0.哈希表总结.md

- 记录Counter
    - 49 字母异位词分组
    - 128 最长连续序列
    - 202 快乐数
    - 217 存在重复元素
    - 242 有效的字母异位词
    - 349 两个数组的交集
    - 350 两个数组的交集 II
    - 383 赎金信
    - 387 字符串中的第一个唯一字符
    - 389 找不同
    - 438 找到字符串中所有字母异位词
    - 594 最长和谐子序列
    - 645 错误的集合
- 记录位置
    - 1 两数之和
    - 3 无重复字符的最长子串
    - 219 存在重复元素 II
    - 697 数组的度：其实是同事记录counter和位置
- 记录mapping
    - 13 罗马数字转整数
    - 205 同构字符串
    - 290 单词规律
- 类设计
    - 208 实现 Trie (前缀树)
    - 705 设计哈希集合
    - 706 设计哈希映射

1. 两数之和.md

给定一个整数数组 nums 和一个整数目标值 target，请你在该数组中找出 和为目标值 target 的那 两个 整数，并返回它们的数组下标。

你可以假设每种输入只会对应一个答案。但是，数组中同一个元素在答案里不能重复出现。

你可以按任意顺序返回答案。

示例 1：

输入：nums = [2,7,11,15], target = 9
输出：[0,1]
解释：因为 nums[0] + nums[1] == 9 ，返回 [0, 1] 。

示例 2：

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

示例 3：

输入：nums = [3,3], target = 6
输出：[0,1]

提示：

```
2 <= nums.length <= 104
-109 <= nums[i] <= 109
-109 <= target <= 109
只会存在一个有效答案
```

```python
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        dic = {}
        for i,j in enumerate(nums):
            if target-j in dic:
                return [dic[target-j], i]
            else:
                dic[j]=i
```

Tips 一边遍历一边写dict，解决有重复数字出现的问题

128. 最长连续序列.md

给定一个未排序的整数数组 nums ，找出数字连续的最长序列（不要求序列元素在原数组中连续）的长度。

请你设计并实现时间复杂度为 O(n) 的算法解决此问题。

示例 1：

输入：nums = [100,4,200,1,3,2]
输出：4
解释：最长数字连续序列是 [1, 2, 3, 4]。它的长度为 4。

示例 2：

输入：nums = [0,3,7,2,5,8,4,6,0,1]
输出：9

提示：

```
0 <= nums.length <= 105
-109 <= nums[i] <= 109
```

```python
class Solution:
    def longestConsecutive(self, nums: List[int]) -> int:
        nset = set(nums)
        curlen = 1
        maxlen = 0
        l = len(nums)
        for n in nums:
            if n-1 not in nset:
                for i in range(1,l):
                    if n+i in nset:
                        curlen+=1
                    else:
                        break

            maxlen = max(maxlen, curlen)
            curlen = 1
        return maxlen
```

Tips

1. 看到时间复杂度是O(n)肯定是要用额外空间来存储hash的
2. 不过这道题还有一个巧妙的点就是在判断n周围的相连数字是否存在于数组中时，只用向后遍历，因为如果n-1存在于数组中，从n开始遍历得到的长度一定不是最大的所以直接跳过就好，等到n-1这个数字的时候再遍历就好

13. 罗马数字转整数.md

罗马数字包含以下七种字符: I， V， X， L，C，D 和 M。

字符　　数值
I　　1
V　　5
X　　10
L　　50
C　　100
D　　500
M　　1000

例如， 罗马数字 2 写做 II ，即为两个并列的 1。12 写做 XII ，即为 X + II 。 27 写做 XXVII, 即为 XX + V + II 。

通常情况下，罗马数字中小的数字在大的数字的右边。但也存在特例，例如 4 不写做 IIII，而是 IV。数字 1 在数字 5 的左边，所表示的数等于大数 5 减小数 1 得到的数值 4 。同样地，数字 9 表示为 IX。这个特殊的规则只适用于以下六种情况：

```
I 可以放在 V (5) 和 X (10) 的左边，来表示 4 和 9。
X 可以放在 L (50) 和 C (100) 的左边，来表示 40 和 90。
C 可以放在 D (500) 和 M (1000) 的左边，来表示 400 和 900。
```

给定一个罗马数字，将其转换成整数。输入确保在 1 到 3999 的范围内。

示例 1:

输入: "III"
输出: 3

示例 2:

输入: "IV"
输出: 4

示例 3:

输入: "IX"
输出: 9

示例 4:

输入: "LVIII"
输出: 58
解释: L = 50, V= 5, III = 3.

