剑指 Offer 03. 数组中重复的数字

找出数组中重复的数字。

在一个长度为 n 的数组 nums 里的所有数字都在 0～n-1 的范围内。数组中某些数字是重复的，但不知道有几个数字重复了，也不知道每个数字重复了几次。请找出数组中任意一个重复的数字。

```
示例 1:

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

```
### set
class Solution:
    def findRepeatNumber(self, nums: List[int]) -> int:
        set1 = set()
        for num in nums:
            if num not in set1:
                set1.add(num)
            else:
                return num
### 排序后 检测左右是否想等

### 一个萝卜一个坑
### 如果遇到num不在index为num的上面 就交换
class Solution:
    def findRepeatNumber(self, nums: List[int]) -> int:
        for index,num in enumerate(nums):
            if nums[num]!=num:
                nums[num],nums[index] = nums[index],nums[num]
            if nums[num] ==num and index!=num:
                return num
```

剑指 Offer 04. 二维数组中的查找

在一个 n * m 的二维数组中，每一行都按照从左到右递增的顺序排序，每一列都按照从上到下递增的顺序排序。请完成一个高效的函数，输入这样的一个二维数组和一个整数，判断数组中是否含有该整数。

```
现有矩阵 matrix 如下：

[
  [1,   4,  7, 11, 15],
  [2,   5,  8, 12, 19],
  [3,   6,  9, 16, 22],
  [10, 13, 14, 17, 24],
  [18, 21, 23, 26, 30]
]
给定 target = 5，返回 true。

给定 target = 20，返回 false。
```

```
#### 二分查找插在这里
class Solution:
    def search(self,nums,target):
        left,right = 0,len(nums)-1
        while right >= left:
            mid = (left +right)//2
            if nums[mid] == target:
                return mid
            elif nums[mid] > target:
                right = mid -1
            else:
                left = mid +1
        return -1
#### 自己写的递归 有点慢?
class Solution:
    def findNumberIn2DArray(self, matrix: List[List[int]], target: int) -> bool:
        if len(matrix)==0 or len(matrix[0])==0:
            return False
        n = len(matrix)
        m = len(matrix[0])
        row = 0
        col = m-1
        if matrix[row][col]>target:
            return self.findNumberIn2DArray(list(map(lambda x:x[:col],matrix)),target)
        elif matrix[row][col]==target:
            return True
        else:
            return self.findNumberIn2DArray(matrix[1:],target)
#### 循环
class Solution:
    def findNumberIn2DArray(self, matrix: List[List[int]], target: int) -> bool:
        if len(matrix)==0:
            return False
        n = len(matrix)
        m = len(matrix[0])
        row = 0
        col = m-1
        while row<n and col >=0:
            if target == matrix[row][col]:
                return True
            elif target>matrix[row][col]:
                row +=1
            else:
                col-=1
        return False
```

5.替换空格

```
#### Python中字符串不可变 所以
### 复杂度都是O N
class Solution:
    def replaceSpace(self, s: str) -> str:
        ans = []
        for _ in s:
            if _!=' ':
                ans.append(_)
            else:
                ans.append('%20')
        return ''.join(ans)
#### 反向思维 如果字符串可变;
#### 先遍历一遍 遇到空格 长度+2
#### i在原字符串最后 j在新字符串最后
#### 倒过来写
#### s[i]==' ' s[j]插入 j-=2
#### s[i]!=' ' s[j] = s[i]
```

7.重建二叉树（前中重建）

输入某二叉树的前序遍历和中序遍历的结果，请重建该二叉树。假设输入的前序遍历和中序遍历的结果中都不含重复的数字。

例如，给出

```
前序遍历 preorder = [3,9,20,15,7]
中序遍历 inorder = [9,3,15,20,7]
返回如下的二叉树:

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

```
### https://leetcode-cn.com/problems/zhong-jian-er-cha-shu-lcof/solution/mian-shi-ti-07-zhong-ji
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def buildTree(self, preorder: List[int], inorder: List[int]) -> TreeNode:
        self.dic = {}
        self.po = preorder
        for a,b in enumerate(inorder):
            self.dic[b] = a
        return self.recur(0,0,len(preorder)-1)
    def recur(self,pre_root,in_left,in_right):
        if in_left > in_right:
            return None
        root = TreeNode(self.po[pre_root])
        ind = self.dic[self.po[pre_root]]
        root.left = self.recur(pre_root+1, in_left, ind-1)
        root.right = self.recur(pre_root+ind - in_left+1,ind+1,in_right)
        return root
```

