# 面试题3：二维数组中的查找P38

public class Solution {

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

int m = 0;//行

int i = array.length-1;//列

while(m < array[0].length && i>=0){

if(array[m][i] > target)//与左上的元素相比较

i--;

else if(array[m][i] < target)

m++;

else

return true;

}

return false;

}

}

# 面试题4：替换空格P44

public class Solution {

public String replaceSpace(StringBuffer str) {

char a[]=new char[str.length()];

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

{

a[i]=str.charAt(i);

}

StringBuffer ss = new StringBuffer();

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

{

if(a[i]==' ')

{

ss.append("%20");

}

else

{

ss.append(a[i]);

}

}

String s = ss.toString();

// System.out.println(s);

return s;

}

}

# 面试题5：输入一个链表，从尾到头打印链表每个节点的值。P51

/\*\*

\* public class ListNode {

\* int val;

\* ListNode next = null;

\*

\* ListNode(int val) {

\* this.val = val;

\* }

\* }

\*

\*/

import java.util.ArrayList;

import java.util.Stack;

public class Solution {

public ArrayList<Integer> printListFromTailToHead(ListNode listNode) {

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

ArrayList<Integer> list=new ArrayList<Integer>();//新生成的从后到前的链

ListNode current=listNode;

while(current!=null){

stack.push(current);

current=current.next;

}

while(!stack.isEmpty()){

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

}

return list;

}

}

# 面试题6：重建二叉树

/\*\*

\* Definition for binary tree

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Solution {

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

//pre前序 in中序

TreeNode root=reConstruct(pre,0,pre.length-1,in,0,in.length-1);

return root;

}

//前序遍历{1,2,4,7,3,5,6,8}和中序遍历序列{4,7,2,1,5,3,8,6}

private TreeNode reConstruct(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]);

//最好把迭代写在循环外

for(int i=startIn;i<=endIn;i++)

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

root.left=reConstruct(pre,startPre+1,startPre+i-startIn,in,startIn,i-1);

root.right=reConstruct(pre,i-startIn+startPre+1,endPre,in,i+1,endIn);

}

return root;

}

}

# 面试题7：用两个栈实现队列P59

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(new Integer(node));

}

public int pop() {

while(!stack2.isEmpty())

{

return stack2.pop();

}

while(!stack1.isEmpty())

{

stack2.push(stack1.pop());

}

return stack2.pop();

}

}

# 面试题8：旋转数组的最小数字

import java.util.ArrayList;

public class Solution {

public int minNumberInRotateArray(int [] array) {

if (array.length == 0)

return 0;

int left=0;

int right=array.length-1;

int mid = left;

if(array[left]<array[right])

{

return array[left];

}

while(array[left]>=array[right])

{

if(right-left ==1)

{

mid=left;

break;

}

mid=(left+right)/2;

if(array[mid]>=array[left])

left=mid;

else if (array[mid]<=array[left])

right=mid;

}

return array[mid];

}

}

# 面试题9：斐波那契数列

public class Solution {

public int Fibonacci(int n)

{

//非递归解法

int [] array={0,1};

if (n<=0) return 0;

if (n<2) return array[n];

int left = 0;

int right = 1;

int fib = 0;

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

{

fib = left+right;

left=right;

right = fib;

}

return fib;

}

}

Log(n)的解法：

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.Scanner;

**import** java.util.Stack;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Scanner sc=**new** Scanner(System.***in***);

**long** n=sc.nextLong();

**if** (n==0) {

System.***out***.println("0");

}**else** **if** (n==1) {

System.***out***.println("1");

}**else** {

**long** [][] a={{1,1},{1,0}};

System.***out***.println(*fibonacci*(n,a)[0][0]);

