牛客网《剑指Offer》66题 题解

1. 字符串的排列

> 输入一个字符串,按字典序打印出该字符串中字符的所有排列。例如输入字符串abc,则打印出由字符a,b,c所能排列出来的所有字符串abc,acb,bac,bca,cab和cba。
>
> 输入描述:
>
> 输入一个字符串,长度不超过9(可能有字符重复),字符只包括大小写字母。

dfs， 但是牛客网有些尴尬，list的顺序还得保障，所以结果得sort一下。为了防止重复，用了set。

```
class Solution:
    def Permutation(self, ss):
        if not ss: return []
        res = set()

        def dfs(res, ss, s):
            if not ss: res.add(s)
            for i in range(len(ss)):
                s = s + ss[i]
                dfs(res, ss[:i] + ss[i + 1:], s)
                s = s[:-1]

        dfs(res, ss, '')
        return sorted(list(res))
```

当然，这只是简单得写法。高级得用swap实现，详情见STL中得permutation经典算法。

2. 链表中倒数第k个结点

> 输入一个链表，输出该链表中倒数第k个结点。

先求链表长度n，然后输出第n-k节点。注意k>n的情况。

```
# -*- coding:utf-8 -*-
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def FindKthToTail(self, head, k):
        # write code here
        n = 0
        p = head
        while p:
            n += 1
            p = p.next

        if k > n: return None
```

```
        t = n - k
        p = head
        for _ in range(t):
            p = p.next
        return p
```

3. 跳台阶

> 一只青蛙一次可以跳上1级台阶，也可以跳上2级。求该青蛙跳上一个n级的台阶总共有多少种跳法。

$f(n)=f(n-1)+f(n-2), f(0)=f(1)=1$，斐波那契数列的变体。

```
class Solution:
    def jumpFloor(self, number):
        # write code here
        a, b = 1, 1
        for i in range(number):
            a, b = b, a + b
        return a
```

4. 变态跳台阶

> 一只青蛙一次可以跳上1级台阶，也可以跳上2级……它也可以跳上n级。求该青蛙跳上一个n级的台阶总共有多少种跳法。

数学推导：$f(n)=f(n-1)+f(n-2)+...+f(1)$，$f(n-1)=f(n-2)+f(n-3)+...+f(1)$

$f(1)=1$，因而有$f(n)=2f(n-1)=2^{n-1}$。

```
class Solution:
    def jumpFloorII(self, number):
        # write code here
        return 2 ** (number-1)
```

5. 不用加减乘除做加法

> 写一个函数，求两个整数之和，要求在函数体内不得使用+、-、*、/四则运算符号。

也就是用二进制的加法，二进制位相加用异或，进位用与并左移一位。

```cpp
class Solution {
public:
    int Add(int num1, int num2)
    {
        int tmp;
         while(num2 != 0){
            tmp = num1 ^ num2;
            num2 = (num1 & num2) << 1;
            num1 = tmp;
        }
        return num1;
    }
};
```

上述的实现有些技巧，主要利用c++整数逸出会变0解决$-1+1=0$的情况，而python则没有逸出的概念，所以我们要与上0xffffffff。

```python
class Solution(object):
    def getSum(self, a, b):
        """
        :type a: int
        :type b: int
        :rtype: int
        """
        # 32 bits integer max
        MAX = 0x7FFFFFFF
        # 32 bits interger min
        MIN = 0x80000000
        # mask to get last 32 bits
        mask = 0xFFFFFFFF
        while b != 0:
            # ^ get different bits and & gets double 1s, << moves carry
            a, b = (a ^ b) & mask, ((a & b) << 1) & mask
        # if a is negative, get a's 32 bits complement positive first
        # then get 32-bit positive's Python complement negative
        return a if a <= MAX else ~(a ^ mask)
```

6. 替换空格

> 请实现一个函数，将一个字符串中的空格替换成“%20”。例如，当字符串为We Are Happy.则经过替换之后的字符串为We%20Are%20Happy。

```python
class Solution:
    # s 源字符串
    def replaceSpace(self, s):
        # write code here
        import re
        s = re.sub(' ', '%20', s)
        return s
```

7. 顺序针打印矩阵

> 输入一个矩阵，按照从外向里以顺时针的顺序依次打印出每一个数字，例如，如果输入如下矩阵： 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 则依次打印出数字1,2,3,4,8,12,16,15,14,13,9,5,6,7,11,10.

```
class Solution:
    # matrix类型为二维列表，需要返回列表
    def printMatrix(self, matrix):
        # write code here
        return matrix and list(matrix.pop(0)) + self.printMatrix(list(zip(*matrix))[::-1])
```

8. 二维数组中的查找

> 在一个二维数组中，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。

类似二叉搜索树，从右上角开始依次左、下遍历，或者从左下角开始依次上、右遍历。

```
class Solution:
    # array 二维列表
    def Find(self, target, array):
        # write code here
        m, n = len(array), len(array[0])
        i, j = 0, n - 1
        while i < m and j >= 0:
            if array[i][j] == target: return True
            elif array[i][j] < target: i += 1
            else: j -= 1
        return False
```

9. 包含min函数的栈

> 定义栈的数据结构，请在该类型中实现一个能够得到栈最小元素的min函数。

同时维护一个最小栈。

```
class Solution:
    def __init__(self):
        self.stack = []
        self.minstack = []

    def push(self, node):
        # write code here
        self.stack.append(node)
        if not self.minstack:
            self.minstack.append(node)
        elif node < self.minstack[-1]:
            self.minstack.append(node)
        else:
```

```
            self.minstack.append(self.minstack[-1])

    def pop(self):
        # write code here
        self.stack.pop()
        self.minstack.pop()

    def top(self):
        # write code here
        return self.stack[-1]

    def min(self):
        # write code here
        return self.minstack[-1]
```

10. 重建二叉树

> 输入某二叉树的前序遍历和中序遍历的结果，请重建出该二叉树。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。例如输入前序遍历序列{1,2,4,7,3,5,6,8}和中序遍历序列{4,7,2,1,5,3,8,6}，则重建二叉树并返回。

```
class Solution:
    def reConstructBinaryTree(self, preorder, inorder):
        """
        :type preorder: List[int]
        :type inorder: List[int]
        :rtype: TreeNode
        """
        if inorder:
            id = inorder.index(preorder.pop(0))
            root = TreeNode(inorder[id])
            root.left = self.reConstructBinaryTree(preorder, inorder[:id])
            root.right = self.reConstructBinaryTree(preorder, inorder[id+1:])
            return root
```

11. 用两个栈实现队列

> 用两个栈来实现一个队列，完成队列的Push和Pop操作。 队列中的元素为int类型。

一个in栈，一个out栈，in栈用于直接push，out栈用于逆向pop in 栈中的元素。

```python
class Solution:
    def __init__(self):
        self.instack = []
        self.outstack = []

    def push(self, node):
        self.instack.append(node)

    def pop(self):
        if not self.outstack:
            while self.instack:
                self.outstack.append(self.instack.pop())
        return self.outstack.pop()
```

12. 旋转数组的最小数字

> 把一个数组最开始的若干个元素搬到数组的末尾，我们称之为数组的旋转。 输入一个非递减排序的数组的一个旋转，输出旋转数组的最小元素。 例如数组{3,4,5,1,2}为{1,2,3,4,5}的一个旋转，该数组的最小值为1。 NOTE：给出的所有元素都大于0，若数组大小为0，请返回0。

二分查找的变体，采用左闭右开的写法，维持l是最小值。

```python
class Solution:
    def minNumberInRotateArray(self, rotateArray):
        if not rotateArray: return 0
        l, h = 0, len(rotateArray) - 1
        while l < h:
            m = (l + h) // 2
            if rotateArray[m] > rotateArray[h]:
                l = m + 1
            else:
                h = m
        return rotateArray[l]
```

