苏州大学python复试上机—苏大老师推荐LeetCode习题【来自mooc】

LeetCode

953. 验证外星语词典 [字符串]
283. 移动零 [数组]
1078. Bigram 分词 [字符串]
1337. 方阵中战斗力最弱的 K 行 [数组]
929. 独特的电子邮件地址 [字符串]
883. 三维形体投影面积 [数学]
575. 分糖果 [哈希表]
118. 杨辉三角 [数组]
509. 斐波那契数 [数组]
521. 最长特殊序列 Ⅰ [字符串]
821. 字符的最短距离 [字符串]
1002. 查找常用字符 [数组 哈希表]
811. 子域名访问计数 [哈希表 字符串]
485. 最大连续1的个数 [数组]
748. 最短完整词 [哈希表 字符串]
171. Excel表列序号 [数组]
168. Excel表列名称
922. 按奇偶排序数组 II [数组]
908. 最小差值 I [数学]
476. 数字的补数 [数学]
905. 按奇偶排序数组 [数组]
561. 数组拆分 I [数组]
面试题 01.01. 判定字符是否唯一 [数组]
657. 机器人能否返回原点 [字符串]
832. 翻转图像 [数组]
804. 唯一摩尔斯密码词 [字符串]
461. 汉明距离 [数学]
1252. 奇数值单元格的数目 [数组]
1323. 6 和 9 组成的最大数字 [数学]
面试题17. 打印从1到最大的n位数 [数学]
771. 宝石与石头 [哈希表]
1046. 最后一块石头的重量 [堆 贪心算法]
1287. 有序数组中出现次数超过25%的元素 [数组]
941. 有效的山脉数组 [数组]
1281. 整数的各位积和之差 [数学]
888. 公平的糖果交换 [数组]
409. 最长回文串 [字符串]

860. 柠檬水找零 [数组]
844. 比较含退格的字符串 [字符串]
1276. 不浪费原料的汉堡制作方案 【贪心]
263. 丑数 [数学]
628. 三个数的最大乘积[数组]
824. 山羊拉丁文 [字符串]
925. 长按键入 [字符串]
1184. 公交站间的距离 [数组]
35. 二分算法
290. 单词规律 [字符串]
914. 卡牌分组[数组]
645. 错误的集合[数组]
560. 和为K的子数组[数组 哈希表]
46. 全排列
Mooc 05_4 集合操作(a < b: a是b的子集)

苏州大学python复试上机—苏大老师推荐LeetCode习题【来自mooc】

LeetCode

953. 验证外星语词典 [字符串]

某种外星语也使用英文小写字母，但可能顺序 `order` 不同。字母表的顺序（ `order` ）是一些小写字母的排列。

给定一组用外星语书写的单词 `words`，以及其字母表的顺序 `order`，只有当给定的单词在这种外星语中按字典序排列时，返回 `true`；否则，返回 `false`。

示例 1：

```
输入：words = ["hello","leetcode"], order = "hlabcdefgijkmnopqrstuvwxyz"
输出：true
解释：在该语言的字母表中，'h' 位于 'l' 之前，所以单词序列是按字典序排列的。
```

示例 2：

输入：words = ["word","world","row"], order = "worldabcefghijkmnpqstuvxyz"
输出：false
解释：在该语言的字母表中，'d' 位于 'l' 之后，那么 words[0] > words[1]，因此单词序列不是按字典序排列的。

示例 3：

输入：words = ["apple","app"], order = "abcdefghijklmnopqrstuvwxyz"
输出：false
解释：当前三个字符 "app" 匹配时，第二个字符串相对短一些，然后根据词典编纂规则 "apple" > "app"，因为 'l' > '∅'，其中 '∅' 是空白字符，定义为比任何其他字符都小（更多信息）。

提示：

1. `1 <= words.length <= 100`
2. `1 <= words[i].length <= 20`
3. `order.length == 26`
4. 在 `words[i]` 和 `order` 中的所有字符都是英文小写字母。

代码：

```
# -*- coding: utf-8 -*-

class Solution:
    def isAlienSorted(self, words, order) -> bool:
        dic = {}
        for e in order:
            dic[e] = dic.get(e, 0) + order.index(e)

        wlen = len(words)
        flag = 0
        for i in range(0, wlen):
            A = words[i]
            Alen = len(A)
            for j in range(i + 1, wlen):
                B = words[j]
                Blen = len(B)
                mlen = Alen if Alen < Blen else Blen     # 小的长度
                for k in range(mlen):
#                     print(A[k], B[k])
```

```
                    if dic[A[k]] < dic[B[k]]:
                        flag = 1
                        break
                    elif dic[A[k]] > dic[B[k]]:
                        return False
        if flag:
            return True
        else:
            return False
        # return words == sorted(words, key=lambda w:[order.index(x) for
x in w])

s = Solution()

print(s.isAlienSorted(words=["hello","leetcode"],
order="hlabcdefgijkmnopqrstuvwxyz"))
print(s.isAlienSorted(words = ["word","world","row"], order =
"worldabcefghijkmnpqstuvxyz"))
print(s.isAlienSorted(words = ["apple","app"], order =
"abcdefghijklmnopqrstuvwxyz"))
print(s.isAlienSorted(["iekm","tpnhnbe"], "loxbzapnmstkhijfcuqdewyvrg"))
print(s.isAlienSorted(["fxasxpc","dfbdrifhp","nwzgs","cmwqriv","ebulyfyve
","miracx","sxckdwzv","dtijzluhts","wwbmnge","qmjwymmyox"]
, "zkgwaverfimqxbnctdplsjyohu"))
print(s.isAlienSorted(["kuvp","q"], "ngxlkthsjuoqcpavbfdermiywz"))
```

[283. 移动零](#) [数组]

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

示例:

```
输入: [0,1,0,3,12]
输出: [1,3,12,0,0]
```

说明:

1. 必须在原数组上操作，不能拷贝额外的数组。
2. 尽量减少操作次数。

代码：

```
class Solution:
    def moveZeroes(self, nums) -> None:
        for e in nums[::]:
            if e == 0:
                nums.remove(e)
                nums.append(0)
        return nums
s = Solution()
print(s.moveZeroes([0, 1, 0, 3, 12]))
```

1078. Bigram 分词 [字符串]

给出第一个词 `first` 和第二个词 `second`，考虑在某些文本 `text` 中可能以 "`first second third`" 形式出现的情况，其中 `second` 紧随 `first` 出现，`third` 紧随 `second` 出现。

对于每种这样的情况，将第三个词 "`third`" 添加到答案中，并返回答案。

示例 1：

```
输入：text = "alice is a good girl she is a good student", first = "a", second = "good"
输出：["girl","student"]
```

示例 2：

```
输入：text = "we will we will rock you", first = "we", second = "will"
输出：["we","rock"]
```

提示：

1. `1 <= text.length <= 1000`
2. `text` 由一些用空格分隔的单词组成，每个单词都由小写英文字母组成
3. `1 <= first.length, second.length <= 10`
4. `first` 和 `second` 由小写英文字母组成

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def findOcurrences(self, text: str, first: str, second: str): #->
List[str]:
        words = text.split(' ')
```

```
        res = []

        wlen = len(words)
        for i in range(0, wlen - 2):
            if words[i] == first and words[i + 1] == second:
                res.append(words[i + 2])
        return res

s = Solution()
print(s.findOcurrences(text = "alice is a good girl she is a good
student", first = "a", second = "good"))
print(s.findOcurrences(text = "we will we will rock you", first = "we",
second = "will"))
```

1337. 方阵中战斗力最弱的 K 行 [数组]

给你一个大小为 `m * n` 的方阵 `mat`，方阵由若干军人和平民组成，分别用 1 和 0 表示。

请你返回方阵中战斗力最弱的 `k` 行的索引，按从最弱到最强排序。

如果第 *i* 行的军人数量少于第 *j* 行，或者两行军人数量相同但 *i* 小于 *j*，那么我们认为第 *i* 行的战斗力比第 *j* 行弱。

军人 **总是** 排在一行中的靠前位置，也就是说 1 总是出现在 0 之前。

示例 1：

```
输入：mat =
[[1,1,0,0,0],        # 0
 [1,1,1,1,0],
 [1,0,0,0,0],        # 2
 [1,1,0,0,0],        # 3
 [1,1,1,1,1]],
k = 3
输出：[2,0,3]
```

示例 2：

