output:****4   
{12, 34, 40, 80} and {1, 5, 40, 80} are the   
longest increasing subsequences.****Input:****arr[] = {10, 22, 9, 33, 21, 50, 41, 60, 80}****Output:****6*

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

Prerequisite: [LCS](https://www.geeksforgeeks.org/longest-common-subsequence-dp-4/), [LIS](https://www.geeksforgeeks.org/construction-of-longest-increasing-subsequence-using-dynamic-programming/)  
**Approach:** The longest increasing subsequence of any sequence is the subsequence of the sorted sequence of itself. It can be solved using a [Dynamic Programming](http://www.geeksforgeeks.org/dynamic-programming/) approach. The approach is the same as the classical [LCS](http://www.geeksforgeeks.org/longest-common-subsequence/) problem but instead of the second sequence, given sequence is taken again in its sorted form.  
**Note:**Array should have distinct elements otherwise it might give wrong result. For example in {1, 1, 1} we know the longest increasing subsequence(a1 < a2 < … ak) is of length 1, but if we try out this example in LIS using LCS method we would get 3 (because it finds the longest common subsequence).   
Below is the implementation of the above approach: 

* C++
* Java
* Python3
* C#
* Javascript

|  |
| --- |
| // C++ implementation of the approach  #include <bits/stdc++.h>  using namespace std;    // Function to return the size of the  // longest increasing subsequence  int LISusingLCS(vector<int>& seq)  {      int n = seq.size();        // Create an 2D array of integer      // for tabulation      vector<vector<int> > L(n + 1, vector<int>(n + 1));        // Take the second sequence as the sorted      // sequence of the given sequence      vector<int> sortedseq(seq);        sort(sortedseq.begin(), sortedseq.end());        // Classical Dynamic Programming algorithm      // for Longest Common Subsequence      for (int i = 0; i <= n; i++) {          for (int j = 0; j <= n; j++) {              if (i == 0 || j == 0)                  L[i][j] = 0;                else if (seq[i - 1] == sortedseq[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 the ans      return L[n][n];  }    // Driver code  int main()  {        vector<int> sequence{ 12, 34, 1, 5, 40, 80 };        cout << LISusingLCS(sequence) << endl;        return 0;  } |

**Output:**

4

**Time Complexity:** O(n2) where n is the length of the sequence.

Maximum length subsequence such that adjacent elements in the subsequence have a common factor

* Difficulty Level : [Hard](https://www.geeksforgeeks.org/hard/)
* Last Updated : 02 Jun, 2021

Given an array **arr[]**, the task is to find the maximum length of a subsequence such that the adjacent elements in the subsequence have a common factor.

**Examples:**

***Input:****arr[] = { 13, 2, 8, 6, 3, 1, 9 }****Output:****5  
Max length subsequence with satisfied conditions: { 2, 8, 6, 3, 9 }*

***Input:****arr[] = { 12, 2, 8, 6, 3, 1, 9 }****Output:****6  
Max length subsequence with satisfied conditions: {12, 2, 8, 6, 3, 9 }*

***Input:****arr[] = { 1, 2, 2, 3, 3, 1 }****Output:****2*

**Approach:** A **naive** approach is to consider all subsequences and check every subsequence whether it satisfies the condition.   
An **efficient** solution is to use **Dynamic programming**. Let dp[i] denote the maximum length of subsequence including arr[i]. Then, the following relation holds for every prime p such that p is a prime factor of arr[i]:

dp[i] = max(dp[i], 1 + dp[pos[p]])

where pos[p] gives the index of p in the array

where it last occurred.

**Explanation:** Traverse the array. For an element arr[i], there are 2 possibilities.

1. If the prime factors of arr[i] have shown their first appearance in the array, then dp[i] = 1
2. If the prime factors of arr[i] have already occurred, then this element can be added in the subsequence since there’s a common factor. Hence dp[i] = max(dp[i], 1 + dp[pos[p]]) where p is the common prime factor and pos[p] is the latest index of p in the array.

