0.总结.md

- 基础遍历
  - [ ] 14 最长公共前缀
  - [ ] 36 有效的数独：用set来存储每行，每列，每个mat的数字，因为只有9个要么满足要么重复
  - [ ] 38 外观数列：注意不要遗漏last one
  - [ ] 228 汇总区间：注意不要遗漏last one
  - [ ] 412 Fizz Buzz
  - [ ] 463 岛屿的周长
  - [ ] 482 密钥格式化
  - [ ] 485 最大连续 1 的个数
  - [ ] 495 提莫攻击： 先加入第一个位置的攻击，后面的判断就简单了
  - [ ] 551 学生出勤记录 I
  - [ ] 599 两个列表的最小索引总和：小表检索大表空间换时间
  - [ ] 643 子数组最大平均数 I
  - [ ] 657 机器人能否返回原点
  - [ ] 661 图片平滑器
  - [ ] 724 寻找数组的中心下标
  - [ ] 747 至少是其他数字两倍的最大数
  - [ ] 766 托普利茨矩阵
  - [ ] 830 较大分组的位置
  - [ ] 832 翻转图像： [i,j] -> [j, n-i-1]
- 矩阵操作
  - [ ] 48 旋转图像
  - [ ] 189 轮转数组
  - [ ] 401 二进制手表
  - [ ] 506 相对名次
  - [ ] 566 重塑矩阵
- 技巧题
  - [ ] 6 Z 字形变换
  - [ ] 54 螺旋矩阵: 注意最后剩余部分的边界问题
  - [ ] 57 插入区间：只判断不重叠区间，剩余都是重叠情况。需要flag来判断新区间是否插入
  - [ ] 59 螺旋矩阵 II
  - [ ] 73 矩阵置零
  - [ ] 238 除自身以外数组的乘积：从左到右+从右到左

- [ ] 448 找到所有数组中消失的数字
- [ ] 561 数组拆分 I
- [ ] 598 范围求和 II
- [ ] 696 计数二进制子串
- [ ] 821 字符的最短距离

14. 最长公共前缀.md

编写一个函数来查找字符串数组中的最长公共前缀。

如果不存在公共前缀，返回空字符串 ""。

示例 1：

输入：strs = ["flower","flow","flight"]
输出："fl"

示例 2：

输入：strs = ["dog","racecar","car"]
输出：""
解释：输入不存在公共前缀。

提示：

```
1 <= strs.length <= 200
0 <= strs[i].length <= 200
strs[i] 仅由小写英文字母组成
```

1. 按位置遍历

```python
class Solution:
    def longestCommonPrefix(self, strs: List[str]) -> str:
        share = []
        l = min([len(s) for s in strs])
        for i in range(l):
            ss = strs[0][i]
            if all([s[i]==ss for s in strs[1:]]):
                share.append(ss)
            else:
                break
        return ''.join(share)
```

2. 按字符遍历

```python
class Solution:
    def longestCommonPrefix(self, strs: List[str]) -> str:
        prefix = strs[0]
        for s in strs[1:]:
            l = min(len(prefix), len(s))
            prefix = prefix[:l]
            if l==0:
                return ''
            for i in range(l):
                if prefix[i]!=s[i]:
                    prefix = prefix[:i]
                    break
        return prefix
```

189. 轮转数组.md

给你一个数组，将数组中的元素向右轮转 k 个位置，其中 k 是非负数。

示例 1:

输入: nums = [1,2,3,4,5,6,7], k = 3
输出: [5,6,7,1,2,3,4]
解释:
向右轮转 1 步: [7,1,2,3,4,5,6]
向右轮转 2 步: [6,7,1,2,3,4,5]
向右轮转 3 步: [5,6,7,1,2,3,4]

示例 2:

输入：nums = [-1,-100,3,99], k = 2
输出：[3,99,-1,-100]
解释:
向右轮转 1 步: [99,-1,-100,3]
向右轮转 2 步: [3,99,-1,-100]

提示：

```
1 <= nums.length <= 105
-231 <= nums[i] <= 231 - 1
0 <= k <= 105
```

进阶：

```
尽可能想出更多的解决方案，至少有 三种 不同的方法可以解决这个问题。
你可以使用空间复杂度为 O(1) 的 原地 算法解决这个问题吗?
```

```python
class Solution:
    def rotate(self, nums: List[int], k: int) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        n = len(nums)
        nums[:] = nums[-k%n:] +nums[:-k%n]
```

哈哈python总有不一样的解法

228. 汇总区间.md

给定一个无重复元素的有序整数数组 nums 。

返回 恰好覆盖数组中所有数字 的 最小有序 区间范围列表。也就是说，nums 的每个元素都恰好被某个区间范围所覆盖，并且不存在属于某个范围但不属于 nums 的数字 x 。

列表中的每个区间范围 [a,b] 应该按如下格式输出：

```
"a->b" ，如果 a != b
"a" ，如果 a == b
```

示例 1：

输入：nums = [0,1,2,4,5,7]
输出：["0->2","4->5","7"]
解释：区间范围是：
[0,2] --> "0->2"
[4,5] --> "4->5"
[7,7] --> "7"

示例 2：

输入：nums = [0,2,3,4,6,8,9]
输出：["0","2->4","6","8->9"]
解释：区间范围是：
[0,0] --> "0"

[2,4] --> "2->4"
[6,6] --> "6"
[8,9] --> "8->9"

示例 3：

输入：nums = []
输出：[]

示例 4：

输入：nums = [-1]
输出：["-1"]

示例 5：

输入：nums = [0]
输出：["0"]

提示：

```
0 <= nums.length <= 20
-231 <= nums[i] <= 231 - 1
nums 中的所有值都 互不相同
nums 按升序排列
```

```python
class Solution:
    def summaryRanges(self, nums: List[int]) -> List[str]:
        res = []
        s = []
        def helper(s):
            if len(s)>1:
                r = '->'.join([s[0],s[-1]])
            else:
                r = str(s[0])
            return r

        for i,n in enumerate(nums):
            if nums[i]-nums[i-1]>1:
                res.append(helper(s))
                s = [str(nums[i])]
            else:
                s.append(str(n))
        if s:
            res.append(helper(s))

        return res
```