}

}

**public** **static** **long**[][] fibonacci(**long** n,**long**[][] a){

**if** (n==2)

**return** a;

**if** (n==3)

**return** *matrixMultiply*(a, a);

**if** (n%2==0) {

**long** [][]b=*matrixMultiply*(*fibonacci*((n-2)/2+1,a), *fibonacci*((n-2)/2+1,a));

**return** *matrixMultiply*(b, a);

}**else** {

**return** *matrixMultiply*(*fibonacci*((n-1)/2+1,a), *fibonacci*((n-1)/2+1,a));

}

}

**public** **static** **long**[][] matrixMultiply(**long**[][] a, **long**[][] b) {

**long** c[][] = **new** **long**[a.length][b[0].length];

**int** x, i, j;

**for** (i = 0; i < a.length; i++) {

**for** (j = 0; j < b[0].length; j++) {

**long** temp = 0;

**for** (x = 0; x < b.length; x++) {

temp += a[i][x] \* b[x][j];

}

c[i][j] = temp;

}

}

**return** c;

}

}

# 面试题10：二进制中1的个数

**public** **class** Main {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

System.***out***.println(*numberOf1*(7));

}

**public** **static** **int** numberOf1(**int** n){

**int** count=0;

**while** (n!=0) {

**if** ((n & 1)!=0)

count++;

n=n>>1;

}

**return** count;

}

}

更好的方法

**public** **class** Main {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

System.***out***.println(*numberOf1*(7));

}

**public** **static** **int** numberOf1(**int** n){

**int** count=0;

**while** (n!=0) {

++count;

n=(n-1)&n;

}

**return** count;

}

}

# 面试题11：数值的整数次方

此题应该加个全局变量标志是否出现底数为零指数小于零的*错误情况*

public class Solution {

public double Power(double base , int exp)

{

int absexp=0;

if(base==0.0&&exp<0)

return 0.0;//底数为零指数小于零的情况

if (exp<0) absexp=(-1)\*exp;

else absexp=exp;

double result = Pow(base,absexp);

if(exp<0) result = 1.0/result;

return result;

}

double Pow(double base,int absexp)

{

double re = 1.0;

for(int i=1;i<=absexp;i++)

{

re \*= base;

}

return re;

}

}

# 面试题12：打印1到最大的n位数

方案1：模拟整数的加法

**import** java.util.Arrays;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

*printToMaxOfNDigits*(2);

}

**public** **static** **void** printToMaxOfNDigits(**int** n)

{

**if** (n<=0)

**return**;

**char** [] number =**new** **char**[n];

Arrays.*fill*(number, '0');

**while** (!*Increment*(number)) {

*printNumber*(number);

System.***out***.println();

}

}

**public** **static** **boolean** Increment(**char**[] number){

**boolean** isOverFlow=**false**;

**int** nTakeOver=0;

**int** nSum;

**int** nLength=number.length;

**for** (**int** i = nLength-1; i>=0; i--) {

nSum=number[i]-'0'+nTakeOver;

**if** (i==nLength-1) {

nSum++;

}

**if** (nSum>=10) {

**if** (i==0) {

isOverFlow=**true**;

}**else** {

nSum-=10;

nTakeOver=1;

number[i]=(**char**) ('0'+nSum);

}

}**else** {

number[i]=(**char**) ('0'+nSum);

**break**;

}

}

**return** isOverFlow;

}

**public** **static** **void** printNumber(**char**[] number){

**boolean** isBeginning0=**true**;

**int** nLength=number.length;

**for** (**int** i = 0; i < nLength; i++) {

**if** (isBeginning0 && number[i]!='0') {

isBeginning0=**false**;

}

**if** (!isBeginning0) {

System.***out***.print(number[i]);

}

}

}

}

方案二：

全排列递归：

**public** **class** Test {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

*printToMaxOfNDigits2*(2);

}

**public** **static** **void** printToMaxOfNDigits2(**int** n){

**if** (n<=0)

**return**;

**char** [] number =**new** **char**[n];

*pring1ToMaxOfNDigitsRecursively*(number,n,0);

}

**public** **static** **void** pring1ToMaxOfNDigitsRecursively(**char**[] number,**int** length,**int** index)

{

**if** (index==length-1) {

**for** (**int** i = 0; i < 10; i++) {

number[index]=(**char**) (i+'0');

*printNumber*(number);

