欢迎大家参与本项目，贡献其他语言版本的代码，拥抱开源，让更多学习算法的小伙伴们收益！

> 匹配问题都是栈的强项

1047. 删除字符串中的所有相邻重复项

https://leetcode-cn.com/problems/remove-all-adjacent-duplicates-in-string/

给出由小写字母组成的字符串 S，重复项删除操作会选择两个相邻且相同的字母，并删除它们。

在 S 上反复执行重复项删除操作，直到无法继续删除。

在完成所有重复项删除操作后返回最终的字符串。答案保证唯一。

示例：

- 输入："abbaca"
- 输出："ca"
- 解释：例如，在 "abbaca" 中，我们可以删除 "bb" 由于两字母相邻且相同，这是此时唯一可以执行删除操作的重复项。之后我们得到字符串 "aaca"，其中又只有 "aa" 可以执行重复项删除操作，所以最后的字符串为 "ca"。

提示：

- 1 <= S.length <= 20000
- S 仅由小写英文字母组成。

思路

题外话

这道题目就像是我们玩过的游戏对对碰，如果相同的元素放在挨在一起就要消除。

可能我们在玩游戏的时候感觉理所当然应该消除，但程序又怎么知道该如果消除呢，特别是消除之后又有新的元素可能挨在一起。

此时游戏的后端逻辑就可以用一个栈来实现（我没有实际考察对对碰或者爱消除游戏的代码实现，仅从原理上进行推断）。

游戏开发可能使用栈结构，编程语言的一些功能实现也会使用栈结构，实现函数递归调用就需要栈，但不是每种编程语言都支持递归，例如：

递归的实现就是：每一次递归调用都会把函数的局部变量、参数值和返回地址等压入调用栈中，然后递归返回的时候，从栈顶弹出上一次递归的各项参数，所以这就是递归为什么可以返回上一层位置的原因。

相信大家应该遇到过一种错误就是栈溢出，系统输出的异常是`Segmentation fault`（当然不是所有的`Segmentation fault`都是栈溢出导致的），如果你使用了递归，就要想一想是不是无限递归了，那么系统调用栈就会溢出。

而且**在企业项目开发中，尽量不要使用递归！**在项目比较大的时候，由于参数多，全局变量等等，使用递归很容易判断不充分return的条件，非常容易无限递归（或者递归层级过深），**造成栈溢出错误（这种问题还不好排查！）**

