**How to check if question can be solved by LCS ?**

Two String/array will be given and in question it will be asked to find out longest/shortest array or String based on some condition.

A **subsequence** is a sequence that can be derived from another sequence by deleting some or no elements without changing the order of the remaining elements.

1. **Longest common sub-sequence/string (Recursive approach)**

String x = "abcdgh";

String y = "abedfhr";

Here common character in string is a, b, d, h so longest common subsequence will be "abdh"

Common substring is "ab" because substring should be common and in sequence.

For any recursive program there is three things we need to do

1. Base Condition
2. Method(IP) -- > method(Smaller IP)
3. Choice Diagram

Base condition for this problem is

m = x.lenght;

n = y.length;

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

return 0;

Suppose we have String like bellow

String x = "abcdgh";

String y = "abedfh";

Here we will compare the last char of x and last char of y. so x[m-1] == y[n-1]. Here it will return true. So in this case we will take last char of string and append it into OP String and remove the last char of the String from the both String.

So now both String will look like

String x = "abcdg";

String y = "abedf";

Now if we compare x[m-1] == y[n-1] it will return false. In this case we need to make smaller IP for x and take the whole y String of y as is and vice-versa. Like bellow

String x = "abcd"; String x = "abcdg";

String y = "abedf"; String y = "abed";

Here we will do like above step because even if x[m-1] != y[n-1] then it might be x[m-2] == y[n-1]

Suppose if string is then in this case it will match x[m-1] == y[n-1]

String x = "abcf";

String y = "abedf";

So we will check x and y string by making it smaller one by one.

**if** (x.charAt(m - 1) == y.charAt(n - 1)) {

**return** 1 + lcs(x, y, m - 1, n - 1);

} **else** {

**return** Math.*max*(lcs(x, y, m, n - 1), lcs(x, y, m - 1, n));

}

For complete code

<https://github.com/hareramcse/Datastructure/blob/master/DP/src/com/hs/dp/lcs/LCSRecursive.java>

1. **Longest common sub-sequence( Memoized approach)**

String x = "abcdgh";

String y = "abedfhr";

In recursive call same method call happens for many times. To escape that repeated method we use memoization.

For this we will take

**int**[][] dp = **new** **int**[m + 1][n + 1];

and before doing any recursive call we will check

**if** (dp[m][n] != 0) {

**return** dp[m][n];

}

Else we will store the recursive call result into dp[m][n]

**if** (x.charAt(m - 1) == y.charAt(n - 1)) {

**return** dp[m][n] = 1 + lcs(x, y, m - 1, n - 1);

} **else** {

**return** dp[m][n] = Math.*max*(lcs(x, y, m, n - 1), lcs(x, y, m - 1, n));

}

<https://github.com/hareramcse/Datastructure/blob/master/DP/src/com/hs/dp/lcs/LCSRecursiveMemoized.java>

1. **Longest common sub-sequence(Tabular approach)**

String x = "abcdgh";

String y = "abedfhr";

In Tabular approach we take matrix of size m+1, n+1 where m is size of x string and n is the size of y string.

**int**[][] dp = **new** **int**[m + 1][n + 1];

**int [][] dp = new int[7][8]**

In recursive call we have base condition. Here base condition is converted into initialization.

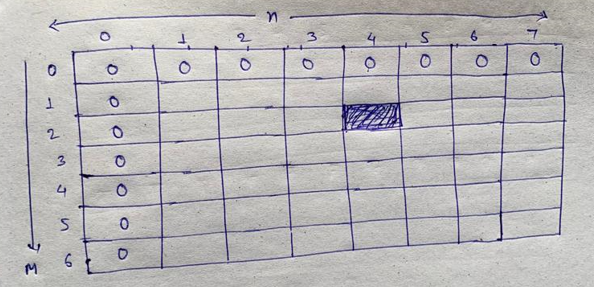
For recursive call we had base condition

**if** (m == 0 || n == 0) {

**return** 0;

}

In initialization phase we initialize the 1st row and 1st column with 0



Here value in dp[2][4] means, LCS count for the String x = ab(Size 2) and String y = abed(Size 4)

We need LCS count for

String x = "abcdgh"; (length m)

String y = "abedfhr"; (length n)

So we need to check the value of dp[m][n]

In this approach we will replace m -> i and n -> j and method name to dp

That’s it.

