|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 题号 | 题目 | 知识点 | 难度 | 通过率 |
| JZ1 | 二维数组中的查找 | 数组 | 中等 | 25.24% |
| JZ2 | 替换空格 | 字符串 | 较难 | 48.32% |
| JZ3 | 从尾到头打印链表 | 链表 | 较难 | 26.65% |
| JZ4 | 重建二叉树 | 树dfs数组 | 中等 | 25.08% |
| JZ5 | 用两个栈实现队列 | 栈 | 简单 | 37.45% |
| JZ6 | 旋转数组的最小数字 | 二分 | 简单 | 33.61% |
| JZ7 | 斐波那契数列 | 数组 | 入门 | 32.69% |
| JZ8 | 跳台阶 | 递归 | 中等 | 37.28% |
| JZ9 | 变态跳台阶 | 贪心 | 简单 | 41.06% |
| JZ10 | 矩形覆盖 | 递归 | 中等 | 35.46% |
| JZ11 | 二进制中1的个数 | 数学 | 中等 | 34.76% |
| JZ12 | 数值的整数次方 | 数学 | 中等 | 32.66% |
| JZ13 | 调整数组顺序使奇数位于偶数前面 | 数组 | 较难 | 45.66% |
| JZ14 | 链表中倒数第k个结点 | 链表 | 中等 | 32.24% |
| JZ15 | 反转链表 | 链表 | 中等 | 32.41% |
| JZ16 | 合并两个排序的链表 | 链表 | 简单 | 28.01% |
| JZ17 | 树的子结构 | 树 | 较难 | 24.47% |
| JZ18 | 二叉树的镜像 | 树 | 简单 | 57.76% |
| JZ19 | 顺时针打印矩阵 | 数组 | 较难 | 18.81% |
| JZ20 | 包含min函数的栈 | 栈 | 较难 | 32.77% |
| JZ21 | 栈的压入、弹出序列 | 栈 | 中等 | 30.47% |
| JZ22 | 从上往下打印二叉树 | 队列树 | 困难 | 28.24% |
| JZ23 | 二叉搜索树的后序遍历序列 | 栈树 | 较难 | 24.52% |
| JZ24 | 二叉树中和为某一值的路径 | 树 | 较难 | 27.04% |
| JZ25 | 复杂链表的复制 | 链表 | 较难 | 21.86% |
| JZ26 | 二叉搜索树与双向链表 | 分治 | 中等 | 28.80% |
| JZ27 | 字符串的排列 | 字符串动态规划 | 较难 | 21.81% |
| JZ28 | 数组中出现次数超过一半的数字 | 数组哈希 | 简单 | 29.11% |
| JZ29 | 最小的K个数 | 堆排序分治 | 中等 | 23.51% |
| JZ30 | 连续子数组的最大和 | 动态规划分治 | 简单 | 37.26% |
| JZ31 | 整数中1出现的次数（从1到n整数中1出现的次数） | 数学 | 中等 | 34.54% |
| JZ32 | 把数组排成最小的数 | 数组 | 较难 | 28.96% |
| JZ33 | 丑数 | 数学二分 | 较难 | 22.40% |
| JZ34 | 第一个只出现一次的字符位置 | 字符串 | 简单 | 29.46% |
| JZ35 | 数组中的逆序对 | 数组 | 困难 | 15.97% |
| JZ36 | 两个链表的第一个公共结点 | 链表 | 中等 | 34.67% |
| JZ37 | 数字在排序数组中出现的次数 | 数组二分 | 中等 | 31.15% |
| JZ38 | 二叉树的深度 | 树 | 简单 | 47.45% |
| JZ39 | 平衡二叉树 | 树dfs | 简单 | 36.61% |
| JZ40 | 数组中只出现一次的数字 | 哈希位运算 | 中等 | 46.87% |
| JZ41 | 和为S的连续正数序列 | 穷举 | 中等 | 28.17% |
| JZ42 | 和为S的两个数字 | 双指针数组 | 中等 | 29.31% |
| JZ43 | 左旋转字符串 | 字符串 | 中等 | 32.44% |
| JZ44 | 翻转单词顺序列 | 字符串 | 较难 | 19.79% |
| JZ45 | 扑克牌顺子 | 字符串 | 中等 | 25.80% |
| JZ46 | 孩子们的游戏(圆圈中最后剩下的数) | 数学 | 中等 | 31.39% |
| JZ47 | 求1+2+3+...+n | 数学 | 中等 | 41.08% |
| JZ48 | 不用加减乘除做加法 | 数学 | 简单 | 44.49% |
| JZ49 | 把字符串转换成整数 | 字符串数学 | 较难 | 21.79% |
| JZ50 | 数组中重复的数字 | 数组 | 简单 | 49.82% |
| JZ51 | 构建乘积数组 | 数组 | 简单 | 38.97% |
| JZ52 | 正则表达式匹配 | 字符串 | 较难 | 28.31% |
| JZ53 | 表示数值的字符串 | 字符串 | 中等 | 34.58% |
| JZ54 | 字符流中第一个不重复的字符 | 字符串 | 中等 | 31.86% |
| JZ55 | 链表中环的入口结点 | 链表 | 中等 | 32.72% |
| JZ56 | 删除链表中重复的结点 | 链表 | 较难 | 20.48% |
| JZ57 | 二叉树的下一个结点 | 树 | 中等 | 30.08% |
| JZ58 | 对称的二叉树 | 树 | 困难 | 31.47% |
| JZ59 | 按之字形顺序打印二叉树 | 树栈 | 较难 | 24.50% |
| JZ60 | 把二叉树打印成多行 | 树bfs | 中等 | 31.69% |
| JZ61 | 序列化二叉树 | 队列树 | 较难 | 22.80% |
| JZ62 | 二叉搜索树的第k个结点 | 树 | 简单 | 24.30% |
| JZ63 | 数据流中的中位数 | 堆排序 | 中等 | 26.38% |
| JZ64 | 滑动窗口的最大值 | 堆双指针 | 较难 | 24.60% |
| JZ65 | 矩阵中的路径 | dfs | 中等 | 35.04% |
| JZ66 | 机器人的运动范围 | 数组 | 较难 | 23.21% |
| JZ67 | 剪绳子 | 数学 | 中等 | 32.22% |

目录

[简单 3](#_Toc26367)

[JZ5复杂链表的复制 3](#_Toc32586)

[JZ6二叉搜索树与双向链表 3](#_Toc25250)

[JZ7字符串的排列 3](#_Toc25826)

[JZ9最小的K个数 4](#_Toc1654)

[JZ16合并两个排序的链表 4](#_Toc24881)

[JZ18二叉树的镜像 4](#_Toc10574)

[JZ28数组中出现次数超过一半的数字 5](#_Toc17403)

[JZ30连续子数组的最大和 6](#_Toc26389)

[JZ34第一个只出现一次的字符位置 6](#_Toc32727)

[JZ38二叉树的深度 6](#_Toc23827)

[JZ39平衡二叉树 7](#_Toc1483)

[JZ48不用加减乘除做加法 7](#_Toc25995)

[JZ50数组中重复的数字 7](#_Toc9985)

[JZ51构建乘积数组 8](#_Toc13616)

[JZ62二叉搜索树的第k个结点 8](#_Toc29923)

[中等 9](#_Toc4407)

[JZ1二维数组中的查找 9](#_Toc7012)

[JZ4重建二叉树 9](#_Toc19223)

[JZ8跳台阶 10](#_Toc20029)

[JZ10矩形覆盖 10](#_Toc22330)

[JZ11二进制中1的个数 10](#_Toc26983)

[JZ12数值的整数次方 11](#_Toc15905)

[JZ14链表中倒数第k个结点 11](#_Toc9759)

[JZ15反转链表 11](#_Toc25553)

[JZ21栈的压入、弹出序列 12](#_Toc4395)

[JZ26二叉搜索树与双向链表 12](#_Toc16257)

[JZ29最小的K个数 13](#_Toc5186)

[JZ31整数中1出现的次数（从1到n整数中1出现的次数） 13](#_Toc11746)

[JZ36两个链表的第一个公共结点 14](#_Toc13256)

[JZ37数字在排序数组中出现的次数 14](#_Toc6775)

[JZ40数组中只出现一次的数字 15](#_Toc27444)

[JZ41和为S的连续正数序列 16](#_Toc24788)

[JZ42和为S的两个数字 16](#_Toc9690)

[JZ43左旋转字符串 17](#_Toc3270)

[JZ45扑克牌顺子 17](#_Toc32507)

[JZ46孩子们的游戏(圆圈中最后剩下的数) 17](#_Toc6308)

[JZ47求1+2+3+...+n 17](#_Toc2877)

[JZ53表示数值的字符串 18](#_Toc871)

[JZ54字符流中第一个不重复的字符 18](#_Toc16123)

[JZ55链表中环的入口结点 18](#_Toc9269)

[JZ57二叉树的下一个结点 19](#_Toc24596)

[JZ60把二叉树打印成多行 19](#_Toc3967)

[JZ63数据流中的中位数 20](#_Toc10010)

[JZ65矩阵中的路径 20](#_Toc16872)

[JZ67剪绳子 21](#_Toc4880)

[较难 21](#_Toc10127)

[JZ2替换空格 21](#_Toc19489)

[JZ3从尾到头打印链表 22](#_Toc2669)

[JZ13调整数组顺序使奇数位于偶数前面 22](#_Toc5359)

[JZ17树的子结构 22](#_Toc2282)

[JZ19顺时针打印矩阵 23](#_Toc1893)

[JZ20包含min函数的栈 24](#_Toc32217)

[JZ23二叉搜索树的后序遍历序列 24](#_Toc322)

[JZ24二叉树中和为某一值的路径 25](#_Toc19821)

[JZ25复杂链表的复制 25](#_Toc1448)

[JZ27字符串的排列 26](#_Toc26360)

[JZ32把数组排成最小的数 26](#_Toc17524)

[JZ33丑数 27](#_Toc2410)

[JZ44翻转单词顺序列 27](#_Toc15674)

[JZ49把字符串转换成整数 27](#_Toc2276)

[JZ52正则表达式匹配 28](#_Toc25352)

[JZ56删除链表中重复的结点 28](#_Toc1509)

[JZ59按之字形顺序打印二叉树 29](#_Toc3939)

[JZ61序列化二叉树 29](#_Toc19158)

[JZ64滑动窗口的最大值 31](#_Toc17954)

[JZ66机器人的运动范围 31](#_Toc27395)

[困难 32](#_Toc6406)

