# 剑指offer JAVA解

- https://github.com/xurui1995/Sword-pointing-to-offer
- 对应的书籍为12年出版的剑指offer
- 所有代码经过本人实现并通过，实现思路请参考书中讲解
- 欢迎提交bug和更优解

## 公共类

BinaryTreeNode

```java
public class BinaryTreeNode {
    private int data;
    private BinaryTreeNode LchildNode;
    private BinaryTreeNode RchildNode;
    public BinaryTreeNode(int data) {
        super();
        this.data = data;
    }

    public int getData() {
        return data;
    }

    public void setData(int data) {
        this.data = data;
    }

    public BinaryTreeNode getLchildNode() {
        return LchildNode;
    }

    public void setLchildNode(BinaryTreeNode lchildNode) {
        LchildNode = lchildNode;
    }

    public BinaryTreeNode getRchildNode() {
        return RchildNode;
    }

    public void setRchildNode(BinaryTreeNode rchildNode) {
        RchildNode = rchildNode;
    }
}
```

Node

```
class Node {
    String data;
    Node next;
    public Node(String data) {
        super();
        this.data = data;
    }

    public Node(String data, Node next) {
        super();
        this.data = data;
        this.next = next;
    }

    public String getData() {
        return data;
    }

    public void setData(String data) {
        this.data = data;
    }

    public Node getNext() {
        return next;
    }

    public void setNext(Node next) {
        this.next = next;
    }

}
```

## 题解

> 2. 设计一个类，我们只能生成该类的一个实例。

```
// 饿汉式  线程安全
class A {
    private static final A instance = new A();
    private A() {
    }

    public static A getInstance() {
        return instance;
    }
}

// 懒汉式 线程安全写法
class B {
    private static volatile B instance = null;

    private B() {
    }

    public static B getInstance() {
        if (instance == null) {
            synchronized (B.class) {
                if (instance == null)
                    instance = new B();
            }
        }
        return instance;
    }
}

// 静态内部类方式 线程安全
class C {
    private C() {

    }
    public static C getInstance() {
        return CHolder.INSTANCE;
    }

    private static class CHolder {
        private static final C INSTANCE = new C();
    }
}
```

> 3. 在一个二维数组中，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，判断数组中是否函数该整数。

```
public class No03 {


    public static void main(String[] args) {
        int[][] arr = {{1, 2, 8, 9},
                {2, 4, 9, 12},
                {4, 7, 10, 13},
                {6, 8, 11, 15}};

        System.out.println(search(arr, 7));
    }

    private static boolean search(int[][] arr, int value) {

        int a = arr[0].length;
        int b = arr.length;
        int i = 0;
        int j = a - 1;

        while (i <= b - 1 && j >= 0) {
            if (arr[i][j] == value) {
                return true;
            }
            if (arr[i][j] > value) {
                j--;
            } else {
                i++;
            }
        }
        return false;
    }
}
```

> 4. 请实现一个函数，把字符串中的每个空格替换成"%20"。例如输入"We are happy"，则输出"We%20are%20happy"

```
public class No04 {

    public static void main(String[] args) {
        String str = "We are happy";
```

```java
        char[] charArray = str.toCharArray();
        System.out.println(change(charArray));

    }

    private static String change(char[] charArray) {

        int n = charArray.length;
        int count = 0;
        for (int i = 0; i < charArray.length; i++) {
            if (charArray[i] == ' ') {
                count++;
            }
        }
        if (count == 0) {
            return null;
        }
        char[] temp = new char[n + 2 * count];
        int j = n + 2 * count - 1;
        int i = n - 1;
        while (i >= 0) {
            if (charArray[i] == ' ') {
                temp[j] = '0';
                temp[j - 1] = '2';
                temp[j - 2] = '%';
                j = j - 3;
            } else {
                temp[j] = charArray[i];
                j--;
            }
            i--;
        }
        return new String(temp);
    }
}
```

> 5. 输入一个链表的头节点，从尾到头打印每个节点的值

```java
public class No05 {

    public static void main(String[] args) {
        Node node1 = new Node("A");
        Node node2 = new Node("B");
        Node node3 = new Node("C");

        Node node4 = new Node("D");
        Node node5 = new Node("E");
        node1.setNext(node2);
        node2.setNext(node3);
        node3.setNext(node4);
        node4.setNext(node5);
        Node newNode = reverse2(node1);
        while (newNode != null) {
            System.out.print(newNode.data + " ");
            newNode = newNode.getNext();
        }

    }

    private static Node reverse(Node head) {
        if (head.next == null) {
            return head;
        }
        Node reverseHead = reverse(head.getNext());
        head.getNext().setNext(head);
        head.setNext(null);
        return reverseHead;
    }

    private static Node reverse2(Node head) {
        Node pre = head;
        Node cur = head.getNext();
        Node temp;
        while (cur != null) {
            temp = cur.getNext();
            cur.setNext(pre);
            pre = cur;
            cur = temp;
        }
        head.setNext(null);
        return pre;
    }
}
```

> 6. 根据前序遍历和中序遍历建立树

```
public class No06 {
    public static void main(String[] args) {
        String preOrder = "12473568";
        String midOrder = "47215386";
        BiTree tree = new BiTree(preOrder, midOrder,
preOrder.length());
        tree.postRootTraverse(tree.root);
    }
}

class BiTree {
    TreeNode root;

    public BiTree(String preOrder, String midOrder, int count) {
        if (count <= 0) {
            return;
        }
        char c = preOrder.charAt(0);
        int i = 0;
        for (; i < count; i++) {
            if (midOrder.charAt(i) == c)
                break;
        }

        root = new TreeNode(c);
        root.setLchild(new BiTree(preOrder.substring(1, i + 1),
midOrder.substring(0, i), i).root);
        root.setRchild(new BiTree(preOrder.substring(i + 1),
midOrder.substring(i + 1), count - i - 1).root);
    }


    public void postRootTraverse(TreeNode root) {
        if (root != null) {
            postRootTraverse(root.getLchild());
            postRootTraverse(root.getRchild());
            System.out.print(root.getData());
        }
    }
}
```

