每道题做完之后小小的优化一下，运行时间就可以大大降低。

# 取模：10\*\*9+7

为什么要模1000000007。参考https://www.liuchuo.net/archives/645

大数相乘，大数的排列组合等为什么要取模？ 为了避免溢出，计算更多的情况。

1000000007是一个质数（素数），对质数取余能最大程度避免结果冲突/重复

int32位的最大值为2147483647，所以对于int32位来说1000000007足够大。

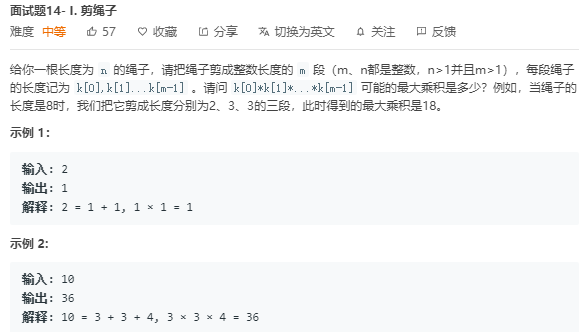
int64位的最大值为2^63-1，用最大值模1000000007的结果求平方，不会在int64中溢出。

所以在大数相乘问题中，因为(a∗b)%c=((a%c)∗(b%c))%c，所以相乘时两边都对1000000007取模，再保存在int64里面不会溢出。

取模之后能够计算更多的情况。题目的测试答案与取模后的结果一致。

总结一下，这道题要模1000000007的根本原因是标准答案模了1000000007。不过大数情况下为了防止溢出，模1000000007是通用做法，原因见第一点。

# 面试题14- I. 剪绳子





## 递归效率太低：

class Solution(object):

def cuttingRope(self, number):

"""

:type n: int

:rtype: int

"""

def fun(number,i):

if number<0:

return -1;

if i==1:

return number;

return max(fun(number-i,i)\*i,fun(number,i-1))

if number<=2:

return 1

if number==3:

return 2

return fun(number,number-1)

## 动态规划：

class Solution(object):

def cuttingRope(self, n):

"""

:type n: int

:rtype: int

"""

dp=[0]\*(n+1)

if n<=2:

return 1

if n==3:

return 2

dp[1]=1;dp[2]=2;dp[3]=3;

for i in range(n+1):

ans=0

for j in range(i):

ans=max(ans,dp[j]\*(i-j),dp[i])

dp[i]=ans

return dp[n]

# 面试题03. 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。

2 <= n <= 100000

我的解答：

辅助列表，遍历时间太长。

## 时间/空间需求思维：

它考察的是程序员的沟通能力，先问面试官要时间/空间需求！！！

1. 只是时间优先就用字典，（哈希）时间O(n)，空间O（n）

把列表换成dict，时间复杂度立马下降。因为列表的查找操作 in 耗时。Dict.get(key) 返回value或none或指定返回值。



其他：哈希冲突到了某一个程度，链表会转为红黑树，本质还是冲突原因导致效率低。Java的map是用红黑树实现的，插入删除都是logn。

1. 还有空间要求，就用原地排序数组，时间O(nlogn)，空间O(1)
2. 时间空间综合考虑：基于数字范围都在0~n-1，可以用鸽巢原理，利用索引与数字的关系，时间O(n)，空间O(1)，修改了原数据。

int findRepeatNumber(vector<int>& nums) {

for(int i=0;i<nums.size();i++){

while(nums[i]!=i){

if(nums[nums[i]] == nums[i]) return nums[i];

int tmp = nums[i];

nums[i] = nums[tmp];

nums[tmp] = tmp;

}

}

return -1;

}

1. 如果面试官要求空间O(1)并且不能修改原数组，还得写成二分法，时间O（nlogn）

二分法的实现除了递归，还可以是while

此题题面为：长度为 n 的数组 nums 里的所有数字都在 0～n-1 的范围内不能用二分法！！！

长度n+1, 1~n代表必定有一个数字出现两次，所以你用二分有个区间的数字在原数组中出现次数是>这个区间范围的

// public int findRepeatNumber(int[] nums) {

// //统计nums中元素位于0到m的数量，如果数量大于这个值，那么重复的元素肯定是位于0到m的

// int min = 0 ;

// int max = nums.length-1;

// while(min<max){

// int mid = (max+min)>>1;

// int count = countRange(nums,min,mid);

// if(count > mid-min+1) {

// max = mid;