Tips

1. 所有在current step存在不向结果写入，在下一步进行判断的case都要小心对末尾元素的处理。干过好几次把tail忘记的事情了

238. 除自身以外数组的乘积.md

给你一个整数数组 nums，返回 数组 answer ，其中 answer[i] 等于 nums 中除 nums[i] 之外其余各元素的乘积。

题目数据 保证 数组 nums之中任意元素的全部前缀元素和后缀的乘积都在 32 位 整数范围内。

请不要使用除法，且在 O(n) 时间复杂度内完成此题。

示例 1:

输入: nums = [1,2,3,4]
输出: [24,12,8,6]

示例 2:

输入: nums = [-1,1,0,-3,3]
输出: [0,0,9,0,0]

提示：

```
2 <= nums.length <= 105
-30 <= nums[i] <= 30
保证 数组 nums之中任意元素的全部前缀元素和后缀的乘积都在  32 位 整数范围内
```

```python
class Solution:
    def productExceptSelf(self, nums: List[int]) -> List[int]:
        answer = [1] * len(nums)
        for i in range(1,len(nums)):
            answer[i] = answer[i-1]* nums[i-1]
        R =1
        for i in range(len(nums)-1, -1,-1):
            answer[i] = R*answer[i]
            R*=nums[i]
        return answer
```

Tips

因为不能使用除法，所以用乘积列表来代表

36. 有效的数独.md

请你判断一个 9x9 的数独是否有效。只需要 根据以下规则 ，验证已经填入的数字是否有效即可。

```
数字 1-9 在每一行只能出现一次。
数字 1-9 在每一列只能出现一次。
数字 1-9 在每一个以粗实线分隔的 3x3 宫内只能出现一次。（请参考示例图）
```

数独部分空格内已填入了数字，空白格用 '.' 表示。

注意：

```
一个有效的数独（部分已被填充）不一定是可解的。
只需要根据以上规则，验证已经填入的数字是否有效即可。
```

示例 1：

输入：board =
[["5","3",".",".","7",".",".",".","."]
,["6",".",".","1","9","5",".",".","."]
,[".","9","8",".",".",".",".","6","."]
,["8",".",".",".","6",".",".",".","3"]
,["4",".",".","8",".","3",".",".","1"]
,["7",".",".",".","2",".",".",".","6"]
,[".","6",".",".",".",".","2","8","."]
,[".",".",".","4","1","9",".",".","5"]
,[".",".",".",".","8",".",".","7","9"]]
输出：true

示例 2：

输入：board =
[["8","3",".",".","7",".",".",".","."]
,["6",".",".","1","9","5",".",".","."]
,[".","9","8",".",".",".",".","6","."]
,["8",".",".",".","6",".",".",".","3"]
,["4",".",".","8",".","3",".",".","1"]
,["7",".",".",".","2",".",".",".","6"]
,[".","6",".",".",".",".","2","8","."]
,[".",".",".","4","1","9",".",".","5"]
,[".",".",".",".","8",".",".","7","9"]]
输出：false

解释：除了第一行的第一个数字从 5 改为 8 以外，空格内其他数字均与 示例1 相同。 但由于位于左上角的 3x3 宫内有两个 8 存在, 因此这个数独是无效的。

提示：

```
board.length == 9
board[i].length == 9
board[i][j] 是一位数字或者 '.'
```

```
class Solution:
    def isValidSudoku(self, board: List[List[str]]) -> bool:
        record = {'row':defaultdict(set),'col' :defaultdict(set), 'square':
defaultdict(set)}
        n = len(board)
        m = int(n**0.5)
        for i in range(n):
            for j in range(n):
                if board[i][j]=='.':
                    continue
                if board[i][j] in record['row'][i] or board[i][j] in record['col'][j]:
                    return False
                sq = (i//m) *m + j//m
                if board[i][j] in record['square'][sq]:
                    return False

                record['row'][i].add(board[i][j])
                record['col'][j].add(board[i][j])
                record['square'][sq].add(board[i][j])
        return True
```

Tips

最简单的方案就是O（n*n）,直接完整遍历整个array，在遍历过程中直接记录每行，每列，每个square里面出现过的数字

38. 外观数列.md

给定一个正整数 n ，输出外观数列的第 n 项。

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

你可以将其视作是由递归公式定义的数字字符串序列：

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

前五项如下：

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

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

例如，数字字符串 "3322251" 的描述如下图：

示例 1：

输入：n = 1
输出："1"
解释：这是一个基本样例。

示例 2：

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

提示：

```
1 <= n <= 30
```

```python
class Solution:
    def countAndSay(self, n: int) -> str:
        def helper(n):
```

```
            # input old n return new n
            number = n[0]
            time =1
            res = ''
            for i in range(1, len(n)):
                if n[i] == n[i-1]:
                    time+=1
                else:
                    res += '{}{}'.format(time, number)
                    time=1
                    number = n[i]
            res += '{}{}'.format(time, number)
            return res

        res = '1'
        for i in range(1,n):
            res = helper(res)
            print(res)
        return res
```

Tips

1. 就是从前往后遍历 得到number+times，注意最后一个位置的边界问题，不要漏掉最后一个

401. 二进制手表.md

二进制手表顶部有 4 个 LED 代表 小时（0-11），底部的 6 个 LED 代表 分钟（0-59）。每个 LED 代表一个 0 或 1，最低位在右侧。

例如，下面的二进制手表读取 "3:25" 。

（图源：WikiMedia - Binary clock samui moon.jpg ，许可协议：Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0) ）

给你一个整数 turnedOn ，表示当前亮着的 LED 的数量，返回二进制手表可以表示的所有可能时间。你可以 按任意顺序 返回答案。

小时不会以零开头：

例如，"01:00" 是无效的时间，正确的写法应该是 "1:00" 。

分钟必须由两位数组成，可能会以零开头：

例如，"10:2" 是无效的时间，正确的写法应该是 "10:02" 。

示例 1：

输入：turnedOn = 1
输出：["0:01","0:02","0:04","0:08","0:16","0:32","1:00","2:00","4:00","8:00"]

示例 2：

输入：turnedOn = 9
输出：[]