[JZ22从上往下打印二叉树 32](#_Toc22077)

[JZ35数组中的逆序对 32](#_Toc12617)

[JZ58对称的二叉树 33](#_Toc28351)

# 简单

## JZ5复杂链表的复制

题目描述：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。

思路：

1、队列是先进先出，栈是先进后出。因此用一个栈用来push，一个栈用来pop。用来pop的栈存放的是push栈弹出的元素，这样就可以用两个栈实现一个队列

package jz\_offer.src.easy.JZ5;

import java.util.Stack;

public class Solution {

Stack<Integer> stack1 = new Stack<Integer>();

Stack<Integer> stack2 = new Stack<Integer>();

public void push(int node) {

stack1.push(node);

}

public int pop() {

if (stack2.size() == 0) {

while (stack1.size() != 0) {

stack2.push(stack1.pop());

}

}

return stack2.pop();

}

}

## JZ6二叉搜索树与双向链表

题目描述：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。

NOTE：给出的所有元素都大于0，若数组大小为0，请返回0。

思路:

1、暴力解法 java.lang包下有很多方法，如Math、String，这道题用了Math.min，相当于查两个数小的那个

2、使用二分查找法

12>>1 相当于除以2，结果是6

15>>1，结果是7

写算法题的时候，位运算除以2更快，写这种更好

注意if

package jz\_offer.src.easy.JZ6;

public class Solution {

//暴力解法

/\*public int minNumberInRotateArray(int [] array) {

if (array.length == 0) {

return 0;

}

int ans = array[0];

for (int i = 1; i < array.length; i++) {

ans=Math.min(ans,array[i]);

}

return ans;

}\*/

//通过二分的方法

public static int minNumberInRotateArray(int[] array) {

if (array.length == 0) return 0;

int l = 0;

int r = array.length - 1;

while (l < r) {

int mid = (l + r) >> 1;

if (array[mid] > array[r]) l = mid + 1;

else if (array[mid] < array[r]) r = mid;

else r--;

}

return array[r];

}

public static void main(String[] args) {

int[] array = {3, 4, 5, 1, 2};

System.out.println(minNumberInRotateArray(array));

}

}

## JZ7字符串的排列

题目描述：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

思路：

1、一种采用递归，时间复杂度大，空间复杂度低。时间复杂度为O(2^n),空间复杂度O(1)

2、一种采用数组存储，把39项都存起来，时间复杂度就低了

3、最后一种最好，时间复杂度和空间复杂度都较低，我是采用这种。时间复杂度为O(n),空间复杂度O(1)

package jz\_offer.src.easy.JZ7;

public class Solution {

public int Fibonacci(int n) {

if (n == 0) return 0;

if (n == 1) return 1;

int one = 0;

int two = 1;

int sum = 0;

for (int i = 2; i <= n; i++) {

sum = one + two;

one = two;

two = sum;

}

return sum;

}

}

## JZ9最小的K个数

题目描述：

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。

思路：

1、这道题通过找规律，可以发现f(n)=2\*f(n-1)

可以用数学方法求解，理解这道题跳台阶的意思，得到这个等式

也可以用找规律法，写出前几个，就能发现这个规律

package jz\_offer.src.easy.JZ9;

public class Solution {

public int JumpFloorII(int target) {

int sum = 1;

if (target == 0 || target == 1) return 1;

for (int i = 1; i < target; i++) {

sum \*= 2;

}

return sum;

}

}

## JZ16合并两个排序的链表

题目描述：

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。

思路：

1、了解到新的规则

int a=2;

int b=a;

a=3;

这时输出a,b均为3

引用对象也是一样的，会同时改变

package jz\_offer.src.easy.JZ16;

public class Solution {

public class ListNode {

int val;

ListNode next = null;

public ListNode(int val) {

this.val = val;

}

}

public ListNode Merge(ListNode list1, ListNode list2) {

ListNode headNode = new ListNode(0);

ListNode moveNode = headNode; //moveNode是headNode的引用,改变moveNode，headNode也会对应改变

while (list1 != null && list2 != null) {

if (list1.val < list2.val) {

moveNode.next = list1;

list1 = list1.next;

} else {

moveNode.next = list2;

list2 = list2.next;

}

moveNode = moveNode.next;

}

if (list1 == null) moveNode.next = list2;

if (list2 == null) moveNode.next = list1;

return headNode.next;

}

}

## JZ18二叉树的镜像

题目描述：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

思路：

1、return不跟任何参数，表示直接将代码结束

2、使用递归交换左右节点，root.left表示左节点，root.right表示右节点

3、两个元素交换就是设置一个临时变量

package jz\_offer.src.easy.JZ18;

public class Solution {

public class TreeNode {

int val = 0;

TreeNode left = null;

TreeNode right = null;

public TreeNode(int val) {

this.val = val;

}

}

public void Mirror(TreeNode root) {

if (root == null) return;

TreeNode temp = root.left;

root.left = root.right;

root.right = temp;

Mirror(root.left);

Mirror(root.right);

}

}

## JZ28数组中出现次数超过一半的数字

题目描述：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。

思路：

1、哈希法，使用HashMap集合。引入外部包，用import

2、候选人法,采用断点可以理解，推荐还是用哈希法吧

哈希HashMap，键唯一，值可以有多个相同

package jz\_offer.src.easy.JZ28;

public class Solution {

//哈希法

/\*public int MoreThanHalfNum\_Solution(int [] array) {

Map<Integer,Integer> map=new HashMap<>();

for (int e : array) {

if(map.get(e)==null){

map.put(e,1);

}else{

map.put(e,map.get(e)+1);

}

if(map.get(e)>array.length/2)

return e;

}

return 0;

}\*/

//候选人法

public static int MoreThanHalfNum\_Solution(int[] array) {

int people = 0, tickets = 0;

// 轮流投票

for (int e : array) {

// 新人上任

if (tickets == 0) {

people = e;

}

// 支持自己，反对别人

if (people == e) {

tickets++;

} else {

tickets--;

}

}

// 检测该数是否为众数

tickets = 0;

for (int e : array) {

if (e == people && ++tickets > array.length / 2) return people;

}

return 0;

}

public static void main(String[] args) {

int[] array = {1, 2, 3, 2, 2, 2, 5, 4, 2};

System.out.println(MoreThanHalfNum\_Solution(array));

}

}

## JZ30连续子数组的最大和

题目描述：输入一个整型数组，数组里有正数也有负数。数组中的一个或连续多个整数组成一个子数组。

求所有子数组的和的最大值。要求时间复杂度为 O(n).

思路：

1、用题目的数组存储数组的值，当遇到数组和为负时，就不用计入情况了。因为当前元素加上一个负数，只会更小，这时重新计数。

2、每次都将计数与max比较，最后得到的就是数组的和的最大值

3、时间复杂度为O(n)

4、相当于拿一个数组存放和的最大值，当前面的值为负数，加上肯定小了，这时就不用考虑，否则可以考虑和最大值比较，看看是不是成为了新的最大值

package jz\_offer.src.easy.JZ30;

public class Solution {

public int FindGreatestSumOfSubArray(int[] array) {

int max = array[0];

for (int i = 1; i < array.length; i++) {

array[i] += Math.max(array[i - 1], 0);

max = Math.max(max, array[i]);

}

return max;

}

}

## JZ34第一个只出现一次的字符位置

题目描述：

在一个字符串(0<=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, 如果没有则返回 -1（需要区分大小写）.（从0开始计数）

思路：

1、使用hashmap，记录character和integer。最后再遍历字符串，第一个输出1的即为第一次出现的字符

package jz\_offer.src.easy.JZ34;

import java.util.\*;

public class Solution {

public int FirstNotRepeatingChar(String str) {

if (str.length() == 0) return -1;

HashMap<Character, Integer> map = new HashMap<>();

for (int i = 0; i < str.length(); i++) {

if (!map.containsKey(str.charAt(i))) {

map.put(str.charAt(i), 1);

} else {

map.put(str.charAt(i), map.get(str.charAt(i)) + 1);

}

}

for (int i = 0; i < str.length(); i++) {

if (map.get(str.charAt(i)) == 1) return i;

}

return -1;

}

}

## JZ38二叉树的深度

题目描述：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。

{1,2,3,4,5,#,6,#,#,7} 返回值：4

思路：

1、 6 4 2 3 # # # # 5 1 # # 7 # # 学会画出这样的二叉树是什么样的，画二叉树都是先画左边再画右边。

2、前序遍历 中序遍历 后序遍历 3种是不一样的。

两种写法：递归写法

非递归写法：使用队列层次遍历

queue.add()增加元素 queue.poll()删除头节点

package jz\_offer.src.easy.JZ38;

import java.util.\*;

public class Solution {

public class TreeNode {

int val = 0;

TreeNode left = null;

TreeNode right = null;

public TreeNode(int val) {

this.val = val;

}

}

//递归解法

public int TreeDepth1(TreeNode root) {

if (root == null) return 0;

//求出左子树的最大深度

int left = TreeDepth1(root.left);

//求出右子树的最大深度

int right = TreeDepth1(root.right);

return Math.max(left, right) + 1;

}

//借助队列，层次遍历

public int TreeDepth2(TreeNode root) {

if (root == null) return 0;

Queue<TreeNode> queue = new LinkedList<TreeNode>();

queue.add(root);

int high = 0;

int size;

TreeNode node;

while (queue.size() != 0) {

size = queue.size();

while (size != 0) {

node = queue.poll();

if (node.left != null)

queue.add(node.left);

if (node.right != null)

queue.add(node.right);

size--;

}

high++;

}

return high;

}

}

## JZ39平衡二叉树

题目描述：输入一棵二叉树，判断该二叉树是否是平衡二叉树。

在这里，我们只需要考虑其平衡性，不需要考虑其是不是排序二叉树

平衡二叉树（Balanced Binary Tree），具有以下性质：

它是一棵空树或它的左右两个子树的高度差的绝对值不超过1，并且左右两个子树都是一棵平衡二叉树。

{1,2,3,4,5,6,7} 返回值：true

思路：

1、JZ38求树高度的方法可以借鉴过来，再加上左右两个子树都是平衡二叉树，这棵树才是平衡二叉树或者空树也是。

2、方法中的变量一定要初始化(赋初值)，类的属性变量可以不初始化。

package jz\_offer.src.easy.JZ39;

public class Solution {

public class TreeNode {

int val = 0;

TreeNode left = null;

TreeNode right = null;

public TreeNode(int val) {

this.val = val;

