简单2_50

1-10

345、反转字符串中的元音字母

编写一个函数，以字符串作为输入，反转该字符串中的元音字母。

> 输入："hello"
> 输出："holle"

> 输入："leetcode"
> 输出："leotcede"

题解

1、双指针

利用字典查询速度快，左右双指针遍历，

由于要交换元音，需要把str转换成list处理，最后再join转回str。

```
class Solution:
    def reverseVowels(self, s: str) -> str:
        dic = {'a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U'}
        lst = list(s)
        n = len(s)
        l, r = 0, n - 1

        while l < r:
            if lst[l] in dic and lst[r] in dic:
                lst[l], lst[r] = lst[r], lst[l]
                l = l + 1
                r = r - 1
            elif lst[l] not in dic:
                l = l + 1
            elif lst[r] not in dic:
                r = r - 1

        return ''.join(lst)
```

2、pop方法

- ls存储单词中的所有元音字母，
- 遍历当前字符串，
    - 若为辅音字母直接添加，
    - 若为元音字母，pop方法取出最后一位元音字母，添加至目标列表

```python
class Solution:
    def reverseVowels(self, s: str) -> str:
        x={'a','e','i','o','u','A','E','I','O','U'}
        res=[]
        ls=[i for i in s if i in x]
        for k in s:
            if k not in x:
                res.append(k)
            else:
                res.append(ls.pop())
        return ''.join(res)
```

349、两个数组的交集

给定两个数组，编写一个函数来计算它们的交集。

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

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

题解

1、set之后，遍历其中一个数组，判断是否在另一个数组出现

```python
class Solution:
    def intersection(self, nums1: List[int], nums2: List[int]) -> List[int]:
        res=[]
        nums1=set(nums1)
        nums2=set(nums2)
        for i in nums1:
            if i in nums2:
                res.append(i)

        return res
```

2、set的intersectio()函数

intersection() 方法用于返回两个或更多集合中都包含的元素，即交集。

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

或者使用交集—&

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

350、两个数组的交集II

给定两个数组，编写一个函数来计算它们的交集。

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

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

题解

1、排序数组+双指针

```
class Solution:
    def intersect(self, nums1: [int], nums2: [int]) -> [int]:
        nums1.sort()
        nums2.sort()
        r = []
        left, right = 0, 0
        while left < len(nums1) and right < len(nums2):
            if nums1[left] < nums2[right]:
                left += 1
            elif nums1[left] == nums2[right]:
                r.append(nums1[left])
                left += 1
                right += 1
            else:
                right += 1
        return r
```

2、同上题解1，但要在判断后删除交集元素

```
class Solution:
    def intersect(self, nums1: List[int], nums2: List[int]) -> List[int]:
        tep = []
        for val in nums1:
            if val in nums2:
                tep.append(val)
                nums2.remove(val)

        return tep
```

3、生成计数字典，

```
from collections import Counter
def intersection(nums1,nums2):
    count1 = Counter(nums1)
    count2 = Counter(nums2)
    intersect = []
    for key in count1:
        if key in count2:
            intersect+=[key]*min(count1[key],count2[key])
    return intersect
```

367、有效的完全平方数

给定一个 正整数 num ，编写一个函数，如果 num 是一个完全平方数，则返回 true ，否则返回 false 。

进阶：不要 使用任何内置的库函数，如 sqrt 。

> 输入：num = 16
> 输出：true

> 输入：num = 14
> 输出：false

题解

1、同2的幂，题解1

```
class Solution:
    def isPerfectSquare(self, num: int) -> bool:
        i=1
        while 1:
            if i*i==num:
                return True
            elif i*i<num:
                i+=1
            else:
                return False
```

2、二分查找

```
class Solution:
    def isPerfectSquare(self, num: int) -> bool:
        l, r = 1, num
        while l < r:
            mid = (l + r) // 2
            if mid * mid < num:
                l = mid + 1
            else:
                r = mid
        return l * l == num
```

3、等差数列

- 有一个公式

  1 + 3 + 5 + 7 + ... (2N-1)= N^2

- 所以任意一个平方数可以表示成这样的奇数序列和。

```python
class Solution:
    def isPerfectSquare(self, num: int) -> bool:
        i = 1
        while num > 0:
            num -= i
            i += 2
        return num == 0
```

4、牛顿迭代法

```python
class Solution:
    def isPerfectSquare(self, num: int) -> bool:
        i = num
        while i * i > num:
            i = (i + num / i) // 2
        return i * i == num
```

374、猜数字大小

猜数字游戏的规则如下：

- 每轮游戏，我都会从 1 到 n 随机选择一个数字。 请你猜选出的是哪个数字。
- 如果你猜错了，我会告诉你，你猜测的数字比我选出的数字是大了还是小了。

你可以通过调用一个预先定义好的接口 int guess(int num) 来获取猜测结果，返回值一共有 3 种可能的情况（-1，1 或 0）：

- -1：我选出的数字比你猜的数字小 pick < num
- 1：我选出的数字比你猜的数字大 pick > num
- 0：我选出的数字和你猜的数字一样。恭喜！你猜对了！pick == num
  返回我选出的数字。

> ```
> 输入：n = 10, pick = 6
> 输出：6
> ```
>
> ```
> 输入：n = 2, pick = 1
> 输出：1
> ```

题解

1、二分查找

```python
class Solution:
    def guessNumber(self, n: int) -> int:
        l, r = 1, n
        while l <= r:
            mid = (l+r)//2
            if guess(mid) == -1:
                r = mid-1
            elif guess(mid) == 1:
                l = mid+1
            elif guess(mid) == 0:
                return mid
```

383、赎金信

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。

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

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

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

题解

1、计数字典

```python
class Solution:
    def canConstruct(self, ransomNote: str, magazine: str) -> bool:
        from collections import Counter
        count1 = Counter(ransomNote)
        count2 = Counter(magazine)
        for s in count1:
            if count1[s]>count2[s]:
                return False

        return True
```

387、字符串中的第一个唯一字符

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。

```
s = "leetcode"
返回 0
```

```
s = "loveleetcode"
返回 2
```

题解

1、创建计数字典

- 字典的key—字母
- 字典的value—（个数，索引）

```
class Solution:
    def firstUniqChar(self, s: str) -> int:
        dic={}
        for i,x in enumerate(s):
            if x not in dic:
                dic[x]=(1,i)
            else:
                dic[x]=(dic[x][0]+1,i)
        for i in dic:
            if dic[i][0]==1:
                return dic[i][1]

        return -1
```

优化—一次遍历

- 如字符不在d中，添加key值及对应下标，
- 若存在更新value值为len(s) + 1

```
class Solution:
    def firstUniqChar(self, s):
        d = {}
        length = len(s)
        for i in range(length):
            if s[i] not in d:
                d[s[i]] = i
            else:
                d[s[i]] = length + 1
        ret = min(d.values())
        return -1 if ret > length else ret
```

2、内置函数字典，

```
class Solution:
    def firstUniqChar(self, s):
        d = Counter(s)
        for i, j in enumerate(s):
            if d[j] == 1:
                return i
        return -1
```

389、找不同

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。

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

> 输入：s = "", t = "y"
> 输出："y"

> 输入：s = "", t = "y"
> 输出："y"

题解

1、排序后对比

```
class Solution:
    def findTheDifference(self, s: str, t: str) -> str:
        s=sorted(s)
        t=sorted(t)
        for i in range(len(s)):
            if t[i]!=s[i]:
                return t[i]

        return t[-1]
```

2、遍历t，查看s中是否存在，存在删除，不存在输出

```
class Solution:
    def findTheDifference(self, s: str, t: str) -> str:
        l_s = list(s)
        for word in list(t):
            if word in l_s:
                l_s.remove(word)
            else:
                return word
```

3、合并s和t，出现奇数次的字母即为答案

```python
class Solution:
    def findTheDifference(self, s: str, t: str) -> str:
        a = "".join([s,t])
        for ch in a:
            if a.count(ch) % 2 == 1:
                return ch
```

4、位运算—异或两次即为答案

```python
class Solution:
    def findTheDifference(self, s: str, t: str) -> str:
        ans = 0
        for i in s:
            ans ^= ord(i)

        for i in t:
             ans ^= ord(i)

        return chr(ans)

        #或者一行
        return chr(reduce(xor, map(ord, s+t)))
```

5、求和再相减

```python
class Solution:
    def findTheDifference(self, s: str, t: str) -> str:
        return chr(sum(map(ord, t))- sum(map(ord, s)))
```

