题目地址(394. 字符串解码)

https://leetcode-cn.com/problems/decode-string/

入选理由

1. 前面说了栈的难度比较大，因此接下来几天都是栈，今天这个就是。
2. 今天的难度相比昨天难度增加。 关键的是这道题的解法很有用，很多力扣的题都用这种思路，甚至是 hard 题目，基本思路也是一样的。实际上，这题就是一个括号匹配而已，匹配的括号对作为一层。大家可以尝试使用递归和迭代两种方式解决，来直观感受一下。

题目描述

给定一个经过编码的字符串，返回它解码后的字符串。

编码规则为: k[encoded_string]，表示其中方括号内部的 encoded_string 正好重复 k 次。注意 k 保证为正整数。

你可以认为输入字符串总是有效的；输入字符串中没有额外的空格，且输入的方括号总是符合格式要求的。

此外，你可以认为原始数据不包含数字，所有的数字只表示重复的次数 k ，例如不会出现像 3a 或 2[4] 的输入。

示例 1:

输入: s = "3[a]2[bc]"
输出: "aaabcbc"
示例 2:

输入: s = "3[a2[c]]"
输出: "accaccacc"
示例 3:

输入: s = "2[abc]3[cd]ef"
输出: "abcabccdcdcdef"
示例 4:

输入: s = "abc3[cd]xyz"
输出: "abccdcdcdxyz"

难度

- 中等

标签

- 栈
- DFS

前置知识

- 栈
- 括号匹配

方法一：栈

思路

题目要求将一个经过编码的字符解码并返回解码后的字符串。题目给定的条件是只有四种可能出现的字符

1. 字母
2. 数字
3. [
4.]

并且输入的方括号总是满足要求的（成对出现），数字只表示重复次数。

那么根据以上条件，可以看出其括号符合栈先进后出的特性以及递归的特质，稍后我们使用递归来解。

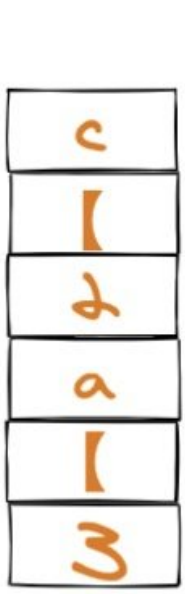
那么现在看一下迭代的解法。

我们可以利用 stack 来实现这个操作，遍历这个字符串 s，判断每一个字符的类型：

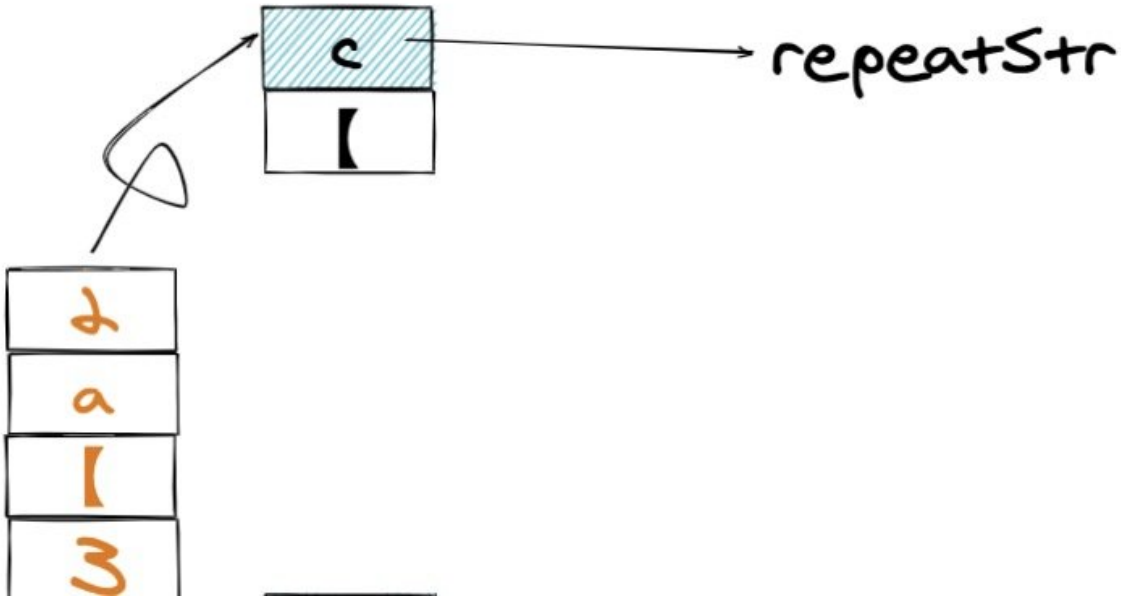
- 如果是字母 --> 添加到 stack 当中
- 如果是数字 --> 先不着急添加到 stack 中 --> 因为有可能有多位
- 如果是 [--> 说明重复字符串开始 --> 将数字入栈 --> 并且将数字清零
- 如果是] --> 说明重复字符串结束 --> 将重复字符串重复前一步储存的数字遍