}

}

public int TreeDepth(TreeNode root) {

if (root == null) return 0;

if (root.left == null && root.right == null) return 1;

return Math.max(TreeDepth(root.left), TreeDepth(root.right)) + 1;

}

public boolean IsBalanced\_Solution(TreeNode root) {

if (root == null) return true;

if (Math.abs(TreeDepth(root.left) - TreeDepth(root.right)) > 1) return false;

return IsBalanced\_Solution(root.left) && IsBalanced\_Solution(root.right);

}

}

## JZ48不用加减乘除做加法

题目描述：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。

思路：

1、异或，两个元素不同，返回1。

2、进位，&，只有都为1时才为1，表示进位相当于左移1位就行。

3、最后把不进位和 进位 相加即可。

用的是异或 ^相加，因为有个判断条件num2已经为0了

package jz\_offer.src.easy.JZ48;

public class Solution {

public int Add(int num1, int num2) {

int temp = 0;

if (num2 == 0) return num1;

while (num2 != 0) {

temp = num1 ^ num2; //异或，计算无进位和

num2 = (num1 & num2) << 1; //计算进位

num1 = temp;

}

return temp;

}

}

## JZ50数组中重复的数字

题目描述：

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。

请找出数组中第一个重复的数字。 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。

返回描述：

如果数组中有重复的数字，函数返回true，否则返回false。

如果数组中有重复的数字，把重复的数字放到参数duplication[0]中。

（ps:duplication已经初始化，可以直接赋值使用。）

思路：

1、使用HashMap，因为它的键是不能重复的，使用containsKey()方法

package jz\_offer.src.easy.JZ50;

import java.util.\*;

public class Solution {

public boolean duplicate(int numbers[], int length, int[] duplication) {

Map<Integer, Integer> map = new HashMap<>();

for (int i = 0; i < length; i++) {

if (map.containsKey(numbers[i])) {

duplication[0] = numbers[i];

return true;

} else {

map.put(numbers[i], null);

}

}

return false;

}

}

## JZ51构建乘积数组

题目描述：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。

（注意：规定B[0] = A[1] \* A[2] \* ... \* A[n-1]，B[n-1] = A[0] \* A[1] \* ... \* A[n-2];）对于A长度为1的情况，B无意义，故而无法构建，因此该情况不会存在。

输入：[1,2,3,4,5] 返回值：[120,60,40,30,24]

思路：

1、B[i]即为除去A[i]所有的值相乘

2、可以在计算B[i]的值的时候，把A[i]暂时先设为1，再用for计算，之后再还原即可

package jz\_offer.src.easy.JZ51;

public class Solution {

public int[] multiply(int[] A) {

int[] B = new int[A.length];

int temp = 0;

for (int i = 0; i < A.length; i++) {

temp = A[i];

B[i] = 1;

A[i] = 1;

for (int k : A) {

B[i] \*= k;

}

A[i] = temp;

}

return B;

}

}

## JZ62二叉搜索树的第k个结点

题目描述：给定一棵二叉搜索树，请找出其中的第k小的结点。

思路：

1、二叉搜索树的中序遍历可以实现排序

2、但一般二叉树都要用两个遍历，才能还原这颗二叉树。前中或中后

package jz\_offer.src.easy.JZ62;

import java.util.\*;

public class Solution {

public class TreeNode {

int val = 0;

TreeNode left = null;

TreeNode right = null;

public TreeNode(int val) {

this.val = val;

}

}

TreeNode KthNode(TreeNode pRoot, int k) {

if (pRoot == null || k < 0) return null;

TreeNode cur = pRoot;

Stack<TreeNode> stack = new Stack<>();

//while内为中序遍历

while (!stack.isEmpty() || cur != null) {

if (cur != null) {

stack.push(cur);

cur = cur.left;

} else {

cur = stack.pop();

if (--k == 0) return cur;

cur = cur.right;

}

}

return null;

}

}

# 中等

## JZ1二维数组中的查找

题目描述：

在一个二维数组中（每个一维数组的长度相同），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。

输入：7,[[1,2,8,9],[2,4,9,12],[4,7,10,13],[6,8,11,15]] 返回：true

思路：

1、时间复杂度：算法中基本操作的执行次数

2、一种暴力求解

3、另一种由于行从左到右递增，列从上到下递增，可以用这个性质。这样时间复杂度就变成行+列的次数，时间复杂度大大减小

package jz\_offer.src.medium.JZ1;

public class Solution {

//暴力法

/\*public boolean Find(int target, int [][] array) {

if(array.length==0)

return false;

for(int i=0;i<array.length;i++){

for(int j=0;j<array[0].length;j++){

if(array[i][j]==target)

return true;

}

}

return false;

}\*/

//时间复杂度相比暴力法较低

public boolean Find(int target, int[][] array) {

int row = array.length;

if (row == 0) return false;

int col = array[0].length;

if (col == 0) return false;

int rows = row - 1;

int cols = 0;

while (rows >= 0 && cols < col) {

if (target < array[rows][cols]) rows--;

else if (target > array[rows][cols]) cols++;

else return true;

}

return false;

}

}

## JZ4重建二叉树

题目描述：

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。

输入：[1,2,3,4,5,6,7],[3,2,4,1,6,5,7]

返回值：{1,2,5,3,4,6,7}

思路:

1、二叉树、二叉树的前序遍历、二叉树的中序遍历是不一样的，返回值相当于二叉树按顺序输出。

2、前序遍历、中序遍历，先把根节点和左右两节点写出来，再补充节点，就可以得到序列。

3、这道题采用递归的思想，从前序遍历和中序遍历得到根节点，类似的得到左右子树的根节点，以此递归。

根据前序和中序遍历的性质，通过递归，重建二叉树。首先找到根节点，然后找到对应的左右子树。

package jz\_offer.src.medium.JZ4;

import java.util.\*;

public class Solution {

public class TreeNode {

//val代表树的根结点的值

int val = 0;

TreeNode left = null;

TreeNode right = null;

public TreeNode(int val) {

this.val = val;

}

}

//pre 树的前序 in树的中序 根据递归还原出整个树

public TreeNode reConstructBinaryTree(int[] pre, int[] in) {

if (pre.length == 0 || in.length == 0) return null;

TreeNode root = new TreeNode(pre[0]);

for (int i = 0; i < in.length; i++) {

if (pre[0] == in[i]) {

//copyOfRange左闭右开

root.left = reConstructBinaryTree(Arrays.copyOfRange(pre, 1, 1 + i), Arrays.copyOfRange(in, 0, i));

root.right = reConstructBinaryTree(Arrays.copyOfRange(pre, i + 1, pre.length), Arrays.copyOfRange(in, i + 1, in.length));

break;

}

}

return root;

}

}

## JZ8跳台阶

题目描述：

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（先后次序不同算不同的结果）。

思路：

1、发现规律是，1级台阶 1 2级台阶2 3级之后就等于前面两个相加即可。

2、有两种求解方法

一种是递归，时间复杂度O(2^n)，面试不要用这种。

一种是设一个变量result，返回result即可，时间复杂度是O(n)，使用这种。

package jz\_offer.src.medium.JZ8;

public class Solution {

public int JumpFloor(int target) {

if (target == 1) return 1;

if (target == 2) return 2;

int one = 1;

int two = 2;

int result = 0;

for (int i = 2; i < target; i++) {

result = one + two;

one = two;

two = result;

}

return result;

}

}

## JZ10矩形覆盖

题目描述：

我们可以用2\*1的小矩形横着或者竖着去覆盖更大的矩形。

请问用n个2\*1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？

比如n=3时，2\*3的矩形块有3种覆盖方法：

思路：

1、找规律，可以用递归或者递推，一般使用递推。因为递归的时间复杂度比较高O(2^n)

2、写出n=1到5时的情况，就可以找出规律

package jz\_offer.src.medium.JZ10;

public class Solution {

public int RectCover(int target) {

if (target == 1 || target == 2) return target;

int one = 1;

int two = 2;

int result = 0;

for (int i = 2; i < target; i++) {

result = one + two;

one = two;

two = result;

}

return result;

}

}

## JZ11二进制中1的个数

题目描述：输入一个整数，输出该数32位二进制表示中1的个数。其中负数用补码表示。

思路：

1、正数的原码、反码、补码是一样的，负数则有区别

计算机的整数计算用反码表示，因为反码可以表示加减，范围则是-128-127

采用反码计算，可以轻松得到答案

符号位为0，为正；符号位为1，为负

负数的反码就是原码取反，补码就是反码加1

原码首位为0是正数，首位为1是负数，因此这道题可以通过左移判断

package jz\_offer.src.medium.JZ11;

public class Solution {

public int NumberOf1(int n) {

int count = 0;

if (n == 0) return 0;

while (n != 0) {

if (n < 0) ++count;

n = (n << 1);

}

return count;

}

//另一种方法，返回整数的二进制数，然后进行判断

//数值用==比较，字符串用equals比较，字符char用=='1'

/\*public int NumberOf1(int n) {

String str=Integer.toBinaryString(n);

int count=0;

for(int i=0;i<str.length();i++){

if(str.charAt(i)=='1')

++count;

}

return count;

}\*/

}

## JZ12数值的整数次方

题目描述：

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。

保证base和exponent不同时为0

输入：2,3 返回值：8.00000

思路：

1、1种使用暴力法，1种使用库函数

package jz\_offer.src.medium.JZ12;

public class Solution {

public double Power1(double base, int exponent) {

return Math.pow(base, exponent);

}

public double Power2(double base, int exponent) {

double pow = 1.00;

if (base == 0 && exponent != 0) return 0.00;

if (exponent == 0 && base != 0) return 1.00;

if (exponent > 0) {

for (int i = 0; i < exponent; i++) {

pow \*= base;

}

} else if (exponent < 0) {

for (int i = 0; i < Math.abs(exponent); i++) {

pow /= base;

}

}

return pow;

}

}

## JZ14链表中倒数第k个结点

题目描述：

输入一个链表，输出该链表中倒数第k个结点。

输入：1,{1,2,3,4,5} 返回值：{5}

思路：

1、还是利用stack后进先出的性质，返回链表的倒数第k个结点

2、stack具有两个方法，push和pop

package jz\_offer.src.medium.JZ14;

import java.util.\*;

public class Solution {

public class ListNode {

int val;

ListNode next = null;

ListNode(int val) {

this.val = val;

}

}

public ListNode FindKthToTail(ListNode head, int k) {

Stack<ListNode> stack = new Stack<>();

ListNode node = null;

if (head == null) return null;

while (head != null) {

stack.push(head);

head = head.next;