6、字典

```python
class Solution:
    def findTheDifference(self, s: str, t: str) -> str:
        countS = collections.Counter(s)
        countT = collections.Counter(t)
        for val in countT:
            if countT[val] != countS[val]:
                return val
```

392、判断子序列

给定字符串 s 和 t ，判断 s 是否为 t 的子序列。

字符串的一个子序列是原始字符串删除一些（也可以不删除）字符而不改变剩余字符相对位置形成的新字符串。（例如，"ace"是"abcde"的一个子序列，而"aec"不是）。

> 输入：s = "abc", t = "ahbgdc"
> 输出：true>

> 输入：s = "axc", t = "ahbgdc"
> 输出：false

题解

1、双指针，一个指s，一个指t，循环t，判断s的指针是否指到最后一位

```
class Solution:
    def isSubsequence(self, s: str, t: str) -> bool:
        if not s:return True
        s1=0
        for t1 in range(len(t)):
            if s[s1]==t[t1]:
                s1+=1
                if s1==len(s):
                    return True

        return False
```

2、递归

```
class Solution:
    def isSubsequence(self, s: str, t: str) -> bool:
        if not s:
            return True
        if s[0] in t:
            inx = t.index(s[0])
            return self.isSubsequence(s[1:], t[inx+1:])
        return False
```

401、二进制手表

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。

```
输入: n = 1
返回: ["1:00", "2:00", "4:00", "8:00", "0:01", "0:02", "0:04", "0:08",
"0:16", "0:32"]
```

题解

1、回溯

- 总体思路
  1. 在10个灯中选num个灯点亮，如果选择的灯所组成的时间已不合理（小时超过11，分钟超过59）就进行剪枝
  2. 也就是从0到10先选一个灯亮，再选当前灯的后面的灯亮，再选后面的灯的后面的灯亮，一直到num个灯点满
- 具体思路
  1. 为了方便计算，分别设置了小时数组和分钟数组
  2. 递归的四个参数分别代表：剩余需要点亮的灯数量，从索引index开始往后点亮灯，当前小时数，当前分钟数
  3. 每次进入递归后，先判断当前小时数和分钟数是否符合要求，不符合直接return
  4. for循环枚举点亮灯的情况，从index枚举到10，每次枚举，
     - 减少一个需要点亮的灯数量num - 1
     - 从当前已点亮的灯后面选取下一个要点亮的灯 i + 1
     - 在hour中增加当前点亮灯的小时数，如果i大于3，当前灯是分钟灯而不是小时灯，则加上0个小时
     - 在minute中增加当前点亮灯的分钟数，如果i没有大于3，当前灯是小时灯而不是分钟灯，则加上0分钟
  5. 当剩余需要点亮的灯数量为0的时候，已枚举完一种情况，根据题目要求的格式加到res列表中

6. 返回res

```
class Solution:
    def readBinaryWatch(self, num: int) -> List[str]:
        hours = [1, 2, 4, 8, 0, 0, 0, 0, 0, 0]
        minutes = [0, 0, 0, 0, 1, 2, 4, 8, 16, 32]
        res = []
        def backtrack(num, index, hour, minute):
            if hour > 11 or minute > 59:
                return
            if num == 0:
                res.append('%d:%02d' % (hour, minute))
                return
            for i in range(index, 10):
                backtrack(num - 1, i + 1, hour + hours[i], minute + minutes[i])

        backtrack(num, 0, 0, 0)
        return res
```

2、暴力法—遍历所有小时和分钟，看起二进制数1的数量

```
class Solution:
    def readBinaryWatch(self, num: int) -> List[str]:
        if num < 0:
            return []
        res = []
        for h in range(12):
            for m in range(60):
                if bin(h).count('1') + bin(m).count('1') == num:
                    res.append('%d:%02d' % (h, m))
        return res
```

11-20

404、左叶子之和

计算给定二叉树的所有左叶子之和。

```
  3
 / \
9  20
  /  \
 15   7

在这个二叉树中，有两个左叶子，分别是 9 和 15，所以返回 24
```

题解

1、DFS

```python
class Solution:
    def sumOfLeftLeaves(self, root: TreeNode) -> int:
        def DFS(root):
            if not root:
                return 0
            #如果是左子树为左叶子节点，答案为左叶子数值加上右子树答案
            if root.left and not root.left.left and not root.left.right:
                return  DFS(root.right) + root.left.val
            # 否则，答案为左右子树答案之和
            return DFS(root.left) + DFS(root.right)
        return DFS(root)
```

2、BFS

```python
class Solution:
    def sumOfLeftLeaves(self, root: TreeNode) -> int:
        import collections
        def BFS(root):
            if not root:
                return 0
            queue = collections.deque()
            queue.append(root)
            res = 0
            while queue:
                node = queue.popleft()
                if node.left and not node.left.left \
                    and not node.left.right:
                    res += node.left.val
                if node.left:
                    queue.append(node.left)
                if node.right:
                    queue.append(node.right)
            return res
        return BFS(root)
```

405、数字转换为十六进制数

给定一个整数，编写一个算法将这个数转换为十六进制数。对于负整数，我们通常使用 [补码运算](#) 方法。

注意

1. 十六进制中所有字母(a-f)都必须是小写。
2. 十六进制字符串中不能包含多余的前导零。如果要转化的数为0，那么以单个字符'0'来表示；对于其他情况，十六进制字符串中的第一个字符将不会是0字符。
3. 给定的数确保在32位有符号整数范围内。
4. 不能使用任何由库提供的将数字直接转换或格式化为十六进制的方法。

> 输入:26
> 输出:"1a"

> 输入:-1
> 输出:"ffffffff"

题解

1、位运算

计算机补码

- 对于正数：正整数的补码是其二进制表示，与原码相同。
- 对于负数：将其原码除符号位外的所有位取反（0变1，1变0，符号位为1不变）后加1。

```
class Solution:
    def toHex(self, num: int) -> str:
        num &= 0xFFFFFFFF
        s = "0123456789abcdef"
        res = ""
        mask = 0b1111
        while num > 0:
            res += s[num & mask]
            num >>= 4
        return res[::-1] if res else "0"
```

- python 中负数前面有无限个 1 表示负数，比如 `5` 的补码形式是 `(0)_infinite 101`，而负数 `-5` 的补码是 `(1)_infinite 011`，也就是前面有无限个 `1`，因此获取负数的补码形式直接用 `hex` 或者 `bin` 是行不通的。
- 因而一个常规的操作，根据本题题意来将负数全部转为正数处理，这样就不会出现负数的 `hex` 函数出现前置 `-` 了。
- 那么就本题举例 `-1` 的 32 位补码形式应该是 `0xffffffff`，所以楼主中调用 `hex` 函数之前，先把负数转化为整数看待，即 `num&0xFFFFFFFF`，所以 `-1` 变成正数 `0xffffffff`，那么就不会出错了，其他负数同理。
- 也就是使用 `&` 操作符，把负数前置无限个 `1` 和 正数 `0xFFFFFFFF` 前置无限个 `0` 与运算，那么负数前置的 `1` 全部被干掉成为 `0`,即成为正数，是 python 中获取 【x 位整型】补码形式的一种操作。

2、库函数

```
class Solution:
    def toHex(self, num: int) -> str:
       return hex(num&0xFFFFFFFF)[2:]
```

409、最长回文串

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。

> 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。

题解

1、计数字典

```
class Solution:
    def longestPalindrome(self, s: str) -> int:
        from collections import Counter
        count1 = Counter(s)
        res=0
        d=0     #判断是否有单数的出现
        for i in count1:
            if count1[i]%2==1:
                x=count1[i]//2
                res+=x*2
                d=1
                continue
            else:
                x=count1[i]//2
                res+=x*2
        if d:
            return res+1
        else:
            return res
```

优化——性能双百

```
res = 0
for letter in set(s):
    count = s.count(letter)
    if count % 2 == 0:
        res += count
    elif count % 2 != 0 and count > 2:
        res += count-1
if len(s) > res:
    res += 1
return res
```

412、Fizz Buzz

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”。

```
n = 15,

返回:
[
 "1",
 "2",
 "Fizz",
 "4",
```

```
 "Buzz",
 "Fizz",
 "7",
 "8",
 "Fizz",
 "Buzz",
 "11",
 "Fizz",
 "13",
 "14",
 "FizzBuzz"]
```

题解

1、正常循环遍历，使用多个if判断

```python
class Solution:
    def fizzBuzz(self, n: int) -> List[str]:
        res=[]
        for i in range(1,n+1):
            if i%3==0 and i%5==0:
                res.append('FizzBuzz')
            elif i%3==0:
                res.append('Fizz')
            elif i%5==0:
                res.append('Buzz')
            else:
                res.append(str(i))

        return res
```

