简单3_50

1-10

628、三个数的最大乘积

给你一个整型数组 `nums` ，在数组中找出由三个数组成的最大乘积，并输出这个乘积。

> 输入：nums = [1,2,3]
> 输出：6

> 输入：nums = [1,2,3,4]
> 输出：24

> 输入：nums = [-1,-2,-3]
> 输出：-6

题解

1、先排序，然后分别考虑不同情况

- 三个都是正数，最大的乘积就是最后的三个数
- 两个负数，一个正数，最大的就是前两个负数，和最后一位的正数

```
class Solution:
    def maximumProduct(self, nums: List[int]) -> int:
        if len(nums)==3:
            return nums[0]*nums[1]*nums[2]

        nums=sorted(nums)
        pos=nums[-1]*nums[-2]*nums[-3]
        neg=nums[0]*nums[1]*nums[-1]

        return max(pos,neg)
```

快排优化

```
class Solution:
    def maximumProduct(self, nums: List[int]) -> int:
        n=len(nums);
        nums.sort();
        if nums[n-1]<0:
            return nums[n-1]*nums[n-2]*nums[n-3];
        elif nums[n-1]==0:
            return 0;
        else:
             return max(nums[n-1]*nums[n-2]*nums[n-3],nums[n-
1]*nums[0]*nums[1]);
```

2、打擂台

1. 参数定义
    - a,b,c：最大的三个数，a>=b>=c
    - d,e：最小的两个数，d<=e
2. 思路
    - 先令a,b,c为float('-inf'),d,e为float('inf')
    - 当前值为num
    - 更新最大的三个数
        1. 如果num>a，则将a更新为num，b,c顺次更新为之前的a,b
        2. 如果b<num<=a，则将b更新为num，c更新为b
        3. 如果c<num<=b，则将c更新为num
    - 更新最小的两个数：
        1. 如果num<d，则将d更新为num，e更新为d
        2. 如果d<num<=e，则将e更新为num

```python
class Solution:
    def maximumProduct(self, nums: List[int]) -> int:
        a = b = c = float('-inf')
        d = e = float('inf')
        for i, num in enumerate(nums):
            # 更新最大三个数
            if num > a:
                a, b, c = num, a, b
            elif num > b:
                b, c = num, b
            elif num > c:
                c = num
            # 更新最小两个数
            if num < d:
                d, e = num, d
            elif num < e:
                e = num

        return max(d * e * a, a * b * c)
```

637、二叉树的层平均值

给定一个非空二叉树, 返回一个由每层节点平均值组成的数组。

> 输入：
> 3
> / \
> 9 20
> / \
> 15   7
> 输出：[3, 14.5, 11]
> 解释：
> 第 0 层的平均值是 3 , 第1层是 14.5 , 第2层是 11 。因此返回 [3, 14.5, 11] 。

题解

1、BFS

```
class Solution:
    def averageOfLevels(self, root: TreeNode) -> List[float]:
        res = []
        if root is None: return res
        que = [root]
        while que:
            n = len(que)
            Sum = 0
            for _ in range(n):
                cur = que.pop(0)
                Sum += cur.val
                if cur.left: que.append(cur.left)
                if cur.right: que.append(cur.right)
            res.append(Sum / n)
        return res
```

2、DFS

```
class Solution:
    def averageOfLevels(self, root: TreeNode) -> List[float]:
        def DFS(pNode, level):
            if pNode is None: return
            if level < len(Sums):
                Sums[level] += pNode.val
                counts[level] += 1
            else:
                Sums.append(pNode.val)
                counts.append(1)
            DFS(pNode.left, level + 1)
            DFS(pNode.right, level + 1)
        Sums = []
        counts = []
        DFS(root, 0)
        return [Sum / Num for Sum, Num in zip(Sums, counts)]
```

643、子数组最大平均数 I

给定 `n` 个整数，找出平均数最大且长度为 `k` 的连续子数组，并输出该最大平均数。

> 输入：[1,12,-5,-6,50,3], k = 4
> 输出：12.75
> 解释：最大平均数 (12-5-6+50)/4 = 51/4 = 12.75

题解

1、暴力解法，窗移遍历每个平均数，选出平均值的最大值——超时间

```python
class Solution:
    def findMaxAverage(self, nums: List[int], k: int) -> float:
        res=float('-inf')
        for i in range(len(nums)-k+1):
            avg=sum(nums[i:i+k])/k
            if res<avg:
                res=avg

        return res
```

2、简化循环内操作，比较窗口最大值

```python
class Solution:
    def findMaxAverage(self, nums: List[int], k: int) -> float:
        if k==1:
            return max(nums)
        res=sum(nums[0:k])
        sums=res
        for i in range(len(nums)-k):
            sums=sums-nums[i]+nums[i+k]
            res=max(res,sums)

        return res/k
```

645、错误的集合

集合 s 包含从 1 到 n 的整数。不幸的是，因为数据错误，导致集合里面某一个数字复制了成了集合里面的另外一个数字的值，导致集合 丢失了一个数字 并且 有一个数字重复 。

给定一个数组 nums 代表了集合 S 发生错误后的结果。

请你找出重复出现的整数，再找到丢失的整数，将它们以数组的形式返回。

> 输入：nums = [1,2,2,4]
> 输出：[2,3]

> 输入：nums = [1,1]
> 输出：[1,2]

题解

1、先找到重复数字，再判断缺失数字

```python
class Solution:
    def findErrorNums(self, nums: List[int]) -> List[int]:
        res=[]
        res.append(sum(nums)-sum(set(nums)))
        nums=sorted(set(nums))
        for i in range(len(nums)):
            if nums[i]!=i+1:
                res.append(i+1)
```

```
                break

        if len(res)==1:
            res.append(len(nums)+1)

        return res
```

优化

```
class Solution:
    def findErrorNums(self, nums: List[int]) -> List[int]:
        return [sum(nums) - sum(set(nums)), sum(list(range(1, len(nums) + 1))) -
sum(set(nums))]
```

2、位运算

1. 先把1到n的所有数和数组nums中的所有数全部异或，可以得到结果xor就是两个待求数的异或
2. 通过位运算xor & - xor取得xor的最右边的1，命名为rightOne，也就是除了该位为1外其他位均为0（两个不同的数的异或结果里至少会有1位为1）
3. 将从1到n的所有数和nums中的所有数根据与(&)上rightOne后是否为0分成两组，待求的两个数必会分开在两个组
4. 求得一个组的异或结果为a，a即为待求的两个数之一，另一个数即为a ^ xor
5. 题目要求重复数在前，缺失数在后，因此还要遍历一下nums数组，如果a在数组中，说明a是重复数，返回[a, a ^ xor]；否则返回[a ^ xor, a]

```
class Solution:
    def findErrorNums(self, nums: List[int]) -> List[int]:
        xor = 0
        for i, num in enumerate(nums):
            xor ^= (i + 1) ^ num
        rightOne = xor & - xor

        a = 0
        for i, num in enumerate(nums):
            if ((i + 1) & rightOne) != 0:
                a ^= (i + 1)
            if (num & rightOne) != 0:
                a ^= num

        return [a, a ^ xor] if a in nums else [a ^ xor, a]
```

653、两数之和IV-输入BST

给定一个二叉搜索树和一个目标结果，如果 BST 中存在两个元素且它们的和等于给定的目标结果，则返回 true。

```
输入:
 5
/ \
3   6
/ \   \
2   4   7

Target = 9

输出: True
```

```
输入:
 5
/ \
3   6
/ \   \
2   4   7

Target = 28

输出: False
```

题解

1、DFS得到搜索树的数组，在使用双指针搜索是否存在两数之和

```
class Solution:
    def findTarget(self, root: TreeNode, k: int) -> bool:
        num=[]
        def DFS(root):
            if not root:return
            num.append(root.val)
            if root.left:DFS(root.left)
            if root.right:DFS(root.right)

        DFS(root)
        num=sorted(num)
        left,right=0,len(num)-1
        while left<right:
            if num[left]+num[right]==k:
                return True
            elif num[left]+num[right]>k:
                right-=1
            else:
                left+=1

        return False
```

2、遍历树查找k-var是否存在

```
class Solution:
    def findTarget(self, root: TreeNode, k: int) -> bool:
        stack = [root]
        s = set()
```

```
        while stack:
            root = stack.pop()
            if k - root.val in s:
                return True
            else:
                s.add(root.val)
            if root.left:
                stack.append(root.left)
            if root.right:
                stack.append(root.right)
        return False
```

657、机器人能否返回原点

在二维平面上，有一个机器人从原点 (0, 0) 开始。给出它的移动顺序，判断这个机器人在完成移动后是否在 (0, 0) 处结束。

移动顺序由字符串表示。字符 move[i] 表示其第 i 次移动。机器人的有效动作有 R（右），L（左），U（上）和 D（下）。如果机器人在完成所有动作后返回原点，则返回 true。否则，返回 false。

注意：机器人"面朝"的方向无关紧要。 "R" 将始终使机器人向右移动一次，"L" 将始终向左移动等。此外，假设每次移动机器人的移动幅度相同。

> 输入: "UD"
> 输出: true
> 解释：机器人向上移动一次，然后向下移动一次。所有动作都具有相同的幅度，因此它最终回到它开始的原点。因此，我们返回 true。

> 输入: "LL"
> 输出: false
> 解释：机器人向左移动两次。它最终位于原点的左侧，距原点有两次 "移动" 的距离。我们返回 false，因为它在移动结束时没有返回原点。

题解

1、分类判断，不同移动做不同标记

```
class Solution:
    def judgeCircle(self, moves: str) -> bool:
        res=[0,0]
        for i in moves:
            if i == 'R':
                res[0]+=1
            elif i == 'L':
                res[0]-=1
            elif i == 'U':
                res[1]+=1
            else:
                res[1]-=1

        return True if res==[0,0] else False
```

2、判断移动操作出现的次数

```
class Solution:
    def judgeCircle(self, moves: str) -> bool:
        h=collections.Counter(moves)
        if h['U']==h['D'] and h['L']==h['R']:
            return True
        return False
```

3、哈希字典

```
class Solution:
    def judgeCircle(self, moves: str) -> bool:
        x, y = 0, 0
        dct = {"U": (-1, 0), "D": (1, 0), "L": (0, -1), "R": (0, 1)}
        for i in moves:
            a, b = dct[i]
            x, y = x+a, y+b
        return x == 0 and y == 0
```

661、图片平滑器

包含整数的二维矩阵 M 表示一个图片的灰度。你需要设计一个平滑器来让每一个单元的灰度成为平均灰度 (向下舍入)，平均灰度的计算是周围的8个单元和它本身的值求平均，如果周围的单元格不足八个，则尽可能多的利用它们。

> 输入:
> [[1,1,1],
> [1,0,1],
> [1,1,1]]
> 输出:
> [[0, 0, 0],
> [0, 0, 0],
> [0, 0, 0]]
> 解释:
> 对于点 (0,0), (0,2), (2,0), (2,2): 平均(3/4) = 平均(0.75) = 0
> 对于点 (0,1), (1,0), (1,2), (2,1): 平均(5/6) = 平均(0.83333333) = 0
> 对于点 (1,1): 平均(8/9) = 平均(0.88888889) = 0

题解

1、暴力求解

记录坐标，然后对坐标进行范围判定，保留有效坐标，然后按顺序为新矩阵赋值即可。

```
class Solution:
    def imageSmoother(self, M: List[List[int]]) -> List[List[int]]:
        row, col = len(M), len(M[0])
        result = [[0 for i in range(col)] for j in range(row)]
        for k in range(row):
            for l in range(col):
```

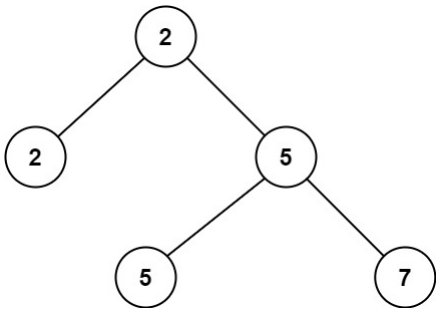
```
                coordinate = [[k-1, l-1], [k-1, l], [k-1, l+1], [k, l-1], [k,l],
[k, l+1],
                [k+1, l-1], [k+1, l], [k+1, l+1]]
                coordinate_valid = []
                for m in coordinate:
                    if 0 <= m[0] < row and 0 <= m[1] < col:
                        coordinate_valid.append(m)
                for n in coordinate_valid:
                    result[k][l] += M[n[0]][n[1]]
                result[k][l] = int(result[k][l]/len(coordinate_valid))
        return result
```