13. 斐波那契数列

> 大家都知道斐波那契数列，现在要求输入一个整数n，请你输出斐波那契数列的第n项。
>
> n<=39

用a, b 两个变量记录前后两个斐波那契数，c记录第几个。(注意n=0, 0)

```python
class Solution:
    def Fibonacci(self, n):
        a, b = 0, 1
        c = 0
        while c < n:
            b, a, c = a, a + b, c + 1
        return a
```

14. 矩阵覆盖

> 我们可以用21的小矩形横着或者竖着去覆盖更大的矩形。请问用n个21的小矩形无重叠地覆盖一个2*n的大矩形，总共有多少种方法？

还是斐波那契的变体形式，不想写递推了，直接递归实现。不过python会超时，所以改用java拉。

第一步：如果选择竖方向填充，则规模缩小为number-1填充

第一步：如果选择横方向填充，则第二排只能横向填充，规模缩小为number-2填充

所以，综上分析，递归式为
rectCover(number)=rectCover(number-1)+rectCover(number-2);
当然边界条件要设定n=1、2、0。

```
public class Solution {
    public int RectCover(int target) {
        if(target==0)
            return 0;
        else if(target==1)
            return 1;
        else if(target==2)
            return 2;
        else
            return RectCover(target-1)+RectCover(target-2);
    }
}
```

15. 二进制中1的个数

> 输入一个整数，输出该数二进制表示中1的个数。其中负数用补码表示。

先求出二进制，然后不同向右移位，统计最低位为1的数量。

```
class Solution:
    def NumberOf1(self, n):
        return sum([(n >> i & 1) for i in range(0, 32)])
```

16. 数值的整数次方

> 给定一个double类型的浮点数base和int类型的整数exponent。求base的exponent次方。

```python
class Solution:
    def Power(self, base, exponent):
        return base ** exponent
```

当然有高深的解法，也就是==>快速幂.(https://www.cnblogs.com/CXCXCXC/p/4641812.html)

```python
class Solution:
    def Power(self, base, exponent):
        ans = 1
        while exponent:
            if exponent & 1: ans *= base
            base *= base
            exponent >>= 1
        return ans
```

这种再牛客网上提交会超时，毕竟python，时间限制的严。但是这确实是O(log n)的解法。

主要思想是分解exponent，如base=a, exponent=11, $\mathbf{11 = 2^3 + 2^1 + 2^0 = 1011}$，也就是说

$$a^{11} = a^{2^0+2^1+2^3} = a^{2^0} * a^{2^1} * a^{2^3}$$

也就是不停判断exponent的最低一位，a则不停翻倍，实现a=>a^2=>a^3，如果b的最低位是1就乘上去呗。

17. 调整数组顺序使奇数位于偶数前面

> 输入一个整数数组，实现一个函数来调整该数组中数字的顺序，使得所有的奇数位于数组的前半部分，所有的偶数位于位于数组的后半部分，并保证奇数和奇数，偶数和偶数之间的相对位置不变。

类似冒泡算法，前偶后奇数就交换：

```python
class Solution:
    def reOrderArray(self, array):
        n = len(array)
        for i in range(n):
            for j in range(n-1, i, -1):
                if array[j] & 1 and not array[j - 1] & 1:
                    array[j], array[j-1] = array[j-1], array[j]
```

当然最简单的还是耗空间的解法，遍历楼。

18. 反转链表

> 输入一个链表，反转链表后，输出链表的所有元素。

链表经典题型拉。三个指针拉，pre, cur, pos。cur.next = pre是关键。

```
class Solution:
    # 返回ListNode
    def ReverseList(self, pHead):
        pre = None
        while pHead:
            pos = pHead.next
            pHead.next = pre
            pre = pHead
            pHead = pos
        return pre
```

19. 合并两个排序的链表

> 输入两个单调递增的链表，输出两个链表合成后的链表，当然我们需要合成后的链表满足单调不减规则。

归并排序呗。

```
class Solution:
    # 返回合并后列表
    def Merge(self, pHead1, pHead2):
        L = ListNode(0)
        p, p1, p2 = L, pHead1, pHead2
        while p1 and p2:
            if p1.val <= p2.val:
                p.next = p1
                p1 = p1.next
            else:
                p.next = p2
                p2 = p2.next
            p = p.next
        if p1: p.next = p1
        if p2: p.next = p2
        return L.next
```

20. 树的子结构

> 输入两棵二叉树A，B，判断B是不是A的子结构。（ps：我们约定空树不是任意一个树的子结构）

递归查找呗

```
class Solution:
    def HasSubtree(self, pRoot1, pRoot2):
        if not pRoot2 or not pRoot1: return False
        if self.isSame(pRoot1, pRoot2):
            return True
        return self.HasSubtree(pRoot1.left, pRoot2) or self.HasSubtree(pRoot1.right,
pRoot2)

    def isSame(self, p1, p2):
        if not p2: return True
        if not p1 and p2: return False
        if p1.val != p2.val: return False
        return self.isSame(p1.left, p2.left) and self.isSame(p1.right, p2.right)
```

21. 二叉树的镜像

> 操作给定的二叉树，将其变换为源二叉树的镜像。 二叉树的镜像定义：源二叉树

```
        8
       /  \
      6   10
     / \  / \
    5  7 9 11
    镜像二叉树
        8
       /  \
      10   6
     / \  / \
    11 9 7  5
```

```
class Solution:
    # 返回镜像树的根节点
    def Mirror(self, root):
        if root:
            self.Mirror(root.left)
            self.Mirror(root.right)
            root.left, root.right = root.right, root.left
        return root
```

22. 栈的压入、弹出序列

> 输入两个整数序列，第一个序列表示栈的压入顺序，请判断第二个序列是否为该栈的弹出顺序。假设压入栈的所有数字均不相等。例如序列1,2,3,4,5是某栈的压入顺序，序列4，5,3,2,1是该压栈序列对应的一个弹出序列，但4,3,5,1,2就不可能是该压栈序列的弹出序列。（注意：这两个序列的长度是相等的）

借用一个辅助的栈，遍历压栈顺序，先将第一个放入栈中，这里是1，然后判断栈顶元素是不是出栈顺序的第一个元素，这里是4，很显然1≠4，所以我们继续压栈，直到相等以后开始出栈，出栈一个元素，则将出栈顺序向后移动一位，直到不相等，这样循环等压栈顺序遍历完成，如果辅助栈还不为空，说明弹出序列不是该栈的弹出顺序。

举例：

入栈1,2,3,4,5

出栈4,5,3,2,1

首先1入辅助栈，此时栈顶1≠4，继续入栈2

此时栈顶2≠4，继续入栈3

此时栈顶3≠4，继续入栈4

此时栈顶4＝4，出栈4，弹出序列向后一位，此时为5，,辅助栈里面是1,2,3

此时栈顶3≠5，继续入栈5

此时栈顶5=5，出栈5,弹出序列向后一位，此时为3，,辅助栈里面是1,2,3

....

