算法——括号匹配问题

原创 王雅雯 LOA算法学习笔记 2021-01-31 23:17

括号匹配问题是算法应用中较为经典的问题，本篇介绍了括号匹配相关的两个基本应用——**括号合法性判定**和**最长有效匹配括号**。

01 括号合法性判定

输入一个字符串，包含[](){}六种符号，判断这个字符串的组成的括号是否合法
leetcode相关题：20 Valid Parentheses

1.1 一种括号情况

一种括号的情况指字符串中只有圆括号，如果想让括号字符串合法，那么必须做到：每个有括号)的左边必须有一个左括号(和它匹配，根据这个思路，可以写出算法：

```
bool isValid(string str){
  int left=0;
  for (char c:str){
    if (c=='(')
      left++;
    else
      left--;
    if (left<0)
      return false;
  }
  return left==0;
}
```

1.2 多种括号情况

根据上个方法的思路，对于多种括号的情况可以尝试增加多个if分支，设置left1、left2、left3分别处理每种括号，但是这种思路仍是不正确的，例如[(])

因此，再多种括号的情况的处理中使用**栈**，定义一个名为left的栈代替之前思路中的left变量，遇到左括号就入栈，遇到有括号就去栈中寻找最近的左括号，看是否匹配。

```
bool isValid(string str){
  stack<char> left;
  for (char c : str){
    if(c=='('||c=='{'||c=='[')
      left.push(c);
    else{
      if(!left.empty()&&leftOf(c)==left.top())
        left.pop();
      else
        return false;
```

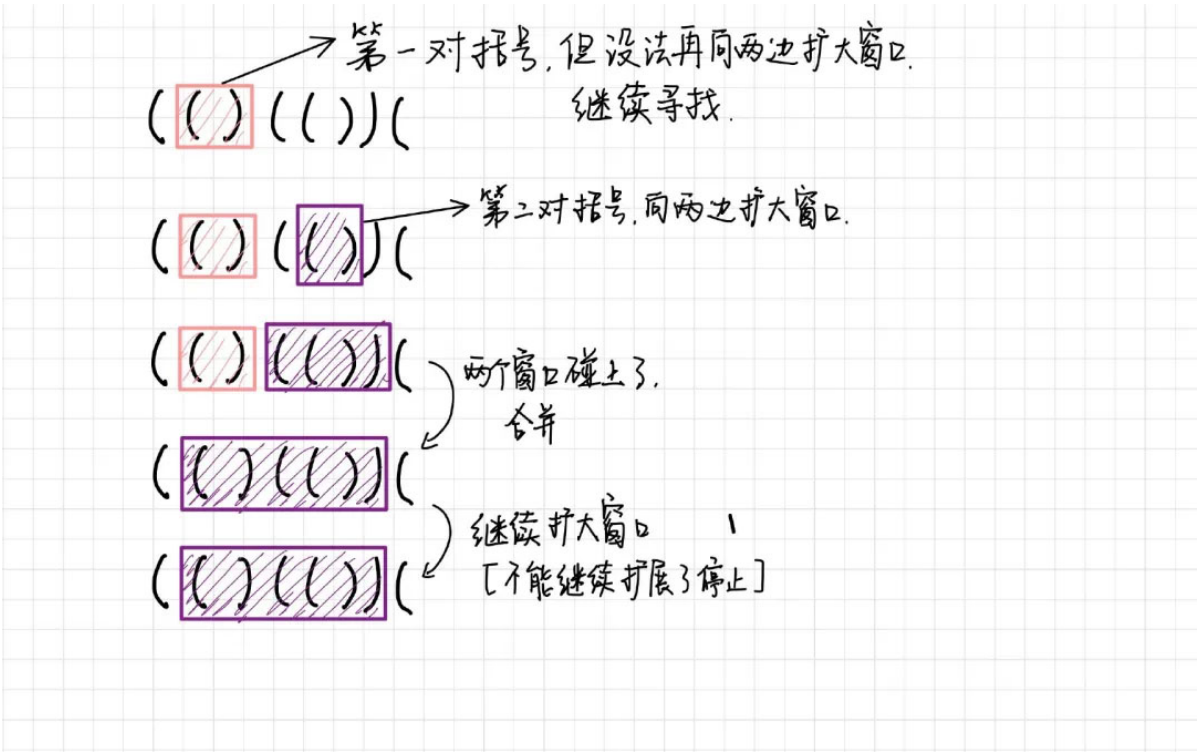
```
    }
  }
  return left.empty();
}
```