414、第三大的数

给你一个非空数组，返回此数组中 **第三大的数** 。如果不存在，则返回数组中最大的数。

```
输入：[3, 2, 1]
输出：1
解释：第三大的数是 1 。
```

```
输入：[1, 2]
输出：2
解释：第三大的数不存在, 所以返回最大的数 2 。
```

```
输入：[2, 2, 3, 1]
输出：1
解释：注意，要求返回第三大的数，是指在所有不同数字中排第三大的数。
此例中存在两个值为 2 的数，它们都排第二。在所有不同数字中排第三大的数为 1 。
```

题解

1、set()函数+remove()函数

```
class Solution:
    def thirdMax(self, nums: List[int]) -> int:
        nums=set(nums)
        if len(nums)<3:return max(nums)
        for i in range(3):
            res=max(nums)
            nums.remove(res)

        return res
```

2、set()函数+sort()函数

```
class Solution:
    def thirdMax(self, nums: List[int]) -> int:
        set_nums=list(set(nums))
        set_nums.sort()
        return set_nums[-1] if len(set_nums)<3 else set_nums[-3]
```

415、字符串相加

给定两个字符串形式的非负整数 `num1` 和`num2` ，计算它们的和。

提示：

1. num1 和num2 的长度都小于 5100
2. num1 和num2 都只包含数字 0-9
3. num1 和num2 都不包含任何前导零
4. 你不能使用任何内建 BigInteger 库， 也不能直接将输入的字符串转换为整数形式

题解

1、创建字典：{字符—数字}

```
class Solution:
    def addStrings(self, num1: str, num2: str) -> str:
        dic={}
        for i in range(10):
            dic[str(i)]=i
        n1,n2=0,0
        for i in range(len(num1)):
            n1+=10**i*dic[num1[-i-1]]
        for i in range(len(num2)):
            n2+=10**i*dic[num2[-i-1]]

        return str(n1+n2)
```

2、双指针—模拟人工加法

```python
class Solution:
    def addStrings(self, num1: str, num2: str) -> str:
        res = ""
        i, j, carry = len(num1) - 1, len(num2) - 1, 0
        while i >= 0 or j >= 0:
            n1 = int(num1[i]) if i >= 0 else 0
            n2 = int(num2[j]) if j >= 0 else 0
            tmp = n1 + n2 + carry
            carry = tmp // 10
            res = str(tmp % 10) + res
            i, j = i - 1, j - 1
        return "1" + res if carry else res
```

434、字符串中的单词数

统计字符串中的单词个数，这里的单词指的是连续的不是空格的字符。

请注意，你可以假定字符串里不包括任何不可打印的字符。

> 输入: "Hello, my name is John"
> 输出: 5
> 解释: 这里的单词是指连续的不是空格的字符，所以 "Hello," 算作 1 个单词。

题解

1、split()函数

```python
class Solution:
    def countSegments(self, s: str) -> int:
        s = s.split(' ')
        cnt = 0
        for i in s:
            if i!= '':cnt+=1
        return cnt
```

2、遍历查找空格

```python
class Solution:
    def countSegments(self, s: str) -> int:
        res = 0
        flag = False
        count = 0

        for i in range(len(s)):
            if s[i] != ' ':
                count += 1
                flag = True
            else:
                if flag and count > 0:
                    res += 1
                    count = 0
```

```
            if i == len(s)-1 and count > 0:
                res += 1

        return res
```

441、排列硬币

你总共有 n 枚硬币，你需要将它们摆成一个阶梯形状，第 k 行就必须正好有 k 枚硬币。

给定一个数字 n，找出可形成完整阶梯行的总行数。

n 是一个非负整数，并且在32位有符号整型的范围内。

> ```
> n = 5
>
> 硬币可排列成以下几行:
> ¤
> ¤ ¤
> ¤ ¤
>
> 因为第三行不完整，所以返回2.
> ```

题解

1、while，每次减去一层的数

```
class Solution:
    def arrangeCoins(self, n: int) -> int:
        if n==1:return 1
        i=1
        while n>=0:
            n-=i
            i+=1

        return i-2
```

2、数学法

题目等价于求满足k(k + 1) <= 2n的k的最大值

$$\max_{k} k \times (k + 1) \leq 2 \times n$$

```
class Solution:
    def arrangeCoins(self, n: int) -> int:
        k = int(math.sqrt(2 * n))
        while k * (k + 1) > 2 * n:
            #不停缩小k值
            k -= 1
        return k
```

3、二分法查找，

```python
class Solution:
    def arrangeCoins(self, n: int) -> int:
        r = int(math.sqrt(2 * n))
        l = 0
        #这里要用<=符号，目的为了缩小区间到l - 1
        while l <= r:
            mid = (r + l) >> 1
            num = mid * (mid + 1)
            if num == 2 * n:
                #正好相等直接返回
                return mid
            elif num < 2 * n:
                #小于的话 l 加一
                l = mid + 1
            else:
                #大于的话 r 减一
                r = mid - 1
        return r
```

4、等差求和公式—`x = ((1+8*n)**0.5 - 1)/2`

```python
class Solution:
    def arrangeCoins(self, n: int) -> int:
        return int(((8 * n + 1) ** 0.5 - 1) // 2)
```

453、最小操作次数使数组元素相等

给定一个长度为 *n* 的 **非空** 整数数组，每次操作将会使 *n* - 1 个元素增加 1。找出让数组所有元素相等的最小操作次数。

> 输入：
> [1,2,3]
> 输出：
> 3
> 解释：
> 只需要3次操作（注意每次操作会增加两个元素的值）：
> [1,2,3] => [2,3,3] => [3,4,3] => [4,4,4]

题解

1、数学推导

1. 假设我们最少的操作次数是k,k次操作后每个元素相等，相等元素设为target
2. 对于整个列表的n - 1个元素都要进行加一操作那么增加的总数是 k * (n - 1)
3. 原本的列表之和为 sum(nums)，k次操作后应该存在这样的关系等式：

   k[最少操作次数] * (n - 1)[每次对n - 1个元素进行操作] + sum(nums)[原列表的和] = target[操作后的相等元素] * n

$$k \times (n - 1) + sum(nums) = target \times n$$

`target`的值是：k + nums中的最小值 即：min(nums) + k

因为对于最小值，你每次的递加都必须对原列表的最小值加一，每次操作中必须覆盖最小值， k次操作后， 最小值就变为了min(nums) + k，该值就是最后的相同值

$$k = sum(nums) - min(nums) \times n$$

```
class Solution:
    def minMoves(self, nums: List[int]) -> int:
        return sum(nums) - len(nums) * min(nums) if len(nums) != 1 else 0
```

2、加法变减法

n个数中n-1个数加1，相当于每次有1个数减1

```
class Solution:
    def minMoves(self, nums: List[int]) -> int:
        n=min(nums)
        res=0
        for i in nums:
            res+=i-n
        return res
```

455、分发饼干

假设你是一位很棒的家长，想要给你的孩子们一些小饼干。但是，每个孩子最多只能给一块饼干。

对每个孩子 i，都有一个胃口值 g[i]，这是能让孩子们满足胃口的饼干的最小尺寸；并且每块饼干 j，都有一个尺寸 s[j] 。如果 s[j] >= g[i]，我们可以将这个饼干 j 分配给孩子 i ，这个孩子会得到满足。你的目标是尽可能满足越多数量的孩子，并输出这个最大数值。

> 输入: g = [1,2,3], s = [1,1]
> 输出: 1
> 解释:
> 你有三个孩子和两块小饼干，3个孩子的胃口值分别是：1,2,3。
> 虽然你有两块小饼干，由于他们的尺寸都是1，你只能让胃口值是1的孩子满足。
> 所以你应该输出1。

> 输入: g = [1,2], s = [1,2,3]
> 输出: 2
> 解释:
> 你有两个孩子和三块小饼干，2个孩子的胃口值分别是1,2。
> 你拥有的饼干数量和尺寸都足以让所有孩子满足。
> 所以你应该输出2.