依次执行，最后辅助栈为空。如果不为空说明弹出序列不是该栈的弹出顺序。

```
class Solution:
    def IsPopOrder(self, pushV, popV):
        n = len(pushV)
        s = []
        j = 0
        for v in pushV:
            s.append(v)
            while j < n and s[-1] == popV[j]:
                s.pop()
                j += 1
        return not s
```

23. 从上往下打印二叉树

> 从上往下打印出二叉树的每个节点，同层节点从左至右打印。

树的层次遍历，队列实现。

```
class Solution:
    # 返回从上到下每个节点值列表，例：[1,2,3]
    def PrintFromTopToBottom(self, root):
        if not root: return []
        queue, res = [root], []
        while queue:
            tmp = []
            while queue:
                p = queue.pop(0)
                res.append(p.val)
                if p.left: tmp.append(p.left)
                if p.right: tmp.append(p.right)
            queue = tmp
        return res
```

24. 二叉搜索树的后序遍历序列

> 输入一个整数数组，判断该数组是不是某二叉搜索树的后序遍历的结果。如果是则输出Yes,否则输出No。假设输入的数组的任意两个数字都互不相同。

分治拉， 后序遍历是这样的（小于根节点，大于根节点， 根节点），找到小于和大于的分界线，从右向左查找，然后递归判断。

```
class Solution:
    def VerifySquenceOfBST(self, sequence):
        if not sequence: return False
        return self.verfify(sequence, 0, len(sequence) - 1)

    def verfify(self, sequence, start, end):
        if start > end: return True
        i = end - 1
        while i >= start and sequence[i] > sequence[end]:
            i -= 1
        if i < start: return True
        for j in range(start, i + 1):
            if sequence[j] > sequence[end]:
                return False
        return self.verfify(sequence, start, i) and self.verfify(sequence, i + 1, end - 1)
```

25. 二叉树中和为某一值的路径

> 输入一颗二叉树和一个整数，打印出二叉树中结点值的和为输入整数的所有路径。路径定义为从树的根结点开始往下一直到叶结点所经过的结点形成一条路径。

dfs

```
class Solution:
    # 返回二维列表，内部每个列表表示找到的路径
    def FindPath(self, root, expectNumber):
        res = []
        path = []

        def dfs(root, expectNumber, path):
            path.append(root.val)
            if not root.left and not root.right and root.val == expectNumber:
res.append(path[:])
            if root.left: dfs(root.left, expectNumber - root.val, path)
            if root.right: dfs(root.right, expectNumber - root.val, path)
            path.pop()

        if not root: return []
        dfs(root, expectNumber, path)
        return res
```

26. 复杂链表的复制

> 输入一个复杂链表（每个节点中有节点值，以及两个指针，一个指向下一个节点，另一个特殊指针指向任意一个节点），返回结果为复制后复杂链表的head。（注意，输出结果中请不要返回参数中的节点引用，否则判题程序会直接返回空）

最简单的方法，先复制一遍不带random的链表，并用字典记录random的指向，第二遍遍历修复random项。复杂度是O(2n)，空间复杂度O(n)。

O(1)空间的解法如下：

1. 在原链表的每个节点后面拷贝出一个新的节点
2. 依次给新的节点的随机指针赋值，而且这个赋值非常容易 cur->next->random = cur->random->next
3. 断开链表可得到深度拷贝后的新链表

```
class RandomListNode:
    def __init__(self, x):
        self.label = x
        self.next = None
        self.random = None

class Solution:
    # 返回 RandomListNode
    def Clone(self, pHead):
        if not pHead: return None
        dic = dict()
        L = RandomListNode(0)
        p, cur = L, pHead
        while cur:
            tmp = RandomListNode(cur.label)
            dic[cur] = tmp
            p.next = tmp
            p, cur = p.next, cur.next

        p, cur = L.next, pHead
        while cur:
            p.random = dic.get(cur.random, None)
            p, cur = p.next, cur.next
        return L.next
```

27. 二叉搜索树与双向链表

> 输入一棵二叉搜索树，将该二叉搜索树转换成一个排序的双向链表。要求不能创建任何新的结点，只能调整树中结点指针的指向。

left指向前一个节点，right指向后一个节点，中序遍历。维护一个left最右的指针，然后修正leftmost, root, right之间的关系。

```
class Solution:
    leftlast = None

    def Convert(self, root):
        if not root: return None
        if not root.left and not root.right:
            self.leftlast = root
            return root
```

```
        left = self.Convert(root.left)
        if left:
            self.leftlast.right = root
            root.left = self.leftlast
        self.leftlast = root
        right = self.Convert(root.right)
        if right:
            right.left = root
            root.right = right
        return left if left else root
```

28. 数组中出现次数超过一半的数字

> 数组中有一个数字出现的次数超过数组长度的一半，请找出这个数字。例如输入一个长度为9的数组{1,2,3,2,2,2,5,4,2}。由于数字2在数组中出现了5次，超过数组长度的一半，因此输出2。如果不存在则输出0。

最简单：

```
from collections import Counter
class Solution:
    def MoreThanHalfNum_Solution(self, numbers):
        if not numbers: return 0
        c = Counter(numbers).most_common(1)[0]
        return c[0] if c[1] > len(numbers)/2 else 0
```

复杂点， 借用众数得思想，一次在数组中删除两个不同得数，直到剩下一个数，如果这个数出现得次数大于一半，这个数最后一定会剩下来。

```
class Solution:
    def MoreThanHalfNum_Solution(self, numbers):
        cand, times = 0, 0
        for number in numbers:
            if times == 0:
                cand, times = number, 1
            elif cand == number:
                times += 1
            else:
                times -= 1

        c = 0
        for number in numbers:
            if number == cand:
                c += 1
        return cand if c > len(numbers) / 2 else 0
```

29. 最小的K个数

> 输入n个整数，找出其中最小的K个数。例如输入4,5,1,6,2,7,3,8这8个数字，则最小的4个数字是1,2,3,4,。

可以用快排，可以用最小堆，还有牛逼得BFPRT算法。

只能说牛客网比较奇葩，返回结果还需要排序，我去，下列代码会超时，so不如用`sorted(a)[:k]`一行秒。

```
class Solution:
    def GetLeastNumbers_Solution(self, a, k):
        if k > len(a): return []
        l, h = 0, len(a) - 1
        i = self.partition(a, l, h)
        while i != k - 1:
            if i > k - 1:
                i = self.partition(a, l, i - 1)
            else:
                i = self.partition(a, i + 1, h)
        return [a[i] for i in range(k)]

    def partition(self, a, l, h):
        tmp = a[l]
        while l < h:
            while l < h and a[h] >= tmp: h -= 1
            a[l] = a[h]
            while l < h and a[l] <= tmp: l += 1
            a[h] = a[l]
        a[l] = tmp
        return l
```

30. 连续子数组的最大和

> HZ偶尔会拿些专业问题来忽悠那些非计算机专业的同学。今天测试组开完会后,他又发话了:在古老的一维模式识别中,常常需要计算连续子向量的最大和,当向量全为正数的时候,问题很好解决。但是,如果向量中包含负数,是否应该包含某个负数,并期望旁边的正数会弥补它呢？例如:{6,-3,-2,7,-15,1,2,2},连续子向量的最大和为8(从第0个开始,到第3个为止)。你会不会被他忽悠住？(子向量的长度至少是1)

_sum局部累加和，_max全部最大和。注意里面_max得初始值，和求max得顺序。

```
class Solution:
    def FindGreatestSumOfSubArray(self, array):
        _max, _sum = -0x80000000, 0
        for i in array:
            _sum += i
            _max = max(_sum, _max) #写在下一句之前，保证_max可以为负数
            if _sum < 0: _sum = 0
        return _max
```

31. 整数中1出现的次数（从1到n整数中1出现的次数）

> 求出1~13的整数中1出现的次数,并算出100~1300的整数中1出现的次数？为此他特别数了一下1~13中包含1的数字有1、10、11、12、13因此共出现6次,但是对于后面问题他就没辙了。ACMer希望你们帮帮他,并把问题更加普遍化,可以很快的求出任意非负整数区间中1出现的次数。

详解：https://leetcode.com/problems/number-of-digit-one/discuss/64381

把数字划分为两部分，如n=3141592, 考虑百分位，m=100, 则a=31415, b = 92 ，然后我们知道n的百分位为1的前缀是从“”到3141，共3142次，后缀都是100次，也就是$(a/10+1)*100$。

再考虑千分位，m=1000, a=3141, b = 592, 千分位为1的前缀是“”到314，共315次，但是千位数是1，后缀并不都是1000次，只有314次，另外一种情况是后缀是“000”到“592”。所以 $(a/10*1000)+(b+1)$。

当前位置是0还是1或者>=2可以用一个公式表达，也就是$(a+8)/10$，最后的部分尾巴用a%10==1判断。

```
class Solution:
    def NumberOf1Between1AndN_Solution(self, n):
        ones, m = 0, 1
        while m <= n:
            a, b = n // m, n % m
            ones += (a + 8) // 10 * m + (a % 10 == 1) * (b + 1)
            m *= 10
        return ones
```

32. 把数组排成最小的数

> 输入一个正整数数组，把数组里所有数字拼接起来排成一个数，打印能拼接出的所有数字中最小的一个。例如输入数组{3，32，321}，则打印出这三个数字能排成的最小数字为321323。

排序，怎么排序呢?