**else** **if** (text1.charAt(i - 1) == text2.charAt(j - 1)) {

dp[i][j] = 1 + dp[i - 1][j - 1];

} **else** {

dp[i][j] = Math.*max*(dp[i - 1][j], dp[i][j - 1]);

}

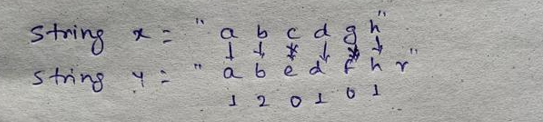
<https://github.com/hareramcse/Datastructure/blob/master/DP/src/com/hs/dp/lcs/LCSTabulation.java>

1. **Longest common substring**

String x = "abcdgh";

String y = "abedfhr";

Here we have to find out Longest common substring. Substring means it should be continuous String.



Here logic is

We will do the same operation like lcs except that. If there is discontinuity then we will make the count = 0;

So in the above example we will have 3 SubString ab, d, h

So max length of the SubString is 2

**else** **if** (x.charAt(i - 1) == y.charAt(j - 1)) {

dp[i][j] = 1 + dp[i - 1][j - 1];

count = Math.*max*(dp[i][j], count);

} **else** {

dp[i][j] = 0;

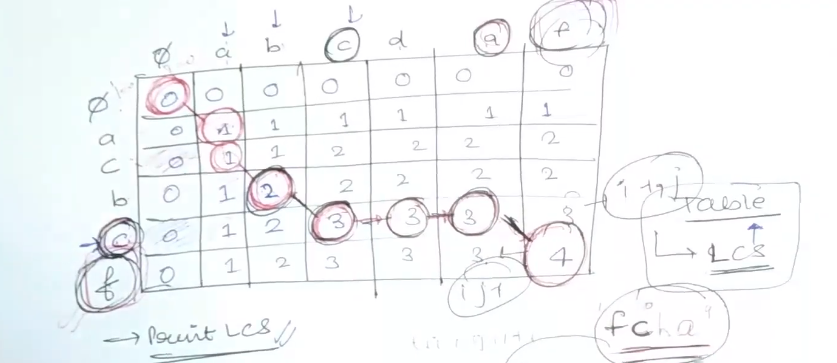
}

<https://github.com/hareramcse/Datastructure/blob/master/DP/src/com/hs/dp/lcs/LCSubstring.java>

1. **LCS print**

String x = "abcdgh";

String y = "abedfhr";



After LCS we will get the matrix like above.

**while** (i > 0 && j > 0) {

**if** (x.charAt(i - 1) == y.charAt(j - 1)) {

result = result + x.charAt(i - 1);

i--;

j--;

} **else** {

**if** (dp[i - 1][j] > dp[i][j - 1])

i--;

**else**

j--;

}

}

After doing this result string will be fcba. We need to reverse it to get the answer as we are traversing it from the end.

<https://github.com/hareramcse/Datastructure/blob/master/DP/src/com/hs/dp/lcs/LCSPrint.java>

1. **Shortest common super sequence String count**

String x = "aggtab";

String y = "gxtxayb";

Given two strings str1 and str2, return *the shortest string that has both*str1*and*str2*as****subsequences***. If there are multiple valid strings, return **any** of them.

Op: 9

Append both String x and y it will become aggtabgxtxayb.

Now in the appended String we have some common char that is gtab.

This gtab is nothing but LCS of the both String. If we see the appended String carefully, we see that this LCS char is coming twice. So we will remove the LCS char from the appended only once. So in the appended String LCS char will be only once.

After removing the LCS char from appended String it will be aggxtxayb and this is SCS of the above String.

Aggxtxayb………………………………………… Length of the above string is 9

Algo:

Append both Strings x and y length and then subtract lcs from it.

<https://github.com/hareramcse/Datastructure/blob/master/DP/src/com/hs/dp/lcs/SCSSTabulation.java>

1. **Minimum no of insertion and deletion to convert String A to B**

String x = "heap";

String y = "pea";

If we delete h and p from String x then it will become ea. Now we will add p before ea then it will become pea.

