题目描述 1 二维数组中是否存在该值

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。

从右上角开始遍历，若 target 大于该值，则向下一行，若小于该值，则向左一列。

```
public class Solution {
    public boolean Find(int target, int [][] array) {
        int row=0;
        int col=array[0].length-1;
        while(row<array.length && col>=0) {
            if(array[row][col]==target) {
                return true;
            }
            else if(array[row][col]<target) {
                row++;
            }
            else {
                col--;
            }
        }
        return false;
    }
}
```

题目描述 2 字符串空格替换成%20

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。

```
public class Solution {
    public String replaceSpace(StringBuffer str) {
        int spaceNum=0;
        for(int i=0;i<str.length();i++) {
            if(str.charAt(i)==' ') {
                spaceNum++;
            }
        }
        int lengthNew=str.length()+spaceNum*2;  // 新长度
        int indexOld=str.length()-1;  // 旧索引
        int indexNew=lengthNew-1;  // 新索引
        str.setLength(lengthNew);
        for(;indexOld>=0;--indexOld) {
            if(str.charAt(indexOld)==' ') {
                str.setCharAt(indexNew--,'0');
                str.setCharAt(indexNew--,'2');
                str.setCharAt(indexNew--,'%');
            }
            else {
                str.setCharAt(indexNew--,str.charAt(indexOld));
            }
        }
        return str.toString();
    }
}
```

题目描述 3 链表反转，输出 ArrayList

输入一个链表，按链表值从尾到头的顺序返回一个 ArrayList

```
/**
*    public class ListNode {
*       int val;
*       ListNode next = null;
*
*       ListNode(int val) {
*          this.val = val;
*       }
*    }
*
*/
import java.util.ArrayList;
public class Solution {
   ArrayList<Integer> arrayList=new ArrayList<Integer>();
   public ArrayList<Integer> printListFromTailToHead(ListNode
listNode) {
       if(listNode!=null) {
          this.printListFromTailToHead(listNode.next);
          arrayList.add(listNode.val);
       }
       return arrayList;
   }
}
```

题目描述 4 根据先中序遍历重建二叉树

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。

```
/**
 * Definition for binary tree
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
```

前序遍历的第一个数肯定为 root，然后分别构建它的左右子树
在中序遍历里找到它的前一个数，若无，则左子树为 NULL；
若有，则构建左子树；再找到它的后一个数，
若无，则右子树为 NULL；若有，则构建右子树；
然后以 root 所在中序遍历数组位置分割成两半，记录前一半，
即左子树的大小，然后在前序遍历中也把左子树和右子树的子串
提出来，分别作为两棵树，重建，直到提出来的子串长度为 0.

```
public class Solution {
    public TreeNode reConstructBinaryTree(int [] pre,int [] in) {
        TreeNode root=reConstructBinaryTree(pre,0,pre.length-
1,in,0,in.length-1);
        return root;
    }
    public TreeNode reConstructBinaryTree(int[] pre,int
startPre,int endPre,int[] in,int startIn,int endIn) {
        if(startPre>endPre ||startIn>endIn) {
            return null;
        }
        TreeNode root=new TreeNode(pre[startPre]);
        for(int i=startIn;i<=endIn;i++) {
            if(in[i]==pre[startPre]) {
                root.left=reConstructBinaryTree(pre,startPre+1,i-
startIn+startPre,in,startIn,i-1);
                root.right=reConstructBinaryTree(pre,i-
startIn+startPre+1,endPre,in,i+1,endIn);
            }
        }
        return root;
    }
}
```

题目描述 5 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。

```
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(stack1.empty() && stack2.empty()) {
            throw new RuntimeException("Queue is empty!");
        }
        if(stack2.empty()) {
            while(!stack1.empty()) {
                stack2.push(stack1.pop());
            }
        }
        return stack2.pop();
    }
}
```

题目描述 6 旋转数组的最小元素

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。

采用二分法解答这个问题，mid = low + (high - low)/2，需要考虑三种情况：

(1)array[mid] > array[high]:

出现这种情况的 array 类似[3,4,5,6,0,1,2]，此时最小数字一定在 mid 的右边。

low = mid + 1

(2)array[mid] == array[high]:

出现这种情况的 array 类似 [1,0,1,1,1] 或者[1,1,1,0,1]，此时最小数字不好判断在 mid 左边还是右边,这时只好一个一个试 ，

high = high - 1

(3)array[mid] < array[high]:

出现这种情况的 array 类似[2,2,3,4,5,6,6],此时最小数字一定就是 array[mid]或者在 mid 的左边。因为右边必然都是递增的。

high = mid

注意这里有个坑：如果待查询的范围最后只剩两个数，那么 mid 一定会指向下标靠前的数字

比如 array = [4,6]

array[low] = 4 ;array[mid] = 4 ; array[high] = 6 ;

如果 high = mid - 1，就会产生错误， 因此 high = mid

但情形(1)中 low = mid + 1 就不会错误

return array[low]和 array[high]都可以

```
import java.util.ArrayList;
public class Solution {
    public int minNumberInRotateArray(int [] array) {
        int low=0;
        int high=array.length-1;
        while(low<high){
            int mid=low+(high-low)/2;
            if(array[mid]>array[high]) {
                low=mid+1;
            }
            else if(array[mid]==array[high]) {
                high=high-1;
            }
            else if(array[mid]<array[high]){
                high=mid;
            }
        }
        return array[low];
    }
}
```

方法二：遍历整个数字，找到 array[i]>array[i+1]的数返回即可。

```
import java.util.ArrayList;
public class Solution {
    public int minNumberInRotateArray(int [] array) {
        if(array.length==0) {
            return 0;
        }
        for(int i=0;i<array.length-1;i++) {
            if(array[i]>array[i+1]) {
                return array[i+1];
            }
        }
        return array[0];
    }
}
```

题目描述 7 斐波那契数列第 n 项

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（从 0 开始，第 0 项为 0）。

n<=39

动态规划：

```
public class Solution {
    public int Fibonacci(int n) {
        int preNum=0;
        int result=1;
        if(n<2) {
            return n;
        }
        for(int i=2;i<=n;i++) {
            result=preNum+result;
            preNum=result-preNum;
        }
        return result;
    }
}
```

递归：

```
public class Solution {
    public int Fibonacci(int n) {
        if(n<2) {
            return n;
        }
        else {
            return Fibonacci(n-1)+Fibonacci(n-2);
        }
    }
}
```

循环：

```
public class Solution {
    public int Fibonacci(int n) {

        int[] s=new int[40];
        s[0]=0;
        s[1]=1;
        for(int i=2;i<40;i++) {
            s[i]=s[i-1]+s[i-2];
        }
        return s[n];
    }
}
```

题目描述 8 青蛙跳台阶 1-2 跳法数

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）

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

题目描述 9 青蛙跳台阶 1-跳法数

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。

```
public class Solution {
    public int JumpFloorII(int target) {
        if(target<=0) {
            return -1;
        }
        else if(target<2) {
            return target;
        }
        else {
            return JumpFloorII(target-1)*2;
        }
    }
}
```

f(n-1) = f(0) + f(1)+f(2)+f(3) + ... + f((n-1)-1) = f(0) + f(1) + f(2) + f(3) + ... + f(n-2)

f(n) = f(0) + f(1) + f(2) + f(3) + ... + f(n-2) + f(n-1) = f(n-1) + f(n-1)

可以得出： f(n) = 2*f(n-1)

```
public class Solution {
    public int JumpFloorII(int target) {
        if(target<0) {
            return -1;
        }
        else if(target<=2) {
            return target;
        }
        else {
            int result=0;
            for(int i=2;i<=target;i++) {
                result+=JumpFloorII(i-1);
            }
            return result+1;
        }
    }
}
```

题目描述 10 2*1 覆盖 2*n 矩形方法数

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？

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

题目描述 11 二进制表示中 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

把一个整数减去 1，再和原整数做与运算，会把该整数最右边一个 1 变成 0.那么一个整数的二进制有多少个 1，就可以进行多少次这样的操作。

```
public class Solution {
    public int NumberOf1(int n) {
        int result=0;
        while(n!=0) {
            result++;
            n=(n-1)&n;
        }
        return result;
    }
}
```

题目描述 12 幂运算

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。

```
public class Solution {
    public double Power(double base, int exponent) {
        double result=1.0;
        for(int i=0;i<Math.abs(exponent);i++) {
            result*=base;
        }
        if(exponent>0) {
            return result;
        }
        else if(exponent<0) {
            return 1/result;
        }
        return 1;
  }
}
```

快速幂算法：

```
public class Solution {
    public double Power(double base, int exponent) {
        double res=1;
        double curr=base;
        int e;
        if(exponent>0) {
            e=exponent;
        }
        else if(exponent<0) {
            if(base==0) {
                throw new RuntimeException("除 0 异常");
            }
            e=-exponent;
        }
        else {
            return 1;
        }
        while(e!=0) {
            if((e&1)==1) {
                res*=curr;
            }
            curr*=curr;
            e>>=1;
        }
```

```
        return exponent>0? res:(1/res);
    }
}
```

题目描述 13 数组奇数和偶数左右分离

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。

```
public class Solution {
    public void reOrderArray(int [] array) {
        int length=array.length;
        int[] res=new int[length];
        int p=0;
        for(int i=0;i<length;i++) {
            if((array[i]&1)==1) {
                res[p]=array[i];
                p++;
            }
        }
        for(int i=0;i<length;i++) {
            if((array[i]&1)==0) {
                res[p]=array[i];
                p++;
            }
        }
        for(int i=0;i<length;i++){
            array[i]=res[i];
        }
    }
}
```

冒泡算法：

```
public class Solution {
    public void reOrderArray(int [] array) {
        for(int i=0;i<array.length;i++) {
            for(int j=array.length-1;j>i;j--) {
                if(array[j]%2==1 && array[j-1]%2==0) {
                    swap(array,j,j-1);
                }
            }
        }
    }
    private void swap(int[] array,int i,int j) {
        int temp=array[i];
        array[i]=array[j];
        array[j]=temp;
    }
}
```

题目描述 14 链表倒数第 k 个结点

输入一个链表，输出该链表中倒数第 k 个结点。

```
/*
public class ListNode {
    int val;
    ListNode next = null;

    ListNode(int val) {
        this.val = val;
    }
}*/
public class Solution {
    public ListNode FindKthToTail(ListNode head,int k) {
        if(head==null || k<=0) {
            return null;
        }
        ListNode pre=head;
        ListNode last=head;
        for(int i=1;i<k;i++) {
            if(pre.next!=null) {
                pre=pre.next;
            }
            else {
                return null;
            }
        }
        while(pre.next!=null) {
            pre=pre.next;
            last=last.next;
        }
        return last;
    }
}
```

题目描述 15 反转链表，输出链表头

输入一个链表，反转链表后，输出新链表的表头。

```
/*
public class ListNode {
    int val;
    ListNode next = null;

    ListNode(int val) {
        this.val = val;
    }
}*/
public class Solution {
    public ListNode ReverseList(ListNode head) {
        if(head==null) {
            return null;
        }
        ListNode pre=null;
        ListNode next=null;
        while(head!=null) {
            next=head.next;
            head.next=pre;
            pre=head;
            head=next;
        }
        return pre;
    }
}
```