671、二叉树中第二小的节点

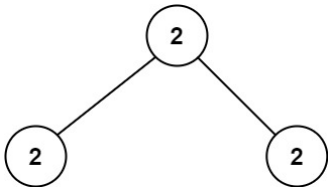
给定一个非空特殊的二叉树，每个节点都是正数，并且每个节点的子节点数量只能为 2 或 0。如果一个节点有两个子节点的话，那么该节点的值等于两个子节点中较小的一个。

更正式地说，root.val = min(root.left.val, root.right.val) 总成立。

给出这样的一个二叉树，你需要输出所有节点中的第二小的值。如果第二小的值不存在的话，输出 -1 。




```
输入：root = [2,2,5,null,null,5,7]
输出：5
解释：最小的值是 2 ，第二小的值是 5 。
```




```
输入：root = [2,2,2]
输出：-1
解释：最小的值是 2，但是不存在第二小的值。
```

题解

1、DFS前序遍历的到数组，set后排序

```
class Solution:
    def findSecondMinimumValue(self, root: TreeNode) -> int:
        nums=[]
        def DFS(root):
```

```
            if not root: return
            nums.append(root.val)
            if root.left:DFS(root.left)
            if root.right:DFS(root.right)

        DFS(root)
        nums=sorted(set(nums))

        if len(nums)>1:
            return nums[1]
        else:
            return -1
```

2、前序遍历

最小的元素一定的根节点，所以我们只要找到比根节点大的节点，直接返回就行了，更不用继续遍历当前节点下面的子节点，因为子节点的值不可能比它还小。

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def findSecondMinimumValue(self, root: TreeNode) -> int:
        def helper(root, val):
            if not root:
                return -1

            if root.val > val:
                return root.val
            left = helper(root.left, val)
            right = helper(root.right, val)
            if left < 0: return right
            if right < 0: return left
            return min(left, right)

        return helper(root, root.val)
```

674、最长连续递增序列

给定一个未经排序的整数数组，找到最长且 连续递增的子序列，并返回该序列的长度。

连续递增的子序列 可以由两个下标 l 和 r（l < r）确定，如果对于每个 l <= i < r，都有 nums[i] < nums[i + 1] ，那么子序列 [nums[l], nums[l + 1], ..., nums[r - 1], nums[r]] 就是连续递增子序列。

> 输入：nums = [1,3,5,4,7]
> 输出：3
> 解释：最长连续递增序列是 [1,3,5], 长度为3。
> 尽管 [1,3,5,7] 也是升序的子序列，但它不是连续的，因为 5 和 7 在原数组里被 4 隔开。

> 输入：nums = [2,2,2,2,2]
> 输出：1
> 解释：最长连续递增序列是 [2], 长度为1。

题解

1、判断最长子序列，大于加一，比较一下，小于就重新开始计数

```python
class Solution:
    def findLengthOfLCIS(self, nums: List[int]) -> int:
        res=1
        l=1
        for i in range(len(nums)-1):
            if nums[i]-nums[i+1]<0:
                l+=1
                res=max(res,l)
            else:
                l=1

        return res
```

2、双指针

```python
class Solution:
    def findLengthOfLCIS(self, nums: List[int]) -> int:
        n = len(nums)
        if n <= 1:
            return n

        ans = 0

        left = 0
        right = 1

        while right < n:
            while right < n and nums[right-1] < nums[right]:
                right += 1
            ans = max(ans, right - left)
            left = right
            right += 1

        return ans
```

3、动态规划

```python
class Solution:
    def findLengthOfLCIS(self, nums: List[int]) -> int:
        if len(nums) == 0:
            return 0
        A = [0 for i in nums]
        A[0] = 1
        for i in range(1, len(nums)):
            if nums[i] > nums[i-1]:
                A[i] = A[i-1] +1
            else:
                A[i] = 1
        return max(A)
```

680、验证回文字符串Ⅱ

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。

> 输入: "aba"
> 输出: True

> 输入: "abca"
> 输出: True
> 解释: 你可以删除c字符。

题解

1、依次删除一个字符，判断是否为回文字符串——超时间

```python
class Solution:
    def validPalindrome(self, s: str) -> bool:
        if s==s[::-1]: return True
        if s[1:]==s[1:][::-1] : return True
        if s[:-1]==s[:-1][::-1] : return True

        for i in range(1,len(s)-1):
            v=s[0:i]+s[i+1:]
            if v==v[::-1]:
                return True

        return False
```

2、双指针，遇到不同字符，删除判断是否为回文字符串

```
class Solution(object):
    def validPalindrome(self, s):
        isPalindrome = lambda x : x == x[::-1]
        left, right = 0, len(s) - 1
        while left <= right:
            if s[left] == s[right]:
                left += 1
                right -= 1
            else:
                return isPalindrome(s[left + 1 : right + 1]) or
isPalindrome(s[left: right])
        return True
```

优化

```
class Solution:
    def validPalindrome(self, s: str) -> bool:
        left,right=0,len(s)-1
        while left<=right:
            if s[left]!=s[right]:
                return s[left+1:right+1]==s[left+1:right+1][::-1] or
s[left:right]==s[left:right][::-1]

            left+=1
            right-=1

        return True
```

11-20

682、棒球比赛

你现在是一场采用特殊赛制棒球比赛的记录员。这场比赛由若干回合组成，过去几回合的得分可能会影响以后几回合的得分。

比赛开始时，记录是空白的。你会得到一个记录操作的字符串列表 ops，其中 ops[i] 是你需要记录的第 i 项操作，ops 遵循下述规则：

1. 整数 x - 表示本回合新获得分数 x
2. "+" - 表示本回合新获得的得分是前两次得分的总和。题目数据保证记录此操作时前面总是存在两个有效的分数。
3. "D" - 表示本回合新获得的得分是前一次得分的两倍。题目数据保证记录此操作时前面总是存在一个有效的分数。
4. "C" - 表示前一次得分无效，将其从记录中移除。题目数据保证记录此操作时前面总是存在一个有效的分数。

请你返回记录中所有得分的总和。

```
输入：ops = ["5","2","C","D","+"]
输出：30
解释：
"5" - 记录加 5 ，记录现在是 [5]
"2" - 记录加 2 ，记录现在是 [5, 2]
"C" - 使前一次得分的记录无效并将其移除，记录现在是 [5].
"D" - 记录加 2 * 5 = 10 ，记录现在是 [5, 10].
"+" - 记录加 5 + 10 = 15 ，记录现在是 [5, 10, 15].
所有得分的总和 5 + 10 + 15 = 30
```

```
输入：ops = ["5","-2","4","C","D","9","+","+"]
输出：27
解释：
"5" - 记录加 5 ，记录现在是 [5]
"-2" - 记录加 -2 ，记录现在是 [5, -2]
"4" - 记录加 4 ，记录现在是 [5, -2, 4]
"C" - 使前一次得分的记录无效并将其移除，记录现在是 [5, -2]
"D" - 记录加 2 * -2 = -4 ，记录现在是 [5, -2, -4]
"9" - 记录加 9 ，记录现在是 [5, -2, -4, 9]
"+" - 记录加 -4 + 9 = 5 ，记录现在是 [5, -2, -4, 9, 5]
"+" - 记录加 9 + 5 = 14 ，记录现在是 [5, -2, -4, 9, 5, 14]
所有得分的总和 5 + -2 + -4 + 9 + 5 + 14 = 27
```

题解

1、if 操作

```python
class Solution:
    def calPoints(self, ops: List[str]) -> int:
        res=[]
        for i in ops:
            if i == 'C':
                res=res[:-1]            #res.pop()
            elif i == 'D':
                res.append(res[-1]*2)
            elif  i == '+':
                res.append(res[-1]+res[-2])
            else:
                res.append(int(i))

        return sum(res)
```

2、字典判断

用字典的写法的优势：即使有再多的条件也可以保证程序的可读性

```python
class Solution(object):
    def calPoints(self, ops):
        score = [0,0]
        for c in ops:
            if c.lstrip('-').isdigit():
                score.append(int(c))
            elif score:
                score = {
                    "+":score + [score[-2] + score[-1]],
                    "C":score[:-1],
                    "D":score + [2*score[-1]]
                    }.get(c)
        return sum(score)
```

693、交替位二进制数

给定一个正整数，检查它的二进制表示是否总是 0、1 交替出现：换句话说，就是二进制表示中相邻两位的数字永不相同。

> 输入：n = 5
> 输出：true
> 解释：5 的二进制表示是：101

> 输入：n = 7
> 输出：false
> 解释：7 的二进制表示是：111.

题解

1、字符操作，

```python
class Solution:
    def hasAlternatingBits(self, n: int) -> bool:
        s=str(bin(n))[2:]
        for i in range(len(s)-1):
            if s[i]==s[i+1]:
                return False

        return True
```

2、位运算

```python
class Solution:
    def hasAlternatingBits(self, n: int) -> bool:
        flag = n & 1        #n & 1 为n的二进制的最后一位数字 为0或者1
        n = n >> 1
        while n > 0:
            if flag == n & 1:
                return False
            flag = n & 1
            n = n >> 1
        return True
```

3、右移后异或，判断是否全为1

```python
class Solution(object):
    def hasAlternatingBits(self, n):
        temp = n ^ (n >> 1)
        return temp & (temp + 1) == 0
```

4、除余，比较

```python
class Solution:
    def hasAlternatingBits(self, n: int) -> bool:
        while n:
            prenum = (n//2) & 1
            nownum = n%2
            if prenum == nownum:
                return False
            n>>=1
        return True
```

696、计数二进制字串

给定一个字符串 s，计算具有相同数量 0 和 1 的非空（连续）子字符串的数量，并且这些子字符串中的所有 0 和所有 1 都是连续的。

重复出现的子串要计算它们出现的次数。

> 输入：“00110011”
> 输出：6
> 解释：有6个子串具有相同数量的连续1和0：“0011”，“01”，“1100”，“10”，“0011” 和 “01”。
>
> 请注意，一些重复出现的子串要计算它们出现的次数。
>
> 另外，“00110011”不是有效的子串，因为所有的0（和1）没有组合在一起。

> 输入：“10101”
> 输出：4
> 解释：有4个子串：“10”，“01”，“10”，“01”，它们具有相同数量的连续1和0。

题解

1、正则表达式

```
class Solution:
    def countBinarySubstrings(self, s: str) -> int:
        # 正则表达式提取连续0或者1字符数组
        m = re.findall('[0]+|[1]+',s)
        # 取m中相邻元素的长度小值为000..1111...的个数， 然后全部取sum就是全部
        return sum([min(len(m[i]),len(m[i+1])) for i in range(len(m)-1)])

输入：00110011
    m=['00', '11', '00', '11']
```

2、一次遍历，每次循环记录相同元素出现次数，一旦改变，记录最小元素

```
class Solution:
    def countBinarySubstrings(self, s: str) -> int:
        seq = [0, 1]
        res = []
        for i in range(1, len(s)):
            if s[i] == s[i-1]:
                seq[1] += 1
            else:
                res.append(min(seq))
                seq[0] = seq[1]
                seq[1] = 1
        res.append(min(seq))
        return sum(res)
```

3、中心扩散法

从符合条件的两个相连位置向两边扩散

```
class Solution:
    def countBinarySubstrings(self, s: str) -> int:
        count = 0
        for i in range(len(s) - 1):
            if int(s[i]) ^ int(s[i + 1]) == 1:
                index = 0
                while i - index >=0 and i + 1 + index <= len(s) - 1:
                    if s[i - index] == s[i] and s[i + 1] == s[i + 1 + index]:
                        count += 1
                        index += 1
                    else:
                        break
        return count
```

697、数组的度

给定一个非空且只包含非负数的整数数组 nums，数组的度的定义是指数组里任一元素出现频数的最大值。

你的任务是在 nums 中找到与 nums 拥有相同大小的度的最短连续子数组，返回其长度。

