Longest Palindromic Subsequence | DP-12

- Difficulty Level : Medium
- Last Updated : 01 Sep, 2021

Given a sequence, find the length of the longest palindromic subsequence in it.

Input : GEEKSFORGEEKS
Output : 5

The longest palindromic subsequence we can get is of length 5. There are more than palindromic subsequences of length 5, for example, EEKEE, EESEE, EEFEE, ..

As another example, if the given sequence is “BBABCBCAB”, then the output should be 7 as “BABCBAB” is the longest palindromic subsequence in it. “BBBBB” and “BBCBB” are also palindromic subsequences of the given sequence, but not the longest ones.

The naive solution for this problem is to generate all subsequences of the given sequence and find the longest palindromic 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 and can efficiently be solved using Dynamic Programming.

1) Optimal Substructure:

Let X[0..n-1] be the input sequence of length n and L(0, n-1) be the length of the longest palindromic subsequence of X[0..n-1].

If last and first characters of X are same, then L(0, n-1) = L(1, n-2) + 2.

Else L(0, n-1) = MAX (L(1, n-1), L(0, n-2)).

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

Following is a general recursive solution with all cases handled.

```
// Every single character is a palindrome of length 1
L(i, i) = 1 for all indexes i in given sequence
// IF first and last characters are not same
If (X[i] != X[j])  L(i, j) =  max{L(i + 1, j),L(i, j - 1)}

// If there are only 2 characters and both are same
Else if (j == i + 1) L(i, j) = 2

// If there are more than two characters, and first and last
// characters are same
Else L(i, j) =  L(i + 1, j - 1) + 2
```

2) Overlapping Subproblems

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

- C++
- C
- Java
- Python3
- C#
- PHP
- Javascript

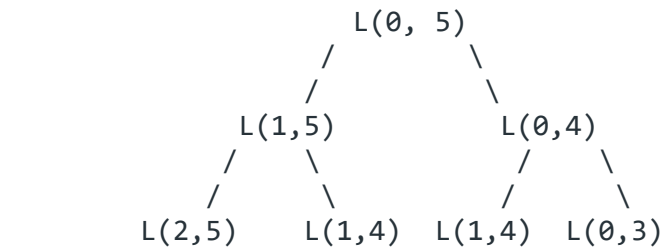
```
//Java program of above approach

class GFG {

    // A utility function to get max of two integers
    static int max(int x, int y) {
        return (x > y) ? x : y;
    }
    // Returns the length of the longest palindromic subsequence in seq

    static int lps(char seq[], int i, int j) {
    // Base Case 1: If there is only 1 character
        if (i == j) {
            return 1;
        }

    // Base Case 2: If there are only 2 characters and both are same
        if (seq[i] == seq[j] && i + 1 == j) {
            return 2;
        }
```

```
    // If the first and last characters match
        if (seq[i] == seq[j]) {
            return lps(seq, i + 1, j - 1) + 2;
        }

    // If the first and last characters do not match
        return max(lps(seq, i, j - 1), lps(seq, i + 1, j));
    }


    /* Driver program to test above function */
    public static void main(String[] args) {
        String seq = "GEEKSFORGEEKS";
        int n = seq.length();
        System.out.printf("The length of the LPS is %d",
lps(seq.toCharArray(), 0, n - 1));

    }
}
```

Output:

```
The length of the LPS is 5
```

Considering the above implementation, the following is a partial recursion tree for a sequence of length 6 with all different characters.



In the above partial recursion tree, L(1, 4) 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. Since the same subproblems are called again, this problem has Overlapping Subproblems property. So LPS problem has both properties (see this and this) of a dynamic programming problem. Like other typical Dynamic Programming(DP) problems, recomputations of same subproblems can be avoided by constructing a temporary array L[][] in a bottom-up manner.

Dynamic Programming Solution

- C++
- Java
- Python
- C#
- PHP
- Javascript

```
// A Dynamic Programming based Java
// Program for the Egg Dropping Puzzle
```

```
class LPS
{

    // A utility function to get max of two integers
    static int max (int x, int y) { return (x > y)? x : y; }

    // Returns the length of the longest
    // palindromic subsequence in seq
    static int lps(String seq)
    {
    int n = seq.length();
    int i, j, cl;
    // Create a table to store results of subproblems
    int L[][] = new int[n][n];

    // Strings of length 1 are palindrome of length 1
    for (i = 0; i < n; i++)
        L[i][i] = 1;

        // Build the table. Note that the lower
        // diagonal values of table are
        // useless and not filled in the process.
        // The values are filled in a manner similar
        //  to Matrix Chain Multiplication DP solution (See
        // https://www.geeksforgeeks.org/matrix-chain-multiplication-dp-8/).
        // cl is length of substring
        for (cl=2; cl<=n; cl++)
        {
            for (i=0; i<n-cl+1; i++)
            {
                j = i+cl-1;
                if (seq.charAt(i) == seq.charAt(j) && cl == 2)
                L[i][j] = 2;
                else if (seq.charAt(i) == seq.charAt(j))
                L[i][j] = L[i+1][j-1] + 2;
                else
                L[i][j] = max(L[i][j-1], L[i+1][j]);
            }
        }

        return L[0][n-1];
    }

    /* Driver program to test above functions */
    public static void main(String args[])
    {
        String seq = "GEEKSFORGEEKS";
        int n = seq.length();
        System.out.println("The length of the lps is "+ lps(seq));
    }
}
/* This code is contributed by Rajat Mishra */
```

Output:

```
The length of the LPS is 7
```

Time Complexity of the above implementation is O(n^2) which is much better than the worst-case time complexity of Naive Recursive implementation.

From <https://www.geeksforgeeks.org/longest-palindromic-subsequence-dp-12/>

1. Optimal Solution top down

```
class Solution {
    public int longestPalindromeSubseq(String s) {
        int n = s.length();
        int[][] dp = new int[n][n];
        for (int i = n - 1; i >= 0; i--) {
            dp[i][i] = 1;
            for (int j = i + 1; j < n; j++) {
                if (s.charAt(i) == s.charAt(j)) {
                    dp[i][j] = dp[i + 1][j - 1] + 2;
                } else {
                    dp[i][j] = Math.max(dp[i + 1][j], dp[i][j - 1]);
                }
            }
        }
        return dp[0][n - 1];
```

```
    }
}
```

2. Best Time Solution

```
class Solution {

    public int longestPalindromeSubseq(String s) {
        final short l = (short) s.length();

        short[] dp = new short[l];


        char[] c = s.toCharArray();
        short max = 0;

        for (short i=0; i<l; i++) {

            dp[i] = 1;

            short currentMax = 0;

            for (short j = (short)(i-1); j>=0; j--) {

                short prev = dp[j];


                if (c[i] == c[j]) dp[j] = (short)(currentMax+2);

                currentMax = (short) Math.max(prev, currentMax);
            }
        }
```

```
        for (int n : dp) max = (short) Math.max(max, n);

        return max;
    }
}
```