Below is the implementation of the above approach:

* C++
* Python3
* Java
* C#
* Javascript

|  |
| --- |
| // C++ implementation of the above approach  #include <bits/stdc++.h>  #define N 100005  #define MAX 10000002    using namespace std;    int lpd[MAX];    // Function to compute least  // prime divisor of i  void preCompute()  {      memset(lpd, 0, sizeof(lpd));      lpd[0] = lpd[1] = 1;      for (int i = 2; i \* i < MAX; i++)      {          for (int j = i \* 2; j < MAX; j += i)          {              if (lpd[j] == 0)              {                  lpd[j] = i;              }          }      }      for (int i = 2; i < MAX; i++)      {          if (lpd[i] == 0)          {              lpd[i] = i;          }      }  }    // Function that returns the maximum  // length subsequence such that  // adjacent elements have a common factor.  int maxLengthSubsequence(int arr[], int n)  {      int dp[N];      unordered\_map<int, int> pos;        // Initialize dp array with 1.      for (int i = 0; i <= n; i++)          dp[i] = 1;        for (int i = 0; i <= n; i++)      {          while (arr[i] > 1)          {              int p = lpd[arr[i]];              if (pos[p])              {                  // p has appeared at least once.                  dp[i] = max(dp[i], 1 + dp[pos[p]]);              }                // Update latest occurrence of prime p.              pos[p] = i;              while (arr[i] % p == 0)                  arr[i] /= p;          }      }        // Take maximum value as the answer.      int ans = 1;      for (int i = 0; i <= n; i++)      {          ans = max(ans, dp[i]);      }        return ans;  }    // Driver code  int main()  {      int arr[] = { 13, 2, 8, 6, 3, 1, 9 };      int n = sizeof(arr) / sizeof(arr[0]);        preCompute();        cout << maxLengthSubsequence(arr, n);      return 0;  } |

**Output**

5

**Time Complexity:** O(N\* log(N))  
**Auxiliary Space:** O(N)

Printing Longest Common Subsequence

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 10 Jun, 2020

Given two sequences, print the longest subsequence present in both of them.

**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.

We have discussed [Longest Common Subsequence (LCS)](https://www.geeksforgeeks.org/dynamic-programming-set-4-longest-common-subsequence/) problem in a [previous post](https://www.geeksforgeeks.org/dynamic-programming-set-4-longest-common-subsequence/). The function discussed there was mainly to find the length of LCS. To find length of LCS, a 2D table L[][] was constructed. In this post, the function to construct and print LCS is discussed.

Following is detailed algorithm to print the LCS. It uses the same 2D table L[][].

**1)** Construct L[m+1][n+1] using the steps discussed in [previous post](https://www.geeksforgeeks.org/dynamic-programming-set-4-longest-common-subsequence).

**2)** The value L[m][n] contains length of LCS. Create a character array lcs[] of length equal to the length of lcs plus 1 (one extra to store \0).

**2)** Traverse the 2D array starting from L[m][n]. Do following for every cell L[i][j]  
…..**a)** If characters (in X and Y) corresponding to L[i][j] are same (Or X[i-1] == Y[j-1]), then include this character as part of LCS.  
…..**b)** Else compare values of L[i-1][j] and L[i][j-1] and go in direction of greater value.

The following table (taken from [Wiki](http://en.wikipedia.org/wiki/Longest_common_subsequence_problem)) shows steps (highlighted) followed by the above algorithm.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Ø | M | Z | J | A | W | X | U |
| 0 | Ø | **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | X | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 2 | M | 0 | **1** | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | J | 0 | 1 | 1 | **2** | 2 | 2 | 2 | 2 |
| 4 | Y | 0 | 1 | 1 | 2 | 2 | 2 | 2 | 2 |
| 5 | A | 0 | 1 | 1 | 2 | **3** | 3 | 3 | 3 |
| 6 | U | 0 | 1 | 1 | 2 | 3 | 3 | 3 | **4** |
| 7 | Z | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 4 |

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

Following is the implementation of above approach.

* C/C++
* Java
* Python
* C#
* PHP

|  |
| --- |
| /\* Dynamic Programming implementation of LCS problem \*/  #include<iostream>  #include<cstring>  #include<cstdlib>  using namespace std;    /\* Returns length of LCS for X[0..m-1], Y[0..n-1] \*/  void lcs( char \*X, char \*Y, int m, int n )  {     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]);       }     }       // Following code is used to print LCS     int index = L[m][n];       // Create a character array to store the lcs string     char lcs[index+1];     lcs[index] = '\0'; // Set the terminating character       // Start from the right-most-bottom-most corner and     // one by one store characters in lcs[]     int i = m, j = n;     while (i > 0 && j > 0)     {        // If current character in X[] and Y are same, then        // current character is part of LCS        if (X[i-1] == Y[j-1])        {            lcs[index-1] = X[i-1]; // Put current character in result            i--; j--; index--;     // reduce values of i, j and index        }          // If not same, then find the larger of two and        // go in the direction of larger value        else if (L[i-1][j] > L[i][j-1])           i--;        else           j--;     }       // Print the lcs     cout << "LCS of " << X << " and " << Y << " is " << lcs;  }    /\* Driver program to test above function \*/  int main()  {    char X[] = "AGGTAB";    char Y[] = "GXTXAYB";    int m = strlen(X);    int n = strlen(Y);    lcs(X, Y, m, n);    return 0;  } |

