# 题目

给你一个只包含 '(' 和 ')' 的字符串，找出最长有效（格式正确且连续）括号子串的长度。

示例 1：

输入：s = "(()"

输出：2

解释：最长有效括号子串是 "()"

示例 2：

输入：s = ")()())"

输出：4

解释：最长有效括号子串是 "()()"

示例 3：

输入：s = ""

输出：0

提示：

0 <= s.length <= 3 \* 104

s[i] 为 '(' 或 ')'

# 分析

## 方法一：动态规划

## 方法二：栈

**思路：**

撇开方法一提及的动态规划方法，相信大多数人对于这题的第一直觉是找到每个可能的子串后判断它的有效性，但这样的时间复杂度会达到 O(n^3)，无法通过所有测试用例。但是通过栈，我们可以在遍历给定字符串的过程中去判断到目前为止扫描的子串的有效性，同时能得到最长有效括号的长度。

具体做法是我们始终保持栈底元素为当前已经遍历过的元素中「最后一个**没有被匹配**的右括号的下标」，这样的做法主要是考虑了边界条件的处理，栈里其他元素维护左括号的下标：

1、对于遇到的每个‘(’，我们将它的下标放入栈中

2、对于遇到的每个‘)’，我们先弹出栈顶元素表示匹配了当前右括号：

1）如果栈为空，说明当前的右括号为没有被匹配的右括号，我们将其下标放入栈中来更新我们之前提到的「最后一个没有被匹配的右括号的下标」

2）如果栈不为空，当前右括号的下标减去栈顶元素即为「以该右括号为结尾的最长有效括号的长度」

我们从前往后遍历字符串并更新答案即可。

需要注意的是，如果一开始栈为空，第一个字符为左括号的时候我们会将其放入栈中，这样就不满足提及的「最后一个没有被匹配的右括号的下标」，为了保持统一，我们在一开始的时候往栈中放入一个值为-1的元素。

**代码：**

class Solution {

public:

int longestValidParentheses(string s) {

int maxans = 0;

stack<int> stk;

stk.push(-1);

for (int i = 0; i < s.length(); i++) {

if (s[i] == '(') {

stk.push(i);

} else {

stk.pop(); // 如果是')'，则表示匹配了，直接弹出

if (stk.empty()) {

stk.push(i);

} else {

maxans = max(maxans, i - stk.top());

}

}

}

return maxans;

}

};

**复杂度分析：**

时间复杂度：O(n)，n是给定字符串的长度。我们只需要遍历字符串一次即可。

空间复杂度：O(n)。栈的大小在最坏情况下会达到n，因此空间复杂度为O(n)。