Longest Valid Parentheses

Given a string containing just the characters `'('` and `')'`, find the length of the longest valid (well-formed) parentheses substring.

For `"(()"`, the longest valid parentheses substring is `"()"`, which has length = 2.

Another example is `")()())"`, where the longest valid parentheses substring is `"()()"`, which has length = 4.

Solution 1

```
class Solution {
public:
    int longestValidParentheses(string s) {
        int n = s.length(), longest = 0;
        stack<int> st;
        for (int i = 0; i < n; i++) {
            if (s[i] == '(') st.push(i);
            else {
                if (!st.empty()) {
                    if (s[st.top()] == '(') st.pop();
                    else st.push(i);
                }
                else st.push(i);
            }
        }
        if (st.empty()) longest = n;
        else {
            int a = n, b = 0;
            while (!st.empty()) {
                b = st.top(); st.pop();
                longest = max(longest, a-b-1);
                a = b;
            }
            longest = max(longest, a);
        }
        return longest;
    }
};
```

The workflow of the solution is as below.

1. Scan the string from beginning to end.
2. If current character is '(', push its index to the stack. If current character is ')' and the character at the index of the top of stack is '(', we just find a matching pair so pop from the stack. Otherwise, we push the index of ')' to the stack.
3. After the scan is done, the stack will only contain the indices of characters which cannot be matched. Then let's use the opposite side - substring between adjacent indices should be valid parentheses.
4. If the stack is empty, the whole input string is valid. Otherwise, we can scan the stack to get longest valid substring as described in step 3.

written by cjxxm original link here

Solution 2

My solution uses DP. The main idea is as follows: I construct a array**longest[]**, for any longest[i], it stores the longest length of valid parentheses which is end at i.
And the DP idea is :
If s[i] is '(', set longest[i] to 0,because any string end with '(' cannot be a valid one.
Else if s[i] is ')'
    If s[i-1] is '(', longest[i] = longest[i-2] + 2
    Else if s[i-1] is ')'**and s[i-longest[i-1]-1] == '('**, longest[i] = longest[i-1] + 2 + longest[i-longest[i-1]-2]
For example, input "()(())", at i = 5, longest array is [0,2,0,0,2,0], longest[5] = longest[4] + 2 + longest[1] = 6.

```
   int longestValidParentheses(string s) {
            if(s.length() <= 1) return 0;
            int curMax = 0;
            vector<int> longest(s.size(),0);
            for(int i=1; i < s.length(); i++){
                if(s[i] == ')'){
                    if(s[i-1] == '('){
                        longest[i] = (i-2) >= 0 ? (longest[i-2] + 2) : 2;
                        curMax = max(longest[i],curMax);
                    }
                    else{ // if s[i-1] == ')', combine the previous length.
                        if(i-longest[i-1]-1 >= 0 && s[i-longest[i-1]-1] == '('){
                            longest[i] = longest[i-1] + 2 + ((i-longest[i-1]-2 >=
0)?longest[i-longest[i-1]-2]:0);
                            curMax = max(longest[i],curMax);
                        }
                    }
                }
                //else if s[i] == '(', skip it, because longest[i] must be 0
            }
            return curMax;
        }
```

Updated: thanks to **Philip0116**, I have a more concise solution(though this is not as readable as the above one, but concise):

```
int longestValidParentheses(string s) {
        if(s.length() <= 1) return 0;
        int curMax = 0;
        vector<int> longest(s.size(),0);
        for(int i=1; i < s.length(); i++){
            if(s[i] == ')' && i-longest[i-1]-1 >= 0 && s[i-longest[i-1]-1] == '('
){
                    longest[i] = longest[i-1] + 2 + ((i-longest[i-1]-2 >= 0)?long
est[i-longest[i-1]-2]:0);
                    curMax = max(longest[i],curMax);
            }
        }
        return curMax;
    }
```

written by jerryrcwong original link here

Solution 3

```
public class Solution {
public int longestValidParentheses(String s) {
    Stack<Integer> stack = new Stack<Integer>();
    int max=0;
    int left = -1;
    for(int j=0;j<s.length();j++){
        if(s.charAt(j)=='(') stack.push(j);
        else {
            if (stack.isEmpty()) left=j;
            else{
                stack.pop();
                if(stack.isEmpty()) max=Math.max(max,j-left);
                else max=Math.max(max,j-stack.peek());
               }
            }
        }
    return max;
}
```

}

written by jmnjmnjmn original link here