0.双指针总结.md

- 基本操作：一个指针代表当前位置，一个指针代表写入位置
  - 26 删除有序数组中的重复项
  - 27 移除元素
  - 80 删除有序数组中的重复项 II
  - 209 长度最小的子数组
  - 283 移动零
- 首尾双指针
  - 11 盛最多水的容器
  - 15 三数之和
  - 16 最接近的三数之和
  - 18 四数之和
  - 125 验证回文串
  - 167 两数之和 II - 输入有序数组
  - 344 反转字符串
  - 345 反转字符串中的元音字母
  - 680 验证回文字符串 II
- 左右边界双指针
  - 5 最长回文子串
  - 647 回文子串
  - 31 下一个排列
  - 904 水果成篮
- 类双输入
  - 75 颜色分类
  - 88 合并两个有序数组
  - 844 比较含退格的字符串
  - 977 有序数组的平方
- 超级技巧
  - 28 实现 strStr()： KMP算法

11. 盛最多水的容器.md

给你 n 个非负整数 a1，a2，...，an，每个数代表坐标中的一个点 (i, ai) 。在坐标内画 n 条垂直线，垂直线 i 的两个端点分别为 (i, ai) 和 (i, 0) 。找出其中的两条线，使得它们与 x 轴共同构成的容器可以容纳最多的水。

说明：你不能倾斜容器。

示例 1：

输入：[1,8,6,2,5,4,8,3,7]
输出：49
解释：图中垂直线代表输入数组 [1,8,6,2,5,4,8,3,7]。在此情况下，容器能够容纳水（表示为蓝色部分）的最大值为 49。

示例 2：

输入：height = [1,1]
输出：1

示例 3：

输入：height = [4,3,2,1,4]
输出：16

示例 4：

输入：height = [1,2,1]
输出：2

提示：

```
n == height.length
2 <= n <= 105
0 <= height[i] <= 104
```

```python
class Solution:
    def maxArea(self, height: List[int]) -> int:
        if not height:
            return 0
        area = 0
        i=0
        j=len(height)-1
        while i<j:
            if height[i] < height[j]:
                area = max(area, (height[i] * (j-i)))
                i +=1
            else:
                area = max(area, (height[j]* (j-i)))
                j-=1
        return area
```

Tips

1. 首尾双指针向前遍历，唯一的技巧在于，移动指针的时候先移动高度更低的指针，因为盛水的量受到短板的影响，所以优先移动短板

125. 验证回文串.md

给定一个字符串，验证它是否是回文串，只考虑字母和数字字符，可以忽略字母的大小写。

说明：本题中，我们将空字符串定义为有效的回文串。

示例 1:

输入: "A man, a plan, a canal: Panama"
输出: true
解释："amanaplanacanalpanama" 是回文串

示例 2:

输入: "race a car"
输出: false
解释："raceacar" 不是回文串

提示：

```
1 <= s.length <= 2 * 105
字符串 s 由 ASCII 字符组成
```

```python
class Solution:
    def isPalindrome(self, s: str) -> bool:
        s = s.strip()
        i=0
        j=len(s)-1
        while i <j:
            print(i,j)

            if not s[i].isalnum():
                i+=1
                continue
            if not s[j].isalnum():
                j-=1
                continue

            if s[i].lower()!=s[j].lower():
                return False
            else:
                i+=1
                j-=1
        return True
```

Tips

就一个新知识就是isalnum()， 判断字符是否非空，且是numeric or alpha

15. 三数之和.md

给你一个包含 n 个整数的数组 nums，判断 nums 中是否存在三个元素 a，b，c ，使得 a + b + c = 0 ？请你找出所有和为 0 且不重复的三元组。

注意：答案中不可以包含重复的三元组。

示例 1：

输入：nums = [-1,0,1,2,-1,-4]
输出：[[-1,-1,2],[-1,0,1]]

示例 2：

输入：nums = []
输出：[]

示例 3：

输入：nums = [0]
输出：[]

提示：

```
0 <= nums.length <= 3000
-105 <= nums[i] <= 105
```

```
class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        if len(nums)<3:
            return []
        nums = sorted(nums)
        n = len(nums)
        res = []
        for i in range(n-2):
            if (i != 0) & (nums[i]==nums[i-1]):
                continue
            target = -nums[i]
            third = n-1
            second = i+1
            while second < third:
```

```
                if (second !=i+1) & (nums[second]==nums[second-1]):
                    second +=1
                    continue
                if nums[second]+nums[third] == target:
                    res.append([nums[i],nums[second],nums[third]])
                    #
                    second+=1
                    third-=1
                elif nums[second]+ nums[third]< target:
                    second+=1
                else:
                    third-=1
        return res
```

TIps

双指针解法

- 指针1向前遍历，只判断是否重复，重复跳过
- 指针2/3是左右指针向中间遍历

16. 最接近的三数之和.md

给定一个包括 n 个整数的数组 nums 和 一个目标值 target。找出 nums 中的三个整数，使得它们的和与 target 最接近。返回这三个数的和。假定每组输入只存在唯一答案。