// }else{

// min = mid+1;

// }

// }

// //最后min=max，最后必然停在这里

// return min;

// }

// //统计范围内元素数量,时间O(n)

// private int countRange(int[] nums,int min,int max){

// int count = 0 ;

// for(int num:nums){

// if(num>=min && num<=max){

// count++;

// }

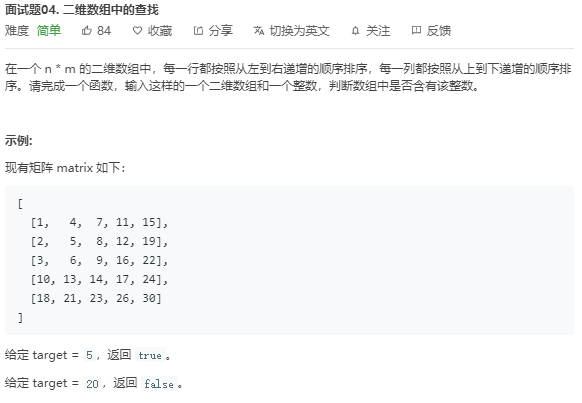
// }

// return count;

// }

}

# 面试题04. 二维数组中的查找



1.常规做法就是剪枝式遍历：时间复杂度高，但运行时间还可以。

1.1数据量大时，可以加入二分搜索。

2.一种巧思：O(m+n)

站在右上角看。这个矩阵其实就像是一个Binary Search Tree。

从右上角开始走，利用这个顺序关系可以在O(m+n)的复杂度下解决这个题：

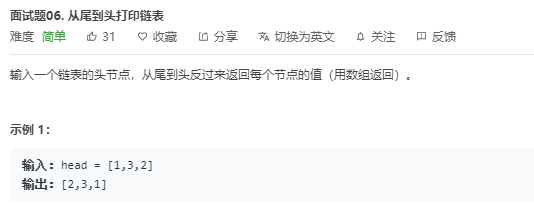
如果当前位置元素比target小，则row++

如果当前位置元素比target大，则col--

如果相等，返回true

如果越界了还没找到，说明不存在，返回false

# 面试题06. 从尾到头打印链表



1. 递归

class Solution(object):

def reversePrint(self, head):

# """

# :type head: ListNode

# :rtype: List[int]

# """

return self.reversePrint(head.next)+[head.val] if head else []

代码比较简洁但是运行时间很长。

1. 辅助栈或数组逆序输出

# 面试题07. 重建二叉树



递归解析：

递推参数： 前序遍历中根节点的索引pre\_root、中序遍历左边界in\_left、中序遍历右边界in\_right。

终止条件： 当 in\_left > in\_right ，子树中序遍历为空，说明已经越过叶子节点，此时返回 nullnull 。

递推工作：

建立根节点root： 值为前序遍历中索引为pre\_root的节点值。

搜索根节点root在中序遍历的索引i： 为了提升搜索效率，本题解使用哈希表 dic 预存储中序遍历的值与索引的映射关系，每次搜索的时间复杂度为 O(1)。

构建根节点root的左子树和右子树： 通过调用 recur() 方法开启下一层递归。

左子树： 根节点索引为 pre\_root + 1 ，中序遍历的左右边界分别为 in\_left 和 i - 1。

右子树： 根节点索引为 i - in\_left + pre\_root + 1（即：根节点索引 + 左子树长度 + 1），中序遍历的左右边界分别为 i + 1 和 in\_right。

返回值： 返回 root，含义是当前递归层级建立的根节点 root 为上一递归层级的根节点的左或右子节点。

链接：https://leetcode-cn.com/problems/zhong-jian-er-cha-shu-lcof/solution/mian-shi-ti-07-zhong-jian-er-cha-shu-di-gui-fa-qin/

class Solution:

def buildTree(self, preorder: List[int], inorder: List[int]) -> TreeNode:

self.dic, self.po = {}, preorder

for i in range(len(inorder)):

self.dic[inorder[i]] = i

return self.recur(0, 0, len(inorder) - 1)

def recur(self, pre\_root, in\_left, in\_right):

if in\_left > in\_right: return # 终止条件：中序遍历为空

root = TreeNode(preorder[pre\_root]) # 建立当前子树的根节点

i = self.dic[preorder[pre\_root]] # 搜索根节点在中序遍历中的索引，从而可对根节点、左子树、右子树完成划分。

root.left = self.recur(pre\_root + 1, in\_left, i - 1) # 开启左子树的下层递归

root.right = self.recur(i - in\_left + pre\_root + 1, i + 1, in\_right) # 开启右子树的下层递归

return root # 返回根节点，作为上层递归的左（右）子节点

我的写法：

# Definition for a binary tree node.

# class TreeNode(object):

# def \_\_init\_\_(self, x):

# self.val = x

# self.left = None

# self.right = None

class Solution(object):

def buildTree(self, preorder, inorder):