题目描述 16 两个单调递增链表合并

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。

```
/*
public class ListNode {
    int val;
    ListNode next = null;

    ListNode(int val) {
        this.val = val;
    }
}*/
public class Solution {
    public ListNode Merge(ListNode list1,ListNode list2) {
        if(list1==null) {
            return list2;
        }
        if(list2==null) {
            return list1;
        }
        ListNode head=null;
        ListNode current=null;
        while(list1!=null && list2!=null) {
            if(list1.val<=list2.val) {
                if(head==null) {
                    head=current=list1;
                }
                else {
                    current.next=list1;
                    current=current.next;
                }
                list1=list1.next;
            }
           else if(list1.val>list2.val) {
                if(head==null) {
                    head=current=list2;
                }
                else {
                    current.next=list2;
                    current=current.next;
                }
                list2=list2.next;
            }
```

```
        }
        if(list1==null) {
            current.next=list2;
        }
        if(list2==null) {
            current.next=list1;
        }
        return head;
    }
}
```

递归：

```
public class Solution {
    public ListNode Merge(ListNode list1,ListNode list2) {
        if(list1==null) {
            return list2;
        }
        if(list2==null) {
            return list1;
        }
        if(list1.val<=list2.val) {
            list1.next=Merge(list1.next,list2);
            return list1;
        }
        else {
            list2.next=Merge(list1,list2.next);
            return list2;
        }
    }
}
```

题目描述 17 二叉树 B 是否为 Ade 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）

```
/**
public class TreeNode {
    int val = 0;
    TreeNode left = null;
    TreeNode right = null;

    public TreeNode(int val) {
        this.val = val;

    }

}
*/
public class Solution {
    public boolean HasSubtree(TreeNode root1,TreeNode root2) {
        if(root1==null || root2==null) {
            return false;
        }
        return isSubtree(root1,root2) ||
            HasSubtree(root1.left,root2) ||
            HasSubtree(root1.right,root2);
    }
    private boolean isSubtree(TreeNode root1,TreeNode root2) {
        if(root2==null) {
            return true;
        }
        if(root1==null) {
            return false;
        }
        if(root1.val==root2.val) {
            return isSubtree(root1.left,root2.left) &&
                isSubtree(root1.right,root2.right);
        }
        return false;
    }
}
```

题目描述 18 二叉树镜像

操作给定的二叉树，将其变换为源二叉树的镜像。

```
/**
public class TreeNode {
    int val = 0;
    TreeNode left = null;
    TreeNode right = null;
    public TreeNode(int val) {
        this.val = val;
    }
}
*/
public class Solution {
    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);
    }
}

public class Solution {
    public void Mirror(TreeNode root) {
        if(root==null) {
            return;
        }
        Stack<TreeNode> stack=new Stack<TreeNode>();
        stack.push(root);
        while(!stack.isEmpty()) {
            TreeNode node=stack.pop();
            if(node.left!=null || node.right!=null) {
                TreeNode temp=node.left;
                node.left=node.right;
                node.right=temp;
            }
            if(node.left!=null) {
                stack.push(node.left);
            }
            if(node.right!=null) {
```

```
                stack.push(node.right);
            }
        }
    }
}
```

题目描述 19 从外向里打印矩阵

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 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.

```
import java.util.ArrayList;
public class Solution {
    public ArrayList<Integer> printMatrix(int [][] matrix) {
        int row=matrix.length;
        int col=matrix[0].length;
        ArrayList<Integer> res = new ArrayList<Integer>();
        int left=0, top=0, right=col-1, bottom=row-1;
        while(left<=right && top<=bottom) {
            for(int i=left;i<=right;i++) {
                res.add(matrix[top][i]);
            }
            for(int i=top+1;i<=bottom;i++) {
                res.add(matrix[i][right]);
            }
            for(int i=right-1;i>=left && top<bottom;i--) {
                res.add(matrix[bottom][i]);
            }
            for(int i=bottom-1;i>top && left<right;i--) {
                res.add(matrix[i][left]);
            }
            left++;
            top++;
            right--;
            bottom--;
        }
        return res;
    }
}
```

题目描述 20 栈中最小元素，时间复杂度 O（1）

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））。

```
import java.util.Stack;
public class Solution {
    Stack<Integer> data = new Stack<Integer>();
    Stack<Integer> min = new Stack<Integer>();
    Integer temp=null;
    public void push(int node) {
        if(temp!=null) {
            if(node<=temp) {
                temp=node;
                min.push(node);
            }
        }
        else {
            temp=node;
            min.push(node);
        }
        data.push(node);
    }

    public void pop() {
        int num1=data.pop();
        int num2=min.pop();
        if(num1!=num2) {
            min.push(num2);
        }
    }

    public int top() {
        int num1=data.pop();
        data.push(num1);
        return num1;
    }

    public int min() {
        int num2=min.pop();
        min.push(num2);
        return num2;
    }
}
```

题目描述 21 是否为栈的弹出顺序

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）

```
import java.util.ArrayList;
import java.util.Stack;

public class Solution {
    public boolean IsPopOrder(int [] pushA,int [] popA) {
        if(pushA.length==0 || popA.length==0) {
            return false;
        }
        Stack<Integer> s=new Stack<Integer>();
        int index=0;
        for(int i=0;i<pushA.length;i++) {
            s.push(pushA[i]);
            while(!s.empty() && s.peek()==popA[index]) {
                s.pop();
                index++;
            }
        }
        return s.empty();
    }
}
```

题目描述 22 层序打印二叉树

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。

```
import java.util.ArrayList;
/**
public class TreeNode {
    int val = 0;
    TreeNode left = null;
    TreeNode right = null;

    public TreeNode(int val) {
        this.val = val;

    }

}
*/
public class Solution {
    ArrayList<Integer> res=new ArrayList<Integer>();
    public ArrayList<Integer> PrintFromTopToBottom(TreeNode root)
{
        ArrayList<Integer> res=new ArrayList<Integer>();
        ArrayList<TreeNode> queue=new ArrayList<TreeNode>();
        if(root==null) {
            return res;
        }
        queue.add(root);
        while(queue.size()!=0) {
            TreeNode temp=queue.remove(0);
            if(temp.left!=null) {
                queue.add(temp.left);
            }
            if(temp.right!=null) {
                queue.add(temp.right);
            }
            res.add(temp.val);
        }
        return res;
    }
}
```

题目描述 23 判断是否二叉搜索树后序遍历结果

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。

BST 的后序序列的合法序列是，对于一个序列 S，最后一个元素是 x （也就是根），
如果去掉最后一个元素的序列为 T，那么 T 满足：T 可以分成两段，
前一段（左子树）小于 x，后一段（右子树）大于 x，且这两段（子树）都是合法的后序序列。
第一种 全是右子树 i=start 调用最后的 isBST 左子树直接返回 true 右子树递归递归最后 root-1=i=start return true
第二种全是左子树 i=root 最后的 isBST 左子树递归递归 最后 root-1=start 返回 true 右子树直接返回 true
第三种 左子树右子树都有正常递归 继续直到进入前面两种情况之一
相等时表示子树个数为 1

```
public class Solution {
    public boolean VerifySquenceOfBST(int [] sequence) {
        if(sequence.length==0) {
            return false;
        }
        return isBST(sequence,0,sequence.length-1);
    }
    private boolean isBST(int[]a,int start,int root) {
        if(start>=root) {
            return true;
        }
        int i=root;
        while(i>start && a[i-1]>a[root]) {
            i--;
        }
        for(int j=start;j<i;j++) {
            if(a[j]>a[root]) {
                return false;
            }
        }
        return isBST(a,start,i-1)&&isBST(a,i,root-1);
    }
}
```

题目描述 24？？ 节点值和为目标值的路径

输入一颗二叉树的跟节点和一个整数，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。(注意:在返回值的 list 中，数组长度大的数组靠前)

```
import java.util.ArrayList;
/**
public class TreeNode {
    int val = 0;
    TreeNode left = null;
    TreeNode right = null;

    public TreeNode(int val) {
        this.val = val;

    }

}
*/
public class Solution {
    private ArrayList<ArrayList<Integer>> res=new
ArrayList<ArrayList<Integer>>();
    private ArrayList<Integer> list=new ArrayList<Integer>();
    public ArrayList<ArrayList<Integer>> FindPath(TreeNode
root,int target) {
        if(root==null) {
            return res;
        }
        list.add(root.val);
        target-=root.val;
        if(target==0 && root.left==null && root.right==null) {
            res.add(new ArrayList<Integer>(list));
        }
        FindPath(root.left,target);
        FindPath(root.right,target);
        list.remove(list.size()-1);
        return res;
    }
}
```

题目描述 25 复制随机指针链表

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）

具体分为三步：

（1）在旧链表中创建新链表，此时不处理新链表的兄弟结点

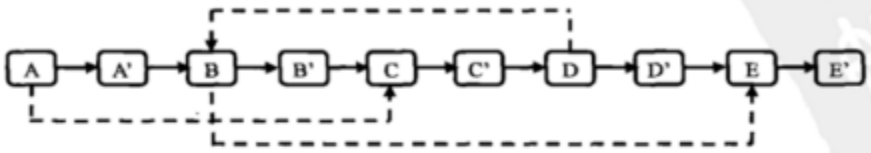


图 4.9 复制复杂链表的第一步

（2）根据旧链表的兄弟结点，初始化新链表的兄弟结点

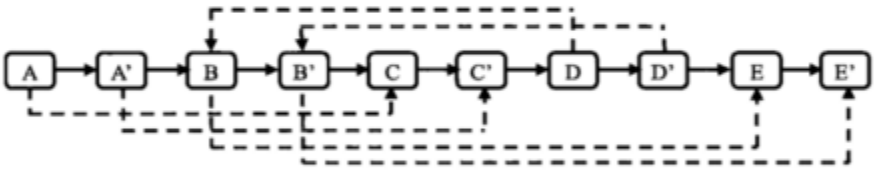


图 4.10 复制复杂链表的第二步

（3）从旧链表中拆分得到新链表

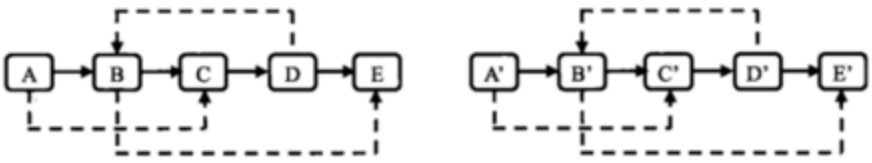


图 4.11 复制复杂链表的第三步


```
/*
public class RandomListNode {
    int label;
    RandomListNode next = null;
    RandomListNode random = null;
    RandomListNode(int label) {
        this.label = label;
    }
}
*/
public class Solution {
    public RandomListNode Clone(RandomListNode pHead) {
        if(pHead==null) {
            return null;
```

```
        }
        RandomListNode pCur=pHead;
        while(pCur!=null) {
            RandomListNode node=new RandomListNode(pCur.label);
            node.next=pCur.next;
            pCur.next=node;
            pCur=pCur.next.next;
        }
        pCur=pHead;
        while(pCur!=null) {
            if(pCur.random!=null) {
                pCur.next.random=pCur.random.next;
            }
            else {
                pCur.next.random=null;
            }
            pCur=pCur.next.next;
        }
        RandomListNode head=pHead.next;
        pCur=pHead;
        while(pCur!=null) {
            RandomListNode cloneNode=pCur.next;
            pCur.next=cloneNode.next;
            if(cloneNode.next!=null) {
                cloneNode.next=cloneNode.next.next;
            }
            pCur=pCur.next;
        }
        return head;
    }
}
```