}

}**else** {

**for** (**int** i = 0; i < 10; i++) {

number[index]=(**char**) (i+'0');

*pring1ToMaxOfNDigitsRecursively*(number, length, index+1);

}

}

}

**public** **static** **void** printNumber(**char**[] number){

**boolean** isBeginning0=**true**;

**int** nLength=number.length;

**for** (**int** i = 0; i < nLength; i++) {

**if** (isBeginning0 && number[i]!='0') {

isBeginning0=**false**;

}

**if** (!isBeginning0) {

System.***out***.print(number[i]);

}

}

**if** (!isBeginning0){

System.***out***.println();

}

}

}

# 面试题14：调整数组顺序使奇数位于偶数前面

//运行时间超时

public void reOrderArray(int [] array)

{

int i = 0;

int temp = 0;

while(i<array.length)

{

if(array[i]%2==0)

{

temp = array[i];

array[i]=array[i+1];

array[array.length-1]=temp;

i--;

}

i++;

}

}

# 面试题15：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)//鲁棒性

{

return null;

}

ListNode pre=head;

ListNode last=head;

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

{

if(pre.next!=null)//鲁棒性

{

pre=pre.next;

}else

return null;

}

while(pre.next!=null)

{

pre = pre.next;

last = last.next;

}

return last;

}

}

# 面试题16：反转链表

/\*

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;

if (head.next == null)

return head;

ListNode pPre = null;

ListNode p = head;

ListNode pNext = head.next;

ListNode newHead = null;

while (p != null) {

pNext = p.next;//一定要记录下来后面的节点

if (pNext == null)

newHead = p;

p.next = pPre;//这里的方向已经转变

pPre = p;//向后走了一位

p = pNext;

}

return newHead;

}

}

# 面试题17：合并两个排序列表

/\*

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&&list2==null)

return null;

else if (list1==null) return list2;

else if (list2==null) return list1;

//前四行考虑鲁棒性

ListNode newHead = null;

ListNode p = list1;

ListNode q = list2;

ListNode temp = newHead;

while(p!=null&&q!=null)

{

if(p.val<=q.val)

{

if(newHead==null)

newHead=temp=p;

else

{

temp.next=p;

temp=temp.next;

}

p=p.next;

}

else

{

if(newHead==null)

newHead=temp=q;

else

{

temp.next=q;

temp=temp.next;

}

q=q.next;

}

}

if(p==null)

{

temp.next=q;

}

if(q==null)

{

temp.next=p;

}

return newHead;

}

}

# 面试题18：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(root2==null) return false;

if(root1==null && root2!=null) return false;

boolean flag = false;

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

flag = DoesTree1HaveTree2(root1,root2);

}

if(!flag)

flag = HasSubtree(root1.left, root2);

if(!flag)

flag = HasSubtree(root1.right, root2);

return flag;

}

boolean DoesTree1HaveTree2(TreeNode p1,TreeNode p2)

{

if(p2==null) return true;

if(p1==null) return false;

if(p1.val!=p2.val) return false;

return DoesTree1HaveTree2(p1.left,p2.left)&&DoesTree1HaveTree2(p1.right,p2.right);

}

}

# 面试题19：二叉树的镜像

/\*\*

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 ;

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

TreeNode temp = null;

temp = root.left;

root.left = root.right;

root.right = temp;

if(root.left!=null) Mirror(root.left);

if(root.right!=null) Mirror(root.right);

}

}

# 面试题21：包含min函数的栈

import java.util.Stack;

public class Solution

{

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

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

int min ;

public void push(int node)

{

stack.push(node);

if(minStack.isEmpty())

minStack.push(node);

else

{

min = minStack.peek();

min = node < min ? minStack.push(node) : minStack.push(min);

}

}

public void pop() {

if(!stack.isEmpty() && !minStack.isEmpty()){

stack.pop();

minStack.pop();

}

}

public int top() {

return stack.peek();//peek 查看栈顶对象而不移除

}

public int min() {

return minStack.peek();

