

如何解决括号相关的问题

 Stars 107k  B站 @labuladong 配套PDF和插件 下载 打卡挑战 报名 精品课程 查看






微信搜一搜

Q labuladong公众号

通知： 数据结构精品课 V1.6 持续更新中， 第八期打卡挑战 开始报名。

读完本文，你不仅学会了算法套路，还可以顺便解决如下题目：

牛客	LeetCode	力扣	难度
-	20. Valid Parentheses	20. 有效的括号	
-	921. Minimum Add to Make Parentheses Valid	921. 使括号有效的最少添加	
-	1541. Minimum Insertions to Balance a Parentheses String	1541. 平衡括号字符串的最少插入次数	

判断有效括号串

对括号的有效性判断多次在笔试中出现，现实中也很常见，比如说我们写的代码，编辑器会检查括号是否正确闭合。而且我们的代码可能会包含三种括号 `[](){}` ，判断起来有一点难度。

来看一看力扣第 20 题「有效的括号」，输入一个字符串，其中包含 `[](){}` 六种括号，请你判断这个字符串组成的括号是否有效。

举几个例子：

Input: "()[]{}"

Output: true

Input: "([)]"

Output: false

Input: "{[]}"

Output: true

解决这个问题之前，我们先降低难度，思考一下，**如果只有一种括号** `()`，应该如何判断字符串组成的括号是否有效呢？

假设字符串中只有圆括号，如果能让括号字符串有效，那么必须做到：

每个右括号 `)` 的左边必须有一个左括号 `(` 和它匹配。

比如说字符串 `(()))((` 中，中间的两个右括号**左边**就没有左括号匹配，所以这个括号组合是无效的。

那么根据这个思路，我们可以写出算法：

```
bool isValid(string str) {  
    // 待匹配的左括号数量  
    int left = 0;  
    for (int i = 0; i < str.size(); i++) {  
        if (s[i] == '(') {  
            left++;  
        } else {  
            // 遇到右括号  
            left--;  
        }  
  
        // 右括号太多  
        if (left == -1)  
            return false;  
    }  
    // 是否所有的左括号都被匹配了  
    return left == 0;  
}
```

如果只有圆括号，这样就能正确判断有效性。对于三种括号的情况，我一开始想模仿这个思路，定义三个变量 `left1`，`left2`，`left3` 分别处理每种括号，虽然要多写不少 `if else` 分支，但是似乎可以解决问题。

但实际上直接照搬这种思路是不行的，比如说只有一个括号的情况下 `((()))` 是有效的，但是多种括号的情况下，`[(())]` 显然是无效的。

仅仅记录每种左括号出现的次数已经不能做出正确判断了，我们要加大存储的信息量，可以利用栈来模仿类似的思路。栈是一种先进后出的数据结构，处理括号问题的时候尤其有用。

我们这道题就用一个名为 `left` 的栈代替之前思路中的 `left` 变量，**遇到左括号就入栈，遇到右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 { // 字符 c 是右括号
            if (!left.empty() && leftOf(c) == left.top())
                left.pop();
            else
                // 和最近的左括号不匹配
                return false;
        }
    }
    // 是否所有的左括号都被匹配了
    return left.empty();
}

char leftOf(char c) {
    if (c == '}') return '{';
    if (c == ')') return '(';
    return '[';
}
```

接下来讲另外两个常见的问题，如何通过最小的插入次数将括号变成有效的？

平衡括号串（一）

先来个简单的，力扣第 921 题「使括号有效的最少添加」：

给你输入一个字符串 `s`，你可以在其中的任意位置插入左括号 `(` 或者右括号 `)`，请问你最少需要几次插入才能使得 `s` 变成一个有效的括号串？

比如说输入 `s = "())(`，算法应该返回 2，因为我们至少需要插入两次把 `s` 变成 `"(())()"`，这样每个左括号都有一个右括号匹配，`s` 是一个有效的括号串。

这其实和前文的判断括号有效性非常类似，我们直接看代码：

