1.数组中重复的数字

题目描述

在一个长度为n的数组里的所有数字都在0~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重复了，也不知道每个数字重复了几次。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}，那么对应的输出是重复的数字2或者3.

要求：时间复杂度：O(n)。 空间复杂度：O(1)。

方法

- 1.遍历数组a。
- 2.a[i]与i比较：如果a[i]等于i，那么继续扫描a[i+1]；如果a[i]不等于i，那么将a[i]与a[a[i]]比较。
- 3.a[i]与a[a[i]]比较：如果a[i]等于a[a[i]]，那么输出第一个重复数字a[i]，函数结束；如果a[i]不等于a[a[i]]，那么将a[i]和a[a[i]]交换，交换后a[i]的值与位置相等。
- 4.交换后重复步骤2继续将a[i]和i进行比较。

代码

```
public class Solution {
    public boolean duplicate(int numbers[],int length,int [] duplication) {
        if(numbers==null||length<=0)
            return false;
        for(int i=0;i<length;i++){
            while(numbers[i]!=i){
                if(numbers[i]==numbers[numbers[i]]){
                    duplication[0]=numbers[i];
                    return true;
                }
                int temp=numbers[i];
                numbers[i]=numbers[temp];
                numbers[temp]=temp;
            }
        }
        return false;
    }
}
```

2.二维数组的查找

题目要求

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。

效率要求：时间复杂度：O(M+N)，其中M为数组行数，N为列数。 空间复杂度：O(1)。

方法

1.选取数组的右上角数字与目标数字进行比较。

2.如果右上角数字等于目标数字，则查找过程结束。

3.如果右上角数字大于目标数字，则剔除该列。

4.如果右上角数字小于目标数字，则剔除该行。

代码

```
public class Solution {
    public boolean Find(int target, int [][] array) {
        if(array==null||array.length==0||array[0].length==0)
            retrun false;
        int rows=array.length;
        int cols=array[0].length;
        int r=0;
        int c=cols-1;
        while(r<=rows-1&&c>=0){
            if(target==array[r][c]){
                return true;
            }
            else if(target>array[r][c]){
                r++;
            }
            else{
                c--;
            }
        }
        return false;
    }
}
```

3.替换空格

题目描述

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

效率要求

时间复杂度O(n)：假设字符串长度为n，每个字符移动1次，一共移动n次。
空间复杂度O(1)：两个指针p1和p2。

方法

- 1.在字符串尾部填充任意字符，使得字符串长度等于替换之后的长度。
- 2.因为一个空格要替换成三个字符(%20)，所以遍历到一个空格时，需要在尾部填充两个任意字符。
- 3.令p1指向字符串原来的末尾位置，p2指向字符串现在的末尾位置。
- 4.p1和p2从后向前遍历。
- 5.如果p1遍历到一个空格时，需要令p2指向的位置依次填充02%。
- 6.如果p1遍历到一个非空格字符时，需要令p2指向的位置填充上p1指向的字符值。

代码

```
public class Solution {
    public String replaceSpace(StringBuffer str) {
        int p1=str.length()-1;
        for(int i=0;i<=p1;i++){
            if(str.charAt(i)==' '){
                str.append("  ");
            }
```

```
        }
        int p2=str.length()-1;
        while(p1>=0&&p2>p1){
            char c=str.charAt(p1);
            p1--;
            if(c==' '){
                str.setCharAt(p2,'0');
                p2--;
                str.setCharAt(p2,'2');
                p2--;
                str.setCharAt(p2,'%');
                p2--;
            }else{
                str.setCharAt(p2,c);
                p2--;
            }
        }
        return str.toString();
    }
}
```

4.从尾到头打印链表

题目描述

输入一个链表，按链表从尾到头的顺序返回一个ArrayList。

方法1 递归

```
1.输入链表[a,b,c]，返回数组[c,b,a]。
2.递归的终止条件就是传入的指针不是空指针。
3.因为a!=null，所以ret.addAll(f(a.next))。
4.因为b==a.next!=null，所以ret.addAll(f(b.next))。
5.因为c==b.next!=null，所以ret.addAll(f(c.next))。
6.因为c.next==null，递归终止，开始返回，return ret。
7.返回5中f的调用，ret.addAll(ret),ret.add(c.val),return ret==[c.val]。
8.返回4中f的调用，ret.addAll([c.val]),ret.add(b.val),return ret= [c.val,b.val]。
9.返回3中f的调用，ret.addAll([c.val,b.val]),ret.add(a.val),return ret=
[c.val,b.val,a.val]。
```

方法1 效率

时间复杂度O(n)：假设链表长度为n,f的时间复杂度为O(1),T(n)=T(n-1)+O(1),T(n)=n*O(1)。
空间复杂度O(n)：假设链表长度为n,需要调用n次递归函数，每次递归函数需要临时占用空间1。

方法1 代码

```
/**
*    public class ListNode {//结点
*        int val;//数据
*        ListNode next = null;//指针：保存下一个结点的地址
*
*        ListNode(int val) {
*            this.val = val;
*        }
*    }
*
```

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

方法2 头插法

- 1.输入链表L[a,b,c]，头插法新建带空头结点链表head[-1,c,b,a]：
  - ①.新建链表的空头结点head，头插法建立新链表的循环终止条件为旧链表头结点(头指针)为空。
  - ②.保存旧链表头结点的后继结点。
  - ③.旧链表头结点的指针指向新链表头结点的后继结点。
  - ④.新链表头结点的指针指向旧链表头结点。
  - ⑤.旧链表头结点指向旧链表头结点的指针。
- 2.新建数组ret，将链表元素按顺序添加至ret，然后返回ret。

方法2 效率

时间复杂度O(n):假设链表长度为n，程序循环次数为2n。

空间复杂度O(1):假设链表长度为n，程序临时占用空间为2。

方法2 代码

```
import java.util.ArrayList;
public class Solution {
    public ArrayList<Integer> printListFromTailToHead(ListNode listNode) {
        ListNode head=new ListNode(-1);
        while(listNode!=null){
            ListNode mem=listNode.next;
            listNode.next=head.next;
            head.next=listNode;
            listNode=mem;
        }
        ArrayList<Integer> ret=new ArrayList<>();
        head=head.next;
        while(head!=null){
            ret.add(head.val);
            head=head.next;
        }
        return ret;
    }
}
```

方法3 栈

- 1.遍历链表将数据按顺序放入栈中。
- 2.按栈的出栈顺序添加至ret中。

方法3 效率

时间复杂度O(n)：程序循环次数为2n。

空间复杂度O(n):程序临时占用空间为n。

方法3 代码

```
import java.util.ArrayList;
public class Solution {
    public ArrayList<Integer> printListFromTailToHead(ListNode listNode) {
        Stack<Integer> stack=new Stack<>();
        while(listNode!=null){
            stack.add(listNode.val);
            listNode=listNode.next;
        }
        ArrayList<Integer> ret=new ArrayList<>();
        while(!stack.isEmpty()){
            ret.add(stack.pop());
        }
        return ret;
    }
}
```

5.重建二叉树

题目描述

输入某二叉树的前序遍历和中序遍历的结果，请重建出该二叉树。假设输入的前序遍历和中序遍历的结果中都不含重复的数字。例如输入前序遍历序列{1,2,4,7,3,5,6,8}和中序遍历序列{4,7,2,1,5,3,8,6}，则重建二叉树并返回。

解题思路

1.前序遍历的第一个结点为根结点，该值将中序遍历结果分为两部分，左部分是树的左子树中序遍历，右部分是树的右子树中序遍历。

2.前序遍历的第二个结点为左子树根结点，该值将左子树中序遍历结果分为两部分，左部分式左子树的左子树中序遍历，右部分是左子树的右子树中序遍历。

3.中序遍历根结点对应为i，然后i的左边是左子树(L2,i-1)，左子树长度为**i-1-L2**，右边是右子树(i+1,l2)，右子树长度为l2-i-1；前序遍历的根结点对应L1，左子树是(L1+1,L1+1+(i-1-L2)),右子树是(L1+1+(i-1-L2)+1,R2)。

效率

时间复杂度：O(n)。

空间复杂度：O(n)。

代码

```
/**
 * 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) {
        return dfs(pre,0,pre.length-1,in,0,in.length-1);
    }
    private TreeNode dfs(int[] a1,int l1,int r1,int[] a2,int l2,int r2){
        if(l1>r1||l2>r2){
            return null;
        }
        TreeNode root=new TreeNode(a1[l1]);
        for(int i=l2;i<=r2;i++){
            if(a2[i]==a1[l1]){
                root.left=dfs(a1,l1+1,l1+i-l2,a2,l2,i-1);
                root.right=dfs(a1,l1+i-l2+1,r1,a2,i+1,r2);
                break;
            }
        }
        return root;
    }
}
```

6.二叉树的下一个结点

题目描述

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

方法

1.如果二叉树为空，则返回空。

2.如果当前节点的右孩子不为空，那么下一个节点就是当前节点右孩子的最左孩子。

3.如果当前节点的右孩子为空，那么分两种情况：

第一种情况：如果该节点是父节点的左孩子，那么下一个节点就是父节点。

第二种情况：如果该节点是父节点的右孩子，那么不断向上遍历父节点，直到找到节点a是其父节点的左孩子，那么下一个节点就是其父节点a.next；如果没找到的话，那么该节点就是尾节点。

