拉勾教育
—互联网人实战大学—

栈：删除最外层括号

简单/计数器、累加、栈

学习目标

拉勾教育
—互联网人实战大学—

了解算法题的解题思路

栈的特点

栈的应用

题目描述

给出一个非空有效字符串S，考虑将其进行原语化分解。

使得：S=P_1+P_2+...+P_k，其中P_i是有效括号字符串原语。

对S进行原语化分解，删除分解中每个原语字符串的最外层括号，返回S。

- 有效括号字符串：例如："()"，"(()())" 和 "(()(()))"
- 原语：如果有效字符串S非空，且不能将其拆分为 S =A+B，我们称其为原语（primitive），其中A和B都是非空有效括号字符串。

题目描述

示例一：

输入："(()())"
输出：""
解释：原语化分解得到 "()" + "()"， 删除每个部分中的最外层括号后得到 "" + "" = ""

示例二：

输入："(()())(())"
输出："()()()"
解释：原语化分解得到 "(()())" + "(())"， 删除每个部分中的最外层括号后得到 "()()" + "()" = "()()()"

示例三：

输入："(()())(())(()(()))"
输出："()()()()(())"
解释：原语化分解得到 "(()())" + "(())" + "(()(()))"，删除每个部分中的最外层括号后得到 "()()" + "()" + "()(())" = "()()()()(())"

一. Comprehend 理解题意

拉勾教育
—互联网人实战大学—

题目基本信息

- 有效括号字符串：括号必然成对出现
- 给定字符串 S 非空，但返回结果可能为空
- 原语化分解只需要进行一级
- 返回删除每个原语的最外层括号后的字符串

"((a)(b))((c))((d)((e)))"

"((a)(b))" "((c))" "((d) ((e)))"

"(a)(b)(c) (d)((e))"

附加信息

- S.length <= 10000
- S[i] 为 "(" 或 ")"

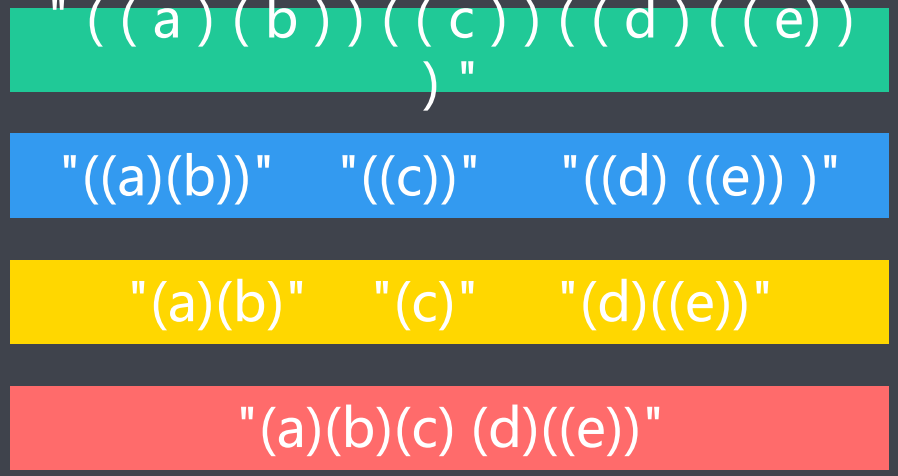
"()()()()(())"