剑指 Offer 09. 用两个栈实现队列

用两个栈实现一个队列。队列的声明如下，请实现它的两个函数 appendTail 和 deleteHead ，分别完成在队列尾部插入整数和在队列头部删除整数的功能。(若队列中没有元素，deleteHead 操作返回 -1)

```python
class CQueue:

    def __init__(self):
        self.A,self.B = [],[]

    def appendTail(self, value: int) -> None:
        self.A.append(value)


    def deleteHead(self) -> int:
        if self.B:
            return self.B.pop()
        if not self.A:
            return -1
        while self.A:
            self.B.append(self.A.pop())
        return self.B.pop()



# Your CQueue object will be instantiated and called as such:
# obj = CQueue()
# obj.appendTail(value)
# param_2 = obj.deleteHead()
```

剑指 Offer 11. 旋转数组的最小数字

简单版本 无重复

153. 寻找旋转排序数组中的最小值

假设按照升序排序的数组在预先未知的某个点上进行了旋转。例如，数组 [0,1,2,4,5,6,7] 可能变为 [4,5,6,7,0,1,2] 。

请找出其中最小的元素。

示例 1：

输入：nums = [3,4,5,1,2]
输出：1
示例 2：

输入：nums = [4,5,6,7,0,1,2]
输出：0
示例 3：

输入：nums = [1]
输出：1

```
class Solution:
    def findMin(self, nums: List[int]) -> int:
        if len(nums)==1:
            return nums[0]
        left = 0
        right = len(nums)-1
        ### 如何判断为最小值 比前一个数大
        while right >=left:
            mid = (left+right)//2
            if nums[right]>nums[left]:
                return nums[left]
            if mid!=0 and nums[mid] < nums[mid-1]:
                return nums[mid]
            #### 等于只是给还剩两位数的时候
            if nums[mid] >= nums[left]:
                left = mid +1
            else:
                ### right = mid 也行 -1也行是因为 上面已经检测过mid不是最小值
                right = mid - 1
```

原题.

假设按照升序排序的数组在预先未知的某个点上进行了旋转。

(例如，数组 [0,1,2,4,5,6,7] 可能变为 [4,5,6,7,0,1,2])。

请找出其中最小的元素。

注意数组中可能存在重复的元素。

示例 1:

输入: [1,3,5]
输出: 1
示例 2:

输入: [2,2,2,0,1]
输出: 0

```
class Solution:
    def findMin(self, nums: List[int]) -> int:
        if len(nums) ==1:
            return nums[0]
        left = 0
        right = len(nums)-1
        while right>=left:
            mid = (left + right)//2
            if nums[right]>nums[left]:
                return nums[left]
            if mid !=0 and nums[mid]<nums[mid-1]:
                return nums[mid]
            if nums[mid]>nums[left]:
                left = mid +1
            elif nums[mid]<nums[left]:
                right = mid
            else:
                left +=1
            ###处理全是1的情况
        return nums[left-1]
```

剑指 Offer 12. 矩阵中的路径

请设计一个函数，用来判断在一个矩阵中是否存在一条包含某字符串所有字符的路径。路径可以从矩阵中的任意一格开始，每一步可以在矩阵

```
[["a","b","c","e"],
["s","f","c","s"],
["a","d","e","e"]]
```