}

}

# 面试题22：栈的压入弹出序列

/\*思路：先循环将pushA中的元素入栈，遍历的过程中检索popA可以pop的元素

\*\*如果循环结束后栈还不空，则说明该序列不是pop序列。

\*/

import java.util.ArrayList;

import java.util.Stack;

public class Solution

{

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

{

Stack stack = new Stack();

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

for( int i=0,j=0; i < pushA.length; i++ ){

stack.push( pushA[i] );

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

stack.pop();

j ++;

}

}

return stack.empty() == true;

}

}

# 面试题23：从上往下打印二叉树

import java.util.\*;

/\*\*

<树的层序遍历>

java中队列queue的使用：

add 增加一个元索 如果队列已满，则抛出一个IIIegaISlabEepeplian异常

remove 移除并返回队列头部的元素 如果队列为空，则抛出一个NoSuchElementException异常

element 返回队列头部的元素 如果队列为空，则抛出一个NoSuchElementException异常

offer 添加一个元素并返回true 如果队列已满，则返回false

poll 移除并返问队列头部的元素 如果队列为空，则返回null

peek 返回队列头部的元素 如果队列为空，则返回null

put 添加一个元素 如果队列满，则阻塞

take 移除并返回队列头部的元素 如果队列为空，则阻塞

\*/

public class Solution

{

public ArrayList<Integer> PrintFromTopToBottom(TreeNode root)

{

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

//list 存放输出序列

if(root==null)

{

return list;

}

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

queue.offer(root);

while (!queue.isEmpty())

{

TreeNode treeNode = queue.poll();

if (treeNode.left != null) {

queue.offer(treeNode.left);

}

if (treeNode.right != null) {

queue.offer(treeNode.right);

}

list.add(treeNode.val);

}

return list;

}

}

# 面试题24：二叉搜索树后序遍历序列

import java.util.\*;

public class Solution

{

public boolean VerifySquenceOfBST(int [] sequence)

{

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

int root = sequence[sequence.length-1];

int i =0;

for(i=0;i<sequence.length-1;i++)

{

if(sequence[i]>root)

break;

}

for(int j=i;j<sequence.length-1;j++)

{

if(sequence[j]<root)

return false;

}

boolean left=true;

boolean right=true;

if(i>0)

left=VerifySquenceOfBST(Arrays.copyOfRange(sequence, 0, i));

if(i<sequence.length-1)

right=VerifySquenceOfBST(Arrays.copyOfRange(sequence, i, sequence.length-1));

return (left&&right);

/\*\*

System.arraycopy(src, 2, dest, 5, 1);

//从src中的第2个位置到dest的第5个位置;总数为1个

dest=Arrays.copyOfRange(src, 2, 4);

//从src中的第2个位置到第4个位置;总数为2个

\*/

}

}

# 面试题25：二叉树中和为某一值的路径

import java.util.\*;

/\*\*

算法思路:

（1）采用深度优先算法，利用栈结构存储访问过的路径结点的值。

（2）当访问到空节点时，直接return 。

（3）当访问到叶子结点时，判断走过的路径的值之和是否为target，如果是，则保存此路径，否则不处理。

（4）先访问左子树，后访问右子树。

（5）在访问完某个结点的左右子树时，需要将此结点的值出栈。

\*/

public class Solution {

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

{

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

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

dfs(lis,root,stack,target);

return lis;

}

//深度优先

public void dfs (ArrayList<ArrayList<Integer>> list, TreeNode root, Stack<Integer> stack, int target)

{

if(root == null)

return ;

if(root.left == null && root.right == null)

//如果是叶子节点

{

if(target == root.val)

{

Iterator<Integer> iter = stack.iterator();

//栈的迭代器

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

while(iter.hasNext())

{

tmp.add(iter.next());

//如果满足条件就将这条路径保存下来

}

tmp.add(root.val);

list.add(tmp);

}

return ;

}

//不是叶子节点就入栈

stack.push(root.val);

//递归左右子树

dfs(list,root.left,stack,target-root.val);

dfs(list,root.right,stack,target-root.val);

//返回到该节点的父节点处

stack.pop();