题目描述 26 二叉搜索树转换成双向链表

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。

```
/**
public class TreeNode {
    int val = 0;
    TreeNode left = null;
    TreeNode right = null;
    public TreeNode(int val) {
        this.val = val;
    }
}
*/
```

二叉查找树（二叉搜索树，二叉排序树）它或者是一棵空树，或者是具有下列性质的二叉树：若它的左子树不空，则左子树上所有结点的值均小于它的根结点的值；若它的右子树不空，则右子树上所有结点的值均大于它的根结点的值；它的左、右子树也分别为二叉排序树。

```
public class Solution {
    TreeNode head=null;
    TreeNode realHead=null;
    public TreeNode Convert(TreeNode pRootOfTree) {
        convertSub(pRootOfTree);
        return realHead;
    }
    private void convertSub(TreeNode pRootOfTree) {
        if(pRootOfTree==null) {
            return;
        }
        convertSub(pRootOfTree.left);
        if(head==null) {
            head=pRootOfTree;
            realHead=pRootOfTree;
        }
        else {
            head.right=pRootOfTree;
            pRootOfTree.left=head;
            head=pRootOfTree;
        }
        convertSub(pRootOfTree.right);
    }
}
```

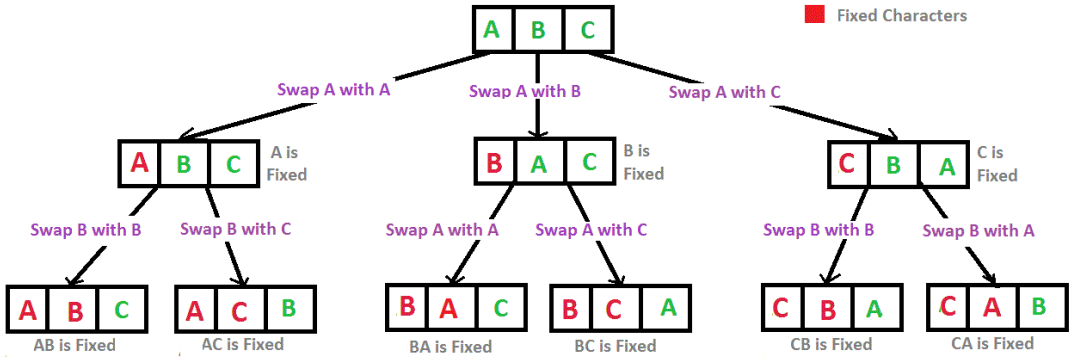
题目描述 27 字符串所有字符排列

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。

基于回溯法思想：



Recursion Tree for Permutations of String "ABC"

这一段就是回溯法，这里以"abc"为例

递归的思想与栈的入栈和出栈是一样的,某一个状态遇到 return 结束了之后，会回到被调用的地方继续执行

1. 第一次进到这里是 ch=['a','b','c'],list=[],i=0，我称为 状态 A ，即初始状态

那么 j=0，swap(ch,0,0)，就是['a','b','c']，进入递归，自己调自己，只是 i 为 1，交换(0,0)位置之后的状态我称为 状态 B

//i 不等于 2，来到这里，j=1，执行第一个 swap(ch,1,1)，这个状态我称为状态 C1 ,再进入 fun 函数，此时标记为 T1，i 为 2，那么这时就进入上一个 if，将"abc"放进 list 中

//////////////-------》此时结果集为["abc"]

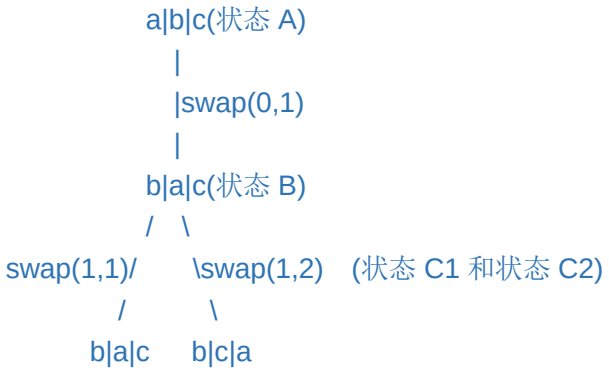
2. 执行完 list.add 之后，遇到 return，回退到 T1 处，接下来执行第二个 swap(ch,1,1)，状态 C1 又恢复为状态 B

恢复完之后，继续执行 for 循环，此时 j=2,那么 swap(ch,1,2),得到"acb"，这个状态我称为 C2,然后执行 fun，此时标记为 T2,发现 i+1=2,所以也被添加进结果集，此时 return 回退到 T2 处往下执行，此时结果集为["abc","acb"]

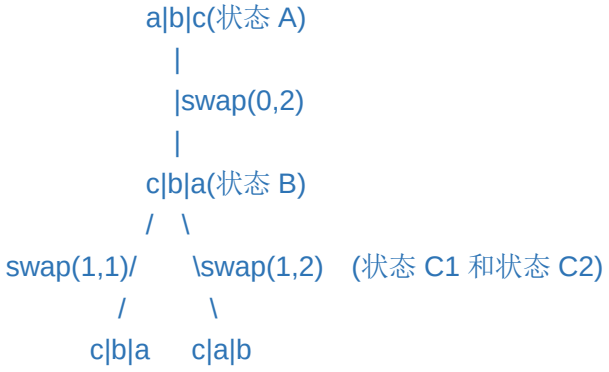
然后执行第二个 swap(ch,1,2)，状态 C2 回归状态 B,然后状态 B 的 for 循环退出回到状态 A

```
                a|b|c(状态 A)
                  |
                  |swap(0,0)
                  |
                a|b|c(状态 B)
                /   \
  swap(1,1)/       \swap(1,2)   (状态 C1 和状态 C2)
          /          \
       a|b|c       a|c|b
```

3.回到状态 A 之后，继续 for 循环，j=1,即 swap(ch,0,1)，即"bac",这个状态可以再次叫做状态 A,下面的步骤同上，此时结果集为["abc","acb","bac","bca"]



4.再继续 for 循环，j=2,即 swap(ch,0,2)，即"cab",这个状态可以再次叫做状态 A，下面的步骤同上，此时结果集为["abc","acb","bac","bca","cab","cba"]



5.最后退出 for 循环，结束。

```
import java.util.*;
public class Solution {
    public ArrayList<String> Permutation(String str) {
        ArrayList<String> res=new ArrayList<String>();
        fun(str.toCharArray(),res,0);
        Collections.sort(res);
        return res;
    }
    private void fun(char[] ch,ArrayList<String> res,int k) {
        if(k==ch.length-1) {
            if(!res.contains(new String(ch))) {
                res.add(new String(ch));
            }
        }
        for(int j=k;j<ch.length;j++) {
            swap(ch,j,k);
            fun(ch,res,k+1);
            swap(ch,j,k);
        }
    }
    private void swap(char[] ch,int i,int j) {
```

```
        if(i!=j) {
            char temp=ch[i];
            ch[i]=ch[j];
            ch[j]=temp;
        }
    }
}
```

题目描述 28 数组次数超过一半的众数

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。

法一：排序后，只有中间的数有可能为这个数字。

```
import java.util.Arrays;
public class Solution {
    public int MoreThanHalfNum_Solution(int [] array) {
        Arrays.sort(array);
        int count=0;
        for(int i=0;i<array.length;i++) {
            if(array[i]==array[array.length/2]) {
                count++;
            }
        }
        if(count>array.length/2) {
            return array[array.length/2];
        }
        else {
            return 0;
        }
    }
}
```

法二：num[数字]=次数。

```
import java.util.Arrays;
public class Solution {
    public int MoreThanHalfNum_Solution(int [] array) {
        int[] num=new int[10];
        for(int i=0;i<array.length;i++) {
            num[array[i]]++;
        }
        for(int i=0;i<num.length;i++) {
            if(num[i]>array.length/2) {
                return i;
            }
        }
        return 0;
    }
}
```

题目描述 29 数组最小的 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,。

使用最大堆保存这 k 个数，每次只和堆顶比，如果比堆顶小，删除堆顶，新数入堆。

```
import java.util.ArrayList;
import java.util.PriorityQueue;
import java.util.Comparator;
public class Solution {
    public ArrayList<Integer> GetLeastNumbers_Solution(int []
input, int k) {
        ArrayList<Integer> res=new ArrayList<Integer>();
        if(k>input.length || k==0) {
            return res;
        }
        PriorityQueue<Integer> maxHeap=new
            PriorityQueue<Integer>(k,new Comparator<Integer>() {
            public int compare(Integer o1,Integer o2) {
                return o2-o1;
            }
        });
        for(int i=0;i<input.length;i++) {
            if(maxHeap.size()!=k) {
                maxHeap.offer(input[i]);
            }
            else if(input[i]<maxHeap.peek()) {
                maxHeap.poll();
                maxHeap.offer(input[i]);
            }
        }
        for(Integer integer:maxHeap) {
            res.add(integer);
        }
        return res;
    }
}
```

使用冒泡排序遍历 k 次即可。

*基于堆排序算法，构建最大堆。时间复杂度为 O(nlogk)

*如果用快速排序，时间复杂度为 O(nlogn)；

*如果用冒泡排序，时间复杂度为 O(n*k

```
import java.util.*;
public class Solution {
    public ArrayList<Integer> GetLeastNumbers_Solution(int []
input, int k) {
        ArrayList<Integer> res=new ArrayList<Integer>();
```

```
        if(k>input.length || k=0) {
            return res;
        }
        for(int i=0;i<k;i++) {
            for(int j=0;j<input.length-i-1;j++) {
                if(input[j]<input[j+1]) {
                    int temp=input[j];
                    input[j]=input[j+1];
                    input[j+1]=temp;
                }
            }
            res.add(input[input.length-i-1]);
        }
        return res;
    }
}
```

题目描述 30 最大连续子序列的和

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)。给一个数组,返回它的最大连续子序列的和,你会不会被他忽悠住?(子向量的长度至少是 1)

法一:动态规划。

对于一个数 A,若是 A 的左边累计数非负,那么加上 A 能使得值不小于 A,认为累计值对整体和是有贡献的。如果前几项累计值负数,则认为有害于总和,total 记录当前值。

```
public class Solution {
    public int FindGreatestSumOfSubArray(int[] array) {
        if(array.length==0) {
            return 0;
        }
        int max=array[0];
        int sum=array[0];
        for(int i=1;i<array.length;i++) {
            if(sum>=0) {
                sum+=array[i];
            }
            else {
                sum=array[i];
            }
            if(sum>max) {
                max=sum;
            }
        }
        return max;
    }
}
```