提示：

```
0 <= turnedOn <= 10
```

```
class Solution:
    def readBinaryWatch(self, turnedOn: int) -> List[str]:
        from itertools import combinations, product
        def hour_comb(n):
            if n ==0:
                return ['0']
            hour = [1,2,4,8]
            pos = combinations(hour, n)
            ans = (str(sum(i)) for i in pos if sum(i)<12)
            return ans

        def min_comb(n):
            if n==0:
                return ['00']
            minu = [1,2,4,8,16,32]
            pos = combinations(minu, n )
            ans = ('{:02d}'.format(sum(i)) for i in pos if sum(i)<60)
            return ans

        ans = []
        for i in range(min(5,turnedOn+1)):
            j = turnedOn -i
            if (j <=6):
                hours = hour_comb(i)
                mins = min_comb(j)
                print(hours)
                print(mins)
                ans+= [':'.join(i) for i in product(hours, mins)]
        return ans
```

Tips

1. 感觉官方的遍历有点坑爹，可以有限遍历hour和min的可能解，然后product和在一起就好
2. 考察点可以是使用combination和product
3. 以及为了不产生重复接，总共10个亮灯最多只遍历5个

412. Fizz Buzz.md

写一个程序，输出从 1 到 n 数字的字符串表示。

1. 如果 n 是3的倍数，输出“Fizz”；
2. 如果 n 是5的倍数，输出“Buzz”；

3.如果 n 同时是3和5的倍数，输出 “FizzBuzz”。

示例：

n = 15,

返回:
[
"1",
"2",
"Fizz",
"4",
"Buzz",
"Fizz",
"7",
"8",
"Fizz",
"Buzz",
"11",
"Fizz",
"13",
"14",
"FizzBuzz"
]

```
class Solution:
    def fizzBuzz(self, n: int) -> List[str]:
        def helper(s):
            if s%15==0:
                return 'FizzBuzz'
            elif s%5==0:
                return 'Buzz'
            elif s%3==0:
                return 'Fizz'
```

```
            else:
                return str(s)

        ans =[]
        for i in range(1,n+1):
            ans.append(helper(i))
        return ans
```

448. 找到所有数组中消失的数字.md

给你一个含 n 个整数的数组 nums ，其中 nums[i] 在区间 [1, n] 内。请你找出所有在 [1, n] 范围内但没有出现在 nums 中的数字，并以数组的形式返回结果。

示例 1：

输入：nums = [4,3,2,7,8,2,3,1]
输出：[5,6]

示例 2：

输入：nums = [1,1]
输出：[2]

提示：

```
n == nums.length
1 <= n <= 105
1 <= nums[i] <= n
```

进阶：你能在不使用额外空间且时间复杂度为 O(n) 的情况下解决这个问题吗? 你可以假定返回的数组不算在额外空间内。

```
class Solution:
    def findDisappearedNumbers(self, nums: List[int]) -> List[int]:
        for num in nums:
            index = abs(num)-1
            nums[index]=abs(nums[index]) *-1
        ans = []

        for i in range(len(nums)):
            if nums[i] >0:
                ans.append(i+1)
        return ans
```

Tips

如果占用额外空间的话很简单只要用hash就行，不用额外空间的解法比较技巧。就是在原有数组上如果i出现过就把i位置的元素变成负数，最后判断哪些位置是正数就是确实信息，说白了就是在原有空间里看看还能咋承载额外信息。

463. 岛屿的周长.md

给定一个 row x col 的二维网格地图 grid ，其中：grid[i][j] = 1 表示陆地， grid[i][j] = 0 表示水域。

网格中的格子 水平和垂直 方向相连（对角线方向不相连）。整个网格被水完全包围，但其中恰好有一个岛屿（或者说，一个或多个表示陆地的格子相连组成的岛屿）。

岛屿中没有“湖”（“湖” 指水域在岛屿内部且不和岛屿周围的水相连）。格子是边长为 1 的正方形。网格为长方形，且宽度和高度均不超过 100 。计算这个岛屿的周长。

示例 1：

输入：grid = [[0,1,0,0],[1,1,1,0],[0,1,0,0],[1,1,0,0]]
输出：16
解释：它的周长是上面图片中的 16 个黄色的边

示例 2：

输入：grid = [[1]]
输出：4

示例 3：

输入：grid = [[1,0]]
输出：4

提示：

```
row == grid.length
col == grid[i].length
1 <= row, col <= 100
grid[i][j] 为 0 或 1
```

```python
class Solution:
    def islandPerimeter(self, grid: List[List[int]]) -> int:
        count = 0
        nrow = len(grid)
        ncol = len(grid[0])
        for i in range(nrow):
            for j in range(ncol):
                if grid[i][j]==1:
```

```
                    count +=4

                    if (i-1>=0) and grid[i-1][j] ==1:
                        count-=1
                    if (i+1<nrow) and grid[i+1][j]==1:
                        count-=1
                    if (j-1>=0) and grid[i][j-1]==1:
                        count-=1
                    if (j+1<ncol) and grid[i][j+1]==1:
                        count-=1
        return count
```

Tips

一个格子的周长是4，如果有1个相邻的格子周长-1，遍历所有格子即可O(m*n)

48. 旋转图像.md

给定一个 n × n 的二维矩阵 matrix 表示一个图像。请你将图像顺时针旋转 90 度。

你必须在 原地 旋转图像，这意味着你需要直接修改输入的二维矩阵。请不要 使用另一个矩阵来旋转图像。

示例 1：

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

示例 2：

输入：matrix = [[5,1,9,11],[2,4,8,10],[13,3,6,7],[15,14,12,16]]
输出：[[15,13,2,5],[14,3,4,1],[12,6,8,9],[16,7,10,11]]

示例 3：

输入：matrix = [[1]]
输出：[[1]]

示例 4：

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

提示：

```
matrix.length == n
matrix[i].length == n
1 <= n <= 20
-1000 <= matrix[i][j] <= 1000
```

1. 使用额外matrix

旋转90度，会发现i行j列，会变成j行倒数i列(n-i-1)

```
class Solution:
    def rotate(self, matrix: List[List[int]]) -> None:
        """
        Do not return anything, modify matrix in-place instead.
        """
        n= len(matrix)
        new_mat = [[0]* n for i in range(n)]
        for i in range(n):
            for j in range(n):
                new_mat[j][n-1-i] = matrix[i][j]
        matrix[:] = new_mat
```

2. 原地变换

难点在于如何不占用额外空间进行变换。把旋转变成两次反转。

先水平翻转(i->n-i-1)

AB

DC.