示例 5:

输入: "MCMXCIV"
输出: 1994
解释: M = 1000, CM = 900, XC = 90, IV = 4.

提示：

```
1 <= s.length <= 15
s 仅含字符 ('I', 'V', 'X', 'L', 'C', 'D', 'M')
题目数据保证 s 是一个有效的罗马数字，且表示整数在范围 [1, 3999] 内
题目所给测试用例皆符合罗马数字书写规则，不会出现跨位等情况。
IL 和 IM 这样的例子并不符合题目要求，49 应该写作 XLIX，999 应该写作 CMXCIX 。
关于罗马数字的详尽书写规则，可以参考 罗马数字 - Mathematics 。
```

```
class Solution:
```

```python
    def romanToInt(self, s: str) -> int:
        roman = {'I':1,'V':5,'X':10,'L':50,'C':100,'D':500,'M':1000}
        n=len(s)
        num = 0
        i=0
        while i < n-1:
            if roman[s[i]] < roman[s[i+1]]:
                num+=roman[s[i+1]]-roman[s[i]]
                i+=2
            else:
                num+=roman[s[i]]
                i+=1
        if i ==n-1:
            num+=roman[s[i]]

        return num
```

137. 只出现一次的数字 II.md

给你一个整数数组 nums ，除某个元素仅出现 一次 外，其余每个元素都恰出现 三次 。请你找出并返回那个只出现了一次的元素。

示例 1：

输入：nums = [2,2,3,2]
输出：3

示例 2：

输入：nums = [0,1,0,1,0,1,99]
输出：99

提示：

```
1 <= nums.length <= 3 * 104
-231 <= nums[i] <= 231 - 1
nums 中，除某个元素仅出现 一次 外，其余每个元素都恰出现 三次
```

进阶：你的算法应该具有线性时间复杂度。 你可以不使用额外空间来实现吗?

```python
class Solution:
    def singleNumber(self, nums: List[int]) -> int:
        counter = collections.Counter(nums)
        for i,j in counter.items():
            if j==1:
                return i
```

常规hash解法

202. 快乐数.md

编写一个算法来判断一个数 n 是不是快乐数。

「快乐数」定义为：

```
对于一个正整数，每一次将该数替换为它每个位置上的数字的平方和。
然后重复这个过程直到这个数变为 1，也可能是 无限循环 但始终变不到 1。
如果 可以变为  1，那么这个数就是快乐数。
```

如果 n 是快乐数就返回 true ； 不是，则返回 false 。

示例 1：

输入：19
输出：true
解释：
12 + 92 = 82
82 + 22 = 68
62 + 82 = 100
12 + 02 + 02 = 1

示例 2：

输入：n = 2
输出：false

提示：

```
1 <= n <= 231 - 1
```

1. Hash解法: O(n)的内存和时间占用

```
class Solution:
    def isHappy(self, n: int) -> bool:
        seen = set()

        def get_next(n):
            total =0
            while n:
                total += (n%10) **2
                n //=10
            return total

        while (n!=1) and (n not in seen):
            seen.add(n)
            n = get_next(n)

        return n==1
```

2. 链表解法：get-next其实是在构建隐式链表，于是是否进入循环就变成了判断链表是否有环，可以用快慢指针解法

O（logn）的时间占用，O（1）的空间

```
class Solution:
    def isHappy(self, n: int) -> bool:
        seen = set()

        def get_next(n):
            total =0
            while n:
                total += (n%10) **2
                n //=10
            return total

        slow=n
        fast=get_next(n)
        while (fast!=1) and slow!=fast:
            slow = get_next(slow)
            fast = get_next(get_next(fast))

        return fast==1
```