一. Comprehend 理解题意

拉勾教育
—互联网人实战大学—

解法一：暴力解法

- 先分解成字符串原语
- 再删除每部分最外层括号
- 返回各部分合并后的结果

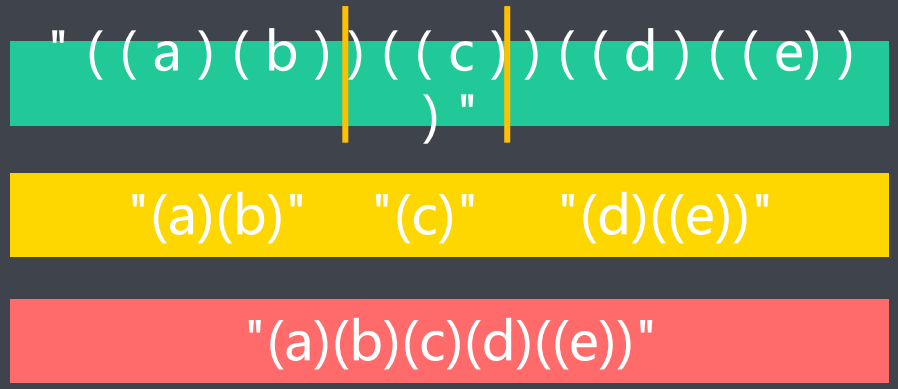


一. Comprehend 理解题意

拉勾教育
—互联网人实战大学—

解法二：优化解法

- 对原字符串进行原语识别
- 获取不包含最外层括号的子串
- 将各部分拼接返回



二. Choose 数据结构及算法思维选择

拉勾教育
—互联网人实战大学—

解法一：分解、删除再合并

- 数据结构：字符串、数组
- 算法思维：遍历、计数器、累加

解法二：定位、直取再合并

- 数据结构：字符串、数组
- 算法思维：遍历、计数器、累加

```
"((a)(b))((c))((d)((e)))"
```

三. Code 基本解法及编码实现

解法一：暴力解法思路分析

1. 定义容器存储原语子串

   new ArrayList<String>();

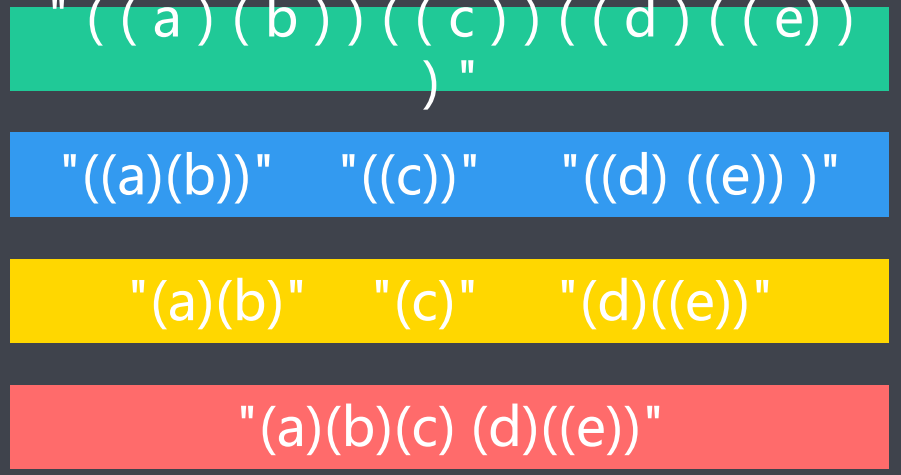
2. 定义左括号、右括号计数器：

   int left = 0, right = 0;

3. 遍历字符串，读取到括号时对应计数器自增

4. 检查是否到达原语结尾，截取原语子串并添加到容器中

5. 遍历容器，删除最外层括号后合并成新串



三. Code 基本解法及编码实现

解法一：边界和细节问题

1. 边界问题：

遍历字符串，注意索引越界：i < S.length()

截取原语字符串时，注意起止索引：[start, end)

"((a)(b))((c))((d)((e)))"

"((a)(b))"　"((c))"　"((d) ((e)))"

2. 细节问题：

需要记录上一次截取原语子串之后的位置

删除原语子串的最外层括号，其实就是重新截取

"(a)(b)"　"(c)"　"(d)((e))"

三. Code 基本解法及编码实现

拉勾教育
—互联网人实战大学—

```java
public String removeOuterParentheses(String S) {
    int len = S.length();
    // 1.定义容器存储原语子串
    List<String> list = new ArrayList<>();
    // 2.定义左括号、右括号计数器
    int left = 0, right = 0, lastOpr = 0;
    // 3.遍历字符串，读取到括号时对应计数器自增
    for (int i = 0; i < len; i++) {
        char c = S.charAt(i);
        if (c == '(') {
            left++;
        } else if (c == ')') {
            right++;
        }
        // 4.检查是否到达某个原语结尾，截取原语子串添加到容器
        if (left == right) {
            list.add(S.substring(lastOpr, i + 1));
            lastOpr = i + 1;
        }
    }
    // 5.遍历容器中的原语子串，删除最外层后合并成新串
    StringBuilder sb = new StringBuilder();
    for (String s : list) {
        sb.append(s.substring(1, s.length() - 1));
    }
    return sb.toString();
}
```

时间复杂度：O(n)

- 遍历整个字符串：O(n)
- 遍历原语子串：O(n)

空间复杂度：O(n)