变成

DC

AB

再按主对角线进行反转i,j -> j,i

DA

CB

```python
class Solution:
    def rotate(self, matrix: List[List[int]]) -> None:
        """
        Do not return anything, modify matrix in-place instead.
        """
        n=len(matrix)
        for i in range(n//2):
            for j in range(n):
                matrix[i][j] ,matrix[n-1-i][j]= matrix[n-1-i][j], matrix[i][j]
        for i in range(n):
            for j in range(i):
                matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]
```

482. 密钥格式化.md

有一个密钥字符串 S ，只包含字母，数字以及 '-'（破折号）。其中， N 个 '-' 将字符串分成了 N+1 组。

给你一个数字 K，请你重新格式化字符串，使每个分组恰好包含 K 个字符。特别地，第一个分组包含的字符个数必须小于等于 K，但至少要包含 1 个字符。两个分组之间需要用 '-'（破折号）隔开，并且将所有的小写字母转换为大写字母。

给定非空字符串 S 和数字 K，按照上面描述的规则进行格式化。

示例 1：

输入：S = "5F3Z-2e-9-w", K = 4
输出："5F3Z-2E9W"
解释：字符串 S 被分成了两个部分，每部分 4 个字符；
　注意，两个额外的破折号需要删掉。

示例 2：

输入：S = "2-5g-3-J", K = 2
输出："2-5G-3J"
解释：字符串 S 被分成了 3 个部分，按照前面的规则描述，第一部分的字符可以少于给定的数量，其余部分皆为 2 个字符。

提示:

```
S 的长度可能很长，请按需分配大小。K 为正整数。
S 只包含字母数字（a-z，A-Z，0-9）以及破折号'-'
S 非空
```

```python
class Solution:
    def licenseKeyFormatting(self, s: str, k: int) -> str:
```

```
        res = []
        cnt = 0
        for i in range(len(s)-1,-1,-1):
            if s[i]=='-':
                continue
            else:
                res.append(s[i].upper())
                cnt +=1
            if cnt%k==0:
                res.append('-')
        if res and res[-1]=='-':
            res.pop()
        return ''.join(res[::-1])
```

Tips

因为第一段位置可变所以从后向前遍历，当凑够4个，这里用的是count%4==0就加入分隔符

485. 最大连续 1 的个数.md

给定一个二进制数组， 计算其中最大连续 1 的个数。

示例：

输入：[1,1,0,1,1,1]
输出：3
解释：开头的两位和最后的三位都是连续 1 ，所以最大连续 1 的个数是 3.

提示：

```
输入的数组只包含 0 和 1 。
输入数组的长度是正整数，且不超过 10,000。
```

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

Tips

常规的cur+max解法，一边向前遍历找max一边计算cur

495. 提莫攻击.md

在《英雄联盟》的世界中，有一个叫 “提莫” 的英雄，他的攻击可以让敌方英雄艾希（编者注：寒冰射手）进入中毒状态。现在，给出提莫对艾希的攻击时间序列和提莫攻击的中毒持续时间，你需要输出艾希的中毒状态总时长。

你可以认为提莫在给定的时间点进行攻击，并立即使艾希处于中毒状态。

示例1:

输入: [1,4], 2
输出: 4
原因: 第 1 秒初，提莫开始对艾希进行攻击并使其立即中毒。中毒状态会维持 2 秒钟，直到第 2 秒末结束。
第 4 秒初，提莫再次攻击艾希，使得艾希获得另外 2 秒中毒时间。
所以最终输出 4 秒。

示例2:

输入: [1,2], 2
输出: 3
原因: 第 1 秒初，提莫开始对艾希进行攻击并使其立即中毒。中毒状态会维持 2 秒钟，直到第 2 秒末结束。
但是第 2 秒初，提莫再次攻击了已经处于中毒状态的艾希。
由于中毒状态不可叠加，提莫在第 2 秒初的这次攻击会在第 3 秒末结束。
所以最终输出 3 。

提示：

```
你可以假定时间序列数组的总长度不超过 10000。
你可以假定提莫攻击时间序列中的数字和提莫攻击的中毒持续时间都是非负整数，并且不超过 10,000,000。
```

```python
class Solution:
    def findPoisonedDuration(self, timeSeries: List[int], duration: int) -> int:
        harm = duration
        pos = timeSeries[0]+duration
        for i in timeSeries[1:]:
            if pos>i:
                harm += i+duration-pos
            else:
                harm+=duration
            pos = i+duration
        return harm
```

Tips

1. 只需要对duration的计算做两种区分，上一次攻击持续的最长时间和当前时间的关系
2. 初始默认计算全部攻击，后面的指补充额外攻击事件

506. 相对名次.md

给出 N 名运动员的成绩，找出他们的相对名次并授予前三名对应的奖牌。前三名运动员将会被分别授予 “金牌”，“银牌” 和“ 铜牌”（"Gold Medal", "Silver Medal", "Bronze Medal"）。

(注：分数越高的选手，排名越靠前。)

示例 1:

输入: [5, 4, 3, 2, 1]
输出: ["Gold Medal", "Silver Medal", "Bronze Medal", "4", "5"]
解释: 前三名运动员的成绩为前三高的，因此将会分别被授予 “金牌”，“银牌”和“铜牌” ("Gold Medal", "Silver Medal" and "Bronze Medal").
余下的两名运动员，我们只需要通过他们的成绩计算将其相对名次即可。

提示:

```
N 是一个正整数并且不会超过 10000。
所有运动员的成绩都不相同。
```

```python
class Solution:
    def findRelativeRanks(self, score: List[int]) -> List[str]:
        n = len(score)
        rank = ["Gold Medal", "Silver Medal", "Bronze Medal"][:n] + list([str(i) for i in range(4,n+1)])
        dic = dict(zip(sorted(score, reverse=True), rank))
        return [dic[i] for i in score]
```

Tips

巧用zip直接生成rank和score的dict，然后按映射直接返回就好

54. 螺旋矩阵.md

给你一个 m 行 n 列的矩阵 matrix ， 请按照 顺时针螺旋顺序 ， 返回矩阵中的所有元素。

示例 1：

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

示例 2：

输入：matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]
输出：[1,2,3,4,8,12,11,10,9,5,6,7]

