Dynamic Programming | Set 12 (Longest Palindromic Subsequence)

Given a sequence, find the length of the longest palindromic subsequence in it. For example, if the given sequence is “BBABCBCAB”, then the output should be 7 as “BABCBAB” is the longest palindromic subseuqnce 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 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)).

Following is a general recursive solution with all cases handled.

```
// Everay single character is a palindrom 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)}
```

Google™ Custom Search

GeeksforGeeks

53,524 people like GeeksforGeeks.

Interview Experiences
Advanced Data Structures
Dynamic Programming
Greedy Algorithms
Backtracking
Pattern Searching
Divide & Conquer
Mathematical Algorithms
Recursion

```
// 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 simple recursive implementation of the LPS problem. The implementation simply follows the recursive structure mentioned above.

```
#include<stdio.h>
#include<string.h>

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

// Returns the length of the longest palindromic subsequence in seq
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 functions */
int main()
{
    char seq[] = "GEEKSFORGEEKS";
    int n = strlen(seq);
    printf ("The lnegth of the LPS is %d", lps(seq, 0, n-1));
    getchar();
    return 0;
```

Popular Posts
All permutations of a given string
Memory Layout of C Programs
Understanding “extern” keyword in C
Median of two sorted arrays
Tree traversal without recursion and without stack!
Structure Member Alignment, Padding and Data Packing
Intersection point of two Linked Lists
Lowest Common Ancestor in a BST.
Check if a binary tree is BST or not
Sorted Linked List to Balanced BST

```
}
```

Output:

```
The lnegth of the LPS is 5
```

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

```
               L(0, 5)
             /        \
            /          \
        L(1,5)          L(0,4)
       /    \            /    \
      /      \          /      \
  L(2,5)    L(1,4)  L(1,4)  L(0,3)
```

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 same suproblems are called again, this problem has Overlapping Subprolems 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 bottom up manner.

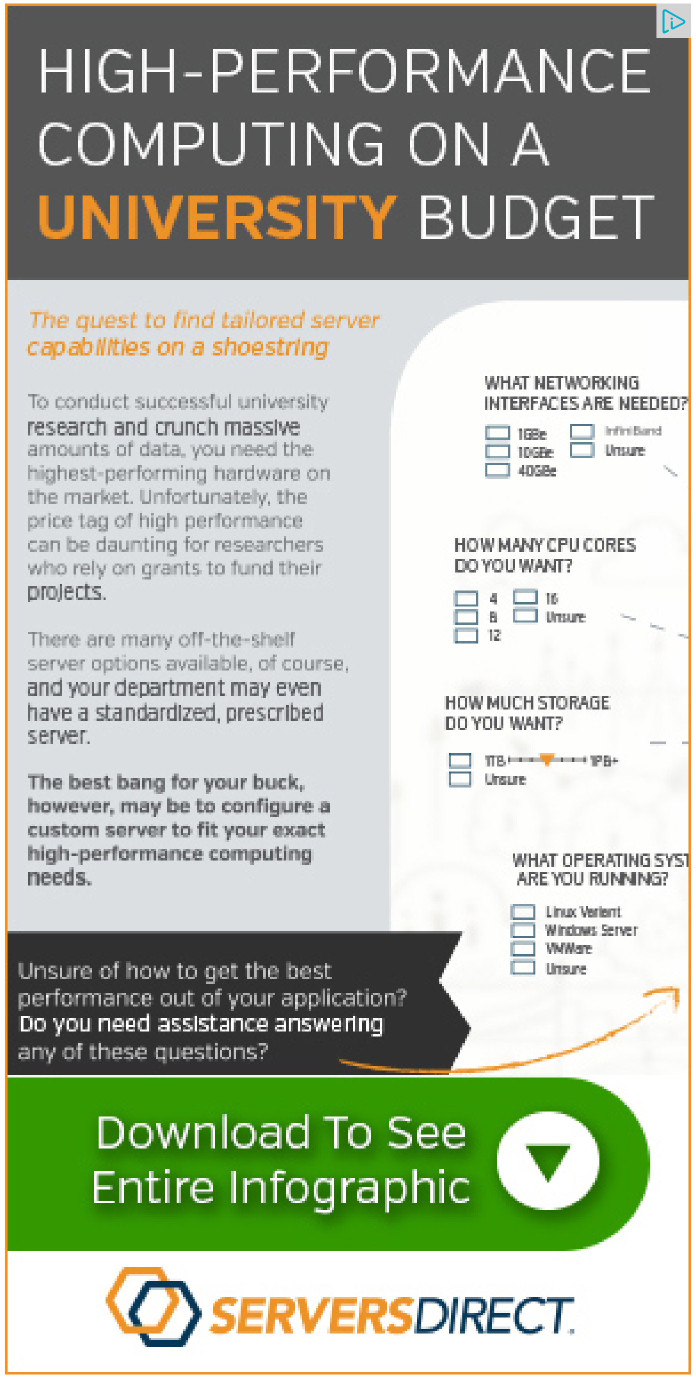
Dynamic Programming Solution