```
输入：mat =
[[1,0,0,0],
 [1,1,1,1],
 [1,0,0,0],
 [1,0,0,0]],
k = 2
输出：[0,2]
```

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def kWeakestRows(self, mat, k: int): # -> List[int]:
        res = []
        mlen = len(mat)
        for i in range(mlen):
            res.append((mat[i].count(1), i))

        res.sort(key=lambda x:x[0])
        res = [e[1] for e in res[:k]]
        return res

s = Solution()
print(s.kWeakestRows(mat = [[1,1,0,0,0],
                            [1,1,1,1,0],
                            [1,0,0,0,0],
                            [1,1,0,0,0],
                            [1,1,1,1,1]], k = 3))

print(s.kWeakestRows(mat = [[1,0,0,0],
                            [1,1,1,1],
                            [1,0,0,0],
                            [1,0,0,0]], k = 2))
```

929. 独特的电子邮件地址 [字符串]

每封电子邮件都由一个本地名称和一个域名组成，以 @ 符号分隔。

例如，在 `alice@leetcode.com`中， `alice` 是本地名称，而 `leetcode.com` 是域名。

除了小写字母，这些电子邮件还可能包含 `'.'` 或 `'+'`。

如果在电子邮件地址的**本地名称**部分中的某些字符之间添加句点（`'.'`），则发往那里的邮件将会转发到本地名称中没有点的同一地址。例如，`"alice.z@leetcode.com”` 和 `“alicez@leetcode.com”` 会转发到同一电子邮件地址。（请注意，此规则不适用于域名。）

如果在**本地名称**中添加加号（`'+'`），则会忽略第一个加号后面的所有内容。这允许过滤某些电子邮件，例如 `m.y+name@email.com` 将转发到 `my@email.com`。（同样，此规则不适用于域名。）

可以同时使用这两个规则。

给定电子邮件列表 `emails`，我们会向列表中的每个地址发送一封电子邮件。实际收到邮件的不同地址有多少？

示例：

```
输入：
["test.email+alex@leetcode.com","test.e.mail+bob.cathy@leetcode.com","tes
temail+david@lee.tcode.com"]
输出：2
解释：实际收到邮件的是 "testemail@leetcode.com" 和
"testemail@lee.tcode.com"。

提示：
1 <= emails[i].length <= 100
1 <= emails.length <= 100
每封 emails[i] 都包含有且仅有一个 '@' 字符。
```

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def numUniqueEmails(self, emails) -> int:
        import re
        eset = set()
        for email in emails:
            s = email.split('@')
            s = s[0].replace('.', '') + '@' + s[1]
            s = re.sub('\+.*@', '@', s)
            eset.add(s)
        print(eset)
        return len(eset)
```

```
s = Solution()
print(s.numUniqueEmails(["test.email+alex@leetcode.com",
                         "test.e.mail+bob.cathy@leetcode.com",
                         "testemail+david@lee.tcode.com"]))
```

883. 三维形体投影面积 [数学]

在 `N * N` 的网格中，我们放置了一些与 x，y，z 三轴对齐的 `1 * 1 * 1` 立方体。

每个值 `v = grid[i][j]` 表示 `v` 个正方体叠放在单元格 `(i, j)` 上。

现在，我们查看这些立方体在 xy、yz 和 zx 平面上的*投影*。

投影就像影子，将三维形体映射到一个二维平面上。

在这里，从顶部、前面和侧面看立方体时，我们会看到“影子”。

返回所有三个投影的总面积。

示例 1：

```
输入：[[2]]
输出：5
```

示例 2：

```
输入：[[1,2],[3,4]]
输出：17
解释：
这里有该形体在三个轴对齐平面上的三个投影（“阴影部分”）。
```

示例 3：

```
输入：[[1,0],[0,2]]
输出：8
```

示例 4：

```
输入：[[1,1,1],[1,0,1],[1,1,1]]
输出：14
```

示例 5：

```
输入：[[2,2,2],[2,1,2],[2,2,2]]
输出：21
```

提示：

- `1 <= grid.length = grid[0].length <= 50`
- `0 <= grid[i][j] <= 50`

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def projectionArea(self, grid):
        x_area = 0
        for x in grid:
            x = [e for e in x if e > 0]
            x_area += len(x)

        y_area = 0
        elems = list(zip(*grid))
        for e in elems:
            y_area += max(e)

        z_area = 0
        for e in grid:
            z_area += max(e)

        areas = x_area + y_area + z_area
        return areas

s = Solution()
print(s.projectionArea([[1,2], [3, 4]]))
print(s.projectionArea([[1,0], [0,2]]))
```

575. 分糖果 [哈希表]

难度简单62

给定一个**偶数**长度的数组，其中不同的数字代表着不同种类的糖果，每一个数字代表一个糖果。你需要把这些糖果**平均**分给一个弟弟和一个妹妹。返回妹妹可以获得的最大糖果的种类数。

示例 1:

```
输入: candies = [1,1,2,2,3,3]
输出: 3
解析: 一共有三种种类的糖果，每一种都有两个。
      最优分配方案：妹妹获得[1,2,3],弟弟也获得[1,2,3]。这样使妹妹获得糖果的种类数最多。
```

示例 2 :

```
输入: candies = [1,1,2,3]
输出: 2
解析: 妹妹获得糖果[2,3],弟弟获得糖果[1,1]，妹妹有两种不同的糖果，弟弟只有一种。这样使得妹妹可以获得的糖果种类数最多。
```

代码：

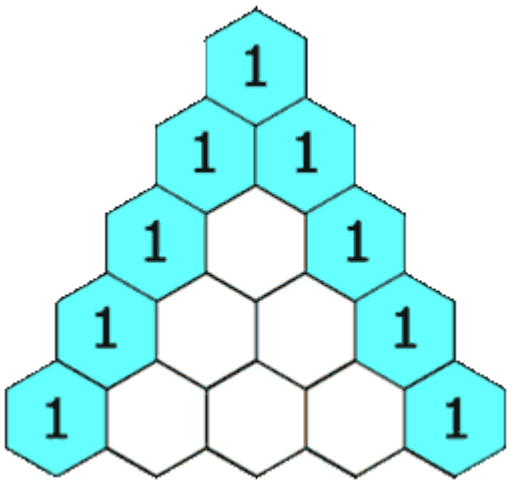
```
# -*- coding: utf-8 -*-
class Solution:
    def distributeCandies(self, candies) -> int:
        clen = len(candies)
        ave = clen // 2

        cset = set()
        for e in candies:
            cset.add(e)
            if len(cset) == ave:
                break
#         print(cset)
        return len(cset)

s = Solution()
print(s.distributeCandies(candies = [1,1,2,2,3,3]))
print(s.distributeCandies(candies = [1,1,2,3]))
```

118. 杨辉三角 [数组]

给定一个非负整数 *numRows*，生成杨辉三角的前 *numRows* 行。



在杨辉三角中，每个数是它左上方和右上方的数的和。

示例:

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

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def generate(self, numRows: int):
        if numRows == 0:
            return []
        if numRows == 1:
            return [[1]]
        if numRows == 2:
            return [[1], [1, 1]]

        res = [[] for i in range(numRows)]
        res[0].append(1)
        res[1] = [1, 1]
        cnt = 1
        for i in range(2, numRows):
```

```
            res[i].append(1)
            for j in range(cnt):
                res[i].append(res[i-1][j] + res[i-1][j+1])
            res[i].append(1)
            cnt = cnt + 1

        print(res)
        return res

s = Solution()
print(s.generate(2))
print(s.generate(5))
```

509. 斐波那契数 [数组]

斐波那契数，通常用 `F(n)` 表示，形成的序列称为**斐波那契数列**。该数列由 `0` 和 `1` 开始，后面的每一项数字都是前面两项数字的和。也就是：

```
F(0) = 0,   F(1) = 1
F(N) = F(N - 1) + F(N - 2), 其中 N > 1.
```

给定 `N`，计算 `F(N)`。

示例 1：

```
输入：2
输出：1
解释：F(2) = F(1) + F(0) = 1 + 0 = 1.
```

示例 2：

```
输入：3
输出：2
解释：F(3) = F(2) + F(1) = 1 + 1 = 2.
```

示例 3：

```
输入：4
输出：3
解释：F(4) = F(3) + F(2) = 2 + 1 = 3.
```

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def fib(self, N: int) -> int:
        if N == 0:
            return 0
        elif N == 1:
            return 1

        a, b = 0, 1
        c = 0
        for i in range(N-1):
            c = a + b
            a = b
            b = c
        return c

s = Solution()
print(s.fib(2))
print(s.fib(3))
print(s.fib(4))
```