}

if (k > stack.size()) return null;

for (int i = 1; i <= k; i++) {

node = stack.pop();

}

return node;

}

}

## JZ15反转链表

题目描述：

输入一个链表，反转链表后，输出新链表的表头。

思路：

1、ListNode包含数据域和指针域

2、链表的反转包含4个步骤，需要两个临时节点记录当前节点的前节点和后节点,通过返回pre，因为最后状态下的head是赋值成pre的，所以返回pre，就能返回这个反转链表。这四个步骤第一次是有点难理解的，多点耐心，还是能分析出来的。有一个head.next=pre的操作，这步实现了反转，需要一个临时节点存储head，这时候再赋值

3、这道题就是要一步一步画图分析，才能理解。需要记住链表反转有4歩操作，需要定义两个指针。使用while循环判断head是否为空，作为跳出条件。

package jz\_offer.src.medium.JZ15;

public class Solution {

public class ListNode {

int val;

ListNode next = null;

public ListNode(int val) {

this.val = val;

}

}

public ListNode ReverseList(ListNode head) {

ListNode pre = null;

ListNode after = null;

while (head != null) {

after = head.next;

head.next = pre;

pre = head;

head = after;

}

return pre;

}

}

## JZ21栈的压入、弹出序列

题目描述：

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可能为该栈的弹出顺序。

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）

输入：[1,2,3,4,5],[4,3,5,1,2] 返回值：false

思路：

1、push 入栈 pop出栈 peek返回栈顶元素，但不移除

2、创建一个stack，使用条件判断，模拟栈的入栈和出栈。如果最后栈为空，表示出栈顺序是对的，返回true

package jz\_offer.src.medium.JZ21;

import java.util.\*;

public class Solution {

public boolean IsPopOrder(int[] pushA, int[] popA) {

if (popA.length == 0 || pushA.length != popA.length) return false;

Stack<Integer> stack = new Stack<>();

int j = 0;

for (int k : pushA) {

stack.push(k);

while (!stack.isEmpty() && stack.peek() == popA[j]) {

stack.pop();

j++;

}

}

return stack.isEmpty();

}

}

## JZ26二叉搜索树与双向链表

题目描述：

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。

思路：

1、二叉查找树，又叫二叉搜索树、二叉排序树。

它的性质：左子树上所有结点的值均小于它的根结点的值，右子树上所有结点的值均大于它的根结点的值。

2、双向链表把它类似于左右节点。

package jz\_offer.src.medium.JZ26;

import java.util.\*;

public class Solution {

public class TreeNode {

int val;

TreeNode left = null;

TreeNode right = null;

public TreeNode(int val) {

this.val = val;

}

}

public TreeNode Convert(TreeNode pRootOfTree) {

if (pRootOfTree == null) return null;

List<TreeNode> list = new ArrayList<>();

Convert(pRootOfTree, list);

return Convert(list);

}

//中序遍历，将TreeNode存入list集合

public void Convert(TreeNode pRootOfTree, List<TreeNode> list) {

if (pRootOfTree.left != null) Convert(pRootOfTree.left, list);

list.add(pRootOfTree);

if (pRootOfTree.right != null) Convert(pRootOfTree.right, list);

}

//添加节点间的指针关系即可，这样就得到双向链表

public TreeNode Convert(List<TreeNode> list) {

for (int i = 0; i < list.size() - 1; i++) {

list.get(i).right = list.get(i + 1);

list.get(i + 1).left = list.get(i);

}

return list.get(0);

}

}

## JZ29最小的K个数

题目描述：

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。

输入：[4,5,1,6,2,7,3,8],4 返回值：[1,2,3,4]

思路：

1、使用快速排序，快速排序算法需要背下来

2、打印数组的话，要用Arrays.toString或for循环才可以

3、如果需要交换数组的两个数，需要传入三个参数，一个数组，两个下标，这样才能成功交换数组元素

package jz\_offer.src.medium.JZ29;

import java.util.\*;

public class Solution {

public ArrayList<Integer> GetLeastNumbers\_Solution(int[] input, int k) {

ArrayList<Integer> list = new ArrayList<>();

if (k > input.length) return list;

quickSort(input, 0, input.length - 1);

for (int i = 0; i < k; i++) {

list.add(input[i]);

}

return list;

}

public int[] quickSort(int[] num, int lo, int hi) {

if (lo > hi) return num;

else {

//将数列最左边第一个数字作为基准数

int left = lo;

int right = hi;

int baseNum = num[lo];

while (right > left) {

//第二步：右边指针找到大于基准数的就停下

while (num[right] >= baseNum && right > left) {

right--;

}

//第二步：左边指针找到小于基准数的就停下

while (num[left] <= baseNum && right > left) {

left++;

}

//交换两个指针最终标记的数字

if (right > left) swap(num, left, right);

}

//当左右两边指针重合时，将基准数与指针指向数字交换

swap(num, left, lo);

//指针左半边递归，以进来的数组的左边为界，右边是左右指针相同时左边一个

quickSort(num, lo, left - 1);

//右边同理

quickSort(num, left + 1, hi);

return num;

}

}

//swap方法：将数组中leftPos和rightPos上的两个数值进行交换

public void swap(int[] num, int left, int right) {

int temp = num[left];

num[left] = num[right];

num[right] = temp;

}

}

## JZ31整数中1出现的次数（从1到n整数中1出现的次数）

题目描述：

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（从1 到 n 中1出现的次数）。

思路：

1、有暴力解法和按位分析两种解法，目前采用暴力解法

2、按位分析主要就是看个位、十位、百位上1的存在，目前还没写解法

package jz\_offer.src.medium.JZ31;

public class Solution {

public int NumberOf1Between1AndN\_Solution(int n) {

int count = 0;

for (int i = 1; i <= n; i++) {

int j = i;

while (j > 0) {

if (j % 10 == 1) {

count++;

break;

}

j /= 10;

}

}

return count;

}

}

## JZ36两个链表的第一个公共结点

题目描述：

输入两个链表，找出它们的第一个公共结点。

（注意因为传入数据是链表，所以错误测试数据的提示是用其他方式显示的，保证传入数据是正确的

思路：

1、使用hashmap存储head1的数据域和指针域，head2看看map的key有没相等的，如果true，则是第一个公共结点

2、这里是常见的类的方法

list.add

list.get(i)

list.contains() 查找有没这个元素

map.put(key,value)

得到value的值 map.get(key)

map.containsKey

map.containsValue

stack.push

stack.pop

stack.peek 返回栈顶元素，但不弹出

String.valueOf() 将其他类型转换为字符串

String.length()

package jz\_offer.src.medium.JZ36;

import java.util.\*;

public class Solution {

public class ListNode {

int val;

ListNode next = null;

ListNode(int val) {

this.val = val;

}

}

public ListNode FindFirstCommonNode(ListNode pHead1, ListNode pHead2) {

HashMap<ListNode, Integer> map = new HashMap<>();

while (pHead1 != null) {

map.put(pHead1, null);

pHead1 = pHead1.next;

}

while (pHead2 != null) {

if (map.containsKey(pHead2)) return pHead2;

pHead2 = pHead2.next;

}

return null;

}

}

## JZ37数字在排序数组中出现的次数

题目描述：

统计一个数字在升序数组中出现的次数。

思路：

1、暴力解法

2、使用二分查找法，因为是有序的

3、一个数组为空时返回0，用的是array.length==0这个判断条件。这个要记住，错过很多次了！！！

package jz\_offer.src.medium.JZ37;

public class Solution {

//暴力解法

/\*public int GetNumberOfK(int [] array , int k) {

int count=0;

for(int i=0;i<array.length;i++){

if(array[i]==k)

count++;

}

return count;

}\*/

//二分查找法

public int GetNumberOfK(int[] array, int k) {

if (array.length == 0) return 0;

int firstPositon = getFirstPositionOfK(array, k);

int lastPositon = getLastPositionOfK(array, k);

if (array[firstPositon] != k) return 0;

return lastPositon - firstPositon + 1;

}

public int getFirstPositionOfK(int[] array, int k) {

int l = 0;

int r = array.length - 1;

while (l < r) {

int mid = (l + r) >> 1;

if (array[mid] < k) {

l = mid + 1;

} else if (array[mid] > k) {

r = mid - 1;

} else {

if (mid - 1 >= 0 && array[mid - 1] == array[mid]) {

r = mid - 1;

} else {

return mid;

}

}

}

return l;

}

public int getLastPositionOfK(int[] array, int k) {

int l = 0;

int r = array.length - 1;

while (l < r) {

int mid = (l + r) >> 1;

if (array[mid] < k) {

l = mid + 1;

} else if (array[mid] > k) {

r = mid - 1;

} else {

if (mid + 1 <= array.length && array[mid + 1] == array[mid]) {

l = mid + 1;

} else {

return mid;

}

}

}

return l;

}

}

## JZ40数组中只出现一次的数字

题目描述：

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。

思路：

1、java集合分为collection和map map的键不可重复

collection分为set、list list有序可重复，set无序不可重复

collection添加元素是add，map添加元素是put

两者删除元素都是remove

list.remove传入的是下标，set.remove直接传入元素

list和set都是contains方法，map是containsKey方法

package jz\_offer.src.medium.JZ40;

import java.util.\*;

public class Solution {

public void FindNumsAppearOnce(int[] array, int num1[], int num2[]) {

TreeSet<Integer> set = new TreeSet<>();

for (Integer a : array) {

if (set.contains(a)) {

set.remove(a);

} else {

set.add(a);

}

}

int i = 0;

for (Integer b : set) {

if (i == 0) {

num1[0] = b;

i = 1;

} else {

num2[0] = b;

}

}

}

}

## JZ41和为S的连续正数序列

题目描述：

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!