代码

```
/*
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){
            TreeLinkNode parent=pNode.next;
            if(parent.left==pNode){
                return parent;
            }
            pNode=pNode.next;
        }
        return null;
    }
}
```

7.用两个栈实现队列

题目描述

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

方法

1.入队：将元素放进栈A。

2.出队：分两种情况

第一种情况，如果栈B为空，那么将栈A的所有元素pop出栈，然后push所有元素进栈B，然后栈B再pop一个元素出栈。

第二种情况，如果栈B非空，那么栈B直接pop一个元素出栈。

代码

```
import java.util.Stack;

public class Solution {
    Stack<Integer> stack1 = new Stack<Integer>();
    Stack<Integer> stack2 = new Stack<Integer>();

    public void push(int node) {
        stack1.push(node);
    }

    public int pop() {
        if(stack2.isEmpty()){
            while(!stack1.isEmpty()){
                stack2.push(stack1.pop());
            }
        }
        return stack2.pop();
    }
}
```

8.斐波那契数列

题目描述

大家都知道斐波那契数列，现在要求输入一个整数n，请你输出斐波那契数列的第n项（从0开始，第0项为0）。(n<=39)。

方法

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

要求时间复杂度O(n)，空间复杂度O(1)

```
public class Solution {
    public int Fibonacci(int n) {
        if(n<=1){
            return n;
        }
        int f0=0;
        int f1=1;
        int f=0;
        for(int i=2;i<=n;i++){
            f=f0+f1;
            f0=f1;
            f1=f;
        }
        return f;
    }
}
```

要求时间复杂度O(n)，空间复杂度O(n)。

```
public class Solution {
    public int Fibonacci(int n){
        if(n<=1){
            return n;
        }
        int[] f=new int[n+1];
        f[0]=0;
        f[1]=1;
        for(int i=2;i<=n;i++){
            f[i]=f[i-1]+f[i-2];
        }
        return f[n];
    }
}
```

9.跳台阶

题目描述

一只青蛙一次可以跳上1级台阶，也可以跳上2级。求该青蛙跳上一个n级的台阶总共有多少种跳法（先后次序不同算不同的结果）。

方法

1.如果n=1,有一种跳法f(n)=1.

2.如果n=2,有两种跳法(1,1)和(2),所以f(n)=2.

3.如果n>=3,可以先跳2阶，再跳n-2阶；或者先挑1阶，再跳n-1阶；所以f(n)=f(n-1)+f(n-2)。

要求时间复杂度O(n)，空间复杂度O(1)

```
public class Solution {
    public int JumpFloor(int target) {
        if(target<=2){
            return target;
        }
        int f=0;
        int f1=1;
        int f2=2;
        for(int i=3;i<=target;i++){
            f=f1+f2;
            f1=f2;
            f2=f;
        }
        return f;
    }
}
```

要求时间复杂度O(n)，空间复杂度O(n)

```
public class Solution {
    public int JumpFloor(int target) {
        if(target<=2){
            return target;
        }
        int[] f=new int[target+1];
        f[1]=1;
        f[2]=2;
        for(int i=3;i<=target;i++){
            f[i]=f[i-1]+f[i-2];
        }
        return f[target];
    }
}
```

10.矩形覆盖

题目描述

我们可以用2*1的小矩形横着或者竖着去覆盖更大的矩形。请问用n个2*1的小矩形无重叠地覆盖一个2*n的大矩形，总共有多少种方法?

方法

1.如果n=1，那么只有一种方法,f(1)=1.

2.如果n=2,那么只有两种方法，横着两个和竖着两个，所以f(2)=2.

3.如果n>=3,可以先竖着放一个,还剩2*(n-1)的矩形；或者先横着放一个，那么必定还得下面横着放一个，还剩2*(n-2)的矩形；所以f(n)=f(n-1)+f(n-2)。

要求时间复杂度O(n)，空间复杂度O(1)

```
public class Solution {
    public int RectCover(int target) {
        if(target<=2){
            return target;
```

```
        }
        int f=0;
        int f1=1;
        int f2=2;
        for(int i=3;i<=target;i++){
            f=f1+f2;
            f1=f2;
            f2=f;
        }
        return f;
    }
}
```

要求时间复杂度O(n)，空间复杂度O(n)

```
public class Solution {
    public int RectCover(int target) {
        if(target<=2){
            return target;
        }
        int[] f=new int[target+1];
        f[1]=1;
        f[2]=2;
        for(int i=3;i<=target;i++){
            f[i]=f[i-1]+f[i-2];
        }
        return f[target];
    }
}
```

11.变态跳台阶

题目描述

一只青蛙一次可以跳上1级台阶，也可以跳上2级……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。

方法

i=1,f(1)=1.

i=2,f(2)=f(1)+1=2.

i=3,f(3)=f(2)+f(1)+1=4.

i=4,f(4)=f(3)+f(2)+f(1)+1=8.

i=n-1,f(n-1)=f(n-2)+f(n-3)+...+f(1)+1.

i=n,f(n)=f(n-1)+...+f(1)+1.

综上,f(n)-f(n-1)=f(n-1),所以f(n)=2*f(n-1).

要求时间复杂度O(n)，空间复杂度O(1)。

```
public class Solution {
    public int JumpFloorII(int target) {
        if(target<=1){
            return target;
        }
        int f=1;
        for(int i=2;i<=target;i++){
            f=2*f;
        }
        return f;
    }
}
```

要求时间复杂度O(n)，空间复杂度O(n)

```
public class Solution {
    public int JumpFloorII(int target) {
        if(target<=1){
            return target;
        }
        int[] f=new int[target+1];
        f[1]=1;
        for(int i=2;i<=target;i++){
            f[i]=2*f[i-1];
        }
        return f[target];
    }
}
```

12.旋转数组的最小数字

题目描述

把一个数组最开始的若干个元素搬到数组的末尾，我们称之为数组的旋转。
输入一个非递减排序的数组的一个旋转，输出旋转数组的最小元素。
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。

方法

1.设置三个指针low、high和mid。

2.如果a[mid]<a[high],表示[mid,high]区间内数组是非递减数组；[low,mid]区间内的数组是旋转数组，此时令high=mid.

3.如果a[mid]>a[high],表示[mid,high]区间内数组是旋转数组；[low,mid]区间内的数组是非递减数组，此时令low=mid+1.

4.如果a[mid]==a[high],特殊情况导致区间无法二分,区间缩小，high=high-1.

5.时间复杂度O(n)=O(logn)+O(n),空间复杂度O(1).

代码

```
import java.util.ArrayList;
public class Solution {
    public int minNumberInRotateArray(int [] array) {
        if(array.length==0){
```

```
            return 0;
        }
        int low=0;
        int high=array.length-1;
        while(low<high){
            int mid=low+(high-low)/2;
            if(array[mid]==array[high]){
                high=high-1;
            }
            else if(array[mid]<array[high]){
                high=mid;
            }
            else{
                low=mid+1;
            }
        }
        return array[low];
    }
}
```

13.矩阵中的路径

题目描述

请设计一个函数，用来判断在一个矩阵中是否存在一条包含某字符串所有字符的路径。路径可以从矩阵中的任意一个格子开始，每一步可以在矩阵中向左，向右，向上，向下移动一个格子。如果一条路径经过了矩阵中的某一个格子，则该路径不能再进入该格子。 例如 a b c e s f c s a d e e 矩阵中包含一条字符串"bcced"的路径，但是矩阵中不包含"abcb"路径，因为字符串的第一个字符b占据了矩阵中的第一行第二个格子之后，路径不能再次进入该格子。

方法

1.对于给定数组，初始化一个标志位数组，初始化为false，表示未走过，true表示已经走过，不能走第二次。

2.根据行数和列数，遍历数组，先找到一个与str字符串的第一个元素相匹配的矩阵元素，进入judge递归函数。

3.根据坐标确定一维数组的位置，因为给定的matrix是一个一维数组。

4.确定递归终止条件：越界；当前找到的值不等于数组对应位置的值；已经走过的；返回true.

