165. 比较版本号

难度 中等 179 收藏 分享 切换为英文 接收动态 反

给你两个版本号 `version1` 和 `version2` ，请你比较它们。

版本号由一个或多个修订号组成，各修订号由一个 `'.'` 连接。每个修订号由 **多位数字** 组成 **前导零** 。每个版本号至少包含一个字符。修订号从左到右编号，下标从 0 开始，最左边的修 0 ，下一个修订号下标为 1 ，以此类推。例如， `2.5.33` 和 `0.1` 都是有效的版本号。

比较版本号时，请按从左到右的顺序依次比较它们的修订号。比较修订号时，只需比较 **忽略后的整数值** 。也就是说，修订号 `1` 和修订号 `001` **相等** 。如果版本号没有指定某个下标处则该修订号视为 `0` 。例如，版本 `1.0` 小于版本 `1.1` ，因为它们下标为 `0` 的修订号相同 `1` 的修订号分别为 `0` 和 `1` ， `0 < 1` 。

返回规则如下：

- 如果 *`version1 > version2`* 返回 `1` ，
- 如果 *`version1 < version2`* 返回 `-1` ，
- 除此之外返回 `0` 。

示例 1：

```
输入：version1 = "1.01", version2 = "1.001"
输出：0
解释：忽略前导零，"01" 和 "001" 都表示相同的整数 "1"
```



```python
class Solution:
    def compareVersion(self, version1: str, version2: str) -> int:
        p1, p2 = 0, 0
        end = max(len(version1), len(version2))
        while p1 < end or p2 < end:
            v1, v2 = 0, 0
            while p1 < len(version1) and version1[p1] != '.':
                v1 = v1 * 10 + int(version1[p1])
                p1 += 1
            while p2 < len(version2) and version2[p2] != '.':
                v2 = v2 * 10 + int(version2[p2])
                p2 += 1
            if v1 > v2:
                return 1
            elif v2 > v1 :
                return -1
            p1 += 1
            p2 += 1
        return 0
```

38. 外观数列

难度 中等 728 收藏 分享 切换为英文 接收动态 反

给定一个正整数 `n` ，输出外观数列的第 `n` 项。

「外观数列」是一个整数序列，从数字 1 开始，序列中的每一项都是对前一项的描述。

你可以将其视作是由递归公式定义的数字字符串序列：

- `countAndSay(1) = "1"`
- `countAndSay(n)` 是对 `countAndSay(n-1)` 的描述，然后转换成另一个数字字

前五项如下：

```
1.     1
2.     11
3.     21
4.     1211
5.     111221
第一项是数字 1
描述前一项，这个数是 1 即 “ 一 个 1 ”，记作 "11"
描述前一项，这个数是 11 即 “ 二 个 1 ” ，记作 "21"
描述前一项，这个数是 21 即 “ 一 个 2 + 一 个 1 ” ，记作 "1211"
描述前一项，这个数是 1211 即 “ 一 个 1 + 一 个 2 + 二 个 1 ” ，记作 "111
```

要 **描述** 一个数字字符串，首先要将字符串分割为 **最小** 数量的组，每个组都由连续的最多 **相**
成。然后对于每个组，先描述字符的数量，然后描述字符，形成一个描述组。要将描述转换
串，先将每组中的字符数量用数字替换，再将所有描述组连接起来。

示例 1：

```
输入：n = 1
输出："1"
解释：这是一个基本样例。
```

示例 2：

```
输入：n = 4
输出："1211"
解释：
countAndSay(1) = "1"
countAndSay(2) = 读 "1" = 一 个 1 = "11"
countAndSay(3) = 读 "11" = 二 个 1 = "21"
countAndSay(4) = 读 "21" = 一 个 2 + 一 个 1 = "12" + "11" = "1211"
```

```python
class Solution:
    def countAndSay(self, n: int) -> str:
        pre = ''
        cur = '1'
        for i in range(1, n):
            pre = cur
            cur = ''
            start = 0
            end = 0
            while end < len(pre):
                while end < len(pre) and pre[end] == pre[start]:
                    end += 1
                cur += str(end - start) + pre[start]
                start = end
        return cur
```

115. 不同的子序列

难度 困难 569 收藏 分享 切换为英文 接收动态 反馈

给定一个字符串 `s` 和一个字符串 `t` ，计算在 `s` 的子序列中 `t` 出现的个数。

字符串的一个 **子序列** 是指，通过删除一些（也可以不删除）字符且不干扰剩余字符相对位置所
字符串。（例如，`"ACE"` 是 `"ABCDE"` 的一个子序列，而 `"AEC"` 不是）