提示：

```
m == matrix.length
n == matrix[i].length
1 <= m, n <= 10
-100 <= matrix[i][j] <= 100
```

```
class Solution:
    def spiralOrder(self, matrix: List[List[int]]) -> List[int]:
        nrow =len(matrix)
        ncol = len(matrix[0])
        left = 0
        right = ncol-1
        bottom = nrow-1
        top =0
        result = []
        while left< right and top<bottom:
            for i in range(left, right):
                result.append(matrix[top][i])
            for i in range(top,bottom):
                result.append(matrix[i][right])
            for i in range(right, left, -1):
                result.append(matrix[bottom][i])
            for i in range(bottom, top,-1):
                result.append(matrix[i][left])
```

```
            left +=1
            right-=1
            bottom-=1
            top+=1
        print(left, right,bottom, top)
        if bottom==top and left == right:
            result.append(matrix[bottom][left])
        elif bottom==top:
            for i in range(left, right+1):
                result.append(matrix[bottom][i])
        elif left==right:
            for i in range(top, bottom+1):
                result.append(matrix[i][left])
        return result
```

Tips

小时候玩拼图的路子，每行/列，都遍历nrow-1个，这样拼起来刚好遍历一个周长。针对最后中间三种特殊情况，剩下1个数字，剩下一行，剩下一列的情况进行特殊处理。注意这时不再遍历边长-1，所以需要+1处理

551. 学生出勤记录 I.md

给你一个字符串 s 表示一个学生的出勤记录，其中的每个字符用来标记当天的出勤情况（缺勤、迟到、到场）。记录中只含下面三种字符：

```
'A': Absent，缺勤
'L': Late，迟到
'P': Present，到场
```

如果学生能够 同时 满足下面两个条件，则可以获得出勤奖励：

```
按 总出勤 计，学生缺勤（'A'）严格 少于两天。
学生 不会 存在 连续 3 天或 连续 3 天以上的迟到（'L'）记录。
```

如果学生可以获得出勤奖励，返回 true ； 否则，返回 false 。

示例 1：

输入：s = "PPALLP"
输出：true
解释：学生缺勤次数少于 2 次，且不存在 3 天或以上的连续迟到记录。

示例 2：

输入：s = "PPALLL"
输出：false
解释：学生最后三天连续迟到，所以不满足出勤奖励的条件。

提示：

```
1 <= s.length <= 1000
s[i] 为 'A'、'L' 或 'P'
```

```python
class Solution:
    def checkRecord(self, s: str) -> bool:
        count_a = 0
        count_l = 0
        for i in s:
            if i=='A':
                count_a+=1
            if count_a>=2:
                return False
            if i=='L':
                count_l+=1
            else:
                count_l = 0
            if count_l>=3:
                return False
        return True
```

Tips

直接遍历然后分别统计A/L的次数，注意L需要在每次断开的时候重新计数

561. 数组拆分 I.md

给定长度为 2n 的整数数组 nums ， 你的任务是将这些数分成 n 对, 例如 (a1, b1), (a2, b2), ..., (an, bn) ， 使得从 1 到 n 的 min(ai, bi) 总和最大。

返回该 最大总和 。

示例 1：

输入：nums = [1,4,3,2]
输出：4
解释：所有可能的分法（忽略元素顺序）为：

1. (1, 4), (2, 3) -> min(1, 4) + min(2, 3) = 1 + 2 = 3
2. (1, 3), (2, 4) -> min(1, 3) + min(2, 4) = 1 + 2 = 3
3. (1, 2), (3, 4) -> min(1, 2) + min(3, 4) = 1 + 3 = 4
   所以最大总和为 4

示例 2：

输入：nums = [6,2,6,5,1,2]
输出：9
解释：最优的分法为 (2, 1), (2, 5), (6, 6). min(2, 1) + min(2, 5) + min(6, 6) = 1 + 2 + 6 = 9

提示：

```
1 <= n <= 104
nums.length == 2 * n
-104 <= nums[i] <= 104
```

```python
class Solution:
    def arrayPairSum(self, nums: List[int]) -> int:
        nums.sort()
        return sum(nums[::2])
```

Tips

为了更小数字的和最大，所以排序后按顺序输出就是和最大的方案

566. 重塑矩阵.md

在 MATLAB 中，有一个非常有用的函数 reshape ，它可以将一个 m x n 矩阵重塑为另一个大小不同（r x c）的新矩阵，但保留其原始数据。

给你一个由二维数组 mat 表示的 m x n 矩阵，以及两个正整数 r 和 c ，分别表示想要的重构的矩阵的行数和列数。

重构后的矩阵需要将原始矩阵的所有元素以相同的 行遍历顺序 填充。

如果具有给定参数的 reshape 操作是可行且合理的，则输出新的重塑矩阵；否则，输出原始矩阵。

示例 1：

输入：mat = [[1,2],[3,4]], r = 1, c = 4
输出：[[1,2,3,4]]

示例 2：

输入：mat = [[1,2],[3,4]], r = 2, c = 4
输出：[[1,2],[3,4]]

提示：

```
m == mat.length
n == mat[i].length
1 <= m, n <= 100
-1000 <= mat[i][j] <= 1000
1 <= r, c <= 300
```

```python
class Solution:
    def matrixReshape(self, mat: List[List[int]], r: int, c: int) -> List[List[int]]:
        nrow = len(mat)
        ncol = len(mat[0])
        if nrow *ncol!= r*c:
            return mat
        nmat = [[0]*c for i in range(r)]
        print(nmat)
        for i in range(int(nrow*ncol)):
            nmat[i//c][i%c] = mat[i//ncol][i%ncol]
        return nmat
```

Tips

注意在遍历element的时候，row和col的位置都是用列数判断的

57. 插入区间.md

给你一个 无重叠的 ，按照区间起始端点排序的区间列表。

在列表中插入一个新的区间，你需要确保列表中的区间仍然有序且不重叠（如果有必要的话，可以合并区间）。

示例 1：

输入：intervals = [[1,3],[6,9]], newInterval = [2,5]
输出：[[1,5],[6,9]]

示例 2：

输入：intervals = [[1,2],[3,5],[6,7],[8,10],[12,16]], newInterval = [4,8]
输出：[[1,2],[3,10],[12,16]]
解释：这是因为新的区间 [4,8] 与 [3,5],[6,7],[8,10] 重叠。