521. 最长特殊序列 Ⅰ [字符串]

给定两个字符串，你需要从这两个字符串中找出最长的特殊序列。最长特殊序列定义如下：该序列为某字符串独有的最长子序列（即不能是其他字符串的子序列）。

子序列可以通过删去字符串中的某些字符实现，但不能改变剩余字符的相对顺序。空序列为所有字符串的子序列，任何字符串为其自身的子序列。

输入为两个字符串，输出最长特殊序列的长度。如果不存在，则返回 -1。

示例：

```
输入: "aba", "cdc"
输出: 3
解析: 最长特殊序列可为 "aba" (或 "cdc")
```

说明: 两个字符串长度均小于100。字符串中的字符仅含有 'a'~'z'。

代码：

```
# -*- coding: utf-8 -*-
# 思路：A,B如果完全一样，输出就是 -1，如果不一样，那直接输出A,B中较长的字符串即可
class Solution:
    def findLUSlength(self, a: str, b: str) -> int:
        return max(len(a), len(b)) if a != b else -1

s = Solution()
print(s.findLUSlength('aba', 'cdc'))
```

821. 字符的最短距离 [字符串]

给定一个字符串 `S` 和一个字符 `C`。返回一个代表字符串 `S` 中每个字符到字符串 `S` 中的字符 `C` 的最短距离的数组。

示例 1:

```
输入: S = "loveleetcode", C = 'e'
输出: [3, 2, 1, 0, 1, 0, 0, 1, 2, 2, 1, 0]
```

说明: 1. 字符串 `S` 的长度范围为 `[1, 10000]`。`C` 是一个单字符，且保证是字符串 `S` 里的字符。`S` 和 `C` 中的所有字母均为小写字母。

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def shortestToChar(self, S: str, C: str):
        index = [i for i in range(len(S)) if S[i] is C]

        res = []
        slen = len(S)
        for i in range(slen):
            rmin = 100000
            for j in index:
                dis = abs(i - j)
                if rmin > dis:
                    rmin = dis
            res.append(rmin)

        print(res)
        return res
```

```
s = Solution()
print(s.shortestToChar(S = "loveleetcode", C = 'e'))
```

1002. 查找常用字符 [数组 哈希表]

给定仅有小写字母组成的字符串数组 `A`，返回列表中的每个字符串中都显示的全部字符（**包括重复字符**）组成的列表。例如，如果一个字符在每个字符串中出现 3 次，但不是 4 次，则需要在最终答案中包含该字符 3 次。

你可以按任意顺序返回答案。

示例 1：

```
输入：["bella","label","roller"]
输出：["e","l","l"]
```

示例 2：

```
输入：["cool","lock","cook"]
输出：["c","o"]
```

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def commonChars(self, A):
        res = list()
        for w in set(A[0]):                    # 遍历每个字符
            cnt = [x.count(w) for x in A]    # 每个单词的 字符w 数量
            a = w * min(cnt)
            for i in a:                        # 防止不是全部都出现的，且分开添加
                res.append(i)
        return res

s = Solution()
print(s.commonChars(["bella","label","roller"]))
print(s.commonChars(["cool","lock","cook"]))
```

811. 子域名访问计数 [哈希表 字符串]

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"]

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"]
```

注意事项：

- `cpdomains`的长度小于`100`。每个域名的长度小于`100`。个域名地址包含一个或两个"."符号。
- 输入中任意一个域名的访问次数都小于`10000`。

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def subdomainVisits(self, cpdomains):
        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()]

s = Solution()
print(s.subdomainVisits(["9001 discuss.leetcode.com"]))
print(s.subdomainVisits(["900 google.mail.com",
                         "50 yahoo.com",
                         "1 intel.mail.com",
                         "5 wiki.org"]))
# ["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。
```

485. 最大连续1的个数 [数组]

给定一个二进制数组， 计算其中最大连续1的个数。

示例 1:

```
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.
```

注意：

- 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。

代码：

```
class Solution:
    def findMaxConsecutiveOnes(self, nums) -> int:
        mcnt = 0
        cnt = 0
        for e in nums:
            if mcnt < cnt:
                mcnt = cnt
            if e == 1:
```

```
                cnt += 1
            elif e == 0:
                cnt = 0
        mcnt = cnt if cnt > mcnt else mcnt
        return mcnt

s = Solution()
print(s.findMaxConsecutiveOnes([1, 1, 0, 1, 1, 1]))
```

748. 最短完整词 [哈希表 字符串]

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

单词在匹配牌照中的字母时不区分大小写，比如牌照中的 `"P"` 依然可以匹配单词中的 `"p"` 字母。

我们保证一定存在一个最短完整词。当有多个单词都符合最短完整词的匹配条件时取单词列表中最靠前的一个。

牌照中可能包含多个相同的字符，比如说：对于牌照 `"PP"`，单词 `"pair"` 无法匹配，但是 `"supper"` 可以匹配。

示例 1：

```
输入：licensePlate = "1s3 PSt", words = ["step", "steps", "stripe",
"stepple"]
输出："steps"
说明：最短完整词应该包括 "s"、"p"、"s" 以及 "t"。对于 "step" 它只包含一个 "s" 所以
它不符合条件。同时在匹配过程中我们忽略牌照中的大小写。
```

示例 2：

```
输入：licensePlate = "1s3 456", words = ["looks", "pest", "stew", "show"]
输出："pest"
说明：存在 3 个包含字母 "s" 且有着最短长度的完整词，但我们返回最先出现的完整词。
```

注意: 牌照（`licensePlate`）的长度在区域 `[1, 7]` 中。牌照（`licensePlate`）将会包含数字、空格、或者字母（大写和小写）。单词列表（`words`）长度在区间 `[10, 1000]` 中。每一个单词 `words[i]` 都是小写，并且长度在区间 `[1, 15]` 中。

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def shortestCompletingWord(self, licensePlate: str, words) -> str:
        import re
        awords = ''.join(re.findall('[a-zA-Z]+', licensePlate.lower()))
        # 按 长度,索引 排序
        words = sorted(words, key=lambda x:(len(x), words.index(x)))
        for word in words:
            for e in awords:
#                print(awords, word, e, awords.count(e), word.count(e))
                                # 如果 word中没有e, awords中e数量 > word中e
数量
                if e not in word or awords.count(e) > word.count(e):
                    break
            else:
                return word
s = Solution()
print(s.shortestCompletingWord(licensePlate = "1s3 PSt", words = ["step",
"steps", "stripe", "stepple"]))
print(s.shortestCompletingWord(licensePlate = "1s3 456", words =
["looks", "pest", "stew", "show"]))
print(s.shortestCompletingWord("GrC8950",
["measure","other","every","base","according","level","meeting","none","m
arriage","rest"]))
print(s.shortestCompletingWord("1s3 456",
["looks","pest","stew","show"]))
```

171. Excel表列序号 [数组]

给定一个Excel表格中的列名称，返回其相应的列序号。

例如，

```
    A -> 1
    B -> 2
    C -> 3
    ...
    Z -> 26
    AA -> 27
    AB -> 28
    ...
```

示例 1:

```
输入: "A"
输出: 1
```

示例 2:

```
输入: "AB"
输出: 28
```

示例 3:

```
输入: "ZY"
输出: 701
```

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def titleToNumber(self, s: str) -> int:
        dic = {}
        for i in range(0, 26):
            dic[chr(ord('A')+i)] = i + 1

        res = 0
        slen = len(s)
        for i in range(slen):
            res += dic[s[i]] * 26**len(s[i+1:])
        return res

s = Solution()
print(s.titleToNumber('A'))
print(s.titleToNumber('ZY'))
print(s.titleToNumber('AAA'))
```

168. Excel表列名称

给定一个正整数，返回它在 Excel 表中相对应的列名称。

例如，

```
    1 -> A
    2 -> B
    3 -> C
    ...
    26 -> Z
    27 -> AA
    28 -> AB
    ...
```

示例 1:

```
输入: 1
输出: "A"
```

示例 2:

```
输入: 28
输出: "AB"
```

示例 3:

```
输入: 701
输出: "ZY"
```

代码：

```
# -*- coding: utf-8 -*-

class Solution:
    def convertToTitle(self, n: int) -> str:

        dic = {}
        for i in range(0, 25):
            dic[i+1] = chr(ord('A')+i)
        dic[0] = 'Z'

        res = ''
        while n > 0:
            res += dic[n % 26]
            if n % 26 != 0:
                n = n // 26
```

```
            else:
                n = (n - 26) // 26

        return res[::-1]

s = Solution()
print(s.convertToTitle(701))
print(s.convertToTitle(52))
```