题目数据保证答案符合 32 位带符号整数范围。

示例 1：

```
输入：s = "rabbbit", t = "rabbit"
输出：3
解释：
如下图所示，有 3 种可以从 s 中得到 "rabbit" 的方案。
rabbbit
rabbbit
rabbbit
```

```python
class Solution:
    def numDistinct(self, s: str, t: str) -> int:
        n1 = len(t)
        n2 = len(s)
        dp = [[0] * (n2 + 1) for _ in range(n1 + 1)]
        for j in range(n2 + 1):
            dp[0][j] = 1
        for i in range(1, n1 + 1):
            for j in range(1, n2 + 1):
                if t[i - 1] == s[j - 1]:
                    dp[i][j] = dp[i - 1][j - 1] + dp[i][j - 1]
                else:
                    dp[i][j] = dp[i][j - 1]
        return dp[-1][-1]
```


72. 编辑距离

难度 困难 1743 收藏 分享 切换为英文 接收动态 反

给你两个单词 `word1` 和 `word2`，请你计算出将 `word1` 转换成 `word2` 所使用的最少操

你可以对一个单词进行如下三种操作：

- 插入一个字符
- 删除一个字符
- 替换一个字符

示例 1：

```
输入：word1 = "horse", word2 = "ros"
输出：3
解释：
horse -> rorse (将 'h' 替换为 'r')
rorse -> rose (删除 'r')
rose -> ros (删除 'e')
```

```python
class Solution:
    def minDistance(self, word1: str, word2: str) -> int:
        n1 = len(word1)
        n2 = len(word2)
        dp = [[0] * (n2 + 1) for _ in range(n1 + 1)]
        for j in range(1, n2 + 1):
            dp[0][j] = dp[0][j - 1] + 1
        for i in range(1, n1 + 1):
            dp[i][0] = dp[i - 1][0] + 1
        for i in range(1, n1 + 1):
            for j in range(1, n2 + 1):
                dp[i][j] = dp[i - 1][j - 1] if word1[i - 1] == word
dp[i][j - 1], dp[i - 1][j]) + 1
        return dp[-1][-1]
```

面试题 01.05. 一次编辑

难度 中等 83 收藏 分享 切换为英文 接收动态 反馈

字符串有三种编辑操作:插入一个字符、删除一个字符或者替换一个字符。 给定两个字符串,数判定它们是否只需要一次(或者零次)编辑。

示例 1:

```
输入:
first = "pale"
second = "ple"
输出: True
```

示例 2:

```
输入:
first = "pales"
second = "pal"
输出: False
```



```python
class Solution:
    def oneEditAway(self, first: str, second: str) -> bool:
        if first == second:
            return True
        n1 = len(first)
        n2 = len(second)
        if abs(n1 - n2) > 1:
            return False

        i, j, k = 0, n1 - 1, n2 - 1
        while i < n1 and i < n2 and first[i] == second[i]:
            i += 1
        while j >= 0 and k >= 0 and first[j] == second[k]:
            k -= 1
            j -= 1
        return k - i < 1 and j - i < 1
```

293. 翻转游戏

难度 简单 27 收藏 分享 切换为英文 接收动态 反馈

你和朋友玩一个叫做「翻转游戏」的游戏。游戏规则如下：

给你一个字符串 `currentState` ，其中只含 `'+'` 和 `'-'` 。你和朋友轮流将 **连续** 的两个 `"++"` 反转成 `"--"` 。当一方无法进行有效的翻转时便意味着游戏结束，则另一方获胜。

计算并返回 **一次有效操作** 后，字符串 `currentState` 所有的可能状态，返回结果可以按任意顺序排列。如果不存在可能的有效操作，请返回一个空列表 `[]` 。

示例 1：

```
输入：currentState = "++++"
输出：["--++","+--+","++--"]
```

示例 2：

```
输入：currentState = "+"
输出：[]
```

```python
class Solution:
    def generatePossibleNextMoves(self, currentState: str) -> List[
        res = []
        for i in range(len(currentState) - 1):
            if currentState[i] == currentState[i + 1] == '+':
                res.append(currentState[:i] + '--' + currentState[i
        return res
```


294. 翻转游戏 II

难度 中等 72 收藏 分享 切换为英文 接收动态 反馈

你和朋友玩一个叫做「翻转游戏」的游戏。游戏规则如下：

给你一个字符串 `currentState` ，其中只含 `'+'` 和 `'-'` 。你和朋友轮流将 **连续** 的两个 `"++"` 反转成 `"--"` 。当一方无法进行有效的翻转时便意味着游戏结束，则另一方获胜。

请你写出一个函数来判定起始玩家 **是否存在获胜的方案** ：如果存在，返回 `true` ；否则，返回 `false` 。

示例 1：

```
输入：currentState = "++++"
输出：true
解释：起始玩家可将中间的 "++" 翻转变为 "+--+" 从而得胜。
```

示例 2：

```
输入：currentState = "+"
输出：false
```

```python
from functools import lru_cache
class Solution:
    @lru_cache(None)
    def canWin(self, s: str) -> bool:
        n = len(s)
        for i in range(n - 1):
            if s[i] == s[i + 1] == '+':
                if not self.canWin(s[:i] + '--' + s[i + 2:]):
                    return True
        return False
```