如果我们有s1=9, s2=31,排列有s1+s2, s2+s1，本地求最小，当然选拼接起来小的那一种。python3取消了sort的cmp方法,但保留在functools中。

```
class Solution:
    # @param {integer[]} nums
    # @return {string}
    def PrintMinNumber(self, nums):
        nums = [str(i) for i in nums]
        from functools import cmp_to_key
        nums.sort(key=cmp_to_key(lambda x, y: int(x+y)-int(y+x)))
        return ''.join(nums) or ''
```

33. 第一个只出现一次的字符

> 在一个字符串(1<=字符串长度<=10000，全部由字母组成)中找到第一个只出现一次的字符,并返回它的位置

判断当前位置得字符s[i]是否在s[:i]+s[i+1:]中。

```python
class Solution:
    def FirstNotRepeatingChar(self, s):
        for i in range(len(s)):
            if s[i] not in s[:i]+s[i+1:]: return i
        return -1
```

34. 数组中的逆序对

> 在数组中的两个数字，如果前面一个数字大于后面的数字，则这两个数字组成一个逆序对。输入一个数组,求出这个数组中的逆序对的总数P。并将P对1000000007取模的结果输出。 即输出P%1000000007
>
> 题目保证输入的数组中没有的相同的数字数据范围：对于%50的数据,size<=10^4 对于%75的数据,size<=10^5 对于%100的数据,size<=$2*10^5$
>
> 示例
>
> 输入 : 1,2,3,4,5,6,7,0
>
> 输出 : 7

归并排序。如果是求重要逆序数，记得先不进行归并，先求count，然后再做一遍归并（leetcode 193）。很遗憾，牛客网超时。

```python
class Solution:
    def merge(self, A, B):
        i, j = 0, 0
        C = []
        count = 0
        while i < len(A) and j < len(B):
            if A[i] <= B[j]:
                C.append(A[i])
                i += 1
            else:
                count += len(A) - i
                C.append(B[j])
                j += 1
            C.extend(A[i:])
            C.extend(B[j:])
            return count, C

    def mergesort(self, nums):
        if len(nums) <= 1:
            return 0, nums
        m = len(nums) // 2
        a, left = self.mergesort(nums[:m])
        b, right = self.mergesort(nums[m:])
        c, res = self.merge(left, right)
        return a + b + c, res

    def InversePairs(self, data):
        c, _ = self.mergesort(data)
        return c % 1000000007
```

35. 两个链表的第一个公共结点

> 输入两个链表，找出它们的第一个公共结点。

例如

```
A:          a1 → a2
                   ↘
                     c1 → c2 → c3
                   ↗
B:     b1 → b2 → b3
```

笨一点得方法是先求出A, B 连链表得长度（假设A比较长)， 那么A从len(A) - len(B)处开始和B比较，这样两者判断得长度一样。

当然，还有比较巧妙地写法。

p1, p2指针分别指向A, B得head，任一指针到达链表尾，则指向另一链表得头，继续第二轮比较。在第二轮中，他们要么相遇， 要么没有。如果相遇，该点即为所求，如果没有相遇，他们会在同一轮到达链表尾，也就是 p1=p2=None，返回None。（详情见https://leetcode.com/problems/intersection-of-two-linked-lists/discuss/49798)

```
class Solution:
    def FindFirstCommonNode(self, headA, headB):
        if not headA or not headB: return None
        p1, p2 = headA, headB
        while p1 is not p2:
            p1 = p1.next if p1 else headB
            p2 = p2.next if p2 else headA
        return p1
```

36. 数字在排序数组中出现的次数

> 统计一个数字在排序数组中出现的次数。

遍历

二分查找left和right

```
import bisect
class Solution:
    def GetNumberOfK(self, data, k):
        left = bisect.bisect_left(data, k)
        right = bisect.bisect_right(data, k)
        return right - left
```

复习下这两种写法：

```
def bisect_left(a, x, lo=0, hi=None):
    """Return the index where to insert item x in list a, assuming a is sorted.
```

```
    The return value i is such that all e in a[:i] have e < x, and all e in
    a[i:] have e >= x.  So if x already appears in the list, a.insert(x) will
    insert just before the leftmost x already there.

    Optional args lo (default 0) and hi (default len(a)) bound the
    slice of a to be searched.
    """

    if lo < 0:
        raise ValueError('lo must be non-negative')
    if hi is None:
        hi = len(a)
    while lo < hi:
        mid = (lo+hi)//2
        if a[mid] < x: lo = mid+1
        else: hi = mid
    return lo
```

```
def bisect_right(a, x, lo=0, hi=None):
    """Return the index where to insert item x in list a, assuming a is sorted.

    The return value i is such that all e in a[:i] have e <= x, and all e in
    a[i:] have e > x.  So if x already appears in the list, a.insert(x) will
    insert just after the rightmost x already there.

    Optional args lo (default 0) and hi (default len(a)) bound the
    slice of a to be searched.
    """

    if lo < 0:
        raise ValueError('lo must be non-negative')
    if hi is None:
        hi = len(a)
    while lo < hi:
        mid = (lo+hi)//2
        if x < a[mid]: hi = mid
        else: lo = mid+1
    return lo
```

37. 二叉树的深度

> 输入一棵二叉树，求该树的深度。从根结点到叶结点依次经过的结点（含根、叶结点）形成树的一条路径，最长路径的长度为树的深度。

```
class Solution:
    def TreeDepth(self, pRoot):
        # write code here
        if not pRoot: return 0
        else: return max(self.TreeDepth(pRoot.left), self.TreeDepth(pRoot.right)) + 1
```

38. 平衡二叉树

> 输入一棵二叉树，判断该二叉树是否是平衡二叉树。

借用上题得求深度函数。

```
class Solution:
    def depth(self, root):
        if not root: return 0
        return max(self.depth(root.left), self.depth(root.right)) + 1

    def IsBalanced_Solution(self, root):
        if not root: return True
        return abs(self.depth(root.left) - self.depth(root.right)) < 2 and
self.IsBalanced_Solution(root.left) and self.IsBalanced_Solution(root.right)
```

39. 数组中只出现一次的数字

> 一个整型数组里除了两个数字之外，其他的数字都出现了两次。请写程序找出这两个只出现一次的数字。

基础版本：一个整型数组里除了一个数字之外，其它的数字都出现了两次。请写程序找出这个只出现一次的数字。

这个比较好做，直接采用异或操作，剩下得就是只出现一次得。

本题是上述得扩展，把数组划分成两部分，每一部分包含一个只出现一次的数字，之后就是异或拉。怎么拆分呢？对所有数组异或，结果肯定不为0，因为有两个只出现一次得数字。在结果数字中找到第一个为1的位的位置，记为第N位。现在我们以第N位是不是1为标准把原数组中的数字分成两个子数组，第一个子数组中每个数字的第N位都为1，而第二个子数组的每个数字的第N位都为0。

详情见（http://blog.csdn.net/u011826264/article/details/39269101）。

```
class Solution:
    # 返回[a,b] 其中ab是出现一次的两个数字
    def FindNumsAppearOnce(self, array):
        n = 0
        for i in array: n ^= i
        n = bin(n)[2:][::-1]
        j = n.find('1')
        a1 = filter(lambda x: bin(x)[2:].zfill(len(n))[::-1][j] == '1', array)
        a2 = filter(lambda x: bin(x)[2:].zfill(len(n))[::-1][j] == '0', array)
        num1, num2 = 0, 0
        for i in a1: num1 ^= i
        for i in a2: num2 ^= i
        return [num1, num2]
```