- 定义容器存储原语子串：O(n)
- 定义计数器：O(1)
- 生成原语子串：O(n)
- 生成新的字符串：O(n)

执行耗时:6 ms,击败了55.89% 的Java用户
内存消耗:38.6 MB,击败了81.09% 的Java用户

三. Code 基本解法及编码实现

拉勾教育
—互联网人实战大学—

解法二：优化解法思路分析

1. 定义容器存储删除外层括号后的原语子串

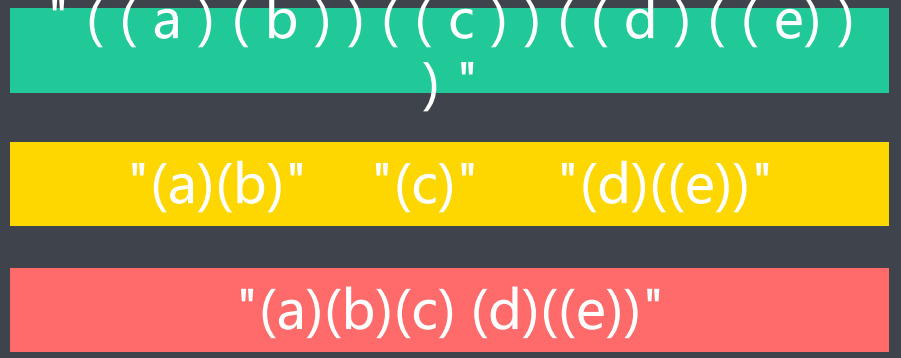
   new StringBuilder();

2. 定义左括号、右括号计数器：

   int left = 0, right = 0;

3. 遍历字符串，读取到括号时对应计数器自增

4. 检查是否到达原语结尾，截取不包含最外层的原语子串并拼接到容器中



三. Code 基本解法及编码实现

拉勾教育
—互联网人实战大学—

```java
public String removeOuterParentheses(String S) {
    int len = S.length();
    // 1.定义容器存储删除外层括号后的原语子串
    StringBuilder sb = new StringBuilder();
    // 2.定义左括号、右括号计数器
    int left = 0, right = 0, lastOpr = 0;
    // 3.遍历字符串，读取到括号时对应计数器自增
    for (int i = 0; i < len; i++) {
        char c = S.charAt(i);
        if (c == '(') {
            left++;
        } else if (c == ')') {
            right++;
        }
        // 4.检查是否到达某个原语结尾，截取不包含最外层的原语子串添加到容器
        if (left == right) {
            sb.append(S.substring(++lastOpr, i));
            lastOpr = i + 1;
        }
    }
    return sb.toString();
}
```

时间复杂度：O(n)

- 遍历整个字符串：O(n)
- ~~遍历原语子串：O(n)~~

空间复杂度：O(n)

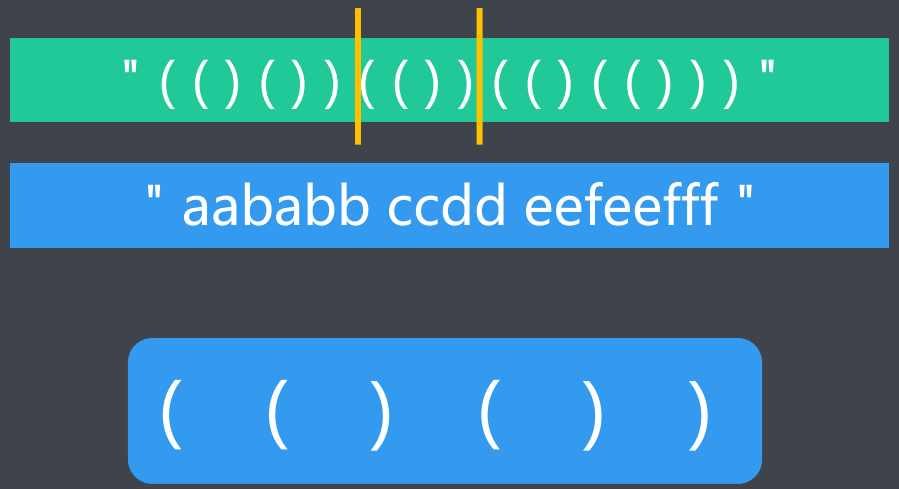
- ~~定义容器存储原语子串：O(n)~~
- 定义计数器：O(1)
- 生成原语子串：O(n)
- 生成新的字符串：O(n)