3. 数学解法：能进入循环的数是有限的找到它们，然后判断是否碰到这些数就行

205. 同构字符串.md

给定两个字符串 s 和 t，判断它们是否是同构的。

如果 s 中的字符可以按某种映射关系替换得到 t ，那么这两个字符串是同构的。

每个出现的字符都应当映射到另一个字符，同时不改变字符的顺序。不同字符不能映射到同一个字符上，相同字符只能映射到同一个字符上，字符可以映射到自己本身。

示例 1:

输入：s = "egg", t = "add"
输出：true

示例 2:

输入：s = "foo", t = "bar"
输出：false

示例 3:

输入：s = "paper", t = "title"
输出：true

提示:

> 可以假设 s 和 t 长度相同。

```python
class Solution:
    def isIsomorphic(self, s: str, t: str) -> bool:
        dic = {}
        for i,j in zip(s,t):
            if i in dic:
                if dic[i]!=j:
                    return False
            else:
                dic[i]=j
        dic = {}
        for j,i in zip(s,t):
            if i in dic:
                if dic[i]!=j:
                    return False
            else:
                dic[i]=j
        return True
```

Tips

1. 同构其实就是A->B的mapping在任意位置都相同。可以用1个dict遍历两边，也可以用两个dict在一次遍历的时候同时判断A->B ,B->A

208. 实现 Trie (前缀树).md

Trie（发音类似 "try"）或者说 前缀树 是一种树形数据结构，用于高效地存储和检索字符串数据集中的键。这一数据结构有相当多的应用情景，例如自动补完和拼写检查。

请你实现 Trie 类：

```
Trie() 初始化前缀树对象。
void insert(String word) 向前缀树中插入字符串 word 。
boolean search(String word) 如果字符串 word 在前缀树中，返回 true（即，在检索之前已经插入）；否则，返回 false 。
boolean startsWith(String prefix) 如果之前已经插入的字符串 word 的前缀之一为 prefix ，返回 true ；否则，返回 false 。
```

示例：

输入
["Trie", "insert", "search", "search", "startsWith", "insert", "search"]
[[], ["apple"], ["apple"], ["app"], ["app"], ["app"], ["app"]]
输出
[null, null, true, false, true, null, true]

解释
Trie trie = new Trie();
trie.insert("apple");
trie.search("apple");   // 返回 True
trie.search("app");   // 返回 False
trie.startsWith("app"); // 返回 True
trie.insert("app");
trie.search("app");   // 返回 True

提示：

```
1 <= word.length, prefix.length <= 2000
word 和 prefix 仅由小写英文字母组成
insert、search 和 startsWith 调用次数 总计 不超过 3 * 104 次
```

```python
class TreeNode():
    def __init__(self, char):
        self.char = char
        self.is_end=False
        self.children = {}

    def insert(self, char):
        self.children[char] = self.children.get(char, TreeNode(char))
        return self.children[char]

    def search(self, char):
        return self.children.get(char, None)


class Trie:
    def __init__(self):
        self._root = TreeNode(None)

    def insert(self, word: str) -> None:
        node = self._root
        for c in word:
            node = node.insert(c)
        node.is_end=True

    def search(self, word: str) -> bool:
        node = self._root
        for c in word:
            node = node.search(c)
            if not node:
                return False
        if node and node.is_end:
            return True
        else:
            return False

    def startsWith(self, prefix: str) -> bool:
        node = self._root
        for c in prefix:
            node = node.search(c)
            if not node:
                return False
        return True
```

Tips

1. 实现很直观就是字典套字典套字典，更偏爱把Node和Trie分开写的方法，这样在需要节点里面保存更多信息的时候可以方便修改node，以及粒度的一致性更好Trie对字符串进行搜索/插入，Node对单一字符进行搜索

插入

217. 存在重复元素.md

给定一个整数数组，判断是否存在重复元素。

如果存在一值在数组中出现至少两次，函数返回 true 。如果数组中每个元素都不相同，则返回 false 。

示例 1:

输入: [1,2,3,1]
输出: true

示例 2:

输入: [1,2,3,4]
输出: false

示例 3:

输入: [1,1,1,3,3,4,3,2,4,2]
输出: true

```python
class Solution:
    def containsDuplicate(self, nums: List[int]) -> bool:
        if len(nums) > len(set(nums)):
            return True
        else:
            return False
```