```
输入：[1, 2, 2, 3, 1]
输出：2
解释：
输入数组的度是2，因为元素1和2的出现频数最大，均为2.
连续子数组里面拥有相同度的有如下所示：
[1, 2, 2, 3, 1], [1, 2, 2, 3], [2, 2, 3, 1], [1, 2, 2], [2, 2, 3], [2, 2]
最短连续子数组[2, 2]的长度为2，所以返回2.
```

题解

1、计数，求索引，得到最小索引

```python
class Solution:
    def findShortestSubArray(self, nums: List[int]) -> int:
        dic=collections.Counter(nums)
        maxn=dic[max(dic,key=dic.get)]
        num=[]
        for i in dic:
            if dic[i]==maxn:
                num.append(i)

        res=float('inf')

        for i in num:
            idx=[x for x,j in enumerate(nums) if j==i]
            res=min(res,idx[-1]-idx[0]+1)

        return res
```

717、1比特与2比特字符

有两种特殊字符。第一种字符可以用一比特0来表示。第二种字符可以用两比特(10 或 11)来表示。

现给一个由若干比特组成的字符串。问最后一个字符是否必定为一个一比特字符。给定的字符串总是由0结束。

```
输入:
bits = [1, 0, 0]
输出: True
解释:
唯一的编码方式是一个两比特字符和一个一比特字符。所以最后一个字符是一比特字符。
```

```
输入:
bits = [1, 1, 1, 0]
输出: False
解释:
唯一的编码方式是两比特字符和两比特字符。所以最后一个字符不是一比特字符。
```

题解

1、标志位

```
class Solution:
    def isOneBitCharacter(self, bits: List[int]) -> bool:
        i=0
        res=False
        while i<len(bits):
            if bits[i]==1:
                res=False
                i+=2
            else:
                res=True
                i+=1

        return res
```

优化

```
class Solution:
    def isOneBitCharacter(self, bits: List[int]) -> bool:
        n = len(bits) - 1
        i = 0
        while i < n:
            if bits[i] == 1:
                i += 2
            else:
                i += 1
        return i == n
```

2、迭代

```
class Solution:
    def isOneBitCharacter(self, bits: List[int]) -> bool:
        for inx, i in enumerate(bits):
            if i:
                bits[inx] = bits[inx+1] = None
        return bits[-1] == 0
```

3、递归

```
class Solution:
    def isOneBitCharacter(self, bits: List[int]) -> bool:
        return False if not bits or bits == [[1, 0]] else True if bits == [0]
else self.isOneBitCharacter(bits[1:]) if bits[0] == 0 else
self.isOneBitCharacter(bits[2:])
```

720、字典中的最长单词

给出一个字符串数组words组成的一本英语词典。从中找出最长的一个单词，该单词是由words词典中其他单词逐步添加一个字母组成。若其中有多个可行的答案，则返回答案中字典序最小的单词。

若无答案，则返回空字符串。

```
输入：
words = ["w","wo","wor","worl", "world"]
输出："world"
解释：
单词"world"可由"w", "wo", "wor", 和 "worl"添加一个字母组成。
```

```
输入：
words = ["a", "banana", "app", "appl", "ap", "apply", "apple"]
输出："apple"
解释：
"apply"和"apple"都能由词典中的单词组成。但是"apple"的字典序小于"apply"。
```

题解

1、字典排序

```
class Solution:
    def longestWord(self, words: List[str]) -> str:
        words.sort()
        maxword =""
        se =set()
        for word in words:
            if len(word) == 1 or word[:-1] in se:
                se.add(word)
                if len(word) > len(maxword):
                    maxword =word
        return maxword
```

2、前缀树

1. 建议首先看leetcode 208题：实现Trie（前缀树）。
2. 首先建立TrieNode类，它是前缀树的节点，有两个属性：end为True时表示以该节点为结尾的单词存在；children表示该节点的子节点。
3. 在二的基础上建立Trie类，里面定义两个方法。其中insert()用于向树中添加单词；search()用于检索该单词的前n项子字符串是否都在树中，举个例子就是：当'moffy'中的'm'、'mo'、'mof'、'moff'、'moffy'都在树中时，则返回True。
4. 所有类建好后，遍历两次words。第一次遍历向树中添加所有单词，构造好前缀树；第二次遍历，在构造好的前缀树中找到满足条件的单词，注意最后结果是：保证单词长度最长；当单词长度一样时，选择字典序靠前的单词。

```
class Solution:
    def longestWord(self, words: List[str]) -> str:
        res=''
        trie=Trie()
        for word in words:
            trie.insert(word)
        for word in words:
            if trie.search(word):
                if len(word) > len(res):
                    res=word
                elif len(word)==len(res) and word < res:
                    res=word
        return res
```

```python
class TrieNode:
    def __init__(self):
        self.end=False
        self.children=collections.defaultdict(TrieNode)

class Trie:
    def __init__(self):
        self.root=TrieNode()

    def insert(self, word: str) -> None:
        node=self.root
        for s in word:
            node=node.children[s]
        node.end=True

    def search(self, word: str) -> bool:
        node=self.root
        for s in word:
            node=node.children.get(s)
            if node is None or not node.end:
                return False
        return True
```

724、寻找数组的中心下标

给你一个整数数组 nums，请编写一个能够返回数组 “中心下标” 的方法。

数组 中心下标 是数组的一个下标，其左侧所有元素相加的和等于右侧所有元素相加的和。

如果数组不存在中心下标，返回 -1 。如果数组有多个中心下标，应该返回最靠近左边的那一个。

注意：中心下标可能出现在数组的两端。

> 输入：nums = [1, 7, 3, 6, 5, 6]
> 输出：3
> 解释：
> 中心下标是 3 。
> 左侧数之和 (1 + 7 + 3 = 11)，
> 右侧数之和 (5 + 6 = 11) ，二者相等。

> 输入：nums = [1, 2, 3]
> 输出：-1
> 解释：
> 数组中不存在满足此条件的中心下标。

> 输入：nums = [2, 1, -1]
> 输出：0
> 解释：
> 中心下标是 0 。
> 下标 0 左侧不存在元素，视作和为 0 ；
> 右侧数之和为 1 + (-1) = 0 ，二者相等。

题解

1、暴力解法，依次计算左右两边的和

```
class Solution:
    def pivotIndex(self, nums: List[int]) -> int:
        if sum(nums[1:])==0:
            return 0

        for i in range(1,len(nums)-1):
            if sum(nums[:i])==sum(nums[i+1:]):
                return i

        if sum(nums[:-1])==0:
            return len(nums)-1
        else:
            return -1
```

2、preSum

1. 它的计算方法是从左向右遍历数组，当遍历到数组的 i 位置时，preSum表示 i 位置左边的元素之和。
2. 我们提前计算出所有元素之和 sums，那么 sums - preSum - nums[i] 就是 i 位置右边元素之和。
    1. 如果 preSum == sums_ - preSum - nums[i]，那么 i 就是满足题目含义的「中心索引」位置。
    2. 如果遍历完数组，都没有发现满足题意的「中心索引」，那么返回 -1

```
class Solution(object):
    def pivotIndex(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        N = len(nums)
        sums_ = sum(nums)
        preSum = 0
        for i in range(N):
            if preSum == sums_ - preSum - nums[i]:
                return i
            preSum += nums[i]
        return -1
```

728、自除数

自除数 是指可以被它包含的每一位数除尽的数。

例如，128 是一个自除数，因为 128 % 1 == 0，128 % 2 == 0，128 % 8 == 0。

还有，自除数不允许包含 0 。

给定上边界和下边界数字，输出一个列表，列表的元素是边界（含边界）内所有的自除数。

```
输入：
上边界left = 1, 下边界right = 22
输出： [1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 15, 22]
```

题解

1、变成字符在依次判断

```python
class Solution:
    def selfDividingNumbers(self, left: int, right: int) -> List[int]:
        res=[]
        nums=range(left,right+1)
        for i in nums:
            s=str(i)
            t=0
            for j in s:
                if j!='0' and i%int(j)==0:
                    t+=1
                else:
                    break
                if t==len(s):
                    res.append(i)

        return res
```

709、转换成小写字母

给你一个字符串 `s` ，将该字符串中的大写字母转换成相同的小写字母，返回新的字符串。

```
输入：s = "Hello"
输出："hello"
```

```
输入：s = "here"
输出："here"
```

题解

1、lower() 和 upper() 函数

```python
class Solution:
    def toLowerCase(self, s: str) -> str:
        return s.lower()
```

2、Ascii码

```
'A' - 'Z' 对应的 ascii 是 65 - 90;
'a' - 'z' 对应的 ascii 是 97 - 122;
大小字母转换相差32，解题只要记住ord(),chr()函数即可
class Solution:
    def toLowerCase(self, str: str) -> str:
        s = []
        for i in str:
            if  65 <= ord(i) <= 90:
                s.append(chr(ord(i) + 32))
            else:
                s.append(i)
        return ''.join(s)
```

3、字典，挨个替换

```
字典法编写有点繁琐，不过测试效率也是最高的
class Solution:
    def toLowerCase(self, str: str) -> str:
        dic = {'A':'a', 'B':'b', 'C':'c', 'D':'d', 'E':'e', 'F':'f',
               'G':'g', 'H':'h', 'I':'i', 'J':'j', 'K':'k', 'L':'l',
               'M':'m', 'N':'n', 'O':'o','P':'p', 'Q':'q', 'R':'r',
               'S':'s', 'T':'t', 'U':'u', 'V':'v', 'W':'w', 'X':'x',
               'Y':'y', 'Z':'z'}
        s = []
        for i in str:
            if dic.get(i):
                s.append(dic[i])
            else:
                s.append(i)
        return ''.join(s)
```

744、寻找比目标字母大的最小字母

给你一个排序后的字符列表 letters ，列表中只包含小写英文字母。另给出一个目标字母 target，请你寻找在这一有序列表里比目标字母大的最小字母。

在比较时，字母是依序循环出现的。举个例子：

如果目标字母 target = 'z' 并且字符列表为 letters = ['a', 'b']，则答案返回 'a'

> 输入:
> letters = ["c", "f", "j"]
> target = "g"
> 输出: "j"
>
> 输入:
> letters = ["c", "f", "j"]
> target = "j"
> 输出: "c"
>
> 输入:
> letters = ["c", "f", "j"]
> target = "k"

> 输出：`"c"`

题解

1、直接比较输出，

```
class Solution:
    def nextGreatestLetter(self, letters: List[str], target: str) -> str:
        for i in letters:
            if i>target:
                return i

        return letters[0]
```

2、二分查找

```
class Solution:
    def nextGreatestLetter(self, letters: List[str], target: str) -> str:
        length = len(letters)
        left = 0
        right = length-1
        while(left<=right):
            mid = (left+right)//2
            if letters[mid]>target:
                right = mid-1
            else:
                left = mid+1
        if left == length:
            return letters[0]
        else:
            return letters[left]
```

21-30

746、寻找最小花费爬楼梯

数组的每个下标作为一个阶梯，第 i 个阶梯对应着一个非负数的体力花费值 cost[i]（下标从 0 开始）。

每当你爬上一个阶梯你都要花费对应的体力值，一旦支付了相应的体力值，你就可以选择向上爬一个阶梯或者爬两个阶梯。

请你找出达到楼层顶部的最低花费。在开始时，你可以选择从下标为 0 或 1 的元素作为初始阶梯。

> 输入：cost = [10, 15, 20]
> 输出：15
> 解释：最低花费是从 cost[1] 开始，然后走两步即可到阶梯顶，一共花费 15 。

> 输入：cost = [1, 100, 1, 1, 1, 100, 1, 1, 100, 1]
> 输出：6
> 解释：最低花费方式是从 cost[0] 开始，逐个经过那些 1 ，跳过 cost[3] ，一共花费 6 。

题解

1、动态规划

1. dp[i]：代表登上每节台阶所花费的最小体力值
2. dp[i] = min(dp[i-1] , dp[i-2]) + cost[i]
3. return min(dp[-1],dp[-2])

