Longest Palindromic Subsequence

Given a string s, find the longest palindromic subsequence's length in s. You may assume that the maximum length of s is 1000.

Example 1:
Input:

```
"bbbab"
```

Output:

```
4
```

One possible longest palindromic subsequence is "bbbb".
Example 2:
Input:

```
"cbbd"
```

Output:

```
2
```

One possible longest palindromic subsequence is "bb".

Solution 1

`dp[i][j]` : the longest palindromic subsequence's length of substring(i, j)
`State transition` :
`dp[i][j] = dp[i+1][j-1] + 2` if s.charAt(i) == s.charAt(j)
otherwise, `dp[i][j] = Math.max(dp[i+1][j], dp[i][j-1])`
`Initialization` : `dp[i][i] = 1`

```
public class Solution {
    public int longestPalindromeSubseq(String s) {
        int[][] dp = new int[s.length()][s.length()];

        for (int i = s.length() - 1; i >= 0; i--) {
            dp[i][i] = 1;
            for (int j = i+1; j < s.length(); 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][s.length()-1];
    }
}
```

Top bottom recursive method with memoization

```
public class Solution {
    public int longestPalindromeSubseq(String s) {
        return helper(s, 0, s.length() - 1, new Integer[s.length()][s.length()]);
    }

    private int helper(String s, int i, int j, Integer[][] memo) {
        if (memo[i][j] != null) {
            return memo[i][j];
        }
        if (i > j)      return 0;
        if (i == j)     return 1;

        if (s.charAt(i) == s.charAt(j)) {
            memo[i][j] = helper(s, i + 1, j - 1, memo) + 2;
        } else {
            memo[i][j] = Math.max(helper(s, i + 1, j, memo), helper(s, i, j - 1,
memo));
        }
        return memo[i][j];
    }
}
```

written by tankztc original link here

Solution 2

Idea:
dp[i][j] = longest palindrome subsequence of s[i to j].
If s[i] == s[j], dp[i][j] = 2 + dp[i+1][j - 1]
Else, dp[i][j] = max(dp[i+1][j], dp[i][j-1])

Rolling array O(2n) space

```
class Solution(object):
    def longestPalindromeSubseq(self, s):
        """
        :type s: str
        :rtype: int
        """
        n = len(s)
        dp = [[1] * 2 for _ in range(n)]
        for j in xrange(1, len(s)):
            for i in reversed(xrange(0, j)):
                if s[i] == s[j]:
                    dp[i][j%2] = 2 + dp[i + 1][(j - 1)%2] if i + 1 <= j - 1 else
2
                else:
                    dp[i][j%2] = max(dp[i + 1][j%2], dp[i][(j - 1)%2])
        return dp[0][(n-1)%2]
```

Further improve space to O(n)

```
class Solution(object):
    def longestPalindromeSubseq(self, s):
        """
        :type s: str
        :rtype: int
        """
        n = len(s)
        dp = [1] * n
        for j in xrange(1, len(s)):
            pre = dp[j]
            for i in reversed(xrange(0, j)):
                tmp = dp[i]
                if s[i] == s[j]:
                    dp[i] = 2 + pre if i + 1 <= j - 1 else 2
                else:
                    dp[i] = max(dp[i + 1], dp[i])
                pre = tmp
        return dp[0]
```

written by jedihy original link here

Solution 3

1. O(2^n) Brute force. If the two ends of a string are the same, then they must be included in the longest palindrome subsequence. Otherwise, both ends cannot be included in the longest palindrome subsequence.

```
    int longestPalindromeSubseq(string s) {
        return longestPalindromeSubseq(0,s.size()-1,s);
    }
    int longestPalindromeSubseq(int l, int r, string &s) {
        if(l==r) return 1;
        if(l>r) return 0;  //happens after "aa"
        return s[l]==s[r] ? 2 + longestPalindromeSubseq(l+1,r-1, s) :
            max(longestPalindromeSubseq(l+1,r, s),longestPalindromeSubseq(l,r-1,
s));
    }
```

2. O(n^2) Memoization

```
    int longestPalindromeSubseq(string s) {
        int n = s.size();
        vector<vector<int>> mem(n,vector<int>(n));
        return longestPalindromeSubseq(0,n-1, s,mem);
    }
    int longestPalindromeSubseq(int l, int r, string &s, vector<vector<int>>& mem)
{
        if(l==r) return 1;
        if(l>r) return 0;
        if(mem[l][r]) return mem[l][r];
        return mem[l][r] = s[l]==s[r] ? 2 + longestPalindromeSubseq(l+1,r-1, s,me
m) :
            max(longestPalindromeSubseq(l+1,r, s,mem),longestPalindromeSubseq(l,r
-1, s,mem));

    }
```

3. O(n^2) dp

```
    int longestPalindromeSubseq(string s) {
        int n = s.size();
        vector<vector<int>> dp(n+1,vector<int>(n));
        for(int i=0;i<n;i++) dp[1][i]=1;
        for(int i=2;i<=n;i++) //length
            for(int j=0;j<n-i+1;j++) {//start index
                dp[i][j] = s[j]==s[i+j-1]?2+dp[i-2][j+1]:max(dp[i-1][j],dp[i-1][j
+1]);
        return dp[n][0];
    }
```

4. O(n^2) time, O(n) space dp. In #3, the current row is computed from the previous 2 rows only. So we don't need to keep all the rows.

```
    int longestPalindromeSubseq(string s) {
        int n = s.size();
        vector<int> v0(n), v1(n,1), v(n), *i_2=&v0, *i_1=&v1, *i_=&v;
        for(int i=2;i<=n;i++) {//length
            for(int j=0;j<n-i+1;j++)//start index
                i_->at(j) = s[j]==s[i+j-1]?2+i_2->at(j+1):max(i_1->at(j),i_1->at(
j+1));
            swap(i_1,i_2);
            swap(i_1,i_); //rotate i_2, i_1, i_
        }
        return i_1->at(0);
    }
```

written by yu6 original link here

From Leetcoder.