写完得感受是，python大法好。

40. 和为S的连续正数序列

> 小明很喜欢数学,有一天他在做数学作业时,要求计算出9~16的和,他马上就写出了正确答案是100。但是他并不满足于此,他在想究竟有多少种连续的正数序列的和为100(至少包括两个数)。没多久,他就得到另一组连续正数和为100的序列:18,19,20,21,22。现在把问题交给你,你能不能也很快的找出所有和为S的连续正数序列? Good Luck!
>
> 输出描述:
>
> 输出所有和为S的连续正数序列。序列内按照从小至大的顺序，序列间按照开始数字从小到大的顺序

双指针，大了就h--,小了就l++。l到h得和就用$(h+l)*(h-l+1)/2$表示。

```
class Solution:
    def FindContinuousSequence(self, tsum):
        res = []
        l, h = 1, 2
        while l < h:
            cur = (h + l) * ( h - l + 1) // 2
            if cur < tsum: h += 1
            if cur == tsum:
                res.append(list(range(l, h+1)))
                l += 1
            if cur > tsum:
                l += 1
        return res
```

41. 和为S的两个数字

> 输入一个递增排序的数组和一个数字S，在数组中查找两个数，是的他们的和正好是S，如果有多对数字的和等于S，输出两个数的乘积最小的。
>
> 输出描述:
>
> 对应每个测试案例，输出两个数，小的先输出。

双指针遍历。

```
class Solution:
    def FindNumbersWithSum(self, array, target):
        n = len(array)
        l, h = 0, n - 1
        while l < h:
            t = array[l] + array[h]
            if t > target:
                h -= 1
            elif t < target:
                l += 1
            else:
                return [array[l], array[h]]
        return []
```

42. 左旋转字符串

> 汇编语言中有一种移位指令叫做循环左移（ROL），现在有个简单的任务，就是用字符串模拟这个指令的运算结果。对于一个给定的字符序列S，请你把其循环左移K位后的序列输出。例如，字符序列S=”abcXYZdef”,要求输出循环左移3位后的结果，即“XYZdefabc”。是不是很简单？OK，搞定它！

字符串截取，用py很好实现，注意取模。

```
class Solution:
    def LeftRotateString(self, s, k):
        if not s: return s
        n = len(s)
        return s[k%n:]+s[:k%n]
```

43. 翻转单词顺序列

> 牛客最近来了一个新员工Fish，每天早晨总是会拿着一本英文杂志，写些句子在本子上。同事Cat对Fish写的内容颇感兴趣，有一天他向Fish借来翻看，但却读不懂它的意思。例如，“student. a am I”。后来才意识到，这家伙原来把句子单词的顺序翻转了，正确的句子应该是“I am a student.”。Cat对一一的翻转这些单词顺序可不在行，你能帮助他么？

python2 和 python3 有些不一样，牛客网是python2，split函数有些小区别。

```
class Solution:
    def ReverseSentence(self, s):
        return ' '.join(s.split(' ')[::-1])
```

44. 扑克牌顺子

> LL今天心情特别好,因为他去买了一副扑克牌,发现里面居然有2个大王,2个小王(一副牌原本是54张-)...他随机从中抽出了5张牌,想测测自己的手气,看看能不能抽到顺子,如果抽到的话,他决定去买体育彩票,嘿嘿！！“红心A,黑桃3,小王,大王,方片5”,“Oh My God!”不是顺子.....LL不高兴了,他想了想,决定大\小 王可以看成任何数字,并且A看作1,J为11,Q为12,K为13。上面的5张牌就可以变成“1,2,3,4,5”(大小王分别看作2和4),“So Lucky!”。LL决定去买体育彩票啦。 现在,要求你使用这幅牌模拟上面的过程,然后告诉我们LL的运气如何。为了方便起见,你可以认为大小王是0。

必须满足两个条件: 1. 除0外没有重复的数; 2. max - min < 5。

```
public class Solution {
    public boolean isContinuous(int [] numbers) {
        if(numbers.length != 5) return false;
        int min = 14;
        int max = -1;
        int flag = 0;
        for(int i = 0; i < numbers.length; i++) {
            int number = numbers[i];
            if(number < 0 || number > 13) return false;
            if(number == 0) continue;
            if(((flag >> number) & 1) == 1) return false;
            flag |= (1 << number);

            if(number > max) max = number;
```

```
            if(number < min) min = number;
            if(max - min >= 5) return false;
        }
        return true;
    }
}
```

45. 孩子们的游戏(圆圈中最后剩下的数)

> 每年六一儿童节,牛客都会准备一些小礼物去看望孤儿院的小朋友,今年亦是如此。HF作为牛客的资深元老,自然也准备了一些小游戏。其中,有个游戏是这样的:首先,让小朋友们围成一个大圈。然后,他随机指定一个数m,让编号为0的小朋友开始报数。每次喊到m-1的那个小朋友要出列唱首歌,然后可以在礼品箱中任意的挑选礼物,并且不再回到圈中,从他的下一个小朋友开始,继续0...m-1报数....这样下去....直到剩下最后一个小朋友,可以不用表演,并且拿到牛客名贵的“名侦探柯南”典藏版(名额有限哦!!-)。请你试着想下,哪个小朋友会得到这份礼品呢？(注：小朋友的编号是从0到n-1)

用数组模拟，但是超时了。

```
class Solution:
    def LastRemaining_Solution(self, n, m):
        if n <= 0 or m <= 0: return -1
        kids = [1] * n
        j = 0
        for i in range(n):
            c = 0
            while c < m - 1:
                j = (j + 1) % n
                if kids[j]: c += 1
            kids[j] = 0

        return j
```

数学的归纳法：

令f[i]表示i个人玩游戏报m退出最后胜利者的编号，最后的结果自然是f[n]。

递推公式

f[1]=0;

f[i]=(f[i-1]+m)%i; (i>1)

```
class Solution:
    def LastRemaining_Solution(self, n, m):
        if n <= 0 or m <= 0: return -1
        s = 0
        for i in range(2, n + 1):
            s = (s + m) % i
        return s
```

46. 求1+2+3+...+n

> 求1+2+3+...+n，要求不能使用乘除法、for、while、if、else、switch、case等关键字及条件判断语句（A?B:C）。

$\frac{n^2+n}{2}$，用乘方、加、右移实现

```
class Solution:
    def Sum_Solution(self, n):
        return (n ** 2 + n) >> 1
```

47. 把字符串转换成整数

> 题目描述
>
> 将一个字符串转换成一个整数，要求不能使用字符串转换整数的库函数。 数值为0或者字符串不是一个合法的数值则返回0
>
> 输入描述:
>
> 输入一个字符串,包括数字字母符号,可以为空
>
> 输出描述:
>
> 如果是合法的数值表达则返回该数字，否则返回0
>
> 示例1
>
> 输入 +2147483647 1a33
>
> 输出 2147483647 0

遍历，遍历。

```
class Solution:
    def StrToInt(self, s):
        if not s: return 0
        n = len(s)
        res, base, i = 0, 1, 0
        flag = True
        if s[0] in '+-': i = 1
        if s[0] == '-': flag = False
        j = n - 1
        while j >= i:
            if s[j].isdigit():
                res += (ord(s[j]) - ord('0')) * base
                base *= 10
            else:
                return 0
            j -= 1
        return res if flag else -res
```

48. 数组中重复的数字

> 在一个长度为n的数组里的所有数字都在0到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是重复的。也不知道每个数字重复几次。请找出数组中任意一个重复的数字。 例如，如果输入长度为7的数组{2,3,1,0,2,5,3}，那么对应的输出是第一个重复的数字2。