法二:使用 i,j 代表头尾指针,求出所有和与 max 比较。

```
public class Solution {
    public int FindGreatestSumOfSubArray(int[] array) {
        int max=Integer.MIN_VALUE;
        int sum=0;
        for(int i=0;i<array.length;i++) {
            sum=0;
            for(int j=i;j<array.length;j++) {
                sum+=array[j];
                if(sum>max) {
                    max=sum;
                }
            }
```

```
        }
        return max;
    }
}
```

题目描述 31 非负整数中 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（从 1 到 n 中 1 出现的次数）

如果要计算百位上 1 出现的次数，它要受到 3 方面的影响：

百位上的数字，百位以下（低位）的数字，百位以上（高位）的数字。

① 如果百位上数字为 0，百位上可能出现 1 的次数由更高位决定。比如：12013，则可以知道百位出现 1 的情况，一共 1200 个。可以看出是由更高位数字（12）决定，并且等于更高位数字（12）乘以 当前位数（100）。

② 如果百位上数字为 1，百位上可能出现 1 的次数不仅受更高位影响还受低位影响。比如：12113，则可以知道百位受高位影响出现的情况一共 1200 个。和上面情况一样，并且等于更高位数字（12）乘以 当前位数（100）。但同时它还受低位影响，百位出现 1 的情况是：12100~12113,一共 114 个，等于低位数字（113）+1。

③ 如果百位上数字大于 1（2~9），则百位上出现 1 的情况仅由更高位决定，则百位出现 1 的情况一共有 1300 个，并且等于更高位数字+1（12+1）乘以当前位数（100）。

```
public class Solution {
    public int NumberOf1Between1AndN_Solution(int n) {
        int res=0;
        int i=1;
        int before=0;
        int current=0;
        int after=0;
        while((n/i)!=0) {
            before=n/(i*10);
            current=(n/i)%10;
            after=n-(n/i)*i;
            if(current==0) {
                res+=before*i;
            }
            else if(current==1) {
                res+=before*i+after+1;
            }
            else {
                res+=(before+1)*i;
            }
            i=i*10;
        }
        return res;
    }
}
```

题目描述 32 数组拼接最小数

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。

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
public class Solution {
    public String PrintMinNumber(int [] numbers) {
        String res="";
        ArrayList<Integer> list=new ArrayList<Integer>();
        for(int i=0;i<numbers.length;i++) {
            list.add(numbers[i]);
        }
        Collections.sort(list,new Comparator<Integer>() {
            public int compare(Integer i1,Integer i2) {
                String s1=i1+""+i2;
                String s2=i2+"" +i1;
                return s1.compareTo(s2);
            }
        });
        for(int l:list) {
            res+=l;
        }
        return res;
    }
}
```

题目描述 33 第 N 个丑数

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。

1.为什么分三个队列？
丑数数组里的数一定是有序的，因为我们是从丑数数组里的数乘以 2,3,5 选出的最小数，一定比以前未乘以 2,3,5 大，同时对于三个队列内部，按先后顺序乘以 2,3,5 分别放入，所以同一个队列内部也是有序的；
2.为什么比较三个队列头部最小的数放入丑数数组？
因为三个队列是有序的，所以取出三个头中最小的，等同于找到了三个队列所有数中最小的。
注意：不可以换为 else if，有可能一个词在两个队列中。

```
import java.util.ArrayList;
public class Solution {
    public int GetUglyNumber_Solution(int index) {
        if(index==0) {
            return 0;
        }
        ArrayList<Integer> list=new ArrayList<Integer>();
        list.add(1);
        int i2=0;
        int i3=0;
        int i5=0;
        while(list.size()<index) {
            int num2=list.get(i2)*2;
            int num3=list.get(i3)*3;
            int num5=list.get(i5)*5;
            int min=Math.min(num2,Math.min(num3,num5));
            list.add(min);
            if(min==num2) {
                i2++;
            }
            if(min==num3) {
                i3++;
            }
            if(min==num5) {
                i5++;
            }
        }
        return list.get(list.size()-1);
    }
}
```

题目描述 34 第一个只出现一次字符的位置

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）。

'z'+1:其实不是从 num[0]开始计数

```
public class Solution {
    public int FirstNotRepeatingChar(String str) {
        char[] c=str.toCharArray();
        int[] num=new int['z'+1];
        for(int i=0;i<c.length;i++) {
            num[c[i]]++;
        }
        for(int i=0;i<c.length;i++) {
            if(num[c[i]]==1) {
                return i;
            }
        }
        return -1;
    }
}
```

题目描述 35 数组逆序对数量

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。

归并排序的改进，把数据分成前后两个数组(递归分到每个数组仅有一个数据项)，合并数组，合并时，出现前面的数组值 array[i]大于后面数组值 array[j]时；则前面数组 array[i]~array[mid]都是大于 array[j]的，count += mid+1 – i

```
public class Solution {
    public int InversePairs(int [] array) {
        if(array==null || array.length==0) {
            return 0;
        }
        int[] copy=new int[array.length];
        int count=getCount(array,copy,0,array.length-1);
        return count;
    }
    private int getCount(int[] array,int[] copy,int low,int high)
{
        if(low==high) {
            return 0;
        }
        int mid=(low+high)/2;
        int leftCount=getCount(array,copy,low,mid)%1000000007;
        int rightCount=getCount(array,copy,mid+1,high)%1000000007;
        int count=0;
        int i=mid;
        int j=high;
        int indexCopy=high;
        while(i>=low && j>mid) {
            if(array[i]>array[j]) {
                count+=j-mid;
                copy[indexCopy--]=array[i--];
                if(count>=1000000007) {
                    count%=1000000007;
                }
            }
            else {
                copy[indexCopy--]=array[j--];
            }
        }
        while(i>=low) {
            copy[indexCopy--]=array[i--];
        }
        while(j>mid) {
```

```
            copy[indexCopy--]=array[j--];
        }
        for(int s=low;s<=high;s++) {
            array[s]=copy[s];
        }
        return (leftCount+rightCount+count)%1000000007;
    }
}
```

题目描述 36 链表公共节点

输入两个链表，找出它们的第一个公共结点。

长度相同：直接遍历到结果。

长度不同，第一次循环遍历差值，第二次循环遍历共同节点。

假定 List1 长度: a+n List2 长度:b+n, 且 a<b

那么 p1 会先到链表尾部，这时 p2 走到 a+n 位置,将 p1 换成 List2 头部

接着 p2 再走 b+n-(n+a) =b-a 步到链表尾部,这时 p1 也走到 List2 的 b-a 位置，还差 a 步就到可能的第一个公共节点。

将 p2 换成 List1 头部，p2 走 a 步也到可能的第一个公共节点。如果恰好 p1==p2,那么 p1 就是第一个公共节点。 或者 p1 和 p2 一起走 n 步到达列表尾部，二者没有公共节点，退出循环。 同理 a>=b.时间复杂度 O(n+a+b)

```
/*
public class ListNode {
    int val;
    ListNode next = null;

    ListNode(int val) {
        this.val = val;
    }
}*/
public class Solution {
    public ListNode FindFirstCommonNode(ListNode pHead1, ListNode
pHead2) {
        ListNode p1=pHead1;
        ListNode p2=pHead2;
        while(p1!=p2) {
            p1=(p1==null?pHead2:p1.next);
            p2=(p2==null?pHead1:p2.next);
        }
        return p1;
    }
}
```

题目描述 37 数字在排序数组中的次数

统计一个数字在排序数组中出现的次数。

有序数组应考虑二分（递归+循环）。

```
public class Solution {
    public int GetNumberOfK(int [] array , int k) {
       if(array.length==0) {
          return 0;
       }
        int firstK=getFirstK(array,k,0,array.length-1);
        int lastK=getLastK(array,k,0,array.length-1);
        if(firstK!=-1 && lastK!=-1) {
            return lastK-firstK+1;
        }
        return 0;
    }
    private int getFirstK(int[] array,int k,int start,int end) {
        int mid=(start+end)/2;
        while(start<=end) {
            if(array[mid]>k) {
                end=mid-1;
            }
            else if(array[mid]<k) {
                start=mid+1;
            }
            else if(mid-1>=start && array[mid-1]==k) {
                end=mid-1;
            }
            else {
                return mid;
            }
            mid=(start+end)/2;
        }
        return -1;
    }
    private int getLastK(int[] array,int k,int low,int high) {
        int mid=(low+high)/2;
        while(low<=high) {
            if(array[mid]>k) {
                high=mid-1;
            }
            else if(array[mid]<k) {
                low=mid+1;
            }
            else if(mid+1<=high && array[mid+1]==k) {
```

```
                low=mid+1;
            }
            else {
                return mid;
            }
            mid=(low+high)/2;
        }
        return -1;
    }
}
```

题目描述 38 二叉树的深度（最长路径）

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。

```
/**
public class TreeNode {
    int val = 0;
    TreeNode left = null;
    TreeNode right = null;

    public TreeNode(int val) {
        this.val = val;

    }
}
*/
public class Solution {
    public int TreeDepth(TreeNode root) {
        if(root==null) {
            return 0;
        }
        return
Math.max(TreeDepth(root.left)+1,TreeDepth(root.right)+1);
    }
}
```

题目描述 39 是否平衡二叉树

输入一棵二叉树，判断该二叉树是否是平衡二叉树。

平衡二叉树：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。

```
public class Solution {
    public boolean IsBalanced_Solution(TreeNode root) {
        return getDepth(root)!=-1;
    }
    private int getDepth(TreeNode root) {
        if(root==null) {
            return 0;
        }
        int left=getDepth(root.left);
        if(left==-1) {
            return -1;
        }
        int right=getDepth(root.right);
        if(right==-1) {
            return -1;
        }
        return Math.abs(left-right)>1? -1:1+Math.max(left,right);
    }
}
```

题目描述 40 整型数组中只出现一次（其他两次）

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

第二个 for()的意思是：sum 的二进制表示中，1 的位数，表示的是两个唯一数字二进制表示中不同的位，我们就找出第一个 1 所在的位数(index)，在第三个 for()循环中按照这个位将数组分成两个子数组，分组标准是数字在这个位上的值是否为 1（此时数字相同的各位也相同，在同一个组中；不同数字，也就不在同一组里）。之后按照异或分别找出那两个唯一数即可。

```
//num1,num2 分别为长度为 1 的数组。传出参数
//将 num1[0],num2[0]设置为返回结果
public class Solution {
    public void FindNumsAppearOnce(int [] array,int num1[] , int
num2[]) {
        if(array==null || array.length<=1) {
            num1[0]=0;
            num2[0]=0;
            return;
        }
        int sum=0;
        int index=0;
        for(int i=0;i<array.length;i++) {
            sum^=array[i];
        }
        for(;index<32;index++) {
            if((sum&(1<<index))!=0) {
                break;
            }
        }
        for(int i=0;i<array.length;i++) {
            if((array[i]&(1<<index))!=0) {
                num1[0]^=array[i];
            }
            else {
                num2[0]^=array[i];
            }
        }
    }
}
```

问题：数组 a 中只有一个数出现一次，其他数都出现了 2 次，找出这个数字。