```
class Solution:
    def minCostClimbingStairs(self, cost: List[int]) -> int:
        dp=[0]*len(cost)
        dp[0]=cost[0]
        dp[1]=cost[1]
        for i in range(2,len(cost)):
            dp[i]=min(dp[i-1],dp[i-2])+cost[i]

        return min(dp[-1],dp[-2])
```

空间优化

```
class Solution:
    def minCostClimbingStairs(self, cost: List[int]) -> int:
        a, b = cost[0], cost[1]
        for i in cost[2:]:
            a, b = b, min(a+i, b+i)
        return min(a, b)
```

2、递归+深度优先搜索

- 递归停止条件
    1. 只有一个元素情况（返回0）

    ```
    if len(cost) == 1:
        return 0
    ```

    2. 只有两个元素情况（返回cost的最小值）

    ```
    if len(cost) == 2:
        return min(cost)
    ```

- 返回以下代码的最小值：
    - 爬一个阶梯（可以理解为[0]+cost，必须从索引为0的元素作为初始阶梯）：

    ```
    cost[0]+self.minCostClimbingStairs(cost[1:])
    ```

    - 爬两个阶梯（可以理解为[0]+cost，必须从索引为0的元素作为初始阶梯）：

    ```
    cost[1]+self.minCostClimbingStairs(cost[2:])
    ```

```python
class Solution:
    def minCostClimbingStairs(self, cost: List[int]) -> int:
        if len(cost) == 1:
            return 0
        if len(cost) == 2:
            return min(cost)
        return min(cost[0]+self.minCostClimbingStairs(cost[1:]),
cost[1]+self.minCostClimbingStairs(cost[2:]))
```

747、至少是其他数字两倍的最大数

给你一个整数数组 nums ， 其中总是存在 唯一的 一个最大整数 。

请你找出数组中的最大元素并检查它是否 至少是数组中每个其他数字的两倍 。如果是，则返回 最大元素的下标 ，否则返回 -1 。

> 输入：nums = [3,6,1,0]
> 输出：1
> 解释：6 是最大的整数，对于数组中的其他整数，6 大于数组中其他元素的两倍。6 的下标是 1 ，所以返回 1 。

> 输入：nums = [1,2,3,4]
> 输出：-1
> 解释：4 没有超过 3 的两倍大，所以返回 -1 。

题解

1、选出最大，遍历二倍是否大于最大值

```python
class Solution:
    def dominantIndex(self, nums: List[int]) -> int:
        maxn=max(nums)
        sor=sorted(nums)
        for i in range(len(sor)-1):
            if sor[i]*2>maxn:
                return -1

        return nums.index(maxn)
```

优化操作步骤

```python
class Solution:
    def dominantIndex(self, nums: List[int]) -> int:
        if len(nums)==1:return 0
        sor=sorted(nums)
        if sor[-2]*2>sor[-1]:
            return -1
        else:
            return nums.index(sor[-1])
```

748、最短补全词

给定一个字符串牌照 licensePlate 和一个字符串数组 words ， 请你找出并返回 words 中的 最短补全词 。

如果单词列表（words）中的一个单词包含牌照（licensePlate）中所有的字母，那么我们称之为 补全词 。在所有完整词中，最短的单词我们称之为 最短补全词 。

单词在匹配牌照中的字母时要：

- 忽略牌照中的数字和空格。
- 不区分大小写，比如牌照中的 "P" 依然可以匹配单词中的 "p" 字母。
- 如果某个字母在牌照中出现不止一次，那么该字母在补全词中的出现次数应当一致或者更多。

> 输入：licensePlate = "1s3 PSt", words = ["step", "steps", "stripe", "stepple"]
> 输出："steps"
> 说明：最短补全词应该包括 "s"、"p"、"s" 以及 "t"。在匹配过程中我们忽略牌照中的大小写。
> "step" 包含 "t"、"p"，但只包含一个 "s"，所以它不符合条件。
> "steps" 包含 "t"、"p" 和两个 "s"。
> "stripe" 缺一个 "s"。
> "stepple" 缺一个 "s"。
> 因此，"steps" 是唯一一个包含所有字母的单词，也是本样例的答案。

> 输入：licensePlate = "1s3 456", words = ["looks", "pest", "stew", "show"]
> 输出："pest"
> 说明：存在 3 个包含字母 "s" 且有着最短长度的补全词，"pest"、"stew"、和 "show" 三者长度相同，但我们返回最先出现的补全词 "pest" 。

题解

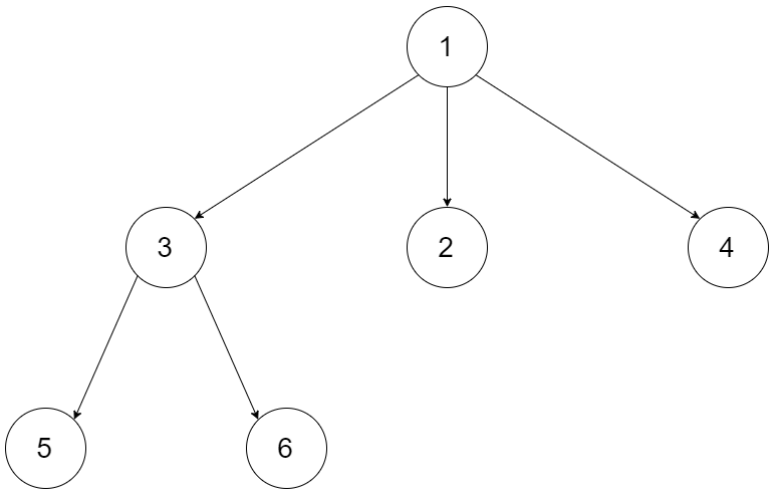
1、首先得到拍照所有小写字符，然后对words排序，

```
class Solution:
    def shortestCompletingWord(self, licensePlate: str, words: List[str]) ->
str:
        licensePlate=licensePlate.lower()
        s=''
        for i in licensePlate:
            if 'a'<=i<='z':
                s+=i
        words=sorted(words,key = lambda i:len(i),reverse=False)
        #print(words)
        for word in words:
            if len(word)>=len(s):
                a=s
                for w in word:
                    if w in a:
                        a=a.replace(w,'',1)
                if len(a)==0:
                    return word
```

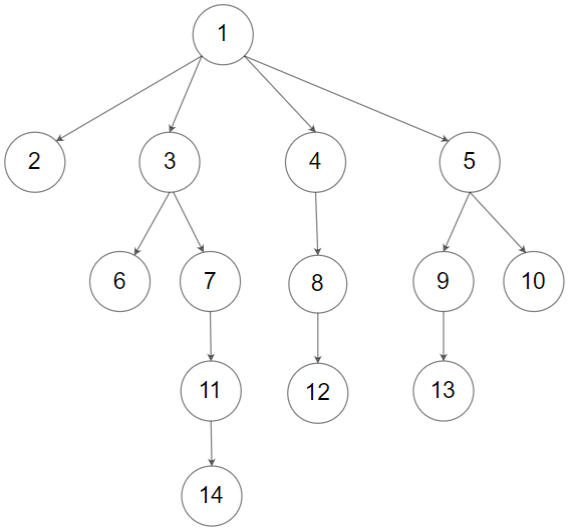
589、N叉树的前序遍历

给定一个 N 叉树，返回其节点值的 **前序遍历** 。

N 叉树 在输入中按层序遍历进行序列化表示，每组子节点由空值 `null` 分隔（请参见示例）。




```
输入：root = [1,null,3,2,4,null,5,6]
输出：[1,3,5,6,2,4]
```




```
输入：root =
[1,null,2,3,4,5,null,null,6,7,null,8,null,9,10,null,null,11,null,12,null,13,
null,null,14]
输出：[1,2,3,6,7,11,14,4,8,12,5,9,13,10]
```

题解

1、递归

```python
class Solution:
    def preorder(self, root: 'Node') -> List[int]:
        res=[]
        def DFS(root):
            if not root: return
            res.append(root.val)
            if root.children:
                for i in root.children:
                    DFS(i)

        DFS(root)

        return res
```

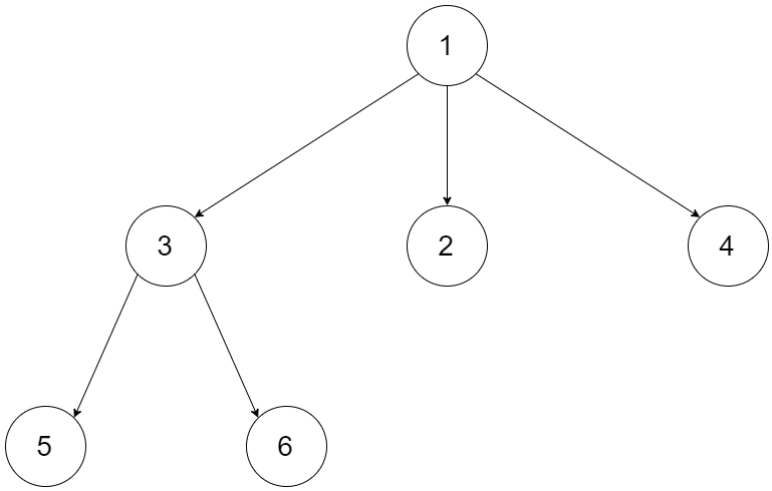
2、迭代

```python
class Solution:
    def preorder(self, root: 'Node') -> List[int]:
        if not root:
            return []
        s = [root]
        res = []
        while s:
            node = s.pop()
            res.append(node.val)
            s.extend(node.children[::-1])
        return res
```

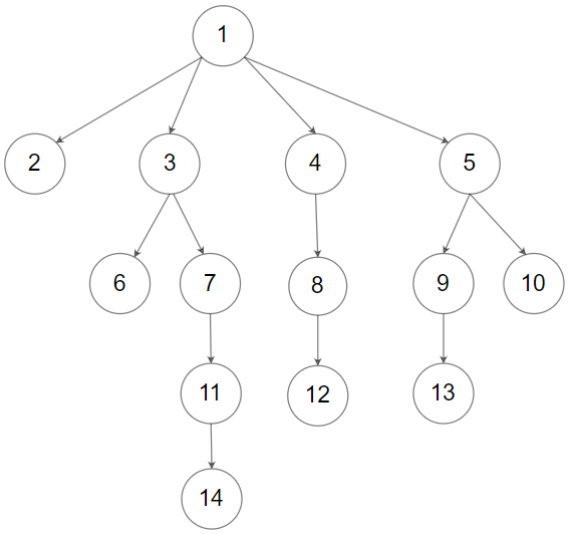
590、N叉树的后序遍历

给定一个 N 叉树，返回其节点值的 **后序遍历** 。

N 叉树 在输入中按层序遍历进行序列化表示，每组子节点由空值 `null` 分隔（请参见示例）。




```
输入：root = [1,null,3,2,4,null,5,6]
输出：[5,6,3,2,4,1]
```



输入：root = [1,null,2,3,4,5,null,null,6,7,null,8,null,9,10,null,null,11,null,12,null,13,null,null,14]
输出：[2,6,14,11,7,3,12,8,4,13,9,10,5,1]

题解

1、DFS

```
class Solution:
    def postorder(self, root: 'Node') -> List[int]:
        res=[]
        def DFS(root):
            if not root:return
            if root.children:
                for i in root.children:
                    DFS(i)
            res.append(root.val)

        DFS(root)
        return res
```

2、迭代

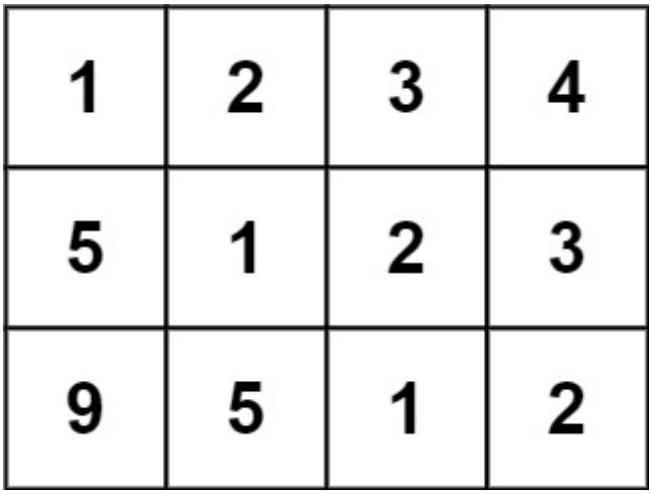
N叉树的后序遍历实际上就是先进行中右左遍历，最后翻转即可