5.如果k走到s最后一位，那么匹配成功。

6.递归寻找周围四个格子是否符合条件，如果有格子符合条件，那么就继续找符合条件的格子的四周是否存在符合条件的格子，直到k不满足递归条件或者到达末尾停止递归。

7.如果递归找不到符合条件的下一个格子，那么还原index的标志位。

代码

```
public class Solution {
    public boolean hasPath(char[] matrix, int rows, int cols, char[] str){
        boolean[][] flag=new boolean[rows][cols];
        for(int i=0;i<rows;i++){
            for(int j=0;j<cols;j++){
                if(dfs(matrix,str,flag,0,i,j,rows,cols)){
                    return true;
                }
```

```
            }
        }
        return false;
    }
    //初始矩阵，字符串，标志位，字符串索引，横坐标，纵坐标，行数，列数
    private boolean dfs(char[] m,char[] s,boolean[][] flag,int k,int i,int j,int
rows,int cols){
        if(i<0||i>=rows||j<0||j>=cols||m[i*cols+j]!=s[k]||flag[i][j]==true){
            return false;//当前节点为不可达叶子节点
        }
        if(k==s.length-1){
            return true;//当前节点为可行解
        }
        flag[i][j]=true;//当前节点标志为走过
        if(dfs(m,s,flag,k+1,i+1,j,rows,cols)||
           dfs(m,s,flag,k+1,i-1,j,rows,cols)||
           dfs(m,s,flag,k+1,i,j+1,rows,cols)||
           dfs(m,s,flag,k+1,i,j-1,rows,cols)){
            return true;
        }
        flag[i][j]=false;//当前节点没有可行解，清空节点状态
        return false;//无解
    }
}
```

14.机器人的运动范围

题目描述

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

方法

使用深度优先搜索(Depth First Search,DFS)方法求解，回溯是深度优先搜索的一种特列，它在一次搜索过程中需要设置一些本次搜索过程的局部状态。

代码

```
public class Solution {
    int cnt=0;
    public int movingCount(int threshold, int rows, int cols){
        boolean[][] flag=new boolean[rows][cols];
        dfs(flag,0,0,rows,cols,0,threshold);
        return cnt;
    }
    //标志位，横坐标，纵坐标，行数，列数，当前节点横纵和，限制大小
    private void dfs(boolean[][] flag,int i,int j,int rows,int cols,int
digitSum,int threshold){
        digitSum=Sum(i)+Sum(j);//计算当前节点横纵和
        if(i<0||i>=rows||j<0||j>=cols||flag[i][j]==true||digitSum>threshold){
            return;//当前节点为不可达节点
        }
        flag[i][j]=true;//当前节点标志为走过
        cnt++;//当前节点为可达节点，计数
        dfs(flag,i,j+1,rows,cols,digitSum,threshold);
```

```
        dfs(flag,i,j-1,rows,cols,digitSum,threshold);
        dfs(flag,i-1,j,rows,cols,digitSum,threshold);
        dfs(flag,i+1,j,rows,cols,digitSum,threshold);
    }
    private int Sum(int i){
        int ds=0;
        while(i>0){
            ds=ds+i%10;
            i=i/10;
        }
        return ds;
    }
}
```

15.剪绳子

题目描述

给你一根长度为n的绳子，请把绳子剪成m段（m、n都是整数，n>1并且m>1），每段绳子的长度记为k[0],k[1],...,k[m]。请问k[0]xk[1]x...xk[m]可能的最大乘积是多少？例如，当绳子的长度是8时，我们把它剪成长度分别为2、3、3的三段，此时得到的最大乘积是18。

贪心算法

时间复杂度O(logn),空间复杂度O(1).
贪心证明:当n>=5的时候，有2(n-1)>n且3(n-3)>n，且3(n-3)>=2(n-2)，所以尽可能的剪长度为3的绳子段。当n=2时，剪一下1*1；当n=3，剪一下1*2；当n=4；剪一下2*2。

```
public class Solution {
    public int cutRope(int target) {
        if(target==2){
            return 1;
        }
        if(target==3){
            return 2;
        }
        if(target%3==0){
            return pow(3,target/3);
        }else if(target%3==1){
            return 2*2*pow(3,(target-4)/3);
        }else{
            return 2*pow(3,(target-2)/3);
        }
    }
    private int pow(int a,int n){
        int s=1;
        for(int i=0;i<n;i++){
            s=s*a;
        }
        return s;
    }
}
```

动态规划

时间复杂度O(n^2)，空间复杂度O(n).

```
public class Solution {
    public int cutRope(int target) {
        //n<=3时必须分段
        if(target==2){
            return 1;
        }
        if(target==3){
            return 2;
        }
        //n>=4时，1，2，3长度为最小单元
        int[] dp=new int[target+1];
        dp[1]=1;
        dp[2]=2;
        dp[3]=3;
        int r=0;
        for(int i=4;i<=target;i++){
            for(int j=1;j<=i/2;j++){
                r=max(r,dp[j]*dp[i-j]);
            }
            dp[i]=r;
        }
        return dp[target];
    }
    private int max(int m,int n){
        if(m>n){
            return m;
        }else{
            return n;
        }
    }
}
```

16.二进制中1的个数

题目描述

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。

左移判断

flag=0001和n做&运算

左移,0010与n做&运算

左移,0100与n做&运算

左移,1000与n做&运算...

```
public class Solution {
    public int NumberOf1(int n) {
        int count=0;
        int flag=1;
        while(flag!=0){
            if((n&flag)!=0){
                count++;
            }
            flag=flag<<1;
        }
        return count;
    }
}
```

减一判断

把一个整数减去1，再和原整数做与运算，会把该整数最右边的1变成0，那么一个整数的二进制表示中有多少个1，就可以进行多少次这种操作。

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

17.数值的整数次方

题目描述

给定一个double类型的浮点数base和int类型的整数exponent。求base的exponent次方。
保证base和exponent不同时为0。

直接计算

时间复杂度O(n)，空间复杂度O(1)。

```
public class Solution {
    public double Power(double base, int exponent) {
        double s=1;
        if(exponent<0){
            if(base==0){
                throw new RuntimeException("分母不能0");
            }
            for(int i=0;i<-exponent;i++){
                s=s/base;
            }
        }else if(exponent==0){
            s=1;
        }else{
            for(int i=0;i<exponent;i++){
            s=s*base;
            }
```

```
        }
        return s;
    }
}
```

快速幂递归

时间复杂度O(logn)，空间复杂度O(1)

```
public class Solution {
    public double Power(double base, int exponent) {
        double r;
        int n=((exponent>=0)?exponent:-exponent);
        if(n==0){
            return 1;
        }
        if(n==1){
            return base;
        }
        if(n%2==0){//n>=3时分奇数和偶数求解
            r=Power(base*base,n/2);
        }else{
            r=base*Power(base*base,(n-1)/2);
        }
        r=((exponent>=0)?r:1/r);
        return r;
    }
}
```

18.打印从1到最大的n位数

题目描述

输入数字 n，按顺序打印出从 1 到最大的 n 位十进制数。比如输入 3，则打印出 1、2、3 一直到最大的 3 位数即 999。

全排列递归

时间复杂度O(n),空间复杂度O(n)。

```
class F{
    void print1ToMaxOfNDigits(int n){
        char[] num=new char[n];
        for(int i=0;i<10;i++){
            num[0]=(char)(i+'0');
            dfs(0,n,num);
        }
    }
    void dfs(int index,int len,char[] num){//索引，数字位数，数字
        if(index==len-1){//递归终止条件
            printn(num);
            return;
        }
        for(int i=0;i<10;i++){
            num[index+1]=(char)(i+'0');
            dfs(index+1,len,num);
        }
```

```
    }
    void printn(char[] num){
        int index=0;
        while(index<num.length&&num[index]=='0'){
            index++;
        }
        while(index<num.length){
            System.out.print(num[index]);
            index++;
        }
        System.out.println();
    }
}
class test{
    public static void main(String[] args){
        F f=new F();
        f.print1ToMaxOfNDigits(3);
    }
}
```

直接计算

时间复杂度O(n)，空间复杂度O(n)。

```
class F{
    void print1ToMaxOfNDigits(int n){
        char[] num=new char[n];
        for(int i=0;i<n;i++){
            num[i]='0';//数组初始化;
        }
        while(judge(num,n)==false){//如果未溢出
            printn(num);//输出
        }

    }
    boolean judge(char[] num,int len){
        boolean isOverfolw=false;//溢出标志
        int carry=0;//进位标志
        for(int i=len-1;i>=0;i--){//从最后一位开始遍历
            int sum=num[i]-'0'+carry;//取第i位的字符转换为数字+进位符
            if(i==len-1){//如果是末位数字，自加
                sum++;
            }
            if(sum>=10){//如果增加到10，进位
                if(i==0){
                    isOverfolw=true;//n+1位溢出
                }else{
                    sum=sum-10;//当前sum清零
                    carry=1;//进位标志置1
                    num[i]=(char)(sum+'0');//赋值
                }
            }else{
                num[i]=(char)(sum+'0');//赋值
                break;
            }
        }
        return isOverfolw;
```

```
    }
    void printn(char[] num){
        int index=0;
        while(index<num.length&&num[index]=='0'){
            index++;
        }
        while(index<num.length){
            System.out.print(num[index]);
            index++;
        }
        System.out.println();
    }
}
class test{
    public static void main(String[] args){
        F f=new F();
        f.print1ToMaxOfNDigits(3);
    }
}
```

19.删除链表中重复的结点

题目描述

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

方法

1.创建空头结点，当前结点指针pre和后继结点指针last。

2.last指针向后搜索第一个与pre不相等的结点。

3.pre指向那个与pre不相等的结点。

代码

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

    ListNode(int val) {
        this.val = val;
    }
}
*/
public class Solution {
    public ListNode deleteDuplication(ListNode pHead){
        if(pHead==null||pHead.next==null){//0或者1个节点，递归终止
            return pHead;//返回链表头节点
        }
        ListNode Head=new ListNode(-1);//创建空头结点
        Head.next=pHead;
        ListNode pre=Head;//前驱节点指针
        ListNode last=Head.next;//工作指针
        while(last!=null){//工作指针非空
            if(last.next!=null&&last.val==last.next.val){//如果当前节点非空且与后继节点相等
```