示例：

输入：nums = [-1,2,1,-4], target = 1
输出：2
解释：与 target 最接近的和是 2 (-1 + 2 + 1 = 2) 。

提示：

```
3 <= nums.length <= 10^3
-10^3 <= nums[i] <= 10^3
-10^4 <= target <= 10^4
```

通过次数262,674
提交次数572,143

```
class Solution:
    def threeSumClosest(self, nums: List[int], target: int) -> int:
        nums = sorted(nums)
```

```
        n = len(nums)
        distance = 2**32-1
        for i in range(n-2):
            j= i+1
            k= n-1
            while j<k:
                total = nums[i]+nums[j]+nums[k]
                if total == target:
                    return target
                if abs(target - total )< distance:
                    distance = abs(target - total )
                    result = total
                if total < target:
                    j+=1
                else:
                    k-=1
        return result
```

Tips

1. 和三数之和的解法一样，只不过题目住说明只有唯一解，所以不需要判断重复值，在寻找最接近的过程中加入全局变量不断判断最接近的distance即可

167. 两数之和 II - 输入有序数组.md

给定一个已按照 非递减顺序排列 的整数数组 numbers ，请你从数组中找出两个数满足相加之和等于目标数 target 。

函数应该以长度为 2 的整数数组的形式返回这两个数的下标值。numbers 的下标 从 1 开始计数 ，所以答案数组应当满足 1 <= answer[0] < answer[1] <= numbers.length 。

你可以假设每个输入 只对应唯一的答案 ，而且你 不可以 重复使用相同的元素。

示例 1：

输入：numbers = [2,7,11,15], target = 9
输出：[1,2]
解释：2 与 7 之和等于目标数 9 。因此 index1 = 1, index2 = 2 。

示例 2：

输入：numbers = [2,3,4], target = 6
输出：[1,3]

示例 3：

输入：numbers = [-1,0], target = -1
输出：[1,2]

提示：

```
2 <= numbers.length <= 3 * 104
-1000 <= numbers[i] <= 1000
numbers 按 非递减顺序 排列
-1000 <= target <= 1000
仅存在一个有效答案
```

```python
class Solution:
    def twoSum(self, numbers: List[int], target: int) -> List[int]:
        i = 0
        j = len(numbers)-1
        while i <j:
            if numbers[i] + numbers[j] ==target:
                return i+1,j+1
            elif numbers[i] + numbers[j] > target:
                j-=1
            else:
                i+=1
```

Tips

1. 类比两数之和I，数组无序时一遍遍历一遍用Hash存储历史数据，数组有序时用两指针分别从小到大，从大到小遍历

18. 四数之和.md

给你一个由 n 个整数组成的数组 nums ，和一个目标值 target 。请你找出并返回满足下述全部条件且不重复的四元组 [nums[a], nums[b], nums[c], nums[d]]：

```
0 <= a, b, c, d < n
a、b、c 和 d 互不相同
nums[a] + nums[b] + nums[c] + nums[d] == target
```

你可以按 任意顺序 返回答案 。

示例 1：

输入：nums = [1,0,-1,0,-2,2], target = 0
输出：[[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]]

示例 2：

输入：nums = [2,2,2,2,2], target = 8
输出：[[2,2,2,2]]

提示：

```
1 <= nums.length <= 200
-109 <= nums[i] <= 109
-109 <= target <= 109
```

```python
class Solution:
    def fourSum(self, nums: List[int], target: int) -> List[List[int]]:
        nums = sorted(nums)
        n = len(nums)
        res = []
        if n<4:
            return res
        for i in range(n-3):
            if (i !=0) and (nums[i]==nums[i-1]):
                continue
            for j in range(i+1, n-2):
                if (j!=i+1) and (nums[j]==nums[j-1]):
                    continue
                x=j+1
                y=n-1
                while x<y:
                    if (x!=j+1) and (nums[x]==nums[x-1]):
                        x+=1
                        continue
                    if (y!=n-1) and (nums[y]==nums[y+1]):
                        y-=1
                        continue
                    total = nums[i]+nums[j]+nums[x]+nums[y]
                    if total==target:
                        res.append([nums[i],nums[j],nums[x],nums[y]])
                        x+=1
                        y-=1
                    elif total<target:
                        x+=1
                    else:
                        y-=1
        return res
```

Tips

在三数之和外面家一层循环O（N^ 3）

1. 第一个i指针顺序遍历，第二个指针i+1，剩余两个指针在剩下的空间进行二分搜索
2. 核心在于剔除重复的解

209. 长度最小的子数组.md

给定一个含有 n 个正整数的数组和一个正整数 target 。

找出该数组中满足其和 ≥ target 的长度最小的 连续子数组 [numsl, numsl+1, ..., numsr-1, numsr] ，并返回其长度。如果不存在符合条件的子数组，返回 0 。