Tips

1. 这题有好多种解法，用set，用sort，用hash都可以

219. 存在重复元素 II.md

给定一个整数数组和一个整数 k，判断数组中是否存在两个不同的索引 i 和 j，使得 nums [i] = nums [j]，并且 i 和 j 的差的 绝对值 至多为 k。

示例 1:

输入: nums = [1,2,3,1], k = 3
输出: true

示例 2:

输入: nums = [1,0,1,1], k = 1
输出: true

示例 3:

输入: nums = [1,2,3,1,2,3], k = 2
输出: false

```python
class Solution:
    def containsNearbyDuplicate(self, nums: List[int], k: int) -> bool:
        dic = {}
        for i,n in enumerate(nums):
            if (n in dic) and ((i-dic[n])<=k):
                return True
            else:
                dic[n] = i
        return False
```

Tips

1. 和两数之和一样，向前遍历的同时保存已经遍历完的数值结果

242. 有效的字母异位词.md

给定两个字符串 s 和 t ，编写一个函数来判断 t 是否是 s 的字母异位词。

注意：若 s 和 t 中每个字符出现的次数都相同，则称 s 和 t 互为字母异位词。

示例 1:

输入: s = "anagram", t = "nagaram"
输出: true

示例 2:

输入: s = "rat", t = "car"
输出: false

提示:

```
1 <= s.length, t.length <= 5 * 104
s 和 t 仅包含小写字母
```

```python
from collections import defaultdict
class Solution:
    def isAnagram(self, s: str, t: str) -> bool:

        res = defaultdict(int)
        n= 0
        for i in s:
            res[i]+=1
            n+=1
        for i in t:
            res[i]-=1
            if res[i]<0:
                return False
            n-=1
        if n!=0:
            return False
        else:
            return True
```

290. 单词规律.md

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

说明:
你可以假设 pattern 只包含小写字母， str 包含了由单个空格分隔的小写字母。

来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/word-pattern

```python
class Solution:
    def wordPattern(self, pattern: str, s: str) -> bool:
        words= s.split(' ')
        if len(pattern)!=len(words):
            return False
        ch2word = {}
        word2ch = {}

        for i, j in zip(pattern, words):
            if (i in ch2word) and (ch2word[i]!=j):
                return False
            elif (j in word2ch) and (word2ch[j]!=i):
                return False
            else:
                word2ch[j]=i
                ch2word[i]=j
        return True
```

Tips

1. 和第205题同构字符串是一摸一样滴，可以选择一个map遍历两遍检查是否存在冲突。也可以两个map遍历一遍

3. 无重复字符的最长子串.md

给定一个字符串 s ，请你找出其中不含有重复字符的 最长子串 的长度。

示例 1:

输入: s = "abcabcbb"
输出: 3
解释: 因为无重复字符的最长子串是 "abc"，所以其长度为 3。

示例 2:

输入: s = "bbbbb"
输出: 1
解释: 因为无重复字符的最长子串是 "b"，所以其长度为 1。

示例 3:

输入: s = "pwwkew"
输出: 3
解释: 因为无重复字符的最长子串是 "wke"，所以其长度为 3。
请注意，你的答案必须是 子串 的长度，"pwke" 是一个子序列，不是子串。

示例 4:

输入: s = ""
输出: 0

提示：

```
0 <= s.length <= 5 * 104
s 由英文字母、数字、符号和空格组成
```

```python
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        dic = {}
        length = 0
        index = -1
        for i,j in enumerate(s):
            pre_index = dic.get(j,-1)
            if pre_index > index:
                #当存在曾经出现的字符后才更新index，重新计算
                index = pre_index
            dic[j] = i
            length = max(length, i-index)
        return length
```

Tips:

1. 只要没有出现重复字符就持续向前迭代，只有当出现重复字符后更新index重新计数
2. 从-1开始，因为如果出现重复应该是从i+1开始遍历，但是第一个字母一定不重复所以不应该从0开始应该从-1开始

349. 两个数组的交集.md

给定两个数组，编写一个函数来计算它们的交集。

示例 1：

输入：nums1 = [1,2,2,1], nums2 = [2,2]
输出：[2]