思路：

1、移动窗口的左右指针，一开始左指针指向0，右指针指向1，通过判断与sum的大小关系，移动左右指针

2、需要注意的就是添加到ans中的list，需要新建一个对象

package jz\_offer.src.medium.JZ41;

import java.util.\*;

public class Solution {

public static ArrayList<ArrayList<Integer>> FindContinuousSequence(int sum) {

ArrayList<ArrayList<Integer>> ans = new ArrayList<>();

ArrayList<Integer> list = new ArrayList<>();

int l = 1;

int r = 2;

list.add(l);

list.add(r);

while (l < (sum + 1) / 2) {

int temp = cal(l, r);

if (temp < sum) {

r++;

list.add(r);

} else if (temp == sum) {

//加入到ans中需要新生成一个对象，不然会随着引用list不断改变

ans.add(new ArrayList<>(list));

r++;

list.add(r);

} else {

l++;

list.remove(0);

}

}

return ans;

}

public static int cal(int l, int r) {

return (l + r) \* (r - l + 1) / 2;

}

public static void main(String[] args) {

System.out.println(FindContinuousSequence(3));

}

}

## JZ42和为S的两个数字

题目描述：

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，使得他们的和正好是S，如果有多对数字的和等于S，输出两个数的乘积最小的。

思路：

1、依旧是采用左右指针法，左指针指向0，右指针指向数组末尾

2、list还有一个clear()方法，清空集合

3、打印集合是[]的形式

package jz\_offer.src.medium.JZ42;

import java.util.\*;

public class Solution {

public ArrayList<Integer> FindNumbersWithSum(int[] array, int sum) {

ArrayList<Integer> list = new ArrayList<>();

int l = 0;

int r = array.length - 1;

while (l < r) {

int a = array[l] + array[r];

if (a < sum) {

l++;

} else if (a > sum) {

r--;

} else {

//易证首尾指针，第一组遇到的是最小的，因此直接返回

list.add(array[l]);

list.add(array[r]);

return list;

}

}

return list;

}

}

## JZ43左旋转字符串

题目描述：

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！

思路：

1、使用String提供的substring()方法即可，左闭右开

package jz\_offer.src.medium.JZ43;

public class Solution {

public String LeftRotateString(String str, int n) {

if (str.length() < n) return "";

return str.substring(n, str.length()) + str.substring(0, n);

}

}

## JZ45扑克牌顺子

题目描述：

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，如果牌能组成顺子就输出true，否则就输出false。为了方便起见,你可以认为大小王是0。

思路：

1、TreeSet的特点是有序，不可重复。可以用来解决此题，如果set的最大值减最小值小于5，则代表可以组成顺子

2、TreeSet的特点还有不会存入重复的数字，因此可以加一个set.size()+n的判断，如果不等于5，返回false

3、TreeSet的底层实现是二叉树

4、TreeSet有序不可重复，它是按照从小到大的顺序给你排好了

5、TreeSet按照自然顺序，LinkedHashSet按照插入顺序

package jz\_offer.src.medium.JZ45;

import java.util.\*;

public class Solution {

public boolean isContinuous(int[] numbers) {

if (numbers.length != 5) return false;

TreeSet<Integer> set = new TreeSet<>();

int num = 0;

for (int number : numbers) {

if (number == 0) num++;

else set.add(number);

}

if ((set.size() + num) != 5) return false;

return set.last() - set.first() < 5;

}

}

## JZ46孩子们的游戏(圆圈中最后剩下的数)

题目描述：

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)，如果没有小朋友，请返回-1

输入：5,3 返回值：3

思路：

1、使用linkedlist，底层是双向链表，查询慢，增删块。list.remove(i)根据下标删除

2、根据下标移除元素，通过p+m-1除以list.size可以判断出哪个小朋友要出列

3、这道题删除用的比较多，所以用LinkedList

package jz\_offer.src.medium.JZ46;

import java.util.\*;

public class Solution {

public int LastRemaining\_Solution(int n, int m) {

if (n <= 0 || m <= 0) return -1;

List<Integer> list = new LinkedList<>();

for (int i = 0; i < n; i++) {

list.add(i);

}

int p = 0;

while (list.size() != 1) {

p = (p + m - 1) % list.size();

list.remove(p);

}

return list.get(0);

}

}

## JZ47求1+2+3+...+n

题目描述：

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）。

思路：

1、题目要求不能用乘除法，因此可以采用递归的方式，但递归又要用到if，我们可以把if改成短路与的方式。

逻辑与(&)和短路与(&&)

package jz\_offer.src.medium.JZ47;

public class Soluition {

public int Sum\_Solution(int n) {

//实际上这样做就是你投机取巧的方式

//因为给定的等差数列的和

//n/2+n方/2

int sum = (int) Math.pow(n, 2) + n;

// 右移一位除以2

return sum >> 1;

}

}

## JZ53表示数值的字符串

题目描述：

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。

思路：

1、使用java自带的函数Double.parseDouble，如果抛出异常，返回false。还有一种情况需要注意，末尾带d或f的，java默认是不会抛出异常的，但我们不把这些字符串当成数值，因此要自己写个if语句

package jz\_offer.src.medium.JZ53;

public class Solution {

public boolean isNumeric(char[] str) {

String num = new String(str);

if (num.endsWith("f") || num.endsWith("F") || num.endsWith("d") || num.endsWith("D"))

return false;

try {

Double.parseDouble(num);

} catch (Exception e) {

return false;

}

return true;

}

}

## JZ54字符流中第一个不重复的字符

题目描述：

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"。

思路：

1、使用StringBuffer先把所有字符先存入字符串，同时map已经存储好了key、value值，进行判断即可，如果等于1，直接返回字符即可，这就是第一次出现的字符

2、提交的代码有多个括号时，要注意括号的位置，括号所包含的范围是不是对的。有时候粗心，括号使用错误，会导致代码提交失败

3、使用HashMap键不能重复，值如果为Integer，也可以传入null

4、HashMap的键不能重复，值可以重复

package jz\_offer.src.medium.JZ54;

import java.util.\*;

public class Solution {

public Map<Character, Integer> map = new HashMap<>();

public StringBuffer str = new StringBuffer();

public int index = 0;

public void Insert(char ch) {

str.append(ch);

if (map.containsKey(ch)) {

map.put(ch, map.get(ch) + 1);

} else {

map.put(ch, 1);

}

}

public char FirstAppearingOnce() {

while (index < str.length()) {

if (map.get(str.charAt(index)) == 1) return str.charAt(index);

index++;

}

return '#';

}

}

## JZ55链表中环的入口结点

题目描述：

给一个链表，若其中包含环，请找出该链表的环的入口结点，否则，输出null。

思路：

1、使用HashMap，如果键重复直接返回即可

2、链表需要自己定义一个类ListNode，树需要自己定义一个类TreeNode

package jz\_offer.src.medium.JZ55;

import java.util.\*;

public class Solution {

public class ListNode {

int val;

ListNode next = null;

ListNode(int val) {

this.val = val;

}

}

public ListNode EntryNodeOfLoop(ListNode pHead) {

if (pHead == null) return null;

Map<ListNode, Integer> map = new HashMap<>();

map.put(pHead, null);

while (pHead.next != null) {

pHead = pHead.next;

if (map.containsKey(pHead)) return pHead;

else map.put(pHead, null);

}

return null;

}

}

## JZ57二叉树的下一个结点

题目描述：

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。

思路：

1、题目给出的节点包含指向父结点的指针，因此我们找出根结点，然后使用递归，将结点按中序遍历存入list。最后使用for循环，判断是否与给出的节点相等，然后返回下一个结点即可

2、前序遍历preorder 中序遍历inorder 后序遍历postorder

3、中序遍历传入根节点，然后使用递归，先遍历左边，然后将node加入链表，再遍历右边

package jz\_offer.src.medium.JZ57;

import java.util.\*;

public class Solution {

public class TreeLinkNode {

int val = 0;

TreeLinkNode left = null;

TreeLinkNode right = null;

//指向父亲结点的指针

TreeLinkNode next = null;

TreeLinkNode(int val) {

this.val = val;

}

}

public List<TreeLinkNode> list = new ArrayList<>();

public TreeLinkNode GetNext(TreeLinkNode pNode) {

TreeLinkNode node = pNode;

while (node.next != null) {

node = node.next;

}

inOrder(node);

for (int i = 0; i < list.size() - 1; i++) {

if (pNode == list.get(i)) return list.get(i + 1);

}

return null;

}

public void inOrder(TreeLinkNode node) {

if (node != null) {

inOrder(node.left);

list.add(node);

inOrder(node.right);

}

}

}

## JZ60把二叉树打印成多行

题目描述：

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。

思路：

1. 使用队列，队列属于collection集合，linkedlist实现了queue接口，因此我们使用linkedlist，stack、queue都是集合

2、利用队列模拟广度优先搜索，有设置一个size，这个很巧妙，也在一个while循环中，深度优先搜索类似于二叉树的前序遍历

3、以后命名可以用res、list、total等等，不要取得太长太复杂

4、创建queue使用linkedList

package jz\_offer.src.medium.JZ60;

import java.util.\*;

public class Solution {

public class TreeNode {

int val = 0;

TreeNode left = null;

TreeNode right = null;

public TreeNode(int val) {

this.val = val;

}

}

ArrayList<ArrayList<Integer>> Print(TreeNode pRoot) {

ArrayList<ArrayList<Integer>> totalList = new ArrayList<>();

if (pRoot == null) return totalList;

ArrayList<Integer> partList;

TreeNode node;

//局部变量需要自己初始化，成员变量jvm会帮我们初始化

int size = 0;

Queue<TreeNode> queue = new LinkedList<>();

queue.add(pRoot);

while (!queue.isEmpty()) {

size = queue.size();

partList = new ArrayList<>();

while (size > 0) {

node = queue.poll();

partList.add(node.val);

if (node.left != null) queue.add(node.left);

if (node.right != null) queue.add(node.right);

size--;

}

totalList.add(partList);

}

return totalList;

}

}

## JZ63数据流中的中位数

题目描述：

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。我们使用Insert()方法读取数据流，使用GetMedian()方法获取当前读取数据的中位数。

思路：

1、我们使用Arraylist存储数据，然后使用工具类对集合进行排序，就可以找出中位数了

2、set集合没有get()方法，因为它是无序的。它有contains方法，判断集合中是否有某个元素

3、Collections.sort(list)，可以对list按字典顺序排序

package jz\_offer.src.medium.JZ63;

import java.util.\*;

public class Solution {

private List<Double> list = new ArrayList<>();

public void Insert(Integer num) {

list.add(num.doubleValue());

Collections.sort(list);

}

public Double GetMedian() {

if (list.size() % 2 != 0) {

return list.get(list.size() / 2);

} else {

return (list.get(list.size() / 2) + list.get(list.size() / 2 - 1)) / 2.0;

}

}

}

## JZ65矩阵中的路径

题目描述：

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。

输入："ABCESFCSADEE",3,4,"ABCCED" 返回值：true

思路：

1、图的遍历算法，深度优先搜索，广度优先搜索

深度优先遍历的原理是递归，广度优先遍历的原理是层次遍历算法，用队列存储

queue.add poll poll和remove的区别 poll和remove都是弹出队列

remove在队列为空的时候会抛出异常

2、深搜加回溯的模板题

package jz\_offer.src.medium.JZ65;

public class Solution {

public boolean hasPath(char[] matrix, int rows, int cols, char[] str) {

char[][] board = new char[rows][cols];

int row = 0;

int col = 0;

for (char value : matrix) {

board[row][col] = value;

col++;

if (col == cols) {

row++;

col = 0;