**Output:**

LCS of AGGTAB and GXTXAYB is GTAB

**References:**  
<http://en.wikipedia.org/wiki/Longest_common_subsequence_problem>

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

Attention reader! Don’t stop learning now. Get hold of all the important DSA concepts with the [**DSA Self Paced Course**](https://practice.geeksforgeeks.org/courses/dsa-self-paced?utm_source=geeksforgeeks&utm_medium=article&utm_campaign=gfg_article_dsa_content_bottom) at a student-friendly price and become industry ready.  To complete your preparation from learning a language to DS Algo and many more,  please refer [**Complete Interview Preparation Course**](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=GeeksforGeeks&utm_medium=Text&utm_campaign=GFG_Article_Bottom_Text_CIP)**.**

In case you wish to attend live classes with industry experts, please refer [**DSA Live Classes**](https://practice.geeksforgeeks.org/courses/geeks-classes-live?utm_source=geeksforgeeks&utm_medium=article&utm_campaign=gfg_article_dsa_content_bottom)

Longest Common Increasing Subsequence (LCS + LIS)

* Difficulty Level : [Hard](https://www.geeksforgeeks.org/hard/)
* Last Updated : 31 Mar, 2021

Prerequisites : [LCS](https://www.geeksforgeeks.org/dynamic-programming-set-4-longest-common-subsequence/), [LIS](https://www.geeksforgeeks.org/dynamic-programming-set-3-longest-increasing-subsequence/)  
Given two arrays, find length of the longest common increasing subsequence [LCIS] and print one of such sequences (multiple sequences may exist)  
Suppose we consider two arrays –   
arr1[] = {3, 4, 9, 1} and   
arr2[] = {5, 3, 8, 9, 10, 2, 1}  
Our answer would be {3, 9} as this is the longest common subsequence which is increasing also.

[Recommended: Please solve it on “***PRACTICE***” first, before moving on to the solution.](https://practice.geeksforgeeks.org/problems/longest-common-increasing-subsequence/0)

The idea is to use dynamic programming here as well. We store the longest common increasing sub-sequence ending at each index of arr2[]. We create an auxiliary array table[] such that table[j] stores length of LCIS ending with arr2[j]. At the end, we return maximum value from this table. For filling values in this table, we traverse all elements of arr1[] and for every element arr1[i], we traverse all elements of arr2[]. If we find a match, we update table[j] with length of current LCIS. To maintain current LCIS, we keep checking valid table[j] values.  
Below is the program to find length of LCIS. 

* C++
* Java
* Python 3
* C#
* PHP
* Javascript

|  |
| --- |
| // A C++ Program to find length of the Longest Common  // Increasing Subsequence (LCIS)  #include<bits/stdc++.h>  using namespace std;    // Returns the length and the LCIS of two  // arrays arr1[0..n-1] and arr2[0..m-1]  int LCIS(int arr1[], int n, int arr2[], int m)  {      // table[j] is going to store length of LCIS      // ending with arr2[j]. We initialize it as 0,      int table[m];      for (int j=0; j<m; j++)          table[j] = 0;        // Traverse all elements of arr1[]      for (int i=0; i<n; i++)      {          // Initialize current length of LCIS          int current = 0;            // For each element of arr1[], traverse all          // elements of arr2[].          for (int j=0; j<m; j++)          {              // If both the array have same elements.              // Note that we don't break the loop here.              if (arr1[i] == arr2[j])                  if (current + 1 > table[j])                      table[j] = current + 1;                /\* Now seek for previous smaller common                 element for current element of arr1 \*/              if (arr1[i] > arr2[j])                  if (table[j] > current)                      current = table[j];          }      }        // The maximum value in table[] is out result      int result = 0;      for (int i=0; i<m; i++)          if (table[i] > result)             result = table[i];        return result;  }    /\* Driver program to test above function \*/  int main()  {      int arr1[] = {3, 4, 9, 1};      int arr2[] = {5, 3, 8, 9, 10, 2, 1};        int n = sizeof(arr1)/sizeof(arr1[0]);      int m = sizeof(arr2)/sizeof(arr2[0]);        cout << "Length of LCIS is "           << LCIS(arr1, n, arr2, m);      return (0);  } |