922. 按奇偶排序数组 II [数组]

给定一个非负整数数组 `A`， A 中一半整数是奇数，一半整数是偶数。

对数组进行排序，以便当 `A[i]` 为奇数时，`i` 也是奇数；当 `A[i]` 为偶数时， `i` 也是偶数。

你可以返回任何满足上述条件的数组作为答案。

示例：

```
输入：[4,2,5,7]
输出：[4,5,2,7]
解释：[4,7,2,5]，[2,5,4,7]，[2,7,4,5] 也会被接受。
```

提示： `2 <= A.length <= 20000` ；`A.length % 2 == 0`； `0 <= A[i] <= 1000`

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def sortArrayByParityII(self, A):
        Alen = len(A)
        res = [0] * Alen
        even, odd = 0, 1
        for e in A:
            if e % 2 == 0:
                res[even] = e
                even = even + 2
            else:
                res[odd] = e
                odd = odd + 2
        print(res)
        return res
```

```
s = Solution()
print(s.sortArrayByParityII([4,2,5,7]))
```

908. 最小差值 I [数学]

给你一个整数数组 `A`，对于每个整数 `A[i]`，我们可以选择处于区间 `[-K, K]` 中的任意数 `x`，将 `x` 与 `A[i]` 相加，结果存入 `A[i]`。

在此过程之后，我们得到一些数组 `B`。

返回 `B` 的最大值和 `B` 的最小值之间可能存在的最小差值。

示例 1：

```
输入：A = [1], K = 0
输出：0
解释：B = [1]
```

示例 2：

```
输入：A = [0,10], K = 2
输出：6
解释：B = [2,8]
```

示例 3：

```
输入：A = [1,3,6], K = 3
输出：0
解释：B = [3,3,3] 或 B = [4,4,4]
```

提示：

1. `1 <= A.length <= 10000` `0 <= A[i] <= 10000` `0 <= K <= 10000`

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def smallestRangeI(self, A, K: int) -> int:
#         数组中最小值和最大值相差大于两倍的K时，最小差值为最大值和最小值之差减两倍的K；
#         当最小值和最大值相差小于或等于两倍的K时，最小差值为0。
        return max(max(A) - min(A) - 2*K, 0)

s = Solution()
print(s.smallestRangeI(A = [1], K = 0))
print(s.smallestRangeI(A = [0,10], K = 2))
print(s.smallestRangeI(A = [1,3,6], K = 3))
```

476. 数字的补数 [数学]

给定一个正整数，输出它的补数。补数是对该数的二进制表示取反。

示例 1:

```
输入: 5
输出: 2
解释: 5 的二进制表示为 101（没有前导零位），其补数为 010。所以你需要输出 2 。
```

示例 2:

```
输入: 1
输出: 0
解释: 1 的二进制表示为 1（没有前导零位），其补数为 0。所以你需要输出 0 。
```

注意: 给定的整数保证在 32 位带符号整数的范围内。 你可以假定二进制数不包含前导零位。本题与 1009 https://leetcode-cn.com/problems/complement-of-base-10-integer/ 相同

```
# -*- coding: utf-8 -*-
class Solution:
    def findComplement(self, num: int) -> int:
        nbin = bin(num)

        comp = '0b'
        for e in nbin[2:]:
            if e == '0':
                comp += '1'
            else:
```

```
                comp += '0'
        return int(comp, 2)

s = Solution()
print(s.findComplement(5))
print(s.findComplement(1))
```

905. 按奇偶排序数组 [数组]

给定一个非负整数数组 `A`，返回一个数组，在该数组中， `A` 的所有偶数元素之后跟着所有奇数元素。

你可以返回满足此条件的任何数组作为答案。

示例：

```
输入：[3,1,2,4]
输出：[2,4,3,1]
输出 [4,2,3,1]，[2,4,1,3] 和 [4,2,1,3] 也会被接受。
```

提示： `1 <= A.length <= 5000` `0 <= A[i] <= 5000`

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def sortArrayByParity(self, A):
        i, j = 0, len(A)-1
        while i < j:
            while i < j and A[i] % 2 == 0:
                i = i + 1
            if i < j:
                t = A[i]
            while i < j and A[j] % 2 == 1:
                j = j - 1
            if i < j:
                A[i] = A[j]
                A[j] = t
        print(A)
        return A
```

```
s = Solution()
print(s.sortArrayByParity([3, 1, 2, 4]))
```

561. 数组拆分 I [数组]

给定长度为 **2n** 的数组, 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。

示例 1:

```
输入: [1,4,3,2]

输出: 4
解释: n 等于 2, 最大总和为 4 = min(1, 2) + min(3, 4).
```

提示:

1. **n** 是正整数,范围在 [1, 10000]. 2. 数组中的元素范围在 [-10000, 10000].

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def arrayPairSum(self, nums) -> int:
        nums.sort()
        print(nums)
        rlen = len(nums)
        nums = [min(nums[i], nums[i+1]) for i in range(0, rlen-1, 2)]
        return sum(nums)

s = Solution()
print(s.arrayPairSum([1, 4, 3, 2]))
```

面试题 01.01. 判定字符是否唯一 [数组]

实现一个算法，确定一个字符串 `s` 的所有字符是否全都不同。

示例 1：

```
输入: s = "leetcode"
输出: false

输入: s = "abc"
输出: true
```

限制： `0 <= len(s) <= 100` 2. 如果你不使用额外的数据结构，会很加分。

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def isUnique(self, astr: str):
        for e in astr:
            if astr.count(e) > 1:
                return False
        return True

s = Solution()
print(s.isUnique('leetcode'))
print(s.isUnique('abc'))
```

657. 机器人能否返回原点 [字符串]

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。
```

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def judgeCircle(self, moves: str) -> bool:
                return moves.count('D')==moves.count('U') and
moves.count('R')==moves.count

s = Solution()
print(s.judgeCircle('UD'))
print(s.judgeCircle('LL'))
print(s.judgeCircle('RRDD'))
```

832. 翻转图像 [数组]

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: [[1,1,0,0],[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]]
```

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

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def flipAndInvertImage(self, A):
```

```
        res = []
        for li in A:
            li.reverse()
            rlen = len(li)
            for i in range(rlen):
                if li[i] == 0:
                    li[i] = 1
                else:
                    li[i] = 0
            res.append(li)
        return res

s = Solution()
print(s.flipAndInvertImage([[1,1,0],[1,0,1],[0,0,0]]))
print(s.flipAndInvertImage([[1,1,0,0],[1,0,0,1],[0,1,1,1],[1,0,1,0]]))
```

804. 唯一摩尔斯密码词 [字符串]

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 "--...--.".
```

注意: 单词列表`words` 的长度不会超过 `100`。每个单词 `words[i]`的长度范围为 `[1, 12]`。每个单词 `words[i]`只包含小写字母。

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def uniqueMorseRepresentations(self, words) -> int:
        import string
        letters = string.ascii_lowercase
        table = [".-","-...","-.-.","-..",".","..-.","-
-.","....","..",".---","-.-",".-..","--","-.","---",".--.","--.-
","-.-.","...","-","..-","...-",".--","-..-","-.--","--.."]
        wlen = len(words)
        T = []
        for i in range(wlen):
            str2 = ''
            wl = len(words[i])
            for j in range(wl):
                k = letters.index(words[i][j])
                str2 += table[k]
            if str2 not in T:
                T.append(str2)
        print(T)
        return len(T)

s = Solution()
print(s.uniqueMorseRepresentations(words = ["gin", "zen", "gig", "msg"]))
```

461. 汉明距离 [数学]

两个整数之间的汉明距离指的是这两个数字对应二进制位不同的位置的数目。

给出两个整数 `x` 和 `y`，计算它们之间的汉明距离。

注意： 0 ≤ `x`, `y` < 231.

示例:

```
输入: x = 1, y = 4
输出: 2
解释:
1   (0 0 0 1)
4   (0 1 0 0)
       ↑   ↑
上面的箭头指出了对应二进制位不同的位置。
```

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def hammingDistance(self, x: int, y: int) -> int:
        xbin = bin(x)[2:]
        ybin = bin(y)[2:]
        print(xbin, ybin)
        bmax = len(xbin) if len(xbin) > len(ybin) else len(ybin)

        res = 0
        xbin = ('{0:0'+str(1+bmax)+'d}').format(int(xbin))
        ybin = ('{0:0'+str(1+bmax)+'d}').format(int(ybin))

        maxb = len(xbin)
        for i in range(maxb):
            if xbin[i] != ybin[i]:
                res += 1

        print(res)
        return res
        # return bin(x ^ y).count('1')
```

```
s = Solution()
print(s.hammingDistance(x=1, y=4))
```

1252. 奇数值单元格的数目 [数组]

给你一个 `n` 行 `m` 列的矩阵，最开始的时候，每个单元格中的值都是 `0`。

另有一个索引数组 `indices`，`indices[i] = [ri, ci]` 中的 `ri` 和 `ci` 分别表示指定的行和列（从 `0` 开始编号）。

你需要将每对 `[ri, ci]` 指定的行和列上的所有单元格的值加 `1`。

请你在执行完所有 `indices` 指定的增量操作后，返回矩阵中 「奇数值单元格」 的数目。

示例 1：

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 3 & 1 \\ 1 & 3 & 1 \end{bmatrix}$$

```
输入：n = 2, m = 3, indices = [[0,1],[1,1]]
输出：6
解释：最开始的矩阵是 [[0,0,0],[0,0,0]]。
第一次增量操作后得到 [[1,2,1],[0,1,0]]。
最后的矩阵是 [[1,3,1],[1,3,1]]，里面有 6 个奇数。
```

提示： `1 <= n <= 50`；`1 <= m <= 50`；`1 <= indices.length <= 100`；`0 <= indices[i][0] < n`；`0 <= indices[i][1] < m`

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def oddCells(self, n, m, indices) -> int:
        ans = [[0]*m for i in range(n)]

        for b in indices:
            row = b[0]
            col = b[1]
```

```
            cnt = len(ans[0])
            for i in range(cnt):
                ans[row][i] += 1

            cnt = len(ans)
            for i in range(cnt):
                ans[i][col] += 1

        ans = [e for ei in ans for e in ei if e % 2 == 1]
        return len(ans)

s = Solution()
print(s.oddCells(n = 2, m = 3, indices = [[0,1],[1,1]]))
print(s.oddCells(n = 2, m = 2, indices = [[1,1],[0,0]]))
print(s.oddCells(2, 2, [[1,1],[0,0]]))
```

1323. 6 和 9 组成的最大数字 [数学]

给你一个仅由数字 6 和 9 组成的正整数 `num`。你最多只能翻转一位数字，将 6 变成 9，或者把 9 变成 6 。

请返回你可以得到的最大数字。

示例 1：

```
输入：num = 9669
输出：9969
解释：
其中最大的数字是 9969 。
```

提示： `1 <= num <= 10^4` ； `num` 每一位上的数字都是 6 或者 9 。

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def maximum69Number (self, num: int) -> int:
        num = str(num)
        return num.replace('6', '9', 1)

s = Solution()
print(s.maximum69Number(9669))
```

面试题17. 打印从1到最大的n位数 [数学]

输入数字 `n`，按顺序打印出从 1 到最大的 n 位十进制数。比如输入 3，则打印出 1、2、3 一直到最大的 3 位数 999。

示例 1:

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

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def printNumbers(self, n: int) :
        res = list(range(1, 10**n))
        return res

s = Solution()
print(s.printNumbers(1))
print(s.printNumbers(2))
```

771. 宝石与石头 [哈希表]

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。

示例 1:

```
输入: J = "aA", S = "aAAbbbb"
输出: 3
```

注意: `S` 和 `J` 最多含有50个字母。 `J` 中的字符不重复。

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def numJewelsInStones(self, J: str, S: str) -> int:
        res = 0
        for s in S:
            if s in J:
                res += 1
        return res

s = Solution()
print(s.numJewelsInStones('aA', 'aAAbbbb'))
print(s.numJewelsInStones('z', 'ZZ'))
```

1046. 最后一块石头的重量 [堆 贪心算法]

有一堆石头，每块石头的重量都是正整数。

每一回合，从中选出两块 **最重的** 石头，然后将它们一起粉碎。假设石头的重量分别为 `x` 和 `y`，且 `x <= y`。那么粉碎的可能结果如下：

- 如果 `x == y`，那么两块石头都会被完全粉碎；
- 如果 `x != y`，那么重量为 `x` 的石头将会完全粉碎，而重量为 `y` 的石头新重量为 `y-x`。

最后，最多只会剩下一块石头。返回此石头的重量。如果没有石头剩下，就返回 `0`。

示例：

```
输入：[2,7,4,1,8,1]
输出：1
解释：
先选出 7 和 8，得到 1，所以数组转换为 [2,4,1,1,1]，
再选出 2 和 4，得到 2，所以数组转换为 [2,1,1,1]，
接着是 2 和 1，得到 1，所以数组转换为 [1,1,1]，
最后选出 1 和 1，得到 0，最终数组转换为 [1]，这就是最后剩下那块石头的重量。
```

提示： `1 <= stones.length <= 30`；`1 <= stones[i] <= 1000`

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def lastStoneWeight(self, stones) -> int:
```

```
        stones.sort(reverse=True)

        i = 0
        while len(stones) > 1:
            x = stones[i+1]
            y = stones[i]
            if x == y:
                stones.remove(x)
                stones.remove(y)
            if x != y:
                stones.remove(x)
                stones.remove(y)
                stones.append(y - x)
            stones.sort(reverse=True)

        if len(stones) > 0:
            return stones[0]
        else:
            return 0

s = Solution()
print(s.lastStoneWeight([2,7,4,1,8,1]))
print(s.lastStoneWeight([5,1,8,10,7]))
```

1287. 有序数组中出现次数超过25%的元素 [数组]

给你一个非递减的 **有序** 整数数组，已知这个数组中恰好有一个整数，它的出现次数超过数组元素总数的 25%。

请你找到并返回这个整数

示例：

```
输入：arr = [1,2,2,6,6,6,6,7,10]
输出：6
```

提示：

- `1 <= arr.length <= 10^4`
- `0 <= arr[i] <= 10^5`

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def findSpecialInteger(self, arr) -> int:
        alen = len(arr) // 4
        for e in arr:
            if arr.count(e) > alen:
                return e
s = Solution()
print(s.findSpecialInteger(arr = [1,2,2,6,6,6,6,7,10]))
```

941. 有效的山脉数组 [数组]

给定一个整数数组 `A`，如果它是有效的山脉数组就返回 `true`，否则返回 `false`。

让我们回顾一下，如果 A 满足下述条件，那么它是一个山脉数组：

- `A.length >= 3`；在 `0 < i < A.length - 1` 条件下，存在 `i`

  使得：`A[0] < A[1] < ... A[i-1] < A[i]`；`A[i] > A[i+1] > ... > A[A.length - 1]`

示例 1：

```
输入：[2,1]
输出：false
输入：[3,5,5]
输出：false
输入：[0,3,2,1]
输出：true
```

提示： `0 <= A.length <= 10000`；`0 <= A[i] <= 10000`

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def validMountainArray(self, A) -> bool:
        i = 0
        alen = len(A)
        while i < alen-1 and A[i] < A[i + 1]:
            i = i + 1
```

```
        if i == 0:
            return False
        if i == alen-1:
            return False
        while i < alen-1 and A[i] > A[i + 1]:
            i = i + 1
        if i == alen - 1:
            return True
        else:
            return False

s = Solution()
print(s.validMountainArray([2, 1]))
print(s.validMountainArray([3, 5, 5]))
print(s.validMountainArray([0, 3, 2, 1]))
```

1281. 整数的各位积和之差 [数学]

给你一个整数 `n`，请你帮忙计算并返回该整数「各位数字之积」与「各位数字之和」的差。

示例 1：

```
输入：n = 234
输出：15
解释：
各位数之积 = 2 * 3 * 4 = 24
各位数之和 = 2 + 3 + 4 = 9
结果 = 24 - 9 = 15
```

示例 2：

```
输入：n = 4421
输出：21
解释：
各位数之积 = 4 * 4 * 2 * 1 = 32
各位数之和 = 4 + 4 + 2 + 1 = 11
结果 = 32 - 11 = 21
```

提示：

- `1 <= n <= 10^5`

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def subtractProductAndSum(self, n: int) -> int:
        from functools import reduce
        nstr = str(n)
        nint = list(map(int, nstr))

        pro = reduce(lambda x, y: x*y, nint)
        su = reduce(lambda x, y: x+y, nint)
        return pro - su

s = Solution()
print(s.subtractProductAndSum(234))
print(s.subtractProductAndSum(4421))
print(s.subtractProductAndSum(114))
```