02 最长有效匹配括号

leetcode题：32 Longest Valid Parentheses
问题描述：给定字符串，只包含()，找出最长的包含有效括号的子串的长度。
示例:
输入: ")()())"
输出: 4
解释: 最长有效括号子串为 "()()"

2.1 方法一：滑动窗口

算法思路:
最短的有效括号是一对括号()，在s中找到这样一对括号，并把这对括号作为一个滑动窗口的中心，分别向左右两侧扩大滑动窗口，直到滑动窗口不再扩大时，记录当前窗口的左右边界，从右边界开始重复查找，直到字符串遍历结束。在扩大滑动窗口时，如果碰到了另一个窗口边界，则应将两个窗口合并，作为新的大窗口。
图解:



伪代码:

```
int LongestValidParentheses(string str){
  max=0;
  for(int i=0;i<str.size();++i){
    //获得()的下标索引
    l=str.indexOf('()');
    if(l==-1)
```

```
        break;
      r=l+1;
      //以此()为中心向左右两边扩展窗口
      [l,r]=expand(str,l,r);
      //扩展窗口到最大
      //当当前窗口的左边界刚好挨着前一个窗口的右边界，则合并两个窗口，并继续扩展
      while(l-1 in map){
        [l,r]=expand(s,map[l-1],r);
      }
      //记录当前窗口的左右边界
      map[r]=l;
      ans=max(max,r-l+1);
    }
    return ans;
  }
```

时间复杂度：O(n)