**Output :**

Length of LCIS is 2

This article is contributed **Rachit Belwariar**. If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.  
Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Attention reader! Don’t stop learning now. Get hold of all the important DSA concepts with the [**DSA Self Paced Course**](https://practice.geeksforgeeks.org/courses/dsa-self-paced?utm_source=geeksforgeeks&utm_medium=article&utm_campaign=gfg_article_dsa_content_bottom) at a student-friendly price and become industry ready.  To complete your preparation from learning a language to DS Algo and many more,  please refer [**Complete Interview Preparation Course**](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=GeeksforGeeks&utm_medium=Text&utm_campaign=GFG_Article_Bottom_Text_CIP)**.**

In case you wish to attend live classes with industry experts, please refer [**DSA Live Classes**](https://practice.geeksforgeeks.org/courses/geeks-classes-live?utm_source=geeksforgeeks&utm_medium=article&utm_campaign=gfg_article_dsa_content_bottom)

Printing Longest Common Subsequence | Set 2 (Printing All)

* Difficulty Level : [Hard](https://www.geeksforgeeks.org/hard/)
* Last Updated : 15 Jun, 2021

Given two sequences, print all longest subsequence present in both of them.  
**Examples:** 

**Input:**

string X = "AGTGATG"

string Y = "GTTAG"

**Output:**

GTAG

GTTG

**Input:**

string X = "AATCC"

string Y = "ACACG"

**Output:**

ACC

AAC

**Input:**

string X = "ABCBDAB"

string Y = "BDCABA"

**Output:**

BCAB

BCBA

BDAB

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

We have discussed Longest Common Subsequence (LCS) problem [here](https://www.geeksforgeeks.org/dynamic-programming-set-4-longest-common-subsequence/). The function discussed there was mainly to find the length of LCS. We have also discussed how to print the longest subsequence [here](https://www.geeksforgeeks.org/printing-longest-common-subsequence/). But as LCS for two strings is not unique, in this post we will print out all the possible solutions to LCS problem.  
Following is detailed algorithm to print the all LCS.  
We construct L[m+1][n+1] table as discussed in the [previous](https://www.geeksforgeeks.org/dynamic-programming-set-4-longest-common-subsequence/) post and traverse the 2D array starting from L[m][n]. For current cell L[i][j] in the matrix,  
a) If the last characters of X and Y are same (i.e. X[i-1] == Y[j-1]), then the character must be present in all LCS of substring X[0…i-1] and Y[0..j-1]. We simply recurse for L[i-1][j-1] in the matrix and append current character to all LCS possible of substring X[0…i-2] and Y[0..j-2].  
b) If the last characters of X and Y are not same (i.e. X[i-1] != Y[j-1]), then LCS can be constructed from either top side of the matrix (i.e. L[i-1][j]) or from left side of matrix (i.e. L[i][j-1]) depending upon which value is greater. If both the values are equal(i.e. L[i-1][j] == L[i][j-1]), then it will be constructed from both sides of matrix. So based on values at L[i-1][j] and L[i][j-1], we go in direction of greater value or go in both directions if the values are equal.  
Below is recursive implementation of above idea – 