拿题目给的例子 `s = "3[a2[c]]"` 来说：

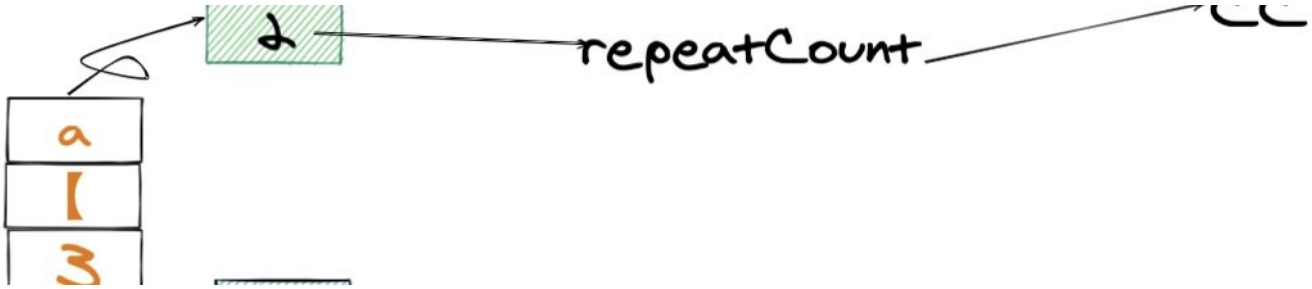
在遇到 】 之前，我们不断执行压栈操作：



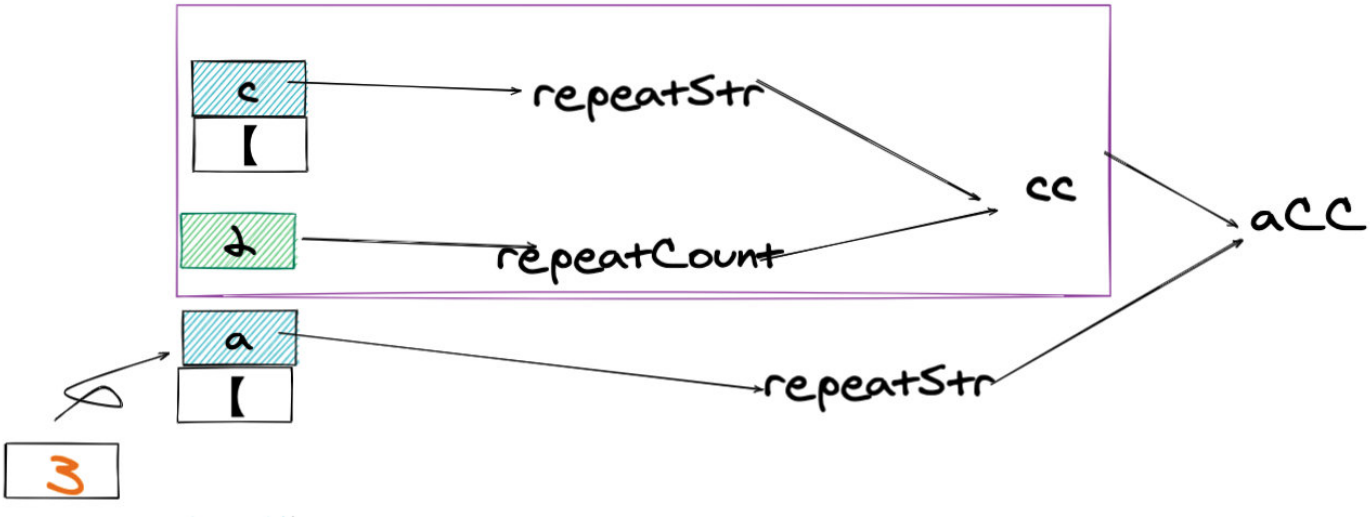
当遇到 】 的时候，说明我们应该出栈了，不断出栈直到找到对应的 【，这中间的字符就是 repeatStr。