示例 2：

输入：nums1 = [4,9,5], nums2 = [9,4,9,8,4]
输出：[9,4]

说明：

> 输出结果中的每个元素一定是唯一的。
> 我们可以不考虑输出结果的顺序。

```
class Solution:
    def intersection(self, nums1: List[int], nums2: List[int]) -> List[int]:
        return list(set(nums1).intersection(set(nums2)))
```

不考虑内存，不考虑两个数组的长度，最简单的方案就是用set自带的功能

350. 两个数组的交集 II.md

给定两个数组，编写一个函数来计算它们的交集。

示例 1：

输入：nums1 = [1,2,2,1], nums2 = [2,2]
输出：[2,2]

示例 2:

输入：nums1 = [4,9,5], nums2 = [9,4,9,8,4]
输出：[4,9]

说明：

> 输出结果中每个元素出现的次数，应与元素在两个数组中出现次数的最小值一致。
> 我们可以不考虑输出结果的顺序。

进阶：

> 如果给定的数组已经排好序呢？你将如何优化你的算法？
> 如果 `nums1` 的大小比 `nums2` 小很多，哪种方法更优？
> 如果 `nums2` 的元素存储在磁盘上，内存是有限的，并且你不能一次加载所有的元素到内存中，你该怎么办？

```
class Solution:
    def intersect(self, nums1: List[int], nums2: List[int]) -> List[int]:
        n1 = len(nums1)
        n2 = len(nums2)
```

```
        def helper(small_n, big_n):
            from collections import defaultdict
            dic_n = defaultdict(int)
            res = []
            for i in small_n:
                dic_n[i]+=1
            for i in big_n:
                if dic_n.get(i,0)>0:
                    dic_n[i]-=1
                    res.append(i)
            return res

        if n1<n2:
            return helper(nums1, nums2)
        else:
            return helper(nums2, nums1)
```

Tips

1. 空间优化，用小的list构建dict，类似于sql中用小表broadcast join 大表

383. 赎金信.md

给定一个赎金信 (ransom) 字符串和一个杂志(magazine)字符串，判断第一个字符串 ransom 能不能由第二个字符串 magazines 里面的字符构成。如果可以构成，返回 true ；否则返回 false。

(题目说明：为了不暴露赎金信字迹，要从杂志上搜索各个需要的字母，组成单词来表达意思。杂志字符串中的每个字符只能在赎金信字符串中使用一次。)

示例 1:

输入：ransomNote = "a", magazine = "b"
输出：false

示例 2:

输入：ransomNote = "aa", magazine = "ab"
输出：false

示例 3:

输入：ransomNote = "aa", magazine = "aab"
输出：true

提示:

你可以假设两个字符串均只含有小写字母。

```
class Solution:
    def canConstruct(self, ransomNote: str, magazine: str) -> bool:
        from collections import Counter
        dic = Counter(magazine)
        for i in ransomNote:
            if dic.get(i,-1)>0:
                dic[i]-=1
            else:
                return False
        return True
```

Tips

Counter 大法构建hash

387. 字符串中的第一个唯一字符.md

给定一个字符串，找到它的第一个不重复的字符，并返回它的索引。如果不存在，则返回 -1。

示例：

s = "leetcode"
返回 0

s = "loveleetcode"
返回 2

提示：你可以假定该字符串只包含小写字母

```python
class Solution:
    def firstUniqChar(self, s: str) -> int:
        from collections import Counter
        dic = Counter(s)
        index = 2**32-1
        for key, val in dic.items():
            if val ==1:
                index = min(index, s.find(key))
        if index== 2**32-1:
            return -1
        else:
            return index
```

Tips

看到次数闭上眼睛用Counter就对了

389. 找不同.md

给定两个字符串 s 和 t，它们只包含小写字母。

字符串 t 由字符串 s 随机重排，然后在随机位置添加一个字母。

请找出在 t 中被添加的字母。

示例 1：

输入：s = "abcd", t = "abcde"
输出："e"
解释：'e' 是那个被添加的字母。

示例 2：

输入：s = "", t = "y"
输出："y"