```
class Solution:
    def postorder(self, root: 'Node') -> List[int]:
        if not root:
            return []
        res = []
        stack = []
        stack.append(root)
        while stack:
            node = stack.pop()
            res.append(node.val)
            stack.extend(node.children)
        return res[::-1]
```

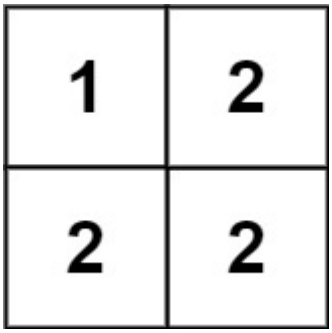
766、托普利茨矩阵

给你一个 m x n 的矩阵 matrix 。如果这个矩阵是托普利茨矩阵，返回 true ； 否则，返回 false 。

如果矩阵上每一条由左上到右下的对角线上的元素都相同，那么这个矩阵是 托普利茨矩阵 。




```
输入：matrix = [[1,2,3,4],[5,1,2,3],[9,5,1,2]]
输出：true
解释：
在上述矩阵中，其对角线为:
"[9]", "[5, 5]", "[1, 1, 1]", "[2, 2, 2]", "[3, 3]", "[4]"。
各条对角线上的所有元素均相同，因此答案是 True 。
```




```
输入：matrix = [[1,2],[2,2]]
输出：false
解释：
对角线 "[1, 2]" 上的元素不同。
```

题解

1、每行错位判断是否相等

```python
class Solution:
    def isToeplitzMatrix(self, matrix: List[List[int]]) -> bool:
        for i in range(len(matrix)-1):
            if matrix[i][:-1]!=matrix[i+1][1:]:
                return False

        return True
```

2、遍历

```python
class Solution:
    def isToeplitzMatrix(self, matrix: List[List[int]]) -> bool:
        # 矩阵的行数
        m = len(matrix)
        # 矩阵列数
        n = len(matrix[0])
        # 遍历矩阵判断每个元素与其右下角元素是否相等
        for i in range(m - 1):
            for j in range(n - 1):
                # 出现不相等，直接返回 False
                if matrix[i][j] != matrix[i + 1][j + 1]:
                    return False
        # 遍历结束后，若均相等，返回 True
        return True
```

771、宝石与石头

给定字符串J 代表石头中宝石的类型，和字符串 S代表你拥有的石头。 S 中每个字符代表了一种你拥有的石头的类型，你想知道你拥有的石头中有多少是宝石。

J 中的字母不重复，J 和 S中的所有字符都是字母。字母区分大小写，因此"a"和"A"是不同类型的石头。

> 输入: J = "aA", S = "aAAbbbb"
> 输出: 3

> 输入: J = "z", S = "ZZ"
> 输出: 0

题解

1、暴力，挨个判断

```python
class Solution:
    def numJewelsInStones(self, jewels: str, stones: str) -> int:
        res=0
        for i in stones:
            if i in jewels:
                res+=1

        return res
```

2、字典计数

```python
class Solution:
    def numJewelsInStones(self, jewels: str, stones: str) -> int:
        res=0
        dic=collections.Counter(stones)
        for i in dic:
            if i in jewels:
                res+=dic[i]

        return res
```

700、二叉搜索树中的搜索

给定二叉搜索树（BST）的根节点和一个值。 你需要在BST中找到节点值等于给定值的节点。 返回以该节点为根的子树。 如果节点不存在，则返回 NULL。

> 给定二叉搜索树:
>
>
>
>   4
>
>   / \
>
>   2   7
>
>   / \
>
> 1   3
>
> 和值: 2
>
> 返回
>
>   2
>
>   / \
>
> 1   3

题解

1、迭代

```python
class Solution:
    def searchBST(self, root: TreeNode, val: int) -> TreeNode:
        # 迭代
        while root:
            if root.val > val:
                root = root.left
            elif root.val < val:
                root = root.right
            else:
                return root
        return None
```

2、递归

```python
class Solution:
    def searchBST(self, root: TreeNode, val: int) -> TreeNode:
        if not root or root.val == val:
            return root
        if root.val > val:
            return self.searchBST(root.left, val)
        if root.val < val:
            return self.searchBST(root.right, val)
        return None
```

704、二分查找

给定一个 n 个元素有序的（升序）整型数组 nums 和一个目标值 target ，写一个函数搜索 nums 中的 target，如果目标值存在返回下标，否则返回 -1。

> ```
> 输入: nums = [-1,0,3,5,9,12], target = 9
> 输出: 4
> 解释: 9 出现在 nums 中并且下标为 4
> ```
>
> ```
> 输入: nums = [-1,0,3,5,9,12], target = 2
> 输出: -1
> 解释: 2 不存在 nums 中因此返回 -1
> ```

题解

1、二分查找算法

```python
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        left,right=0,len(nums)-1
        while left<=right:
            mid=(left+right)//2
            if nums[mid]==target:
                return mid
            elif nums[mid]>target:
                right=mid-1
            else:
                left=mid+1

        return -1
```

788、旋转数字

我们称一个数 X 为好数, 如果它的每位数字逐个地被旋转 180 度后，我们仍可以得到一个有效的，且和 X 不同的数。要求每位数字都要被旋转。

如果一个数的每位数字被旋转以后仍然还是一个数字， 则这个数是有效的。0, 1, 和 8 被旋转后仍然是它们自己；2 和 5 可以互相旋转成对方（在这种情况下，它们以不同的方向旋转，换句话说，2 和 5 互为镜像）；6 和 9 同理，除了这些以外其他的数字旋转以后都不再是有效的数字。

现在我们有一个正整数 N, 计算从 1 到 N 中有多少个数 X 是好数？

> 输入：10
> 输出：4
> 解释：
> 在[1, 10]中有四个好数： 2, 5, 6, 9。
> 注意 1 和 10 不是好数，因为他们在旋转之后不变。

题解

1、逐个判断，

```
class Solution:
    def rotatedDigits(self, n: int) -> int:
        res=[]
        for i in range(1,n+1):
            sts=str(i)
            t=''
            for s in sts:
                if s=='0' or s=='1' or s=='8':
                    t+=s
                elif s=='2':
                    t+='5'
                elif s=='5':
                    t+='2'
                elif s=='6':
                    t+='9'
                elif s=='9':
                    t+='6'
                else:
                    break
            if len(t)==len(sts) and t!=sts:
                res.append(i)

        return len(res)
```

2、字符判断，

这个数至少有一个好数字 其余的数字必须是普通数或者好数

```
class Solution:
    def rotatedDigits(self, N: int) -> int:
        # 2,5,6,9是1位数好数字， 1,8是1位数普通数字，
        # 0也是特殊普通数不能再首位，在运算时添加到p[i]里
        # 其余的数字3，4，7是坏数
        c,p = {0:[2,5,6,9]}, {0:[1,8]}
        # 以此生成2，3，4位的好数字c2,3,4和普通数字p2,3,4
        for i in range(1,4):
            # 前一项是前次位数的好数集合+非坏尾数(c[0]+p[0]+0)
            # 后一项是前次位数的普通数集合+好数尾数(c[0])
            c[i] = [10*a+b for a in c[i-1] for b in c[0]+p[0]+[0]]+[10*a+b for a
in p[i-1] for b in c[0]]
```

```
            p[i] = [10*a+b for a in p[i-1] for b in p[0]+[0]]
            # 比N小一共有多少个
        return len([l for k in c for l in c[k] if l<=N])
```

3、动态规划

1. 好数是旋转后不同的数比如(2,5,6,9) tag = 1
2. 普数是旋转以后相同的数如(0,1,8) tag = 0
3. 坏数是旋转后不能成立的数如(3,4,7) tag = -1

```
class Solution:
    def rotatedDigits(self, N: int) -> int:
        d = [0, 0, 1, -1, -1, 1, 1, -1, 0, 1] + [0] * (N - 9)
        for i in range(N + 1):
            if d[i // 10] == -1 or d[i % 10] == -1:
                d[i] = -1
            elif d[i // 10] == 1 or d[i % 10] == 1:
                d[i] = 1
        return d[:N].count(1)
```

31-40

796、旋转字符串

给定两个字符串, A 和 B。

A 的旋转操作就是将 A 最左边的字符移动到最右边。 例如, 若 A = 'abcde'，在移动一次之后结果就是'bcdea' 。如果在若干次旋转操作之后，A 能变成B，那么返回True。

> 示例 1:
> 输入: A = 'abcde', B = 'cdeab'
> 输出: true
>
> 示例 2:
> 输入: A = 'abcde', B = 'abced'
> 输出: false

题解

1、循环判断

```
class Solution:
    def rotateString(self, s: str, goal: str) -> bool:
        if not s and not goal:return True
        for i in range(len(s)-1):
            st=s[i+1:]+s[:i+1]
            if st==goal:
                return True

        return False
```

2、判断长度相等及B在A+A中

```
class Solution:
    def rotateString(self, A: str, B: str) -> bool:
        return len(A) == len(B) and B in A+A
```

3、递归

```
def rotateString(self, A: str, B: str) -> bool:
        m=10
        def check(A,B,p):
            if A==B:
                return True
            if p==len(A):
                return False
            if (A[1:]+A[0])==B:
                return True
            else:
                return check(A[1:]+A[0],B,p+1)

        return check(A,B,0)`
```

804、唯一摩尔斯密码词

国际摩尔斯密码定义一种标准编码方式，将每个字母对应于一个由一系列点和短线组成的字符串， 比如: "a" 对应 ".-", "b" 对应 "-...", "c" 对应 "-.-.", 等等。

为了方便，所有26个英文字母对应摩尔斯密码表如下:

```
[".-","-...","-.-.","-..",".","..-.","--.","....","..",".---","-.-",".-..","--
","-.","---",".--.","--.-",".-.","...","-","..-","...-",".--","-..-","-.--","-
--.."]
```

给定一个单词列表，每个单词可以写成每个字母对应摩尔斯密码的组合。例如，"cab" 可以写成 "-.-..--...", (即 "-.-." + ".-" + "-..." 字符串的结合)。我们将这样一个连接过程称作单词翻译。

返回我们可以获得所有词不同单词翻译的数量。

> 例如:
> 输入: words = ["gin", "zen", "gig", "msg"]
> 输出: 2
> 解释:
> 各单词翻译如下:
> "gin" -> "--...-."
> "zen" -> "--...-."
> "gig" -> "--...--."
> "msg" -> "--...--."
>
> 共有 2 种不同翻译, "--...-." 和 "--...--.".

题解

1、哈希表

```
class Solution(object):
    def uniqueMorseRepresentations(self, words):
        MORSE = [".-","-...","-.-.","-..",".","..-.","--.",
                 "....","..",".---","-.-",".-..","--","-.",
                 "---",".--.","--.-",".-.","...","-","..-",
                 "...-",".--","-..-","-.--","--.."]

        seen = {"".join(MORSE[ord(c) - ord('a')] for c in word)
                for word in words}

        return len(seen)
```

```
class Solution:
    def uniqueMorseRepresentations(self, words: List[str]) -> int:
        list = [".-","-...","-.-.","-..",".","..-.","--.","....","..",".---
","-.-",".-..","--","-.","---",".--.","--.-",".-.","...","-","..-","...-",".--
","-..-","-.--","--.."]
        ans = set()
        for word in words:
            s = ''
            for ch in word:
                s += list[ord(ch)-ord('a')]
            ans.add(s)
        return (len(ans))
```

806、写字符串需要的行数

我们要把给定的字符串 S 从左到右写到每一行上，每一行的最大宽度为100个单位，如果我们在写某个字母的时候会使这行超过了100 个单位，那么我们应该把这个字母写到下一行。我们给定了一个数组 widths ，这个数组 widths[0] 代表 'a' 需要的单位， widths[1] 代表 'b' 需要的单位，..., widths[25] 代表 'z' 需要的单位。

现在回答两个问题：至少多少行能放下S，以及最后一行使用的宽度是多少个单位？将你的答案作为长度为2的整数列表返回。

> 示例 2:
> 输入:
> widths =
> [4,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10]
> S = "bbbcccdddaaa"
> 输出: [2, 4]
> 解释:
> 除去字母'a'所有的字符都是相同的单位10，并且字符串 "bbbcccdddaa" 将会覆盖 9 * 10 + 2 * 4 = 98 个单位.
> 最后一个字母 'a' 将会被写到第二行，因为第一行只剩下2个单位了。
> 所以，这个答案是2行，第二行有4个单位宽度。

题解

1、超过100增加一行