```
    public static int find1From2(int[] a){
        int len = a.length, res = 0;
        for(int i = 0; i < len; i++){
            res = res ^ a[i];
        }
```

```
        return res;
    }
```

问题：数组 a 中只有一个数出现一次，其他数字都出现了 3 次，找出这个数字。

```
    public static int find1From3(int[] a){
        int[] bits = new int[32];
        int len = a.length;
        for(int i = 0; i < len; i++){
            for(int j = 0; j < 32; j++){
                bits[j] = bits[j] + ( (a[i]>>>j) & 1);
            }
        }
        int res = 0;
        for(int i = 0; i < 32; i++){
            if(bits[i] % 3 !=0){
                res = res | (1 << i);
            }
        }
        return res;
    }
```

题目描述 41 所有和为 S 的连续序列

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。

1）由于我们要找的是和为 S 的连续正数序列，因此这个序列是个公差为 1 的等差数列，而这个序列的中间值代表了平均值的大小。假设序列长度为 n，那么这个序列的中间值可以通过（S / n）得到，知道序列的中间值和长度，也就不难求出这段序列了。

2）满足条件的 n 分两种情况：

n 为奇数时，序列中间的数正好是序列的平均值，所以条件为：(n & 1) == 1 && sum % n == 0;

n 为偶数时，序列中间两个数的平均值是序列的平均值，而这个平均值的小数部分为 0.5，所以条件为：(sum % n) * 2 == n.

3）由题可知 n >= 2，那么 n 的最大值是多少呢？我们完全可以将 n 从 2 到 S 全部遍历一次，但是大部分遍历是不必要的。为了让 n 尽可能大，我们让序列从 1 开始，根据等差数列的求和公式：S = (1 + n) * n / 2，得到$n < \sqrt{2S}$.

最后举一个例子，假设输入 sum = 100，我们只需遍历 n = 13~2 的情况（按题意应从大到小遍历），n = 8 时，得到序列[9, 10, 11, 12, 13, 14, 15, 16]；n = 5 时，得到序列[18, 19, 20, 21, 22]。完整代码：时间复杂度为$O(\sqrt{n})$。

```
import java.util.ArrayList;
public class Solution {
    public ArrayList<ArrayList<Integer> >
FindContinuousSequence(int sum) {
        ArrayList<ArrayList<Integer>> res=new
ArrayList<ArrayList<Integer>>();
        for(int n=(int)Math.sqrt(2*sum);n>=2;n--) {
            if(((n&1)==1 && sum%n==0) || (sum%n)*2==n) {
                ArrayList<Integer> list=new ArrayList<Integer>();
                for(int k=(sum/n)-(n-1)/2,j=0;j<n;k++,j++) {
                    list.add(k);
                }
                res.add(list);
            }
        }
        return res;
    }
}
```

题目描述 42 递增数组中两个数的和为 S，选择乘积最小的，小的数先输出

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。

输出描述:

对应每个测试案例，输出两个数，小的先输出。

```
import java.util.ArrayList;
public class Solution {
    public ArrayList<Integer> FindNumbersWithSum(int [] array,int
sum) {
        ArrayList<Integer> res=new ArrayList<Integer>();
        if(array==null || array.length<2) {
            return res;
        }
        int i=0;
        int j=array.length-1;
        while(i<j) {
            if(array[i]+array[j]==sum) {
                res.add(array[i]);
                res.add(array[j]);
                return res;
            }
            else if(array[i]+array[j]<sum) {
                i++;
            }
            else if(array[i]+array[j]>sum) {
                j--;
            }
        }
        return res;
    }
}
```

题目描述 43 字符串循环左移 N 位

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！

```
public class Solution {
    public String LeftRotateString(String str,int n) {
        if(str.length()<n) {
            return "";
        }
        char[] ch=str.toCharArray();
        reverse(ch,0,n-1);
        reverse(ch,n,ch.length-1);
        reverse(ch,0,ch.length-1);
        return new String(ch);
    }
    private void reverse(char[] ch,int low,int high) {
        while(low<high) {
            char temp=ch[low];
            ch[low]=ch[high];
            ch[high]=temp;
            low++;
            high--;
        }
    }
}
```

题目描述 44 翻转句子中的单词

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？

```
public class Solution {
    public String ReverseSentence(String str) {
        if(str.length()==0) {
            return "";
        }
        int blank=-1;
        char[] ch=str.toCharArray();
        reverse(ch,0,ch.length-1);
        for(int i=0;i<ch.length;i++) {
            if(ch[i]==' ') {
                reverse(ch,blank+1,i-1);
                blank=i;
            }
        }
        reverse(ch,blank+1,ch.length-1);
        return new String(ch);
    }
    private void reverse(char[] ch,int low,int high) {
        while(low<high) {
            char temp=ch[low];
            ch[low]=ch[high];
            ch[high]=temp;
            low++;
            high--;
        }
    }
}
```

题目描述 45 纸牌抽顺子（两张大小王为任意数）

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。

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;
    }
}
```

题目描述 46 m 个小朋友，报数 n-1 的小朋友出，求最后剩下的小朋友

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)

```
public class Solution {
    public int LastRemaining_Solution(int n, int m) {
        if(n<1 || m<1) {
            return -1;
        }
        int[] array=new int[n];
        int i=-1;
        int step=0;
        int count=n;  // 剩余人数
        while(count>0) {
            i++;
            if(i>=n) {  // 循环
                i=0;
            }
            if(array[i]==-1) {  // 出去的小朋友跳过
                continue;
            }
            step++;
            if(step==m) {
                array[i]=-1;  // 出去
                step=0;  // 重新计步数
                count--;
            }
        }
        return i;
    }
}
```

题目描述 47 求 1 到 n 的和

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）。

1.需利用逻辑与的短路特性实现递归终止。

2.当 n==0 时，(n>0)&&((sum+=Sum_Solution(n-1))>0)只执行前面的判断，为 false，然后直接返回 0；

3.当 n>0 时，执行 sum+=Sum_Solution(n-1)，实现递归计算 Sum_Solution(n)，0 可任意取值。

```
public class Solution {
    public int Sum_Solution(int n) {
        int sum=n;
        boolean flag=(n>0)&&((sum+=Sum_Solution(n-1))>0);
        return sum;
    }
}
```

题目描述 48 求两个整数的和

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。

两个数异或：相当于每一位相加，而不考虑进位；

两个数相与，并左移一位：相当于求得进位；

将上述两步的结果相加

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

题目描述 49 字符串转换成整数，不合法返回 0

将一个字符串转换成一个整数(实现 Integer.valueOf(string)的功能，但是 string 不符合数字要求时返回 0)，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。

```
public class Solution {
    public int StrToInt(String str) {
        if(str.length()==0 || str=="") {
            return 0;
        }
        int start=0;
        int symbol=1;
        char[] c=str.toCharArray();
        if(c[0]=='+') {
            start=1;
        }
        else if(c[0]=='-') {
            start=1;
            symbol=-1;
        }
        int result=0;
        for(int i=start;i<c.length;i++) {
            if(c[i]>'9' || c[i]<'0') {
                return 0;
            }
            result=result*10+c[i]-'0';
        }
        return result*symbol;
    }
}
```

题目描述 50 返回任意重复的元素，元素数值均小于长度

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。

Parameters:

numbers: an array of integers

length: the length of array numbers

duplication: (Output) the duplicated number in the array number,length of duplication array is 1,so using duplication[0] = ? in implementation;

Here duplication like pointor in C/C++, duplication[0] equal *duplication in C/C++

这里要特别注意~返回任意重复的一个，赋值 duplication[0]

Return value: true if the input is valid, and there are some duplications in the array number

otherwise false

题目里写了数组里数字的范围保证在 0 ~ n-1 之间，所以可以利用现有数组设置标志，当一个数字被访问过后，可以设置对应位上的数 + n，之后再遇到相同的数时，会发现对应位上的数已经大于等于 n 了，那么直接返回这个数即可。

```
public class Solution {
    public boolean duplicate(int numbers[],int length,int []
duplication) {
        for(int i=0;i<length;i++) {
            int index=numbers[i];
            if(index>=length) {
                index=index-length;
            }
            if(numbers[index]>=length) {
                duplication[0]=index;
                return true;
            }
            numbers[index]=numbers[index]+length;
        }
        duplication[0]=-1;
        return false;
    }
}
```

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

```
                duplication[0]=numbers[i];
                return true;
            }
            k[numbers[i]]=true;
        }
        duplication[0]=-1;
        return false;
    }
}
```

题目描述 51 构建数组，B[i]= A[0]*A[1]*...*A[i-1]*A[i+1]*...*A[n-1]，不使用除法

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。

```
import java.util.ArrayList;
public class Solution {
    public int[] multiply(int[] A) {
        int length=A.length;
        int[] B=new int[A.length];
        if(length==0) {
            return B;
        }
        B[0]=1;
        for(int i=1;i<length;i++) {
            B[i]=A[i-1]*B[i-1];
        }
        int temp=1;
        for(int j=length-2;j>=0;j--) {
            temp=temp*A[j+1];
            B[j]=B[j]*temp;
        }
        return B;
    }
}
```

题目描述 52 判断与包括'.'、'*'的正则表达式是否匹配

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

dp[i+1][j+1]=

- dp[i+1][j-1]：匹配 0 个
- dp[i+1][j]：匹配一个
- dp[i][j+1])：匹配多个

```
public class Solution {
    public boolean match(char[] str, char[] pattern) {
        if(str==null || pattern==null) {
            return false;
        }
        boolean[][] dp=new boolean[str.length+1][pattern.length+1];
        dp[0][0]=true;
        for(int j=0;j<pattern.length;j++) {
            if(pattern[j]=='*' && dp[0][j-1]) {
                dp[0][j+1]=true;
            }
        }
        for(int i=0;i<str.length;i++) {
            for(int j=0;j<pattern.length;j++) {
                if(pattern[j]=='.' || pattern[j]==str[i]) {
                    dp[i+1][j+1]=dp[i][j];
                }
                if(pattern[j]=='*') {
                    if(pattern[j-1]!=str[i] && pattern[j-1]!='.') {
                        dp[i+1][j+1]=dp[i+1][j-1];
                    }
                    else {
                        dp[i+1][j+1]=(dp[i+1][j-1] || dp[i+1][j] ||
dp[i][j+1]);
                    }
                }
            }
        }
        return dp[str.length][pattern.length];
    }
}
```

题目描述 53 判断字符串是否表示数值

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:

XY: 表示 X 后面跟着 Y，这里 X 和 Y 分别是正则表达式的一部分

(X):子表达式，将 X 看做是一个整体

[abc] : 表示可能是 a，可能是 b，也可能是 c。

[^abc]: 表示不是 a,b,c 中的任意一个

[a-zA-Z]: 表示是英文字母

[0-9]:表示是数字

?: 表示出现 0 次或 1 次

+: 表示出现 1 次或多次

*: 表示出现 0 次、1 次或多次

```
public class Solution {
    private int index=0;
    public boolean isNumeric(char[] str) {
        String string=String.valueOf(str);
        return string.matches("[\\+\\-]?[0-9]*(\\.[0-
9]+)?([eE][\\+\\-]?[0-9]+)?");
    }
}

public class Solution {
    private int index=0;
    public boolean isNumeric(char[] str) {
        if(str.length<1) {
            return false;
        }
        boolean flag=scanInteger(str);
        if(index<str.length && str[index]=='.') {
            index++;
            flag=scanUnsignedInteger(str) || flag;
        }
        if(index<str.length && (str[index]=='e' ||
str[index]=='E')) {
            index++;
            flag=scanInteger(str) && flag;
        }
        return flag && index==str.length;
    }
    private boolean scanInteger(char[] str) {
        if(index<str.length && (str[index]=='+' || str[index]=='-
')) {
```

```
            index++;
        }
        return scanUnsignedInteger(str);
    }
    private boolean scanUnsignedInteger(char[] str) {
        int start=index;
        while(index<str.length && str[index]>='0' &&
str[index]<='9') {
            index++;
        }
        return start<index;
    }
}
```

题目描述 54 找出字符流中第一个只出现一次的字符

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。

```
public class Solution {
    //Insert one char from stringstream
    StringBuilder sb=new StringBuilder();
    int[] num=new int['z'+1];
    public void Insert(char ch)
    {
        sb.append(ch);
        num[ch]+=1;
    }
  //return the first appearence once char in current stringstream
    public char FirstAppearingOnce()
    {
        char[] ch=sb.toString().toCharArray();
        for(char c:ch) {
            if(num[c]==1) {
                return c;
            }
        }
        return '#';
    }
}
```

题目描述 55 链表的入口结点

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。

第一步，找环中相汇点。分别用 slow,fast 指向链表头部，slow 每次走一步，fast 每次走二步，直到 slow==fast 找到在环中的相汇点。

第二步，找环的入口。接上步，当 slow==fast 时，fast 所经过节点数为 2x, slow 所经过节点数为 x,设环中有 n 个节点, fast 比 slow 多走一圈(n 圈也可以)有 2x=n+x; n=x;可以看出 slow 实际走了一个环的步数，再让 fast 指向链表头部，slow 位置不变，slow, fast 每次走一步直到 slow==fast; 此时 slow 指向环的入口。

```
/*
 public class ListNode {
    int val;
    ListNode next = null;

    ListNode(int val) {
        this.val = val;
    }
}
*/
public class Solution {
    public ListNode EntryNodeOfLoop(ListNode pHead) {
        if(pHead==null || pHead.next==null) {
            return null;
        }
        ListNode slow=pHead.next;
        ListNode fast=pHead.next.next;
        while(fast!=null && fast.next!=null) {
            fast=fast.next.next;
            slow=slow.next;
            if(fast==slow) {
                fast=pHead;
                while(fast!=slow) {
                    fast=fast.next;
                    slow=slow.next;
                }
                return slow;
            }
        }
        return null;
    }
}
```

题目描述 56 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

```
/*
 public class ListNode {
    int val;
    ListNode next = null;

    ListNode(int val) {
        this.val = val;
    }
}
*/
public class Solution {
    public ListNode deleteDuplication(ListNode pHead) {
        ListNode first=new ListNode(-1);
        first.next=pHead;
        ListNode p=pHead;
        ListNode last=first;
        while(p!=null && p.next!=null) {
            if(p.val==p.next.val) {
                int val=p.val;
                while(p!=null && p.val==val) {
                    p=p.next;
                }
                last.next=p;  // 这里不可 last=p;p=p.next;可能两个连续重复数值
            }
            else {
                last=p;
                p=p.next;
            }
        }
        return first.next;
    }
}
```

题目描述 57 中序遍历的下一个结点

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。

中序遍历：左根右。

当有右结点时，返回右结点中的最左结点。

当没有右结点时，寻找第一个当前节点是父节点左孩子的节点。

```
/*
public class TreeLinkNode {
    int val;
    TreeLinkNode left = null;
    TreeLinkNode right = null;
    TreeLinkNode next = null;

    TreeLinkNode(int val) {
        this.val = val;
    }
}
*/
public class Solution {
    public TreeLinkNode GetNext(TreeLinkNode pNode) {
        if(pNode==null) {
            return null;
        }
        if(pNode.right!=null) {
            pNode=pNode.right;
            while(pNode.left!=null) {
                pNode=pNode.left;
            }
            return pNode;
        }
        while(pNode.next!=null) {
            if(pNode.next.left==pNode) {
                return pNode.next;
            }
            pNode=pNode.next;
        }
        return null;
    }
}
```

题目描述 58 二叉树是否镜像

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。

```
/*
public class TreeNode {
    int val = 0;
    TreeNode left = null;
    TreeNode right = null;

    public TreeNode(int val) {
        this.val = val;
    }
}
*/
public class Solution {
    boolean isSymmetrical(TreeNode pRoot)
    {
        if(pRoot==null) {
            return true;
        }
        return comRoot(pRoot.left,pRoot.right);
    }
    private boolean comRoot(TreeNode left,TreeNode right) {
        if(left==null) {
            return right==null;
        }
        if(right==null) {
            return false;
        }
        if(left.val!=right.val) {
            return false;
        }
        return
comRoot(left.right,right.left)&&comRoot(left.left,right.right);
    }
}
```

题目描述 59 之字形打印二叉树

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。

```
import java.util.ArrayList;
import java.util.Stack;/*
public class TreeNode {
    int val = 0;
    TreeNode left = null;
    TreeNode right = null;
    public TreeNode(int val) {
        this.val = val;
    }
}
*/
public class Solution {
    public ArrayList<ArrayList<Integer> > Print(TreeNode pRoot) {
        int layer=1;
        Stack<TreeNode> stack1=new Stack<TreeNode>();
        stack1.push(pRoot);
        Stack<TreeNode> stack2=new Stack<TreeNode>();
        ArrayList<ArrayList<Integer>> res=new
ArrayList<ArrayList<Integer>>();
        if(pRoot==null) {
            return res;
        }
        while(!stack1.isEmpty() || !stack2.isEmpty()) {
            if(layer%2==1) {
                ArrayList<Integer> layerlist=new
ArrayList<Integer>();
                while(!stack1.isEmpty()) {
                    TreeNode node=stack1.pop();
                    layerlist.add(node.val);
                    if(node.left!=null) {
                        stack2.push(node.left);
                    }
                    if(node.right!=null) {
                        stack2.push(node.right);
                    }
                }
                res.add(layerlist);
                layer++;
            }
            else {
```

```
            ArrayList<Integer> layerlist=new
ArrayList<Integer>();
            while(!stack2.isEmpty()) {
                TreeNode node=stack2.pop();
                layerlist.add(node.val);
                if(node.right!=null) {
                    stack1.push(node.right);
                }
                if(node.left!=null) {
                    stack1.push(node.left);
                }
            }
            res.add(layerlist);
            layer++;
        }
    }
    return res;
  }
}
```

题目描述 60 上下左右打印二叉树

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。

```
import java.util.ArrayList;
import java.util.LinkedList;
/*
public class TreeNode {
    int val = 0;
    TreeNode left = null;
    TreeNode right = null;

    public TreeNode(int val) {
        this.val = val;

    }

}
*/
```

使用深度递归的方式。

```
public class Solution {
    ArrayList<ArrayList<Integer> > Print(TreeNode pRoot) {
        ArrayList<ArrayList<Integer>> res=new
ArrayList<ArrayList<Integer>>();
        depth(pRoot,1,res);
        return res;
    }
    private void depth(TreeNode root,int
depth,ArrayList<ArrayList<Integer>> res) {
        if(root==null) {
            return;
        }
        if(depth>res.size()) {
            res.add(new ArrayList<Integer>());
        }
        res.get(depth-1).add(root.val);
        depth(root.left,depth+1,res);
        depth(root.right,depth+1,res);
    }
}
```

ArrayList 是实现了基于动态数组的数据结构，LinkedList 基于链表的数据结构。

对于随机访问 get 和 set，ArrayList 觉得优于 LinkedList，因为 LinkedList 要移动指针。

对于新增和删除操作 add 和 remove，LinedList 比较占优势，因为 ArrayList 要移动数据。

使用链表来存储结点并打印。Start 和 end 来表示一行是否结束。

```
public class Solution {
```

```
    ArrayList<ArrayList<Integer> > Print(TreeNode pRoot) {
        ArrayList<ArrayList<Integer>> res=new
ArrayList<ArrayList<Integer>>();
        if(pRoot==null) {
            return res;
        }
        LinkedList<TreeNode> queue=new LinkedList<TreeNode>();
        ArrayList<Integer> layerlist=new ArrayList<Integer>();
        queue.add(pRoot);
        while(!queue.isEmpty()) {
            int size=queue.size();
            while(size!=0) {
                TreeNode node=queue.remove();
                layerlist.add(node.val);
                size--;
                if(node.left!=null) {
                    queue.add(node.left);
                }
                if(node.right!=null) {
                    queue.add(node.right);
                }
            }
            res.add(layerlist);
            layerlist=new ArrayList<Integer>();
        }
        return res;
    }
}
```

题目描述 61 序列化二叉树和反序列化二叉树

请实现两个函数，分别用来序列化和反序列化二叉树

```
/*
public class TreeNode {
    int val = 0;
    TreeNode left = null;
    TreeNode right = null;

    public TreeNode(int val) {
        this.val = val;
    }
}
*/
public class Solution {
    int index=-1;
    String Serialize(TreeNode root) {
        StringBuffer sb=new StringBuffer();
        if(root==null) {
            sb.append("#,");
            return sb.toString();
        }
        sb.append(root.val+",");
        sb.append(Serialize(root.left));
        sb.append(Serialize(root.right));
        return sb.toString();
  }
    TreeNode Deserialize(String str) {
        index++;
        int len=str.length();
        if(index>len) {
            return null;
        }
        String[] strr=str.split(",");
        TreeNode node=null;
        if(!strr[index].equals("#")) {
            node=new TreeNode(Integer.valueOf(strr[index]));
            node.left=Deserialize(str);
            node.right=Deserialize(str);
        }
        return node;
  }
}
```

题目描述 62* 二叉搜索树第 k 小的结点

给定一棵二叉搜索树，请找出其中的第 k 小的结点。例如，（5，3，7，2，4，6，8） 中，按结点数值大小顺序第三小结点的值为 4。

```
/*
public class TreeNode {
    int val = 0;
    TreeNode left = null;
    TreeNode right = null;

    public TreeNode(int val) {
        this.val = val;
    }
}
*/
```