}

}

StringBuilder sb = new StringBuilder();

for (char c : str) {

sb.append(c);

}

String word = sb.toString();

boolean[][] vis = new boolean[board.length][board[0].length];

for (int i = 0; i < board.length; i++) {

for (int j = 0; j < board[i].length; j++) {

if (solve(board, word, i, j, vis, 0)) {

// 找到一种情况即可

return true;

}

}

}

return false;

}

private boolean solve(char[][] board, String word, int x, int y, boolean[][] vis, int index) {

// 越界处理以及每个方格只能访问一次

if (x < 0 || x >= board.length || y < 0 || y >= board[0].length || vis[x][y]) return false;

// 匹配到某一位置不满足条件

if (word.charAt(index) != board[x][y]) return false;

// 匹配成功

if (index == word.length() - 1) return true;

vis[x][y] = true; // x，y位置的标记

boolean flag = solve(board, word, x + 1, y, vis, index + 1) ||

solve(board, word, x - 1, y, vis, index + 1) ||

solve(board, word, x, y + 1, vis, index + 1) ||

solve(board, word, x, y - 1, vis, index + 1);

vis[x][y] = false; // x，y位置的标记状态回溯

return flag;

}

}

## JZ67剪绳子

题目描述：

给你一根长度为n的绳子，请把绳子剪成整数长的m段（m、n都是整数，n>1并且m>1，m<=n），每段绳子的长度记为k[1],...,k[m]。请问k[1]x...xk[m]可能的最大乘积是多少？

例如，当绳子的长度是8时，我们把它剪成长度分别为2、3、3的三段，此时得到的最大乘积是18。

输入描述:输入一个数n，意义见题面。（2 <= n <= 60）

输入：8 返回值：18

思路：

1、通过找规律，可以看出，题目就是想要绳子的2和3的数目越多越好，这样得到的值是最大的。

2、我是通过对前面几个数字，进行一个枚举。发现算法计算的一个规律，写了一个算法来算，并没有用到动态规划来做，因为当时理解不了动态规划的思路。

3、我这个算法相当于是找规律，总结出来的

package jz\_offer.src.medium.JZ67;

public class Solution {

public int cutRope(int target) {

if (target < 4)

return target - 1;

int m = target / 3;

int n = target % 3;

if (n == 1) {

m = m - 1;

n = 4;

} else if (n == 0) {

n = 1;

}

return (int) (n \* Math.pow(3, m));

}

}

# 较难

## JZ2替换空格

题目描述：

请实现一个函数，将一个字符串中的每个空格替换成“%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。

思路：

1、再用一个stringbuffer存储新字符串即可

2、stringbuffer转成string，用toString方法

3、遍历一棵树比较常用的方法就是递归

package jz\_offer.src.hard.JZ2;

public class Solution {

public String replaceSpace(StringBuffer str) {

StringBuilder sb = new StringBuilder();

for (int i = 0; i < str.length(); i++) {

if (str.charAt(i) == ' ') sb.append("%20");

else sb.append(str.charAt(i));

}

return sb.toString();

}

}

## JZ3从尾到头打印链表

题目描述：

输入一个链表，按链表从尾到头的顺序返回一个ArrayList。

输入:{67,0,24,58} 返回值：[58,24,0,67]

思路：

1、使用栈，利用后进先出，从尾到头返回链表

2、stack.pop 弹出 stack.peek 返回栈顶元素但不弹出

package jz\_offer.src.hard.JZ3;

import java.util.\*;

public class Solution {

public class ListNode {

int val;

ListNode next = null;

ListNode(int val) {

this.val = val;

}

}

public ArrayList<Integer> printListFromTailToHead(ListNode listNode) {

ArrayList<Integer> list = new ArrayList<>();

Stack<ListNode> stack = new Stack<>();

if (listNode == null) return list;

while (listNode != null) {

stack.push(listNode);

listNode = listNode.next;

}

while (!stack.isEmpty()) {

list.add(stack.pop().val);

}

return list;

}

}

## JZ13调整数组顺序使奇数位于偶数前面

题目描述：

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。

思路：

1、使用暴力方法，时间复杂度其实也就O(n)，采用一个临时数组记录顺序，然后再赋值给原数组，完成此题

package jz\_offer.src.hard.JZ13;

public class Solution {

public void reOrderArray(int[] array) {

int[] temp = new int[array.length];

int index = 0;

for (int j : array) {

if (j % 2 != 0) {

temp[index] = j;

index++;

}

}

for (int j : array) {

if (j % 2 == 0) {

temp[index] = j;

index++;

}

}

//等价于System.arraycopy(temp, 0, array, 0, temp.length);

for (int i = 0; i < temp.length; i++) {

array[i] = temp[i];

}

}

}

## JZ17树的子结构

题目描述：

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）

输入：{8,8,#,9,#,2,#,5},{8,9,#,2} 返回值：true

思路：

1、代码分成两部分，先找到入门，这时第一步。然后再比较入口的左右子树，再写一个函数来判断。

需要用到flag1和flag2两个标志位，利于判断。虽然代码量大了，但思路还是清晰的

package jz\_offer.src.hard.JZ17;

public class Solution {

public class TreeNode {

int val = 0;

TreeNode left = null;

TreeNode right = null;

public TreeNode(int val) {

this.val = val;

}

}

public boolean HasSubtree(TreeNode root1, TreeNode root2) {

if (root1 == null || root2 == null) return false;

return dfs(root1, root2);

}

public boolean dfs(TreeNode root1, TreeNode root2) {

boolean flag1 = false;

boolean flag2 = false;

boolean flag = false;

if (root1.val == root2.val) flag = judge(root1, root2);

if (flag) return true;

if (root1.left != null) flag1 = dfs(root1.left, root2);

if (root1.right != null) flag2 = dfs(root1.right, root2);

return flag1 || flag2;

}

public boolean judge(TreeNode root1, TreeNode root2) {

if (root1 == null) return false;

if (root2 == null) return true;

if (root1.val == root2.val) {

boolean flag1 = true;

boolean flag2 = true;

if (root1.left != null || root2.left != null) flag1 = judge(root1.left, root2.left);

if (root1.right != null || root2.right != null) flag2 = judge(root1.right, root2.right);

return flag1 && flag2;

} else {

return false;

}

}

}

## JZ19顺时针打印矩阵

题目描述：

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4 X 4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.

输入：[[1,2],[3,4]] 返回值：[1,2,4,3]

思路：

1、定义4个变量，up,down,left,right。通过判断上下、左右边界是否交错，跳出break循环即返回正确的值

2、按上右下左的顺序编写代码

3、整体是一个while(true)循环，然后使用4个for，有条件判定跳出

package jz\_offer.src.hard.JZ19;

import java.util.\*;

public class Solution {

public ArrayList<Integer> printMatrix(int[][] matrix) {

ArrayList<Integer> list = new ArrayList<>();

if (matrix == null || matrix.length == 0 || matrix[0].length == 0) return list;

int up = 0;

int down = matrix.length - 1;

int left = 0;

int right = matrix[0].length - 1;

while (true) {

for (int col = left; col <= right; col++) {

list.add(matrix[up][col]);

}

up++;

if (up > down) break;

for (int row = up; row <= down; row++) {

list.add(matrix[row][right]);

}

right--;

if (right < left) break;

for (int col = right; col >= left; col--) {

list.add(matrix[down][col]);

}

down--;

if (down < up) break;

for (int row = down; row >= up; row--) {

list.add(matrix[row][left]);

}

left++;

if (left > right) break;

}

return list;

}

}

## JZ20包含min函数的栈

题目描述：

定义栈的数据结构，请在该类型中实现一个能够得到栈中所含最小元素的min函数（时间复杂度应为O（1））。

思路：

1、通过使用java提供的栈集合，使用两个栈，来找到到栈中的最小元素

2、stack.peek() 返回栈顶元素但不弹出

3、这道题要求两个栈中的元素个数一样，否则无法通过用例

package jz\_offer.src.hard.JZ20;

import java.util.\*;

public class Solution {

Stack<Integer> total = new Stack<>();

Stack<Integer> little = new Stack<>();

public void push(int node) {

total.push(node);

if (little.isEmpty() || node < little.peek()) {

little.push(node);

} else {

little.push(little.peek());

}

}

public void pop() {

total.pop();

little.pop();

}

public int top() {

return total.peek();

}

public int min() {

return little.peek();

}

}

## JZ23二叉搜索树的后序遍历序列

题目描述：

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。

如果是则返回true,否则返回false。假设输入的数组的任意两个数字都互不相同。

输入：[4,8,6,12,16,14,10] 返回值：true

思路：

1、二叉搜索树的定义：它的左子树小于根节点的值，它的右子树大于根节点的值。它的左右子树也为二叉搜索树

2、后序遍历，最后一个数字一定是根结点

3、这道题的思路还是根据多情况分析来判断，通过各种情况的分析，最后得出结论

4、采用递归的思路，用两个if条件判断是否满足后序遍历和二叉搜索树的性质。

5、使用递归的话，记得要设置一个出口

package jz\_offer.src.hard.JZ23;

import java.util.\*;

public class Solution {

public boolean verify(ArrayList<Integer> list) {

if (list.size() == 0 || list.size() == 1) return true;

ArrayList<Integer> minList = new ArrayList<>();

ArrayList<Integer> maxList = new ArrayList<>();

int endNumber = list.get(list.size() - 1);

int minIndex = -1;

int maxIndex = -1;

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

if (list.get(i) < endNumber) {

if (minIndex == -1) minIndex = i;

minList.add(list.get(i));

}

if (list.get(i) > endNumber) {

if (maxIndex == -1) maxIndex = i;

maxList.add(list.get(i));

}

}

if (minIndex != -1 && maxIndex != -1) {

if (minIndex > maxIndex) return false;

for (int i = maxIndex; i < list.size(); i++) {

if (list.get(i) < endNumber) return false;

}

}

return verify(minList) && verify(maxList);

}

public boolean VerifySquenceOfBST(int[] sequence) {

ArrayList<Integer> list = new ArrayList<>();

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

for (int j : sequence) {

list.add(j);

}

return verify(list);

}

}

## JZ24二叉树中和为某一值的路径

题目描述：

输入一颗二叉树的根节点和一个整数，按字典序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。

思路:

1、使用递归遍历左右子树，如果target==0，就把结果加入到result中最后返回。并且此时的左右子树还必须为null

2、终止条件为左右子树都为空

package jz\_offer.src.hard.JZ24;

import java.util.\*;

public class Solution {

public class TreeNode {

int val = 0;

TreeNode left = null;

TreeNode right = null;

public TreeNode(int val) {

this.val = val;

}

}

public void find(TreeNode root, int target, ArrayList<ArrayList<Integer>> result, ArrayList<Integer> path) {

//递归终止条件

if (root == null) return;

path.add(root.val);

target -= root.val;

if (target == 0 && root.left == null && root.right == null) {

result.add(path);

return;

}

//一个path往左子树，一个path往右子树，所以要new一个ArrayList

find(root.left, target, result, new ArrayList<>(path));

find(root.right, target, result, new ArrayList<>(path));

}

public ArrayList<ArrayList<Integer>> FindPath(TreeNode root, int target) {

ArrayList<ArrayList<Integer>> result = new ArrayList<>();

if (root == null) return result;

ArrayList<Integer> path = new ArrayList<>();

find(root, target, result, path);

return result;

}

}

## JZ25复杂链表的复制

题目描述：

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random指向一个随机节点），请对此链表进行深拷贝，并返回拷贝后的头结点。

（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）

思路：

1、输出结果不要返回节点引用，就是需要开辟新的内存空间返回，即深拷贝

2、深拷贝和浅拷贝的区别：

浅拷贝（shallowCopy）只是增加了一个指针指向已存在的内存地址。

深拷贝（deepCopy）是增加了一个指针并且申请了一个新的内存，使这个增加的指针指向这个新的内存。

B复制A，A改变，B没有改变，是深拷贝。

3、通过一个map结构，去维护原链表中结构与新链表中结构的关系，注意的是需要开辟一个新的内存空间

4、这道题的指针指向问题比较难理解，需要开辟一个新的内存空间返回，否则结果会直接返回空

5、map.getOrDefault(pHead,null) 有就返回pHead，没有就返回空

package jz\_offer.src.hard.JZ25;

import java.util.\*;

public class Solution {

public class RandomListNode {

int label;

RandomListNode next = null;

RandomListNode random = null;

RandomListNode(int label) {

this.label = label;

}

}

public RandomListNode Clone(RandomListNode pHead) {

Map<RandomListNode, RandomListNode> map = new HashMap<>();

RandomListNode node = pHead;

while (node != null) {

RandomListNode Node = new RandomListNode(node.label);

map.put(node, Node);

node = node.next;

}

node = pHead;

while (node != null) {

RandomListNode Node = map.get(node);

Node.next = map.get(node.next);

Node.random = map.get(node.random);

node = node.next;

}

return map.getOrDefault(pHead, null);

}

}

## JZ27字符串的排列

题目描述：

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,则按字典序打印出由字符a,b,c所能排列出来的所有字符串abc,acb,bac,bca,cab和cba。

输入："ab" 返回值：["ab","ba"]

思路：

1、二分查找一般是用while语句，时间复杂度是logn，一般写时间复杂度低的算法。

2、这道题的思路是递归，得到所有的可能结果放到list中，再去重，用collections工具类排序后得到正确结果。

package jz\_offer.src.hard.JZ27;

import java.util.\*;

public class Solution {

private String change(char[] a) {

StringBuilder res = new StringBuilder();

for (char value : a) {

res.append(value);

}

return res.toString();

}

private void solve(ArrayList<String> ans, char[] a, int index, int length) {

if (index == length - 1) {

String res = change(a);

ans.add(res);

} else {

// 就说明现在要去确定index位置的字符

for (int i = index; i < length; i++) {

char temp = a[i];

a[i] = a[index];

a[index] = temp;

// 当前index位置的字符已经通过交换找到了，那么就递归去找下一个位置的字符

solve(ans, a, index + 1, length);

// 其实就是去为了消除当前层去递归的时候的进行交换字符的影响，如果不消除的话，那么就会造成原index位置的字符发生变化

temp = a[i];

a[i] = a[index];

a[index] = temp;

}

}

}

public ArrayList<String> Permutation(String str) {

char[] a = str.toCharArray();

ArrayList<String> ans = new ArrayList<>();

solve(ans, a, 0, str.length());

ans = new ArrayList<String>(new HashSet<String>(ans)); // 去重操作

Collections.sort(ans); // 字典排序 -> ans.sort(null);

return ans;

}

}

## JZ32把数组排成最小的数

题目描述：

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。

输入：[3,32,321] 返回值："321323"

思路：

1、通过两个for，遍历所有情况，交换i和j的位置，从而最小值

2、Integer.valueOf 将string转换为数字

3、这道题相当于用选择排序的方式，找到了最小数字

package jz\_offer.src.hard.JZ32;

public class Solution {

public String PrintMinNumber(int[] numbers) {

StringBuilder str = new StringBuilder();

for (int i = 0; i < numbers.length; i++) {

for (int j = i + 1; j < numbers.length; j++) {

int a = Integer.parseInt(numbers[i] + "" + numbers[j]);

int b = Integer.parseInt(numbers[j] + "" + numbers[i]);

if (a > b) {

int temp = numbers[i];

numbers[i] = numbers[j];

numbers[j] = temp;

}

}

}

for (int number : numbers) {

str.append(number);

}

return str.toString();

}

}

## JZ33丑数

题目描述：

把只包含质因子2、3和5的数称作丑数（Ugly Number）。

例如6、8都是丑数，但14不是，因为它包含质因子7。 习惯上我们把1当做是第一个丑数。

求按从小到大的顺序的第N个丑数。

输入：7 返回值：8

思路：

1、通过遍历丑数\*2，丑数\*3，丑数\*5的队列，用a[0]当成第一个丑数，依次幅值，得到第N个丑数

2、数组中存放的都是丑数，丑数\*丑数=丑数

3、Math.min只能求两个数的最小值，因此要用两次Math.min

package jz\_offer.src.hard.JZ33;

public class Solution {

public int GetUglyNumber\_Solution(int index) {

if (index == 0)

return 0;

int[] a = new int[index];

//第一个丑数是1

a[0] = 1;

int index1 = 0;

int index2 = 0;

int index3 = 0;

for (int i = 1; i < index; i++) {

a[i] = Math.min(Math.min(a[index1] \* 2, a[index2] \* 3), a[index3] \* 5);

if (a[index1] \* 2 == a[i]) index1++;

if (a[index2] \* 3 == a[i]) index2++;

if (a[index3] \* 5 == a[i]) index3++;

}

return a[index - 1];

}

}

## JZ44翻转单词顺序列

题目描述：

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？

输入："nowcoder. a am I" 返回值："I am a nowcoder."

思路：

1、采用java提供的库函数，如split，分割数组。还有stringbuffer.append，注意看题目输入和返回值之间的关系，从而找到快速的解法

package jz\_offer.src.hard.JZ44;

public class Solution {

public String ReverseSentence(String str) {

StringBuilder sb = new StringBuilder();

if (str.trim().equals("")) return str;

String[] strs = str.split(" ");

for (int i = strs.length - 1; i > 0; i--) {

sb.append(strs[i]).append(" ");

}

sb.append(strs[0]);

return sb.toString();

}

}

## JZ49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："+2147483647" 返回值：2147483647

思路：

1、按过程分析，考虑到所有的情况，最后输出该数值

package jz\_offer.src.hard.JZ49;

public class Solution {

public int StrToInt(String str) {

int len = str.length();

int index = 0;

// 第一步，删除前面的空格

while (index < len) {

if (str.charAt(index) == ' ') {

index++;

} else {

break;

}

}

int flag = 0;

long ans = 0; // 最终返回的结果

while (index < len) {

// "3-2"

if (str.charAt(index) == '-' || str.charAt(index) == '+') {

if (flag != 0) return 0; // "-123-3", 第二个-号是非法字符， 返回0

flag = str.charAt(index) == '-' ? -1 : 1;

} else if (str.charAt(index) >= '0' && str.charAt(index) <= '9') {

ans = ans \* 10 + str.charAt(index) - '0'; // "-123"

if (judge(ans, flag)) { // 对ans是否溢出int类型做下判断

return 0;

}

} else {

return 0; // 既不是数字，也不是正负号，那就是其他字符了，返回0

}

index++;

}

return flag == -1 ? (int) ans \* (-1) : (int) ans;

}

private boolean judge(long ans, int flag) {

if (flag == -1) {

return ans \* (-1) < Integer.MIN\_VALUE;

} else {

return ans > Integer.MAX\_VALUE;

}

}

}

## JZ52正则表达式匹配

题目描述：

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

输入："aaa","a\*a" 返回：true

思路：

1、直接使用java提供的库函数，String.matches。算是投机取巧的方法，可以判断是否符合正则表达式

package jz\_offer.src.hard.JZ52;

public class Solution {

public boolean match(char[] str, char[] pattern) {

return new String(str).matches(new String(pattern));

}

}

## JZ56删除链表中重复的结点

题目描述：

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

输入：{1,2,3,3,4,4,5} 返回值：{1,2,5}

思路：

1、创建临时结点变量，使用尾接法

package jz\_offer.src.hard.JZ56;

public class Solution {

public class ListNode {

int val;

ListNode next = null;

ListNode(int val) {

this.val = val;

}

}

private ListNode change(ListNode x) {

int temp = x.val;

while (x != null && x.val == temp) {

x = x.next;

}

return x;

}

public ListNode deleteDuplication(ListNode pHead) {

ListNode ans = pHead; // 最终链表的头节点

// 确定最终链表的头节点

while (ans != null) {

if (ans.next != null && ans.val == ans.next.val) {

// 当前ans所指的节点是重复节点

ans = change(ans);

} else {

// 当前ans所指的节点就是我们最终链表的头节点

break;

}

}

if (ans == null) return null;

// 判断从ans到链表的尾部，判断每一个节点是否为重复节点。

ListNode lastNode = ans; // 最终链表的尾部节点

ListNode removeNode = lastNode.next; // 遍历剩余的节点的变量

while (removeNode != null) {

if (removeNode.next != null && removeNode.val == removeNode.next.val) {

// 当前removeNode所指的节点是重复节点

removeNode = change(removeNode);

} else {

lastNode.next = removeNode;

lastNode = removeNode;

removeNode = removeNode.next;

}

}

lastNode.next = null; // 1 -> 2 -> 3 -> 4 -> 4

return ans;