"""

:type preorder: List[int]

:type inorder: List[int]

:rtype: TreeNode

"""

self.dic={}

def recur(pre\_root,in\_left,in\_right):

if in\_left>in\_right:

return

i=self.dic[preorder[pre\_root]]

root=TreeNode(preorder[pre\_root])

root.left=recur(pre\_root+1,in\_left,i-1)

root.right=recur(pre\_root+1+i-in\_left,i+1,in\_right)

return root

for i in range(len(inorder)):

self.dic[inorder[i]]=i

return recur(0,0,len(inorder)-1)

# 面试题09. 用两个栈实现队列

用python写当然是很简单。

题目的要点思想是双栈。

官方的方法在两个栈之间的操作有些频繁，其实是可以省去很多步骤的，主要的思想是：

stack\_in只负责进入

stack\_out只负责取出

只有stack\_out为空时才把stack\_in的所有元素倾倒进stack\_out中，这样顺序就不会乱了

class CQueue:

def \_\_init\_\_(self):

self.stack\_in = []

self.stack\_out = []

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

self.stack\_in.append(value)

def deleteHead(self) -> int:

if not self.stack\_out:

if not self.stack\_in: # 都为空

return -1

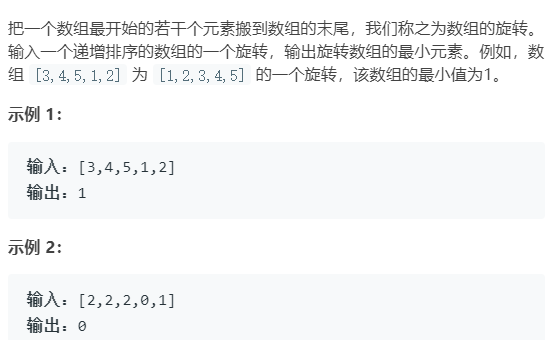
else: # 把in栈中的东西全部倒入out栈中

while self.stack\_in:

self.stack\_out.append(self.stack\_in.pop())

return self.stack\_out.pop()

# 面试题11. 旋转数组的最小数字



要求时间复杂度O（log n）

## 特殊的二分：

class Solution(object):

def minArray(self, numbers):

"""

:type numbers: List[int]

:rtype: int

"""

# for i in range(1,len(numbers)):

# if numbers[i]<numbers[i-1]:

# return numbers[i]

# return numbers[0]

l=0

r=len(numbers)-1

while l<r:

mid=(l+r)/2

if numbers[mid]>numbers[r]:# 头部在右边

l=mid+1

elif numbers[mid]<numbers[r]:# 头部在左边(含自身)

r=mid

else:

r-=1 #判断不出头部在哪边，但不会损失头部

return numbers[r]

# 面试题17. 打印从1到最大的n位数

大数越界情况下的打印。需要解决以下三个问题：

1. 表示大数的变量类型：

无论是 short / int / long ... 任意变量类型，数字的取值范围都是有限的。因此，大数的表示应用**字符串 String 类型。**

2. 生成数字的字符串集：

使用 int 类型时，每轮可通过 +1+1 生成下个数字，而此方法无法应用至 String 类型。并且，String 类型的数字的进位操作效率较低，例如 "9999" 至 "10000" 需要从个位到千位循环判断，进位 4 次。