题解

1、遍历g，找到s最大的分给g合适的同学，remove

```
class Solution:
```

```
    def findContentChildren(self, g: List[int], s: List[int]) -> int:
        if not s or not g:return 0
        res=0
        for i in range(len(g)):
            if not s:break
            g_m=max(g)
            s_m=max(s)
            if s_m>=g_m:
                res+=1
                s.remove(s_m)
            g.remove(g_m)

        return res
```

2、双指针

1. 首先要对g 和 s排序：题中没有说明输入是有序序列
2. 设两个指针分别指向两个列表的末尾如果满足`s[r_s] >= g[r_g]`，则都向前移动：
3. 否则只需要移动g的指针

```
class Solution:
    def findContentChildren(self, g: List[int], s: List[int]) -> int:
        #首先要对g 和 s排序：题中没有说明输入是有序序列
        g.sort()
        s.sort()
        #两个指针分别指向两个列表的末尾
        r_g = len(g) - 1
        r_s = len(s) - 1
        count = 0
        #循环条件
        while r_g >= 0 and r_s >= 0:
            #两个指针分别指向两个列表的末尾如果满足`s[r_s] >= g[r_g]`，则都向前移动
            if s[r_s] >= g[r_g]:
                count += 1
                r_g -= 1
                r_s -= 1
            #否则只需要移动g的指针
            else:
                r_g -= 1
        return count
```

21-30

459、重复的子字符串

给定一个非空的字符串，判断它是否可以由它的一个子串重复多次构成。给定的字符串只含有小写英文字母，并且长度不超过10000。

> 输入: "abab"
> 输出: True
> 解释: 可由子字符串 "ab" 重复两次构成。

```
输入: "aba"
输出: False
```

```
输入: "abcabcabcabc"
输出: True
解释: 可由子字符串 "abc" 重复四次构成。 (或者子字符串 "abcabc" 重复两次构成。)
```

题解

1、构造双倍字符串

- 假设 s 可由 子串 x 重复 n 次构成，即 s = nx
- 则有 s+s = 2nx
- 移除 s+s 开头和结尾的字符，变为 (s+s)[1:-1]，则破坏了开头和结尾的子串 x
- 此时只剩 2n-2 个子串
- 若 s 在 (s+s)[1:-1] 中，则有 2n-2 >= n，即 n >= 2
- 即 s 至少可由 x 重复 2 次构成
- 否则，n < 2，n 为整数，只能取 1，说明 s 不能由其子串重复多次构成

```python
class Solution:
    def repeatedSubstringPattern(self, s: str) -> bool:
        return s in (s+s)[1:-1]
```

2、乘法运算

建立一个子串，子串乘以倍数等于s那么就符合题意

```python
class Solution:
    def repeatedSubstringPattern(self, s: str) -> bool:
        subset = ''
        for i in range(len(s)//2):
            subset += s[i]
            if len(s) % len(subset) == 0 and subset*(len(s)//len(subset)) == s:
                return True

        return False
```

3、KMP

next数组记载了一个字符串的部分匹配值，根据next数组可以计算出子字符串长度

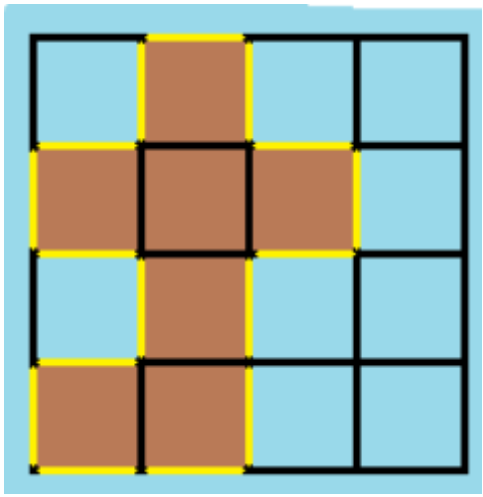
```python
class Solution:
    def repeatedSubstringPattern(self, s: str) -> bool:
        pnext = [0, 0]
        j = 0

        for i in range(1, len(s)):
            while j > 0 and s[i] != s[j]:
                j = pnext[j]
            if s[j] == s[i]:
                j += 1
            pnext.append(j)

        return len(s) % (len(s)-pnext[-1]) == 0 and pnext[-1] > 0
```

463、岛屿的周长

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。



> 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

题解

1、每个方块为1，边长+4，如果周围四个位置也是陆地，则边长-1

```
class Solution:
    def islandPerimeter(self, grid: List[List[int]]) -> int:
        res=0
        for i in range(len(grid)):
            for j in range(len(grid[0])):
                if grid[i][j]==1:
                    res+=4
                    if i>0 and grid[i-1][j]==1:
                        res-=1
                    if i<len(grid)-1 and grid[i+1][j]==1:
                        res-=1
                    if j>0 and grid[i][j-1]==1:
                        res-=1
                    if j<len(grid[0])-1 and grid[i][j+1]==1:
                        res-=1

        return res
```

2、卷积

算周长说白了就是找边缘，找边缘说白了就是锐化，所以可以按照锐化的算法来算。

这个卷积核就是拉普拉斯算子[[0,-1,0],[-1,4,-1],[0,-1,0]]的一半，正负无所谓，只要异号就行，[[2,-1],[-1,0]]和[[-2,1],[1,0]]的结果是一样的。直接用拉普拉斯算子算也可以，不过因为拉普拉斯算子是二阶微分算子，每条边找了两次，所以结果要除2。

```
from scipy.signal import convolve2d
class Solution:
    def islandPerimeter(self, grid: List[List[int]]) -> int:
        return int(abs(convolve2d(grid,[[-2,1],[1,0]])).sum())
```

476、数字的补数

给你一个 **正** 整数 `num` ，输出它的补数。补数是对该数的二进制表示取反。

> 输入：num = 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 。

> 输入：num = 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 。

题解

1、变二进制str，遍历一遍取反，

```
class Solution:
    def findComplement(self, num: int) -> int:
        num_s=str(bin(num))[2:]
        res_s=''
        for i in num_s:
            if i=='0':
                res_s+='1'
            else:
                res_s+='0'
        return int(res_s,2)
```

2、数学计算

num和补数相加，就是满数位1的二进制数，即2**(n-1)-1

```
class Solution:
    def findComplement(self, num: int) -> int:
        return 2**(len(bin(num))-2)-num-1
        #    补数=num取反+1
```

482、密钥格式化

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。

> 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。

> 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。

题解

1、拆分变大写，len(s)%k，余数不为0，则为第一部分，

```
class Solution:
    def licenseKeyFormatting(self, s: str, k: int) -> str:
        s=s.replace('-','').upper()
        res=''
        n=len(s)%k
        if n!=0:
            res=s[:n]
        while n<len(s):
            res+='-'
            if len(res)==1:
                res=''
            for i in range(k):
                res+=s[n+i]
            n+=k

        return res
```

2、逆序排列

```
class Solution:
    def licenseKeyFormatting(self, S: str, K: int) -> str:
        tmp = ''.join(S.split('-')).upper()[::-1]
        res = [tmp[i:i+K] for i in range(0,len(tmp),K)]
        return '-'.join(res)[::-1]
```

485、最大连续1的个数

给定一个二进制数组， 计算其中最大连续 1 的个数。

> 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.

题解

1、存储临时长度

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

        return res
```

2、转成字符串—根据'0'分割，选取最长连续1字符

```
class Solution:
    def findMaxConsecutiveOnes(self, nums: List[int]) -> int:
        return max([len(s) for s in (''.join([str(n) for n in
nums])).split('0')])
```

3、动态规划

当前位置的连续1的数量

```
class Solution:
    def findMaxConsecutiveOnes(self, nums: List[int]) -> int:
        # 动态数组
        for i in range(1, len(nums)):
            if nums[i] == 1:
                nums[i] = nums[i] + nums[i - 1]
        return max(nums)
```

492、构造矩形

现给定一个具体的矩形页面面积，你的任务是设计一个长度为 L 和宽度为 W 且满足以下要求的矩形的页面。要求：

1. 你设计的矩形页面必须等于给定的目标面积。

2. 宽度 W 不应大于长度 L，换言之，要求 L >= W 。
3. 长度 L 和宽度 W 之间的差距应当尽可能小。

> 输入: 4
> 输出: [2, 2]
> 解释: 目标面积是 4， 所有可能的构造方案有 [1,4], [2,2], [4,1]。
> 但是根据要求2，[1,4] 不符合要求; 根据要求3，[2,2] 比 [4,1] 更能符合要求. 所以输出长度 L 为 2， 宽度 W 为 2。

题解

1、从平方根开始遍历，找到余数为0的数

```
class Solution:
    def constructRectangle(self, area: int) -> List[int]:
        x=int(area**0.5)
        if x*x==area:
            return [x,x]
        else:
            for i in range(x+1,area+1):
                if area%i==0:
                    return [i,area//i]
```

优化