示例 1：

输入：target = 7, nums = [2,3,1,2,4,3]
输出：2
解释：子数组 [4,3] 是该条件下的长度最小的子数组。

示例 2：

输入：target = 4, nums = [1,4,4]
输出：1

示例 3：

输入：target = 11, nums = [1,1,1,1,1,1,1,1]
输出：0

提示：

```
1 <= target <= 109
1 <= nums.length <= 105
1 <= nums[i] <= 105
```

进阶：

```
如果你已经实现 O(n) 时间复杂度的解法, 请尝试设计一个 O(n log(n)) 时间复杂度的解法。
```

```
class Solution:
    def minSubArrayLen(self, target: int, nums: List[int]) -> int:
        total = 0
```

```
        res = 2**32
        pointer =0
        for i in range(len(nums)):
            total+=nums[i]
            while total>=target:
                res= min(res, i-pointer+1)
                total -= nums[pointer]
                pointer+=1
        if res ==2**32:
            return 0
        return res
```

Tips

1. 如果使用暴力解法复杂度是O(n^2)
2. 这里使用双指针解法，一个指针正常遍历数组，另外一个指针当当前子序列之和>=target的时候，开始向前移动直到sum<target

26. 删除有序数组中的重复项.md

给你一个有序数组 nums ，请你 原地 删除重复出现的元素，使每个元素 只出现一次 ，返回删除后数组的新长度。

不要使用额外的数组空间，你必须在 原地 修改输入数组 并在使用 O(1) 额外空间的条件下完成。

说明:

为什么返回数值是整数，但输出的答案是数组呢?

请注意，输入数组是以「引用」方式传递的，这意味着在函数里修改输入数组对于调用者是可见的。

你可以想象内部操作如下:

// nums 是以“引用”方式传递的。也就是说，不对实参做任何拷贝
int len = removeDuplicates(nums);

// 在函数里修改输入数组对于调用者是可见的。
// 根据你的函数返回的长度, 它会打印出数组中 该长度范围内 的所有元素。
for (int i = 0; i < len; i++) {
    print(nums[i]);
}

示例 1：

输入：nums = [1,1,2]
输出：2, nums = [1,2]
解释：函数应该返回新的长度 2 ，并且原数组 nums 的前两个元素被修改为 1, 2 。不需要考虑数组中超出新长度后面的元素。

示例 2：

输入：nums = [0,0,1,1,1,2,2,3,3,4]
输出：5, nums = [0,1,2,3,4]
解释：函数应该返回新的长度 5 ， 并且原数组 nums 的前五个元素被修改为 0, 1, 2, 3, 4 。不需要考虑数组中超出新长度后面的元素。

提示：

```
0 <= nums.length <= 3 * 104
-104 <= nums[i] <= 104
nums 已按升序排列
```

```
class Solution:
    def removeDuplicates(self, nums: List[int]) -> int:
        index = 0
        for i in nums[1:]:
            if i != nums[index]:
                index+=1
                nums[index]=i

        return index+1
```

Tips:

双指针解法一个遍历数组，一个只有当当前元素不重复的情况下再向前移动的指针

27. 移除元素.md

给你一个数组 nums 和一个值 val，你需要 原地 移除所有数值等于 val 的元素，并返回移除后数组的新长度。

不要使用额外的数组空间，你必须仅使用 O(1) 额外空间并 原地 修改输入数组。

元素的顺序可以改变。你不需要考虑数组中超出新长度后面的元素。

说明:

为什么返回数值是整数，但输出的答案是数组呢?

请注意，输入数组是以「引用」方式传递的，这意味着在函数里修改输入数组对于调用者是可见的。

你可以想象内部操作如下:

// nums 是以“引用”方式传递的。也就是说，不对实参作任何拷贝
int len = removeElement(nums, val);

```
// 在函数里修改输入数组对于调用者是可见的。
// 根据你的函数返回的长度, 它会打印出数组中 该长度范围内 的所有元素。
for (int i = 0; i < len; i++) {
    print(nums[i]);
}
```

示例 1：

输入：nums = [3,2,2,3], val = 3
输出：2, nums = [2,2]
解释：函数应该返回新的长度 2, 并且 nums 中的前两个元素均为 2。你不需要考虑数组中超出新长度后面的元素。例如，函数返回的新长度为 2 ，而 nums = [2,2,3,3] 或 nums = [2,2,0,0]，也会被视作正确答案。

示例 2：

输入：nums = [0,1,2,2,3,0,4,2], val = 2
输出：5, nums = [0,1,4,0,3]
解释：函数应该返回新的长度 5, 并且 nums 中的前五个元素为 0, 1, 3, 0, 4。注意这五个元素可为任意顺序。你不需要考虑数组中超出新长度后面的元素。

提示：

```
0 <= nums.length <= 100
0 <= nums[i] <= 50
0 <= val <= 100
```

```python
class Solution:
    def removeElement(self, nums: List[int], val: int) -> int:
        index = 0
        for i in nums:
            if i !=val:
                nums[index] =i
                index+=1

        return index
```

Tips:

1. 和前一题删除数组中的重复项思路是一样的，加一个额外的指针

28. 实现 strStr().md

实现 strStr() 函数。

给你两个字符串 haystack 和 needle ，请你在 haystack 字符串中找出 needle 字符串出现的第一个位置（下标从 0 开始）。如果不存在，则返回 -1 。

说明：

当 needle 是空字符串时，我们应当返回什么值呢？这是一个在面试中很好的问题。

对于本题而言，当 needle 是空字符串时我们应当返回 0 。这与 C 语言的 strstr() 以及 Java 的 indexOf() 定义相符。

示例 1：

输入：haystack = "hello", needle = "ll"
输出：2

示例 2：

输入：haystack = "aaaaa", needle = "bba"
输出：-1

示例 3：

输入：haystack = "", needle = ""
输出：0

提示：

```
0 <= haystack.length, needle.length <= 5 * 104
haystack 和 needle 仅由小写英文字符组成
```

1. 朴素方法：遍历每个字符看是否==needle，复杂度O(m*n)

2. KMP算法复杂度O(m+n)

- 生成每个字符的偏移表，即该字符在needle中是否出现，以及出现的位置

```
class Solution:
    def strStr(self, haystack: str, needle: str) -> int:
        def getnext(needle):
            #如果当前不匹配，下一步从needle的第几个位置开始匹配
            l = len(needle)
            next = [-1] * l
            ptr1 = 0 #遍历指针
```

```
            ptr2 = -1 #前缀指针
            while ptr1 < (l-1):
                if ptr2==-1 or needle[ptr1]==needle[ptr2]:
                    ptr1+=1
                    ptr2+=1
                    next[ptr1] = ptr2
                else:
                    ptr2 = next[ptr2]
            return next

        if not needle:
            return 0
        next = getnext(needle)

        i = j = 0
        b = len(haystack)
        a = len(needle)
        while (i < b and j < a):
            if j == -1 or needle[j] == haystack[i]:
                i += 1
                j += 1
            else:
                j = next[j]
        if j == a:
            return i - j
        else:
            return -1
```

1. 生成next指针：
    1. 如果当前不匹配，下一步从needle的第几个位置开始匹配
        1. 第一个位置为-1，用来指示haystack向前移动一位，和needle首位开始匹配
        2. 其余无前缀的位置为0，意味着haystack当前位置和needle首位开始匹配
        3. 有前缀的为前缀位置i，意味着haystack当前位置和needle[i]开始匹配
2. 双指针遍历
    1. 当j=-1，都向前移动一位，haystack从下一位匹配，needle从首位匹配
    2. 当当前位置匹配，向前移动一位
    3. 当不匹配，needle指针根据next会退到对应位置

283. 移动零.md

给定一个数组 nums，编写一个函数将所有 0 移动到数组的末尾，同时保持非零元素的相对顺序。

示例:

输入: [0,1,0,3,12]
输出: [1,3,12,0,0]

说明:

必须在原数组上操作，不能拷贝额外的数组。
尽量减少操作次数。

```
class Solution:
    def moveZeroes(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        n=len(nums)
        index =0
        for i in nums:
            if i!=0:
                nums[index]=i
                index+=1
        nums[index:] = [0] * (n-index)
```

Tips

和剔除有序数组中的重复元素是相同的思路，通过额外的pointer来保留想要保留的元素

31. 下一个排列.md

实现获取 **下一个排列** 的函数，算法需要将给定数字序列重新排列成字典序中下一个更大的排列（即，组合出下一个更大的整数）。

如果不存在下一个更大的排列，则将数字重新排列成最小的排列（即升序排列）。

必须 原地 修改，只允许使用额外常数空间。

示例 1：

```
输入：nums = [1,2,3]
输出：[1,3,2]
```

示例 2：

```
输入：nums = [3,2,1]
输出：[1,2,3]
```

示例 3：

```
输入：nums = [1,1,5]
输出：[1,5,1]
```

示例 4：

```
输入：nums = [1]
输出：[1]
```

提示：

- `1 <= nums.length <= 100`
- `0 <= nums[i] <= 100`

```
class Solution:
    def nextPermutation(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        index = -1
        n=len(nums)
        for i in range(n-2,-1,-1):
            if nums[i] < nums[i+1]:
                index = i
                print(index)
                break

        if index ==-1:
            #nums完全降序排列，转换成升序
            nums[:] = nums[::-1]
            return nums

        #switch bigger value with index
        for i in range(n-1,index,-1):
            if nums[i]> nums[index]:
                nums[i], nums[index] = nums[index],nums[i]
                break

        nums[index+1:] = nums[index+1:][::-1]

        return nums
```

Tips

1. 举个例子4987
2. 从后往前搜索，[j,end]必须是降序，找到第一个非降序i A[i]<A[j]【4，9】
3. 在[j,end]中搜索到min(A[k])>A[i], i和k交换顺序【4和7】
4. 把[j,end]进行reorder变成升序【7489】