150. 逆波兰表达式求值

难度 中等 381 收藏 分享 切换为英文 接收动态 反馈

根据 逆波兰表示法，求表达式的值。

有效的算符包括 `+`、`-`、`*`、`/` 。每个运算对象可以是整数，也可以是另一个逆波兰表达式

说明：

- 整数除法只保留整数部分。
- 给定逆波兰表达式总是有效的。换句话说，表达式总会得出有效数值且不存在除数为 0

示例 1：

```
输入：tokens = ["2","1","+","3","*"]
输出：9
解释：该算式转化为常见的中缀算术表达式为：((2 + 1) * 3) = 9
```

示例 2：

```
输入：tokens = ["4","13","5","/","+"]
输出：6
解释：该算式转化为常见的中缀算术表达式为：(4 + (13 / 5)) = 6
```



```python
class Solution:
    def evalRPN(self, tokens: List[str]) -> int:
        stack = []
        for s in tokens:
            try:
                stack.append(int(s))
            except:
                num2 = stack.pop()
                num1 = stack.pop()
                stack.append(self.operate(num1, num2, s))
        return stack[0]

    def operate(self, num1, num2, op):
        if op == '+':
            return num1 + num2
        elif op == '-':
            return num1 - num2
        elif op == '*':
            return num1 * num2
        elif op == '/':
            return int(num1 / num2)
```

题目描述

小猿同学特别热心肠，喜欢帮助同事~听闻有一个同事需要搬家连忙过去帮忙。小猿将物品放到箱子里。再将小箱子放到大箱子里。小猿突然忘了用了几个箱子，你能帮帮它吗？[]代表一个箱子，[]3代表3个箱子，[[]3]代表1个大箱子里放了3个小箱子一共有4个箱子，[[]3]2代表有2个大箱子，每个大箱子里放了3个小箱子，一共有8个箱子。

输入描述:

一行字符串boxes，代表箱子的摆放情况。（boxes长度len，2<=len<=10^6）

对于输入的箱子：
1、保证一定是完整的箱子[]，不会出现半个箱子[
2、保证箱子套箱子的层数dep，1<=dep<=10
3、[]x，2<=x<=9（[]x见题意）

```python
def findbox(s:str) -> int:
    n = len(s)
    stack = []
    for i in range(n):
        if s[i] == '[':
            stack.append(-1)
        elif s[i] == ']':
            tmp = 1
            while stack[-1] != -1:
                tmp += stack.pop()
            stack.pop()
            stack.append(tmp)
        else:
            top = stack.pop()
            stack.append(top * int(s[i]))

    return sum(stack)
```

压缩字符串的规则如下：

1. 如果字母 x 连续出现 n 次（10000>n>1），则表示为 (a)n
2. 可以嵌套，比如 ((a)2(b)2)2 ，表示的是 aabbaabb 的压缩后结果
3. 只出现一次的字母不进行压缩，a 的压缩后结果仍然为 a

输入为一个字符串的压缩结果，请输出压缩前的字符串


```python
class Solution:
    def decompress(self, s):
        stack = []
        n = len(s)
        for i in range(n):
            if s[i] == '(':
                stack.append(-1)
            elif s[i].isalpha():
                stack.append(s[i])
            elif s[i] ==')':
                tmp = ''
                while stack[-1] != -1:
                    tmp = stack.pop() + tmp
                stack.pop()
                stack.append(tmp)
            else:
                top = stack.pop()
                stack.append(top * int(s[i]))
        return ''.join(stack)
```

最短回文串

单词规律

290. 单词规律

难度 简单 372 收藏 分享 切换为英文 接收动态 反

给定一种规律 `pattern` 和一个字符串 `str` ，判断 `str` 是否遵循相同的规律。

这里的 **遵循** 指完全匹配，例如， `pattern` 里的每个字母和字符串 `str` 中的每个非空单词
着双向连接的对应规律。

示例1:

```
输入: pattern = "abba", str = "dog cat cat dog"
输出: true
```

示例 2:

```
输入:pattern = "abba", str = "dog cat cat fish"
输出: false
```

示例 3:

```
输入: pattern = "aaaa", str = "dog cat cat dog"
输出: false
```



```python
class Solution:
    def wordPattern(self, pattern: str, s: str) -> bool:
        res = s.split()
        return list(map(pattern.index, pattern)) == list(map(r

    def wordPattern(self, pattern: str, s: str) -> bool:
        word2ch = dict()
        ch2word = dict()
        words = s.split()
        if len(pattern) != len(words):
            return False

        for ch, word in zip(pattern, words):
            if (word in word2ch and word2ch[word] != ch) or (c
                return False
            ch2word[ch] = word
            word2ch[word] = ch

        return True
```