执行耗时:4 ms,击败了67.43% 的Java用户
内存消耗:38.5 MB,击败了91.37% 的Java用户

四. Consider 思考更优解

拉勾教育
—互联网人实战大学—

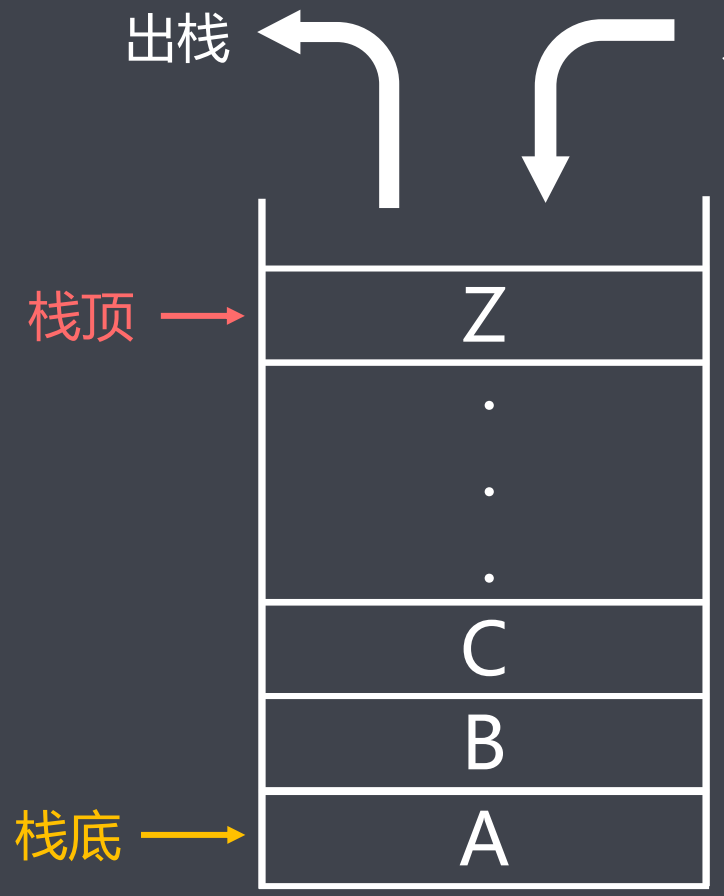
1. 剔除无效代码或优化空间消耗
   - 只需要一个计数器："("自增，")"自减
   - 如果字符串数据只是两两配对怎么处理?
   - 有没有一种支持后进先出的数据结构?
2. 寻找更好的算法思维
   - 借鉴其它算法



五. Code 最优解思路及编码实现

关键知识点：栈（Stack）

- 限定仅在表尾进行插入和删除操作的线性表
- 栈顶（Top）：操作数据的一端，即表尾
- 栈底（Bottom）：线性表的另一端，即表头
- 特点：LIFO（Last In First Out），后进先出
- 基本操作：
  - 进栈：push()，在栈顶插入元素，入栈、压栈
  - 出栈：pop()，从栈顶删除数据
  - 判断空：isEmpty()



栈的示意图

五. Code 最优解思路及编码实现

最优解：栈解法

1. 使用数组模拟一个栈，临时存储字符，替代计数器

   push(Character) ; pop(); isEmpty()