344. 反转字符串.md

编写一个函数，其作用是将输入的字符串反转过来。输入字符串以字符数组 s 的形式给出。

不要给另外的数组分配额外的空间，你必须原地修改输入数组、使用 O(1) 的额外空间解决这一问题。

示例 1：

输入：s = ["h","e","l","l","o"]
输出：["o","l","l","e","h"]

示例 2：

输入：s = ["H","a","n","n","a","h"]
输出：["h","a","n","n","a","H"]

提示：

```
1 <= s.length <= 105
s[i] 都是 ASCII 码表中的可打印字符
```

```python
class Solution:
    def reverseString(self, s: List[str]) -> None:
        """
        Do not return anything, modify s in-place instead.
        """
        i=0
        j=len(s)-1
        while i<j:
            s[i],s[j] = s[j],s[i]
            i+=1
            j-=1
```

345. 反转字符串中的元音字母.md

给你一个字符串 s ，仅反转字符串中的所有元音字母，并返回结果字符串。

元音字母包括 'a'、'e'、'i'、'o'、'u'，且可能以大小写两种形式出现。

示例 1：

输入：s = "hello"
输出："holle"

示例 2：

输入：s = "leetcode"
输出："leotcede"

提示：

```
1 <= s.length <= 3 * 105
s 由 可打印的 ASCII 字符组成
```

```python
class Solution:
    def reverseVowels(self, s: str) -> str:
        white_list = {'a','e','i','o','u','A','E','I','O','U'}
        i=0
        j=len(s)-1
        s=list(s)
        while i<j:
            if (s[i] in white_list) and (s[j] in white_list):
                s[i],s[j] = s[j],s[i]
                i+=1
                j-=1
            elif s[i] in white_list:
                j-=1
            elif s[j] in white_list:
                i+=1
            else:
                i+=1
                j-=1

        return ''.join(s)
```

Tips

没啥多说的，都是双指针的解法，类似的还有反转数组，

5. 最长回文子串.md

给你一个字符串 s，找到 s 中最长的回文子串。

示例 1：

输入：s = "babad"
输出："bab"
解释："aba" 同样是符合题意的答案。

示例 2：

输入：s = "cbbd"
输出："bb"

示例 3：

输入：s = "a"
输出："a"

示例 4：

输入：s = "ac"
输出："a"

提示：

```
1 <= s.length <= 1000
s 仅由数字和英文字母（大写和/或小写）组成
```

```python
class Solution:
    def longestPalindrome(self, s: str) -> str:
        l = len(s)
        if l<=1:
            return s
        def get_boundary(left, right):
            while left >=0 and right< l and s[left] ==s[right]:
                left-=1
                right+=1
            return left, right
        start, end =0,0

        for i in range(0, l-1):
            left1, right1 = get_boundary(i, i+1)
            left2, right2 = get_boundary(i, i+2)
```

```
            print(left1, right1)
            print(left2, right2)
            if right1-left1 > end-start:
                start = left1
                end = right1
            if right2-left2> end-start:
                start = left2
                end = right2
        return s[(start+1):end]
```

Tips

动态规划和双指针的解法都是O(n^2)的复杂度，那当然是双指针的解法想起来更容易一些。从中心点出发向左右扩充寻找左右边界即可

647. 回文子串.md

给你一个字符串 s ，请你统计并返回这个字符串中 回文子串 的数目。

回文字符串 是正着读和倒过来读一样的字符串。

子字符串 是字符串中的由连续字符组成的一个序列。

具有不同开始位置或结束位置的子串，即使是由相同的字符组成，也会被视作不同的子串。

示例 1：

输入：s = "abc"
输出：3
解释：三个回文子串: "a", "b", "c"

示例 2：

输入：s = "aaa"
输出：6
解释：6个回文子串: "a", "a", "a", "aa", "aa", "aaa"

提示：

```
1 <= s.length <= 1000
s 由小写英文字母组成
```

1. 动态规划

```
class Solution:
```

```
    def countSubstrings(self, s: str) -> int:
        l = len(s)
        counter = 0
        dp = [[False]* l for _ in range(l)]
        for i in range(l-1,-1,-1):
            for j in range(i,l):
                if s[i]==s[j]:
                    if j-i<=1:
                        counter+=1
                        dp[i][j] = True
                    else:
                        if dp[i+1][j-1]:
                            counter+=1
                            dp[i][j]=True

        return counter
```

Tip

1. $dp[i][j]$是【i，j】之间字符是否是回文字符，每碰到一个true，result+=1
2. 状态转移
    1. s[i]==s[j]:
        1. j-1<=1:result+1
        2. 需要进一步判断内部是否为回文$dp[i+1][j-1]$
    2. s[i]!=s[j]: 没啥可说的继续保持False的常态
3. 初始化都是False即可，因为会从i==j的对角线开始更新所以不需要特殊的初始化
4. 遍历顺序，因为$dp[i][j]$需要用到$dp[i+1][j-1]$ 所以很自然是向左上方矩阵进行更新。