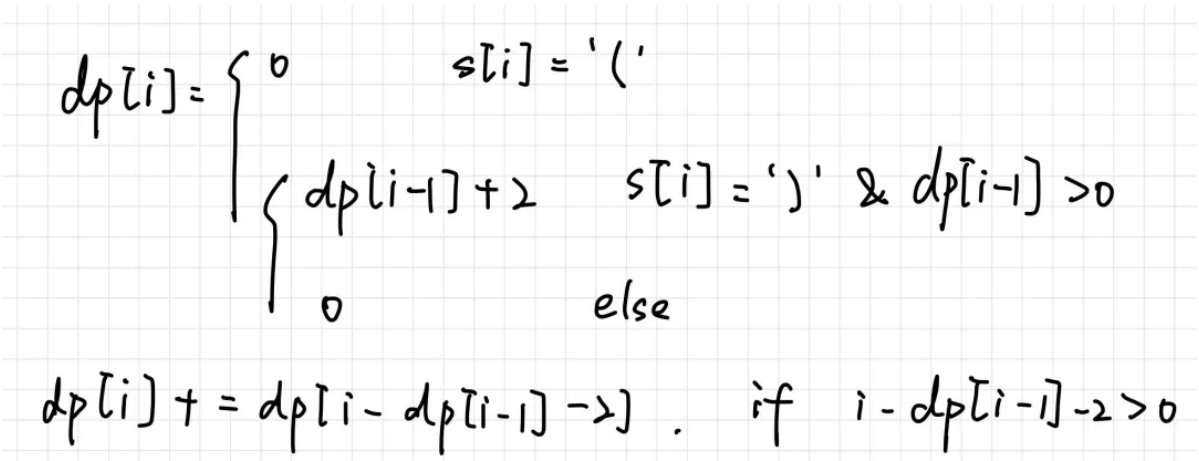
2.2 方法二：动态规划

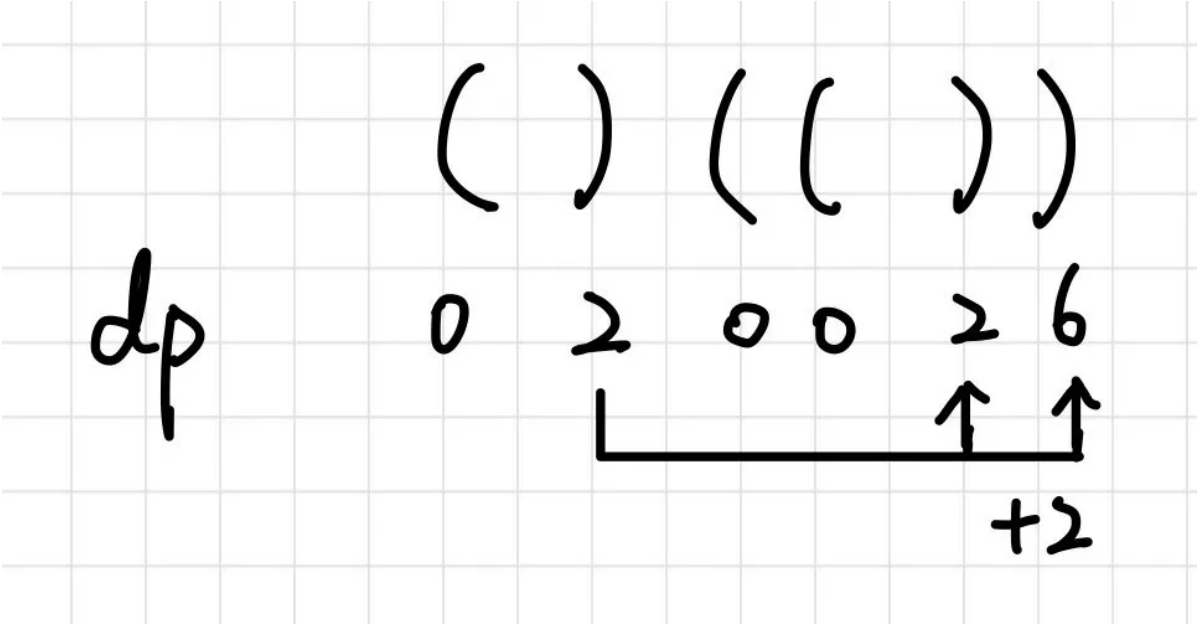
算法思路：

用动态规划数组来记录：当前坐标为结尾的有效括号字符串的长度

- 当前字符是'('，此时dp[i]=0
- 当前字符时')'
  - 若前一个字符是'('，则他们可以组成一对儿，dp[i]至少是2
  - 若前一个字符是')'，则检查坐标i-1的状态，若dp[i-1]=0，则dp[i]=0；若dp[i-1]>0，则i之前有一段有效括号字符串，只需要判断这段字符串前的字符是不是'('，若是，则dp[i]=dp[i-1]+2，否则dp[i]=0

图解：





伪代码：

```
int LongestValidParentheses(string s){
  //动态数组初始化为0
  dp[n]={0}
  for (int i =1;i<s.size();++i){
    if (s[i]==')'){
      //如果遇到)则到前面去找和他匹配的(,如果存在则在有效长度上加2
      if(i-dp[i-1]-1>=0&&s[i-dp[i-1]-1]=='('){
        dp[i]=dp[i-1]+2;
        //如果(前面还有有效长度则继续加上
        if(i-dp[i-1]-2>0){
          dp[i]+=dp[i-dp[i-1]-2];
        }
      }
    }
  }
  //返回dp中的最大值
  return max(dp[...],0)
}
```

时间复杂度：O(n)

2.3 方法三：栈

使用栈的思路和括号的合法性判定中的方法相似，在该题中，栈中存储的不是括号，而是括号的下标。在遍历过程中，如果遇到')'，就从栈中弹出和它对应的'('的下边，用valid数组对下标的位置进行标记，最后返回的结果就是valid数组中最长的连续的1序列。

伪代码：

```
int LongestValidParentheses(string s){
  stack<char> stacks;
  valid[n]={0};
  for (int i=0;i<s.size();++i){
```

```
    if(s[i]=='(')
      stacks.push(i);
    if(s[i]==')'&&stacks.length>0){
      valid[i]=1;
      valid[stacks.pop()]=1;
    }
  }
  ans=0;
  ans=valid最长的连续的1的序列;
  return ans;
}
```

时间复杂度：O(n)