}

}

## JZ59按之字形顺序打印二叉树

题目描述：

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。

输入：{8,6,10,5,7,9,11} 返回值：[[8],[10,6],[5,7,9,11]]

思路：

1、可以使用两个栈，存入每层结点的左右节点。按照题目要求的之字形设计入栈出栈顺序，即可完成。

2、主要就是利用两个栈存储，并且利用后进先出的性质

package jz\_offer.src.hard.JZ59;

import java.util.\*;

public class Solution {

public class TreeNode {

int val = 0;

TreeNode left = null;

TreeNode right = null;

public TreeNode(int val) {

this.val = val;

}

}

public ArrayList<ArrayList<Integer>> Print(TreeNode pRoot) {

ArrayList<ArrayList<Integer>> result = new ArrayList<>();

if (pRoot == null) return result;

Stack<TreeNode> tmp = new Stack<>();

Stack<TreeNode> tmp1 = new Stack<>();

tmp.push(pRoot);

while (tmp.size() > 0 || tmp1.size() > 0) {

ArrayList<Integer> list = new ArrayList<>();

if (tmp.size() > 0) {

int size = tmp.size();

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

TreeNode node = tmp.pop();

if (node.left != null) tmp1.push(node.left);

if (node.right != null) tmp1.push(node.right);

list.add(node.val);

}

result.add(list);

continue;

}

int size = tmp1.size();

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

TreeNode node = tmp1.pop();

if (node.right != null) tmp.push(node.right);

if (node.left != null) tmp.push(node.left);

list.add(node.val);

}

result.add(list);

}

return result;

}

}

## JZ61序列化二叉树

题目描述：

请实现两个函数，分别用来序列化和反序列化二叉树

二叉树的序列化是指：把一棵二叉树按照某种遍历方式的结果以某种格式保存为字符串，从而使得内存中建立起来的二叉树可以持久保存。

序列化可以基于先序、中序、后序、层序的二叉树遍历方式来进行修改，序列化的结果是一个字符串，序列化时通过 某种符号表示空节点（#），以 ！ 表示一个结点值的结束（value!）。

二叉树的反序列化是指：根据某种遍历顺序得到的序列化字符串结果str，重构二叉树。

例如，我们可以把一个只有根节点为1的二叉树序列化为"1,"，然后通过自己的函数来解析回这个二叉树。

输入：{8,6,10,5,7,9,11} 返回值：{8,6,10,5,7,9,11}

思路：

1、二叉树的宽搜，以层为单位，先搜第一层，再搜第二层，再搜第三层

Queue queue=new LinkedList 使用队列是用Linkedlist

2、序列化操作是通过对二叉树宽搜，如果当前节点是null的话，然后判断当前节点的后面的节点当中是否有非空节点，如果有，就将null写入序列，反之不写入。反序列化操作也是采用宽搜的方式，对于最终的二叉树的节点逐个的去创建。

package jz\_offer.src.hard.JZ61;

import java.util.\*;

public class Solution {

public class TreeNode {

int val = 0;

TreeNode left = null;

TreeNode right = null;

public TreeNode(int val) {

this.val = val;

}

}

String Serialize(TreeNode root) {

StringBuilder ans = new StringBuilder("[");

Queue<TreeNode> queue = new LinkedList<>();

queue.add(root);

int sum = 1; // 用来记录当前节点及其后面非空节点的个数

while (!queue.isEmpty() && root != null) {

TreeNode node = queue.poll();

if (node == null) {

ans.append("null");

} else {

ans.append(node.val);

sum--;

if (node.left != null) sum++;

if (node.right != null) sum++;

queue.add(node.left);

queue.add(node.right);

}

if (sum != 0) {

ans.append(",");

} else {

break;

}

}

ans.append("]");

return ans.toString();

}

TreeNode Deserialize(String data) {

String s = data.substring(1, data.length() - 1);

if ("".equals(s)) return null; // data = "[]"

String[] a = s.split(",");

int index = 0;

Queue<TreeNode> queue = new LinkedList<>();

TreeNode root = new TreeNode(change(a[index++]));

queue.add(root);

while (!queue.isEmpty() && index < a.length) {

TreeNode node = queue.poll();

if (!"null".equals(a[index])) {

node.left = new TreeNode(change(a[index++]));

queue.add(node.left);

} else {

index++;

}

if (index < a.length && !"null".equals(a[index])) {

node.right = new TreeNode(change(a[index++]));

queue.add(node.right);

} else {

index++;

}

}

return root;

}

public int change(String s) {

int res = 0;

int i = 0;

int flag = 1;

if (s.charAt(0) == '-') {

i++;

flag = -1;

}

for (; i < s.length(); i++) {

res = res \* 10 + s.charAt(i) - '0';

}

return res \* flag;

}

}

## JZ64滑动窗口的最大值

题目描述：

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]}。

窗口大于数组长度的时候，返回空。

输入：[2,3,4,2,6,2,5,1],3 返回值：[4,4,6,6,6,5]

思路：

1、使用通俗易懂的方法，记录滑动窗口内的最大值下标，与最右边的元素比较。如果最大值不在滑动窗口中了，则重新找最大值，再记录滑动窗口下标。

package jz\_offer.src.hard.JZ64;

import java.util.\*;

public class Solution {

public ArrayList<Integer> maxInWindows(int[] num, int size) {

ArrayList<Integer> list = new ArrayList<>();

if (num.length == 0 || size == 0 || size > num.length) return list;

int low = 0;

int high = size - 1;

int len = num.length;

int index = -1;

int max = num[0];

while (high < len) {

if (index >= low && index <= high) {

if (max < num[high]) {

max = num[high];

index = high;

}

} else {

max = num[low];

index = low;

for (int i = low + 1; i <= high; i++) {

if (max < num[i]) {

max = num[i];

index = i;

}

}

}

low++;

high++;

list.add(max);

}

return list;

}

}

## JZ66机器人的运动范围

题目描述：

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？

输入：5,10,10 返回：21

思路：

1、这道题的思路和JZ65差不多，依旧是深搜。深搜的思想就是递归

package jz\_offer.src.hard.JZ66;

public class Solution {

private int sum = 0;

public int movingCount(int threshold, int rows, int cols) {

boolean[][] vis = new boolean[rows][cols];

solve(0, 0, rows, cols, vis, threshold);

return sum;

}

private void solve(int x, int y, int rows, int cols, boolean[][] vis, int threshold) {

if (x < 0 || x >= rows || y < 0 || y >= cols || vis[x][y] || cul(x, y) > threshold) return;

vis[x][y] = true;

sum++;

solve(x + 1, y, rows, cols, vis, threshold);

solve(x - 1, y, rows, cols, vis, threshold);

solve(x, y + 1, rows, cols, vis, threshold);

solve(x, y - 1, rows, cols, vis, threshold);

}

private int cul(int x, int y) {

int res = 0;

while (x != 0) {

res += x % 10;

x /= 10;

}

while (y != 0) {

res += y % 10;

y /= 10;

}

return res;

}

}

# 困难

## JZ22从上往下打印二叉树

题目描述：

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。

输入：{5,4,#,3,#,2,#,1} 返回值：[5,4,3,2,1]

思路：

1、广度优先搜索，采用队列的数据结构，和二叉树的层次遍历类似。

2、一个java文件，只能定义一个public的类。但public的类，内部可以再定义多个public的类，叫做内部类。

package jz\_offer.src.deephard.JZ22;

import java.util.\*;

public class Solution {

public class TreeNode {

int val = 0;

TreeNode left = null;

TreeNode right = null;

public TreeNode(int val) {

this.val = val;

}

}

public ArrayList<Integer> PrintFromTopToBottom(TreeNode root) {

ArrayList<Integer> list = new ArrayList<>();

if (root == null) return list;

Queue<TreeNode> queue = new LinkedList<>();

TreeNode node = root;

while (node != null) {

list.add(node.val);

if (node.left != null) queue.add(node.left);

if (node.right != null) queue.add(node.right);

node = queue.poll();

}

return list;

}

}

## JZ35数组中的逆序对

题目描述：

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

对于50%的数据,ize≤10^4

对于75%的数据,size≤10^5

对于100%的数据,size≤2∗10^5

输入：[1,2,3,4,5,6,7,0] 输出：7

思路：

1、使用归并排序，这道题暂时还未掌握，归并排序不会

package jz\_offer.src.deephard.JZ35;

public class Solution {

private long sum; // 用来去统计逆序对的个数

public int InversePairs(int[] array) {

sum = 0;

int l = 0;

int r = array.length - 1;

divide(l, r, array);

return (int) (sum % 1000000007);

}

private void divide(int l, int r, int[] array) {

if (l != r) {

int mid = (l + r) >> 1;

divide(l, mid, array);

divide(mid + 1, r, array);

merge(l, r, mid, array);

}

}

private void merge(int l, int r, int mid, int[] array) {

int i = l; // 左区间的起点

int j = mid + 1; // 右区间的起点

int[] temp = new int[r - l + 1];

int index = 0;

while (i <= mid && j <= r) {

if (array[i] > array[j]) {

temp[index++] = array[j++];

sum += mid - i + 1; // 这一行是核心，去统计逆序对个数，统计的基础是在归并排序的合并过程中，合并的两个子序列都是有序的

} else {

temp[index++] = array[i++];

}

}

while (i <= mid) {

temp[index++] = array[i++];

}

while (j <= r) {

temp[index++] = array[j++];

}

index = 0;

for (int k = l; k <= r; k++) {

array[k] = temp[index++];

}

}

}

## JZ58对称的二叉树

题目描述：

请实现一个函数，用来判断一棵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。

输入：{8,6,6,5,7,7,5} 返回值：true

思路：

1、采用递归的思想，判断节点的左右节点的值是否相同，如果不相同，就不是对称的

2、主要就是递归的思想，递归的终止条件要分析一下。递归的思想经常出现，必须掌握

3、这道题使用递归的思想，还是比较容易理解的

package jz\_offer.src.deephard.JZ58;

public class Solution {

public class TreeNode {

int val = 0;

TreeNode left = null;

TreeNode right = null;

public TreeNode(int val) {

this.val = val;

}

}

boolean isSymmetrical(TreeNode pRoot) {

if (pRoot == null) return true;

return solve(pRoot.left, pRoot.right);

}

private boolean solve(TreeNode node1, TreeNode node2) {

//这些都是递归终止条件，必须写

if (node1 == null && node2 == null) return true;

if (node1 == null || node2 == null) return false;

if (node1.val != node2.val) return false;

return solve(node1.left, node2.right) && solve(node1.right, node2.left);

}

}