So Deletion = 2 and insertion will be 1

LCS of the above String is ea so its length will be 2

So we can conclude that

No of deletion will be X – LCS (4 – 2) = 2

No of insertion will be Y – LCS (3 – 2) = 1

<https://github.com/hareramcse/Datastructure/blob/master/DP/src/com/hs/dp/lcs/MinNoOfInsertionDeletionToConvertAtoB.java>

1. **Longest palindromic subsequence**

String x = "aggtab";

Given a string s, find *the longest palindromic****subsequence****'s length in* s.

Steps:

To use LCS we need 2 Strings….but in the input only one String is given

For the 2nd String we can reverse the 1st String and can take as 2nd String.

So String x = “aggtab”;

String y = “batgga”;

Here LCS of x and y is agga and its length will be 4

And this will be the longest palindromic sequence for the String “aggtab”

<https://github.com/hareramcse/Datastructure/blob/master/DP/src/com/hs/dp/lcs/LPSTabulation.java>

**10) Minimum no of deletion in string to make it palindrome.**

String x = "aggtab";

LPS of the above String is agga so its length will be 4. And total length of the above String is 6. It means we should remove 2 char from String x to make it palindrome.

If we find out x.length() – lcs then it will be min no of deletion.

<https://github.com/hareramcse/Datastructure/blob/master/DP/src/com/hs/dp/lcs/MinNoOfDeletionToConvertIntoPolindrom.java>

**11) Print shortest super common sequence**

String x = "aggtab";

String y = "gxtxayb";

Algo:

LCS of the above String will be gtab.

Start from the end i=m, j=n

While i>0 && j>0

Check if i-1th char and j-1th char is same

If it is same then add into result. And i--; j—

If dp[i-1][j] > dp[j][i-1] then add the str1[i-1]th char to result and i--

Else add str2[j-1]th char to result and j--

If any of the string ends early…append that string to result.

And last reverse the string……this is shortest common super sequence

<https://github.com/hareramcse/Datastructure/blob/master/DP/src/com/hs/dp/lcs/SCSSPrint.java>

**12) Longest repeating subsequence**

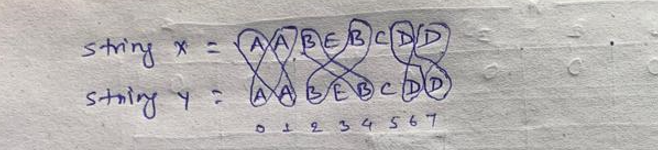
String x = "AABEBCDD";

Find the longest repeating subsequence of the above String.

Solution:

We can take same String as String y

String y = "AABEBCDD"



Here we can see the ABD is coming twice. If we remove the ABD from the above String then we will have only EC

In the above String we can see that E and C is coming only once. And the index of E and C in both String is same.

For other String ABD we see that is is occurring 2 times at 0 and 1 index. So we can map one A(0th index) to i and other A(1st Index) to j. similarly for B and D.

Here we can see that we can have ABD by using i != j.

And EC will come when we do i == j

Now we can find out the LCS of the above String. While finding the LCS we will add the condition i != j

<https://github.com/hareramcse/Datastructure/blob/master/DP/src/com/hs/dp/lcs/LongestRepeatingSubsequence.java>

**13) Sequence Pattern Matching**

String x = "abc";

String y = "ahbgdc";

Given two strings x and y, return true*if*x*is a****subsequence****of*y*, or*false*otherwise*.

Sol:

Find the LCS of the above String which is abc. If x.length == LCS length then return true else false.

<https://github.com/hareramcse/Datastructure/blob/master/DP/src/com/hs/dp/lcs/SequencePatternMatching.java>

14) **Minimum no of insertion to make a String palindrome.**

String x = "aggtab";

Find the minimum no of insertion into this string so that it will be palindrome.

Sol:

LPS of the above String is agga and length of x String is 6

So, Min insertion to make it palindrome will be x.length – LPS count.

<https://github.com/hareramcse/Datastructure/blob/master/DP/src/com/hs/dp/lcs/MinNoOfInsertionToConvertIntoPolindrom.java>