```
class Solution(object):
    def constructRectangle(self, area):
        nq = int(area**0.5)
        while area%nq != 0:
            nq = nq-1
        return [area/nq,nq]
```

496、下一个更大元素 I

给你两个 没有重复元素 的数组 nums1 和 nums2 ， 其中nums1 是 nums2 的子集。

请你找出 nums1 中每个元素在 nums2 中的下一个比其大的值。

nums1 中数字 x 的下一个更大元素是指 x 在 nums2 中对应位置的右边的第一个比 x 大的元素。如果不存在，对应位置输出 -1 。

> 输入: nums1 = [4,1,2], nums2 = [1,3,4,2].
> 输出: [-1,3,-1]
> 解释:
> 对于 num1 中的数字 4 ，你无法在第二个数组中找到下一个更大的数字，因此输出 -1 。
> 对于 num1 中的数字 1 ，第二个数组中数字1右边的下一个较大数字是 3 。
> 对于 num1 中的数字 2 ，第二个数组中没有下一个更大的数字，因此输出 -1 。

> 输入: nums1 = [2,4], nums2 = [1,2,3,4].
> 输出: [3,-1]
> 解释:
> 对于 num1 中的数字 2 ，第二个数组中的下一个较大数字是 3 。
> 对于 num1 中的数字 4 ，第二个数组中没有下一个更大的数字，因此输出 -1 。

题解

1、暴力搜索，

```
class Solution:
    def nextGreaterElement(self, nums1: List[int], nums2: List[int]) ->
List[int]:
        #字典存储结果
        total_list = {}
        #遍历每个元素
        for i in range(len(nums2)):
            for j in range(i + 1, len(nums2)):
                #第一个比当前值大的元素即是当前值key的value
                if nums2[j] > nums2[i]:
                    total_list[nums2[i]] = nums2[j]
                    break
                #如果没有则-1
                if j == len(nums2) - 1:
                    total_list[nums2[i]] = -1
        total_list[nums2[-1]] = -1
        result = []
        #返回每个key值对应的value
        for num in nums1:
            result.append(total_list[num])
        return result
```

2、单调栈

```
class Solution:
    def nextGreaterElement(self, nums1: List[int], nums2: List[int]) ->
List[int]:
        #单调栈
        #字典存储结果
        nums = {}
        #列表模拟栈
        s = []
        #遍历每个字符串
        for i in range(len(nums2)):
            #如果当前栈为空，则直接入栈
            if len(s) == 0:
                s.append(nums2[i])
            else:
                #当前值小于栈顶元素，则入栈
                if nums2[i] < s[-1]:
                    s.append(nums2[i])
                elif nums2[i] > s[-1]:
                    #当前值大于栈顶元素，不停出栈，把所有栈顶key值的value赋值为当前值
                    while len(s) > 0 and s[-1] < nums2[i]:
                        nums[s[-1]] = nums2[i]
```

```
                    s.pop()
                #当前值入队列
                s.append(nums2[i])
        #如果栈中还有元素，则全部赋值为-1，表示右边没有更大值
        while len(s) > 0:
            nums[s[-1]] = -1
            s.pop()
        result = []
        #返回每个key值对应的value
        for i in nums1:
            result.append(nums[i])
        return result
```

500、键盘行

给你一个字符串数组 words ， 只返回可以使用在 美式键盘 同一行的字母打印出来的单词。键盘如下图所示。

美式键盘 中：

- 第一行由字符 "qwertyuiop" 组成。
- 第二行由字符 "asdfghjkl" 组成。
- 第三行由字符 "zxcvbnm" 组成。

> ```
> 输入：words = ["Hello","Alaska","Dad","Peace"]
> 输出：["Alaska","Dad"]
> ```
>
> ```
> 输入：words = ["adsdf","sfd"]
> 输出：["adsdf","sfd"]
> ```

题解

1、创建字典，遍历单词字母，如果最大等于最小，则表示在同一行

```
class Solution:
    def findWords(self, words: List[str]) -> List[str]:
        dic={}
        for i in 'qwertyuiop':
            dic[i]=1
        for i in 'asdfghjkl':
            dic[i]=2
        for i in 'zxcvbnm':
            dic[i]=3

        res=[]
        for word in words:
            wor=word.lower()
            tmp=[]
            for w in wor:
                tmp.append(dic[w])
            if max(tmp)==min(tmp):
                res.append(word)
```

```
        return res
```

2、使用交集

```
class Solution:
    def findWords(self, words: List[str]) -> List[str]:
        l1 = 'qwertyuiopQWERTYUIOP'
        l2 = 'asdfghjklASDFGHJKL'
        l3 = 'zxcvbnmZXCVBNM'
        l1,l2,l3 = set(l1),set(l2),set(l3)
        final = []
        for i in range(len(words)):
            s = set(words[i])
            if s&l1 == s or s&l2==s or s&l3==s:
                final.append(words[i])
        return final
```

501、二叉搜索树中的众数

给定一个有相同值的二叉搜索树（BST），找出 BST 中的所有众数（出现频率最高的元素）。

假定 BST 有如下定义：

- 结点左子树中所含结点的值小于等于当前结点的值
- 结点右子树中所含结点的值大于等于当前结点的值
- 左子树和右子树都是二叉搜索树

> ```
> BST [1,null,2,2]
> 1
>  \
>   2
>  /
> 2
> 返回[2]
> ```

题解

1、DFS—使用DFS制作二叉树的字典（数字，个数），输出个数最多的数字

```
class Solution:
    def findMode(self, root: TreeNode) -> List[int]:
        dic={}
        res=[]
        def dfs(root):
            if not root:return
            if root.val not in dic:
                dic[root.val]=1
            else:
                dic[root.val]+=1
            dfs(root.left)
            dfs(root.right)
```

```
        dfs(root)
        max_d=max(dic.values())
        for i in dic:
            if dic[i]==max_d:
                res.append(i)

        return res
```

2、BFS—同上，创造{数字-个数}字典

```
class Solution:
    def findMode(self, root: TreeNode) -> List[int]:

        if not root:
            return []

        ##用字典存每个节点出现的值
        dicts = {}

        ##使用BFS访问每个节点的值
        deque = collections.deque()
        deque.append(root)

        while len(deque)>0:
            currVal = deque.popleft()
            if currVal.val not in dicts:
                dicts[currVal.val] = 1
            else:
                dicts[currVal.val]+=1

            if currVal.left:
                deque.append(currVal.left)
            if currVal.right:
                deque.append(currVal.right)

        # print(dicts,max(dicts.values()))
        maxVal =  max(dicts.values())
        res = []
        for item,val in dicts.items():
            if val>=maxVal:
                res.append(item)
        return  res
```

3、中序遍历

- 搜索二叉树的中序遍历是有序数组
- 中序遍历二叉树将其转化为有序数组
- 寻找众数只需判断当前数值是否等于前一个数值
- 无需另立`ls`来记录整个数组
- 只需记录上一个数值`last`

```
class Solution:
    def findMode(self, root: TreeNode) -> List[int]:
        ans=[]
        most=0
        last=None
```

```
        cnt=0

        def inorder(node):
            if not node: return
            nonlocal ans,most,last,cnt
            if node.left: inorder(node.left)
            if node.val==last:
                cnt+=1
            else: cnt=1
            if cnt==most: ans.append(node.val)
            elif cnt>most:
                most=cnt
                ans=[node.val]
            last=node.val
            if node.right: inorder(node.right)

        inorder(root)
        return ans
```

504、七进制数

给定一个整数，将其转化为7进制，并以字符串形式输出。

> 输入: 100
> 输出: "202"

> 输入: -7
> 输出: "-10"

题解

1、进制算法

```
class Solution:
    def convertToBase7(self, num: int) -> str:
        if num<0:
            num=abs(num)
            res='-'
        else:
            res=''
        a=''
        while num//7!=0:
            a+=str(num%7)
            num//=7
        a+=str(num)
        return res+a[::-1]
```

2、递归

```python
class Solution:
    def convertToBase7(self, num: int) -> str:
        if num < 0:
            return "-" + self.convertToBase7(-num)
        if num < 7:
            return str(num)
        return self.convertToBase7(num // 7) + str(num % 7)
```

31-40

506、相对名次

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。)

> 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。

题解

1、创造一个排序数组，根据索引返回最终名次