* C++
* Java
* Python3
* C#
* Javascript

|  |
| --- |
| /\* Dynamic Programming implementation of LCS problem \*/  #include <bits/stdc++.h>  using namespace std;    // Maximum string length  #define N 100    int L[N][N];    /\* Returns set containing all LCS for X[0..m-1], Y[0..n-1] \*/  set<string> findLCS(string X, string Y, int m, int n)  {      // construct a set to store possible LCS      set<string> s;        // If we reaches end of either string, return      // a empty set      if (m == 0 || n == 0)      {          s.insert("");          return s;      }        // If the last characters of X and Y are same      if (X[m - 1] == Y[n - 1])      {          // recurse for X[0..m-2] and Y[0..n-2] in          // the matrix          set<string> tmp = findLCS(X, Y, m - 1, n - 1);            // append current character to all possible LCS          // of substring X[0..m-2] and Y[0..n-2].          for (string str : tmp)              s.insert(str + X[m - 1]);      }        // If the last characters of X and Y are not same      else      {          // If LCS can be constructed from top side of          // the matrix, recurse for X[0..m-2] and Y[0..n-1]          if (L[m - 1][n] >= L[m][n - 1])              s = findLCS(X, Y, m - 1, n);            // If LCS can be constructed from left side of          // the matrix, recurse for X[0..m-1] and Y[0..n-2]          if (L[m][n - 1] >= L[m - 1][n])          {              set<string> tmp = findLCS(X, Y, m, n - 1);                // merge two sets if L[m-1][n] == L[m][n-1]              // Note s will be empty if L[m-1][n] != L[m][n-1]              s.insert(tmp.begin(), tmp.end());          }      }      return s;  }    /\* Returns length of LCS for X[0..m-1], Y[0..n-1] \*/  int LCS(string X, string Y, int m, int n)  {      // Build L[m+1][n+1] in bottom up fashion      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];  }    /\* Driver program to test above function \*/  int main()  {      string X = "AGTGATG";      string Y = "GTTAG";      int m = X.length();      int n = Y.length();        cout << "LCS length is " << LCS(X, Y, m, n) << endl;        set<string> s = findLCS(X, Y, m, n);        for (string str : s)          cout << str << endl;        return 0;  } |

**Output:**

LCS length is 4

GTAG

GTTG

References: [Wikibooks – Reading out all LCSs](https://en.wikibooks.org/wiki/Algorithm_Implementation/Strings/Longest_common_subsequence#Reading_out_all_LCSs)  
This article is contributed by **Aditya Goel**. If you like GeeksforGeeks and would like to contribute, you can also write an article using [write.geeksforgeeks.org](http://www.write.geeksforgeeks.org/) or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.  
Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Attention reader! Don’t stop learning now. Get hold of all the important DSA concepts with the [**DSA Self Paced Course**](https://practice.geeksforgeeks.org/courses/dsa-self-paced?utm_source=geeksforgeeks&utm_medium=article&utm_campaign=gfg_article_dsa_content_bottom) at a student-friendly price and become industry ready.  To complete your preparation from learning a language to DS Algo and many more,  please refer [**Complete Interview Preparation Course**](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=GeeksforGeeks&utm_medium=Text&utm_campaign=GFG_Article_Bottom_Text_CIP)**.**

In case you wish to attend live classes with industry experts, please refer [**DSA Live Classes**](https://practice.geeksforgeeks.org/courses/geeks-classes-live?utm_source=geeksforgeeks&utm_medium=article&utm_campaign=gfg_article_dsa_content_bottom)

Longest common subsequence with permutations allowed

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 06 Nov, 2020

Given two strings in lowercase, find the longest string whose permutations are subsequences of given two strings. The output longest string must be sorted.

**Examples:**

Input : str1 = "pink", str2 = "kite"

Output : "ik"

The string "ik" is the longest sorted string

whose one permutation "ik" is subsequence of

"pink" and another permutation "ki" is

subsequence of "kite".

Input : str1 = "working", str2 = "women"

Output : "now"

Input : str1 = "geeks" , str2 = "cake"

Output : "ek"

Input : str1 = "aaaa" , str2 = "baba"

Output : "aa"

[Recommended: Please solve it on “***PRACTICE***” first, before moving on to the solution.](https://practice.geeksforgeeks.org/problems/lcs-with-permutations/0)

The idea is to count characters in both strings.

1. calculate frequency of characters for each string and store them in their respective count arrays, say count1[] for str1 and count2[] for str2.
2. Now we have count arrays for 26 characters. So traverse count1[] and for any index ‘i’ append character (‘a’+i) in resultant string ‘result’ by min(count1[i], count2[i]) times.
3. Since we traverse count array in ascending order, our final string characters will be in sorted order.
   * C++
   * Java
   * Python 3
   * C#
   * PHP

|  |
| --- |
| // C++ program to find LCS with permutations allowed  #include<bits/stdc++.h>  using namespace std;    // Function to calculate longest string  // str1     --> first string  // str2     --> second string  // count1[]  --> hash array to calculate frequency  //              of characters in str1  // count[2]  --> hash array to calculate frequency  //              of characters in str2  // result   --> resultant longest string whose  // permutations are sub-sequence of given two strings  void longestString(string str1, string str2)  {      int count1[26] = {0}, count2[26]= {0};        // calculate frequency  of characters      for (int i=0; i<str1.length(); i++)          count1[str1[i]-'a']++;      for (int i=0; i<str2.length(); i++)          count2[str2[i]-'a']++;        // Now traverse hash array      string result;      for (int i=0; i<26; i++)            // append character ('a'+i) in resultant          // string 'result' by min(count1[i],count2i])          // times          for (int j=1; j<=min(count1[i],count2[i]); j++)              result.push\_back('a' + i);        cout << result;  }    // Driver program to run the case  int main()  {      string str1 = "geeks", str2 = "cake";      longestString(str1, str2);      return 0;  } |