}

}

# 面试题26：复杂链表的复制（分步法）

import java.util.\*;

/\*

public class RandomListNode { 分步解决法：

int label;

RandomListNode next = null;

RandomListNode random = null;

RandomListNode(int label) {

this.label = label;

}

}

1.根据原始链表的每个节点N创建对应的N’，我们将N’链在N的后面。

2.设置复制出来的random,N是N’的next指向的节点，那么S'也是S的next指向的结点

3.拆分这个链表：奇数位置是原有的链表，偶数位置是复制出来的链表。

\*/

public class Solution

{

public RandomListNode Clone(RandomListNode pHead)

{

CloneNodes(pHead);

ConnectRandom(pHead);

return (Reconnect(pHead));

}

void CloneNodes(RandomListNode pHead)

{

RandomListNode pNode = pHead;

while(pNode != null)

{

RandomListNode pCloned = new RandomListNode(0);

pCloned.next = pNode.next;

pCloned.label = pNode.label;

pCloned.random = null;

pNode.next = pCloned;

pNode = pCloned.next;

}

}

void ConnectRandom(RandomListNode pHead)

{

RandomListNode pNode = pHead;

while(pNode!=null)

{

RandomListNode pCloned = pNode.next;

if(pNode.random!=null)

{

pCloned.random = pNode.random.next;

//是原始random节点的下一个节点

}

pNode = pCloned.next;

}

}

RandomListNode Reconnect(RandomListNode pHead)

{

RandomListNode pNode = pHead;

RandomListNode pClonedHead = null;

RandomListNode pClonedNode = null;

if(pNode != null)//处理头节点

{

pClonedHead = pClonedNode = pNode.next;

pNode.next = pClonedNode.next;

pNode = pNode.next;

//pNode向后走一位

}

while(pNode != null)

{

pClonedNode.next = pNode.next;

pClonedNode = pClonedNode.next;

pNode.next = pClonedNode.next;

pNode = pNode.next;

}

return pClonedHead;

}

}

# 面试题27：二叉搜索树与双向链表

非递归：

/\*\*

public class TreeNode {

int val = 0;

TreeNode left = null;

TreeNode right = null;

public TreeNode(int val) {

this.val = val;

}

}

方法一：非递归版

解题思路：

1.核心是中序遍历的非递归算法。

2.修改当前遍历节点与前一遍历节点的指针指向。

\*/

import java.util.Stack;

public class Solution {

public TreeNode Convert(TreeNode root)

{

if(root==null)

return null;

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

TreeNode p = root;

TreeNode pre = null;// 用于保存中序遍历序列的上一节点

boolean isFirst = true;

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

while(p!=null){

stack.push(p);

p = p.left;

}

p = stack.pop();

if(isFirst){

root = p;// 将中序遍历序列中的第一个节点记为root

pre = root;

isFirst = false;

}else{

pre.right = p;

p.left = pre;

pre = p;

}

p = p.right;

}

return root;

}

}

递归：

# 面试题28：字符串的排列（全排列，按照字典序输出）

/\*\*

该问题可以看成两步：

1.求所有可能出现在第一位的字符

2.固定第一个字符，求后面所有可能的排列

<递归思想>

\*/

import java.util.\*;

public class Solution

{

public ArrayList<String> Permutation(String str)

{

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

if (str == null || str.length() > 9 || str.length()==0)

{

return result;

}

str = str.trim();//去掉空格 就是字符串前后的空白

Permut(str.toCharArray(), 0, result);

HashSet<String> hs = new HashSet<String>(result);

//用hashset去重，但并未按照字典序排列

ArrayList<String> resultdic = new ArrayList<String>(hs);

Collections.sort(resultdic); //去重后的序列按照字典序排列

return resultdic;

}

void Permut(char[] data, int beginIdx,ArrayList<String> result)

{

if (beginIdx == data.length)//只剩一个字符的时候

{

result.add(new String(data));

}

else

{

for (int i = beginIdx; i < data.length;i++)