示例 3：

输入：s = "a", t = "aa"
输出："a"

示例 4：

输入：s = "ae", t = "aea"
输出："a"

提示：

```
0 <= s.length <= 1000
t.length == s.length + 1
s 和 t 只包含小写字母
```

1. 常规解法依旧使用Counter构建Hash，和赎金信一致

```python
class Solution:
    def findTheDifference(self, s: str, t: str) -> str:
        dic = Counter(s)
        for i in t:
            if dic.get(i,-1)>0:
                dic[i]-=1
            else:
                return i
```

2. 巧妙但不通用解法，因为只多一个str可以直接用ASCII求和找到多出来的一个

```python
class Solution:
    def findTheDifference(self, s: str, t: str) -> str:
        total = 0
        for i in t:
            total += ord(i)
        for j in s:
            total -=ord(j)
        return chr(total)
```

438. 找到字符串中所有字母异位词.md

给定两个字符串 s 和 p，找到 s 中所有 p 的 异位词 的子串，返回这些子串的起始索引。不考虑答案输出的顺序。

异位词 指由相同字母重排列形成的字符串（包括相同的字符串）。

示例 1:

输入: s = "cbaebabacd", p = "abc"
输出: [0,6]
解释:
起始索引等于 0 的子串是 "cba", 它是 "abc" 的异位词。
起始索引等于 6 的子串是 "bac", 它是 "abc" 的异位词。

示例 2:

输入: s = "abab", p = "ab"
输出: [0,1,2]
解释:
起始索引等于 0 的子串是 "ab", 它是 "ab" 的异位词。
起始索引等于 1 的子串是 "ba", 它是 "ab" 的异位词。
起始索引等于 2 的子串是 "ab", 它是 "ab" 的异位词。

提示:

```
1 <= s.length, p.length <= 3 * 104
s 和 p 仅包含小写字母
```

```python
class Solution:
    def findAnagrams(self, s: str, p: str) -> List[int]:
        n, m, res = len(s), len(p), []
        if n < m: return res
        p_cnt = [0] * 26
        s_cnt = [0] * 26
        for i in range(m):
            p_cnt[ord(p[i]) - ord('a')] += 1
            s_cnt[ord(s[i]) - ord('a')] += 1
        if s_cnt == p_cnt:
            res.append(0)

        for i in range(m, n):
            s_cnt[ord(s[i - m]) - ord('a')] -= 1
            s_cnt[ord(s[i]) - ord('a')] += 1
            if s_cnt == p_cnt:
                res.append(i - m + 1)
        return res
```

Tips

用了滑动窗口记录在s中每个len(p)的窗口s的Counter，每次向前移动一位就进行一次更新

49. 字母异位词分组.md

给你一个字符串数组，请你将 字母异位词 组合在一起。可以按任意顺序返回结果列表。

字母异位词 是由重新排列源单词的字母得到的一个新单词，所有源单词中的字母都恰好只用一次。

示例 1:

输入: strs = ["eat", "tea", "tan", "ate", "nat", "bat"]
输出: [["bat"],["nat","tan"],["ate","eat","tea"]]

示例 2:

输入: strs = [""]
输出: [[""]]

示例 3:

输入: strs = ["a"]
输出: [["a"]]

提示：

```
1 <= strs.length <= 104
0 <= strs[i].length <= 100
strs[i] 仅包含小写字母
```

```python
class Solution:
    def groupAnagrams(self, strs: List[str]) -> List[List[str]]:
        result = defaultdict(list)
        for s in strs:
            target = ''.join(sorted(s))
            result[target].append(s)
        return list(result.values())
```

Tips

用sort之后的string作为index即可。提醒自己一下，每次字符串都想用split()/split('') 来分割，但其实string本身就是iterable可以直接sorted

594. 最长和谐子序列.md

和谐数组是指一个数组里元素的最大值和最小值之间的差别 正好是 1 。

现在，给你一个整数数组 nums ，请你在所有可能的子序列中找到最长的和谐子序列的长度。

数组的子序列是一个由数组派生出来的序列，它可以通过删除一些元素或不删除元素、且不改变其余元素的顺序而得到。

示例 1：

输入：nums = [1,3,2,2,5,2,3,7]
输出：5
解释：最长的和谐子序列是 [3,2,2,2,3]