> 7. 两个栈建立队列

```
import java.util.Stack;

public class No07 {

    private Stack s1 = new Stack();
    private Stack s2 = new Stack();

    public void offer(Object x) {
```

```
    public void offer(Object x) {
        s1.push(x);

    }

    public void poll() {
        if (s1.size() == 0 && s2.size() == 0) {
            try {
                throw new Exception("empty queue");
            } catch (Exception e) {
                e.printStackTrace();
            }
        } else {
            if (s2.size() != 0) {
                System.out.println(s2.peek().toString());
                s2.pop();
            } else {
                while (s1.size() > 0) {
                    s2.push(s1.pop());
                }
                System.out.println(s2.peek().toString());
                s2.pop();
            }
        }

    }

    public static void main(String[] args) {

        No07 queue = new No07();
        queue.offer("a");
        queue.offer("b");
        queue.offer("c");
        queue.poll();
        queue.poll();
        queue.poll();
        queue.poll();
    }

}
```

> 8. 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。输入一个递增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。

```
public class No08 {

    public static void main(String[] args) {
        int[] arr = {3, 4, 5, 1, 2};
        System.out.println(findMin(arr));

    }

    public static int findMin(int[] arr) {
        int left = 0;
        int right = arr.length - 1;
        if (arr[right] > arr[left]) {
            try {
                throw new Exception("非旋转数组");
            } catch (Exception e) {
                e.printStackTrace();
                return -1;
            }
        }
        while (left < right) {
            int mid = (left + right) / 2;
            //对于{1,0,1,1,1}之类的特殊处理
            if (arr[mid] == arr[left] && arr[left] == arr[right]) {
                return seachMin(arr, left, right);
            }
            if (right - left == 1)
                break;
            if (arr[mid] >= arr[left]) {
                left = mid;
            } else {
                right = mid;
            }
        }
        return arr[right];
    }

    private static int seachMin(int[] arr, int left, int right) {
        int result = arr[left];
        for (int i = left + 1; i <= right; ++i) {
            if (arr[i] < result)
                result = arr[i];
        }
        return result;
    }

}
```

> 9. 写一个函数，输入n，求斐波那契数列的第n项（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级台阶总共有多少种跳法）

```
public class No09 {
```

```java
public class No09 {

    public static void main(String[] args) {
        System.out.println(fibonacci(5));
        System.out.println(getMethodNumber(10));
    }

    private static long fibonacci(int n) {
        long[] a = {0, 1};
        if (n < 2)
            return a[n];
        long fib1 = 0;
        long fib2 = 1;
        long fibN = 0;
        for (int i = 2; i <= n; i++) {
            fibN = fib1 + fib2;
            fib1 = fib2;
            fib2 = fibN;
        }
        return fibN;

    }


    private static int getMethodNumber(int n) {
        if (n == 0)
            return 0;
        if (n == 1)
            return 1;
        if (n == 2)
            return 2;

        return getMethodNumber(n - 1) + getMethodNumber(n - 2);
    }
}
```

> 10. 请实现一个函数，输入一个整数，输出该二进制表示中1的个数。例如把9表示成二进制是1001，有2位是1。

```java
public class No10 {

    public static void main(String[] args) {
        System.out.println(getNum(9));
    }

    public static int getNum(int n) {
        int num = 0;
        while (n != 0) {
            num++;
            n = (n - 1) & n;
        }
        return num;
    }
}
```

> 11. 实现函数double Power(double base,int exponent)，求base的exponent次方。不得使用库函数，同时不需要考虑大数问题。

```
public class No11 {

    public static void main(String[] args) {
        System.out.println(Power(2.0, 3));
    }

    public static double Power(double base, int exponent) {
        if (exponent == 0)
            return 1;
        if (exponent == 1)
            return base;

        double result = Power(base, exponent >> 1);
        result *= result;
        if ((exponent & 0x1) == 1) {
            result *= base;
        }
        return result;
    }
}
```

> 12. 输入数字n，按顺序打印出从1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。

```
public class No12 {

    public static void main(String[] args) {
        printNum(3);
    }

    private static void printNum(int n) {
        if (n < 0)
            return;
        int[] array = new int[n];
        printArray(array, 0);
    }

    private static void printArray(int[] array, int n) {

        if (n != array.length) {
            for (int i = 0; i < 10; i++) {
                array[n] = i;
                printArray(array, n + 1);
            }
        } else {
            boolean flag = false;
            for (int j = 0; j < array.length; j++) {
                if (array[j] != 0) {
                    flag = true;
                }
                if (flag) {
                    System.out.print(array[j]);
                }
            }
            // 去掉空白行
            if (flag) {
                System.out.println();
            }
        }
    }
}
```

> 13. 给定单向链表的头指针和一个结点指针， 定义一个函数在O（1）时间删除该节点

```
public class No13 {

    public static void main(String[] args) {
        Node a = new Node("A");
        Node b = new Node("B");
        Node c = new Node("C");

        Node d = new Node("D");
        a.setNext(b);
        b.setNext(c);
        c.setNext(d);
        delete(a, d);
        Node temp = a;
        while (temp != null) {
            System.out.println(temp.getData());
            temp = temp.next;
        }
    }

    private static void delete(Node head, Node c) {
        // 如果是尾节点,只能遍历删除
        if (c.next == null) {
            while (head.next != c) {
                head = head.next;
            }
            head.next = null;
        } else if (head == c) {
            head = null;
        } else {
            c.setData(c.getNext().getData());
            c.setNext(c.getNext().getNext());
        }

    }
}
```

> 14. 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，使得所有奇数位于数组的前半部分，所有偶数位于数组的后半部分。

```
public class No14 {

    public static void main(String[] args) {
        int[] array = {3, 7, 4, 8, 23, 56, 77, 89, 46, 11, 66, 77};
```

```java
        mysort(array);
        for (int a : array) {

            System.out.println(" " + a);
        }
    }

    private static void mysort(int[] array) {
        if (array == null) {
            return;
        }
        int left = 0;
        int right = array.length - 1;
        while (left < right) {
            while (left < right && !isEven(array[left])) {
                left++;
            }
            while (left < right && isEven(array[right])) {
                right--;
            }
            if (left < right) {
                int temp = array[right];
                array[right] = array[left];
                array[left] = temp;
            }
            if (left >= right) {
                break;
            }
        }
    }

    private static boolean isEven(int i) {
        return (i & 0x1) == 0;
    }

}
```