示例 3：

输入：intervals = [], newInterval = [5,7]
输出：[[5,7]]

示例 4：

输入：intervals = [[1,5]], newInterval = [2,3]
输出：[[1,5]]

示例 5：

输入：intervals = [[1,5]], newInterval = [2,7]
输出：[[1,7]]

提示：

```
0 <= intervals.length <= 104
intervals[i].length == 2
0 <= intervals[i][0] <= intervals[i][1] <= 105
intervals 根据 intervals[i][0] 按 升序 排列
newInterval.length == 2
0 <= newInterval[0] <= newInterval[1] <= 105
```

```python
class Solution:
    def insert(self, intervals: List[List[int]], newInterval: List[int]) ->
List[List[int]]:
        res  = []
        inserted = False
        for it in intervals:
            if it[1] < newInterval[0]:
                res.append(it)
            elif it[0] > newInterval[1]:
                if not inserted:
                    res.append(newInterval)
                    inserted=True
                res.append(it)
            else:
                newInterval[0] = min(it[0], newInterval[0])
                newInterval[1] = max(it[1], newInterval[1])
        if not inserted:
            res.append(newInterval)
        return res
```

Tips

因为interval本身是不重叠且有序的，所以每一步只用判断当前区间和左右边界的关系

1. 如果不重叠，且新区间在右边，把当前区间插入
2. 如果重叠则更新左右边界，保留两侧最大的边界。这一步需要注意不能直接insert更新后的区间，因为需要依赖下一个区间来判断是否需要进一步合并
3. 如果不重叠且新区间在左边，则所有区间已经合并完毕，则把新区间和当前区间一起插入

59. 螺旋矩阵 II.md

给你一个正整数 n ，生成一个包含 1 到 n2 所有元素，且元素按顺时针顺序螺旋排列的 n x n 正方形矩阵 matrix 。

示例 1：

输入：n = 3
输出：[[1,2,3],[8,9,4],[7,6,5]]

示例 2：

输入：n = 1
输出：[[1]]

提示：

```
1 <= n <= 20
```

```
class Solution:
    def generateMatrix(self, n: int) -> List[List[int]]:
        num = 1
        mat = [[0]*n for i in range(n)]
        left =0
        right = n-1
        top=0
        bottom = n-1
        while left<right and top<bottom:
            for c in range(left,right):
                mat[top][c] = num
                num+=1
            for r in range(top,bottom):
                mat[r][right] = num
```

```
                num+=1
            for c in range(right, left,-1):
                mat[bottom][c] = num
                num+=1
            for r in range(bottom, top,-1):
                mat[r][left] = num
                num +=1

            left +=1
            right-=1
            bottom-=1
            top+=1

        if left ==right and  top==bottom:
            mat[left][top]= num
        elif left == right:
            for r in range(top, bottom+1):
                mat[r][left]= num
                num+=1
        else:
            for c in range(left, right+1):
                mat[top][c] = num
                num+=1
        return mat
```

Tips

和上一道螺旋矩阵解法相同

598. 范围求和 II.md

给定一个初始元素全部为 0，大小为 m*n 的矩阵 M 以及在 M 上的一系列更新操作。

操作用二维数组表示，其中的每个操作用一个含有两个正整数 a 和 b 的数组表示，含义是将所有符合 0 <= i < a 以及 0 <= j < b 的元素 M[i][j] 的值都增加 1。

在执行给定的一系列操作后，你需要返回矩阵中含有最大整数的元素个数。

示例 1:

输入:
m = 3, n = 3
operations = [[2,2],[3,3]]
输出: 4
解释:
初始状态, M =
[[0, 0, 0],
 [0, 0, 0],
 [0, 0, 0]]

执行完操作 [2,2] 后, M =
[[1, 1, 0],
 [1, 1, 0],
 [0, 0, 0]]

执行完操作 [3,3] 后, M =
[[2, 2, 1],
 [2, 2, 1],
 [1, 1, 1]]

M 中最大的整数是 2, 而且 M 中有4个值为2的元素。因此返回 4。

注意:

```
m 和 n 的范围是 [1,40000]。
a 的范围是 [1,m]，b 的范围是 [1,n]。
操作数目不超过 10000。
```

```python
class Solution:
    def maxCount(self, m: int, n: int, ops: List[List[int]]) -> int:
        if not ops:
            return m*n
        row = ops[0][0]
        col = ops[0][1]
        for op in ops[1:]:
            row = min(row, op[0])
            col = min(col, op[1])
        return row*col
```

Tips

最大元素个数，就是所有操作重叠的部分，也就是最小row和最小col的面积

599. 两个列表的最小索引总和.md

假设Andy和Doris想在晚餐时选择一家餐厅，并且他们都有一个表示最喜爱餐厅的列表，每个餐厅的名字用字符串表示。

你需要帮助他们用最少的索引和找出他们共同喜爱的餐厅。 如果答案不止一个，则输出所有答案并且不考虑顺序。 你可以假设总是存在一个答案。

示例 1:

输入:
["Shogun", "Tapioca Express", "Burger King", "KFC"]
["Piatti", "The Grill at Torrey Pines", "Hungry Hunter Steakhouse", "Shogun"]
输出: ["Shogun"]

解释: 他们唯一共同喜爱的餐厅是“Shogun”。

示例 2:

输入:
["Shogun", "Tapioca Express", "Burger King", "KFC"]
["KFC", "Shogun", "Burger King"]
输出: ["Shogun"]
解释: 他们共同喜爱且具有最小索引和的餐厅是“Shogun”，它有最小的索引和1(0+1)。

提示:

```
两个列表的长度范围都在 [1, 1000]内。
两个列表中的字符串的长度将在[1，30]的范围内。
下标从0开始，到列表的长度减1。
两个列表都没有重复的元素。
```

```python
class Solution:
    def findRestaurant(self, list1: List[str], list2: List[str]) -> List[str]:
        dic = {}
        for i,j in enumerate(list1):
            dic[j]=i

        index = 2**32-1
        ans = []
        for i,j in enumerate(list2):
            if j in dic:
                if dic[j]+i <index:
                    ans = [j]
                    index = dic[j]+i
                elif dic[j]+i == index:
                    ans.append(j)
        return ans
```