但矩阵中不包含字符串“abfb”的路径，因为字符串的第一个字符b占据了矩阵中的第一行第二个格子之后，路径不能再次进入这个格子。

示例 1：

输入：board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCCED"
输出：true
示例 2：

输入：board = [["a","b"],["c","d"]], word = "abcd"
输出：false

```
class Solution:
    def exist(self, board: List[List[str]], word: str) -> bool:
        for i in range(len(board)):
            for j in range(len(board[0])):
                if board[i][j] == word[0]:
                    if self.find_exist(board,word,i,j):
                        return True
        return False


    def find_exist(self,board,word,i,j):
        u = 0
        d = len(board)-1
        l = 0
        r = len(board[0])-1
        s_r = i
        s_c = j
        mark = [[0]*len(board[0]) for _ in range(len(board))]
        mark[i][j] = 1
        ###上下左右
        dirs = [[1,0],[-1,0],[0,-1],[0,1]]
        n = 1
        def backtrack(mark,i,j,n):
            if n ==len(word):
                return True
            for dir in dirs:
                tmp = [i+dir[0],j+dir[1]]
                if tmp[0]< u or tmp[0]>d or tmp[1]<l or tmp[1]>r or mark[tmp[0]][tmp[1]]==1:
                    continue
                else:
                    if board[tmp[0]][tmp[1]] == word[n]:
                        mark[tmp[0]][tmp[1]] = 1
                        ####一定要if加判断 不然False也会出来
                        if backtrack(mark,tmp[0],tmp[1],n+1):
                            return True
                        mark[tmp[0]][tmp[1]] = 0
            return False
        return backtrack(mark,i,j,n)
```

剑指 Offer 13. 机器人的运动范围

地上有一个m行n列的方格，从坐标 [0,0] 到坐标 [m-1,n-1] 。一个机器人从坐标 [0, 0] 的格子开始移动，它每次可以向左、右、上、

示例 1：

输入：m = 2, n = 3, k = 1
输出：3
示例 2：

输入：m = 3, n = 1, k = 0
输出：1

```
###是回溯吗？好像不是 好像是BFS
### 这里的回溯不需要 不需要回溯mark的状态矩阵  因为每个点能否到达其上下左右不取决于他的之前的轨迹
class Solution:
    def movingCount(self, m: int, n: int, k: int) -> int:
        if k<0:
            return 0
        u = 0
        d = m - 1
        l = 0
        r = n - 1
        mark = [[0]*n for _ in range(m)]
        mark[0][0] = 1
        ### 只考虑 右下就可以
        dirs = [[1,0],[-1,0],[0,1],[0,-1]]
        def backtrack(i,j,k):
            for dir in dirs:
                tmp = [i+dir[0],j+dir[1]]
                if tmp[0]<u or tmp[0]>d or tmp[1]<l or tmp[1]>r or mark[tmp[0]][tmp[1]]==1 or se
                    continue
                else:
                    mark[tmp[0]][tmp[1]] = 1
                    backtrack(tmp[0],tmp[1],k)
        backtrack(0,0,k)
        ans = 0
        for i in range(m):
            for j in range(n):
                ans += mark[i][j]
        return ans


    def cal_sum(self,i,j):
        def sum1(i):
            sum2 = 0
            for _ in list(str(i)):
                sum2+=int(_)
            return sum2
        return sum1(i) + sum1(j)
```

剑指 Offer 14- I. 剪绳子

```
给你一根长度为 n 的绳子，请把绳子剪成整数长度的 m 段 (m、n都是整数, n>1并且m>1) , 每段绳子的长度记为 k[0],k[1]...k[m-1] 

示例 1:

输入: 2
输出: 1
解释: 2 = 1 + 1, 1 × 1 = 1
示例 2:

输入: 10
输出: 36
解释: 10 = 3 + 3 + 4, 3 × 3 × 4 = 36
```

```python
class Solution:
    def cuttingRope(self, n: int) -> int:
        #### 动态规划求解
        #### dp[i]表示长度为i的绳子切成后最大乘积
        #### dp[0] = 0 dp[1] = 0 必须得切m>1
        #### i被人 分成 j 和 i-j max(j*dp[i-j],j*(i-j))

        dp = [0]*(n+1)
        dp[1] = 0
        k = 2
        while k<=n:
            tmp_max = 0
            for j in range(1,k):
                tmp_max = max(tmp_max,max(j*dp[k-j],j*(k-j)))
            dp[k] = tmp_max
            k+=1
        return dp[k-1]
```

