[Leetcode] Longest Valid Parentheses 最长有效括号对

java 算法 leetcode **ethannnli** 2015年08月23日发布

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.

栈法 Stack

复杂度

时间 O(N) 空间 O(N)

思路

用Stack的方法本质上和Valid Parentheses是一样的，一个右括号能消去Stack顶上的一个左括号。不同的是，为了能够计算括号对的长度我们还需要记录括号们的下标。这样在弹出一个左括号后，我们可以根据当前坐标减去栈中上一个（也就是Pop过后的Top元素）的坐标来得到该有效括号对的长度。

代码

```
                //如果当前栈顶是左括号，则消去并计算长度
                 if(!stk.isEmpty() && stk.peek().symb=='('){
                     int curLen = 0;
                     stk.pop();
                     if(stk.isEmpty()){
                         curLen = i + 1;
                     } else {
                         curLen = i - stk.peek().indx;
                     }
                     maxLen = Math.max(maxLen, curLen);
                 } else {
                //如果栈顶是右括号或者是空栈，则将右括号也push进栈，它的坐标将方便之后计算长度
                     stk.push(new Parenthese(i, ')'));
                 }
             }
         }
         return maxLen;
     }

     public class Parenthese {
         int indx;
         char symb;
         public Parenthese (int i, char s){
             this.indx = i;
             this.symb = s;
         }
     }
 }
```

动态规划法 Dynamic Programming

复杂度

时间 O(N) 空间 O(N)

思路

动态规划法将大问题化为小问题，我们不一定要一下子计算出整个字符串中最长括号对，我们可以先从后向前，一点一点计算。假设d[i]是从下标i开始到字符串结尾最长括号对长度，s[i]是字符串下标为i的括号。如果s[i-1]是左括号，如果i + d[i] + 1是右括号的话，那d[i-1] = d[i] + 1。如果不是则为0。如果s[i-1]是右括号，因为不可能有右括号开头的括号对，所以d[i-1] = 0。

代码

```
public class Solution {
    public int longestValidParentheses(String s) {
        int[] dp = new int[s.length()];
        int maxLen = 0;
        for(int i = s.length()-2; i >=0; i--){
            if(s.charAt(i)=='('){
                int end = i + dp[i+1] + 1;
                if(end < s.length() && s.charAt(end)==')'){
                    dp[i] = dp[i+1] + 2;
                    if(end + 1 < s.length()){
                        dp[i] += dp[end + 1];
                    }
                }
            }
            maxLen = Math.max(maxLen, dp[i]);
        }
        return maxLen;
    }
}
```

后续 Follow Up

Q：能否不用额外空间求解？
A：可以，但是要提高时间复杂度。比如(((())，先遍历一遍将所有的()替换成00，得到((0000)，再遍历一遍，替换所有的(00...00)这种形式字符串为000...000，这里我们得到(000000，直到遍历完无法替换更多括号为之。如果所有符号都是0，说明是有效的。这样的时间复杂度是O(N)。

2015年08月23日发布

你可能感兴趣的文章

字符串处理文章outline 821 浏览

leetcode 算法解析（一）：260. Single Number III 390 浏览

Maximal Rectangle@LeetCode 609 浏览

讨论区

使用评论询问更多信息或提出修改意见，请不要在评论里回答问题

提交评论

评论支持部分 Markdown 语法： **bold** _italic_ [link](http://example.com) > 引用 `code` - 列表 。
同时，被你 @ 的用户也会收到通知

本文隶属于专栏

Ethan Li 的技术专栏

全栈工程师的修炼旅途

ethannnli
作者

关注专栏

系列文章

[Leetcode] Walls and Gates 墙与门 2.4k 浏览

[Leetcode] Binary Tree Longest Consecutive Sequence 二叉搜索树最长序列 1 收藏， 2.4k 浏览

相关收藏夹

换一组

android测试
5 个条目 | 1 人关注

Java/JVM
17 个条目 | 1 人关注

spring学习
4 个条目 | 1 人关注

分享扩散：