> 15. 输入一个链表，输出该链表中倒数第K个结点。为了符合大多数人的习惯，本题从1开始计数，即链表的尾结点是倒数第1个结点。例如一个链表有6个结点，从头结点开始它们的值依次是1,2,3,4,5,6。这个链表的倒数第3个结点是值为4的结点。（注意代码鲁棒性，考虑输入空指针，链表结点总数少于k，输入的k参数为0）

```java
public class No15 {

    public static void main(String[] args) {
        Node a = new Node("1");
        Node b = new Node("2");
        Node c = new Node("3");
        Node d = new Node("4");
        Node e = new Node("5");
        Node f = new Node("6");
        a.setNext(b);
        b.setNext(c);
        c.setNext(d);
        d.setNext(e);
        e.setNext(f);

        System.out.print(FindDataFromTail(a, 5));
    }

    private static String FindDataFromTail(Node a, int k) {

        if (a == null)
            return null;
        if (k == 0) {
            System.out.println("k应该从1开始");
            return null;
        }
        Node node1 = a;
        Node node2 = null;
        for (int i = 0; i < k - 1; i++) {
            if (node1.getNext() == null) {
                System.out.println("k不应该大于链表长度");
                return null;
            }
            node1 = node1.getNext();
        }
        node2 = a;

        while (node1.getNext() != null) {
            node1 = node1.getNext();
            node2 = node2.getNext();
        }
        return node2.getData();

    }
}
```

> 15_2. 求链表的中间结点。如果链表中结点总数为奇数，返回中间结点；如果结点总数为偶数，返回中间两个结点的任意一个

```java
public class No15_2 {
```

```
public class No15_2 {
    public static void main(String[] args) {
        Node a = new Node("1");
        Node b = new Node("2");
        Node c = new Node("3");
        Node d = new Node("4");
        Node e = new Node("5");
        Node f = new Node("6");
        Node g = new Node("7");

        a.setNext(b);
        b.setNext(c);
        c.setNext(d);
        d.setNext(e);
        e.setNext(f);
        f.setNext(g);
        Node mid = getMid(a);
        System.out.println(mid.getData());
    }

    private static Node getMid(Node a) {

        if (a == null) {
            return null;
        }
        Node slow = a;
        Node fast = a;
        while (fast.getNext() != null && fast.getNext().getNext()
!= null) {
            slow = slow.getNext();
            fast = fast.getNext().getNext();
        }

        return slow;
    }

}
```

> 16. 同第5题

> 17. 输入两个递增排序的链表，合并这两个链表并使新链表中结点仍然是按照递增排序的。例如输入1->3->5->7和2->4->6->8，则合并之后的升序链表应该是1->2->3->4->5->6->7->8 。

```java
public class No17 {

    public static void main(String[] args) {
        Node node1 = new Node(1);
        Node node2 = new Node(3);
        Node node3 = new Node(5);
        Node node4 = new Node(7);
        node1.setNext(node2);
        node2.setNext(node3);
        node3.setNext(node4);
        Node node5 = new Node(2);
        Node node6 = new Node(4);
        Node node7 = new Node(6);

        node5.setNext(node6);
        node6.setNext(node7);
        Node head = merge(node1, node5);
        while (head != null) {
            System.out.print(head.getData() + " ");
            head = head.getNext();
        }

    }

    private static Node merge(Node a, Node b) {
        if (a == null && b == null)
            return null;
        if (a == null)
            return b;
        if (b == null)
            return a;

        Node head = a.getData() > b.getData() ? b : a;
        Node index1 = head.getNext();
        Node index2 = head == a ? b : a;

        while (index1 != null && index2 != null) {
            if (index1.getData() < index2.getData()) {
                head.setNext(index1);
                index1 = index1.getNext();
            } else {
                head.setNext(index2);
                index2 = index2.getNext();
            }
            head = head.getNext();
        }
        if (index1 == null) {

            while (index2 != null) {
                head.setNext(index2);
                index2 = index2.getNext();
                head = head.getNext();
            }
        } else {
            while (index1 != null) {
                head.setNext(index1);
                index1 = index1.getNext();
                head = head.getNext();
            }
        }
        return a.getData() > b.getData() ? b : a;
    }
}
```

> 18. 输入两颗二叉树A和B，判断B是不是A的子结构

```
public class No18 {

    public static void main(String[] args) {

        BinaryTreeNode node1 = new BinaryTreeNode(8);
        BinaryTreeNode node2 = new BinaryTreeNode(8);
        BinaryTreeNode node3 = new BinaryTreeNode(7);
        BinaryTreeNode node4 = new BinaryTreeNode(9);
        BinaryTreeNode node5 = new BinaryTreeNode(2);
        BinaryTreeNode node6 = new BinaryTreeNode(4);
        BinaryTreeNode node7 = new BinaryTreeNode(7);
        node1.setLchildNode(node2);
        node1.setRchildNode(node3);
        node2.setLchildNode(node4);
        node2.setRchildNode(node5);
        node5.setLchildNode(node6);
        node5.setRchildNode(node7);

        BinaryTreeNode a = new BinaryTreeNode(8);
        BinaryTreeNode b = new BinaryTreeNode(9);
        BinaryTreeNode c = new BinaryTreeNode(2);
        a.setLchildNode(b);
        a.setRchildNode(c);
        System.out.println(hasSubTree(node1, a));
    }

    private static boolean hasSubTree(BinaryTreeNode root1,
BinaryTreeNode root2) {
```

```
BinaryTreeNode root2) {
        boolean result = false;
        if (root1 != null && root2 != null) {
            if (root1.getData() == root2.getData()) {
                result = doseTree1HaveTree2(root1, root2);
                if (!result) {
                    result = hasSubTree(root1.getLchildNode(),
root2);
                }
                if (!result)
                    result = hasSubTree(root1.getRchildNode(),
root2);
            }
        }
        return result;

    }

    private static boolean doseTree1HaveTree2(BinaryTreeNode root1,
                                              BinaryTreeNode root2)
{
        if (root2 == null)
            return true;
        if (root1 == null)
            return false;
        if (root1.getData() != root2.getData()) {
            return false;
        }

        return doseTree1HaveTree2(root1.getLchildNode(),
root2.getLchildNode())
                && doseTree1HaveTree2(root1.getRchildNode(),
root2.getRchildNode());
    }

}
```