必须要对每一个递归调用返回值进行判断 if(node != null){return node;},
判断返回值是否为 null，如果为 null 就说明在没找到，要继续执行 index++ ; if(index == k){...}的寻找过程，
如果返回不为空，则说明在递归调用判断子节点的时候已经找到符合要求的节点了，则将找到的节点
逐层向上传递返回。直到返回到第一次调用 KthNode 的地方。
如果不对递归调用的返回值做判断，即不执行 if(node != null){return node;}，那所找到符合 要求的节点只能返回到上一层，不能返回到顶层（可以自己调试，然后观察方法栈的调用变化）

```
public class Solution {
    int index=0;
    TreeNode KthNode(TreeNode pRoot, int k) {
        if(pRoot!=null) {
            TreeNode node=KthNode(pRoot.left,k);
            if(node!=null) {
                return node;
            }
            index++;
            if(index==k) {
                return pRoot;
            }
            node=KthNode(pRoot.right,k);
            if(node!=null) {
                return node;
            }
        }
        return null;
    }
}
```

题目描述 63 数据流的中位数

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。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。

使用大根堆和小根堆。

```
import java.util.PriorityQueue;
import java.util.Comparator;
public class Solution {
    private int count=0;
    private PriorityQueue<Integer> minHeap=new PriorityQueue<>();
    private PriorityQueue<Integer> maxHeap=new PriorityQueue<>(new
Comparator<Integer>() {
        public int compare(Integer o1,Integer o2) {
            return o2-o1;
        }
    });
    public void Insert(Integer num) {
        if(count%2==0) {
            maxHeap.offer(num);
            int filiterMaxNum=maxHeap.poll();
            minHeap.offer(filiterMaxNum);
        }
        else {
            minHeap.offer(num);
            int filiterMinNum=minHeap.poll();
            maxHeap.offer(filiterMinNum);
        }
        count++;
    }

    public Double GetMedian() {
        if(count%2==0) {
            return new Double(maxHeap.peek()+minHeap.peek())/2;
        }
        else {
            return new Double(minHeap.peek());
        }
    }
}
```

题目描述 64 所有滑动窗口里的最大值

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]}。

用一个双端队列保存索引，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。

```
import java.util.ArrayList;
import java.util.LinkedList;
public class Solution {
    public ArrayList<Integer> maxInWindows(int [] num, int size) {
        ArrayList<Integer> res=new ArrayList<Integer>();
        if(size<=0 || num.length<size) {
            return res;
        }
        LinkedList<Integer> qmax=new LinkedList<Integer>();
        for(int i=0;i<num.length;i++) {
            while(!qmax.isEmpty() && num[qmax.peekLast()]<num[i]) {
                qmax.pollLast();
            }
            qmax.add(i);
            if(qmax.peekFirst()==i-size) {
                qmax.pollFirst();
            }
            if(i>=size-1) {
                res.add(num[qmax.peekFirst()]);
            }
        }
        return res;
    }
}
```

题目描述 65 二维矩阵是否包括字符串路径

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，则之后不能再次进入这个格子。 例如 a b c e s f c s a d e e 这样的 3 X 4 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。

回溯算法。

```
public class Solution {
    public boolean hasPath(char[] matrix, int rows, int cols,
char[] str) {
        boolean[] flag=new boolean[matrix.length];
        for(int i=0;i<rows;i++) {
            for(int j=0;j<cols;j++) {
                if(helper(matrix,rows,cols,i,j,str,0,flag)) {
                    return true;
                }
            }
        }
        return false;
    }
    private boolean helper(char[] matrix,int rows,int cols,int
i,int j,
                           char[] str,int k,boolean[] flag) {
        int index=i*cols+j;
        if(i<0 || i>=rows || j<0 || j>=cols ||
matrix[index]!=str[k] || flag[index]==true) {
            return false;
        }
        if(k==str.length-1) {
            return true;
        }
        flag[index]=true;
        if(helper(matrix,rows,cols,i-1,j,str,k+1,flag)
          || helper(matrix,rows,cols,i+1,j,str,k+1,flag)
          || helper(matrix,rows,cols,i,j-1,str,k+1,flag)
          || helper(matrix,rows,cols,i,j+1,str,k+1,flag)) {
            return true;
        }
        flag[index]=false;
        return false;
    }
}
```

题目描述 66 机器人可到达格子数，位数相加小于 k

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？

```
public class Solution {
    public int movingCount(int threshold, int rows, int cols) {
        boolean[][] flag=new boolean[rows][cols];
        return helper(0,0,rows,cols,threshold,flag);
    }
    private int helper(int i,int j,int rows,int cols,
                       int threshold,boolean[][] flag) {
        if(i<0 || i>=rows || j<0 || j>=cols ||
sumnum(i)+sumnum(j)>threshold
           || flag[i][j]==true) {
            return 0;
        }
        flag[i][j]=true;
        return helper(i-1,j,rows,cols,threshold,flag)
            +helper(i+1,j,rows,cols,threshold,flag)
            +helper(i,j-1,rows,cols,threshold,flag)
            +helper(i,j+1,rows,cols,threshold,flag)+1;
    }
    private int sumnum(int i) {
        int sum=0;
        while(i>0) {
            sum+=i%10;
            i=i/10;
        }
        return sum;
    }
}
```

题目描述 67 快速排序

```
public class sort2 {
    public void main(int[] num) {
        sort(num,0,num.length-1);
    }
    Private void sort(int[] array,int low,int high) {
        if(low>high) {
            return;
        }
        int i=low+1;
        int ht=high;
        int lt=low;
        int val=array[low];
        while(i<=ht) {
            if(array[i]<val) {
                swap(array,i,lt);
                i+=1;
                lt+=1;
            }
            else if(array[i]>val) {
                swap(array,i,ht);
                ht--;
            }
            else {
                i++;
            }
        }
        sort(array,low,lt-1);
        sort(array,ht+1,high);
    }
    private static void swap(int[] array,int i,int j) {
        int temp=array[i];
        array[i]=array[j];
        array[j]=temp;
    }
}
```

题目描述 68 归并排序

```
public class sort3 {
    public void sort(int[] num) {
        MergeSort(num,0,num.length-1);
    }
    private void MergeSort(int[] array,int low,int high) {
        if(low==high) {
            return;
        }
        int middle=(low+high)/2;
        MergeSort(array,low,middle);
        MergeSort(array,middle+1,high);
        MergeArr(array,low,middle,high);
    }
    private void MergeArr(int[] array,int low,int middle,int high)
{
        int[] copy=new int[high-low+1];
        int i=low;
        int j=middle+1;
        int copyIndex=0;
        while(i<=middle && j<=high) {
            copy[copyIndex++]=array[i]<array[j]?
array[i++]:array[j++];
        }
        while(i<=middle) {
            copy[copyIndex++]=array[i++];
        }
        while(j<=high) {
            copy[copyIndex++]=array[j++];
        }
        int index=0;
        while(low<=high) {
            array[low++]=copy[index++];
        }
    }
}
```

题目描述 69 堆排序

```
public class sort5 {
    public <E extends Comparable> void heapSort(E[] list){
        Heap<E> heap=new Heap<E>(list);
        for (int i=list.length-1;i>=0;i--){
            list[i]=heap.remove();
        }
    }
    public void main(int[] list) {
        heapSort(list);
        for (int i=0;i<list.length;i++) {
            System.out.print(list[i]+" ");
        }
    }
}
class Heap<E extends Comparable> {
    private ArrayList<E> list=new ArrayList<E>();
    public Heap(){
    }
    public Heap(E[] Objects){
        for (int i=0;i<Objects.length;i++){
            add(Objects[i]);
        }
    }
    public void add(E newObject){
        list.add(newObject);
        int currentIndex=list.size()-1;
        while (currentIndex>0){
            int parentIndex=(currentIndex-1)/2;
            if
(list.get(currentIndex).compareTo(list.get(parentIndex))>0){
                E temp=list.get(currentIndex);
                list.set(currentIndex,list.get(parentIndex));
                list.set(parentIndex,temp);
            }else {
                break;
            }
            currentIndex=parentIndex;
        }
    }
    public E remove(){
        if (list.size()==0){
            return null;
        }
```

```
        E removedObject=list.get(0);
        list.set(0,list.get(list.size()-1));
        list.remove(list.size()-1);
        int currentIndex=0;
        while (currentIndex<list.size()){
            int leftChildIndex=2*currentIndex+1;
            int rightChildIndex=2*currentIndex+2;
            if (leftChildIndex>=list.size()){
                break;//超出范围说明该节点已经不具备子节点，是程序结束的标志
            }
            int maxIndex=leftChildIndex;
            //先令左边为最大下标，因为上面的判断只能得到左节点符合条件，右节点
需要在判断一次
            if (rightChildIndex<list.size()){
                if
(list.get(maxIndex).compareTo(list.get(rightChildIndex))<0){
                    maxIndex=rightChildIndex;
                }
            }
            //找到左右节点中的最大节点的下标后，进行判断，交换
            if
(list.get(currentIndex).compareTo(list.get(maxIndex))<0){
                E temp=list.get(maxIndex);
                list.set(maxIndex,list.get(currentIndex));
                list.set(currentIndex,temp);
                currentIndex=maxIndex;
            }else {
                break;
            }
        }
        return removedObject;
    }
}
```

题目描述 70 冒泡排序

```
    private static void sort1(int[] array) {
        for(int i=0;i<array.length-1;i++) {
            for(int j=0;j<array.length-1-i;j++) {
                if(array[j]>array[j+1]) {
                    swap(array,j,j+1);
                }
            }
        }
    }
```

题目描述 71 选择排序

```
    private static void sort2(int[] array) {
        for(int i=0;i<array.length-1;i++) {
            for(int j=i+1;j<array.length;j++) {
                if(array[i]>array[j]) {
                    swap(array,i,j);
                }
            }
        }
    }
```

题目描述 72 插入排序

```
    private static void sort3(int[] array) {
        for(int i=1;i<array.length;i++) {
            int j;
            int temp=array[i];
            for(j=i-1;j>=0 && array[j]>temp;j--) {
                array[j+1]=array[j];
            }
            array[j+1]=temp;
        }
    }
```

希尔排序：插入排序改良，步长从大到小调整。

桶排序：计数，将桶进行排序，打印桶次数。

题目描述 73 树前序遍历

```
public List<Integer> preorderTraversal(TreeNode root) {
       List<Integer> res = new ArrayList<Integer>();
       if (root == null) {
          return res;
       }
       Stack<TreeNode> stack = new Stack<TreeNode>();
       stack.push(root);
       while (!stack.isEmpty()) {
          TreeNode node = stack.pop();
          res.add(node.val);
          if (node.right != null) {
             stack.push(node.right);
          }
          if (node.left != null) {
             stack.push(node.left);
          }
       }
       return res;
    }
```

题目描述 74 树中序遍历

```
public static List<Integer> inorderTraversal(TreeNode root) {
       List<Integer> res = new ArrayList<Integer>();
       if (root == null) {
          return res;
       }
       Stack<TreeNode> stack = new Stack<TreeNode>();
       stack.push(root);
       TreeNode node=root;
       while(!stack.empty()) {
          if(node !=null && node.left!=null){
             stack.push(node.left);
             node = node.left;
          }else{
             node = stack.pop();
             res.add(node.val);
             if(node !=null && node.right!=null){
                stack.push(node.right);
                node = node.right;
             }
             else {
                node = null;
          }
       return res;
    }
```