剑指 Offer 15. 二进制中1的个数

- 不是很有意思

请实现一个函数，输入一个整数（以二进制串形式），输出该数二进制表示中 1 的个数。例如，把 9 表示成二进制是 1001，有 2 位是 1

```
示例 1:

输入：00000000000000000000000000001011
输出：3
解释：输入的二进制串 00000000000000000000000000001011 中，共有三位为 '1'。
示例 2:

输入：00000000000000000000000010000000
输出：1
解释：输入的二进制串 00000000000000000000000010000000 中，共有一位为 '1'。
示例 3:

输入：11111111111111111111111111111101
输出：31
解释：输入的二进制串 11111111111111111111111111111101 中，共有 31 位为 '1'。
```

```python
##### &与运算 最后位是1 就是1 不是1就是0
#####>>右移动1位
##### 复杂度是 n的位数
class Solution:
    def hammingWeight(self, n: int) -> int:
        ans = 0
        while n:
            ans += n&1
            n>>=1
        return ans

#### n&n-1
#### 1010 0100 n
#### 1010 0011 n-1
#### 1010 0000 n&(n-1)
#### 复杂度是 n的1的个数
class Solution:
    def hammingWeight(self, n: int) -> int:
        ans = 0
        while n:
            ans +=1
            n = n&(n-1)
        return ans
```

剑指 Offer 18. 删除链表的节点

给定单向链表的头指针和一个要删除的节点的值，定义一个函数删除该节点。

返回删除后的链表的头节点。

注意：此题对比原题有改动

示例 1:

输入: head = [4,5,1,9], val = 5
输出: [4,1,9]
解释: 给定你链表中值为 5 的第二个节点，那么在调用了你的函数之后，该链表应变为 4 -> 1 -> 9.
示例 2:

输入: head = [4,5,1,9], val = 1
输出: [4,5,9]
解释: 给定你链表中值为 1 的第三个节点，那么在调用了你的函数之后，该链表应变为 4 -> 5 -> 9.

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution:
    def deleteNode(self, head: ListNode, val: int) -> ListNode:
        dummy = ListNode(0)
        dummy.next = head
        dummy1 = dummy
        while dummy.next:
            if dummy.next.val == val:
                dummy.next = dummy.next.next
                break
            dummy = dummy.next
        return dummy1.next
```

```python
class Solution:
    def deleteNode(self, head: ListNode, val: int) -> ListNode:
        dummy = ListNode(0)
        dummy.next = head
        if head.val ==val:
            dummy.next = dummy.next.next
            return dummy.next
        while head.next:
            if head.next.val == val:
                head.next = head.next.next
                break
            head = head.next
        return dummy.next
```

剑指 Offer 22. 链表中倒数第k个节点

输入一个链表，输出该链表中倒数第k个节点。为了符合大多数人的习惯，本题从1开始计数，即链表的尾节点是倒数第1个节点。例如，一个

示例：

给定一个链表: `1->2->3->4->5`, 和 `k = 2`.

返回链表 `4->5`.

```
class Solution:
    def getKthFromEnd(self, head: ListNode, k: int) -> ListNode:
        fast = head
        slow = head
        for _ in range(k-1):
            fast = fast.next
        while fast.next:
            fast = fast.next
            slow = slow.next
        return slow
```

28.对称的二叉树

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

class Solution:
    def isSymmetric(self, root: TreeNode) -> bool:
        if root is None:
            return True
        return self.is_sym(root.left,root.right)

    def is_sym(self,left,right):
        if left is None and right is None:
            return True
        if left is None or right is None:
            return False
        if left.val != right.val:
            return False
        return self.is_sym(left.left,right.right) and self.is_sym(left.right,right.left)
```

32.1.从上到下打印二叉树

```
class Solution:
    def levelOrder(self, root: TreeNode) -> List[int]:
        if root is None:
            return []
        res = []
        queue = [root]
        while queue:
            size_level = len(queue)
            for _ in range(size_level):
                tmp = queue.pop(0)
                res.append(tmp.val)
                if tmp.left:
                    queue.append(tmp.left)
                if tmp.right:
                    queue.append(tmp.right)
        return res
```

32.2.从上到下打印二叉树

```
class Solution:
    def levelOrder(self, root: TreeNode) -> List[int]:
        if root is None:
            return []
        res = []
        queue = [root]
        while queue:
            res_level = []
            size_level = len(queue)
            for _ in range(size_level):
                tmp = queue.pop(0)
                res_level.append(tmp.val)
                if tmp.left:
                    queue.append(tmp.left)
                if tmp.right:
                    queue.append(tmp.right)
            res.append(res_level)
        return res
```