1. **Output:**
3. ek
4. **Time Complexity:** O(m + n) where m and n are lengths of input strings.  
   **Auxiliary Space:** O(1)  
     
   If you have another approach to solve this problem then please share.
5. This article is contributed by [Shashank Mishra ( Gullu )](https://practice.geeksforgeeks.org/user-profile.php?user=Shashank%20Mishra). If you like GeeksforGeeks and would like to contribute, you can also write an article using [contribute.geeksforgeeks.org](http://www.contribute.geeksforgeeks.org/) or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.
6. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.
7. Attention reader! Don’t stop learning now. Get hold of all the important DSA concepts with the [**DSA Self Paced Course**](https://practice.geeksforgeeks.org/courses/dsa-self-paced?utm_source=geeksforgeeks&utm_medium=article&utm_campaign=gfg_article_dsa_content_bottom) at a student-friendly price and become industry ready.  To complete your preparation from learning a language to DS Algo and many more,  please refer [**Complete Interview Preparation Course**](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=GeeksforGeeks&utm_medium=Text&utm_campaign=GFG_Article_Bottom_Text_CIP)**.**
8. In case you wish to attend live classes with industry experts, please refer [**DSA Live Classes**](https://practice.geeksforgeeks.org/courses/geeks-classes-live?utm_source=geeksforgeeks&utm_medium=article&utm_campaign=gfg_article_dsa_content_bottom)

Longest common subsequence with permutations allowed

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 06 Nov, 2020

Given two strings in lowercase, find the longest string whose permutations are subsequences of given two strings. The output longest string must be sorted.

**Examples:**

Input : str1 = "pink", str2 = "kite"

Output : "ik"

The string "ik" is the longest sorted string

whose one permutation "ik" is subsequence of

"pink" and another permutation "ki" is

subsequence of "kite".

Input : str1 = "working", str2 = "women"

Output : "now"

Input : str1 = "geeks" , str2 = "cake"

Output : "ek"

Input : str1 = "aaaa" , str2 = "baba"

Output : "aa"

[Recommended: Please solve it on “***PRACTICE***” first, before moving on to the solution.](https://practice.geeksforgeeks.org/problems/lcs-with-permutations/0)

The idea is to count characters in both strings.

1. calculate frequency of characters for each string and store them in their respective count arrays, say count1[] for str1 and count2[] for str2.
2. Now we have count arrays for 26 characters. So traverse count1[] and for any index ‘i’ append character (‘a’+i) in resultant string ‘result’ by min(count1[i], count2[i]) times.
3. Since we traverse count array in ascending order, our final string characters will be in sorted order.
   * C++
   * Java
   * Python 3
   * C#
   * PHP

|  |
| --- |
| // C++ program to find LCS with permutations allowed  #include<bits/stdc++.h>  using namespace std;    // Function to calculate longest string  // str1     --> first string  // str2     --> second string  // count1[]  --> hash array to calculate frequency  //              of characters in str1  // count[2]  --> hash array to calculate frequency  //              of characters in str2  // result   --> resultant longest string whose  // permutations are sub-sequence of given two strings  void longestString(string str1, string str2)  {      int count1[26] = {0}, count2[26]= {0};        // calculate frequency  of characters      for (int i=0; i<str1.length(); i++)          count1[str1[i]-'a']++;      for (int i=0; i<str2.length(); i++)          count2[str2[i]-'a']++;        // Now traverse hash array      string result;      for (int i=0; i<26; i++)            // append character ('a'+i) in resultant          // string 'result' by min(count1[i],count2i])          // times          for (int j=1; j<=min(count1[i],count2[i]); j++)              result.push\_back('a' + i);        cout << result;  }    // Driver program to run the case  int main()  {      string str1 = "geeks", str2 = "cake";      longestString(str1, str2);      return 0;  } |

1. **Output:**
3. ek
4. **Time Complexity:** O(m + n) where m and n are lengths of input strings.  
   **Auxiliary Space:** O(1)  
     