观察可知，生成的列表实际上是 **n 位0 - 9的全排列，因此可避开进位操作，通过递归生成数字的String列表。**

3. 递归生成全排列：

基于分治算法的思想，先固定高位，向低位递归。

class Solution:

def printNumbers(self, n: int) -> [int]:

def dfs(x):

if x == n: # 终止条件：已固定完所有位

res.append(''.join(num)) # 拼接 num 并添加至 res 尾部

return

for i in range(10): # 遍历 0 - 9

num[x] = str(i) # 固定第 x 位为 i

dfs(x + 1) # 开启固定第 x + 1 位

num = ['0'] \* n # 起始数字定义为 n 个 0 组成的字符列表

res = [] # 数字字符串列表

dfs(0) # 开启全排列递归

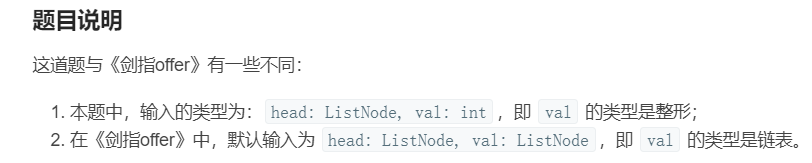
return ','.join(res) # 拼接所有数字字符串，使用逗号隔开，并返回

当前的生成方法仍有以下问题：

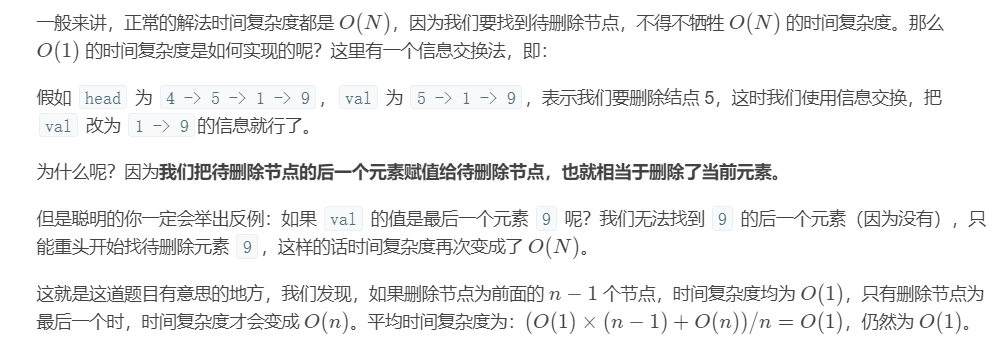
删除高位多余的0；s.lstrip(‘0’)

列表从1开始；if s!=‘0’：

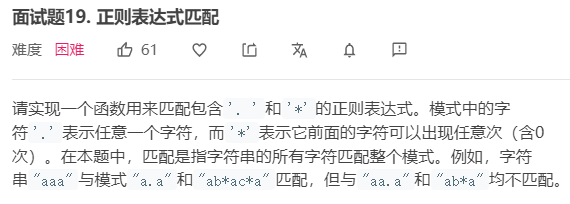
# 面试题18. 删除链表的节点



val: ListNode 解法：信息交换法



# 面试题19. 正则表达式匹配



Python正则回顾：

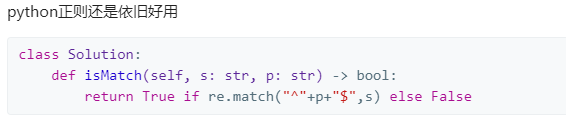
# 符号的正则[\+\-]?

# 数字正则(\d+\.\d+|\.\d+|\d+\.|\d+)对应0.0，.0，0.，0

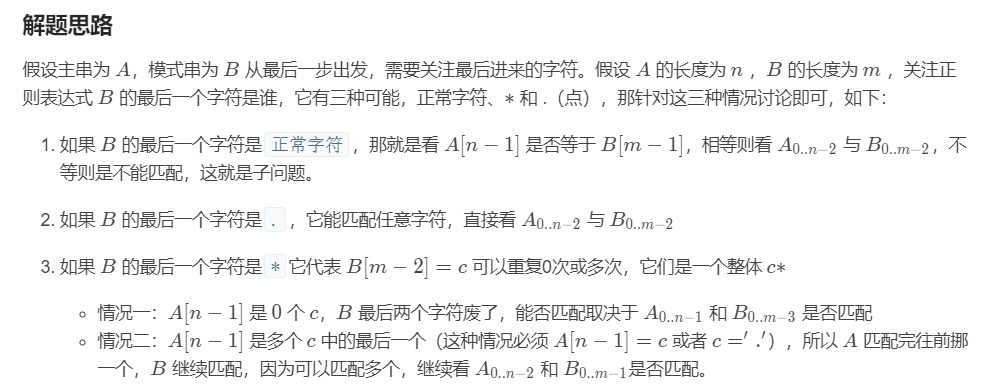
# e指数 (e[\+\-]?\d+)?