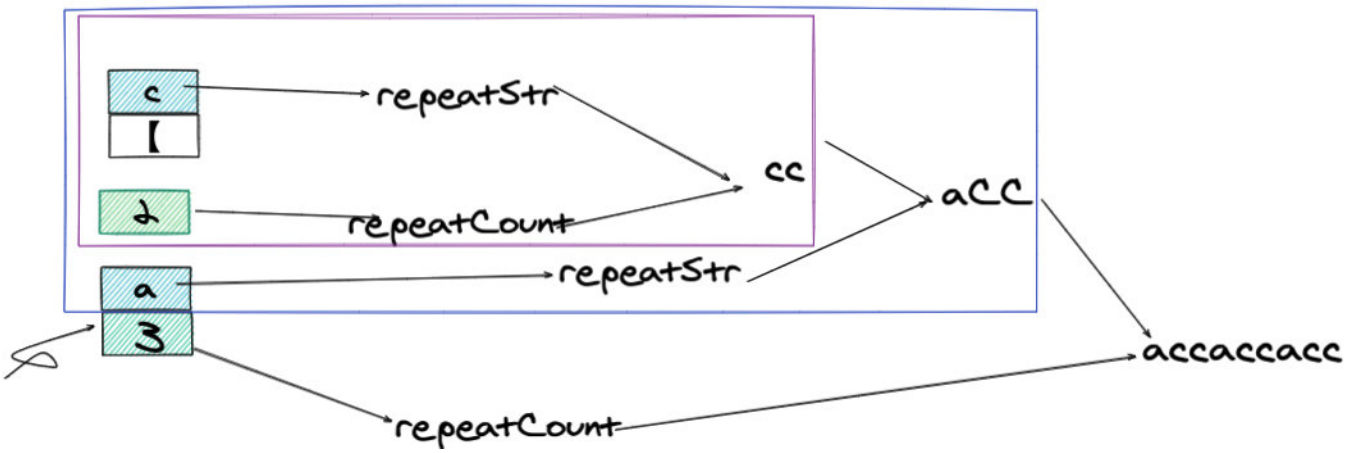
但是要重复几次呢？ 我们需要继续出栈，直到非数字为止，这个数字我们记录为 repeatCount。



而最终的字符串就是 repeatCount 个 repeatStr 拼接的形式。 **并将其看成一个字母压入栈中**。



继续，后面的逻辑是一样的：



（最终图）

代码

代码支持：Python

Python：

```python
class Solution:
    def decodeString(self, s: str) -> str:
        stack = []
        for c in s:
            if c == ']':
                repeatStr = ''
                repeatCount = ''
                while stack and stack[-1] != '[':
                    repeatStr = stack.pop() + repeatStr
                # pop 掉 "["
                stack.pop()
                while stack and stack[-1].isnumeric():
                    repeatCount = stack.pop() + repeatCount
                stack.append(repeatStr * int(repeatCount))
            else:
                stack.append(c)
        return "".join(stack)
```

复杂度分析

- 时间复杂度：$O(N)$，其中 N 为解码后的 s 的长度。
- 空间复杂度：$O(N)$，其中 N 为解码后的 s 的长度。

方法二：递归

思路

递归的解法也是类似。由于递归的解法并不比迭代书写简单，以及递归我们将在第三节讲述。

主逻辑仍然和迭代一样。 只不过每次碰到左括号就进入递归，碰到右括号就跳出递归返回即可。

唯一需要注意的是，我这里使用了 start 指针跟踪当前遍历到的位置， 因此如果使用递归需要在递归返回后更新指针。

总结一下：

- 遇到数字，我们需要将其累加到 count 中。（因为数字可能有多位）
- 遇到普通字符，将其直接添加到栈
- 遇到 "["，开启一轮新的递归。由于需要继续上次递归结束的地方继续处理，因此递归的返回值需要包含索引，返回元组可以做到这一点
- 遇到 "]"，结束递归，返回索引和栈上的字符。

这是一个典型的 DFS + 栈的题目，值得大家掌握。

代码

```
class Solution:

    def decodeString(self, s: str) -> str:
        def dfs(start):
            repeat_str = repeat_count = ''
            while start < len(s):
                if s[start].isnumeric():
                    repeat_count += s[start]
                elif s[start] == '[':
                    # 更新指针
                    start, t_str = dfs(start + 1)
                    # repeat_count 仅作用于 t_str，而不作用于当前的 repeat_str
                    repeat_str = repeat_str + t_str * int(repeat_count)
                    repeat_count = ''
                elif s[start] == ']':
                    return start, repeat_str
                else:
                    repeat_str += s[start]
                start += 1
            return repeat_str
        return dfs(0)
```

复杂度分析

- 时间复杂度：$O(N)$，其中 N 为解码后的 s 的长度。
- 空间复杂度：$O(N)$，其中 N 为解码后的 s 的长度。

相关题目

- 字符串扩展 和这道题一模一样，只是把 [ ] 换成了 ( )
- S 表达式 一样的使用栈轻松解决的经典题目

更多题解可以访问我的 LeetCode 题解仓库：https://github.com/azl397985856/leetcode 。 目前已经 37K star 啦。

大家也可以关注我的公众号《力扣加加》获取更多更新鲜的 LeetCode 题解

欢迎长按关注

上一页 下一页