```
int minAddToMakeValid(string s) {
    // res 记录插入次数
    int res = 0;
    // need 变量记录右括号的需求量
    int need = 0;

    for (int i = 0; i < s.size(); i++) {
        if (s[i] == '(') {
            // 对右括号的需求 + 1
            need++;
        }

        if (s[i] == ')') {
            // 对右括号的需求 - 1
            need--;

            if (need == -1) {
                need = 0;
                // 需插入一个左括号
                res++;
            }
        }
    }

    return res + need;
}
```

这段代码就是最终解法，核心思路是以左括号为基准，通过维护对右括号的需求数 `need`，来计算最小的插入次数。需要注意两个地方：

1、当 `need == -1` 的时候意味着什么？

因为只有遇到右括号 `)` 的时候才会 `need--`，`need == -1` 意味着右括号太多了，所以需要插入左括号。

比如说 `s = ")))"` 这种情况，需要插入 2 个左括号，使得 `s` 变成 `"()()"`，才是一个有效括号串。

2、算法为什么返回 `res + need`？

因为 `res` 记录的左括号的插入次数，`need` 记录了右括号的需求，当 for 循环结束后，若 `need` 不为 0，那么就意味着右括号还不够，需要插入。

比如说 `s = ")))("` 这种情况，插入 2 个左括号之后，还要再插入 1 个右括号，使得 `s` 变成 `"()()())"`，才是一个有效括号串。

以上就是这道题的思路，接下来我们看一道进阶题目，如果左右括号不是 1:1 配对，会出现什么问题呢？

平衡括号串（二）

这是力扣第 1541 题「平衡括号字符串的最少插入次数」：

现在假设 1 个左括号需要匹配 2 个右括号才叫做有效的括号组合，那么给你输入一个括号串 `s`，请问你如何计算使得 `s` 有效的最小插入次数呢？

核心思路还是和刚才一样，通过一个 `need` 变量记录对右括号的需求数，根据 `need` 的变化来判断是否需要插入。

第一步，我们按照刚才的思路正确维护 `need` 变量：

```
int minInsertions(string s) {
    // need 记录需右括号的需求量
    int res = 0, need = 0;

    for (int i = 0; i < s.size(); i++) {
        // 一个左括号对应两个右括号
        if (s[i] == '(') {
            need += 2;
        }

        if (s[i] == ')') {
            need--;
        }
    }

    return res + need;
}
```

```

    }
}

return res + need;
}

```

现在想一想，当 `need` 为什么值的时候，我们可以确定需要进行插入？

首先，类似第一题，当 `need == -1` 时，意味着我们遇到一个多余的右括号，显然需要插入一个左括号。

比如说当 `s = ")"`，我们肯定需要插入一个左括号让 `s = "()"`，但是由于一个左括号需要两个右括号，所以对右括号的需求量变为 1：

```

if (s[i] == ')') {
    need--;
    // 说明右括号太多了
    if (need == -1) {
        // 需要插入一个左括号
        res++;
        // 同时，对右括号的需求变为 1
        need = 1;
    }
}
}

```

另外，当遇到左括号时，若对右括号的需求量为奇数，需要插入 1 个右括号。因为一个左括号需要两个右括号嘛，右括号的需求必须是偶数，这一点也是本题的难点。

所以遇到左括号时要做如下判断：

```

if (s[i] == '(') {
    need += 2;
    if (need % 2 == 1) {
        // 插入一个右括号
        res++;
        // 对右括号的需求减一
        need--;
    }
}

```

综上，我们可以写出正确的代码：

```

int minInsertions(string s) {
    int res = 0, need = 0;

    for (int i = 0; i < s.size(); i++) {
        if (s[i] == '(') {
            need += 2;
            if (need % 2 == 1) {
                res++;
                need--;
            }
        }

        if (s[i] == ')') {
            need--;
            if (need == -1) {
                res++;
                need = 1;
            }
        }
    }

    return res + need;
}

```

综上，三道括号相关的问题就解决了，其实我们前文 [有效括号生成算法](#) 也是括号相关的问题，但是使用的回溯算法技巧，和本文的几道题差别还是蛮大的，有兴趣的读者可以去看看。