> 19. 请完成一个函数，输入一个二叉树，该函数输出它的镜像

```
public class No19 {
    public static void main(String[] args) {
        BinaryTreeNode node1 = new BinaryTreeNode(8);
        BinaryTreeNode node2 = new BinaryTreeNode(6);
        BinaryTreeNode node3 = new BinaryTreeNode(10);
        BinaryTreeNode node4 = new BinaryTreeNode(5);
        BinaryTreeNode node5 = new BinaryTreeNode(7);
        BinaryTreeNode node6 = new BinaryTreeNode(9);
        BinaryTreeNode node7 = new BinaryTreeNode(11);
        node1.setLchildNode(node2);
        node1.setRchildNode(node3);
        node2.setLchildNode(node4);
        node2.setRchildNode(node5);
        node3.setLchildNode(node6);
        node3.setRchildNode(node7);
        mirror(node1);
        print(node1);
    }

    private static void print(BinaryTreeNode root) {
        if (root != null) {
            System.out.println(root.getData());
            print(root.getLchildNode());
            print(root.getRchildNode());
        }
    }

    private static void mirror(BinaryTreeNode root) {
        if (root == null) {
            return;
        }
        if (root.getLchildNode() == null && root.getRchildNode() ==
null) {
            return;
        }
        BinaryTreeNode temp = root.getLchildNode();
        root.setLchildNode(root.getRchildNode());
        root.setRchildNode(temp);
        mirror(root.getLchildNode());
        mirror(root.getRchildNode());
    }

}
```

> 20. 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

```
public class No20 {


    public static void main(String[] args) {
        int[][] a = create(5, 5);
        int[][] a = create(5, 5);
        print(a);
        clockWisePrint(a, 0, 4);
    }

    private static void clockWisePrint(int[][] a, int i, int j) {
        if (j < i)
            return;
        if (j == i) {
            System.out.print(a[i][j] + " ");
            return;
        }
        int y = i;
        while (y <= j) {
            System.out.print(a[i][y] + " ");
            y++;
        }
        y = i + 1;
        while (y <= j) {
            System.out.print(a[y][j] + " ");
            y++;
        }
        y = j - 1;
        while (y >= i) {
            System.out.print(a[j][y] + " ");
            y--;
        }

        y = j - 1;
        while (y >= i + 1) {
            System.out.print(a[y][i] + " ");
            y--;
        }


        clockWisePrint(a, i + 1, j - 1);

    }

    private static void print(int[][] a) {
        for (int i = 0; i < a.length; i++) {
            for (int j = 0; j < a[i].length; j++) {
                System.out.print(a[i][j] + " ");
            }
            System.out.println();
        }
    }

    public static int[][] create(int row, int line) {
        int[][] a = new int[row][line];
        int num = 1;

        for (int i = 0; i < row; i++) {
            for (int j = 0; j < line; j++) {
                a[i][j] = num++;
            }
        }
        return a;
    }

}
```

> 21. 定义栈的数据结构，请在该类型中实现一个能够得到栈的最小元素的min函数。在该栈中，调用min、push以及pop的时间复杂度都是O(1)。

```
import java.util.Stack;

public class No21 {
```

```java
    public static void main(String[] args) {
        MyStack a = new MyStack();

        System.out.println(a.min());
        a.push(10);
        a.push(11);
        System.out.println(a.min());
    }

}

class MyStack {
    private Stack<Integer> stack1, stackHelp;

    public MyStack() {
        stack1 = new Stack<Integer>();
        stackHelp = new Stack<Integer>();
    }

    public void push(int num) {
        stack1.push(num);
        if (stackHelp.size() == 0 || num < stackHelp.peek()) {
            stackHelp.push(num);
        } else {
            stackHelp.push(stackHelp.peek());
        }
    }

    public void pop() {
        stack1.pop();
        stackHelp.pop();

    }

    public Integer min() {
        if (stackHelp.size() == 0) {
            return null;
        }
        return stackHelp.peek();
    }

}
```

> 22. 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是某栈的压栈序列，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。

```
import java.util.Stack;

public class No22 {

    public static void main(String[] args) {
        Integer[] pushOrder = {1, 2, 3, 4, 5};
        Integer[] popOrder = {4, 5, 3, 1, 2};
        System.out.println(isRight(pushOrder, popOrder));
    }

    private static boolean isRight(Integer[] pushOrder, Integer[]
popOrder) {

        Stack<Integer> stack = new Stack<Integer>();
        int pushIndex = 0;
        for (int i = 0; i < popOrder.length; i++) {
            if (!stack.isEmpty() && stack.peek() == popOrder[i])
                stack.pop();
            else {
                while (pushIndex <= pushOrder.length - 1 &&
pushOrder[pushIndex] != popOrder[i]) {
                    stack.push(pushOrder[pushIndex++]);
                }
                if (pushIndex > pushOrder.length - 1) {
                    return false;
                } else {
                    pushIndex++;
                }
            }
        }
        return true;
    }

}
```

> 23. 从上往下打印出二叉树的每个结点，同一层的结点按照从左到右的顺序打印

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

public class No23 {
    public static void main(String[] args) {
        BinaryTreeNode node1 = new BinaryTreeNode(8);
        BinaryTreeNode node2 = new BinaryTreeNode(6);
        BinaryTreeNode node3 = new BinaryTreeNode(10);
        BinaryTreeNode node4 = new BinaryTreeNode(5);
        BinaryTreeNode node5 = new BinaryTreeNode(7);
        BinaryTreeNode node6 = new BinaryTreeNode(9);
        BinaryTreeNode node7 = new BinaryTreeNode(11);
        node1.setLchildNode(node2);
        node1.setRchildNode(node3);
        node2.setLchildNode(node4);
        node2.setRchildNode(node5);
        node3.setLchildNode(node6);
        node3.setRchildNode(node7);

        printFromTopToBottom(node1);
    }

    private static void printFromTopToBottom(BinaryTreeNode root) {
        if (root == null)
            return;
        Queue<BinaryTreeNode> queue = new
LinkedList<BinaryTreeNode>();
        queue.add(root);
        while (!queue.isEmpty()) {
            BinaryTreeNode node = queue.poll();
            System.out.println(node.getData());
            if (node.getLchildNode() != null) {
                queue.add(node.getLchildNode());
            }
            if (node.getRchildNode() != null) {
                queue.add(node.getRchildNode());
            }
        }
    }

}
```

> 24. 输入一个整数数组，判断该数组是不是某二叉搜索树的后序遍历的结果。如果是则返回true，否则返回false。假设输入的数组的任意两个数字都互不相同。