rs1=re.findall("^[\-\+]?(\d+\.\d+|\.\d+|\d+\.|\d+)(\e[\-\+]?\d+)?$",s.strip())

^匹配开头 $匹配结尾



递归思路：



class Solution {

public boolean isMatch(String A, String B) {

// 如果字符串长度为0，需要检测下正则串

if (A.length() == 0) {

// 如果正则串长度为奇数，必定不匹配，比如 "."、"ab\*",必须是 a\*b\*这种形式，\*在奇数位上

if (B.length() % 2 != 0) return false;

int i = 1;

while (i < B.length()) {

if (B.charAt(i) != '\*') return false;

i += 2;

}

return true;

}

// 如果字符串长度不为0，但是正则串没了，return false

if (B.length() == 0) return false;

// c1 和 c2 分别是两个串的当前位，c3是正则串当前位的后一位，如果存在的话，就更新一下

char c1 = A.charAt(0), c2 = B.charAt(0), c3 = 'a';

if (B.length() > 1) {

c3 = B.charAt(1);

}

// 和dp一样，后一位分为是 '\*' 和不是 '\*' 两种情况

if (c3 != '\*') {

// 如果该位字符一样，或是正则串该位是 '.',也就是能匹配任意字符，就可以往后走

if (c1 == c2 || c2 == '.') {

return isMatch(A.substring(1), B.substring(1));

} else {

// 否则不匹配

return false;

}

} else {

// 如果该位字符一样，或是正则串该位是 '.'，和dp一样，有看和不看两种情况

if (c1 == c2 || c2 == '.') {

return isMatch(A.substring(1), B) || isMatch(A, B.substring(2));

} else {

// 不一样，那么正则串这两位就废了，直接往后走

return isMatch(A, B.substring(2));

}

}

}

}

*作者：jerry\_nju*

*链接：https://leetcode-cn.com/problems/zheng-ze-biao-da-shi-pi-pei-lcof/solution/zhu-xing-xiang-xi-jiang-jie-you-qian-ru-shen-by-je/*

# 面试题26：树的子结构

**菜鸡的代码（我）：80ms**

class Solution(object):

def isSubStructure(self, A, B):

if B==None: return False

l=[] # 获得所有需要比较的起点

def head(A,B,l):

if A==None: return

if A.val==B.val:

l.append(A)

if A.left!=None: head(A.left,B,l)

if A.right!=None: head(A.right,B,l)

head(A,B,l)

def isvalidhead(A,B): #判断从A的这个节点出发是否能配对

if B==None: return False

if A==None:

if B!=None:

return False

else:

return True

if A.val!=B.val:

return False

else:

if B.left!=None and B.right!=None:

return (isvalidhead(A.left,B.left) and isvalidhead(A.right,B.right))

elif B.left!=None:

return isvalidhead(A.left,B.left)

elif B.right!=None:

return isvalidhead(A.right,B.right)

else:

return True

for i in l:

if isvalidhead(i,B):

return True

return False

# 我的代码长是因为有很多条件判断，去掉这些条件判断代码是很简洁，但运行时间变长了，因为递归深度增加了。

**大佬的代码：120ms**

class Solution:

def isSubStructure(self, A: TreeNode, B: TreeNode) -> bool:

def recur(A, B):

if not B: return True

if not A or A.val != B.val: return False

return recur(A.left, B.left) and recur(A.right, B.right)

return bool(A and B) and (recur(A, B) or self.isSubStructure(A.left, B) or self.isSubStructure(A.right, B))

**大佬的代码：**

class Solution {

public boolean isSubStructure(TreeNode A, TreeNode B) {

if(A== null || B == null)return false;

return fun(A,B) || isSubStructure(A.left,B) || isSubStructure(A.right,B);

}

public boolean fun(TreeNode a,TreeNode b){

if(b == null) return true;

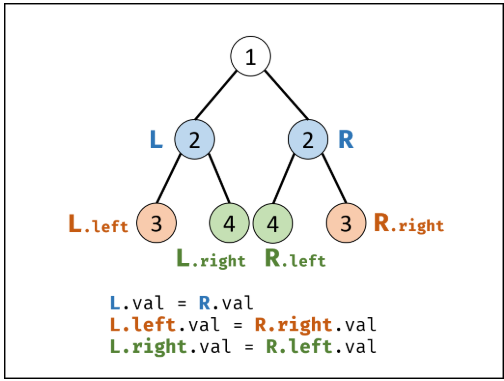
if(a == null) return false;

return (a.val == b.val) && fun(a.left,b.left) && fun(a.right,b.right);