```python
class Solution:
    def findRelativeRanks(self, score: List[int]) -> List[str]:
        res=[0]*len(score)
        num=sorted(score,reverse=True)
        for i in range(len(num)):
            if i ==0:
                res[score.index(num[i])]='Gold Medal'
            elif i==1:
                res[score.index(num[i])]='Silver Medal'
            elif i==2:
                res[score.index(num[i])]='Bronze Medal'
            else:
                res[score.index(num[i])]=str(i+1)

        return res
```

507、完美数

对于一个 正整数，如果它和除了它自身以外的所有 正因子 之和相等，我们称它为 「完美数」。

给定一个 整数 n， 如果是完美数，返回 true，否则返回 false

> 输入：28
> 输出：True
> 解释：28 = 1 + 2 + 4 + 7 + 14
> 1，2，4，7，和 14 是 28 的所有正因子。

> 输入：num = 6
> 输出：true

题解

1、遍历num//2+1，找到余数为0的数，保存求和—超时间

```
class Solution:
    def checkPerfectNumber(self, num: int) -> bool:
        res=[]
        for i in range(1,num//2+1):
            if num%i==0:
                res.append(i)

        return sum(res)==num
```

2、优化，枚举sqrt

```
class Solution:
    def checkPerfectNumber(self, num: int) -> bool:
        #对于数字1:直接返回`False`
        if num == 1:
            return False
        #计数从1开始
        total = 1
        #我们只需要判断`2-int(sqrt(num))+1`的数，全部累加
        for i in range(2, int(math.sqrt(num)) + 1):
            if num % i == 0:
                #这里total要加上i和num // i
                total += (i + num // i)
        return total == num
```

3、32位内只有5个完美数

```
class Solution:
    def checkPerfectNumber(self, num: int) -> bool:
        return num in {6, 28, 496, 8128, 33550336}
```

520、检测大写字母

给定一个单词，你需要判断单词的大写使用是否正确。

我们定义，在以下情况时，单词的大写用法是正确的：

- 全部字母都是大写，比如"USA"。
- 单词中所有字母都不是大写，比如"leetcode"。
- 如果单词不只含有一个字母，只有首字母大写， 比如 "Google"。

否则，我们定义这个单词没有正确使用大写字母。

> 输入: "USA"
> 输出: True

> 输入: "FlaG"
> 输出: False

题解

1、islower和issupper函数

```
class Solution:
    def detectCapitalUse(self, word: str) -> bool:
        return word.islower() or word.isupper() or (word[0].isupper() and
word[1:].islower())
```

2、判断大写字母和小写字母数量

```
class Solution(object):
    def detectCapitalUse(self, word):
        lower = upper = 0
        strLength = len(word)
        # 通过比较ASCII的大小来统计字符串中大小写字符的个数
        for i in range(0,strLength):
            if word[i] >= 'a':
                lower = lower+1
            else:
                upper = upper+1
        return lower == strLength or upper == strLength or (upper==1 and word[0]
<'a')
```

521、最长特殊序列 I

给你两个字符串，请你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。

> 输入: "aba", "cdc"
> 输出: 3
> 解释: 最长特殊序列可为 "aba" (或 "cdc")，两者均为自身的子序列且不是对方的子序列。

> 输入：a = "aaa", b = "bbb"
> 输出：3

> 输入：a = "aaa", b = "aaa"
> 输出：-1

题解

1、a，b不相等的话就返回a，b最长的长度，否则就-1

```
class Solution:
    def findLUSlength(self, a: str, b: str) -> int:
        return max(len(a),len(b)) if a!=b else -1
```

530、二叉搜索树的最小绝对差

给你一棵所有节点为非负值的二叉搜索树，请你计算树中任意两节点的差的绝对值的最小值。

> 输入：
>
> 1
> \
>   3
> /
> 2
>
> 输出：
> 1

题解

1、保存到数组，排序，输出最小绝对差

```
class Solution:
    def getMinimumDifference(self, root: TreeNode) -> int:
        res=1000
        dic=[]
        def dfs(root):
            if not root:return
            dic.append(root.val)
            dfs(root.left)
            dfs(root.right)
        dfs(root)
        dic=sorted(dic)
```

```
        for i in range(len(dic)-1):
            res=min(res,dic[i+1]-dic[i])

        return res
```

2、中序遍历

```
class Solution:
    def getMinimumDifference(self, root: TreeNode) -> int:
        pre = None
        res_min = float("inf")
        def inorder(root):
            nonlocal res_min, pre
            if not root:
                return None
            inorder(root.left)
            # 处理节点
            if pre:
                res_min = min(res_min, root.val - pre.val)
            pre = root
            inorder(root.right)
        inorder(root)
        return res_min
```

3、使用栈

```
class Solution:
    def getMinimumDifference(self, root: TreeNode) -> int:
        pre,ans=float(-inf),float(inf)
        stack=[]
        while stack or root:
            while root:
                stack.append(root)
                root=root.left
            root=stack.pop(-1)
            ans=min(ans,abs(root.val-pre))
            pre=root.val
            root=root.right
        return ans
```

509、斐波那契数

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
```

给你 `n` ，请计算 `F(n)` 。

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

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

题解

1、就正常循环就好了

```
class Solution:
    def fib(self, n: int) -> int:
        a, b = 0, 1
        for i in range(n):
            a, b = b, a + b
        return a
```

2、递归

```
class Solution:
    def fib(self, n: int) -> int:
        @lru_cache(None)
        def f(n):
            if n < 2:
                return n
            return f(n - 1) + f(n - 2)

        return f(n)
```

3、动态规划

```
class Solution:
    def fib(self, n: int) -> int:
        data = [0, 1]
        for i in range(n-1):
            data.append(data[-2] + data[-1])
        return data[n]
```

541、反转字符串II

给定一个字符串 s 和一个整数 k，你需要对从字符串开头算起的每隔 2k 个字符的前 k 个字符进行反转。

- 如果剩余字符少于 k 个，则将剩余字符全部反转。
- 如果剩余字符小于 2k 但大于或等于 k 个，则反转前 k 个字符，其余字符保持原样。

> 输入: s = "abcdefg", k = 2
> 输出: "bacdfeg"

题解

1、使用list

```
class Solution:
    def reverseStr(self, s: str, k: int) -> str:
        n=len(s)
        arr=list(s)
        for i in range(0,n,2*k):
            arr[i:i+k]=arr[i:i+k][::-1]
        return "".join(arr)
```

2、python切片

```
class Solution:
    def reverseStr(self, s: str, k: int) -> str:
        res, flag = "", True
        for i in range(0, len(s), k):
            res += s[i:i + k][::-1] if flag else s[i:i+k]
            flag = not flag
        return res
```

551、学生出勤记录 I

给定一个字符串来代表一个学生的出勤记录，这个记录仅包含以下三个字符：

1. 'A' : Absent，缺勤
2. 'L' : Late，迟到
3. 'P' : Present，到场

如果一个学生的出勤记录中不超过一个'A'(缺勤)并且不超过两个连续的'L'(迟到),那么这个学生会被奖赏。

你需要根据这个学生的出勤记录判断他是否会被奖赏。

> 输入: "PPALLP"
> 输出: True

> 输入: "PPALLL"
> 输出: False

题解

1、条件判断

```
class Solution:
    def checkRecord(self, s: str) -> bool:
        A=0
        L=0
        for i in s:
            if i == 'P':
                L=0
            elif i =="A":
```

```
                L=0
                A+=1
                if A>1:
                    return False
            elif i == 'L':
                L+=1
                if L>2:
                    return False

        return True
```

2、简单写法

```
class Solution:
    def checkRecord(self, s: str) -> bool:
        if s.count("A") > 1 or "LLL" in s:
            return False
        return True
```

557、反转字符串中的单词Ⅲ

给定一个字符串，你需要反转字符串中每个单词的字符顺序，同时仍保留空格和单词的初始顺序。

> 输入："Let's take LeetCode contest"
> 输出："s'teL ekat edoCteeL tsetnoc"

题解

1、split后反转拼接

```
class Solution:
    def reverseWords(self, s: str) -> str:
        s=s.split()
        res=[]
        for i in s:
            res.append(i[::-1])

        return ' '.join(res)
```