```python
class Solution:
    def numberOfLines(self, widths: List[int], S: str) -> List[int]:
        count = 0
        lines = 0
        for s in S:
            count += widths[ord(s)-97]
            if count > 100:
                lines += 1
                count = widths[ord(s)-97]

        return [lines+1, count]
```

811、子域名访问计数

一个网站域名，如"discuss.leetcode.com"，包含了多个子域名。作为顶级域名，常用的有"com"，下一级则有"leetcode.com"，最低的一级为"discuss.leetcode.com"。当我们访问域名"discuss.leetcode.com"时，也同时访问了其父域名"leetcode.com"以及顶级域名 "com"。

给定一个带访问次数和域名的组合，要求分别计算每个域名被访问的次数。其格式为访问次数+空格+地址，例如："9001 discuss.leetcode.com"。

接下来会给出一组访问次数和域名组合的列表cpdomains 。要求解析出所有域名的访问次数，输出格式和输入格式相同，不限定先后顺序。

> 示例 1:
> 输入:
> ["9001 discuss.leetcode.com"]
> 输出:
> ["9001 discuss.leetcode.com", "9001 leetcode.com", "9001 com"]
> 说明:
> 例子中仅包含一个网站域名："discuss.leetcode.com"。按照前文假设，子域名"leetcode.com"和"com"都会被访问，所以它们都被访问了9001次。

> 示例 2
> 输入:
> ["900 google.mail.com", "50 yahoo.com", "1 intel.mail.com", "5 wiki.org"]
> 输出:
> ["901 mail.com","50 yahoo.com","900 google.mail.com","5 wiki.org","5 org","1 intel.mail.com","951 com"]
> 说明:
> 按照假设，会访问"google.mail.com" 900次，"yahoo.com" 50次，"intel.mail.com" 1次，"wiki.org" 5次。
> 而对于父域名，会访问"mail.com" 900+1 = 901次，"com" 900 + 50 + 1 = 951次，和 "org" 5 次。

题解

1、split函数可以指定最大切割次数

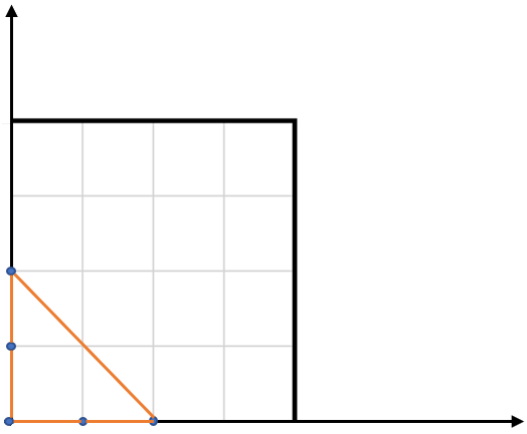
```python
class Solution:
    def subdomainVisits(self, cpdomains: List[str]) -> List[str]:
        if not cpdomains:
            return []

        res = {}
        for case in cpdomains:
            time, domain = case.split()
            length = len(domain.split('.'))
            for num in range(length):
                dm = domain.split('.', num)[-1]
                res[dm] = res.get(dm, 0) + int(time)
        return [str(v)+' '+k for k, v in res.items()]
```

812、最大三角形面积

给定包含多个点的集合，从其中取三个点组成三角形，返回能组成的最大三角形的面积。

> 示例:
> 输入: points = [[0,0],[0,1],[1,0],[0,2],[2,0]]
> 输出: 2
> 解释:
> 这五个点如下图所示。组成的橙色三角形是最大的，面积为2。



题解

1、暴力求解

三角形面积=`|(x1 * y2 + x2 * y3 + x3 * y1 - y1 * x2 - y2 * x3 - y3 * x1)|/2`

```python
class Solution:
    def largestTriangleArea(self, points: List[List[int]]) -> float:
        res=0
        for i in points:
            for j in points:
                for k in points:
                    res=max(res,abs(i[0]*j[1]+j[0]*k[1]+k[0]*i[1]-i[1]*j[0]-
j[1]*k[0]-k[1]*i[0]))
        return res/2
```

819、最常见的单词

给定一个段落 (paragraph) 和一个禁用单词列表 (banned)。返回出现次数最多，同时不在禁用列表中的单词。

题目保证至少有一个词不在禁用列表中，而且答案唯一。

禁用列表中的单词用小写字母表示，不含标点符号。段落中的单词不区分大小写。答案都是小写字母。

> 输入:
> paragraph = "Bob hit a ball, the hit BALL flew far after it was hit."
> banned = ["hit"]
> 输出: "ball"
> 解释:
> "hit" 出现了3次，但它是一个禁用的单词。
> "ball" 出现了2次 (同时没有其他单词出现2次)，所以它是段落里出现次数最多的，且不在禁用列表中的单词。
> 注意，所有这些单词在段落里不区分大小写，标点符号需要忽略（即使是紧挨着单词也忽略， 比如 "ball,"），
> "hit"不是最终的答案，虽然它出现次数更多，但它在禁用单词列表中。

题解

1、Counter从most common的元素逐一排查在不在banned 列表里

```
class Solution:
    def mostCommonWord(self, paragraph: str, banned: List[str]) -> str:
        for (a,b) in Counter(re.findall("\w+",paragraph.lower())).most_common():
            if a not in banned:
                return a
```

821、字符的最短距离

给你一个字符串 s 和一个字符 c ，且 c 是 s 中出现过的字符。

返回一个整数数组 answer ，其中 answer.length == s.length 且 answer[i] 是 s 中从下标 i 到离它 最近 的字符 c 的 距离 。

两个下标 i 和 j 之间的 距离 为 abs(i - j) ，其中 abs 是绝对值函数。

> 输入：s = "loveleetcode", c = "e"
> 输出：[3,2,1,0,1,0,0,1,2,2,1,0]
> 解释：字符 'e' 出现在下标 3、5、6 和 11 处（下标从 0 开始计数）。
> 距下标 0 最近的 'e' 出现在下标 3 ，所以距离为 abs(0 - 3) = 3 。
> 距下标 1 最近的 'e' 出现在下标 3 ，所以距离为 abs(1 - 3) = 3 。
> 对于下标 4 ，出现在下标 3 和下标 5 处的 'e' 都离它最近，但距离是一样的 abs(4 - 3) == abs(4 - 5) = 1 。
> 距下标 8 最近的 'e' 出现在下标 6 ，所以距离为 abs(8 - 6) = 2 。

题解

1、获得索引后，遍历索引得到距离最近的

```
class Solution:
    def shortestToChar(self, s: str, c: str) -> List[int]:
        res=[]
        idx=[i for i,j in enumerate(s) if j == c]
        for i in range(len(s)):
            x=10000
            for j in idx:
                x=min(x,abs(i-j))
            res.append(x)

        return res
```

2、双指针

- 一个是要计算相对位置的a，另一个是目标字符c，二者之间的位置我们可以分三种情况考虑
    1. c只出现一次，且a在c的两侧
       这时的相对距离，毋庸置疑应该等于 abs(index(a) - index(c))
    2. c出现两次，且a在两个c的中间
       这时的相对距离，应该计算a到两个c的距离然后取较小值
    3. c出现三次或三次以上，我们可以拆分成上述两种情况来分别计算
- 定义两个指针，移动策略如下
    - 一开始两指针均指向字符串头部
    - 右指针向前移动，直到遇见一个c，这时满足情况1
    - 更新左指针为右指针，右指针继续向右移动，直到再遇到c（满足情况2）或者走到字符串尾部（满足情况1）
- 注意
    - 计算距离的时机应该在右指针更新的时候，右指针第二次之后，直到走到尾部，都按第2种情况计算，右指针第一次更新或者走到尾部，按照第1中情况判断
    - 要考虑右指针指向尾部以及尾部字符恰好是c的情况，所以条件判断应该有优先级

```
class Solution:
    def shortestToChar(self, s: str, c: str) -> List[int]:
        # 定义变量 left记录上一个c的位置，如果存在的话
        left = right_cnt = 0
        answer = [0]*len(s)

        # 循环，找c，计算距离
        for right in range(len(s)):    # 右指针从0遍历到len(s)-1
            if s[right] == c:    # 如果右指针指向的字符是c，那么计算一次距离
                if not right_cnt:    # 如果右指针第一次更新，说明在c的一侧
                    answer[left:right+1] = [abs(right-i) for i in range(left,
right+1)]
                else:    # 如果右指针不是第一次更新，说明在两个c中间
                    answer[left:right+1] = [min(abs(right-i), abs(i-left)) for i
in range(left, right+1)]
                # 更新左指针以及出现c的次数
                left = right
                right_cnt += 1
            else:
                # 注意这个条件判断要在else里面，以排除最后一个元素是c的情况
```

```
                if right == len(s)-1:
                    answer[left:right+1] = [abs(left-i) for i in range(left,
right+1)]
        return answer
```

824、山羊拉丁文

给定一个由空格分割单词的句子 S。每个单词只包含大写或小写字母。

我们要将句子转换为 “Goat Latin”（一种类似于 猪拉丁文 - Pig Latin 的虚构语言）。

山羊拉丁文的规则如下：

- 如果单词以元音开头（a, e, i, o, u），在单词后添加"ma"。
  例如，单词"apple"变为"applema"。
- 如果单词以辅音字母开头（即非元音字母），移除第一个字符并将它放到末尾，之后再添加"ma"。
  例如，单词"goat"变为"oatgma"。
- 根据单词在句子中的索引，在单词最后添加与索引相同数量的字母'a'，索引从1开始。
  例如，在第一个单词后添加"a"，在第二个单词后添加"aa"，以此类推。
  返回将 S 转换为山羊拉丁文后的句子。

> 输入: "The quick brown fox jumped over the lazy dog"
> 输出: "heTmaa uickqmaaa rownbmaaaa oxfmaaaaa umpedjmaaaaaa overmaaaaaaa
> hetmaaaaaaaa azylmaaaaaaaaa ogdmaaaaaaaaaa"

题解

1、按照规则做就好

```
class Solution:
    def toGoatLatin(self, S: str) -> str:
        ans=[]
        res=S.split()
        vowel=['a','e','i','o','u','A','E','I','O','U']

        for i in range(len(res)):
            if res[i][0] in vowel:
                ans.append(res[i]+'ma'+'a'*(i+1))
            else:
                ans.append(res[i][1:]+res[i][0]+'ma'+'a'*(i+1))

        return ' '.join(ans)
```

830、较大分组的位置

在一个由小写字母构成的字符串 s 中，包含由一些连续的相同字符所构成的分组。

例如，在字符串 s = "abbxxxxzyy" 中，就含有 "a", "bb", "xxxx", "z" 和 "yy" 这样的一些分组。

分组可以用区间 [start, end] 表示，其中 start 和 end 分别表示该分组的起始和终止位置的下标。上例中的 "xxxx" 分组用区间表示为 [3,6] 。

我们称所有包含大于或等于三个连续字符的分组为 较大分组 。

找到每一个 较大分组 的区间，按起始位置下标递增顺序排序后，返回结果。

> 输入：s = "abbxxxxzzy"
> 输出：[[3,6]]
> 解释："xxxx" 是一个起始于 3 且终止于 6 的较大分组。

> 输入：s = "abcdddeeeeaabbbcd"
> 输出：[[3,5],[6,9],[12,14]]
> 解释：较大分组为 "ddd", "eeee" 和 "bbb"

题解

1、双指针，一个指针指向一个单词开头，另一个指针向后遍历，遇到第一个不同的单词，索引相减>2保存

```
class Solution:
    def largeGroupPositions(self, s: str) -> List[List[int]]:
        res=[]
        start=0
        for i in range(1,len(s)):
            if s[i]!=s[start]:
                if i-start>2:
                    res.append([start,i-1])
                start=i
            if i==(len(s)-1) and i-start>1:
                res.append([start,i])


        return res
```

2、正则表达式

```
class Solution:
    def largeGroupPositions(self, s: str) -> List[List[int]]:
        return ([[i.start(), i.end()-1] for i in re.finditer(r'([a-z])
(\1{2,})',s)])
```

832、翻转图像

给定一个二进制矩阵 A，我们想先水平翻转图像，然后反转图像并返回结果。

水平翻转图片就是将图片的每一行都进行翻转，即逆序。例如，水平翻转 [1, 1, 0] 的结果是 [0, 1, 1]。

反转图片的意思是图片中的 0 全部被 1 替换， 1 全部被 0 替换。例如，反转 [0, 1, 1] 的结果是 [1, 0, 0]。