{

if(i!=beginIdx && data[i]==data[beginIdx]) continue;

char temp = data[beginIdx];

data[beginIdx] = data[i];

data[i] = temp;

Permut(data, beginIdx + 1, result);

//为了防止重复的情况，还需要将begin处的元素重新换回来

//恢复打扫战场，恢复为原来子串， data共享

temp = data[beginIdx];

data[beginIdx] = data[i];

data[i] = temp;

}

}

}

}

# 面试题29：数组中出现次数超过一半的数字

public class Solution {

public int MoreThanHalfNum\_Solution(int [] array)

{

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

if (array.length ==1) return array[0];

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

int middle = array[array.length/2-1];

boolean MoreThanHalf = CheckMoreThanHalf(array,middle);

if( MoreThanHalf==true) return array[middle];

else return 0;

}

private static void quickSort(int data[], int low, int hight)

{

int i=low,j=hight;

int mid=data[(low+hight)/2];//中间的数

while(i <= j)

{

while(data[i]<mid) i++;

while(data[j]>mid) j--;

if(i<=j)

{

int temp = data[i];

data[i] = data[j];

data[j] = temp;

i++;

j--;

}

}

if(i<hight) quickSort(data, i, hight);

if(j>low) quickSort(data, low, j);

}

boolean CheckMoreThanHalf(int[]array,int number)

{

int times = 0;

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

{

if(array[i]==number) times++;

}

boolean isMoreThanHalf = true;

if(times\*2<=array.length)

{

isMoreThanHalf = false;

}

return isMoreThanHalf;

}

}

# 面试题30：最小的K个数

import java.util.\*;

public class Solution {

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

{

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

if(k==0) return arr;

if(input.length<k) return arr;

// if(input.length==k) return arr;

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

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

{

arr.add(input[i]);

}

return arr;

}

private static void quickSort(int data[], int low, int hight)

{

int i=low,j=hight;

int mid=data[(low+hight)/2];//中间的数

while(i <= j)

{

while(data[i]<mid) i++;

while(data[j]>mid) j--;

if(i<=j)

{

int temp = data[i];

data[i] = data[j];

data[j] = temp;

i++;

j--;

}

}

if(i<hight) quickSort(data, i, hight);

if(j>low) quickSort(data, low, j);

}

}

# 面试题31：连续子数组的最大和

public class Solution {

public int FindGreatestSumOfSubArray(int[] array)

{

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

int sum = array[0];

int savedsum = array[0];

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

{

sum+=array[i];

if(sum<array[i])

{

sum=array[i];

// i--;

}

if(sum>=savedsum)

savedsum = sum;

}

return savedsum;

}

}

# 面试题32：从1到n的数字中1出现的次数

这谁写的算法，我也是醉了，把从1到n所有的数字拼接在一起。。。字符串好好长，然后分割存到数组，，，当n足够大时。。。。堆栈够用吗？！还有这个时间复杂度是什么？!!!!!!!!!

巧妙算法：拼接 分割 计算

import java.util.\*;

public class Solution

{

public int NumberOf1Between1AndN\_Solution(int n)

{

if(n==0) return 0;

if(n==1) return 1;

StringBuffer temp = new StringBuffer();

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

{

temp.append(i);

}

String temp2 =temp.toString().replaceAll("1", "AAA1");

String [] ts =temp2.split("1") ;

int num = ts.length-1;

return num;

}

}

# 面试题34：丑数

import java.util.\*;

public class Solution

{

public int GetUglyNumber\_Solution(int index)

{

if(index<=0) return 0;

int [] arr = new int [index];

arr[0] = 1;

if(index>1) arr[1] = 2;

if(index>2) arr[2] = 3;

if(index>3) arr[3] = 4;

if(index>4) arr[4] = 5;

if(index>5)

{

for(int i = 5 ;i < index; i ++)

{

int min2 = 0, min3 = 0, min5 = 0;

int tmp = 0;

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

tmp = arr[j]\*2;

if(tmp > arr[i-1]) {

min2 = tmp;

break;

}

}

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

tmp = arr[j]\*3;

if(tmp > arr[i-1]) {

min3 = tmp;

break;

