题目描述

- 为了提升数据传输的效率，会对传输的报文进行压缩处理。
- 输入一个压缩后的报文，请返回它解压后的原始报文。
- 压缩规则：n[str]，表示方括号内部的 str 正好重复 n 次。
- 注意 n 为正整数（0 < n <= 100），str只包含小写英文字母，不考虑异常情况。

输入描述

输入压缩后的报文：

1）不考虑无效的输入，报文没有额外的空格，方括号总是符合格式要求的；

2）原始报文不包含数字，所有的数字只表示重复的次数 n ，例如不会出现像 5b 或 3[8] 的输入；

输出描述

解压后的原始报文

注：原始报文长度不会超过1000，不考虑异常的情况

用例

输入	3[k]2[mn]
输出	kkkmnmn
说明	k 重复3次，mn 重复2次，最终得到 kkkmnmn

输入	3[m2[c]]
输出	mccmccmcc
说明	m2[c] 解压缩后为 mcc，重复三次为 mccmccmcc

题目解析

本题可以使用栈结构解题。思路也比较简单明了。

我们只需要统计出如下几个关键信息即可：

- 要被重复的子串（可以分解记录子串两边的'['，']'的位置）
- 要被重复的子串的重复次数

定义一个栈stack，然后开始遍历输入字符串str的每一个字符c：

- 如果 c == '['，则可以统计到两个信息：

  1. 要被重复的子串的起始位置
  2. 根据题目描述，'['的前面必然是重复次数，因此'['可以作为重复次数结束统计的标志

- 如果 c == ']'，则可以得到要被重复的子串的结束位置，此时我们可以进行生成解压串，来替换掉对应范围的压缩串
- 如果 c 是数字，则必然是重复次数的组成，此时可以记录下来，当遇到'['时，可以得到一个重复次数
- 如果 c 是其他字符，则压入栈中。

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

rl.on("line", (line) => {
  console.log(getResult(line));
});

/* 解法逻辑 */
function getResult(str) {
  const stack = [];

  // idxs记录要被重复的子串的起始位置
  const idxs = [];
  // nums记录要被重复的子串的重复次数，和idxs对应
  const nums = [];

  // tmpRepeatCount记录正在拼接的重复次数字符串
  const tmpRepeatCount = [];

```

```
  // 遍历输入的字符串，c是当前正在遍历的字符
  for (let c of str) {
    if (c == "[") {
      // 此时tmpRepeatCount已记录完当前重复子串对应的重复次数的所有字符
      const repeatCount = Number(tmpRepeatCount.join(""));
      nums.push(repeatCount);
      tmpRepeatCount.length = 0;

      // 记录要被重复的子串的起始位置
      idxs.push(stack.length);
    } else if (c == "]") {
      // 需要被重复的子串在栈中的起始位置
      const start = idxs.pop();
      // 需要被重复的次数
      const repeatCount = nums.pop();
      // 需要被重复的子串
      const repeatStr = stack.splice(start).join("");

      // 将新串压入栈中
      stack.push(new Array(repeatCount).fill(repeatStr).join(""));
    } else if (c >= "0" && c <= "9") {
      tmpRepeatCount.push(c);
    } else {
      stack.push(c);
    }
  }

  return stack.join("");
}
```

Java算法源码

```java
import java.util.LinkedList;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    String str = sc.nextLine();
    System.out.println(getResult(str));
  }

  public static String getResult(String str) {
    // Java可以使用StringBuilder模拟栈
    StringBuilder sb = new StringBuilder();

    // idxs记录要被重复的子串的起始位置
    LinkedList<Integer> idxs = new LinkedList<>();
    // nums记录要被重复的子串的重复次数，和idxs对应
    LinkedList<Integer> nums = new LinkedList<>();
    // tmpRepeatCount记录重复次数的的字符组成
    StringBuilder tmpRepeatCount = new StringBuilder();

    // 遍历输入的字符串
    for (int i = 0; i < str.length(); i++) {
      // c是当前正在遍历的字符
      char c = str.charAt(i);

      if (c == '[') {
        // 此时tmpRepeatCount已记录完当前重复子串对应的重复次数的所有字符
        int repeatCount = Integer.parseInt(tmpRepeatCount.toString());
        nums.add(repeatCount);
        tmpRepeatCount = new StringBuilder();

```

```
        // 记录要被重复的子串的起始位置
        idxs.add(sb.length());
      } else if (c == ']') {
        // 需要被重复的子串在栈中的起始位置
        int start = idxs.removeLast();
        // 需要被重复的次数
        int repeatCount = nums.removeLast();
        // 需要被重复的子串
        String repeatStr = sb.substring(start);

        // 重复后的新串
        StringBuilder tmp = new StringBuilder();
        for (int j = 0; j < repeatCount; j++) tmp.append(repeatStr);

        // 替换对应子串为重复后的新串
        sb.replace(start, sb.length(), tmp.toString());
      } else if (c >= '0' && c <= '9') {
        tmpRepeatCount.append(c);
      } else {
        sb.append(c);
      }
    }

    return sb.toString();
  }
}
```

Python算法源码

```
# 输入获取
s = input()


# 算法入口
def getResult(s):
    stack = []

    # idxs记录要被重复的子串的起始位置
    idxs = []
    # nums记录要被重复的子串的重复次数，和idxs对应
    nums = []

    # tmpRepeatCount记录重复次数的的字符组成
    tmpRepeatCount = []

```

```
    # 遍历输入的字符串，c是当前正在遍历的字符
    for c in s:
        if c == '[':
            # 此时tmpRepeatCount已记录完当前重复子串对应的重复次数的所有字符
            repeatCount = int("".join(tmpRepeatCount))
            nums.append(repeatCount)
            tmpRepeatCount = []

            # 记录要被重复的子串的起始位置
            idxs.append(len(stack))
        elif c == ']':
            # 需要被重复的子串在栈中的起始位置
            start = idxs.pop()
            # 需要被重复的次数
            repeatCount = nums.pop()
            # 需要被重复的子串
            repeatStr = "".join(stack[start:])

            # 先删除要被替换的子串，然后加入新串（即重复对应次数的子串）
            stack = stack[:start]
            stack.append("".join([repeatStr] * repeatCount))
        elif '0' <= c <= '9':
            tmpRepeatCount.append(c)
        else:
            stack.append(c)

    return "".join(stack)


# 算法调用
print(getResult(s))
```