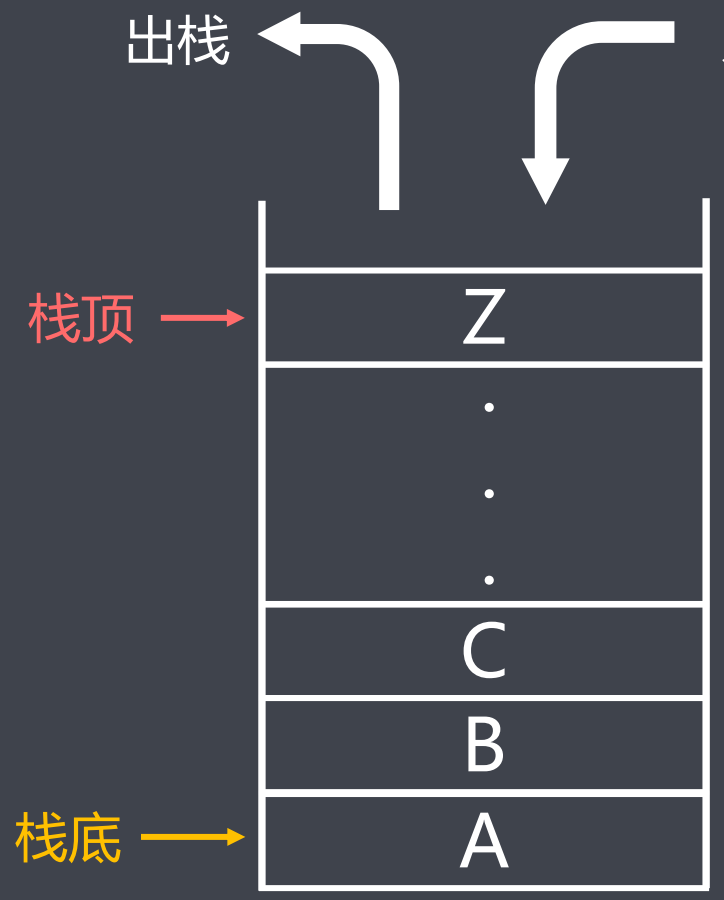
2. 遍历字符串，根据情况进行入栈/出栈操作

   读取到左括号，左括号入栈

   读取到右括号，左括号出栈

3. 判断栈是否为空，若为空，找到了一个完整的原语

4. 截取不含最外层括号的原语子串并进行拼接



栈的示意图

五. Code 最优解思路及编码实现

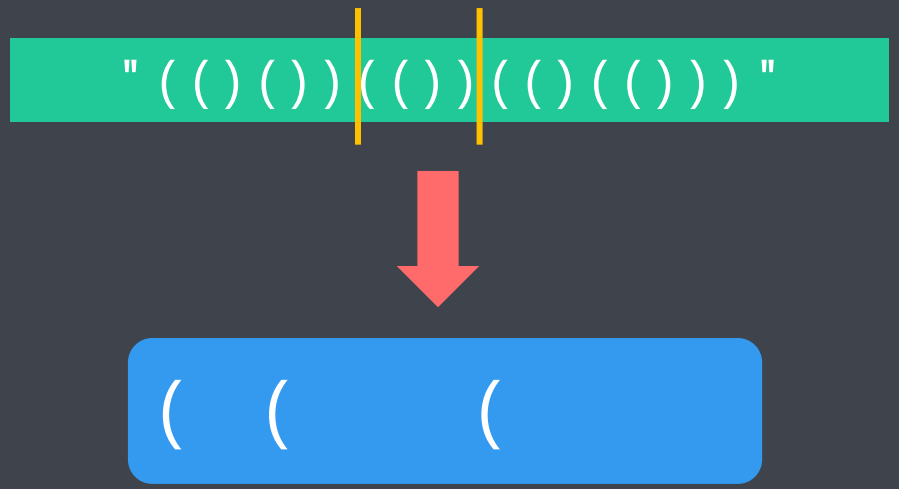
拉勾教育
—互联网人实战大学—

最优解：边界和细节问题

1. 边界问题：

遍历字符串，注意索引越界：i < S.length()

截取原语字符串时，注意起止索引：[start, end)



2. 细节问题：

需要记录上一次截取原语子串之后的位置

右括号无需进栈

五. Code 最优解思路及编码实现

拉勾教育
—互联网人实战大学—

```java
public String removeOuterParentheses(String S) {
    StringBuilder result = new StringBuilder();
    // 1.使用栈临时存储字符，替代计数器
    Stack<Character> stack = new Stack<>();
    int lastOpr = 0;
    // 2.遍历字符串，根据情况
    for (int i = 0; i < S.length(); i++) {
        char ch = S.charAt(i);
        if (ch == '(') { // 遇到左括号，左括号入栈
            stack.push(ch);
        } else if (ch == ')') { // 遇到右括号，左括号出栈
            stack.pop(); // 栈顶的左括号出栈
        }
        // 3.判断栈是否为空，若为空，找到了一个完整的原语
        if (stack.isEmpty()) {
            // 4.截取不含最外层括号的原语子串并进行拼接
            result.append(S.substring(lastOpr + 1, i));
            lastOpr = i + 1;// 往后找，再次初始化原语开始位置
        }
    }
    return result.toString();
}
```

时间复杂度：O(n)