```
                while(last.next!=null&&last.val==last.next.val){//寻找第一个与当前结点不相等的结点
                    last=last.next;
                }
                pre.next=last.next;//前驱节点指针指向第一个不重复的节点
                last=last.next;
            }else{//如果不重复
                pre=pre.next;//前驱节点指针后移
                last=last.next;//工作指针后移
            }
        }
    return Head.next;//返回头节点
    }
}
```

20.正则表达式匹配

题目描述

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

代码

```
public class Solution {
    public boolean match(char[] str, char[] pattern){
        int i=0;
        int j=0;
        return dfs(str,i,pattern,j);
    }
    private boolean dfs(char[] s,int i,char[] p,int j){
        if(i==s.length&&j==p.length){//s和p到达尾部，匹配成功
            return true;
        }
        if(i!=s.length&&j==p.length){//p先到达尾部,匹配失败
            return false;
        }
        //1.模式的第二个字符为*
        if(j+1<p.length&&p[j+1]=='*'){
            if((i!=s.length&&p[j]==s[i])||(i!=s.length&&p[j]=='.')){//如果字符串与模式字符匹配
                return dfs(s,i,p,j+2)||dfs(s,i+1,p,j+2)||dfs(s,i+1,p,j);//三种匹配
                //①.模式后移2字符，相当于x*被忽略
                //②.字符串后移1字符，模式后移2字符
                //③.字符串后移1字符，模式不变，即继续匹配字符下一位，因为*可以匹配多位
            }else{//如果字符串与模式字符不匹配
              return dfs(s,i,p,j+2);//模式后移2位
            }
        }
        //2.模式的第二个字符不是*
        if((i!=s.length&&p[j]==s[i])||(i!=s.length&&p[j]=='.')){//如果字符串与模式字符匹配
            return dfs(s,i+1,p,j+1);//字符串和模式都后移1位
        }else{
            return false;
```

```
        }
    }
}
```

21.表示数值的字符串

题目描述

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

代码

```
public class Solution {
    public boolean isNumeric(char[] str) {
        //e、小数点、符号位的计数器
        //e不能在尾部
        //不能出现两个e
        //不能出现两个小数点,小数点不能放在e后面
        //如果不在首位出现+-符号，那么必须在e后面
        //不能出现非法字符
        if(str[str.length-1]=='e'){
            return false;
        }
        int f1=0,f2=0,f3=0;
        for(int i=0;i<str.length;i++){
            if(str[i]=='e'||str[i]=='E'){
                f1++;
                if(f1>1){
                    return false;
                }
            }else if(str[i]=='.'){
                f2++;
                if(f1>0||f2>1){
                    return false;
                }
            }else if(str[i]=='+'||str[i]=='-'){
                if(i>0){
                    if(str[i-1]!='e'&&str[i-1]!='E'){
                        return false;
                    }
                }
            }else if(str[i]>'9'||str[i]<'0'){
                return false;
            }
        }
        return true;
    }
}
```

22.调整数组顺序使奇数位于偶数前面

题目描述

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

方法一 冒泡排序

时间复杂度O(n^2)，空间复杂度O(1)。

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

方法二 新建数组

时间复杂度O(n)，空间复杂度O(n)。

```
public class Solution {
    public void reOrderArray(int [] array) {
        int[] a=new int[array.length];
        for(int i=0;i<array.length;i++){
            a[i]=array[i];
        }
        int k=0;
        for(int i=0;i<a.length;i++){
            if(a[i]%2==1){
                array[k]=a[i];
                k++;
            }
        }
        for(int i=0;i<a.length;i++){
            if(a[i]%2==0){
                array[k]=a[i];
                k++;
            }
        }
    }
}
```

23.链表中倒数第k个结点

题目描述

输入一个链表，输出该链表中倒数第k个结点。

程序代码

时间复杂度O(n),空间复杂度O(1)

```
/*
public class ListNode {
    int val;
```

```
    ListNode next = null;

    ListNode(int val) {
        this.val = val;
    }
}*/
public class Solution {
    public ListNode FindKthToTail(ListNode head,int k) {
        int i=0;
        for(ListNode p=head;p!=null;p=p.next){
            i++;
        }
        int j=0;
        for(ListNode p=head;p!=null;p=p.next){
            if(j+k==i){
                return p;
            }
            j++;
        }
        return null;
    }
}
```

24.链表中环的入口结点

题目描述

给一个链表，若其中包含环，请找出该链表的环的入口结点，否则，输出null。

方法

慢指针p1,步长为1；快指针p2，步长为2。

假设慢指针走x步之后两指针相遇，那么有2x-x=nr，解得x=nr，其中r为环的长度，即慢指针走的长度为环长的倍数。

此时慢指针距离起点x，假设环入口距离起点y,那么慢指针距离环入口x-y。

快指针p2移到起点，且步长为1，与p1同时出发，如果p2到达环入口，那么p2走了y步。

此时慢指针也走了y步，慢指针距离环入口的长度为(x-y)+y=x=nr，所以慢指针也到达环入口。

代码

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

    ListNode(int val) {
        this.val = val;
    }
}
*/
public class Solution {
    public ListNode EntryNodeOfLoop(ListNode pHead){
        ListNode p1=pHead;
        ListNode p2=pHead;
        int flag=0;
```

```
        while(p2.next!=null){
            p1=p1.next;
            p2=p2.next.next;
            if(p1==p2){
                flag=1;
                break;
            }
        }
        if(flag==1){
            p2=pHead;
            while(true){
                if(p1==p2){
                    return p1;
                }
                p1=p1.next;
                p2=p2.next;
            }
        }
        return null;
    }
}
```

25.反转链表

题目描述

输入一个链表，反转链表后，输出新链表的表头。

方法一 新建空头节点迭代

时间复杂度O(n),空间复杂度O(1)。

1.创建新链表空头节点。

2.保存旧链表头指针head。

3.旧链表头节点的next指针指向新链表空头节点的next。

4.新链表空头节点的指针指向旧链表头节点head。

5.旧链表头指针head后移。

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

    ListNode(int val) {
        this.val = val;
    }
}*/
public class Solution {
    public ListNode ReverseList(ListNode head) {
        ListNode p=new ListNode(-1);
        while(head!=null){
            ListNode mem=head.next;
            head.next=p.next;
            p.next=head;
            head=mem;
```

```
        }
        return p.next;
    }
}
```

方法二 递归

时间复杂度O(n),空间复杂度O(n)。

先反转后面的链表，走到链表的末端结点。

再将当前节点设置为后面节点的后续节点。

```
/*
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||head.next==null){
            return head;
        }
        ListNode node=ReverseList(head.next);
        head.next.next=head;
        head.next=null;
        return node;
    }
}
```

26.合并两个排序的链表

题目描述

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

方法一 新建空头节点迭代

时间复杂度O(n)空间复杂度O(1)。

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

    ListNode(int val) {
        this.val = val;
    }
}*/
public class Solution {
    public ListNode Merge(ListNode list1,ListNode list2) {
        ListNode list3=new ListNode(-1);
        ListNode p=list3;
```

```
        while(list1!=null&&list2!=null){
            if(list1.val<=list2.val){
                p.next=list1;
                list1=list1.next;
            }else{
                p.next=list2;
                list2=list2.next;
            }
            p=p.next;
        }
        if(list1!=null){
            p.next=list1;
        }
        if(list2!=null){
            p.next=list2;
        }
        return list3.next;
    }
}
```

方法二 递归

```
/*
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;
        }
        if(list1.val<=list2.val){
            list1.next=Merge(list1.next,list2);
            return list1;
        }else{
            list2.next=Merge(list1,list2.next);
            return list2;
        }

    }
}
```

27.树的子结构

题目描述

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

方法

【HasSubtree函数】

当Tree1和Tree2都不为null的时候，进行比较

如果找到了对应Tree2的根节点的点，以这个根节点为起点dfs判断是否包含Tree2

如果找不到，那么就再去root的左儿子当作起点，去判断是否包含Tree2

如果还找不到，那么就去root的右儿子当作起点，去判断是否包含Tree2

返回结果

【dfs函数】

如果Tree2已经遍历完都能对应，返回True

如果Tree2没有遍历完，Tree却遍历完，返回false

如果其中有一个点没有对应上，返回false

如果根节点对应的上，那么就分别dfs取子节点里面匹配

代码

```
/**
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) {
        boolean flag=false;
        if(root1!=null&&root2!=null){
            if(root1.val==root2.val){
                flag=dfs(root1,root2);
            }
            if(!flag){
                flag=HasSubtree(root1.left,root2);
            }
            if(!flag){
                flag=HasSubtree(root1.right,root2);
            }
        }
        return flag;
    }
    private boolean dfs(TreeNode node1,TreeNode node2){
        if(node2==null){
            return true;
        }
        if(node1==null){
            return false;
        }
```

```
        if(node1.val!=node2.val){
            return false;
        }
        return dfs(node1.left,node2.left)&&dfs(node1.right,node2.right);
    }
}
```