   If you have another approach to solve this problem then please share.
5. This article is contributed by [Shashank Mishra ( Gullu )](https://practice.geeksforgeeks.org/user-profile.php?user=Shashank%20Mishra). If you like GeeksforGeeks and would like to contribute, you can also write an article using [contribute.geeksforgeeks.org](http://www.contribute.geeksforgeeks.org/) or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.
6. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.
7. Attention reader! Don’t stop learning now. Get hold of all the important DSA concepts with the [**DSA Self Paced Course**](https://practice.geeksforgeeks.org/courses/dsa-self-paced?utm_source=geeksforgeeks&utm_medium=article&utm_campaign=gfg_article_dsa_content_bottom) at a student-friendly price and become industry ready.  To complete your preparation from learning a language to DS Algo and many more,  please refer [**Complete Interview Preparation Course**](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=GeeksforGeeks&utm_medium=Text&utm_campaign=GFG_Article_Bottom_Text_CIP)**.**
8. In case you wish to attend live classes with industry experts, please refer [**DSA Live Classes**](https://practice.geeksforgeeks.org/courses/geeks-classes-live?utm_source=geeksforgeeks&utm_medium=article&utm_campaign=gfg_article_dsa_content_bottom)

LCS (Longest Common Subsequence) of three strings

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 17 Jun, 2021

Given 3 strings of all having length < 100,the task is to find the longest common sub-sequence in all three given sequences.  
**Examples:** 

Input : str1 = "geeks"

str2 = "geeksfor"

str3 = "geeksforgeeks"

Output : 5

Longest common subsequence is "geeks"

i.e., length = 5

Input : str1 = "abcd1e2"

str2 = "bc12ea"

str3 = "bd1ea"

Output : 3

Longest common subsequence is "b1e"

i.e. length = 3.

[Recommended: Please solve it on “***PRACTICE*** ” first, before moving on to the solution.](https://practice.geeksforgeeks.org/problems/lcs-of-three-strings/0)

This problem is simply an extension of [LCS](https://www.geeksforgeeks.org/dynamic-programming-set-4-longest-common-subsequence/)  
Let the input sequences be X[0..m-1], Y[0..n-1] and Z[0..o-1] of lengths m, n and o respectively. And let L(X[0..m-1], Y[0..n-1], Z[0..o-1]) be the lengths of LCS of the three sequences X, Y and Z. Following is the implementation:

The idea is to take a 3D array to store the

length of common subsequence in all 3 given

sequences i. e., L[m + 1][n + 1][o + 1]

1- If any of the string is empty then there

is no common subsequence at all then

L[i][j][k] = 0

2- If the characters of all sequences match

(or X[i] == Y[j] ==Z[k]) then

L[i][j][k] = 1 + L[i-1][j-1][k-1]

3- If the characters of both sequences do

not match (or X[i] != Y[j] || X[i] != Z[k]

|| Y[j] !=Z[k]) then

L[i][j][k] = max(L[i-1][j][k],

L[i][j-1][k],

L[i][j][k-1])

Below is implementation of above idea.

* C++
* Java
* Python3
* C#
* PHP
* Javascript

|  |
| --- |
| // C++ program to find LCS of three strings  #include<bits/stdc++.h>  using namespace std;    /\* Returns length of LCS for X[0..m-1], Y[0..n-1]     and Z[0..o-1] \*/  int lcsOf3( string X, string Y, string Z, int m,                                 int n, int o)  {      int L[m+1][n+1][o+1];        /\* Following steps build L[m+1][n+1][o+1] in         bottom up fashion. Note that L[i][j][k]         contains length of LCS of X[0..i-1] and         Y[0..j-1]  and Z[0.....k-1]\*/      for (int i=0; i<=m; i++)      {          for (int j=0; j<=n; j++)          {              for (int k=0; k<=o; k++)              {                  if (i == 0 || j == 0||k==0)                      L[i][j][k] = 0;                    else if (X[i-1] == Y[j-1] && X[i-1]==Z[k-1])                      L[i][j][k] = L[i-1][j-1][k-1] + 1;                    else                      L[i][j][k] = max(max(L[i-1][j][k],                                           L[i][j-1][k]),                                       L[i][j][k-1]);              }          }      }        /\* L[m][n][o] contains length of LCS for        X[0..n-1] and Y[0..m-1] and Z[0..o-1]\*/      return L[m][n][o];  }    /\* Driver program to test above function \*/  int main()  {      string X = "AGGT12";      string Y = "12TXAYB";      string Z = "12XBA";        int m = X.length();      int n = Y.length();      int o = Z.length();        cout << "Length of LCS is " << lcsOf3(X, Y,                                      Z, m, n, o);        return 0;  } |