2、栈

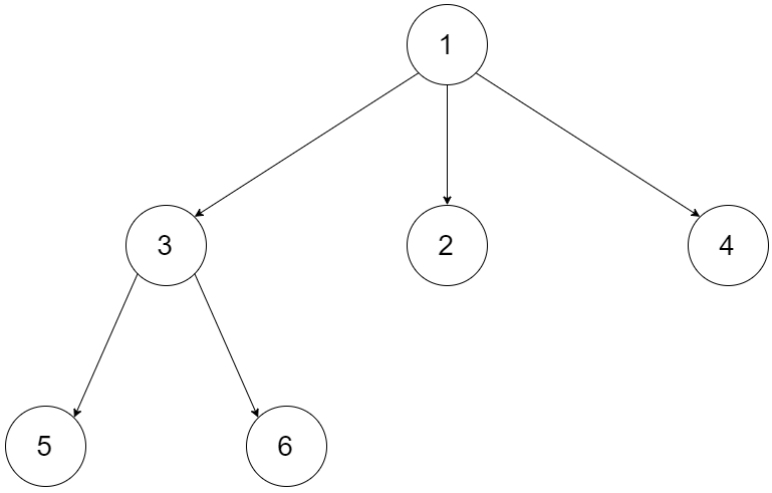
```
class Solution:
    def reverseWords(self, s: str) -> str:
        stack, res, s = [], "", s + " "
        for i in s:
            stack.append(i)
            if i == " ":
                while(stack):
                    res += stack.pop()
        return res[1:]
```

559、N叉树的最大深度

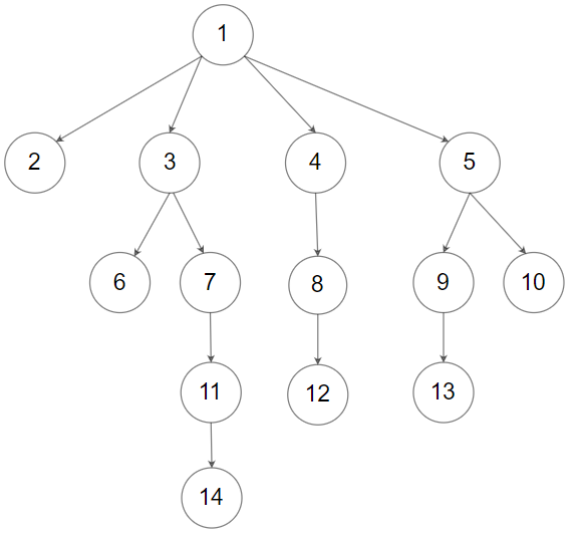
给定一个 N 叉树，找到其最大深度。

最大深度是指从根节点到最远叶子节点的最长路径上的节点总数。

N 叉树输入按层序遍历序列化表示，每组子节点由空值分隔（请参见示例）。




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




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

题解

1、BFS

```python
class Solution:
    def maxDepth(self, root: 'Node') -> int:
        if not root: return 0
        depth=1
        queue=[(root,depth)]
        while queue:
            root,depth=queue.pop(0)
            if root.children:
                for node in root.children:
                    queue.append((node,depth+1))
        return depth
```

2、DFS

```python
class Solution:
    def maxDepth(self, root: 'Node') -> int:
        def getDepth(node: 'Node'):
            if not node: return 0
            child_depth = 0
            for i in range(len(node.children)):
                child_depth = max(child_depth, getDepth(node.children[i]))
            return 1 + child_depth

        return getDepth(root)
```

3、迭代DFS

```python
class Solution:
    def maxDepth(self, root: 'Node') -> int:
        if not root: return 0
        final_depth=1
        stack=[(root,final_depth)]
        while stack:
            root,depth=stack.pop()
            final_depth=max(final_depth,depth)
            if root.children:
                for node in root.children:
                    stack.append((node,depth+1))
        return final_depth
```

41-50

561、数组拆分 I

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。

```
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
```

```
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
```

题解

1、排序，选择第2i个数

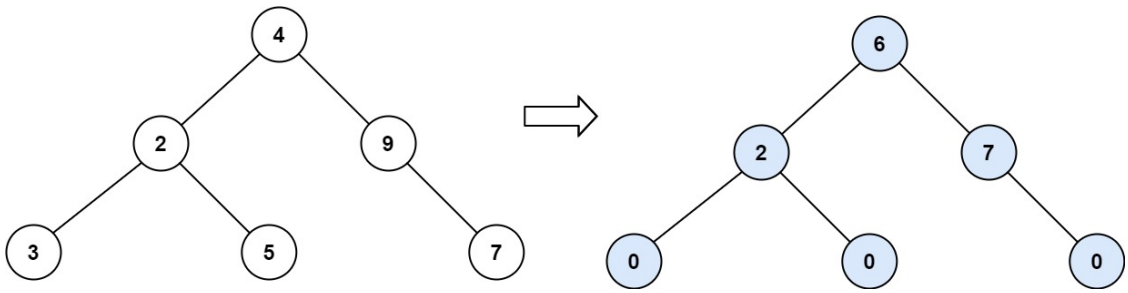
```python
class Solution:
    def arrayPairSum(self, nums: List[int]) -> int:
        res=0
        nums=sorted(nums)
        for i in range(len(nums)//2):
            res+=nums[i*2]

        return res
```

563、二叉树的坡度

给定一个二叉树，计算 整个树 的坡度 。

一个树的 节点的坡度 定义即为，该节点左子树的节点之和和右子树节点之和的 差的绝对值 。如果没有左子树的话，左子树的节点之和为 0 ； 没有右子树的话也是一样。空结点的坡度是 0 。

整个树 的坡度就是其所有节点的坡度之和。



```
输入：root = [4,2,9,3,5,null,7]
输出：15
解释：
节点 3 的坡度：|0-0| = 0（没有子节点）
节点 5 的坡度：|0-0| = 0（没有子节点）
节点 7 的坡度：|0-0| = 0（没有子节点）
节点 2 的坡度：|3-5| = 2（左子树就是左子节点，所以和是 3 ；右子树就是右子节点，所以和是 5 ）
节点 9 的坡度：|0-7| = 7（没有左子树，所以和是 0 ；右子树正好是右子节点，所以和是 7 ）
节点 4 的坡度：|(3+5+2)-(9+7)| = |10-16| = 6（左子树值为 3、5 和 2 ，和是 10 ；右子树值为 9 和 7 ，和是 16 ）
坡度总和：0 + 0 + 0 + 2 + 7 + 6 = 15
```

题解

1、DFS

```python
class Solution:
    def findTilt(self, root: TreeNode) -> int:
        self.res=0
        def dfs(root):
            if not root:return 0
            left=dfs(root.left)
            right=dfs(root.right)
            self.res+=abs(left-right)

            return left+root.val+right

        dfs(root)
        return self.res
```

566、重塑矩阵

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

给出一个由二维数组表示的矩阵，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。

```
输入:
nums =
[[1,2],
[3,4]]
r = 1, c = 4
输出:
[[1,2,3,4]]
解释:
行遍历nums的结果是 [1,2,3,4]。新的矩阵是 1 * 4 矩阵，用之前的元素值一行一行填充新矩阵。
```

```
输入:
nums =
[[1,2],
[3,4]]
r = 2, c = 4
输出:
[[1,2],
[3,4]]
解释:
没有办法将 2 * 2 矩阵转化为 2 * 4 矩阵。 所以输出原矩阵。
```

题解

1、mat展开，两个循环——双百

```
class Solution:
    def matrixReshape(self, mat: List[List[int]], r: int, c: int) ->
List[List[int]]:
        x,y=len(mat),len(mat[0])
        if x*y!=r*c:
            return mat
        res=[[0]*c for _ in range(r)]
        num=[]
        for i in mat:
            num.extend(i)
        n=0
        for i in range(r):
            for j in range(c):
                res[i][j]=num[n]
                n+=1

        return res
```

2、numpy包的reshape函数

```
class Solution(object):
    def matrixReshape(self, nums, r, c):
        M, N = len(nums), len(nums[0])
        if M * N != r * c:
            return nums
        import numpy as np
        return np.asarray(nums).reshape((r, c))
```

3、不展开，使用除法找位置

```
def matrixReshape(self, nums, r, c):
    row, col = len(nums), len(nums[0])
    if row * col != r * c:
        return nums
    res = [[None] * c for _ in xrange(r)]
    for i in range(r*c):
        res[i//c][i%c] = nums[i//col][i%col]
    return res
```

572、另一个树的子树

给定两个非空二叉树 s 和 t，检验 s 中是否包含和 t 具有相同结构和节点值的子树。s 的一个子树包括 s 的一个节点和这个节点的所有子孙。s 也可以看做它自身的一棵子树。

> ```
>   3
>  / \
> 4   5
> / \
> 1   2
>
> 4
> / \
> 1   2
> 返回 true，因为 t 与 s 的一个子树拥有相同的结构和节点值。
> ```
>
> ```
>   3
>  / \
> 4   5
> / \
> 1   2
>  /
> 0
>
> 4
> / \
> 1   2
> 返回 false
> ```