```
输入：[[1,1,0],[1,0,1],[0,0,0]]
输出：[[1,0,0],[0,1,0],[1,1,1]]
解释：首先翻转每一行: [[0,1,1],[1,0,1],[0,0,0]]；
  然后反转图片: [[1,0,0],[0,1,0],[1,1,1]]
```

题解

1、按行遍历，每个数翻转，然后再水平翻转

```
class Solution:
    def flipAndInvertImage(self, image: List[List[int]]) -> List[List[int]]:
        for i in range(len(image)):
            for j in range(len(image[0])):
                image[i][j]=0 if image[i][j] else 1
            image[i]=image[i][::-1]

        return image
```

2、见缝插针法

- 这两个数是不同的，比如说一个是 1，一个是 0，那么先 10 反转，则一个是 0，一个是 1，再左右翻转，又变回一个是 1，一个是 0
    - 这说明当两个数是不同的时候，不用做任何事情
    - 当两个数相同的时候，要同时异或或被 1 减，即 10 反转

```
class Solution:
    def flipAndInvertImage(self, A: List[List[int]]) -> List[List[int]]:
        for row in A:
            for j in range((len(row) + 1) // 2):
                if row[j] == row[-1-j]:              # 采用Python化的符号索引
                    row[j] = row[-1-j] = 1 - row[j]
        return A
```

3、一行代码

```
return [1 - x for x in row[::-1] for row in A]
```

```
return [[j ^ 1 for j in row[::-1]] for row in A]
```

```
return [list(map(lambda x:1-x,row[::-1])) for row in A]
```

41-50

836、矩形重叠

矩形以列表 [x1, y1, x2, y2] 的形式表示，其中 (x1, y1) 为左下角的坐标，(x2, y2) 是右上角的坐标。矩形的上下边平行于 x 轴，左右边平行于 y 轴。

如果相交的面积为 正 ，则称两矩形重叠。需要明确的是，只在角或边接触的两个矩形不构成重叠。

给出两个矩形 rec1 和 rec2 。如果它们重叠，返回 true；否则，返回 false 。

```
输入：rec1 = [0,0,2,2], rec2 = [1,1,3,3]
输出：true
```

```
输入：rec1 = [0,0,1,1], rec2 = [1,0,2,1]
输出：false
```

```
输入：rec1 = [0,0,1,1], rec2 = [2,2,3,3]
输出：false
```

题解

1、两线段交集长度，当两段线交集长度有一个为负数或0，则说明两个矩形不构成重叠。

```python
class Solution:
    def isRectangleOverlap(self, rec1: List[int], rec2: List[int]) -> bool:
        if (min(rec1[2],rec2[2])-max(rec1[0],rec2[0]))<=0 or
        (min(rec1[3],rec2[3])-max(rec1[1],rec2[1]))<=0:
            return False
        else:
            return True
```

844、比较含退格的字符串

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，判断二者是否相等，并返回结果。 # 代表退格字符。

注意：如果对空文本输入退格字符，文本继续为空。

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

```
输入：S = "ab##", T = "c#d#"
输出：true
解释：S 和 T 都会变成 “”。
```

```
输入：S = "a##c", T = "#a#c"
输出：true
解释：S 和 T 都会变成 “c”。
```

题解

1、定义退格函数，返回处理后的字符串，，，函数通过从后向前遍历，记录#个数，为0时保存字符

```python
class Solution:
    def backspaceCompare(self, s: str, t: str) -> bool:
```

```
        def remove(st):
            re=0
            x=''
            for i in range(len(st)):
                if st[-1-i]=='#':
                    re+=1
                else:
                    if re==0:
                        x=x+st[-1-i]
                    else:
                        re-=1

            return x[::-1]

        return remove(s)==remove(t)
```

2、双指针

1. 准备两个指针 i, j分别指向 S，T 的末位字符，再准备两个变量 skipS，skipT来分别存放 S，T字符串中的 # 数量。
2. 从后往前遍历 SS，所遇情况有三，如下所示：
    1. 若当前字符是 #，则 skipS 自增 1；
    2. 当前字符不是 #，且 skipS 不为 0，则 skipS 自减 1；
    3. 若当前字符不是 #，且 skipS 为 0，则代表当前字符不会被消除，我们可以用来和 T 中的当前字符作比较。
3. 若对比过程出现 S, T 当前字符不匹配，则遍历结束，返回 false，若 S，T 都遍历结束，且都能一一匹配，则返回 true。

```
class Solution:
    def backspaceCompare(self, S: str, T: str) -> bool:
        i, j = len(S) - 1, len(T) - 1
        skipS = skipT = 0

        while i >= 0 or j >= 0:
            while i >= 0:
                if S[i] == "#":
                    skipS += 1
                    i -= 1
                elif skipS > 0:
                    skipS -= 1
                    i -= 1
                else:
                    break
            while j >= 0:
                if T[j] == "#":
                    skipT += 1
                    j -= 1
                elif skipT > 0:
                    skipT -= 1
                    j -= 1
                else:
                    break
            if i >= 0 and j >= 0:
                if S[i] != T[j]:
                    return False
            elif i >= 0 or j >= 0:
```

```
                return False
            i -= 1
            j -= 1

        return True
```

3、使用栈

```
class Solution:
    def backspaceCompare(self, S: str, T: str) -> bool:
        def get_result(strs):
            stack = []
            for i in strs:
                if i == '#':
                    if stack:
                        stack.pop()
                else:
                    stack.append(i)
            return stack

        return get_result(S) == get_result(T)
```

852、山脉数组的峰顶索引

符合下列属性的数组 arr 称为 山脉数组：

- arr.length >= 3
- 存在 i（0 < i < arr.length - 1）使得：
  - arr[0] < arr[1] < ... arr[i-1] < arr[i]
  - arr[i] > arr[i+1] > ... > arr[arr.length - 1]

给你由整数组成的山脉数组 arr，返回任何满足 arr[0] < arr[1] < ... arr[i - 1] < arr[i] > arr[i + 1] > ... > arr[arr.length - 1] 的下标 i 。

> ```
> 输入：arr = [0,1,0]
> 输出：1
> ```
>
> ```
> 输入：arr = [24,69,100,99,79,78,67,36,26,19]
> 输出：2
> ```

题解

1、题目保证arr是一个山脉数组，直接输出最大值的索引就好了

```
class Solution:
    def peakIndexInMountainArray(self, arr: List[int]) -> int:
        return arr.index(max(arr))
```

2、顺序查找

```python
class Solution:
    def peakIndexInMountainArray(self, arr: List[int]) -> int:
        # 顺序查找最大值
        for i in range(1, len(arr) - 1):
            if arr[i] > arr[i + 1]:
                return i
```

3、二分查找

- 若中点值比右方值大，说明极值点在中点左侧(包括中点)
- 若中点值比右方小，说明极值点在中点右侧(不包括中点)

```python
class Solution:
    def peakIndexInMountainArray(self, arr: List[int]) -> int:
        # 二分查找最大值
        left, right = 0, len(arr) - 1
        while left < right:
            mid = (left + right) // 2
            if arr[mid] > arr[mid + 1]:
                right = mid
            else:
                left = mid + 1
        return left
```

4、三分查找

即用两个点将区间三等分，通过比较两个三等分点可以排除掉左区间或右区间

- 若左等分点大于右等分点，有两种情况：极大值点在左区间或中区间，所以可以排除右区间
- 若左等分点小于右等分点，有两种情况：极大值点在中区间或右区间，所以可以排除左区间

```python
class Solution:
    def peakIndexInMountainArray(self, arr: List[int]) -> int:
        # 三分查找最大值
        left, right = 0, len(arr) - 1
        while left < right:
            m = (right - left) // 3
            m1 = left + m
            m2 = right - m
            if arr[m1] > arr[m2]:
                right = m2 - 1
            else:
                left = m1 + 1
        return left
```

859、亲密字符串

给定两个由小写字母构成的字符串 A 和 B ， 只要我们可以通过交换 A 中的两个字母得到与 B 相等的结果，就返回 true ； 否则返回 false 。

交换字母的定义是取两个下标 i 和 j （下标从 0 开始），只要 i!=j 就交换 A[i] 和 A[j] 处的字符。例如，在 "abcd" 中交换下标 0 和下标 2 的元素可以生成 "cbad" 。

```
输入： A = "ab", B = "ba"
输出： true
解释： 你可以交换 A[0] = 'a' 和 A[1] = 'b' 生成 "ba"，此时 A 和 B 相等。
```

```
输入： A = "ab", B = "ab"
输出： false
解释： 你只能交换 A[0] = 'a' 和 A[1] = 'b' 生成 "ba"，此时 A 和 B 不相等。
```

```
输入： A = "aa", B = "aa"
输出： true
解释： 你可以交换 A[0] = 'a' 和 A[1] = 'a' 生成 "aa"，此时 A 和 B 相等。
```

题解

1、判断返回True的条件即可

- 大条件：len(A) == len(B)
  - 有两个不同地方(i,j)，且A[i]=B[j],A[j]=B[i]
  - 二：完全相同，一个数组中存在重复数字

```
class Solution:
    def buddyStrings(self, A: str, B: str) -> bool:
        index =[]
        if len(A) == len(B):
            for i in range(len(A)):
                if A[i] != B[i]:
                    index.append(i)
            if len(index) == 2 and A[index[0]] == B[index[1]] and A[index[1]] ==
B[index[0]]:
                return True
            if len(index) == 0 and len(A)-len(set(A)) > 0:
                return True
        return False
```

860、柠檬水找零

在柠檬水摊上，每一杯柠檬水的售价为 5 美元。

顾客排队购买你的产品，（按账单 bills 支付的顺序）一次购买一杯。

每位顾客只买一杯柠檬水，然后向你付 5 美元、10 美元或 20 美元。你必须给每个顾客正确找零，也就是说净交易是每位顾客向你支付 5 美元。

注意，一开始你手头没有任何零钱。

如果你能给每位顾客正确找零，返回 true ，否则返回 false

```
输入：[5,5,5,10,20]
输出：true
解释：
前 3 位顾客那里，我们按顺序收取 3 张 5 美元的钞票。
第 4 位顾客那里，我们收取一张 10 美元的钞票，并返还 5 美元。
第 5 位顾客那里，我们找还一张 10 美元的钞票和一张 5 美元的钞票。
由于所有客户都得到了正确的找零，所以我们输出 true
```

```
输入：[5,5,10,10,20]
输出：false
解释：
前 2 位顾客那里，我们按顺序收取 2 张 5 美元的钞票。
对于接下来的 2 位顾客，我们收取一张 10 美元的钞票，然后返还 5 美元。
对于最后一位顾客，我们无法退回 15 美元，因为我们现在只有两张 10 美元的钞票。
由于不是每位顾客都得到了正确的找零，所以答案是 false。
```

题解

1、字典保存拥有零钱，判断False的条件

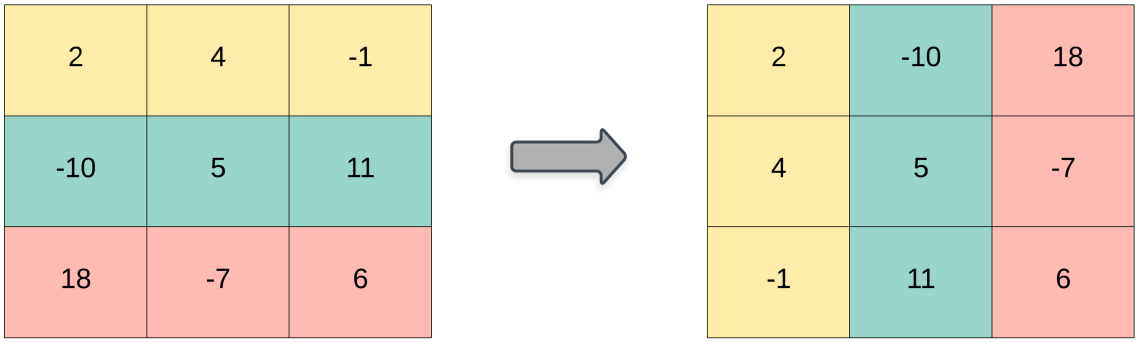
```python
class Solution:
    def lemonadeChange(self, bills: List[int]) -> bool:
        dic={'5':0,'10':0}
        for i in bills:
            if i==5:
                dic['5']+=1
            elif i==10:
                dic['10']+=1
                if dic['5']>0:
                    dic['5']-=1
                else:
                    return False
            else:
                if dic['10']>0 and dic['5']>0:
                    dic['10']-=1
                    dic['5']-=1
                elif dic['5']>2:
                    dic['5']-=3
                else:
                    return False

        return True
```

867、转置矩阵

给你一个二维整数数组 `matrix`， 返回 `matrix` 的 **转置矩阵** 。

矩阵的 **转置** 是指将矩阵的主对角线翻转，交换矩阵的行索引与列索引。