Tips

双列表遍历问题，可以类比sql中的sort merge join，这时最优的方案是去遍历其中的一张表，把另一张表写成hash。这让空间复杂度是O(n1), 时间复杂度是O（n1+n2）。选择把那一张表写成hash，这个就是看你用时间换空间还是空间换时间了。spark里会默认把小表写成hash，因为需要做broadcast分发需要考虑分发成本，和分发到executor的内存占用

6. Z 字形变换.md

将一个给定字符串 s 根据给定的行数 numRows ，以从上往下、从左到右进行 Z 字形排列。

比如输入字符串为 "PAYPALISHIRING" 行数为 3 时，排列如下：

P A H N
A P L S I I G
Y I R

之后，你的输出需要从左往右逐行读取，产生出一个新的字符串，比如："PAHNAPLSIIGYIR"。

请你实现这个将字符串进行指定行数变换的函数：

string convert(string s, int numRows);

示例 1：

输入：s = "PAYPALISHIRING", numRows = 3
输出："PAHNAPLSIIGYIR"

示例 2：

输入：s = "PAYPALISHIRING", numRows = 4
输出："PINALSIGYAHRPI"
解释：
P I N
A LS I G
YA H R
P I

示例 3：

输入：s = "A", numRows = 1
输出："A"

提示：

```
1 <= s.length <= 1000
s 由英文字母（小写和大写）、',' 和 '.' 组成
1 <= numRows <= 1000
```

```python
class Solution:
    def convert(self, s: str, numRows: int) -> str:
        if numRows<=1:
            return s
```

```
        result = ''
        n = len(s)
        index = 0
        gap = 2*(numRows-1)
        for i in range(numRows):
            if (i==0) or (i==numRows-1):
                index=i
                while index<n:
                    result+=s[index]
                    index+=gap
            else:
                index=i
                gap1 = gap-2*i
                gap2 = 2*i
                while index<n:
                    result+=s[index]
                    index+=gap1
                    if index <n:
                        result+=s[index]
                        index+=gap2
                    else:
                        break

        return result
```

Tips

1. 感觉像在做奥数pattern题
    1. 第一行和最后一行string之间相差gap= 2*(numRows-1)
    2. 中间行有点复杂，会有两个gap，gap1=gap-2*i*,*gap2*=2i,交替

643. 子数组最大平均数 I.md

给你一个由 n 个元素组成的整数数组 nums 和一个整数 k 。

请你找出平均数最大且 长度为 k 的连续子数组，并输出该最大平均数。

任何误差小于 10-5 的答案都将被视为正确答案。

示例 1：

输入：nums = [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

示例 2：

输入：nums = [5], k = 1
输出：5.00000

提示：

```
n == nums.length
1 <= k <= n <= 105
-104 <= nums[i] <= 104
```

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

Tips

稍微有一点技巧就是求window内的K个数字之和时可以滚动计算，不用每次都求sum

657. 机器人能否返回原点.md

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。

示例 1:

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。

示例 2:

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。

```
class Solution:
    def judgeCircle(self, moves: str) -> bool:
        h =0
        v =0
        for i in moves:
            if i=='U':
                v+=1
            elif i=='D':
                v-=1
            elif i=='L':
                h+=1
            else:
                h-=1
        return h==0 and v==0
```

661. 图片平滑器.md

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。

示例 1:

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

注意:

```
给定矩阵中的整数范围为 [0, 255]。
矩阵的长和宽的范围均为 [1, 150]。
```

```python
class Solution:
    def imageSmoother(self, img: List[List[int]]) -> List[List[int]]:
        nrow = len(img)
        ncol = len(img[0])
        ans = [[0] *ncol for i in range(nrow)]

        for r in range(nrow):
            for c in range(ncol):
                count = 0
                for i in (r-1, r,r+1):
                    for j in (c-1,c,c+1):
                        if i>=0 and i<nrow and j>=0 and j<ncol:
                            ans[r][c] += img[i][j]
                            count+=1
                ans[r][c]//=count
        return ans
```

Tips

没啥可说的遍历就完了，就是需要判断下index是否valid

696. 计数二进制子串.md

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。

示例 1 :

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。

示例 2 :

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。

提示：

```
s.length 在1到50,000之间。
s 只包含“0”或“1”字符。
```

```python
class Solution:
    def countBinarySubstrings(self, s: str) -> int:
        counter = []
        cur = 1
        for i,c in enumerate(s[1:]):
            if c==s[i]:
                cur+=1
            else:
                counter.append(cur)
                cur =1
        counter.append(cur)
        print(counter)
        total = 0
        for i in range(1,len(counter)):
            total += min(counter[i],counter[i-1])
        return total
```

Tips

重复子串的个数，00111包含两个，也就是子串数=min（count0,count1），所以先保留所有连续0/1的个数，然后相邻两个取min再求和即可

优化版就是counter只保存2位，把0/1交替存到counter[0]和counter[1]

724. 寻找数组的中心下标.md

给你一个整数数组 nums ，请计算数组的 中心下标 。

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。

如果中心下标位于数组最左端，那么左侧数之和视为 0 ，因为在下标的左侧不存在元素。这一点对于中心下标位于数组最右端同样适用。

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

示例 1：

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 sum = nums[0] + nums[1] + nums[2] = 1 + 7 + 3 = 11 ，
右侧数之和 sum = nums[4] + nums[5] = 5 + 6 = 11 ，二者相等。

示例 2：

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。

示例 3：

输入：nums = [2, 1, -1]
输出：0
解释：
中心下标是 0 。
左侧数之和 sum = 0 ，（下标 0 左侧不存在元素），
右侧数之和 sum = nums[1] + nums[2] = 1 + -1 = 0 。

提示：

```
1 <= nums.length <= 104
-1000 <= nums[i] <= 1000
```

```python
class Solution:
    def pivotIndex(self, nums: List[int]) -> int:
        total = sum(nums)
        left = 0
        for i in range(len(nums)):
            if left == total -left - nums[i]:
                return i
            left += nums[i]
        return -1
```