题解

1、判断子树

- 判断两个树是否相等的三个条件是与的关系
    1. 当前两个树的根节点值相等；
    2. 并且，s 的左子树和 t 的左子树相等；
    3. 并且，s 的右子树和 t 的右子树相等。
- 而判断 t 是否为 s 的**子树**的三个条件是**或**的关系，即：
    1. 当前两棵树相等；
    2. 或者，t 是 s 的左子树；
    3. 或者，t 是 s 的右子树。

```
class Solution(object):
    def isSubtree(self, s, t):
        """
        :type s: TreeNode
        :type t: TreeNode
        :rtype: bool
        """
        if not s and not t:
            return True
```

```
        if not s or not t:
            return False
        return self.isSameTree(s, t) or self.isSubtree(s.left, t) or
self.isSubtree(s.right, t)

    def isSameTree(self, s, t):
        if not s and not t:
            return True
        if not s or not t:
            return False
        return s.val == t.val and self.isSameTree(s.left, t.left) and
self.isSameTree(s.right, t.right)
```

2、字符串比较

```
class Solution(object):
    def isSubtree(self, s, t):
        def ser(root):

            if not root:    return '#'
            st=' '+str(ser(root.left))+' '+str(ser(root.right))+'
'+str(root.val)+' ' #前后加空格，避免[12][1]的情况误判
            return st

        return ser(t) in ser(s)
```

575、分糖果

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。

> 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。

> 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。

题解

1、set函数之后，对半分

```
class Solution:
    def distributeCandies(self, candyType: List[int]) -> int:
        num=set(candyType)
        if len(num)<=len(candyType)//2:
            return len(num)
        else:
            return len(candyType)//2
```

优化

```
class Solution:
    def distributeCandies(self, candies: List[int]) -> int:
        return min(len(candies) // 2, len(set(candies)))
```

594、最长和谐子序列

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。

> 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]

> 输入：nums = [1,2,3,4]
> 输出：2

题解

1、创造{数字-个数}字典，遍历字典，判断相邻数之和最多的

```
class Solution:
    def findLHS(self, nums: List[int]) -> int:
        dic={}
        for i in nums:
            if i not in dic:
                dic[i]=1
            else:
                dic[i]+=1

        res=0
        for i in dic:
            if i+1 in dic:
                res=max(res,dic[i]+dic[i+1])
            elif i-1 in dic:
                res=max(res,dic[i]+dic[i-1])
            else:
                continue

        return res
```

2、排序+双指针

```python
class Solution:
    def findLHS(self, nums: List[int]) -> int:
        nums.sort()
        left=0
        res=0
        for right in range(len(nums)):
            while nums[right]-nums[left]>1:
                left+=1
            if nums[right]-nums[left]==1:
                res=max(res,right-left+1)
        return res
```

598、范围求和

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。

> 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。

题解

1、其实就是选取，改变的最小行和最小列，相乘是结果

```python
class Solution:
    def maxCount(self, m: int, n: int, ops: List[List[int]]) -> int:
        if not ops:return m*n
        x,y=100000000,1000000000
        for i in range(len(ops)):
            if ops[i][0]==0 or ops[i][1]==0:
                continue
            x=min(x,ops[i][0])
            y=min(y,ops[i][1])
        if x*y==100000000000000000:
            return m*n
        else:
            return x*y
```

优化——感觉没有考虑ops其中一个数为0的情况

```python
class Solution:
    def maxCount(self, m: int, n: int, ops: List[List[int]]) -> int:
        return min([a[0] for a in ops]) * min([a[1] for a in ops]) if ops else m
* n
```

599、两个列表的最小索引总和

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。

> 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”。

> 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)。

题解

1、写的很短，创造很多字典

```python
class Solution:
    def findRestaurant(self, list1: List[str], list2: List[str]) -> List[str]:
        dic_1={}
        dic_2={}
        for i in range(len(list1)):
```

```
            if list1[i] not in dic_1:
                dic_1[list1[i]]=i
        for i in range(len(list2)):
            if list2[i] not in dic_2:
                dic_2[list2[i]]=i

        dic_3={}
        for i in dic_1:
            if i in dic_2:
                dic_3[i]=dic_1[i]+dic_2[i]

        dic_4=sorted(dic_3.items(),key=lambda x:x[1])
        res=[]
        for i in dic_4:
            if not res:
                res.append(i[0])
            else:
                if i[1]==dic_3[res[0]]:
                    res.append(i[0])
                else:
                    break

        return res
```

优化——只是代码简洁，但运行时间更长

```
class Solution:
    def findRestaurant(self, list1: List[str], list2: List[str]) -> List[str]:
        k = 2001
        dic = {}
        if len(list1)<len(list2):
            list1,list2 = list2,list1
        for i in range(len(list1)):
            dic[list1[i]] = i
        for j in range(len(list2)):
            val = dic.get(list2[j])
            if val is not None:
                if val+j < k:
                    res = []
                    k = val+j
                    res.append(list2[j])
                elif val+j == k:
                    res.append(list2[j])
        return res
```

2、字典记录{共同喜欢的商店—索引和}，返回索引和并列最小的商店名

```
class Solution:
    def findRestaurant(self, list1: List[str], list2: List[str]) -> List[str]:
        d = {x: list1.index(x) + list2.index(x) for x in set(list1) &
set(list2)}
        return [x for x in d if d[x] == min(d.values())]
```

605、种花问题

假设有一个很长的花坛，一部分地块种植了花，另一部分却没有。可是，花不能种植在相邻的地块上，它们会争夺水源，两者都会死去。

给你一个整数数组 flowerbed 表示花坛，由若干 0 和 1 组成，其中 0 表示没种植花，1 表示种植了花。另有一个数 n ，能否在不打破种植规则的情况下种入 n 朵花？能则返回 true ，不能则返回 false。

> 输入：flowerbed = [1,0,0,0,1], n = 1
> 输出：true

> 输入：flowerbed = [1,0,0,0,1], n = 2
> 输出：false

题解

1、前后加0，按条件种花

```
class Solution:
    def canPlaceFlowers(self, flowerbed: List[int], n: int) -> bool:
        count=0
        flowerbed=[0]+flowerbed+[0]
        for i in range(1,len(flowerbed)-1):
            if flowerbed[i]==0 and flowerbed[i-1]==0 and flowerbed[i+1]==0:
                flowerbed[i]=1
                count=count+1
            if count>=n:
                return True
        return False
```

2、指针

```
class Solution:
    def canPlaceFlowers(self, flowerbed: List[int], n: int) -> bool:
        flowerbed = flowerbed + [0]
        res = 0
        i,length = 0,len(flowerbed)-1

        while i < length:
            if flowerbed[i]:
                i += 2
            elif flowerbed[i+1]:
                i += 3
            else:
                res += 1
                i += 2

        return res >= n
```

606、根据二叉树创建字符串

你需要采用前序遍历的方式，将一个二叉树转换成一个由括号和整数组成的字符串。

空节点则用一对空括号 "()" 表示。而且你需要省略所有不影响字符串与原始二叉树之间的一对一映射关系的空括号对。

> 输入: 二叉树: [1,2,3,4]
>
> 1
>
> / \
>
> 2 3
>
> /
>
> 4
>
> 输出: "1(2(4))(3)"
>
> 解释: 原本将是“1(2(4)())(3())”，
> 在你省略所有不必要的空括号对之后，
> 它将是“1(2(4))(3)”。

> 输入: 二叉树: [1,2,3,null,4]
>
> 1
>
> / \
>
> 2 3
>
> \
>
> 4
>
> 输出: "1(2()(4))(3)"
>
> 解释: 和第一个示例相似，
> 除了我们不能省略第一个对括号来中断输入和输出之间的一对一映射关系。

题解

1、前序遍历

```
class Solution:
    def tree2str(self, root: TreeNode) -> str:
        def dfs(root):
            if not root.left and root.right:
                return str(root.val) + '()'+'('+dfs(root.right)+')'
            elif not root.right and root.left:
                return str(root.val)+'('+dfs(root.left)+')'
            elif not root.left and not root.right:
                return str(root.val)
            else:
                return
str(root.val)+'('+dfs(root.left)+')'+'('+dfs(root.right)+')'

        return dfs(root)
```

优化：

```python
class Solution:
    def tree2str(self, t: TreeNode) -> str:
        if not t:
            return ''
        left = '('+self.tree2str(t.left)+')' if (t.left or t.right) else ''
        right = '('+self.tree2str(t.right)+')' if t.right else ''
        return str(t.val) +left + right
```