32.3从上到下打印二叉树

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

class Solution:
    def levelOrder(self, root: TreeNode) -> List[int]:
        if root is None:
            return []
        k = 0
        res = []
        queue = [root]
        while queue:
            k+=1

            res_level = []
            size_level = len(queue)
            for _ in range(size_level):
                tmp = queue.pop(0)
                res_level.append(tmp.val)
                if tmp.left:
                    queue.append(tmp.left)
                if tmp.right:
                    queue.append(tmp.right)
            if k%2 ==0:
                res.append(res_level[::-1])
            else:
                res.append(res_level)
        return res
```

34.二叉树中和为某一值的路径

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

class Solution:
    def pathSum(self, root: TreeNode, sum: int) -> List[List[int]]:
        final_list = []
        def backtrack(path,root,sum1):
            if root is None:
                return
            if root.left is None and root.right is None and root.val == sum1:
                final_list.append(path+[root.val])
                return
            for node in [root.left,root.right]:
                if node is None:
                    continue
                backtrack(path+[root.val], node, sum1-root.val)
        backtrack([], root, sum)
        return final_list

### 看了题解的改变
### 主要是那个for一定只循环两次 可以写成2个backtrack
### 提前改变了path然后再变回来
class Solution:
    def pathSum(self, root: TreeNode, sum: int) -> List[List[int]]:
        final_list = []
        def backtrace(path,root,sum):
            if root is None:
                return
            path.append(root.val)
            sum -= root.val
            if not root.left and not root.right and sum == 0:
                final_list.append(path[:])
            backtrace(path, root.left, sum)
            backtrace(path, root.right, sum)
            path.pop()
        backtrace([], root, sum)
        return final_list

#### 这题的DFS解法 没咋懂 回头再看下
https://leetcode-cn.com/problems/er-cha-shu-zhong-he-wei-mou-yi-zhi-de-lu-jing-lcof/solution/sar
```

55.二叉树的深度

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

class Solution:
    def maxDepth(self, root: TreeNode) -> int:
        if root is None:
            return 0
        left_depth = self.maxDepth(root.left)
        right_depth = self.maxDepth(root.right)
        return max(left_depth,right_depth)+1

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

class Solution:
    def maxDepth(self, root: TreeNode) -> int:
        if root is None:
            return 0
        level = 0
        queue = [root]
        while queue:
            size_level = len(queue)
            for _ in range(size_level):
                tmp = queue.pop(0)
                if tmp.left:
                    queue.append(tmp.left)
                if tmp.right:
                    queue.append(tmp.right)
            level+=1
        return level
```

55.平衡二叉树

```python
### 第一种调用上面函数
### 这种方法应该想到 为啥想不到呢
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
# 有重复 NlogN
class Solution:
    def isBalanced(self, root: TreeNode) -> bool:
        if root is None:
            return True
        left_depth  = self.maxDepth(root.left)
        right_depth = self.maxDepth(root.right)
        if abs(left_depth - right_depth) >1:
            return False
        return self.isBalanced(root.left) and self.isBalanced(root.right)
    def maxDepth(self, root: TreeNode) -> int:
        if root is None:
            return 0
        left_depth = self.maxDepth(root.left)
        right_depth = self.maxDepth(root.right)
        return max(left_depth,right_depth)+1

# 不重复
class Solution:
    def isBalanced(self, root: TreeNode) -> bool:
        return self.maxDepth(root) >=0
    def maxDepth(self,root):
        if root is None:
            return 0
        left = self.maxDepth(root.left)
        right = self.maxDepth(root.right)
        if left >= 0 and right>=0 and abs(left-right)<=1:
            return max(left,right)+1
        else:
            return -1
```

68.二叉树的最近公共祖先

```
#### 不是特别明白
### https://leetcode-cn.com/problems/er-cha-shu-de-zui-jin-gong-gong-zu-xian-lcof/solution/mian-
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def lowestCommonAncestor(self, root: TreeNode, p: TreeNode, q: TreeNode) -> TreeNode:
        if root is None or root ==p or root == q:
            return root
        left = self.lowestCommonAncestor(root.left, p, q)
        right = self.lowestCommonAncestor(root.right, p, q)
        if not left and not right:
            return
        if not left:
            return right
        if not right:
            return left
        return root
```