28.二叉树的镜像

题目描述

操作给定的二叉树，将其变换为源二叉树的镜像。

输入描述

二叉树的镜像定义：源二叉树

```
    8
   / \
  6  10
 /\ /\
5 7 9 11
```

镜像二叉树

```
    8
   / \
  10  6
 /\ /\
11 9 7 5
```

代码

```
/**
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);
    }
}
```

29.对称的二叉树

题目描述

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。

代码

```
/*
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 dfs(pRoot.left,pRoot.right);
    }
    boolean dfs(TreeNode p1,TreeNode p2){
        if(p1==null&&p2==null){
            return true;
        }
        if(p1==null||p2==null){
            return false;
        }
        if(p1.val!=p2.val){
            return false;
        }
        return dfs(p1.left,p2.right)&&dfs(p1.right,p2.left);
    }
}
```

30.顺时针打印矩阵

题目描述

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

方法

设置矩阵的up,down,left,right，每打印一圈的数字，up++,down--,left++,right--。

代码

```
import java.util.ArrayList;
public class Solution {
    public ArrayList<Integer> printMatrix(int [][] matrix) {
        ArrayList<Integer> res=new ArrayList<>();
```

```
        int up=0;
        int down=matrix.length-1;
        int left=0;
        int right=matrix[0].length-1;
        while(left<=right&&up<=down){
            for(int i=left;i<=right;i++){
                res.add(matrix[up][i]);
            }
            for(int i=up+1;i<=down;i++){
                res.add(matrix[i][right]);
            }
            if(up!=down){
                for(int i=right-1;i>=left;i--){
                    res.add(matrix[down][i]);
                }
            }
            if(left!=right){
                for(int i=down-1;i>=up+1;i--){
                    res.add(matrix[i][left]);
                }
            }
            up++;
            down--;
            left++;
            right--;
        }
        return res;
    }
}
```

31.包含min函数的栈

题目描述

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

程序代码

```
import java.util.Stack;

public class Solution {

    Stack<Integer> dataStack=new Stack<>();
    Stack<Integer> minStack=new Stack<>();
    public void push(int node) {
        dataStack.push(node);
        if(minStack.isEmpty()){
            minStack.push(node);
        }else{
            minStack.push(min(minStack.peek(),node));
        }
    }
    int min(int a,int b){
        return a<b?a:b;
    }
    public void pop() {
        dataStack.pop();
```

```
            minStack.pop();
        }

        public int top() {
            return dataStack.peek();
        }

        public int min() {
            return minStack.peek();
        }
    }
```

32.栈的压入、弹出序列

题目描述

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

方法

创建辅助栈。

遍历压栈顺序数组放入栈。

判断栈顶元素与出栈顺序数组是否相等。

代码

```
import java.util.ArrayList;
import java.util.Stack;
public class Solution {
    public boolean IsPopOrder(int [] pushA,int [] popA) {
        Stack<Integer> s=new Stack<Integer>();
        int j=0;
        for(int i=0;i<pushA.length;i++){
            s.push(pushA[i]);
            while(!s.isEmpty()&&popA[j]==s.peek()){
                s.pop();
                j++;
            }
        }
        return s.isEmpty();
    }
}
```

33.从上往下打印二叉树

题目描述

从上往下打印出二叉树的每个节点，同层节点从左至右打印。

方法

将第一个元素加入队列

队列不为空时取队首元素

将下一层元素加入队尾

调到第二步，直到队列为空

代码

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

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

    }

}
*/
public class Solution {
    public ArrayList<Integer> PrintFromTopToBottom(TreeNode root) {
        Queue<TreeNode> q=new LinkedList<>();
        ArrayList<Integer> r=new ArrayList<>();
        q.add(root);
        while(!q.isEmpty()){
            int count=q.size();
            for(int i=0;i<count;i++){
                TreeNode t=q.poll();
                if(t!=null){
                    r.add(t.val);
                    q.add(t.left);
                    q.add(t.right);
                }
            }
        }
        return r;
    }
}
```

34.把二叉树打印成多行

题目描述

从上到下按层打印二叉树，同一层结点从左至右输出。每一层输出一行。

代码

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;

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

```
    TreeNode right = null;

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

    }
}
*/
public class Solution {
    ArrayList<ArrayList<Integer> > Print(TreeNode pRoot) {
        Queue<TreeNode> q=new LinkedList<>();
        ArrayList<ArrayList<Integer>> r=new ArrayList<>();
        q.add(pRoot);
        while(!q.isEmpty()){
            int count=q.size();
            ArrayList<Integer> list=new ArrayList<>();
            for(int i=0;i<count;i++){
                TreeNode node=q.poll();
                if(node!=null){
                    list.add(node.val);
                    q.add(node.left);
                    q.add(node.right);
                }
            }
            if(list.size()!=0){
                r.add(list);
            }
        }
        return r;
    }
}
```

35.按之字形顺序打印二叉树

题目描述

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

代码

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;
/*
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) {
        Queue<TreeNode> q=new LinkedList<>();
        ArrayList<ArrayList<Integer>> r=new ArrayList<>();
        q.add(pRoot);
        int flag=0;
        while(!q.isEmpty()){
            int count=q.size();
            ArrayList<Integer> list=new ArrayList<>();
            for(int i=0;i<count;i++){
                TreeNode node=q.poll();
                if(node!=null){
                    list.add(node.val);
                    q.add(node.left);
                    q.add(node.right);
                }
            }
            flag++;
            if(flag%2==0){
                list=rever(list);
            }
            if(list.size()!=0){
                r.add(list);
            }
        }
        return r;
    }
    private ArrayList<Integer> rever(ArrayList<Integer> list){
        ArrayList<Integer> r=new ArrayList<>();
        for(int i=list.size()-1;i>=0;i--){
            r.add(list.get(i));
        }
        return r;
    }
}
```

36.二叉搜索树的后序遍历序列

题目描述

输入一个整数数组，判断该数组是不是某二叉搜索树的后序遍历的结果。如果是则输出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。

方法

1.序列最后一个元素是根。

2.根将前面序列分为两段，左子树和右子树。

3.左子树序列最后一个元素是根，右子树序列最后一个元素是根。

4.递归。

代码

```
public class Solution {
    public boolean VerifySquenceOfBST(int [] sequence) {
        if(sequence.length==0){
            return false;
        }
```

```
        return dfs(sequence,0,sequence.length-1);
    }
    boolean dfs(int[] s,int first,int last){
        if(last<first){
            return true;
        }
        int rootVal=s[last];
        int cut;
        for(cut=first;cut<last;cut++){
            if(s[cut]>rootVal){
                break;
            }
        }
        for(int i=cut;i<last;i++){
            if(s[i]<rootVal){
                return false;
            }
        }
        return dfs(s,first,cut-1)&&dfs(s,cut,last-1);
    }
}
```

37.二叉树和为某一值的路径

题目描述

输入一颗二叉树的根节点和一个整数，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)

方法

1.用前序遍历dfs的方式访问某一节点时，把节点添加到路径中，并在target中减掉。

2.如果该节点时叶子节点并且目标值target为零，则当前路径符合要求，打印列表。

3.如果当前节点不是叶子节点，继续访问叶子节点。

4.当前节点访问结束后，递归函数将自动回到它的父节点，在路径列表中删除当前节点。

代码

```
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>> ret=new ArrayList<>();
    public ArrayList<ArrayList<Integer>> FindPath(TreeNode root,int target) {
```

```
        ArrayList<Integer> list=new ArrayList<>();
        dfs(root,target,list);
        return ret;
    }
    private void dfs(TreeNode node,int target,ArrayList<Integer> list){
        if(node==null){
            return;
        }
        list.add(node.val);
        target=target-node.val;
        if(target==0&&node.left==null&&node.right==null){
            ArrayList<Integer> temp=new ArrayList<>(list);
            ret.add(temp);
        }else{
            dfs(node.left,target,list);
            dfs(node.right,target,list);
        }
        list.remove(list.size()-1);
    }
}
```

38.复杂链表的复制

题目描述

输入一个复杂链表（每个节点中有节点值，以及两个指针，一个指向下一个节点，另一个特殊指针指向任意一个节点），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）

方法

1.在每个节点后面插入复制的节点

2.对复制节点的random链接进行赋值

3.拆分

代码

```
/*
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;
        // 插入新节点
        RandomListNode p = pHead;
        while (p != null) {
            RandomListNode node = new RandomListNode(p.label);
            node.next = p.next;
```

```
            p.next = node;
            p = p.next.next;
        }
        // 建立 random 链接
        p = pHead;
        while (p != null) {
            if(p.random!=null){
                p.next.random = p.random.next;
            }
            p = p.next.next;
        }
        // 拆分
        p = pHead;
        RandomListNode ret = pHead.next;
        while (p.next!= null) {
            RandomListNode mem=p.next;
            p.next = p.next.next;
            p=mem;
        }
        return ret;
    }
}
```

39.二叉搜索树与双向链表

题目描述

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

方法

二叉搜索树的中序遍历就是一个排序顺序表

代码

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

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

    }

}
*/
public class Solution {
    private TreeNode head=null;
    private TreeNode p=null;
    public TreeNode Convert(TreeNode pRootOfTree) {
        dfs(pRootOfTree);
        return head;
    }
    private void dfs(TreeNode node){
        if(node==null){
```