判断i是否在n[:i]之中就行，但是python怎么也过不了。

```
class Solution:
    # 这里要特别注意~找到任意重复的一个值并赋值到duplication[0]
    # 函数返回True/False
    def duplicate(self, numbers, duplication):
        # write code here
        for i in range(len(numbers)):
            if numbers[i] in numbers[:i]:
                duplication[0] = numbers[i]
                return True
        return False
```

java可以过的版本：

```
public boolean duplicate(int numbers[], int length, int[] duplication) {
        boolean[] k = new boolean[length];
        for (int i = 0; i < k.length; i++) {
            if (k[numbers[i]] == true) {
                duplication[0] = numbers[i];
                return true;
            }
            k[numbers[i]] = true;
        }
        return false;
    }
```

49. 构建乘积数组

> 给定一个数组A[0,1,...,n-1],请构建一个数组B[0,1,...,n-1],其中B中的元素B[i]=A[0]*A[1]...A[i-1]*A[i+1]...A[n-1]。不能使用除法。

left保持左侧连乘，right保存右侧起的连乘。

```
class Solution:
    def multiply(self, A):
        left, right = [1], [1]
        for i in A:
            left.append(left[-1] * i)
        for i in A[::-1]:
            right.append(right[-1] * i)
        B = []
        for i in range(len(A)):
            B.append(left[i] * right[-2-i])
        return B
```

50. 正则表达式匹配

> 请实现一个函数用来匹配包括'.'和'*'*的正则表达式。模式中的字符*'.'*表示任意一个字符，而*'*'表示它前面的字符可以出现任意次（包含0次）。 在本题中，匹配是指字符串的所有字符匹配整个模式。例如，字符串"aaa"与模式"a.a"和"ab*ac*a"匹配，但是与"aa.a"和"ab*a"均不匹配

动态规划拉。

This problem has a typical solution using Dynamic Programming. We define the state `P[i][j]` to be `true` if `s[0..i)` matches `p[0..j)` and `false` otherwise. Then the state equations are:

1. `P[i][j] = P[i - 1][j - 1]`, if `p[j - 1] != '*' && (s[i - 1] == p[j - 1] || p[j - 1] == '.')`;
2. `P[i][j] = P[i][j - 2]`, if `p[j - 1] == '*'` and the pattern repeats for `0` times;
3. `P[i][j] = P[i - 1][j] && (s[i - 1] == p[j - 2] || p[j - 2] == '.')`, if `p[j - 1] == '*'` and the pattern repeats for at least `1` times.

```
class Solution:
    # s, pattern都是字符串
    def match(self, s, p):
        m, n = len(s), len(p)
        dp = [[False] * (n + 1) for _ in range(m + 1)]
        dp[0][0] = True
        for i in range(m+1):
            for j in range(1, n+1):
                if p[j - 1] == '*':
                    dp[i][j] = dp[i][j - 2] or (
                            i > 0 and (s[i - 1] == p[j - 2] or p[j - 2] == '.') and dp[i -
1][j])
                else:
                    dp[i][j] = i > 0 and dp[i - 1][j - 1] and (s[i - 1] == p[j - 1] or p[j
- 1] == '.')
        return dp[m][n]
```

51. 表示数值的字符串

> 请实现一个函数用来判断字符串是否表示数值（包括整数和小数）。例如，字符串"+100","5e2","-123","3.1416"和"-1E-16"都表示数值。 但是"12e","1a3.14","1.2.3","+-5"和"12e+4.3"都不是。

正则表达式比较简单。

```
import re

class Solution:
    # s字符串
    def isNumeric(self, s):
        return re.match(r"^[+-]?[0-9]*(\.[0-9]*)?([eE][+-]?[0-9]+)?$", s) is not None
```

52. 字符流中第一个不重复的字符

> 请实现一个函数用来找出字符流中第一个只出现一次的字符。例如，当从字符流中只读出前两个字符"go"时，第一个只出现一次的字符是"g"。当从该字符流中读出前六个字符“google"时，第一个只出现一次的字符是"l"。
>
> 输出描述:
>
> 如果当前字符流没有存在出现一次的字符，返回#字符。

用一个队列（保证输出第一个出现一次的字符）和数组（anscii码256）实现，每次insert时候判断当前字符的次数，如果只出现一次，插入到队列中，输出时要再判断一下该字符是否只出现一次。

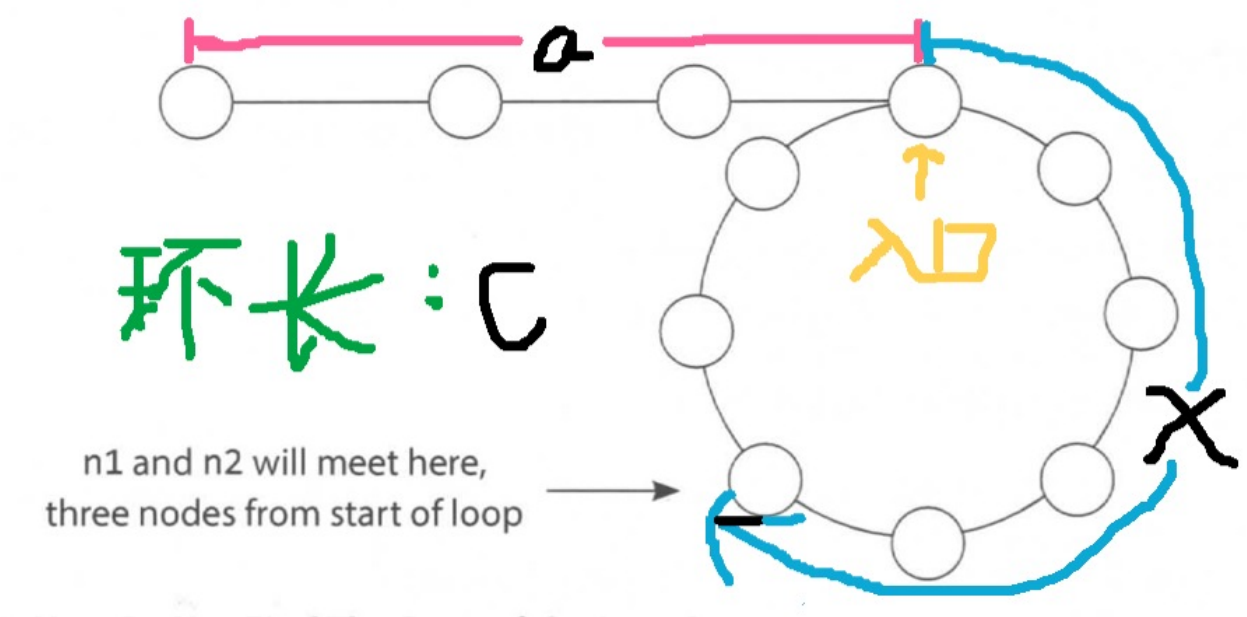
```
class Solution:
    def __init__(self):
        self.s = [0] * 256
        self.q = []

    def FirstAppearingOnce(self):
        while self.q and self.s[ord(self.q[0])] > 1: self.q.pop(0)
        if not self.q: return '#'
        return self.q[0]

    def Insert(self, c):
        self.s[ord(c)] += 1
        if self.s[ord(c)] == 1:
            self.q.append(c)
```

53. 链表中环的入口结点

> 一个链表中包含环，请找出该链表的环的入口结点。

快慢指针发现是否有环（leetcode 141）：

```
class Solution(object):
    def hasCycle(self, head):
        """
        :type head: ListNode
        :rtype: bool
        """
        if not head: return False
        p1, p2 = head, head
        while p1.next and p2.next and p2.next.next:
            p1, p2 = p1.next, p2.next.next
            if p1 == p2:
                return True
        return False
```