```
public class No24 {

    public static void main(String[] args) {
        int[] array = {5, 7, 6, 9, 11, 10, 8};
        // int[] array={7,4,6,5};

        boolean b = verfiySequenceOfBST(array, 0, 6);
        System.out.println(b);
    }

    private static boolean verfiySequenceOfBST(int[] array, int
start, int end) {

        if (array == null || start > end || start < 0 || end < 0)
            return false;

        if (start == end)
            return true;

        int root = array[end];

        int i = start;
        for (; i <= end; i++) {
            if (array[i] > root)
                break;
        }

        int j = i;
        for (; j <= end; j++) {
            if (array[j] < root)
                return false;
        }

        boolean left = true;
        if (i > start) {
            left = verfiySequenceOfBST(array, start, i - 1);
        }

        boolean right = true;
        if (i < end) {

            right = verfiySequenceOfBST(array, i, end - 1);
        }
        return (left && right);
    }

}
```

> 25. 输入一颗二叉树和一个整数，打印出二叉树中结点值的和为输入整数的所有路径。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。

```
import java.util.Stack;

public class No25 {

    public static void main(String[] args) {

        BinaryTreeNode root = new BinaryTreeNode(10);
        BinaryTreeNode node1 = new BinaryTreeNode(5);
        BinaryTreeNode node2 = new BinaryTreeNode(4);
        BinaryTreeNode node3 = new BinaryTreeNode(7);
        BinaryTreeNode node4 = new BinaryTreeNode(12);
        root.setLchildNode(node1);
        root.setRchildNode(node4);
        node1.setLchildNode(node2);
        node1.setRchildNode(node3);
        findPath(root, 22);
    }

    private static void findPath(BinaryTreeNode root, int i) {
        if (root == null)
            return;
        Stack<Integer> stack = new Stack<Integer>();
        int currentSum = 0;
        findPath(root, i, stack, currentSum);
    }

    private static void findPath(BinaryTreeNode root, int i,
                                 Stack<Integer> stack, int
currentSum) {
        currentSum += root.getData();
        stack.push(root.getData());
        if (root.getLchildNode() == null && root.getRchildNode() ==
null) {
            if (currentSum == i) {
                System.out.println("找到路径");
                for (int path : stack) {
                    System.out.println(path + " ");
                }
            }
        }
        if (root.getLchildNode() != null) {
            findPath(root.getLchildNode(), i, stack, currentSum);
        }
        if (root.getRchildNode() != null) {
            findPath(root.getRchildNode(), i, stack, currentSum);
        }
```

```
            stack.pop();
        }

}
```

> 26. 请实现函数ComplexListNode* Clone(ComplexListNode* pHead)，复制一个复杂链表。在复杂链表中，每个结点除了有一个m_pNext指针指向下一个结点外，还有一个m_pSibling指向链表中的任意结点或者NULL。

```
public class No26 {

    public static void main(String[] args) {
        ComplexListNode node1 = new ComplexListNode(1);
        ComplexListNode node2 = new ComplexListNode(2);
        ComplexListNode node3 = new ComplexListNode(3);
        ComplexListNode node4 = new ComplexListNode(4);
        ComplexListNode node5 = new ComplexListNode(5);
        node1.next = node2;
        node2.next = node3;
        node3.next = node4;
        node4.next = node5;
        node1.sibling = node3;
        node2.sibling = node5;
        node4.sibling = node2;
        ComplexListNode result = clone(node1);
        while (result != null) {
            System.out.println(result.data);
            result = result.next;
        }
    }

    private static ComplexListNode clone(ComplexListNode head) {
        cloneNodes(head);
        copySibingNodes(head);
        return separateNodes(head);
    }

    private static ComplexListNode separateNodes(ComplexListNode
head) {
        ComplexListNode node = head;
        ComplexListNode cloneHead = null;
        ComplexListNode cloneNode = null;
        if (node != null) {
            cloneNode = node.next;
            cloneHead = cloneNode;
```

```java
            node.next = cloneNode.next;
            node = node.next;

        }
        while (node != null) {
            cloneNode.next = node.next;
            cloneNode = cloneNode.next;
            node.next = cloneNode.next;
            node = node.next;
        }
        return cloneHead;
    }

    private static void copySibingNodes(ComplexListNode head) {
        ComplexListNode node = head;
        while (node != null) {
            ComplexListNode cloneNode = node.next;

            if (node.sibling != null) {
                cloneNode.sibling = node.sibling.next;
            }
            node = cloneNode.next;
        }

    }

    private static void cloneNodes(ComplexListNode head) {
        ComplexListNode node = head;
        while (node != null) {
            ComplexListNode cloneNode = new
ComplexListNode(node.data);
            cloneNode.next = node.next;
            node.next = cloneNode;
            node = cloneNode.next;
        }
    }

}

class ComplexListNode {
    int data;
    ComplexListNode next;
    ComplexListNode sibling;

    public ComplexListNode(int data) {
        super();
        this.data = data;
    }
}
```

> 27. 输入一颗二叉搜索树，将该二叉搜索树转换成一个排序的双向链表。要求不能创建任何新的结点，只能调整树中结点指针的指向。

```
public class No27 {

    public static void main(String[] args) {
        BinaryTreeNode root = new BinaryTreeNode(10);
        BinaryTreeNode node1 = new BinaryTreeNode(6);
        BinaryTreeNode node2 = new BinaryTreeNode(14);
        BinaryTreeNode node3 = new BinaryTreeNode(4);
        BinaryTreeNode node4 = new BinaryTreeNode(8);
        BinaryTreeNode node5 = new BinaryTreeNode(12);
        BinaryTreeNode node6 = new BinaryTreeNode(16);
        root.setLchildNode(node1);
        root.setRchildNode(node2);
        node1.setLchildNode(node3);
        node1.setRchildNode(node4);
        node2.setLchildNode(node5);
        node2.setRchildNode(node6);