```
            return;
        }
        dfs(node.left);
        if(p==null){
           p=node;
           head=node;
        }else{
            p.right=node;
            node.left=p;
            p=p.right;
        }
        dfs(node.right);
    }
}
```

40.序列化二叉树

题目描述

请实现两个函数，分别用来序列化和反序列化二叉树

二叉树的序列化是指：把一棵二叉树按照某种遍历方式的结果以某种格式保存为字符串，从而使得内存中建立起来的二叉树可以持久保存。序列化可以基于先序、中序、后序、层序的二叉树遍历方式来进行修改，序列化的结果是一个字符串，序列化时通过 某种符号表示空节点（#），以！ 表示一个结点值的结束（value!）。

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

以前序遍历为例编写程序。

代码

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

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

    }

}
*/
public class Solution {
    String Serialize(TreeNode root) {
        StringBuilder str=new StringBuilder();
        dfs1(root,str);
        return str.toString();
   }
    void dfs1(TreeNode root,StringBuilder str){
        if(root==null){
            str.append("#!");
        }else{
            str.append(root.val+"!");
            dfs1(root.left,str);
            dfs1(root.right,str);
```

```
        }
    }
    int i=-1;
    TreeNode Deserialize(String str) {
        return dfs2(str);
   }
    TreeNode dfs2(String str){
        i++;
        String[] s=str.split("!");
        TreeNode node=null;
        if(!s[i].equals("#")){
            node=new TreeNode(Integer.valueOf(s[i]));
            node.left=dfs2(str);
            node.right=dfs2(str);
        }
        return node;
    }
}
```

41.字符串的排列

题目描述

输入一个字符串,按字典序打印出该字符串中字符的所有排列。例如输入字符串abc,则打印出由字符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。

思路

1.求出所有可能出现再第一位置的字母，即begin与后面所有与它不同的字母进行交换。

2.固定第一个字母，求后面字母的全排列。

代码

```
import java.util.ArrayList;
import java.util.Collections;
public class Solution {
    public ArrayList<String> Permutation(String str) {
        ArrayList<String> list=new ArrayList<>();
        if(str.length()==0){
            return list;
        }
        char[] c=str.toCharArray();
        //递归的初始值为(str数组，空list,初始下标0)
        dfs(c,list,0);
        Collections.sort(list);
        return list;
    }
    private void dfs(char[] c,ArrayList<String> list,int i){
        //如果下标移动到c数组的末尾，将c转为字符串加入list中
        if(i==c.length-1){
            String s=new String(c);
            //判断是否重复
            if(!list.contains(s)){
                list.add(s);
```

```
                return;
            }
        }else{
            for(int j=i;j<c.length;j++){
                swap(c,i,j);
                dfs(c,list,i+1);
                swap(c,i,j);
            }
        }
    }
    private void swap(char[] c,int i,int j){
        char temp=c[i];
        c[i]=c[j];
        c[j]=temp;
    }
}
```

42.数组中出现次数超过一半的数字

题目描述

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

方法

要求时间复杂度O(n)，空间复杂度O(1).

摩尔投票算法

1.如果有符合条件的数字，那么它出现的次数比其他所有数字出现的次数和还要多。

2.在遍历数组时保存两个值，数字和次数。

3.遍历下一个数字时，如果它与之前保存到数字相同，那么次数+1，否则次数-1。

4.如果次数为0，那么保存下一个数字，并将次数置为1。

5.遍历结束后，保存的数字即出现次数最多的数字，判断是否符合条件。

代码

```
public class Solution {
    public int MoreThanHalfNum_Solution(int [] array) {

        int ret=array[0];
        int cnt=1;
        for(int i=1;i<array.length;i++){
            if(cnt==0){
                ret=array[i];
                cnt=1;
            }
            if(array[i]==ret){
                cnt++;
            }else{
                cnt--;
            }
        }
        cnt=0;
```

```
        for(int i=0;i<array.length;i++){
            if(array[i]==ret){
                cnt++;
            }
        }
        if(cnt>array.length/2){
            return ret;
        }else{
            return 0;
        }
    }
}
```

43.最小的k个数

题目描述

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

方法一 快速分割

基于数组的第k个数字来调整，使得比第k个数字小的所有数字都位于数组的左边，比第k个数字大的所有数字都位于数组的右边。调整之后，位于数组左边的k个数字就是最小的k个数字（这k个数字不一定是排序的）。

时间复杂度O(N),空间复杂度O(1)。

```
import java.util.ArrayList;
public class Solution {
    public ArrayList<Integer> GetLeastNumbers_Solution(int [] input, int k) {
        ArrayList<Integer> ret=new ArrayList<>();
        if(k<=0||k>input.length){
            return ret;
        }
        int start=0;
        int end=input.length-1;
        int pIndex=part(input,start,end);//主元下标
        while(pIndex!=k-1){//如果主元下标等于k,结束循环
            if(pIndex<k-1){//如果主元下标小于k
                start=pIndex+1;//对大于k的子序列进行分割
                pIndex=part(input,start,end);//继续返回子序列的主元下标
            }else{//如果主元下标大于k
                end=pIndex-1;//然后对小于k的子序列进行分割
                pIndex=part(input,start,end);//继续返回子序列的主元下标
            }
        }
        for(int i=0;i<k;i++){
            ret.add(input[i]);
        }
        return ret;
    }
    private int part(int[] a,int left,int right){
        int p=a[left];//主元为左指针元素
        while(left<right){//如果左指针等于右指针，结束循环
            while(left<right&&a[right]>=p){//如果右指针指元素大于等于主元
                right--;//右指针左移
            }
```

```
            swap(a,left,right);//右指针找到比主元小的元素，交换
            while(left<right&&a[left]<p){//如果左指针元素小于主元
                left++;//左指针右移
            }
            swap(a,left,right);//左指针找到比主元大的元素，交换
        }
        return left;//返回将数组分割为两半的主元下标
    }
    private void swap(int[] a,int i,int j){
        int t=a[i];
        a[i]=a[j];
        a[j]=t;
    }
}
```

方法二 最大堆

最大堆是一个完全二叉树，每个父节点大于子节点，树根节点最大。

优先队列的每次出队都是队列中最小或者最大元素，可以通过最大堆实现优先队列。

使用数组前k个元素初始化一个容量为k的优先队列。

然后遍历数组a[k]至a[n-1]，如果有比队首小的元素，插入队列，然后把队首元素删除。

时间复杂度O(NlogK)，空间复杂度O(k)。

```
import java.util.Comparator;
import java.util.ArrayList;
import java.util.PriorityQueue;
public class Solution {
    public ArrayList<Integer> GetLeastNumbers_Solution(int [] input, int k) {
        ArrayList<Integer> ret=new ArrayList<>();
        if(input==null||k<=0||k>input.length){
            return ret;
        }
        Comparator<Integer> cmp=new Comparator<Integer>(){
            @Override //重载比较器，降序队列
            public int compare(Integer o1,Integer o2){
                return o2-o1;
            }
        };
        PriorityQueue<Integer> q=new PriorityQueue<>(k,cmp);
        for(int i=0;i<input.length;i++){
            if(i<k){
                q.add(input[i]);
            }else{
                if(input[i]<q.peek()){
                    q.poll();
                    q.add(input[i]);
                }
            }
        }
        for(int i=0;i<k;i++){
            ret.add(q.peek());
            q.poll();
        }
        return ret;
    }
```

```
}
```

44.数据流中的中位数

题目描述

如何得到一个数据流中的中位数？如果从数据流中读出奇数个数值，那么中位数就是所有数值排序之后位于中间的数值。如果从数据流中读出偶数个数值，那么中位数就是所有数值排序之后中间两个数的平均值。我们使用Insert()方法读取数据流，使用GetMedian()方法获取当前读取数据的中位数。

方法

1.使用java集合PriorityQueue设置一个小顶堆和大顶堆

2.大顶堆存较小的数，从大到小排列

3.小顶堆存较大的数，从小到大排列

4.要求小顶堆中的元素都大于等于大顶堆中的元素

5.当已排序数目为偶数时，元素插入大顶堆，再将大顶堆中最大值插入小顶堆

6.当已排序数目为奇数时，元素插入小顶堆，再将小顶堆中最小值插入大顶堆

7.如果当前个数为偶数，中位数为小顶堆和大顶堆根节点的平均值

8.如果当前个数为奇数，中位数为小顶堆的根节点

代码

```
import java.util.PriorityQueue;
import java.util.Comparator;
public class Solution {

    Comparator<Integer> cmp=new Comparator<Integer>(){
        @Override
        public int compare(Integer o1,Integer o2){
            return o2-o1;
        }
    };
    private PriorityQueue<Integer> minHeap=new PriorityQueue<>();
    private PriorityQueue<Integer> maxHeap=new PriorityQueue<>(cmp);

    int count=0;
    public void Insert(Integer num) {
        if(count%2==0){
            maxHeap.offer(num);
            int max=maxHeap.poll();
            minHeap.offer(max);
        }else{
            minHeap.offer(num);
            int min=minHeap.poll();
            maxHeap.offer(min);
        }
        count++;
    }

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

```
        }else{
            return new Double(minHeap.peek());
        }
    }
}
```

45.字符流中第一个不重复的字符

题目描述

请实现一个函数用来找出字符流中第一个只出现一次的字符。例如，当从字符流中只读出前两个字符"go"时，第一个只出现一次的字符是"g"。当从该字符流中读出前六个字符“google"时，第一个只出现一次的字符是"l"。

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。

方法

时间复杂度O(1),空间复杂度O(n)

1.用一个256大小的数组统计每个字符出现的次数

2.用一个队列插入元素

3.如果队列非空并且队首元素的次数大于1次，该队首元素出列。

4.如果最后队列为空，则没有只出现一次的字符返回#；否则返回第一个只出现一次的字符。

代码