从链表起始处到环入口长度为：a，从环入口到Faster和Slower相遇点长度为：x，整个环长为：c。

假设从开始到相遇，Slower走过的路程长为s，由于Faster的步速是Slower的2倍，那么Faster在这段时间走的路程长为2s。

而对于Faster来说，他走的路程还等于之前绕整个环跑的n圈的路程nc，加上最后这一次遇见Slower的路程s。

所以我们有：

$2s = nc + s$

对于Slower来说，他走的路程长度s还等于他从链表起始处到相遇点的距离，所以有：

$s = a + x$

通过以上两个式子代入化简有：

$a + x = nc$

$a = nc - x$

$a = (n-1)c + c-x$

$a = kc + (c-x)$

那么可以看出，c-x，就是从相遇点继续走回到环入口的距离。上面整个式子可以看出，如果此时有个pointer1从起始点出发并且同时还有个pointer2从相遇点出发继续往前走（都只迈一步），那么绕过k圈以后， pointer2会和pointer1在环入口相遇。这样，换入口就找到了。(摘自 https://www.cnblogs.com/springfor/p/3862125.html)

```
class Solution(object):
    def detectCycle(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """
        if not head: return None
        p1, p2 = head, head
        hasCycle = False

        while p1.next and p2.next and p2.next.next:
```

```
            p1, p2 = p1.next, p2.next.next
            if p1 == p2:
                hasCycle = True
                break
        if not hasCycle: return None
        p2 = head
        while p1 != p2:
            p1, p2 = p1.next, p2.next
        return p1
```

54. 删除链表中重复的结点

> 在一个排序的链表中，存在重复的结点，请删除该链表中重复的结点，重复的结点不保留，返回链表头指针。 例如，链表1->2->3->3->4->4->5 处理后为 1->2->5

重复的保留一个(leetcode 83):

```
class Solution:
    def deleteDuplication(self, head):
        if not head: return head
        cur = head
        while cur and cur.next:
            pos = cur.next
            if pos.val == cur.val:
                cur.next = pos.next
                del pos
            else:
                cur = pos
        return head
```

一个不留的(leetcode 82):

```
class Solution(object):
    def deleteDuplicates(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """
        if not head: return head
        dump = ListNode(0)
        pre, dump.next, cur = dump, head, head
        while cur:
            while cur.next and cur.next.val == cur.val:
                cur = cur.next
            if pre.next == cur:  # 说明不重复，没有删除元素
                pre = pre.next
            else:
                pre.next = cur.next
            cur = cur.next
        return dump.next
```

55. 二叉树的下一个结点

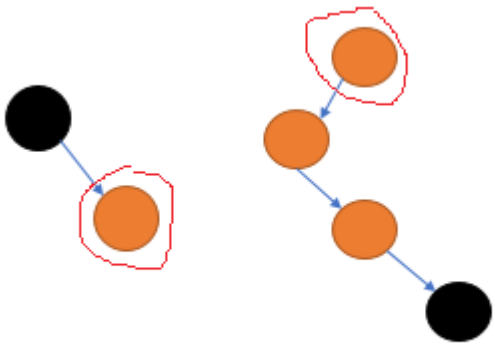
> 给定一个二叉树和其中的一个结点，请找出中序遍历顺序的下一个结点并且返回。注意，树中的结点不仅包含左右子结点，同时包含指向父结点的指针。

分析二叉树的下一个节点，一共有以下情况：

1.二叉树为空，则返回空；

2.节点右孩子存在，则设置一个指针从该节点的右孩子出发，一直沿着指向左子结点的指针找到的叶子节点即为下一个节点；

3.节点不是根节点。如果该节点是其父节点的左孩子，则返回父节点；否则继续向上遍历其父节点的父节点，重复之前的判断，返回结果。



```
class TreeLinkNode:
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None
        self.next = None

class Solution:
    def GetNext(self, root):
        if not root: return None
        if root.right:
            root = root.right
            while root.left:
                root = root.left
            return root
        while root.next:
            p = root.next
            if p.left == root: return p
            root = root.next
        return None
```

56. 对称的二叉树

> 请实现一个函数，用来判断一颗二叉树是不是对称的。注意，如果一个二叉树同此二叉树的镜像是同样的，定义其为对称的。

一开始想得是层次遍历，然后判断每层是否对称，后来发现不对，因为，如如果某一层是 4 # 2 4，满足要求但不是对称得。

其实就是dfs啊。

```
class Solution:
    def isSymmetrical(self, root):
        def check(left, right):
            if not left or not right: return left == right
            return left.val == right.val and check(left.left, right.right) and
check(left.right, right.left)
        return root is None or check(root.left, root.right)
```

57. 按之字形顺序打印二叉树

> 请实现一个函数按照之字形打印二叉树，即第一行按照从左到右的顺序打印，第二层按照从右至左的顺序打印，第三行按照从左到右的顺序打印，其他行以此类推。

层次遍历：

```
class Solution:
    # 返回二维列表[[1,2],[4,5]]
    def Print(self, root):
        if not root: return []
        queue = [root]
        res = []
        flag = True
        i = 0
        while queue:
            tmp, tmpval = [], []
            i = 1 if not i else 0
            while queue:
                p = queue.pop(0)
                if i: tmpval.append(p.val)
                else: tmpval.insert(0, p.val)
                if p.left: tmp.append(p.left)
                if p.right: tmp.append(p.right)
            queue = tmp
            res.append(tmpval)
        return res
```

我以前leetcode上的写法：

```
class Solution(object):
    def zigzagLevelOrder(self, root):
        """
        :type root: TreeNode

        :rtype: List[List[int]]
```

```
        """
        import collections
        queue, res = collections.deque([(root, 0)]), []
        while queue:
            node, level = queue.popleft()
            if node:
                if len(res) < level + 1:
                    res.insert(level, [])
                if level % 2 == 0:
                    res[level].append(node.val)
                else:
                    res[level].insert(0, node.val)
                queue.append((node.left, level + 1))
                queue.append((node.right, level + 1))
        return res
```

58. 把二叉树打印成多行

> 从上到下按层打印二叉树，同一层结点从左至右输出。每一层输出一行。

层次遍历。我的习惯性做法：

```
class Solution:
    # 返回二维列表[[1,2],[4,5]]
    def Print(self, root):
        if not root: return []
        queue = [root]
        res = []
        while queue:
            tmp, tmpval = [], []
            while queue:
                p = queue.pop(0)
                tmpval.append(p.val)
                if p.left: tmp.append(p.left)
                if p.right: tmp.append(p.right)
            queue = tmp
            res.append(tmpval)
        return res
```

当然，可以改进下：

```
class Solution:
    # 返回二维列表[[1,2],[4,5]]
    def Print(self, root):
        if not root: return []
        queue = [root]
        res = []
        while queue:
            tmp = []
            n = len(queue)

            for p in queue:
```

```
                tmp.append(p.val)
            res.append(tmp)
            for _ in range(n):
                p = queue.pop(0)
                if p.left: queue.append(p.left)
                if p.right: queue.append(p.right)
        return res
```

59. 序列化二叉树

> 请实现两个函数，分别用来序列化和反序列化二叉树

先序遍历。

```
class Solution:
    def serialize(self, root):
        def doit(node):
            if node:
                vals.append(str(node.val))
                doit(node.left)
                doit(node.right)
            else:
                vals.append('#')
        vals = []
        doit(root)
        return ' '.join(vals)

    def deserialize(self, data):
        def doit():
            val = next(vals)
            if val == '#':
                return None
            node = TreeNode(int(val))
            node.left = doit()
            node.right = doit()
            return node
        vals = iter(data.split())
        return doit()
```

60. 二叉搜索树的第k个结点

> 给定一颗二叉搜索树，请找出其中的第k大的结点。例如， 5 / \ 3 7 /\ /\ 2 4 6 8 中，按结点数值大小顺序第三个结点的值为4。

中序遍历吧：

```
class Solution:
    # 返回对应节点TreeNode
    def KthNode(self, root, k):
        if k < 1 or not root: return None
        res = []
```

```
        def dfs(root, res):
            if not root: return
            dfs(root.left, res)
            res.append(root)
            dfs(root.right, res)

        dfs(root, res)
        return res[k-1] if k <= len(res) else None
```