```
输入：matrix = [[1,2,3],[4,5,6],[7,8,9]]
输出：[[1,4,7],[2,5,8],[3,6,9]]
```

```
输入：matrix = [[1,2,3],[4,5,6]]
输出：[[1,4],[2,5],[3,6]]
```

题解

1、两个遍历

```
class Solution:
    def transpose(self, matrix: List[List[int]]) -> List[List[int]]:
        res=[]
        for j in range(len(matrix[0])):
            x=[]
            for i in range(len(matrix)):
                x.append(matrix[i][j])
            res.append(x)

        return res
```

2、一行代码

```
class Solution:
    def transpose(self, matrix: List[List[int]]) -> List[List[int]]:
        # 1. 嵌套列表表达式
        return [[m[i]  for m in matrix] for i in range(len(matrix[0]))]

        #  2. 解包 zip函数
        return list(zip(*matrix))
```

868、二进制间距

给定一个正整数 n，找到并返回 n 的二进制表示中两个 相邻 1 之间的 最长距离 。如果不存在两个相邻的 1，返回 0 。

如果只有 0 将两个 1 分隔开（可能不存在 0 ），则认为这两个 1 彼此 相邻 。两个 1 之间的距离是它们的二进制表示中位置的绝对差。例如，"1001" 中的两个 1 的距离为 3

> 输入：n = 22
> 输出：2
> 解释：
> 22 的二进制是 "10110" 。
> 在 22 的二进制表示中，有三个 1，组成两对相邻的 1 。
> 第一对相邻的 1 中，两个 1 之间的距离为 2 。
> 第二对相邻的 1 中，两个 1 之间的距离为 1 。
> 答案取两个距离之中最大的，也就是 2 。

> 输入：n = 8
> 输出：0
> 解释：
> 8 的二进制是 "1000" 。
> 在 8 的二进制表示中没有相邻的两个 1，所以返回 0 。

题解

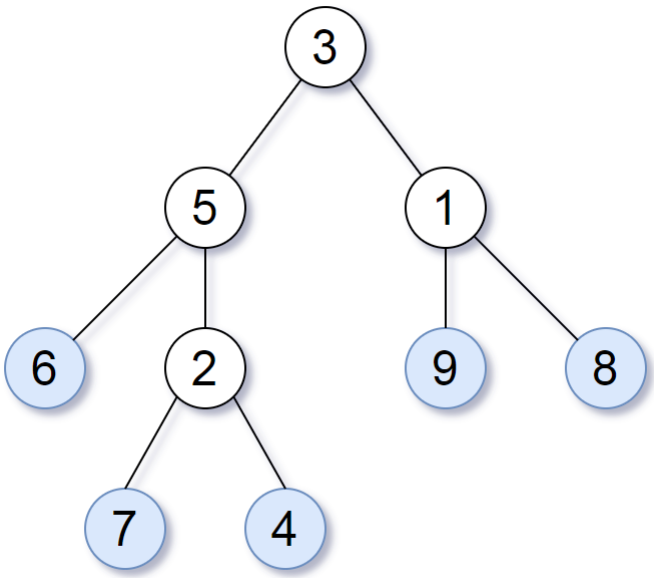
1、首先获得1的索引数组，然后选择相邻最大距离的

```python
class Solution:
    def binaryGap(self, n: int) -> int:
        st=str(bin(n))[2:]
        idx=[]
        for i in range(len(st)):
            if st[i]=='1':
                idx.append(i)
        res=0
        if len(idx)==1:return 0
        for i in range(len(idx)-1):
            res=max(res,idx[i+1]-idx[i])

        return res
```

872、叶子相似的树

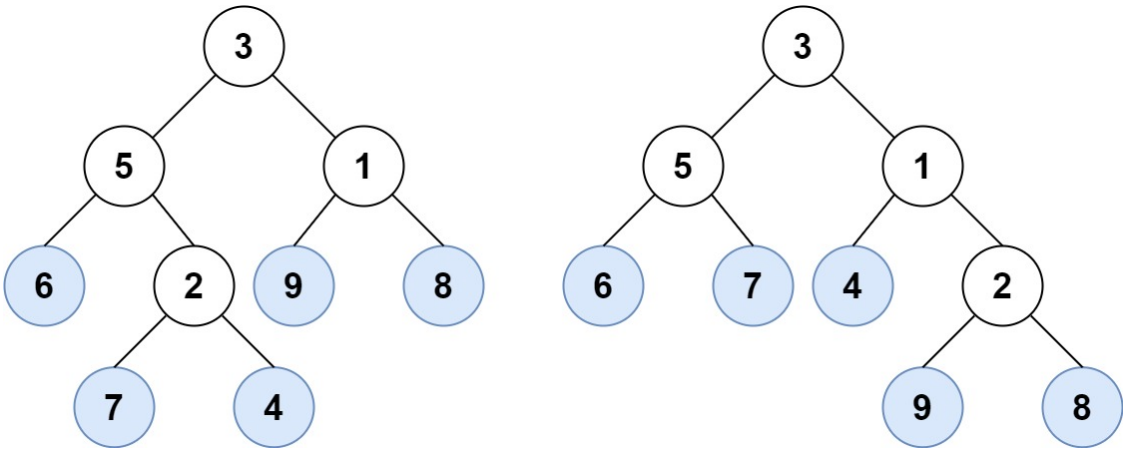
请考虑一棵二叉树上所有的叶子，这些叶子的值按从左到右的顺序排列形成一个 **叶值序列** 。



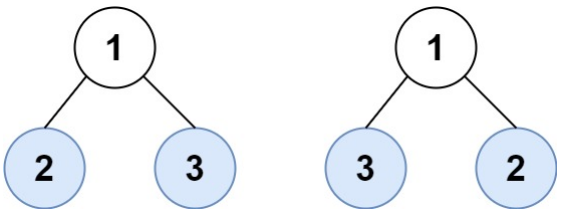
举个例子，如上图所示，给定一棵叶值序列为 (6, 7, 4, 9, 8) 的树。

如果有两棵二叉树的叶值序列是相同，那么我们就认为它们是 叶相似 的。

如果给定的两个根结点分别为 root1 和 root2 的树是叶相似的，则返回 true；否则返回 false 。




```
输入：root1 = [3,5,1,6,2,9,8,null,null,7,4], root2 =
[3,5,1,6,7,4,2,null,null,null,null,null,null,9,8]
输出：true
```




```
输入：root1 = [1,2,3], root2 = [1,3,2]
输出：false
```

题解

1、DFS

得到每个数的叶子列表，对比是否相等即可

```python
class Solution:
    def leafSimilar(self, root1: TreeNode, root2: TreeNode) -> bool:
        res1=[]
        res2=[]
        def DFS1(root):
            if not root:return
            if root.left:DFS1(root.left)
            if root.right:DFS1(root.right)
            if not root.left and not root.right:
                res1.append(root.val)
            return res1
        def DFS2(root):
            if not root:return
            if root.left:DFS2(root.left)
            if root.right:DFS2(root.right)
            if not root.left and not root.right:
                res2.append(root.val)
            return res2

        return DFS1(root1)==DFS2(root2)
```

2、BFS

```python
class Solution:
    def leafSimilar(self, root1: TreeNode, root2: TreeNode) -> bool:
        def BFS(root):
            stack=[root]
            res=[]
            while stack:
                rt=stack.pop()
                if not rt.left and  not rt.right:
                    res.append(rt.val)
                if rt.left:stack.append(rt.left)
                if rt.right:stack.append(rt.right)

            return res

        return BFS(root1)==BFS(root2)
```

874、模拟行走机器人

机器人在一个无限大小的 XY 网格平面上行走，从点 (0, 0) 处开始出发，面向北方。该机器人可以接收以下三种类型的命令 commands：

- -2： 向左转 90 度
- -1： 向右转 90 度
- 1 <= x <= 9： 向前移动 x 个单位长度

在网格上有一些格子被视为障碍物 obstacles 。第 i 个障碍物位于网格点 obstacles[i] = (xi, yi) 。

机器人无法走到障碍物上，它将会停留在障碍物的前一个网格方块上，但仍然可以继续尝试进行该路线的其余部分。

返回从原点到机器人所有经过的路径点（坐标为整数）的最大欧式距离的平方。（即，如果距离为 5，则返回 25

> 输入：commands = [4,-1,3], obstacles = []
> 输出：25
> 解释：
> 机器人开始位于 (0, 0)：
> 1. 向北移动 4 个单位，到达 (0, 4)
> 2. 右转
> 3. 向东移动 3 个单位，到达 (3, 4)
> 距离原点最远的是 (3, 4) ，距离为 32 + 42 = 25

> 输入：commands = [4,-1,4,-2,4], obstacles = [[2,4]]
> 输出：65
> 解释：机器人开始位于 (0, 0)：
> 1. 向北移动 4 个单位，到达 (0, 4)
> 2. 右转
> 3. 向东移动 1 个单位，然后被位于 (2, 4) 的障碍物阻挡，机器人停在 (1, 4)
> 4. 左转
> 5. 向北走 4 个单位，到达 (1, 8)
> 距离原点最远的是 (1, 8) ，距离为 12 + 82 = 65

题解

1、字典保存转向

```
class Solution:
    def robotSim(self, commands: List[int], obstacles: List[List[int]]) -> int:
        # 字典存储某个方向(key)对应的 [x方向移动，y方向移动，当前方向的左侧，当前方向的右侧](val)
        direction = {'up': [0, 1, 'left', 'right'],
                     'down': [0, -1, 'right', 'left'],
                     'left': [-1, 0, 'down', 'up'],
                     'right': [1, 0, 'up', 'down']}
        x, y = 0, 0
        dir = 'up'
        res = 0
        obstacles = set(map(tuple, obstacles))
        for command in commands:
            if command > 0:  # 正数的话进行模型行进操作
                for step in range(command):
                    if (x + direction[dir][0], y + direction[dir][1]) not in obstacles:
                        x += direction[dir][0]
                        y += direction[dir][1]
                        res = max(res, x ** 2 + y ** 2)
                    else:
                        break
            else:  # 负数的话只改变行进方向
                if command == -1:  # 右转
                    dir = direction[dir][3]
                else:  # 即command == -2，左转
                    dir = direction[dir][2]
        return res
```

876、链表的中间节点

给定一个头结点为 `head` 的非空单链表，返回链表的中间结点。

如果有两个中间结点，则返回第二个中间结点。

> 输入：[1,2,3,4,5]
> 输出：此列表中的结点 3 (序列化形式：[3,4,5])
> 返回的结点值为 3 。 (测评系统对该结点序列化表述是 [3,4,5])。
> 注意，我们返回了一个 ListNode 类型的对象 ans，这样：
> ans.val = 3, ans.next.val = 4, ans.next.next.val = 5, 以及
> ans.next.next.next = NULL.

> 输入：[1,2,3,4,5,6]
> 输出：此列表中的结点 4 (序列化形式：[4,5,6])
> 由于该列表有两个中间结点，值分别为 3 和 4，我们返回第二个结点。

题解

1、快慢指针

```
class Solution:
    def middleNode(self, head: ListNode) -> ListNode:
        slow,fist=head,head
        while fist:
            try:
                fist=fist.next.next
            except:
                break
            slow=slow.next


        return slow
```

```
class Solution:
    def middleNode(self, head: ListNode) -> ListNode:
        slow = head
        fast = head
        while fast and fast.next:
            slow = slow.next
            fast = fast.next.next

        return slow
```

2、计算长度

```
class Solution:
    def middleNode(self, head: ListNode) -> ListNode:
        length = 0
        cur = head
        # 计算长度
```

```
        while cur:
            length += 1
            cur = cur.next
        length = length//2+1
        # 查找节点
        cur = head
        while length-1:
            cur = cur.next
            length -= 1
        return cur
```