Tips

就读懂题目遍历就好，right=total-left-i，每个位置判断left是否和right相同就好

73. 矩阵置零.md

给定一个 m x n 的矩阵，如果一个元素为 0 ，则将其所在行和列的所有元素都设为 0 。请使用 原地 算法。

进阶：

```
一个直观的解决方案是使用  O(mn) 的额外空间，但这并不是一个好的解决方案。
一个简单的改进方案是使用 O(m + n) 的额外空间，但这仍然不是最好的解决方案。
你能想出一个仅使用常量空间的解决方案吗?
```

示例 1：

输入：matrix = [[1,1,1],[1,0,1],[1,1,1]]
输出：[[1,0,1],[0,0,0],[1,0,1]]

示例 2：

输入：matrix = [[0,1,2,0],[3,4,5,2],[1,3,1,5]]
输出：[[0,0,0,0],[0,4,5,0],[0,3,1,0]]

提示：

```
m == matrix.length
n == matrix[0].length
1 <= m, n <= 200
-231 <= matrix[i][j] <= 231 - 1
```

```python
class Solution:
    def setZeroes(self, matrix: List[List[int]]) -> None:
        """
        Do not return anything, modify matrix in-place instead.
        """
        nrow = len(matrix)
        ncol = len(matrix[0])
        firstrow = any([i==0 for i in matrix[0]])
        firstcol = any([matrix[i][0] ==0 for i in range(nrow)])

        for i in range(1, nrow):
            for j in range(1, ncol):
                if matrix[i][j]==0:
                    matrix[i][0] =0
                    matrix[0][j] =0

        for i in range(1, nrow):
            for j in range(1, ncol):
                if matrix[i][0] ==0:
                    matrix[i][j] = 0
                if matrix[0][j]==0:
                    matrix[i][j] = 0

        if firstrow:
            for i in range(ncol):
                matrix[0][i] = 0
```

```
        if firstcol:
            for i in range(nrow):
                matrix[i][0] = 0
```

Tips

这道题的难点在于需要使用常量空间。所以这里使用矩阵的第一行和第一列来存放第i行/列是否应该被置为0.再用两个常量来判断第一行和第一列本身是否应该都被置为0

747. 至少是其他数字两倍的最大数.md

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 。

示例 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 。

示例 2：

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 。

示例 3：

输入：nums = [1]
输出：0
解释：因为不存在其他数字，所以认为现有数字 1 至少是其他数字的两倍。

提示：

```
1 <= nums.length <= 50
0 <= nums[i] <= 100
nums 中的最大元素是唯一的
```

```
class Solution:
    def dominantIndex(self, nums: List[int]) -> int:
        first = 0
        second = 0
        index = 0
```

```
        for pos, i in enumerate(nums):
            if i > first:
                first, second = i, first
                index = pos
            elif i > second:
                second = i
        if first>=2*second :
            return index
        else:
            return -1
```

Tips

至少是其他数字的两倍大，所以需要保留最大而第二大的两个数字

766. 托普利茨矩阵.md

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 。

示例 1：

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 。

示例 2：

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

提示：

```
m == matrix.length
n == matrix[i].length
1 <= m, n <= 20
0 <= matrix[i][j] <= 99
```

进阶：

> 如果矩阵存储在磁盘上，并且内存有限，以至于一次最多只能将矩阵的一行加载到内存中，该怎么办？
> 如果矩阵太大，以至于一次只能将不完整的一行加载到内存中，该怎么办？

```python
class Solution:
    def isToeplitzMatrix(self, matrix: List[List[int]]) -> bool:
        nrow = len(matrix)
        ncol = len(matrix[0])
        for i in range(1, nrow):
            for j in range(1,ncol):
                if matrix[i][j] != matrix[i-1][j-1]:
                    return False
        return True
```

Tips

如果只能把一行放入矩阵，就每次只放一行，然后遍历下一行

如果只能放部分就切分小矩阵，每个小矩阵都满足条件大矩阵就满足

821. 字符的最短距离.md

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。

示例 1：

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) = 2 。
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 。

示例 2：

输入：s = "aaab", c = "b"
输出：[3,2,1,0]

提示：

```
1 <= s.length <= 104
s[i] 和 c 均为小写英文字母
题目数据保证 c 在 s 中至少出现一次
```

```python
class Solution:
    def shortestToChar(self, s: str, c: str) -> List[int]:
        prev = -10**4
        ans = []
        for i,j in enumerate(s):
            if j==c:
                prev=i
            ans.append(i-prev)
        print(ans)
        prev = 10**4
        for i in range(len(s)-1,-1,-1):
            if s[i]==c:
                prev = i
            ans[i] = min(ans[i],prev-i)
        print(ans)
        return ans
```

Tips

1. 在需要比较左右的问题中，永远先只比较一边。于是先从左向右遍历，得到左边距离最近的，再从右向左遍历用右边距离最近的更新左边
2. 双指针解法，一个指向previous，一个指向当前

830. 较大分组的位置.md

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。

示例 1：

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。

示例 2：

输入：s = "abc"
输出：[]
解释："a","b" 和 "c" 均不是符合要求的较大分组。

示例 3：

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

示例 4：

输入：s = "aba"
输出：[]

提示：

```
1 <= s.length <= 1000
s 仅含小写英文字母
```

```python
class Solution:
    def largeGroupPositions(self, s: str) -> List[List[int]]:
        ans = []
        start = 0
        end =0

        for i in range(1,len(s)):
            if s[i]==s[i-1]:
                end+=1
            else:
                if end-start>=2:
                    ans.append([start,end])
                start = i
                end = i
        if end-start>=2:
            ans.append([start,end])
        return ans
```

Tips

和228题很像，不要忘记尾部情况，不要忘记尾部，不要忘记尾部。所有需要对前一个位置进行判断的case都可能会碰到尾部情况。

832. 翻转图像.md

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]。

示例 1：

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]]

示例 2：

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

提示：

```
1 <= A.length = A[0].length <= 20
0 <= A[i][j] <= 1
```

```python
class Solution:
    def flipAndInvertImage(self, image: List[List[int]]) -> List[List[int]]:
        nrow = len(image)
        ncol = len(image[0])
        for i in range(nrow):
            for j in range(ncol//2):
                image[i][j],image[i][ncol-j-1] = 1-image[i][ncol-j-1],1-image[i][j]
            if ncol %2:
                image[i][ncol//2] = 1-image[i][ncol//2]
        return image
```