        BinaryTreeNode head = covert(root);

        while (head != null) {
            System.out.println(head.getData());
            head = head.getRchildNode();
        }
    }

    private static BinaryTreeNode covert(BinaryTreeNode root) {
        BinaryTreeNode lastNodeList = null;
        lastNodeList = convertNode(root, lastNodeList);
        while (lastNodeList != null && lastNodeList.getLchildNode()
!= null) {
            lastNodeList = lastNodeList.getLchildNode();
        }
        return lastNodeList;
    }

    private static BinaryTreeNode convertNode(BinaryTreeNode root,
                                              BinaryTreeNode
lastNodeList) {
        if (root == null)
            return null;
        BinaryTreeNode current = root;
        if (current.getLchildNode() != null) {
            lastNodeList = convertNode(current.getLchildNode(),

lastNodeList);
        }

        current.setLchildNode(lastNodeList);

        if (lastNodeList != null) {
            lastNodeList.setRchildNode(current);
        }
        lastNodeList = current;
        if (current.getRchildNode() != null) {
            lastNodeList = convertNode(current.getRchildNode(),
lastNodeList);
        }
        return lastNodeList;
    }

}
```

> 28. 输入一个字符串，打印出该字符串中字符的所有排列。例如输入字符串abc， 则打印出由字符串a、b、c所能排列出来的所有字符串abc、acb、bac、bca、cab和cba

```
public class No28 {

    public static void main(String[] args) {
        myPrint("abc");
    }

    private static void myPrint(String str) {
        if (str == null)
            return;
        char[] chs = str.toCharArray();
        myPrint(chs, 0);
    }

    private static void myPrint(char[] str, int i) {
        if (i >= str.length)
            return;
        if (i == str.length - 1) {
            System.out.println(String.valueOf(str));
        } else {
            for (int j = i; j < str.length; j++) {
                char temp = str[j];
                str[j] = str[i];
                str[i] = temp;

                myPrint(str, i + 1);

                temp = str[j];
                str[j] = str[i];
                str[i] = temp;
            }
        }
    }

}
```

> 29. 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。

```
public class No29 {

    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 2, 2, 2, 5, 4, 2};
```

```java
        System.out.println(findNum(arr));
    }

    private static Integer findNum(int[] arr) {
        if (arr == null)
            return null;

        int result = arr[0];
        int count = 1;

        for (int i = 1; i < arr.length; i++) {
            if (count == 0) {
                result = arr[i];
                count = 1;
            } else if (arr[i] == result) {
                count++;
            } else {
                count--;
            }
        }
        return result;
    }
}
```

> 30. 题目：输入n个整数，输出其中最小的k个。例如输入1，2，3，4，5，6，7和8这8个数字，则最小的4个数字为1，2，3和4。

```java
public class TopK {

    public static void main(String[] args) {
        int[] arr = {1, 3, 4, 2, 7, 8, 9, 10, 14, 16};
        System.out.println(minK(arr, 1));
        System.out.println(minK(arr, 2));
        System.out.println(minK(arr, 3));
        System.out.println(minK(arr, 4));
        System.out.println(minK(arr, 5));
        System.out.println(minK(arr, 6));
    }

    public static int minK(int[] arr, int k) {
        return minK(arr, k, 0, arr.length - 1);
    }

    public static int minK(int[] arr, int k, int start, int end) {
        int mid = partition(arr, start, end);
        if (mid - start == k - 1) {
            return arr[mid];
        } else if (mid - start > k - 1) {

            return minK(arr, k, start, mid - 1);
        } else {
            return minK(arr, k - 1 - (mid - start), mid + 1, end);
        }
    }


    public static int partition(int[] arr, int start, int end) {
        int key = arr[start];
        int keyIndex = start;
        start++;
        for (int i = start; i <= end; i++) {
            if (arr[i] < key) {
                swap(arr, i, start);
                start++;
            }
        }
        swap(arr, keyIndex, start - 1);
        return start - 1;
    }


    public static void swap(int[] arr, int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }

}
```

> 31. 输入一个整型数组，数组里有正数，也有负数。数组中一个或连续的多个整数组成一个子数组。求所有子数组的和的最大值。要求时间复杂度为O(n)

```
public class No31 {

    public static void main(String[] args) {
        int[] arr = {1, -2, 3, 10, -4, 7, 2, -5};
        System.out.println(maxSub(arr));
    }

    private static int maxSub(int[] arr) {
        int max = 0;
        int n = arr.length;
        int sum = 0;
        for (int i = 0; i < n; i++) {
            sum += arr[i];
            if (sum > max)
                max = sum;
            else if (sum < 0)
                sum = 0;
        }
        return max;

    }

}
```

> 32. 输入一个整数n，求从1到n这n个整数的十进制表示中1出现的次数。例如输入12，从1到12这些整数中包含1的数字有1,10,11和12,1一共出现5次。

```
public class No32 {

    public static void main(String[] args) {
        System.out.println(countOne(115));
    }

    private static long countOne(int n) {
        long count = 0;
        long i = 1;
        long current = 0, after = 0, before = 0;
        while ((n / i) != 0) {
            current = (n / i) % 10;
            before = n / (i * 10);
            after = n - (n / i) * i;

            if (current > 1)
                count = count + (before + 1) * i;
            else if (current == 0)
                count = count + before * i;
            else if (current == 1)
                count = count + before * i + after + 1;
            i = i * 10;
        }
        return count;
    }

}
```

> 33. 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，打印能拼接出所有数字中最小的一个。例如输入数组{3,32,321}，则打印出这3个数字能排成的最小数字321323

```
public class No33 {

    public static void main(String[] args) {
        int[] array = {321, 32, 3};
        printMin(array);
    }

    private static void printMin(int[] array) {
        int[] clone = array.clone();
        printMin(clone, 0, clone.length - 1);
        for (int i : clone)
            System.out.print(i);
    }

    private static void printMin(int[] array, int start, int end) {

        if (start < end) {
            int main_number = array[end];
            int small_cur = start;

            for (int j = start; j < end; j++) {
                if (isSmall(String.valueOf(array[j]),
String.valueOf(main_number))) {
                    int temp = array[j];
                    array[j] = array[small_cur];
                    array[small_cur] = temp;
                    small_cur++;
                }
            }
            array[end] = array[small_cur];
            array[small_cur] = main_number;
            printMin(array, 0, small_cur - 1);
            printMin(array, small_cur + 1, end);
        }

    }

    public static boolean isSmall(String m, String n) {
        String left = m + n;
        String right = n + m;
        boolean result = false;
        for (int i = 0; i < left.length(); i++) {
            if (left.charAt(i) < right.charAt(i))
                return true;
            else if (left.charAt(i) > right.charAt(i))
                return false;
        }

        return result;
    }

}
```

> 34. 我们把只包含因子2,3和5的数称作丑数。求按从小到大的顺序的第1500个丑数。例如6、8都是丑数，但14不是，因为它包含因子7.习惯上我们把1当做第一个丑数。