```
#include<stdio.h>
#include<string.h>

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

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


   // Strings of length 1 are palindrome of lentgh 1
```


HIGH-PERFORMANCE COMPUTING ON A UNIVERSITY BUDGET
The quest to find tailored server capabilities on a shoestring
To conduct successful university research and crunch massive amounts of data, you need the highest-performing hardware on the market. Unfortunately, the price tag of high performance can be daunting for researchers who rely on grants to fund their projects.
There are many off-the-shelf server options available, of course, and your department may even have a standardized, prescribed server.
The best bang for your buck, however, may be to configure a custom server to fit your exact high-performance computing needs.
WHAT NETWORKING INTERFACES ARE NEEDED?
HOW MANY CPU CORES DO YOU WANT?
HOW MUCH STORAGE DO YOU WANT?
WHAT OPERATING SYSTEM ARE YOU RUNNING?
Unsure of how to get the best performance out of your application? Do you need assistance answering any of these questions?
Download To See Entire Infographic
SERVERSDIRECT

```
    for (i = 0; i < n; i++)
       L[i][i] = 1;

     // Build the table. Note that the lower diagonal values of table a
     // useless and not filled in the process. The values are filled in
     // manner similar to Matrix Chain Multiplication DP solution (See
     // http://www.geeksforgeeks.org/archives/15553). cl is length of
     // substring
     for (cl=2; cl<=n; cl++)
     {
         for (i=0; i<n-cl+1; i++)
         {
             j = i+cl-1;
             if (str[i] == str[j] && cl == 2)
                L[i][j] = 2;
             else if (str[i] == str[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 */
int main()
{
    char seq[] = "GEEKS FOR GEEKS";
    int n = strlen(seq);
    printf ("The lnegth of the LPS is %d", lps(seq));
    getchar();
    return 0;
}
```

Output:

```
The lnegth 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.

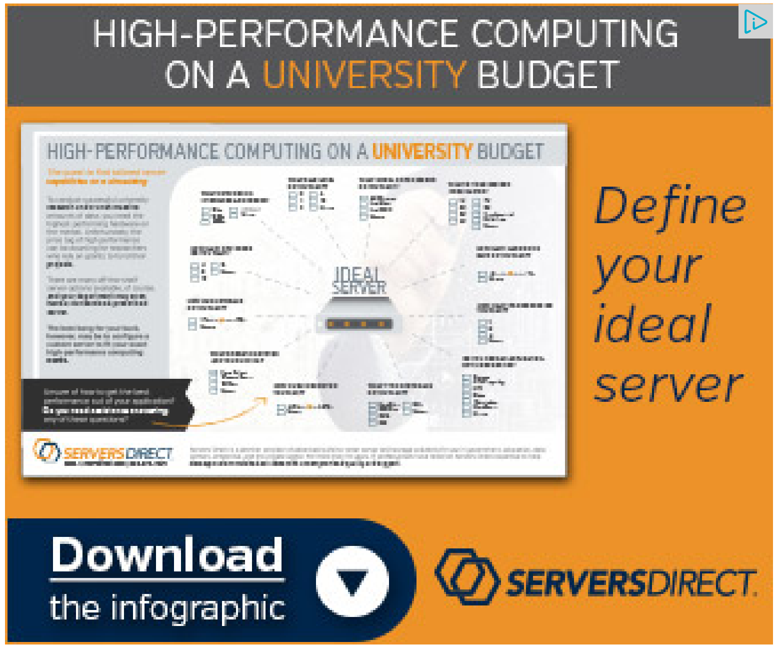
This problem is close to the Longest Common Subsequence (LCS) problem. In fact, we can use LCS as a subroutine to solve this problem. Following is the two step solution that uses LCS.
1) Reverse the given sequence and store the reverse in another array say rev[0..n-1]

2) LCS of the given sequence and rev[] will be the longest palindromic sequence.
This solution is also a O(n^2) solution.

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

References:

http://users.eecs.northwestern.edu/~dda902/336/hw6-sol.pdf



Related Tpoics:

- Backtracking | Set 8 (Solving Cryptarithmetic Puzzles)
- Tail Recursion
- Find if two rectangles overlap
- Analysis of Algorithm | Set 4 (Solving Recurrences)
- Print all possible paths from top left to bottom right of a mXn matrix
- Generate all unique partitions of an integer
- Russian Peasant Multiplication
- Closest Pair of Points | O(nlogn) Implementation

7 Tweet 1 0

Writing code in comment? Please use **ideone.com** and share the link here.