题目描述 75 树后序遍历

```
public List<Integer> preorderTraversal(TreeNode root) {
       List<Integer> res = new ArrayList<Integer>();
       if (root == null) {
           return res;
       }
       Stack<TreeNode> stack = new Stack<TreeNode>();
       stack.push(root);
       TreeNode pre=null;
       while (!stack.isEmpty()) {
           TreeNode node = stack.peek();
           res.add(node.val);
           if (node.left!=null && node.right!=null ||
(pre!=null && (node.left==pre || node.right==pre))) {
              stack.pop();
              list.add(node.val);
           }
           else {
              if (node.right != null) {
                  stack.push(node.right);
              }
              if (node.left != null) {
                  stack.push(node.left);
              }
           }
       }
       return res;
   }
```

题目描述 76 股票交易问题

```
// 股票买卖一次
public int maxProfit(int[] prices) {
    int buy=Integer.MIN_VALUE;
    int sell=0;
    for(int i=0;i<prices.length;i++) {
        buy=Math.max(buy,-prices[i]);
        sell=Math.max(sell,buy+prices[i]);
    }
    return sell;
}

// 股票买卖两次
public int maxProfit(int[] prices) {
    int buy1=Integer.MIN_VALUE;
    int sell1=0;
    int buy2=Integer.MIN_VALUE;
    int sell2=0;
    for(int i=0;i<prices.length;i++) {
        buy1=Math.max(buy,-prices[i]);
        sell1=Math.max(sell,buy+prices[i]);
        buy2=Math.max(buy2,sell1-prices[i]);
        sell2=Math.max(sell2,buy2+prices[i]);
    }
    return sell2;
}

// 股票买卖无限次
public int maxProfit(int[] prices) {
    int buy=Integer.MIN_VALUE;
    int sell=0;
    for(int i=0;i<prices.length;i++) {
        buy=Math.max(buy,sell-prices[i]);
        sell=Math.max(sell,buy+prices[i]);
    }
    return sell;
}

// 股票买卖 K 次
public int maxProfit(int[] prices) {
    if(k>prices.length/2) {
        int buy=Integer.MIN_VALUE;
        int sell=0;
        for(int i=0;i<prices.length;i++) {
```

```
            buy=Math.max(buy,sell-prices[i]);
            sell=Math.max(sell,buy+prices[i]);
        }
        return sell;
    }
    int[] buy=new int[k+1];
    int[] sell=new int[k+1];
    for(int i=0;i<prices.length;i++) {
        for(int j=1;j<=k;j++) {
            buy[j]=Math.max(buy[j],sell[j-1]-prices[i]);
            sell[j]=Math.max(sell[j],buy[j]+prices[i]);
    }
    return sell[k-1];
}

// 股票一天冷却期
public int maxProfit(int[] prices) {
    int buy=Integer.MIN_VALUE;
    int sell=0;
    int pre=0;
    for(int i=0;i<prices.length;i++) {
        int temp=sell;
        buy=Math.max(buy,pre-prices[i]);
        sell=Math.max(sell,buy+prices[i]);
        pre=temp;
    }
    return sell;
}

// 股票买卖有手续费
public int maxProfit(int[] prices,int fee) {
    int buy=Integer.MIN_VALUE;
    int sell=0;
    for(int i=0;i<prices.length;i++) {
        buy=Math.max(buy,sell-fee-prices[i]);
        sell=Math.max(sell,buy+prices[i]);
    }
    return sell;
}
```

题目描述 77 硬币找零

```
// 硬币找零，动态规划
public int makeChange(int[] coins,int moneys) {
    int[] dp=new int[moneys+1];
    Arrays.fill(dp,moneys+1);
    dp[0]=0;
    for(int money=1;money<=moneys;money++) {
        for(int j=0;j<coins.length;j++) {
            if(coins[j]<=money) {
                dp[money]=Math.min(dp[money],dp[money-coins[j]]+1);
            }
        }
    }
    return dp[moneys]==moneys+1? -1:dp[moneys];
}
```

题目描述 78 KMP

```
public int strstr(String str, String dest) {
    if(dest==0) {
        return 0;
    }
    if(str==0) {
        return -1;
    }
    int[] next=kmpnext(dest);
    return kmp(str,dest,next);
}

public int kmp(String str,String dest,int[] next) {
    for(int i=0,j=0;i<str.length();i++) {
        while(j>0 && str.charAt(i)!=dest.charAt(j)) {
            j=next[j-1];
        }
        if(str.charAt(i)==dest.charAt(j)) {
            j++;
        }
        if(j==dest.length()) {
            return i-j+1;
        }
    }
    return -1;
}

public int[] kmpnext(String dest) {
    int[] next=new int[dest.length()];
    next[0]=0;
    for(int i=1,j=0;i<dest.length();i++) {
        if(j>0 && dest.charAt(i)!=dest.charAt(j)) {
            j=next[j-i];
        }
        if(dest.charAt(i)==dest.charAt(j)) {
            j++;
        }
        next[i]=j;
    }
    return next;
}
```

题目描述 79 最长公共子序列

```
public int LCS(int[] A,int n,int[] B,int m) {
    int[][] dp=new int[n+1][m+1];
    for(int i=1;i<=n;i++) {
        for(int j=1;j<=m;j++) {
            if(A[i-1]==B[j-1]) {
                dp[i][j]=dp[i-1]dp[j-1]+1;
            }
            else {
                dp[i][j]=Math.max(dp[i][j-1],dp[i-1][j]);
            }
        }
    }
    return dp[n][m];
}
```

题目描述 80 最长递增子序列

```
// 最长递增子序列（不连续）
public int LIS(int[] A) {
    int res=0;
    int[] dp=new int[A.length];
    dp[0]=1;
    for(int i=1;i<A.length;i++) {
        dp[i]=1;
        for(int j=0;j<i;j++) {
            if(A[i]>A[j]) {
                dp[i]=Math.max(dp[i],dp[j]+1);
            }
        }
        res=Math.max(res,dp[i]);
    }
    return res;
}

// 最长递增子序列（连续）
public int LIS(int[] A) {
    int res=0;
    int[] dp=new int[A.length];
    dp[0]=1;
    for(int i=1;i<A.length;i++) {
        dp[i]=1;
        if(A[i]>A[i-1]) {
            dp[i]=dp[i-1]+1;
        }
        res=Math.max(res,dp[i]);
    }
    return res;
}
```

题目描述 81 最大子序列乘积

```
// 最大子序列乘积（不连续）
public int maxProduct(int[] A) {
    int max=A[0];
    int min=A[0];
    int res=max;
    for(int i=1;i<A.length;i++) {
        int cur1=max*A[i];
        int cur2=min*A[i];
        max=Math.max(max,cur1);
        min=Math.min(min,cur2);
        res=Math.max(max,res);
    }
    return res;
}

// 最大子序列乘积（连续） {
public int maxProduct(int[] A) {
    int max=A[0];
    int min=A[0];
    int res=max;
    for(int i=1;i<A.length;i++) {
        int cur1=max*A[i];
        int cur2=min*A[i];
        max=Math.max(Math.max(cur1,cur2),A[i]);
        min=Math.min(Math.min(cur1,cur2),A[i]);
        res=Math.max(max,res);
    }
    return res;
}
```

题目描述 82 0-1 背包问题

```
public int oneZeroBag(int[] v,int[] s,int c) {
    int[][] dp=new int[v.length+1][c+1];
    for(int i=0;i<=c.length;i++) {
        dp[0][i]=0;
    }
    for(int i=1;i<=v.length;i++) {
        for(int j=0;j<=c;j++) {
            if(s[i-1]<=j) {
                dp[i][j]=Math.max(dp[i-1][j],dp[i-1][j-s[i-1]]+v[i-
1]);
            }
            else {
                dp[i][j]=dp[i-1][j];
        }
    }
    return dp[v.length][c];

    int cbag=bag;
    ArrayList<Integer> res=new ArrayList<Integer>();
    for(int i=v.length;i>0;i--) {
        if(dp[i][cbag]>dp[i-1][cbag]) {
            res.add(i);
            cbag-=s[i-1];
        }
    }
    return res;
}
```

题目描述 83 无向图最短路径

```
public int minGraph(int[][] A) {
    int[] dp=new int[A.length];
    Arrays.fill(dp,Integer.MAX_VALUE);
    int[] pre=new int[A.length];
    dp[0]=0;
    for(int i=0;i<A.length;i++) {
        for(j=0;j<A.length;j++) {
            if(A[i][j]!=0) {
                int d=dp[i]+A[i][j];
                if(d<dp[j]) {
                    dp[j]=d;
                    pre[j]=i+1;
                }
            }
        }
    }
    for(int i=A.length-1;i>0;) {
        System.out.println(pre[i]);
        i=pre[i]-1;
    }
    return dp[A.length-1];
}
```

题目描述 84 工作最大收益

```
public int maxProfit(int[] v,int[] pre) {
    int[] dp=new int[v.length+1];
    dp[0]=0;
    dp[1]=v[0];
    for(int i=1;i<=v.length;i++) {
        int choice=dp[pre[i-1]]+v[i-1];
        int notChoice=dp[i-1];
        dp[i]=Math.max(choice,notChoice);
    }
    return dp[V.length];
}
```

题目描述 85 切割最大收益

```
public int maxPro(int[] L,int[] v,int n) {
    int[] dp=new int[n+1];
    for(int i=1;i<=n;i++) {
        max=-1;
        for(int j=1;j<=i;j++) {
            max=Math.max(max,dp[i-j]+v[j-1]);
        }
        dp[i]=max;
    }
    return dp[n];
}
```

题目描述 86 三角形最短路径

```
public int min(List<List<Integer>> triangle) {
    int[] dp=new int[triangle.size()];
    for(int i=triangle.size()-1;i<=0;i--) {
        for(int j=0;j<triangle.get(i).size();j++) {
            if(i==triangle.size()-1) {
                dp[j]=triangle.get(i).get(j);
            }
            else {

    dp[j]=Math.min(triangle.get(i).get(j)+dp[j],triangle.get(i).ge
t(j)+dp[j+1]);
            }
        }
    }
    return dp[0];
}
```

题目描述 87 最小编辑距离

```
public int minEdit(String word1,String word2) {
    int n=word1.length();
    int m=word2.length();
    int[][] dp=new int[n+1][m+1];
    for(int i=0;i<=n;i++) {
        dp[i][0]=i;
    }
    for(int j=0;j<=m;j++) {
        dp[0][j]=j;
    }
    for(int i=1;i<=n;i++) {
        for(int j=1;j<=m;j++) {
            if(word1.charAt(i-1)==word2.charAt(j-1)) {
                dp[i][j]=dp[i-1][j-1];
            }
            else {
                dp[i][j]=Math.min(dp[i-1][j-1],Math.min(dp[i-
1][j],dp[i][j-1]))+1;
            }
        }
    }
    return dp[n][m];
}
```