```
import java.util.*;
public class Solution {
    private int[] cnts=new int[256];
    private Queue<Character> queue=new LinkedList<>();
    //Insert one char from stringstream
    public void Insert(char ch)
    {
        cnts[ch]++;
        queue.add(ch);
        while(!queue.isEmpty()&&cnts[queue.peek()]>1){
            queue.poll();
        }
    }
    //return the first appearence once char in current stringstream
    public char FirstAppearingOnce()
    {
        if(queue.isEmpty()){
            return '#';
        }else{
            return queue.peek();
        }
    }
}
```

46.连续子数组的最大和

题目描述

给定一个数组array，计算数组的最大连续子序列的和.

方法-DP

1.计算以array[i]为结尾的子数组和的最大值.

状态转移方程：dp[i]=max{dp[i-1]+array[i],array[i]}.

2.取最大的一个max{dp[i]}.

时间复杂度O(n).

代码

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

47.从1到n整数中1出现的次数

方法

假设对于一个k位的数字n按照数位分类讨论：

- 1.个位上1出现的个数：
  - `n/10+(n%10!=0?1:0)`
- 2.十位上1出现的个数：
  - `k=n%100`
  - `(n/100)*10+(if(k>19) 10 else if(k<10) 0 else k-10+1`
- 3.百位上1出现的个数：
  - `k=n%1000`
  - `(n/1000)*100+(if(k>199) 100 else if(k<100) 0 else k-100+1`

归纳：

- 1.个位
  - `k=n%10`
  - `(n/10)*1+(if(k>1) 1 else if(k<1) 0 else k-1+1)`
- 2.i位(i=1表示个位，i=10表示十位)
  - `k=n%(i*10)`
  - `(n/(i*10))*i+(if(k>i*2-1) i else if(k<i) 0 else k-i+1)`

效率

- 因为数字n有O(logn)位，所以时间复杂度O(logn).

```
public class Solution {
```

```
    public int NumberOf1Between1AndN_Solution(int n) {
        int count=0;
        for(int i=1;i<=n;i=i*10){
            int k=n%(i*10);
            if(k<i){
                count+=(n/(i*10))*i;
            }else if(k>i*2-1){
                count+=(n/(i*10))*i+i;
            }else{
                count+=(n/(i*10))*i+(k-i+1);
            }
        }
        return count;
    }
}
```

48.把数组排成最小的数

题目描述

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

方法

自定义一个比较大小的函数，比较两个字符串s1, s2大小的时候，先将它们拼接起来，比较s1+s2,和s2+s1那个大，如果s1+s2大，那说明s2应该放前面，所以按这个规则，s2就应该排在s1前面。

效率

时间复杂度：O(nlogn)。

代码

```
class Solution {
public:
    string PrintMinNumber(vector<int> numbers) {
        int len=numbers.size();
        if(len==0) return "";
        sort(numbers.begin(),numbers.end(),cmp);
        string res;
        for(int i=0;i<len;i++){
            res+=to_string(numbers[i]);
        }
        return res;
    }
    static bool cmp(int a,int b){
        string A=to_string(a)+to_string(b);
        string B=to_string(b)+to_string(a);
        return A<B;
    }
};
```

49.丑数

题目描述

把只包含质因子2、3和5的数称作丑数（Ugly Number）。例如6、8都是丑数，但14不是，因为它包含质因子7。 习惯上我们把1当做是第一个丑数。求按从小到大的顺序的第N个丑数。

解题思路

丑数由`2*i,3*i,4*i`组成。

第一个丑数是`1`

第二个丑数是`min(2*1,3*1,5*1)=2`

第三个丑数是`min(2*2,3*1,5*1)=3`

第四个丑数是`min(2*2,3*2,5*1)=4`

第五个丑数是`min(2*3,3*2,5*1)=5`

第六个丑数是`min(2*3,3*2,5*2)=6`

第k个丑数是`dp[k]=min(2*dp[t2],3*dp[t3],5*dp[t5])`

效率

- 时间复杂度：O(n)
- 空间复杂度：O(n)

代码

```
class Solution {
public:
    int GetUglyNumber_Solution(int index) {
        if(index<7) return index;
        vector<int> dp(index);
        dp[0]=1;
        int t2=0,t3=0,t5=0;
        for(int i=1;i<index;i++){
            dp[i]=min(dp[t2]*2,min(dp[t3]*3,dp[t5]*5));
            if(dp[i]==dp[t2]*2) t2++;
            if(dp[i]==dp[t3]*3) t3++;
            if(dp[i]==dp[t5]*5) t5++;
        }
        return dp[index-1];
    }
};
```

50.第一个只出现一次的字符位置

题目描述

在一个字符串(0<=字符串长度<=10000，全部由字母组成)中找到第一个只出现一次的字符,并返回它的位置, 如果没有则返回 -1（需要区分大小写）.

解题思路

建立哈希表对字符str[i]进行计数，<str[i],count>。

效率

时间复杂度：O(n)

空间复杂度：O(n)

代码

```
class Solution {
public:
    int FirstNotRepeatingChar(string str) {
        map<char,int> a;
        for(int i=0;i<str.size();i++){
            a[str[i]]++;
        }
        for(int i=0;i<str.size();i++){
            if(a[str[i]]==1)
                return i;
        }
        return -1;
    }
};
```

51.数组中的逆序对

题目描述

在数组中的两个数字，如果前面一个数字大于后面的数字，则这两个数字组成一个逆序对。输入一个数组,求出这个数组中的逆序对的总数P。并将P对1000000007取模的结果输出。 即输出P%1000000007。

解题思路

归并排序。

效率

时间复杂度：O(nlogn)

空间复杂度：O(n)

代码

```
public class Solution {
    int count;
    public int InversePairs(int [] array) {
        count=0;
        if(array!=null)
            mergeSortCount(array,0,array.length-1);
        return count;
    }
    public void mergeSortCount(int[] a,int low,int high){
        int mid=(low+high)/2;
        if(low<high){
            mergeSortCount(a,low,mid);
            mergeSortCount(a,mid+1,high);
            merge(a,low,mid,high);
        }
    }
    public void merge(int[] a,int low,int mid,int high){
        int[] temp=new int[high-low+1];
        int i=low;
        int j=mid+1;
        int k=0;
        while(i<=mid&&j<=high){
            if(a[i]<a[j]){
                temp[k++]=a[i++];
```

```
            }else{
                temp[k++]=a[j++];
                count=(count+(mid-i+1))%1000000007;
            }
        }
        while(i<=mid){
            temp[k++]=a[i++];
        }
        while(j<=high){
            temp[k++]=a[j++];
        }
        for(int k2=0;k2<temp.length;k2++){
            a[k2+low]=temp[k2];
        }
    }
}
```

52.两个链表的第一个公共结点

题目描述

输入两个链表，找出它们的第一个公共结点。

方法

用两个指针扫描"两个链表"，最终两个指针如果同时到达null，返回null；如果同时到达公共结点，返回公共结点。

代码

```
/*
struct ListNode {
    int val;
    struct ListNode *next;
    ListNode(int x) :
            val(x), next(NULL) {
    }
};*/
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;
    }
};
```

53.数字在排序数组中出现的次数

题目描述

统计一个数字在排序数组中出现的次数。

方法

二分查找升序数组中第一个大于等于k的下标fisrt和第一个大于等于k+1的下标last，返回last-first。

代码

```
public class Solution {
    public int GetNumberOfK(int [] array , int k) {
        int first = binarySearch(array, k);
        int last = binarySearch(array, k + 1);
        if(first == array.length){
            return 0;
        }else{
            return last - first;
        }
    }
    private int binarySearch(int[] array,int k){
        int low = 0, high = array.length-1;
        while(low <= high){
            int mid = low + (high - low)/2;
            if(k <= array[mid]){
                high = mid - 1;
            }else{
                low = mid + 1;
            }
        }
        return low;
    }
}
```

54.二叉搜索树的第k个结点

题目描述

给定一棵二叉搜索树，请找出其中的第k小的结点。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。

方法

二叉搜索树中序遍历有序。

代码

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

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

    }

}
*/
public class Solution {
    private TreeNode ret;
    private int cnt = 0;
    TreeNode KthNode(TreeNode pRoot, int k){
```

```
        inOrder(pRoot,k);
        return ret;
    }
    private void inOrder(TreeNode root,int k){
        if(root == null || cnt >= k){
            return;
        }
        inOrder(root.left, k);
        cnt++;
        if(cnt == k){
            ret = root;
        }
        inOrder(root.right, k);
    }


}
```

55.二叉树的深度

题目描述

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

方法

DFS

代码

```
/**
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;
        }
        int left = TreeDepth(root.left);
        int right = TreeDepth(root.right);
        return left > right ? left + 1 : right + 1;
    }
}
```