888. 公平的糖果交换 [数组]

爱丽丝和鲍勃有不同大小的糖果棒：`A[i]` 是爱丽丝拥有的第 `i` 块糖的大小，`B[j]` 是鲍勃拥有的第 `j` 块糖的大小。

因为他们是朋友，所以他们想交换一个糖果棒，这样交换后，他们都有相同的糖果总量。*（一个人拥有的糖果总量是他们拥有的糖果棒大小的总和。）*

返回一个整数数组 `ans`，其中 `ans[0]` 是爱丽丝必须交换的糖果棒的大小，`ans[1]` 是 Bob 必须交换的糖果棒的大小。

如果有多个答案，你可以返回其中任何一个。保证答案存在。

示例 1：

```
输入：A = [1,1], B = [2,2]
输出：[1,2]
```

示例 2：

```
输入：A = [1,2], B = [2,3]
输出：[1,2]
```

示例 3：

```
输入：A = [2], B = [1,3]
输出：[2,3]
```

示例 4：

```
输入：A = [1,2,5], B = [2,4]
输出：[5,4]
```

提示：

- `1 <= A.length <= 10000` ；`1 <= B.length <= 10000` ；`1 <= A[i] <= 100000`
- `1 <= B[i] <= 100000` ；保证爱丽丝与鲍勃的糖果总量不同。；答案肯定存在。

代码：

```
# -*- coding: utf-8 -*-
class Solution:
#    输入：A = [1,1], B = [2,2]
#    输出：[1,2]
    def fairCandySwap(self, A, B):
        sa, sb = sum(A), sum(B)
        diff = (sa - sb) / 2
        A.sort()
        B.sort()
        i, j = 0, 0

        while A[i] - B[j] != diff:
            if A[i] - B[j] > diff:   # A[i] - B[j] > diff: j++
                j = j + 1
            if A[i] - B[j] < diff:
                i = i + 1

        return [A[i], B[j]]

s = Solution()
print(s.fairCandySwap(A = [1,1], B = [2,2]))
print(s.fairCandySwap(A = [1,2], B = [2,3]))
print(s.fairCandySwap(A = [2], B = [1,3]))
print(s.fairCandySwap(A = [1,2,5], B = [2,4]))
```

409. 最长回文串 [字符串]

给定一个包含大写字母和小写字母的字符串，找到通过这些字母构造成的最长的回文串。

在构造过程中，请注意区分大小写。比如 `"Aa"` 不能当做一个回文字符串。

注意: 假设字符串的长度不会超过 1010。

示例 1:

```
输入:
"abccccdd"
输出:
7
解释:
我们可以构造的最长的回文串是"dccaccd", 它的长度是 7。
```

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def longestPalindrome(self, s: str) -> int:
        from collections import Counter
        counter = Counter(s)

        res, odd_flag = 0, 0   # 结果, 奇数次flag设置为0
        for k, v in counter.items():
            res += (v // 2) * 2  # 偶数则偶数次；奇数，则小于奇数的最大偶数
            if v % 2 == 1:       # 有奇数，则最后再+1
                odd_flag = 1

        return res + odd_flag

s = Solution()
print(s.longestPalindrome("abccccdd"))
print(s.longestPalindrome("ccc"))
print(s.longestPalindrome("bananas"))
```

860. 柠檬水找零 [数组]

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` 。

示例 1：

```
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。
```

示例 2：

```
输入：[5,5,10]
输出：true
```

示例 3：

```
输入：[10,10]
输出：false
```

示例 4：

```
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。
```

提示：

- `0 <= bills.length <= 10000`
- `bills[i]` 不是 `5` 就是 `10` 或是 `20`

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def lemonadeChange(self, bills) -> bool:
        five, ten = 0, 0
        for b in bills:
            if b == 5:
                five += 1
            elif b == 10:
                if not five:
                    return False
                five -= 1
                ten += 1
            else:
                if ten and five:
                    ten -= 1
                    five -= 1
                elif five > 2:
                    five -= 3
                else:
                    return False
        return True

s = Solution()

print(s.lemonadeChange([5,5,5,10,20]))
print(s.lemonadeChange([5,5,10,10,20]))
```

844. 比较含退格的字符串 [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。

示例 1：

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

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def backspaceCompare(self, S: str, T: str) -> bool:
        st1 = ''
        st2 = ''
        for e in S:
            if e != '#':
                st1 += e
            else:
                st1 = st1[:-1]
        for e in T:
            if e != '#':
                st2 += e
            else:
                st2 = st2[:-1]

        if st1 == st2:
            return True
        else:
            return False

s = Solution()
print(s.backspaceCompare(S = "ab##", T = "c#d#"))
```

1276. 不浪费原料的汉堡制作方案 【贪心】

圣诞活动预热开始啦，汉堡店推出了全新的汉堡套餐。为了避免浪费原料，请你帮他们制定合适的制作计划。

给你两个整数 `tomatoSlices` 和 `cheeseSlices`，分别表示番茄片和奶酪片的数目。不同汉堡的原料搭配如下：

- **巨无霸汉堡：**4 片番茄和 1 片奶酪
- **小皇堡：**2 片番茄和 1 片奶酪

请你以 `[total_jumbo, total_small]`（[巨无霸汉堡总数，小皇堡总数]）的格式返回恰当的制作方案，使得剩下的番茄片 `tomatoSlices` 和奶酪片 `cheeseSlices` 的数量都是 `0`。

如果无法使剩下的番茄片 `tomatoSlices` 和奶酪片 `cheeseSlices` 的数量为 `0`，就请返回 `[]`。

示例：

示例 1：
输入：tomatoSlices = 16, cheeseSlices = 7
输出：[1,6]
解释：制作 1 个巨无霸汉堡和 6 个小皇堡需要 4*1 + 2*6 = 16 片番茄和 1 + 6 = 7 片奶酪。不会剩下原料。

示例 2：
输入：tomatoSlices = 17, cheeseSlices = 4
输出：[]
解释：只制作小皇堡和巨无霸汉堡无法用光全部原料。

示例 3：
输入：tomatoSlices = 4, cheeseSlices = 17
输出：[]
解释：制作 1 个巨无霸汉堡会剩下 16 片奶酪，制作 2 个小皇堡会剩下 15 片奶酪。

示例 4：
输入：tomatoSlices = 0, cheeseSlices = 0
输出：[0,0]

示例 5：
输入：tomatoSlices = 2, cheeseSlices = 1
输出：[0,1]

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def numOfBurgers(self, tomatoSlices: int, cheeseSlices: int):
        low, high = 0, tomatoSlices // 4
#         巨无霸汉堡：4 片番茄和 1 片奶酪
#         小皇堡：2 片番茄和 1 片奶酪
        while low <= high:
            mid = (low + high) // 2
            # 巨无霸 的 数据比
            tomatos, cheeses = tomatoSlices - mid*4, cheeseSlices - mid

            # 符合 小皇煲 的 数据比
            if tomatos == cheeses*2:
                return [mid, cheeses]
            elif tomatos > cheeses*2:
```

```
                low = mid + 1
            else:
                high = mid - 1
        return []

s = Solution()
#print(s.numOfBurgers(tomatoSlices = 48, cheeseSlices = 16))
print(s.numOfBurgers(tomatoSlices = 16, cheeseSlices = 7))
#print(s.numOfBurgers(tomatoSlices = 17, cheeseSlices = 4))
```

263. 丑数 [数学]

编写一个程序判断给定的数是否为丑数。

丑数就是只包含质因数 `2, 3, 5` 的**正整数**。

示例 1:

```
输入: 6
输出: true
解释: 6 = 2 × 3
```

示例 2:

```
输入: 8
输出: true
解释: 8 = 2 × 2 × 2
```

示例 3:

```
输入: 14
输出: false
解释: 14 不是丑数，因为它包含了另外一个质因数 7。
```

说明：

1. `1` 是丑数。
2. 输入不会超过 32 位有符号整数的范围: [−231, 231 − 1]。

```
# -*- coding: utf-8 -*-
class Solution:
    def isUgly(self, num: int) -> bool:
```

```
        if num == 0:
            return False
        if num == 1:
            return True
        elif num % 2 == 0:
            return self.isUgly(num // 2)
        elif num % 3 == 0:
            return self.isUgly(num // 3)
        elif num % 5 == 0:
            return self.isUgly(num // 5)
        else:
            return False

s = Solution()
print(s.isUgly(6))
print(s.isUgly(8))
print(s.isUgly(14))
```