2. 双指针解法

```
class Solution:
    def countSubstrings(self, s: str) -> int:
        l = len(s)

        def count_substring(ptr1,ptr2):
            counter = 0
            while ptr1>=0 and ptr2<l and s[ptr1]==s[ptr2]:
                counter+=1
                ptr1-=1
                ptr2+=1
            return counter

        res = 0
        for i in range(l):
```

```
            n1 = count_substring(i,i)
            res+=n1
            if i<l-1:
                n2 = count_substring(i,i+1)
                res+=n2
        return res
```

Tips

回文串感觉用简单明了的双指针还是更简单一些，只需要判断从每个位置开始（奇/偶）分别有几个回文串就行

680. 验证回文字符串 II.md

给定一个非空字符串 s，最多删除一个字符。判断是否能成为回文字符串。

示例 1:

输入: s = "aba"
输出: true

示例 2:

输入: s = "abca"
输出: true
解释: 你可以删除c字符。

示例 3:

输入: s = "abc"
输出: false

提示:

```
1 <= s.length <= 105
s 由小写英文字母组成
```

```
class Solution:
    def validPalindrome(self, s: str) -> bool:
        def checkPalindrom(left, right):
            while left < right:
                if s[left]==s[right]:
                    left+=1
                    right-=1
                else:
```

```
                return False
            return True
        left = 0
        right = len(s)-1
        while left < right:
            if s[left]==s[right]:
                left+=1
                right-=1
            else:
                check = checkPalindrom(left+1, right) or checkPalindrom(left, right-1)
                return check
        return True
```

Tips

用相同的方式在出现不是回文字符的时候，移一位继续检查

75. 颜色分类.md

给定一个包含红色、白色和蓝色，一共 n 个元素的数组，原地对它们进行排序，使得相同颜色的元素相邻，并按照红色、白色、蓝色顺序排列。

此题中，我们使用整数 0、 1 和 2 分别表示红色、白色和蓝色。

示例 1：

输入：nums = [2,0,2,1,1,0]
输出：[0,0,1,1,2,2]

示例 2：

输入：nums = [2,0,1]
输出：[0,1,2]

示例 3：

输入：nums = [0]
输出：[0]

示例 4：

输入：nums = [1]
输出：[1]

提示：

```
n == nums.length
1 <= n <= 300
nums[i] 为 0、1 或 2
```

进阶：

```
你可以不使用代码库中的排序函数来解决这道题吗?
你能想出一个仅使用常数空间的一趟扫描算法吗?
```

```python
class Solution:
    def sortColors(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        p0 = p1 = 0
        for i in range(len(nums)):
            if nums[i]==1:
                nums[i], nums[p1] = nums[p1], nums[i]
                p1+=1

            elif nums[i]==0:
                nums[i],nums[p0] = nums[p0], nums[i]
                if p0<p1:
                    nums[i], nums[p1] = nums[p1], nums[i]
                p1+=1
                p0+=1
```

Tips

1. 如果是单指针需要遍历两次，第一次把所有的0都移到最前面，第二次吧所有的1都移到0的后面
2. 如果是双指针，一个指向0的位置，一个指向1的位置，唯一需要注意的就是因为0和1本身有位置要求，所以当双指针位置不满足的时候需要进行二次换位

80. 删除有序数组中的重复项 II.md

给你一个有序数组 nums ，请你 原地 删除重复出现的元素，使每个元素 最多出现两次 ，返回删除后数组的新长度。

不要使用额外的数组空间，你必须在 原地 修改输入数组 并在使用 O(1) 额外空间的条件下完成。

说明：

为什么返回数值是整数，但输出的答案是数组呢?

请注意，输入数组是以「引用」方式传递的，这意味着在函数里修改输入数组对于调用者是可见的。

你可以想象内部操作如下:

```
// nums 是以“引用”方式传递的。也就是说，不对实参做任何拷贝
int len = removeDuplicates(nums);

// 在函数里修改输入数组对于调用者是可见的。
// 根据你的函数返回的长度, 它会打印出数组中 该长度范围内 的所有元素。
for (int i = 0; i < len; i++) {
    print(nums[i]);
}
```

示例 1：

输入：nums = [1,1,1,2,2,3]
输出：5, nums = [1,1,2,2,3]
解释：函数应返回新长度 length = 5, 并且原数组的前五个元素被修改为 1, 1, 2, 2, 3 。 不需要考虑数组中超出新长度后面的元素。

示例 2：

输入：nums = [0,0,1,1,1,1,2,3,3]
输出：7, nums = [0,0,1,1,2,3,3]
解释：函数应返回新长度 length = 7, 并且原数组的前五个元素被修改为 0, 0, 1, 1, 2, 3, 3 。 不需要考虑数组中超出新长度后面的元素。

提示：

```
1 <= nums.length <= 3 * 104
-104 <= nums[i] <= 104
nums 已按升序排列
```



```python
class Solution:
    def removeDuplicates(self, nums: List[int]) -> int:
        pos = 1
        counter =1
        cur = nums[0]
        for i in nums[1:]:
            if i != cur:
                nums[pos]= i
                cur = i
```

```
                counter =1
                pos+=1
            elif counter<2:
                nums[pos]=i
                counter +=1
                pos+=1
        return pos
```