}

}

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

tmp = arr[j]\*5;

if(tmp > arr[i-1]) {

min5 = tmp;

break;

}

}

int min =Math.min(min2,min3);

min = Math.min(min,min5);

arr[i] = min;

}

}

return arr[index-1];

}

}

# 面试题35：第一个只出现一次的字符

import java.util.\*;

public class Solution

{

//空间换时间 哈希表

public int FirstNotRepeatingChar(String str)

{

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

//char占一个字节，8位，最多表示256种字符

int []hash=new int [256];

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

{

hash[str.charAt(i)]++;

}

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

{

if(hash[str.charAt(j)]==1)

return j;

//找到的第一个出现次数为1的字符

}

return -1;

}

}

# 面试题37：两个链表第一个公共节点

/\*

public class ListNode {

int val;

ListNode next = null;

ListNode(int val) {

this.val = val;

}

}\*/

import java.util.\*;

public class Solution {

public ListNode FindFirstCommonNode(ListNode pHead1, ListNode pHead2)

{

int len1 = getListLength(pHead1);

int len2 = getListLength(pHead2);

int len;

ListNode plong = pHead1;

ListNode pshort = pHead2;

if(len1>=len2)

{

plong = pHead1;

pshort = pHead2;

len = len1-len2;

}

else

{

plong = pHead2;

pshort = pHead1;

len = len2-len1;

}

for(int i =0;i<len;i++)

{

plong = plong.next;

}

while((plong!=null)&&(pshort!=null)&&(plong!=pshort))

{

plong = plong.next;

pshort = pshort.next;

}

ListNode pFirstmeet = plong;

return pFirstmeet;

}

public static int getListLength(ListNode head)

{

int len=0;

while(head!=null)

{

len++;

head=head.next;

}

return len;

}

}

# 面试题38：数字在排序数组中出现的次数

import java.util.\*;

public class Solution {

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

{

int count = 0;

int len = array.length;

if(array!=null)

{

int first = getFirstK(array,k,0,len-1);

int end = getLastK(array,k,0,len-1);

if(first>-1&&end>-1)

count =end-first+1;

}

return count;

}

public static int getFirstK(int[] array,int k,int start,int end)

{

if(start>end)

return -1;

int mid=(start+end)/2;

int midData=array[mid];

if(midData==k){

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

return mid;

}

else

end=mid-1;

}

else if(midData>k)

end=mid-1;

else start=mid+1;

return getFirstK(array, k, start, end);

}

public static int getLastK(int[] array,int k,int start,int end)

{

if(start>end)

return -1;

int mid=(start+end)/2;

int midData=array[mid];

if(midData==k){

if(mid<array.length-1&&array[mid+1]!=k||mid==array.length-1){

return mid;

}

else

start=mid+1;

}

else if(midData<k)

start=mid+1;

else end=mid-1;

return getLastK(array, k, start, end);

}

}

# 面试题39：二叉树的深度

public class Solution {

public int TreeDepth(TreeNode root)

{

if (root==null) return 0;

int nleft = TreeDepth(root.left);

int nright = TreeDepth(root.right);

if(nleft>nright) return nleft+1;

else return nright+1;

}

}

# 面试题40：平衡二叉树

//笨方法

import java.util.\*;

public class Solution {

//后序遍历二叉树并且一边遍历一遍判断每个结点是不是平衡的

public boolean IsBalanced\_Solution(TreeNode root)

{

if (root == null)

return true;

if (Math.abs(Thight(root.left)-Thight(root.right)) > 1)

return false;

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

}

int Thight(TreeNode root)

{

if (root == null)

return 0;

int hight = Math.max(Thight(root.left),Thight(root.right)) + 1;

return hight;

}

}

# 面试题41：数组中只出现一次的数字

//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

import java.util.\*;

public class Solution {

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

{

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

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

{

//如果arraylist中没有这个值 就添加他，如果已经存在了，就删去之前有的这个值

//最后剩下的两个值 就是只有一个的数

if(!list.contains(array[i]))

list.add(array[i]);

else

list.remove(new Integer(array[i]));