示例 2：

输入：nums = [1,2,3,4]
输出：2

示例 3：

输入：nums = [1,1,1,1]
输出：0

提示：

```
1 <= nums.length <= 2 * 104
-109 <= nums[i] <= 109
```

```python
class Solution:
    def findLHS(self, nums: List[int]) -> int:
        from collections import defaultdict
        dic = defaultdict(int)
        for i in nums:
            dic[i]+=1
        ans = 0

        for i in dic:
            if i+1 in dic:
                ans = max(ans, dic[i]+dic[i+1])
        return ans
```

Tips

这是一道伪装成array的hash问题，其实是求相邻元素个数之和的max。注意这里的子序列并不一定是连续的，所以才可以转换成Hash问题来解

645. 错误的集合.md

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。

示例 1：

输入：nums = [1,2,2,4]
输出：[2,3]

示例 2：

输入：nums = [1,1]
输出：[1,2]

提示：

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

1. 字典法用空间换时间，时间&空间复杂度都是O(n)

```python
class Solution:
    def findErrorNums(self, nums: List[int]) -> List[int]:
        dic = set()
        total = 0
        for i in nums:
            if i in dic:
                ans = [i]
            else:
                dic.add(i)
                total+=i
        n = len(nums)

        ans.append(int((1+n) *n/2)-total)
        return ans
```

697. 数组的度.md

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。

示例 1：

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.

示例 2：

输入：[1,2,2,3,1,4,2]
输出：6

提示：

```
nums.length 在1到 50,000 区间范围内。
nums[i] 是一个在 0 到 49,999 范围内的整数。
```

```python
class Solution:
    def findShortestSubArray(self, nums: List[int]) -> int:
        dic = {}
        for i,n in enumerate(nums):
            if n in dic:
                dic[n][0]+=1
                dic[n][2]=i
            else:
                dic[n]=[1,i,i]

        max_count = 0
        min_len = 0
        for val in dic.values():
            if val[0] > max_count:
                max_count = val[0]
                min_len = val[2]-val[1]+1
            elif val[0]==max_count:
                min_len = min(min_len, val[2]-val[1]+1)
        return min_len
```

Tips

dic分别保存counter，start_pos, end_pos，然后遍历dic.values, 不断更新最大counter对应的最小len

705. 设计哈希集合.md

不使用任何内建的哈希表库设计一个哈希集合（HashSet）。

实现 MyHashSet 类：

```
void add(key) 向哈希集合中插入值 key 。
bool contains(key) 返回哈希集合中是否存在这个值 key 。
void remove(key) 将给定值 key 从哈希集合中删除。如果哈希集合中没有这个值，什么也不做。
```

示例：

输入：
["MyHashSet", "add", "add", "contains", "contains", "add", "contains", "remove", "contains"]
[[], [1], [2], [1], [3], [2], [2], [2], [2]]
输出：
[null, null, null, true, false, null, true, null, false]

解释：
MyHashSet myHashSet = new MyHashSet();
myHashSet.add(1);   // set = [1]
myHashSet.add(2);   // set = [1, 2]
myHashSet.contains(1); // 返回 True
myHashSet.contains(3); // 返回 False ，（未找到）
myHashSet.add(2);   // set = [1, 2]
myHashSet.contains(2); // 返回 True
myHashSet.remove(2);   // set = [1]
myHashSet.contains(2); // 返回 False ，（已移除）

提示：

```
0 <= key <= 106
最多调用 104 次 add、remove 和 contains 。
```

1. 复杂度时间O（1），空间O（范围）

python list的index操作的是O（1）的复杂度，因为是直接找的对应地址的值

这里是用空间换时间

```python
class MyHashSet:

    def __init__(self):
```

```
        self.set= [False] * (10**6+1)


    def add(self, key: int) -> None:
        self.set[key] = True


    def remove(self, key: int) -> None:
        self.set[key]=False


    def contains(self, key: int) -> bool:
        return self.set[key]




# Your MyHashSet object will be instantiated and called as such:
# obj = MyHashSet()
# obj.add(key)
# obj.remove(key)
# param_3 = obj.contains(key)
```