- 遍历整个字符串：O(n)
- ~~遍历原语子串：O(n)~~

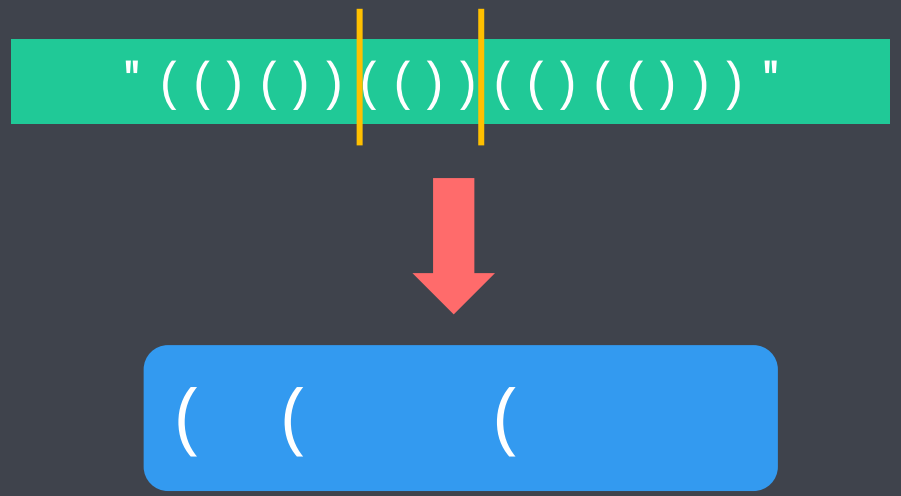
空间复杂度：O(n)

- ~~定义容器存储原语子串：O(n)~~
- ~~定义计数器：O(1)~~
- 使用栈临时存储字符：O(n)
- 生成原语子串：O(n)
- 生成新的字符串：O(n)

执行耗时:9 ms,击败了38.99% 的Java用户
内存消耗:38.2 MB,击败了98.65% 的Java用户

五. Code 最优解思路及编码实现

最优解：优化代码，直接用数组取代栈



1. 直接用数组取代栈

   创建数组、栈顶索引，使用数组操作入栈和出栈

2. 将原字符串转为数组进行遍历

   char[] s = S.toCharArray();

3. 去掉截取子串的操作，将原语字符直接拼接

   读取到左括号：此前有数据，当前必属原语

   读取到右括号：匹配后不为0，当前必属原语

五. Code 最优解思路及编码实现

拉勾教育
—互联网人实战大学—

```java
public String removeOuterParentheses(String S) {
    StringBuilder result = new StringBuilder();
    // 1.直接用数组取代栈
    int index = -1; // 栈顶索引
    int len = S.length();
    char[] stack = new char[len];
    // 2.将原字符串转为数组进行遍历
    char[] s = S.toCharArray();
    for (int i = 0; i < len; i++) {
        char ch = s[i];
        if (ch == '(') {
            // 3.去掉截取子串的操作，将原语字符直接拼接
            if (index > -1) { // 此前有数据，
                result.append(ch); // 则当前字符必然是原语的一部分
            }
            stack[index++] = ch; // 遇到左括号，左括号入栈
        } else /*if (ch == ')')*/ {
            stack[index--] = '\u0000'; // 遇到右括号，左括号出栈
            // 3.去掉截取子串的操作，将原语字符直接拼接
            if (index > -1) { // 右括号匹配之后不为0，
                result.append(ch); // 则当前字符必然是原语的一部分
            }
        }
    }
    return result.toString();
}
```

时间复杂度：O(n)

- 遍历整个字符串：O(n)

空间复杂度：O(n)

- 创建一个数组替代栈：O(n)
- 字符串转字符数组：O(n)
- 生成新的字符串：O(n)

执行耗时:3 ms,击败了90.24% 的Java用户
内存消耗:38.3 MB,击败了97.90% 的Java用户

五. Code 最优解思路及编码实现

拉勾教育
—互联网人实战大学—

```java
public String removeOuterParentheses(String S) {
    StringBuffer sb = new StringBuffer();
    // 1.定义计数器count，统计左右括号出现的次数
    int count = 0;
    int len = S.length();
    char[] s = S.toCharArray(); // 字符数组
    // 2.遍历字符串，根据读取数据进行计数
    for (int i = 0; i < len; i++) {
        char c = s[i];
        if (c == '(') {
            // 3.根据计数情况判断左右括号是否属于原语
            if (count > 0) { // 此前有数据，
                sb.append(c); // 则当前必然是原语的一部分
            }
            count++; // 读取到左括号自增
        } else {
            count--; // 读取到右括号自减
            // 3.根据计数情况判断左右括号是否属于原语
            if (count > 0) { // 右括号之后不为0，count自减后仍有数据，
                sb.append(c); // 则当前必然是原语的一部分
            }
        }
    }
    // 4.将原语拼接到缓冲区并返回
    return sb.toString();
}
```