**Output:** 

Length of LCS is 2

**Another approach:**(Using recursion) 

* C++
* Java
* Python3
* C#
* PHP
* Javascript

|  |
| --- |
| // C++ program to find LCS of three strings  #include<bits/stdc++.h>  using namespace std;        string X = "AGGT12";      string Y = "12TXAYB";      string Z = "12XBA";    int dp[100][100][100];    /\* Returns length of LCS for X[0..m-1], Y[0..n-1]  and Z[0..o-1] \*/  int lcsOf3(int i, int j,int k)  {      if(i==-1||j==-1||k==-1)          return 0;      if(dp[i][j][k]!=-1)          return dp[i][j][k];        if(X[i]==Y[j] && Y[j]==Z[k])          return dp[i][j][k] = 1+lcsOf3(i-1,j-1,k-1);      else          return dp[i][j][k] = max(max(lcsOf3(i-1,j,k),                              lcsOf3(i,j-1,k)),lcsOf3(i,j,k-1));  }    // Driver code  int main()  {      memset(dp, -1,sizeof(dp));      int m = X.length();      int n = Y.length();      int o = Z.length();        cout << "Length of LCS is " << lcsOf3(m-1,n-1,o-1);  // this code is contributed by Kushdeep Mittal  } |

**Output:** 

Length of LCS is 2

This article is contributed by [**Sahil Chhabra (akku)**](https://practice.geeksforgeeks.org/user-profile.php?user=sahil_coder). If you like GeeksforGeeks and would like to contribute, you can also write an article using [write.geeksforgeeks.org](http://www.write.geeksforgeeks.org/) or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.  
Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Attention reader! Don’t stop learning now. Get hold of all the important DSA concepts with the [**DSA Self Paced Course**](https://practice.geeksforgeeks.org/courses/dsa-self-paced?utm_source=geeksforgeeks&utm_medium=article&utm_campaign=gfg_article_dsa_content_bottom) at a student-friendly price and become industry ready.  To complete your preparation from learning a language to DS Algo and many more,  please refer [**Complete Interview Preparation Course**](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=GeeksforGeeks&utm_medium=Text&utm_campaign=GFG_Article_Bottom_Text_CIP)**.**

In case you wish to attend live classes with industry experts, please refer [**DSA Live Classes**](https://practice.geeksforgeeks.org/courses/geeks-classes-live?utm_source=geeksforgeeks&utm_medium=article&utm_campaign=gfg_article_dsa_content_bottom)

C++ Program for Longest Common Subsequence

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 04 Dec, 2018

*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”. So a string of length n has 2^n different possible subsequences.

It is a classic computer science problem, the basis of [diff](http://en.wikipedia.org/wiki/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.

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])

|  |
| --- |
| /\* 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\n", lcs(X, Y, m, n));        return 0;  } |

**Output:**

Length of LCS is 4

Following is a tabulated implementation for the LCS problem.

|  |
| --- |
| /\* Dynamic Programming C/C++ 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)  {      int L[m + 1][n + 1];      int 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];  }    /\* 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\n", lcs(X, Y, m, n));        return 0;  } |

**Output:**

Length of LCS is 4

Please refer complete article on [Dynamic Programming | Set 4 (Longest Common Subsequence)](https://www.geeksforgeeks.org/longest-common-subsequence/) for more details!

Want to learn from the best curated videos and practice problems, check out the [**C++ Foundation Course**](https://practice.geeksforgeeks.org/courses/CPP-Foundation?utm_source=GeeksforGeeks&utm_medium=Text&utm_campaign=GFG_Article_Bottom_Text_CPP) for Basic to Advanced C++ and [**C++ STL Course**](https://practice.geeksforgeeks.org/courses/cpp-stl?utm_source=GeeksforGeeks&utm_medium=Text&utm_campaign=GFG_Article_Bottom_Text_CPP_STL) for the language and STL. To complete your preparation from learning a language to DS Algo and many more, please refer [**Complete Interview Preparation Course**](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=GeeksforGeeks&utm_medium=Text&utm_campaign=GFG_Article_Bottom_Text_CIP)**.**