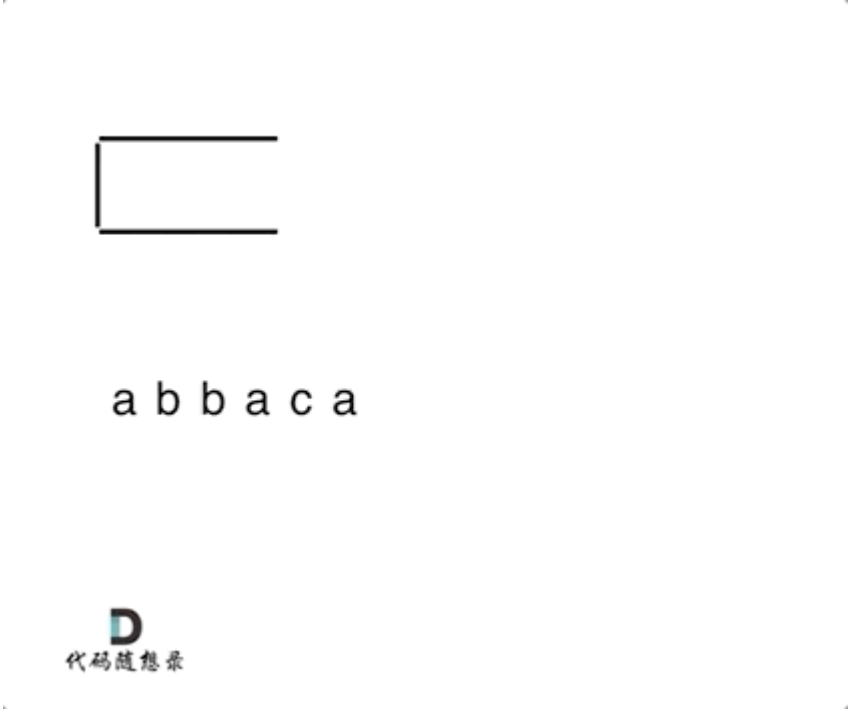
好了，题外话over，我们进入正题。

正题

本题要删除相邻相同元素，其实也是匹配问题，相同左元素相当于左括号，相同右元素就是相当于右括号，匹配上了就删除。

那么再来看一下本题：可以把字符串顺序放到一个栈中，然后如果相同的话 栈就弹出，这样最后栈里剩下的元素都是相邻不相同的元素了。

如动画所示：



从栈中弹出剩余元素，此时是字符串ac，因为从栈里弹出的元素是倒叙的，所以在对字符串进行反转一下，就得到了最终的结果。

C++代码：

```
class Solution {
public:
    string removeDuplicates(string S) {
        stack<char> st;
        for (char s : S) {
            if (st.empty() || s != st.top()) {
                st.push(s);
            } else {
                st.pop(); // s 与 st.top()相等的情况
            }
        }
        string result = "";
        while (!st.empty()) { // 将栈中元素放到result字符串汇总
            result += st.top();
            st.pop();
        }
        reverse (result.begin(), result.end()); // 此时字符串需要反转一下
        return result;

    }
};
```

当然可以拿字符串直接作为栈，这样省去了栈还要转为字符串的操作。

代码如下：

```
class Solution {
public:
    string removeDuplicates(string S) {
        string result;
        for(char s : S) {
```

```
            if(result.empty() || result.back() != s) {
                result.push_back(s);
            }
            else {
                result.pop_back();
            }
        }
        return result;
    }
};
```

其他语言版本

Java：

使用 Deque 作为堆栈

```
class Solution {
    public String removeDuplicates(String S) {
        //ArrayDeque会比LinkedList在除了删除元素这一点外会快一点
        //参考：https://stackoverflow.com/questions/6163166/why-is-arraydeque-better-than-l
        ArrayDeque<Character> deque = new ArrayDeque<>();
        char ch;
        for (int i = 0; i < S.length(); i++) {
            ch = S.charAt(i);
            if (deque.isEmpty() || deque.peek() != ch) {
                deque.push(ch);
            } else {
                deque.pop();
            }
        }
        String str = "";
        //剩余的元素即为不重复的元素
        while (!deque.isEmpty()) {
            str = deque.pop() + str;
        }
        return str;
    }
}
```

拿字符串直接作为栈，省去了栈还要转为字符串的操作。

```
class Solution {
    public String removeDuplicates(String s) {
        // 将 res 当做栈
        StringBuffer res = new StringBuffer();
        // top为 res 的长度
        int top = -1;
        for (int i = 0; i < s.length(); i++) {
            char c = s.charAt(i);
            // 当 top > 0,即栈中有字符时，当前字符如果和栈中字符相等，弹出栈顶字符，同时 top--
            if (top >= 0 && res.charAt(top) == c) {
                res.deleteCharAt(top);
```

```
                top--;
            // 否则，将该字符 入栈，同时top++
            } else {
                res.append(c);
                top++;
            }
        }
        return res.toString();
    }
}
```

拓展：双指针

```
class Solution {
    public String removeDuplicates(String s) {
        char[] ch = s.toCharArray();
        int fast = 0;
        int slow = 0;
        while(fast < s.length()){
            // 直接用fast指针覆盖slow指针的值
            ch[slow] = ch[fast];
            // 遇到前后相同值的，就跳过，即slow指针后退一步，下次循环就可以直接被覆盖掉了
            if(slow > 0 && ch[slow] == ch[slow - 1]){
                slow--;
            }else{
                slow++;
            }
            fast++;
        }
        return new String(ch,0,slow);
    }
}
```

Python：

```
class Solution:
    def removeDuplicates(self, s: str) -> str:
        t = list()
        for i in s:
            if t and t[-1] == i:
                t.pop(-1)
            else:
                t.append(i)
        return "".join(t)  # 字符串拼接
```

Go：

```
func removeDuplicates(s string) string {
    var stack []byte
    for i := 0; i < len(s);i++ {
        // 栈不空 且 与栈顶元素不等
        if len(stack) > 0 && stack[len(stack)-1] == s[i] {
            // 弹出栈顶元素 并 忽略当前元素(s[i])
```

```
            stack = stack[:len(stack)-1]
        }else{
            // 入栈
            stack = append(stack, s[i])
        }
    }
    return string(stack)
}
```

javaScript:

```
/**
 * @param {string} s
 * @return {string}
 */
var removeDuplicates = function(s) {
    const stack = [];
    for(const x of s) {
        let c = null;
        if(stack.length && x === (c = stack.pop())) continue;
        c && stack.push(c);
        stack.push(x);
    }
    return stack.join("");
};
```

- 作者微信：程序员Carl
- B站视频：代码随想录
- 知识星球：代码随想录