2. 拉链法不定长数组：用时间换空间，时间复杂度O（N/bucket），空间复杂度O（数据范围s）

- 对key进行hash，得到位置
- 这时不同的key可能存在冲突，所以每个位置并不是一个值，而是一个链表/数组，用于存储该位置的所有元素
- 节省内存的点在于初始化时只初始化所有位置，每个位置只有当对应元素加入才创建

```
lass MyHashSet:

    def __init__(self):
        self.bucket =1000
        self.set= [[] for _ in range(self.bucket)]

    def hash(self, key):
        return key % self.bucket

    def add(self, key):
        index = self.hash(key)
        if key in self.set[index]:
            return
        self.set[index].append(key)

    def remove(self, key: int) -> None:
        index = self.hash(key)
```

```
        try:
            self.set[index].pop(self.set[index].index(key))
        except:
            return

    def contains(self, key: int) -> bool:
        index = self.hash(key)
        if key in self.set[index]:
            return True
        else:
            return False
```

706. 设计哈希映射.md

不使用任何内建的哈希表库设计一个哈希映射（HashMap）。

实现 MyHashMap 类：

```
MyHashMap() 用空映射初始化对象
void put(int key, int value) 向 HashMap 插入一个键值对 (key, value) 。如果 key 已经存在于映射中，则更新其对应的值 value 。
int get(int key) 返回特定的 key 所映射的 value ；如果映射中不包含 key 的映射，返回 -1 。
void remove(key) 如果映射中存在 key 的映射，则移除 key 和它所对应的 value 。
```

示例：

输入：
["MyHashMap", "put", "put", "get", "get", "put", "get", "remove", "get"]
[[], [1, 1], [2, 2], [1], [3], [2, 1], [2], [2], [2]]
输出：
[null, null, null, 1, -1, null, 1, null, -1]

解释：
MyHashMap myHashMap = new MyHashMap();
myHashMap.put(1, 1); // myHashMap 现在为 [[1,1]]
myHashMap.put(2, 2); // myHashMap 现在为 [[1,1], [2,2]]
myHashMap.get(1); // 返回 1 ，myHashMap 现在为 [[1,1], [2,2]]
myHashMap.get(3); // 返回 -1（未找到），myHashMap 现在为 [[1,1], [2,2]]
myHashMap.put(2, 1); // myHashMap 现在为 [[1,1], [2,1]]（更新已有的值）
myHashMap.get(2); // 返回 1 ，myHashMap 现在为 [[1,1], [2,1]]
myHashMap.remove(2); // 删除键为 2 的数据，myHashMap 现在为 [[1,1]]
myHashMap.get(2); // 返回 -1（未找到），myHashMap 现在为 [[1,1]]

提示：

```
0 <= key, value <= 106
最多调用 104 次 put、get 和 remove 方法
```

1. 和一题相同只不过把存储的bool值改成了default的-1

```
class MyHashMap:

    def __init__(self):
        self.set = [-1] *  (10**6+1)

    def put(self, key: int, value: int) -> None:
        self.set[key] = value


    def get(self, key: int) -> int:
        return self.set[key]


    def remove(self, key: int) -> None:
        self.set[key] = -1



# Your MyHashMap object will be instantiated and called as such:
# obj = MyHashMap()
# obj.put(key,value)
# param_2 = obj.get(key)
# obj.remove(keya
```

2. 不定长数组存储key, val

```
class MyHashMap:

    def __init__(self):
        self.bucket = 1000
        self.set = [[] for i in range(self.bucket)]

    def hash(self, key):
        return key %self.bucket

    def put(self, key: int, value: int) -> None:
        index = self.hash(key)
        for i in self.set[index]:
            if i[0]==key:
```

```
                i[1] = value
                return
        self.set[index].append([key, value])


    def get(self, key: int) -> int:
        index = self.hash(key)
        for i in self.set[index]:
            if i[0]==key:
                return i[1]
        return -1

    def remove(self, key: int) -> None:
        index = self.hash(key)
        for pos, i in enumerate(self.set[index]):
            if i[0]==key:
                self.set[index].pop(pos)
```