时间复杂度：O(n)

- 遍历整个字符串：O(n)

空间复杂度：O(n)

- 字符串转字符数组：O(n)
- 计数器：O(1)
- 生成新的字符串：O(n)

执行耗时:2 ms,击败了100% 的Java用户
内存消耗:38.1 MB,击败了98.95% 的Java用户

六. Change 变形延伸

拉勾教育
—互联网人实战大学—

题目变形

- （练习）删除最内层的括号
- （练习）用链表实现一个栈（链栈）

延伸扩展

- 逆波兰表达式求值：给定数字和运算符，根据规则计算出结果
- 子程序调用：方法依次进栈，最后调用的最先结束

本题来源

- Leetcode 1021 https://leetcode-cn.com/problems/remove-outermost-parentheses/

总结

拉勾教育
—互联网人实战大学—

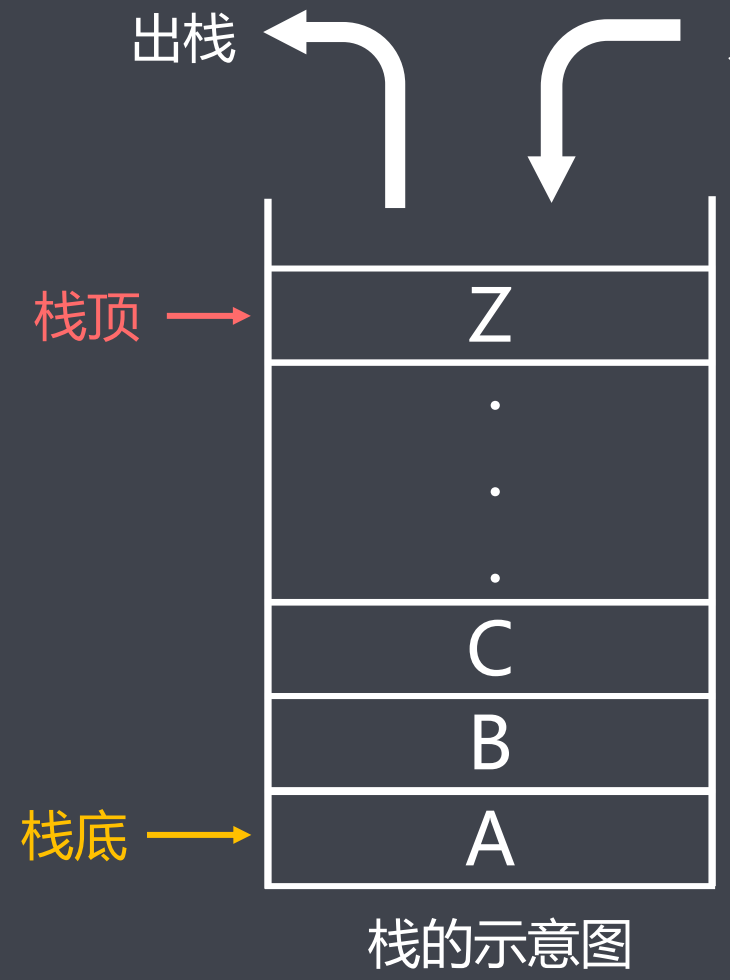
6C解题法

栈的基本概念和特点

- 限定仅在表尾进行插入和删除操作的线性表
- LIFO（Last In First Out），后进先出
- 基本操作：进栈（入栈/压栈）、出栈

算法思维

- 计数器
- 累加



栈的示意图

课后练习

拉勾教育
—互联网人实战大学—

1. 最小栈（Leetcode 155/简单）
2. 栈的压入、弹出序列（剑指 Offer 31/简单）
3. 堆盘子（面试题 03.03/中等）
4. 餐盘栈（Leetcode 1172/困难）

拉勾教育

—互联网人实战大学—

下载「拉勾教育App」
获取更多内容