```
public class No34 {

    public static void main(String[] args) {
        System.out.println(getUgly(20));

    }

    private static int getUgly(int n) {
        if (n < 0)
            return 0;
        int[] uglyArray = new int[n];
        uglyArray[0] = 1;
        int multiply2 = 0;
        int multiply3 = 0;
        int multiply5 = 0;
        for (int i = 1; i < n; i++) {
            int min = getMin(uglyArray[multiply2] * 2,
uglyArray[multiply3] * 3, uglyArray[multiply5] * 5);
            uglyArray[i] = min;
            System.out.println(uglyArray[i]);
            while (uglyArray[multiply2] * 2 == uglyArray[i])
                multiply2++;
            while (uglyArray[multiply3] * 3 == uglyArray[i])
                multiply3++;
            while (uglyArray[multiply5] * 5 == uglyArray[i])
                multiply5++;
        }
        return uglyArray[n - 1];
    }

    private static int getMin(int i, int j, int k) {
        int min = (i < j) ? i : j;
        return (min < k) ? min : k;
    }

}
```

> 35. 在字符串中找出第一个只出现一次的字符。如输入"abaccdeff"，则输出'b'

```
import java.util.LinkedHashMap;

public class No35 {
    public static void main(String[] args) {
        System.out.println(firstOnceNumber("abaccdeff"));

    }

    private static Character firstOnceNumber(String str) {
        if (str == null)
            return null;
        char[] strChar = str.toCharArray();
        LinkedHashMap<Character, Integer> hash = new
LinkedHashMap<Character, Integer>();
        for (char item : strChar) {
            if (hash.containsKey(item))
                hash.put(item, hash.get(item) + 1);
            else
                hash.put(item, 1);
        }
        for (char key : hash.keySet()) {
            if (hash.get(key) == 1) {
                return key;
            }
        }
        return null;

    }

}
```

> 36. 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

```
public class No36 {

    public static void main(String[] args) {
        int[] arr = {7, 5, 6, 4};
        System.out.println(getInversePairs(arr));

    }

    private static int getInversePairs(int[] arr) {
        if (arr == null)
            return 0;
        int[] clone = arr.clone();
        return mergeSort(arr, clone, 0, arr.length - 1);
    }
    private static int mergeSort(int[] array, int[] result, int
start, int end) {

        if (start == end) {
            result[start] = array[start];
            return 0;
        }
        int length = (end - start) / 2;
        int left = mergeSort(result, array, start, start + length);
        int right = mergeSort(result, array, start + length + 1,
end);
        int leftIndex = start + length;
        int rightIndex = end;
        int count = 0;
        int point = rightIndex;
        while (leftIndex >= start && rightIndex >= start + length +
1) {
            if (array[leftIndex] > array[rightIndex]) {
                result[point--] = array[leftIndex--];
                count += rightIndex - start - length;

            } else {
                result[point--] = array[rightIndex--];
            }
        }
        for (int i = leftIndex; i >= start; i--)
            result[point--] = array[i];
        for (int j = rightIndex; j >= start + length + 1; j--)
            result[point--] = array[j];
        return left + right + count;

    }

}
```

> 37. 输入两个单向链表，找出它们的第一个公共结点。

```
public class No37 {

    public static void main(String[] args) {

        ListNode head1 = new ListNode();
        ListNode second1 = new ListNode();
        ListNode third1 = new ListNode();
        ListNode forth1 = new ListNode();
        ListNode fifth1 = new ListNode();
        ListNode head2 = new ListNode();
        ListNode second2 = new ListNode();
        ListNode second2 = new ListNode();
        ListNode third2 = new ListNode();
        ListNode forth2 = new ListNode();
        head1.nextNode = second1;
        second1.nextNode = third1;
        third1.nextNode = forth1;
        forth1.nextNode = fifth1;
        head2.nextNode = second2;
        second2.nextNode = forth1;
        third2.nextNode = fifth1;
        head1.data = 1;
        second1.data = 2;
        third1.data = 3;
        forth1.data = 6;
        fifth1.data = 7;
        head2.data = 4;
        second2.data = 5;
        third2.data = 6;
        forth2.data = 7;
        System.out.println(findFirstCommonNode(head1, head2).data);

    }

    public static ListNode findFirstCommonNode(ListNode head1,
ListNode head2) {
        int len1 = getListLength(head1);
        int len2 = getListLength(head2);
        ListNode longListNode = null;
        ListNode shortListNode = null;
        int dif = 0;
        if (len1 > len2) {
            longListNode = head1;
            shortListNode = head2;
            dif = len1 - len2;
        } else {
            longListNode = head2;
            shortListNode = head1;
            dif = len2 - len1;
        }
        for (int i = 0; i < dif; i++) {
            longListNode = longListNode.nextNode;
        }
        while (longListNode != null && shortListNode != null
                && longListNode != shortListNode) {
            longListNode = longListNode.nextNode;
            shortListNode = shortListNode.nextNode;
        }
        return longListNode;
    }
    private static int getListLength(ListNode head1) {

        int result = 0;
        if (head1 == null)
            return result;
        ListNode point = head1;
        while (point != null) {
            point = point.nextNode;
            result++;
        }
        return result;
    }
}

class ListNode {
    int data;
    ListNode nextNode;
}
```

38. 统计一个数字在排序数组中出现的次数。例如输入排序数组{1,2,3,3,3,3,4,5}和数字3，由于3在这个数组中出现了4次，因此输出4。

```
public class No38 {

    public static void main(String[] args) {
        int[] array = {1, 2, 3, 3, 3, 3, 4, 5};
        System.out.println(getNumberOfK(array, 3));
    }

    private static int getNumberOfK(int[] array, int k) {
        int num = 0;
        if (array != null) {
            int first = getFirstK(array, k, 0, array.length - 1);
            int last = getLastK(array, k, 0, array.length - 1);
            //System.out.println(last);

            if (first > -1 && last > -1)
                num = last - first + 1;
        }
        return num;
    }

    private 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);
    }

    private 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);
    }

}
```

> 39. 输入一颗二叉树的根节点，求该树的深度。从根节点到叶节点依次经过的结点（含根、叶结点）形成树的一条路径，最长路径的长度为树的深度。

```
public class No39 {
```

```
public class No39 {

    public int treeDepth(BinaryTreeNode root) {

        if (root == null) return 0;

        int left = treeDepth(root.getLchildNode());

        int right = treeDepth(root.getRchildNode());

        return (left > right) ? left + 1 : right + 1;