628. 三个数的最大乘积[数组]

给定一个整型数组，在数组中找出由三个数组成的最大乘积，并输出这个乘积。

示例 1:

```
输入: [1,2,3]
输出: 6
```

示例 2:

```
输入: [1,2,3,4]
输出: 24
```

注意:

1. 给定的整型数组长度范围是[3,104]，数组中所有的元素范围是[-1000, 1000]。
2. 输入的数组中任意三个数的乘积不会超出32位有符号整数的范围。

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def maximumProduct(self, nums) -> int:
        nums.sort()
        return max(nums[-1]*nums[-2]*nums[-3],nums[-1]*nums[0]*nums[1])

s = Solution()
print(s.maximumProduct([-4,-3,-2,-1,60]))
print(s.maximumProduct([-1,-2,1,2,3]))
```

824. 山羊拉丁文 [字符串]

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。

示例 1:

```
输入: "I speak Goat Latin"
输出: "Imaa peaksmaaa oatGmaaaa atinLmaaaaa"
```

示例 2:

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

说明:

- `S` 中仅包含大小写字母和空格。单词间有且仅有一个空格。
- `1 <= S.length <= 150`。

代码：

```
# -*- coding: utf-8 -*-

class Solution:
    def toGoatLatin(self, S: str) -> str:
        res = []
        S = S.split()
        cnt = 1
        for e in S:
            if e[0].lower() in ['a','e','i', 'o', 'u']:
                res.append(e + 'ma' + cnt*'a')
            else:
                res.append(e[1:] + e[0] + 'ma' + cnt*'a')
            cnt = cnt + 1
        return ' '.join(res)

s = Solution()
print(s.toGoatLatin("I speak Goat Latin"))
print(s.toGoatLatin("The quick brown fox jumped over the lazy dog"))
```

925. 长按键入 [字符串]

你的朋友正在使用键盘输入他的名字 `name`。偶尔，在键入字符 `c` 时，按键可能会被*长按*，而字符可能被输入 1 次或多次。

你将会检查键盘输入的字符 `typed`。如果它对应的可能是你的朋友的名字（其中一些字符可能被长按），那么就返回 `True`。

示例 1：

```
输入：name = "alex", typed = "aaleex"
输出：true
解释：'alex' 中的 'a' 和 'e' 被长按。
```

示例 2：

```
输入：name = "saeed", typed = "ssaaedd"
输出：false
解释：'e' 一定需要被键入两次，但在 typed 的输出中不是这样。
```

示例 3：

```
输入：name = "leelee", typed = "lleeelee"
输出：true
```

示例 4：

```
输入：name = "laiden", typed = "laiden"
输出：true
解释：长按名字中的字符并不是必要的。
```

提示：

1. `name.length <= 1000`
2. `typed.length <= 1000`
3. `name` 和 `typed` 的字符都是小写字母。

```
# -*- coding: utf-8 -*-
class Solution:
    def isLongPressedName(self, name: str, typed: str) -> bool:
         charSetN = set(name)
         charSetT = set(typed)
         for item in charSetN:
             if item not in charSetT:
                 return False
         for item in charSetT:
            if item not in charSetN:
                return False

         i, j = 0, 0
         while i < len(name) and j < len(typed):
             if name[i] == typed[j]:
                i += 1
                j += 1
             else:
                 if j > 0 and typed[j] == typed[j-1]:
                     j += 1
                 else:
                     return False
         if i == len(name):   # 所有name都用完了
             return True
         else:
             return False
```

```
s = Solution()
print(s.isLongPressedName(name = "alex", typed = "aaleex"))
print(s.isLongPressedName(name = "saeed", typed = "ssaaedd"))
print(s.isLongPressedName('abc', 'aaxbc'))
print(s.isLongPressedName("zlexya",
"aazlllllllllllllleexxxxxxxxxxxxxxxxya"))
```

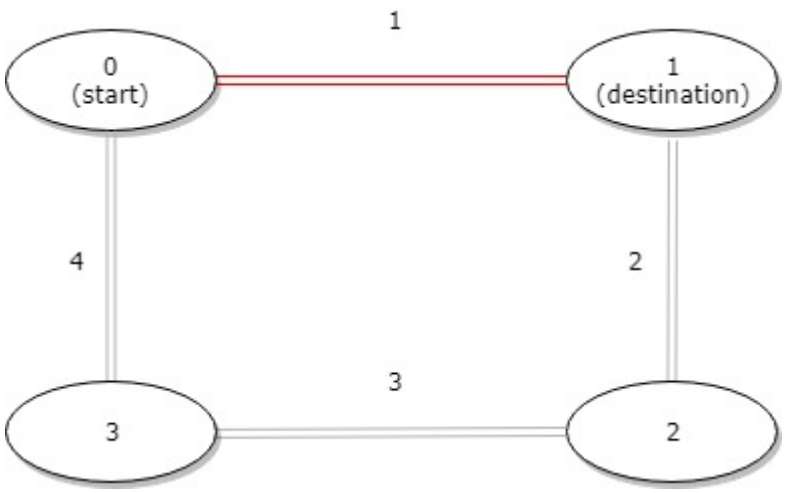
1184. 公交站间的距离 [数组]

环形公交路线上有 `n` 个站，按次序从 `0` 到 `n - 1` 进行编号。我们已知每一对相邻公交站之间的距离，`distance[i]` 表示编号为 `i` 的车站和编号为 `(i + 1) % n` 的车站之间的距离。

环线上的公交车都可以按顺时针和逆时针的方向行驶。

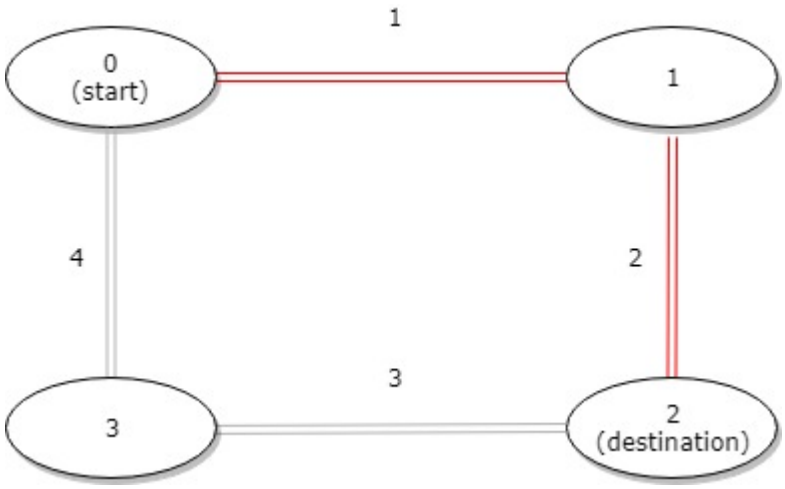
返回乘客从出发点 `start` 到目的地 `destination` 之间的最短距离。

示例 1：



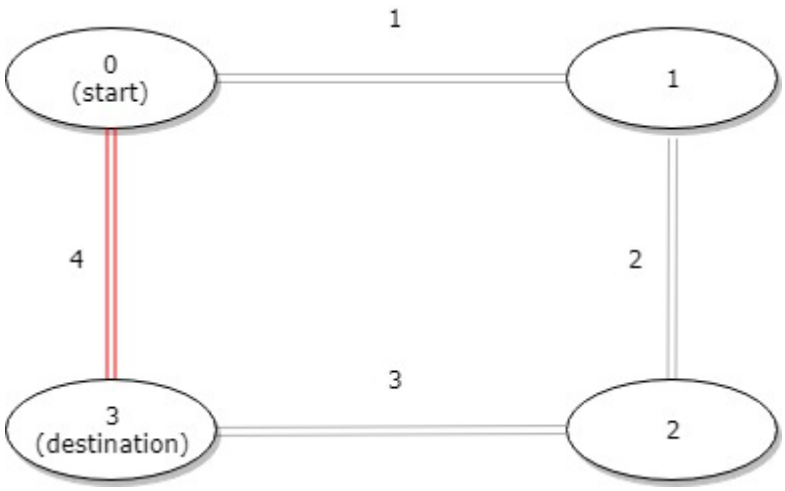