844. 比较含退格的字符串.md

给定 s 和 t 两个字符串，当它们分别被输入到空白的文本编辑器后，请你判断二者是否相等。# 代表退格字符。

如果相等，返回 true ；否则，返回 false 。

注意：如果对空文本输入退格字符，文本继续为空。

示例 1：

输入：s = "ab#c", t = "ad#c"
输出：true
解释：S 和 T 都会变成 “ac”。

示例 2：

输入：s = "ab##", t = "c#d#"
输出：true
解释：s 和 t 都会变成 “”。

示例 3：

输入：s = "a##c", t = "#a#c"
输出：true
解释：s 和 t 都会变成 “c”。

示例 4：

输入：s = "a#c", t = "b"
输出：false
解释：s 会变成 “c”，但 t 仍然是 “b”。

提示：

```
1 <= s.length, t.length <= 200
s 和 t 只含有小写字母以及字符 '#'
```

进阶：

> 你可以用 O(N) 的时间复杂度和 O(1) 的空间复杂度解决该问题吗?

1. 简单的栈解法：多了O（N+M）的内存占用

```
class Solution:
    def backspaceCompare(self, s: str, t: str) -> bool:
        def rebuild(s):
            news = []
            for i in range(len(s)):
                if s[i]!='#':
                    news.append(s[i])
                else:
                    try:
                        news.pop()
                    except:
                        continue
            return news

        return rebuild(s)==rebuild(t)
```

2. 技巧从后向前遍历

```
# class Solution:
#     def backspaceCompare(self, s: str, t: str) -> bool:
#         def rebuild(s):
#             news = []
#             for i in range(len(s)):
#                 if s[i]!='#':
#                     news.append(s[i])
#                 else:
#                     try:
#                         news.pop()
#                     except:
#                         continue
#             return news

#         return rebuild(s)==rebuild(t)

class Solution:
    def backspaceCompare(self, s: str, t: str) -> bool:
        i=len(s)-1
        j=len(t)-1
        skips=0
```

```
        skipt=0
        while i>=0 or j>=0:
            while i>=0:
                if s[i]=='#':
                    skips+=1
                    i-=1
                elif skips>0:
                    i-=1
                    skips-=1
                else:
                    break

            while j>=0:
                if t[j]=='#':
                    skipt+=1
                    j-=1
                elif skipt>0:
                    j-=1
                    skipt-=1
                else:
                    break

            if i<0 and j<0:
                return True
            elif i<0 or j<0:
                return False
            elif s[i]!=t[j]:
                return False
            else:
                i-=1
                j-=1
        return True
```

Tips

如果用双指针法，corner case要注意的点真的是多到爆炸。。。

1. 每个数组都是各自遍历直到第一个valid的字符，遍历过程不断统计需要skip的字符（注意一次只能skip一个，如果skip多个可能会skip掉#）
2. 到valid字符时，需要再判断一次pos，当两个pos都>=0时判断数值是否一样，当两个pos有一个>=0时为False，如果两个都<=0这里是True！

88. 合并两个有序数组.md

给你两个按 非递减顺序 排列的整数数组 nums1 和 nums2，另有两个整数 m 和 n ，分别表示 nums1 和 nums2 中的元素数目。

请你 合并 nums2 到 nums1 中，使合并后的数组同样按 非递减顺序 排列。

注意：最终，合并后数组不应由函数返回，而是存储在数组 nums1 中。为了应对这种情况，nums1 的初始长度为 m + n，其中前 m 个元素表示应合并的元素，后 n 个元素为 0 ，应忽略。nums2 的长度为 n 。

示例 1：

输入：nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3
输出：[1,2,2,3,5,6]
解释：需要合并 [1,2,3] 和 [2,5,6] 。
合并结果是 [1,2,2,3,5,6] ，其中斜体加粗标注的为 nums1 中的元素。

示例 2：

输入：nums1 = [1], m = 1, nums2 = [], n = 0
输出：[1]
解释：需要合并 [1] 和 [] 。
合并结果是 [1] 。

示例 3：

输入：nums1 = [0], m = 0, nums2 = [1], n = 1
输出：[1]
解释：需要合并的数组是 [] 和 [1] 。
合并结果是 [1] 。
注意，因为 m = 0 ，所以 nums1 中没有元素。nums1 中仅存的 0 仅仅是为了确保合并结果可以顺利存放到 nums1 中。

提示：

```
nums1.length == m + n
nums2.length == n
0 <= m, n <= 200
1 <= m + n <= 200
-109 <= nums1[i], nums2[j] <= 109
```