56.平衡二叉树

题目描述

输入一棵二叉树，判断该二叉树是否是平衡二叉树。

方法

DFS

代码

```
public class Solution {
    private boolean isBalanced = true;
    public boolean IsBalanced_Solution(TreeNode root) {
        dfs(root);
        return isBalanced;
    }
    private int dfs(TreeNode root){
        if(root == null){
            return 0;
        }
        int left = dfs(root.left);
        int right = dfs(root.right);
        if((left - right > 1) || (left - right) < -1){
            isBalanced = false;
        }
        return left > right ? left + 1 : right + 1;
    }
}
```

57.数组中只出现一次的数字

题目描述

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

方法

1.任何一个数字异或自己都等于0。

2.找出两个不同数字的第一个不同位。

3.根据那个位进行位运算分组。

代码

```
//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 < 2){
            return;
        }
        int temp = 0;
        for(int i = 0; i < array.length; i++){
            temp = temp ^ array[i];
        }
        int index = findFirstOne(temp);
        for(int i = 0; i < array.length; i++){
            if(group(array[i], index)){
                num1[0] = num1[0] ^ array[i];
```

```
            }
            else{
                num2[0] = num2[0] ^ array[i];
            }
        }
    }
    private int findFirstOne(int num){
        int index = 0;
        while(((num & 1) == 0) && (index < 32)){
            num = num >> 1;
            index++;
        }
        return index;
    }
    private boolean group(int num, int index){
        num = num >> index;
        return (num & 1) == 1;
    }
}
```

58.和为S的两个数字

题目描述

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

方法

双指针：从数组两端向中间遍历。时间复杂度O(n)。

代码

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

59.和为S的连续正数序列

题目描述

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。

方法

left、right双指针遍历。

代码

```
import java.util.ArrayList;
public class Solution {
    public ArrayList<ArrayList<Integer> > FindContinuousSequence(int sum) {
        ArrayList<ArrayList<Integer>> res = new ArrayList<>();
        int left = 1, right = 2;
        while(right > left){
            int cur = (left + right) * (right - left + 1) / 2;
            if(cur < sum){
                right++;
            }else if(cur > sum){
                left++;
            }else{
                ArrayList<Integer> list = new ArrayList<>();
                for(int i = left; i <= right; i++){
                    list.add(i);
                }
                res.add(list);
                left++;
            }
        }
        return res;
    }
}
```

60.翻转单词顺序列

题目描述

```
Input:
"I am a student."

Output:
"student. a am I"
```

空间复杂度要求O(1)。

方法

1.将字符串转换成数组。

2.单词翻转。

3.整句翻转。

代码

```
public class Solution {
    public String ReverseSentence(String str) {
        char[] chars = str.toCharArray();
        int left = 0, right = 0;
        while(right <= str.length()){
            if(right == str.length() || chars[right] == ' '){
```

```
                reverse(chars, left, right - 1);
                left = right + 1;
            }
            right++;
        }
        reverse(chars, 0, str.length()-1);
        return new String(chars);
    }
    private void reverse(char[] chars, int left, int right){
        while(left < right){
            char temp = chars[left];
            chars[left] = chars[right];
            chars[right] = temp;
            left++;
            right--;
        }
    }
}
```

61.左旋转字符串

题目描述

```
Input:
S="abcXYZdef"
K=3

Output:
"XYZdefabc"
```

要求空间复杂度O(1)。

方法

三次翻转。

代码

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

```
        }
    }
}
```

62.滑动窗口的最大值

题目描述

给定一个数组和滑动窗口的大小，找出所有滑动窗口里数值的最大值。

例如，如果输入数组 {2, 3, 4, 2, 6, 2, 5, 1} 及滑动窗口的大小 3，那么一共存在 6 个滑动窗口，他们的最大值分别为 {4, 4, 6, 6, 6, 5}。

要求时间复杂度O(n)。

方法

1.如果当前数字大于队尾，则删除队尾，直到当前数字小于等于队尾，然后当前数字进入队尾。

2.如果当前数字小于队尾，则当前数字进入入队尾。

3.如果队首超出窗口范围，则删除队首元素。

代码

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

63.扑克牌顺子

题目描述

五张牌，其中大小鬼为癞子，牌面为 0。判断这五张牌是否能组成顺子。

方法

1.排序

2.计算0的个数

3.计算所有非0的相邻数字间隔总数

4.如果0的个数与相邻数字间隔总数相等就是顺子

5.如果非0相邻数字间隔之间出现对子，就不是顺子

代码

```
import java.util.Arrays;
public class Solution {
    public boolean isContinuous(int [] numbers) {
        if(numbers.length < 5){
            return false;
        }
        Arrays.sort(numbers);
        int count1 = 0;
        for(int i = 0; i < numbers.length; i++){
            if(numbers[i] == 0){
                count1++;
            }
        }
        int count2 = 0;
        for(int i = count1; i < numbers.length - 1; i++){
            if(numbers[i] == numbers[i+1]){
                return false;
            }
            count2 =count2 + numbers[i+1] - numbers[i] - 1;
        }
        return count1 >= count2 ? true : false;

    }
}
```

64.约瑟夫环问题

题目描述

已知n个人（以编号0，1，2...n - 1分别表示）围坐在一张圆桌周围。从编号为0的人开始报数，数到m的那个人出列；下一个人又从0开始报数，数到m的那个人又出列；依此规律重复下去，直到圆桌周围的人全部出列。

输出最后一个人的开始编号。

方法

1.n个人时，假设最后一个人的编号为dp[n]。因为数到m的那个人会出列，那么第一轮编号为(m-1)%n的人会出列，编号为(m+0)%n的人为下一轮报数为0的人，编号为(m+i)%n的人为下一轮报数为i的人。

2.n-1个人时，假设最后一个人的编号为dp[n-1]。报数为i的人对应着上一轮编号为(m+i)%n的人。

3.报数dp[n-1]的人对应上一轮编号为(m+dp[n-1])%n的人。所以有dp[n]=(dp[n-1]+m)%n。

代码

```
public class Solution {
    public int LastRemaining_Solution(int n, int m) {
        if(n == 0){
            return -1;
        }else if(n == 1){
            return 0;
        }else{
            return (LastRemaining_Solution(n - 1, m) + m) % n;
        }
    }
}
```

65.求1+2+3+...+n

题目描述

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

方法

使用递归解法最重要的是指定返回条件，但是本题无法直接使用 if 语句来指定返回条件。

条件与 && 具有短路原则，即在第一个条件语句为 false 的情况下不会去执行第二个条件语句。利用这一特性，将递归的返回条件取非然后作为 && 的第一个条件语句，递归的主体转换为第二个条件语句，那么当递归的返回条件为 true 的情况下就不会执行递归的主体部分，递归返回。

本题的递归返回条件为 n <= 0，取非后就是 n > 0；递归的主体部分为 sum += Sum_Solution(n - 1)，转换为条件语句后就是 (sum += Sum_Solution(n - 1)) > 0。

代码

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

66.不用加减乘除做加法

题目描述

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

方法

各位相加 + 计算进位，直到进位为0退出。

1.十进制思想

5+7 各位相加：2 进位：10

2+10 各位相加：12 进位：0

12+0

2.二进制计算过程

5+7 各位相加：101^111=010 进位：101&111=101 (<<1=1010)

2+10 各位相加：010^1010=1000 进位：010&1010=010 (<<1=0100)

8+4 各位相加：1000^0100=1100 进位1000&0100=0

12+0

代码

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

67.构建乘积数组

题目描述

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

方法

B[i]=A[0]*A[1]*...*A[i-1]*A[i+1]*...*A[n-1]
从左到右算B[i]=A[0]*A[1]*...*A[i-1]
从右到左算B[i]*=A[i+1]*...*A[n-1]

代码

```
import java.util.ArrayList;
public class Solution {
    public int[] multiply(int[] A) {
        int[] B = new int[A.length];
        int temp1 = 1;
        for(int left = 0; left < A.length; left++){
            B[left] = temp1;
            temp1 = temp1 * A[left];
        }
        int temp2 =1;
        for(int right = A.length - 1; right >= 0; right--){
            B[right] = B[right] * temp2;
            temp2 = temp2 * A[right];
        }
        return B;
    }
}
```

68.把字符串转换成整数

题目描述

将一个字符串转换成一个整数，字符串不是一个合法的数值则返回 0，要求不能使用字符串转换整数的库函数。

```
Iuput:
+2147483647
1a33

Output:
2147483647
0
```

注意

Integer.MAX_VALUE = 2147483647, Integer.MIN_VALUE = -2147483648。

代码

```
public class Solution {
    public int StrToInt(String str) {
        if(str.length() == 0){
            return 0;
        }
        boolean isNegative = (str.charAt(0) == '-');
        int ret = 0;
        for(int i = 0; i < str.length(); i++){
            if(i == 0 && (str.charAt(i) == '+' || str.charAt(i) == '-')){
                continue;
            }
            if(str.charAt(i) < '0' || str.charAt(i) > '9'){
                return 0;
            }
            if(i == str.length() - 1 && ret == Integer.MAX_VALUE / 10){
                if(!isNegative && str.charAt(i) > '7'){
                    return 0;
                }
                if(isNegative && str.charAt(i) > '8'){
                    return 0;
                }
            }
            ret = ret * 10 + (str.charAt(i) - '0');
        }
        return isNegative ? -ret: ret;
    }
}
```