}

if(list.size()>1)

{

num1[0]=list.get(0);

num2[0]=list.get(1);

}

}

}

# 面试题41（1）： 和为S的两个数字

import java.util.ArrayList;

public class Solution

{

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

{

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

if(array == null ||array.length<2) return list;

int before = 0;

int after = array.length-1;

while(before<after)

{

int s = array[before]+array[after];

if(s==sum)

{

list.add(array[before]);

list.add(array[after]);

return list;

}

else if(s<sum)

before++;

else

after--;

}

return list;

}

}

# 面试题41（2）： 和为S的连续整数序列

import java.util.ArrayList;

public class Solution {

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

{

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

if(sum<3) return Slist;

int small = 1;

int big = 2;

while(small<big)

{

int s = (small+big)\*(big-small+1)/2;

if(s==sum)

{

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

for(int i =small;i<=big;i++)

nlist.add(i);

Slist.add(nlist);

small++;

continue;

}

else if(s>sum)

small++;

else

big++;

}

return Slist;

}

}

# 面试题42（1）： 翻转单词顺序

import java.util.\*;

import java.lang.\*;

public class Solution {

public String ReverseSentence(String str)

{

StringBuffer result = new StringBuffer ("");

String sss = str;

if(sss.trim().equals(""))

return sss;

//.trim()删除所有开始和结尾的空字符

String[] array = sss.split(" ");

int i =0;

int j = array.length-1;

while(i<j)

{

String temp = array[j];

array[j] = array[i];

array[i] = temp;

i++;

j--;

}

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

{

result.append(array[k]);

result.append(" ");

}

result.setLength(result.length() - 1);

String fina = result.toString();

return fina;

}

}

# 面试题42（2）：左旋转字符串

import java.util.\*;

import java.lang.\*;

public class Solution {

public String LeftRotateString(String str,int n)

{

if(str.length()==0||n==0) return str;

int len = str.length();

int k = n;

while(k>len)

{

k = k%len;

}

char[] ch=str.toCharArray();

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

{

char temp = ch[0];

for(int j = 0;j<len-1;j++)

{

ch[j]=ch[j+1];

}

ch[len-1] = temp;

}

String s = String.valueOf(ch);

return s;

}

}

# 面试题44：扑克牌顺子

# 面试题49：把字符串转换成整数

import java.util.\*;

import java.lang.\*;

import java.io.\*;

public class Solution

{

public int StrToInt(String str)

{

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

char[] ch=str.toCharArray();

int len = ch.length;

int sum = 0;

int temp = 1;

if(ch[0]==43)

ch[0]='0';

if(ch[0]==45)

{

ch[0]='0';

temp = -1;

}

for(int i =0;i<len;i++)

{

if(ch[i]<48||ch[i]>57)

{

sum = 0;

break;

}

int c = Integer.parseInt(String.valueOf(ch[i]));

sum \*= 10;

sum = sum + c;

}

sum = sum\*temp;

return sum;

}

}

# 英文版新增面试题51：数组中重复的数字

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

{

int [] compare = new int[length];

boolean dupl = false;

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

{

if(compare[numbers[i]]==1)

{

duplication[0] = numbers[i];

dupl = true;

break;

}

else

compare[numbers[i]] = 1;

}

return dupl;

}

}

# 英文版新增面试题52：构建乘积数组

import java.util.ArrayList;

public class Solution {

public int[] multiply(int[] A)

{

int n = A.length;

int [] c=new int [n];

c[0]=A[0];

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

c[i]=c[i-1]\*A[i];

int [] d=new int [n];

d[n-1]=A[n-1];

for(int j=n-2;j>=0;j--)

d[j]=d[j+1]\*A[j];

int B []=new int [n];

B[0]=d[1];

B[n-1]=c[n-2];

for(int k=1;k<A.length-1;k++)

{

B[k]=c[k-1]\*d[k+1];

}

return B;

}

}

# 英文版新增面试题54：表示数值的字符串

英文版新增面试题51：