解法1. 额外占用O（m+n）的空间

```
class Solution:
    def merge(self, nums1: List[int], m: int, nums2: List[int], n: int) -> None:
```

```
        """
        Do not return anything, modify nums1 in-place instead.
        """
        nums =[]
        i=0
        j=0
        while True:
            if i == m:
                nums += nums2[j:n]
                break
            if j == n:
                nums+= nums1[i:m]
                break

            if nums1[i]<nums2[j]:
                nums.append(nums1[i])
                i+=1
            else:
                nums.append(nums2[j])
                j+=1
            print(nums)
        nums1[:]=nums
```

解法2.直接在nums1上修改，nums1连地方都给你留好了，干啥不用

```
class Solution:
    def merge(self, nums1: List[int], m: int, nums2: List[int], n: int) -> None:
        """
        Do not return anything, modify nums1 in-place instead.
        """
        i = m+n-1
        m-=1
        n-=1

        while True:
            if m <0:
                nums1[:n+1] = nums2[:n+1]
                break
            if n<0:
                break

            if nums1[m] > nums2[n]:
                nums1[i] = nums1[m]
                m-=1
            else:
                nums1[i] = nums2[n]
```

```
                n-=1
            i-=1
```

Tips

1. 以上是O(m+n)的解法，空间占用也是O(m+n)
2. 需要注意的是最后是在原地修改nums1，在原地修改不能直接赋值，需要在原始container中修改nums[:]

904. 水果成篮.md

在一排树中，第 i 棵树产生 tree[i] 型的水果。
你可以从你选择的任何树开始，然后重复执行以下步骤：

> 把这棵树上的水果放进你的篮子里。如果你做不到，就停下来。
> 移动到当前树右侧的下一棵树。如果右边没有树，就停下来。

请注意，在选择一颗树后，你没有任何选择：你必须执行步骤 1，然后执行步骤 2，然后返回步骤 1，然后执行步骤 2，依此类推，直至停止。

你有两个篮子，每个篮子可以携带任何数量的水果，但你希望每个篮子只携带一种类型的水果。

用这个程序你能收集的水果树的最大总量是多少?

示例 1：

输入：[1,2,1]
输出：3
解释：我们可以收集 [1,2,1]。

示例 2：

输入：[0,1,2,2]
输出：3
解释：我们可以收集 [1,2,2]
如果我们从第一棵树开始，我们将只能收集到 [0, 1]。

示例 3：

输入：[1,2,3,2,2]
输出：4
解释：我们可以收集 [2,3,2,2]
如果我们从第一棵树开始，我们将只能收集到 [1, 2]。

示例 4：

输入：[3,3,3,1,2,1,1,2,3,3,4]
输出：5
解释：我们可以收集 [1,2,1,1,2]
如果我们从第一棵树或第八棵树开始，我们将只能收集到 4 棵水果树。

提示：

```
1 <= tree.length <= 40000
0 <= tree[i] < tree.length
```

```python
class Solution:
    def totalFruit(self, fruits: List[int]) -> int:
        from collections import defaultdict
        count = defaultdict(int)
        num = 0
        left = 0
        l = len(fruits)
        for right in range(l):
            count[fruits[right]]+=1
            while len(count)>=3:
                count[fruits[left]]-=1
                if count[fruits[left]] ==0:
                    del count[fruits[left]]
                left +=1
            num = max(num, right - left+1)
        return num
```

Tips

左右双指针问题，和209长度右指针顺序遍历，左指针只有当[left,right]之间的水果>=3种的时候开始i向前移动直到剩下两种

977. 有序数组的平方.md

给你一个按 非递减顺序 排序的整数数组 nums，返回 每个数字的平方 组成的新数组，要求也按 非递减顺序 排序。

示例 1：

输入：nums = [-4,-1,0,3,10]
输出：[0,1,9,16,100]
解释：平方后，数组变为 [16,1,0,9,100]
排序后，数组变为 [0,1,9,16,100]

示例 2：

输入：nums = [-7,-3,2,3,11]
输出：[4,9,9,49,121]

提示：

```
1 <= nums.length <= 104
-104 <= nums[i] <= 104
nums 已按 非递减顺序 排序
```

进阶：

```
请你设计时间复杂度为 O(n) 的算法解决本问题
```

```python
class Solution:
    def sortedSquares(self, nums: List[int]) -> List[int]:
        #找到负数的位置
        neg_pos = -1
        for i,j in enumerate(nums):
            if j <0:
                neg_pos=i
            else:
                break

        result = []
        pos_pos = neg_pos+1
        l = len(nums)
        while neg_pos>=0 or pos_pos < l:
            if neg_pos <0:
                result.append(nums[pos_pos]**2)
                pos_pos+=1

            elif pos_pos >=l:
                result.append(nums[neg_pos]**2)
```

```
                neg_pos-=1

            elif nums[neg_pos]**2 < nums[pos_pos]**2:
                result.append(nums[neg_pos]**2)
                neg_pos -=1
            elif nums[neg_pos]**2 >= nums[pos_pos]**2:
                result.append(nums[pos_pos]**2)
                pos_pos +=1
        return result
```

Tips

这里的双指针是双输入问题，一个是正数部分，一个是负数部分，进行sort merge