```
输入：distance = [1,2,3,4], start = 0, destination = 1
输出：1
解释：公交站 0 和 1 之间的距离是 1 或 9，最小值是 1。
```

示例 2：




```
输入：distance = [1,2,3,4], start = 0, destination = 2
输出：3
解释：公交站 0 和 2 之间的距离是 3 或 7，最小值是 3。
```

示例 3：




```
输入：distance = [1,2,3,4], start = 0, destination = 3
输出：4
解释：公交站 0 和 3 之间的距离是 6 或 4，最小值是 4。
```

提示：

- `1 <= n <= 10^4`
- `distance.length == n`
- `0 <= start, destination < n`
- `0 <= distance[i] <= 10^4`

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def distanceBetweenBusStops(self, distance, start: int, destination:
int) -> int:
        forward = 0
        backward = 0
        dlen = len(distance)
        for i in range(0, dlen):
            forward += distance[(start + i) % dlen]
            if (start + i + 1) % dlen == destination:
                break
        for i in range(0, dlen):
            backward += distance[(start + dlen - i - 1) % dlen]
            if (start + dlen - i - 1) % dlen == destination:
                break
        return forward if forward < backward else backward

s = Solution()
print(s.distanceBetweenBusStops(distance = [1,2,3,4], start = 0,
destination = 1))
print(s.distanceBetweenBusStops(distance = [1,2,3,4], start = 0,
destination = 3))
```

35. 二分算法

```
# -*- coding: utf-8 -*-
class Solution:
    def searchInsert(self, nums, target: int) -> int:
        nlen = len(nums)
        if nlen == 0:
            return 0
        if target < nums[0]:
            return 0
        if target > nums[nlen - 1]:
            return nlen
        low, high = 0, nlen - 1
        while low <= high:
            mid = (low + high) // 2
            if nums[mid] == target:
                return mid
            elif nums[mid] > target:
```

```
                high = mid - 1
            else:
                high = mid + 1
        return low

s = Solution()
#print(s.searchInsert([1,3,5,6], 5))
#print(s.searchInsert([1,3,5,6], 7))
print(s.searchInsert([1], 0))
```

290. 单词规律 [字符串]

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

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

示例1:

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

示例 2:

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

示例 3:

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

示例 4:

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

代码：

```
# -*- coding: utf-8 -*-

class Solution:
```

```
    def wordPattern(self, pattern: str, str1: str) -> bool:
        pli = list(pattern)
        sli = str1.split()
        plen = len(pli)
        slen = len(sli)
        if plen != slen:
            return False
        for i in range(slen):
            if pli.index(pli[i]) != sli.index(sli[i]):
                return False
        return True

s = Solution()
print(s.wordPattern("abba", "dog cat cat dog"))
print(s.wordPattern("aaa", "aa aa aa aa"))
```

914. 卡牌分组[数组]

给定一副牌，每张牌上都写着一个整数。

此时，你需要选定一个数字 `X`，使我们可以将整副牌按下述规则分成 1 组或更多组：

- 每组都有 `X` 张牌。
- 组内所有的牌上都写着相同的整数。

仅当你可选的 `X >= 2` 时返回 `true`。

示例 1：

```
输入：[1,2,3,4,4,3,2,1]
输出：true
解释：可行的分组是 [1,1]，[2,2]，[3,3]，[4,4]
```

示例 2：

```
输入：[1,1,1,2,2,2,3,3]
输出：false
解释：没有满足要求的分组。
```

示例 3：

输入：[1]
输出：false
解释：没有满足要求的分组。

示例 4：

输入：[1,1]
输出：true
解释：可行的分组是 [1,1]

示例 5：

输入：[1,1,2,2,2,2]
输出：true
解释：可行的分组是 [1,1]，[2,2]，[2,2]
提示：

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def hasGroupsSizeX(self, deck) -> bool:
        import math
        dlen = len(deck)
        if dlen < 2:
            return False

        cnt = [deck.count(e) for e in deck]

        mygcd = 9999999
        for i in range(dlen - 1):
            t = math.gcd(cnt[i], cnt[i + 1])
            if mygcd > t:
                mygcd = t

        if mygcd == 1:
            return False

        for i in range(dlen - 1):
            if cnt[i] % mygcd != 0:
                return False
```

```
        return True

s = Solution()
print(s.hasGroupsSizeX([1,2,3,4,4,3,2,1]))
print(s.hasGroupsSizeX([1,1,2,2,2,2]))
print(s.hasGroupsSizeX([1,1,1,1,2,2,2,2,2,2]))
```

645. 错误的集合[数组]

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元素复制了成了集合里面的另外一个元素的值，导致集合丢失了一个整数并且有一个元素重复。

给定一个数组 `nums` 代表了集合 `S` 发生错误后的结果。你的任务是首先寻找到重复出现的整数，再找到丢失的整数，将它们以数组的形式返回。

示例 1:

```
输入: nums = [1,2,2,4]
输出: [2,3]
```

注意:

1. 给定数组的长度范围是 [2, 10000]。
2. 给定的数组是无序的。

代码：

```
# -*- coding: utf-8 -*-
class Solution:
    def findErrorNums(self, nums):
        ans = []
        nlen = len(nums)

        counts = [0]*(nlen+1)
        for e in nums:
            counts[e] += 1

        print(counts)
        for i in range(1, nlen+1):
            if counts[i] == 0:
                if len(ans) < 1:
                    ans.append(i)
```

```
                else:
                    ans.append(i)
                    break
            elif counts[i] == 2:
                if len(ans) < 1:
                    ans.append(i)
                else:
                    ans.append(i)
                    break

        if counts[ans[0]] > 1:
            return ans
        else:
            return ans[::-1]

s = Solution()
print(s.findErrorNums([3,2,3,4,6,5]))
```

560. 和为K的子数组[数组 哈希表]

给定一个整数数组和一个整数 **k**，你需要找到该数组中和为 **k** 的连续的子数组的个数。

示例 1：

```
输入:nums = [1,1,1], k = 2
输出: 2 , [1,1] 与 [1,1] 为两种不同的情况。
```

说明：

1. 数组的长度为 [1, 20,000]。
2. 数组中元素的范围是 [-1000, 1000] ，且整数 **k** 的范围是 [-1e7, 1e7]。

代码：

```
# -*- coding: utf-8 -*-

class Solution:
    def subarraySum(self, nums, k: int) -> int:
        ans = 0
        sum = 0
        hash = {0 : 1}
        nlen = len(nums)
```

```
        for i in range(nlen):
            sum += nums[i]
            if ((sum - k) in hash):
                ans += hash[sum - k]     # +1
            if (sum in hash):            # sum == 0
                hash[sum] += 1           # 如果sum 能加出 hash中的key， 则对
应key的value+1
            else:
                hash[sum] = 1

        return ans

s = Solution()
print(s.subarraySum(nums = [1,1,1], k = 2))
print(s.subarraySum([28,54,7,-70,22,65,-6], 100))
```

46. 全排列

```
# -*- coding: utf-8 -*-
from typing import List
class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        self.ans = []
        self.nlen = len(nums)
        self.t = [0]*self.nlen

        self.visit = [0] * self.nlen
        def dfs(nums, cur):
            if cur == self.nlen:
                self.ans.append(self.t.copy())
                return

            for i in range(self.nlen):
                if not self.visit[i]:
                    self.t[cur] = nums[i]
                    self.visit[i] = True       # 回溯
                    dfs(nums, cur + 1)
                    self.visit[i] = False      # 回溯
        dfs(nums, 0)
        return self.ans
```

```
    # In [6]: list(permutations([1,2,3], 3))
    # Out[6]: [(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3,
2, 1)]
    def permute2(self, nums: List[int]) -> List[List[int]]:
        from itertools import permutations
        ans = [list(x) for x in permutations(nums, len(nums))]
        return ans

s = Solution()
print(s.permute([1, 2, 3]))
print(s.permute2([1, 2, 3]))
```

Mooc 05_4 集合操作(a < b: a是b的子集)

```
from typing import List

class Solution:
    def findWords(self, words: List[str]) -> List[str]:
        line1 = set('qwertyuiop')
        line2 = set('asdfghjkl')
        line3 = set('zxcvbnm')

        ans = []
        for word in words:
            wset = set(word.lower())
            if wset < line1 or wset < line2 or wset < line3:
                ans.append(word)
        return ans

def main():
    s = Solution()
    print(s.findWords(["Hello", "Alaska", "Dad", "Peace"]))

if __name__ == "__main__":
    main()
```