    }

}
```

> 40. 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。要求时间复杂度是O(n)，空间复杂度O(1)

```
public class No40 {

    public static void main(String[] args) {
        int[] array = {2, 4, 3, 6, 3, 2, 5, 5};
        findNumsAppearOnce(array);
    }

    private static void findNumsAppearOnce(int[] array) {
        if (array == null)
            return;
        int num = 0;
        for (int i : array) {
            num ^= i;
        }
        int index = findFirstBitIs1(num);
        int number1 = 0;
        int number2 = 0;
        for (int i : array) {
            if (isBit1(i, index))
                number1 ^= i;
            else
                number2 ^= i;
        }
        System.out.println(number1);
        System.out.println(number2);
    }

    private static boolean isBit1(int number, int index) {
        number = number >> index;
        return (number & 1) == 0;
    }

    private static int findFirstBitIs1(int num) {
        int index = 0;
        while ((num & 1) == 0) {
            num = num >> 1;
            index++;
        }
        return index;
    }

}
```

> 41. 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，使得它们的和正好是s。如果有多对数字的和等于s，输出任意一对即可。

```
public class No41 {

    public static void main(String[] args) {
        int[] data = {1, 2, 4, 7, 11, 15};
```

```java
        System.out.println(findNumberWithSum(data, 15));
    }

    private static boolean findNumberWithSum(int[] data, int sum) {
        boolean found = false;
        if (data == null)
            return found;
        int num1 = 0;
        int num2 = 0;
        int start = 0;
        int end = data.length - 1;
        while (start < end) {
            int curSum = data[start] + data[end];
            if (curSum == sum) {
                num1 = data[start];
                num2 = data[end];
                found = true;
                break;
            } else if (curSum > sum)
                end--;
            else
                start++;
        }
        System.out.println(num1);
        System.out.println(num2);

        return found;
    }

}
```

> 42. 输入一个英文句子，翻转句子中单词的顺序，但单词内字符的顺序不变。为简单起见，标点符号和普通字母一样处理。例如输入字符串"I am student."，则输出"student. a am I"。

```java
public class No42 {

    public static void main(String[] args) {
        String string = "I am a student.";
        reverseSentence(string);
    }

    private static void reverseSentence(String str) {
        if (str == null)
            return;
        char[] arr = str.toCharArray();

        reverse(arr, 0, arr.length - 1);
        int start = 0;
        int end = 0;
        for (char i = 0; i < arr.length; i++) {
            if (arr[i] == ' ') {
                reverse(arr, start, end);
                end++;
                start = end;
            } else if (i == arr.length) {
                end++;
                reverse(arr, start, end);
            } else {
                end++;
            }
        }

        for (char c : arr) {
            System.out.print(c);
        }
    }

    private static void reverse(char[] arr, int start, int end) {
        for (int i = start, j = end; i <= j; i++, j--) {
            char temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }

}
```

> 43. 把n个骰子仍在地上，所有骰子朝上一面的点数之和为s，输入n，打印出s的所有可能的值出现的概率

```
public class No43 {

    public static void main(String[] args) {
        printProbability(2);
    }

    private static void printProbability(int num) {
        if (num < 1)
            return;
        int gMaxValue = 6;
        int[][] probabilities = new int[2][];
        probabilities[0] = new int[gMaxValue * num + 1];
        probabilities[1] = new int[gMaxValue * num + 1];
        int flag = 0;
        for (int i = 1; i <= gMaxValue; i++) {
            probabilities[flag][i] = 1;
        }
        for (int k = 2; k <= num; k++) {
            for (int i = 0; i < k; i++) {
                probabilities[1 - flag][i] = 0;
            }
            for (int i = k; i <= gMaxValue * k; i++) {
                probabilities[1 - flag][i] = 0;
                for (int j = 1; j <= i && j <= gMaxValue; j++)
                    probabilities[1 - flag][i] +=
probabilities[flag][i - j];
            }
            flag = 1 - flag;

        }
        double total = Math.pow(gMaxValue, num);
        for (int i = num; i <= gMaxValue * num; i++) {

            double ratio = (double) probabilities[flag][i] / total;

            System.out.print(i + " ");

            System.out.println(ratio);

        }
    }
}
```

> 44.从扑克牌中随机抽5张牌，判断是不是一个顺子，即这5张牌是不是连续的。2~10为数字本身，A为1，J为11，Q为12，K为13，而大、小王可以看成任意数字。

```java
import java.util.Arrays;


public class No44 {

    public static void main(String[] args) {
        int[] array = {0, 4, 6, 8, 0};
        System.out.println(isContinuous(array));

    }

    private static boolean isContinuous(int[] arr) {

        if (arr == null || arr.length != 5)
            return false;
        Arrays.sort(arr);
        int numberZero = 0;
        int numberGap = 0;
        for (int i = 0; i < arr.length && arr[i] == 0; i++)
            numberZero++;

        int small = numberZero;
        int big = small + 1;
        while (big < arr.length) {
            if (arr[small] == arr[big])
                return false;

            numberGap += arr[big] - arr[small] - 1;
            small = big;
            big++;
        }
        return numberGap <= numberZero;
    }

}
```

> 45. 0~n-1这n个数字排列成一个圆圈，从数字0开始每次从这个圆圈中删除第m个数字。求出这个圆圈里剩下的最后一个数字。

```java
public class No45 {

    public static void main(String[] args) {
        System.out.println(lastRemaining(6, 3));
    }


    public static int lastRemaining(int n, int m) {
        if (n < 1 || m < 1)
            return -1;
        int last = 0;
        for (int i = 2; i <= n; i++) {
            last = (last + m) % i;
        }
        return last;
    }

}
```

> 47. 写一个函数，求两个整数之和，要求函数体内部不能使用"+"、"-"、"*"、""。

```
public class No47 {

    public static void main(String[] args) {
        System.out.println(add(5, 8));
    }

    private static int add(int num1, int num2) {
        int sum, carry;
        do {
            sum = num1 ^ num2;
            carry = (num1 & num2) << 1;
            num1 = sum;
            num2 = carry;
        } while (num2 != 0);
        return num1;
    }

}
```