}

}

# 面试题28. 对称的二叉树



**比较的是L与R**

**不是根节点的左节点与右节点**

class Solution:

def isSymmetric(self, root: TreeNode) -> bool:

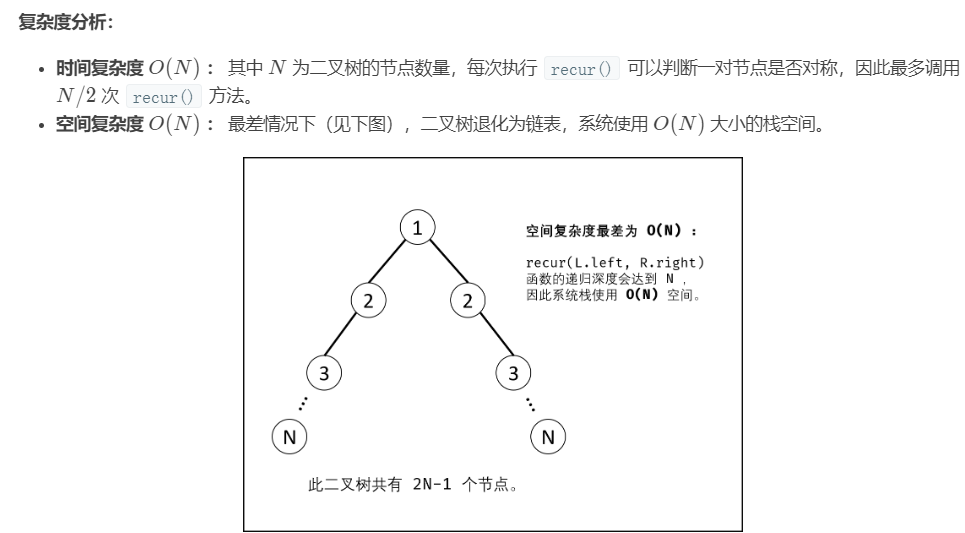
def recur(L, R):

if not L and not R: return True

if not L or not R or L.val != R.val: return False

return recur(L.left, R.right) and recur(L.right, R.left)

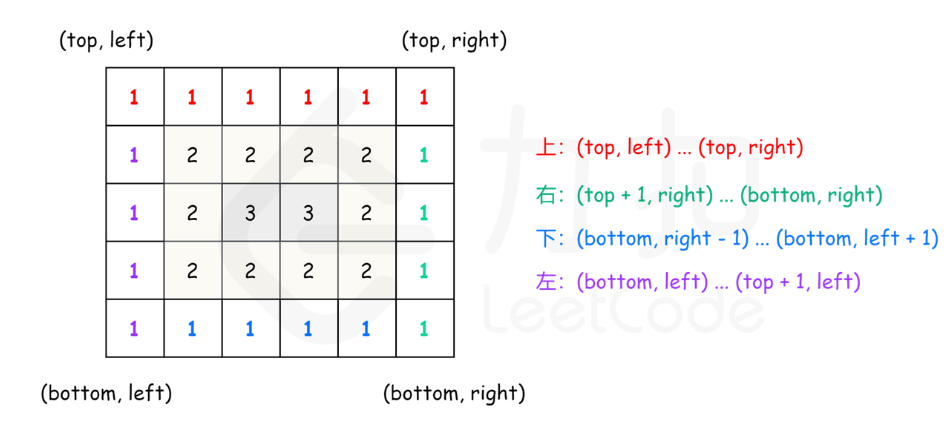
return recur(root.left, root.right) if root else True



# 面试题29. 顺时针打印矩阵

自己写的思路太乱了。

第二次碰到这个题竟然没通过。



尤其注意边界条件！

class Solution:

def spiralOrder(self, matrix: List[List[int]]) -> List[int]:

if not matrix or not matrix[0]:

return list()

rows, columns = len(matrix), len(matrix[0])

order = list()

left, right, top, bottom = 0, columns - 1, 0, rows - 1

while left <= right and top <= bottom:

for column in range(left, right + 1):

order.append(matrix[top][column])

for row in range(top + 1, bottom + 1):

order.append(matrix[row][right])

if left < right and top < bottom:

for column in range(right - 1, left, -1):

order.append(matrix[bottom][column])

for row in range(bottom, top, -1):

order.append(matrix[row][left])

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

return order