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; }

\* }

\*/

import java.util.\*;

public class Solution {

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

if(pre.length==0 || in.length==0)

{

return null;

}

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

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

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

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

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

}

}

return node;

}

}

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) {

if(!stack2.isEmpty()){

stack1.push(stack2.pop());

}

stack1.push(node);

}

public int pop() {

while(!stack1.isEmpty()){

stack2.push(stack1.pop());

}

return stack2.pop();

}

}

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。

import java.util.ArrayList;

public class Solution {

public int minNumberInRotateArray(int [] array) {

int min=0;

for(int i=0;i<array.length-1;i++){

if(array[i]>array[i+1])

{

min=array[i+1];

}

}

return min;

}

}

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

public class Solution {

public int Fibonacci(int n) {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*递归\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if(n==0){

return 0;

}

if(n==1){

return 1;

}

if(n>1){

return Fibonacci(n-2)+Fibonacci(n-1);

}

return 0;\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*循环\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int qt=0;

int zt=1;

int result=0;

if(n==0){return 0;}

if(n==1){return 1;}

if(n>1){

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

result=zt+qt;

qt=zt;

zt=result;

}

}

return result;

}

}