改进下，这个递归有点难懂，在到达k个节点之前是不会返回的，到达k时返回return root, 上层的就继续返回return left or return right：

```
class Solution:
    # 返回对应节点TreeNode
    count = 0
    def KthNode(self, root, k):
        if root:
            left = self.KthNode(root.left, k)
            if left: return left
            self.count += 1
            if self.count == k: return root
            right = self.KthNode(root.right, k)
            if right: return right
        return None
```

61. 数据流中的中位数

> 如何得到一个数据流中的中位数？如果从数据流中读出奇数个数值，那么中位数就是所有数值排序之后位于中间的数值。如果从数据流中读出偶数个数值，那么中位数就是所有数值排序之后中间两个数的平均值。

解题思路就是用两个堆，一个大顶堆，一个小顶堆来过滤数据。大顶堆代表排序后得左边数据，小顶堆代表右半边数据。维持这一特性即可。

```
import heapq
class Solution:
    def __init__(self):
        self.maxheap = []
        self.minheap = []
        self.count = 0

    def Insert(self, num):
        self.count += 1
        if self.count % 2 == 0:
            heapq.heappush(self.maxheap, -num)
            tmp = -heapq.heappop(self.maxheap)
            heapq.heappush(self.minheap, tmp)
        else:
            heapq.heappush(self.minheap, num)
            tmp = -heapq.heappop(self.minheap)
            heapq.heappush(self.maxheap, tmp)
```

```
    def GetMedian(self):
        if self.count % 2 == 0:
            return (-self.maxheap[0] + self.minheap[0]) / 2
        else:
            return -self.maxheap[0]
```

还是牛客网过不了，奇怪得报错。

牛客网的一个写法：

```
class Solution {
  priority_queue<int, vector<int>, less<int> > p;
  priority_queue<int, vector<int>, greater<int> > q;
  pulic void Insert(int num){
      if(p.empty() || num <= p.top()) p.push(num);
      else q.push(num);
      if(p.size() == q.size() + 2) q.push(p.top()), p.pop();
      if(p.size() + 1 == q.size()) p.push(q.top()), q.pop();
  }
  double GetMedian(){
    return p.size() == q.size() ? (p.top() + q.top()) / 2.0 : p.top();
  }
}
```

```
class Solution {
    vector<int> v;
    int n;
    public:
        void Insert(int num){
            v.push_back(num);
            n = v.size();
            for(int i = n - 1; i > 0 && v[i] < v[i - 1]; --i) swap(v[i], v[i - 1]);
        }
        double GetMedian(){
          return (v[(n - 1) >> 1] + v[n >> 1]) / 2.0;
        }
};
*/
```

62. 滑动窗口的最大值

> 给定一个数组和滑动窗口的大小，找出所有滑动窗口里数值的最大值。例如，如果输入数组{2,3,4,2,6,2,5,1}及滑动窗口的大小3，那么一共存在6个滑动窗口，他们的最大值分别为{4,4,6,6,6,5}；针对数组{2,3,4,2,6,2,5,1}的滑动窗口有以下6个：{[2,3,4],2,6,2,5,1}，{2,[3,4,2],6,2,5,1}，{2,3,[4,2,6],2,5,1}，{2,3,4,[2,6,2],5,1}，{2,3,4,2,[6,2,5],1}，{2,3,4,2,6,[2,5,1]}。

我的想法简单，维持滑动窗口得最大值，就三种情况，最大值在当前位置，在上一窗口和当前窗口重叠得部分，上一轮最大值被抛弃。

```python
class Solution:
    def maxInWindows(self, num, size):
        if size < 1 or size > len(num): return []
        _max = max(num[:size])
        res = [_max]
        for i in range(size, len(num)):
            if num[i] > num[i - size]:
                _max = max(_max, num[i])
            elif _max == num[i - size]:
                _max = max(num[i - size + 1:i + 1])
            res.append(_max)
        return res
```

感觉牛客网上推荐得答案也不是太好，话不如直接max函数：

```java
import java.util.*;
/**
用一个双端队列，队列第一个位置保存当前窗口的最大值，当窗口滑动一次
1.判断当前最大值是否过期
2.新增加的值从队尾开始比较，把所有比他小的值丢掉
*/
public class Solution {
   public ArrayList<Integer> maxInWindows(int [] num, int size)
    {
        ArrayList<Integer> res = new ArrayList<>();
        if(size == 0) return res;
        int begin;
        ArrayDeque<Integer> q = new ArrayDeque<>();
        for(int i = 0; i < num.length; i++){
            begin = i - size + 1;
            if(q.isEmpty())
                q.add(i);
            else if(begin > q.peekFirst())
                q.pollFirst();

            while((!q.isEmpty()) && num[q.peekLast()] <= num[i])
                q.pollLast();
            q.add(i);
            if(begin >= 0)
                res.add(num[q.peekFirst()]);
        }
        return res;
    }
}
```

63. 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个格子开始，每一步可以在矩阵中向左，向右，向上，向下移动一个格子。如果一条路径经过了矩阵中的某一个格子，则该路径不能再进入该格子。 例如 a b c e s f c s a d e e 矩阵中包含一条字符串"bcced"的路径，但是矩阵中不包含"abcb"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，路径不能再次进入该格子。

dfs

```
class Solution:
    def hasPath(self, matrix, rows, cols, path):
        visit = [False] * (cols *rows)

        def dfs(matrix, rows, cols, path, visit, i, j, k):
            index = i * cols + j
            if i < 0 or j < 0 or i >= rows or j >= cols or visit[index] or matrix[index] !=
path[k]:
                return False
            if k == len(path) - 1: return True
            visit[index] = True
            if dfs(matrix, rows, cols, path, visit, i - 1, j, k + 1) or \
                    dfs(matrix, rows, cols, path, visit, i + 1, j, k + 1) or \
                    dfs(matrix, rows, cols, path, visit, i, j - 1, k + 1) or \
                    dfs(matrix, rows, cols, path, visit, i, j + 1, k + 1):
                return True
            visit[index] = False
            return False

        for i in range(rows):
            for j in range(cols):
                if dfs(matrix, rows, cols, path, visit, i, j, 0):
                    return True
        return False
```

64. 机器人的运动范围

> 地上有一个m行和n列的方格。一个机器人从坐标0,0的格子开始移动，每一次只能向左，右，上，下四个方向移动一格，但是不能进入行坐标和列坐标的数位之和大于k的格子。 例如，当k为18时，机器人能够进入方格（35,37），因为3+5+3+7 = 18。但是，它不能进入方格（35,38），因为3+5+3+8 = 19。请问该机器人能够达到多少个格子？

dfs拉，改改上一题得就行。

```
class Solution:
    def movingCount(self, threshold, rows, cols):
        visit = [[False] * cols for _ in range(rows)]

        def dfs(rows, cols, visit, i, j, k):
            if i < 0 or j < 0 or i >= rows or j >= cols or visit[i][j] or \
                    sum(map(int, [i for i in str(i) + str(j)])) > k:
                return
            visit[i][j] = True
            dfs(rows, cols, visit, i - 1, j, k)
            dfs(rows, cols, visit, i + 1, j, k)
            dfs(rows, cols, visit, i, j - 1, k)
            dfs(rows, cols, visit, i, j + 1, k)
```

```
        for i in range(rows):
            for j in range(cols):
                dfs(rows, cols, visit, i, j, threshold)
        return sum([sum(i) for i in visit])
```

结论：

刷到最后20道左右时，就有些浮躁，想不出题解就直接看我leetcode上的solution了，尴尬，刷题还是不能浮躁，得沉住气。

并且，刷完发现少了2道题，这，懒得找了，哈哈。

当然，我的所有代码都是python3写的，有的在牛客网网通过不了，因为牛客网只支持python2。by，全文中得其他语言所写代码几乎是网上借鉴得写法。

2018.1.5 中科院软件所