程序设计比赛准备

> 加油加油冲冲冲！！！

注意：数据范围，Long，越界

快读快写板

```
import java.io.*;
import java.math.BigDecimal;
import java.math.BigInteger;
import java.util.StringTokenizer;


public class FastReadAndWrite {

    public static void main(String[] args) {
        InputStream inputStream = System.in;
        OutputStream outputStream = System.out;
        InputReader in = new InputReader(inputStream);
        PrintWriter out = new PrintWriter(outputStream);
        Task solver = new Task();
        solver.solve(in, out);
        out.close();
    }

    static class Task {

        public void solve(InputReader in, PrintWriter out) {


        }
    }



    static class InputReader {
        public BufferedReader reader;
        public StringTokenizer tokenizer;

        public InputReader(InputStream stream) {
            reader = new BufferedReader(new InputStreamReader(stream),
32768);
            tokenizer = null;
        }

        public String next() {
            while (tokenizer == null || !tokenizer.hasMoreTokens()) {
                try {
                    tokenizer = new StringTokenizer(reader.readLine());
                } catch (IOException e) {
                    throw new RuntimeException(e);
                }
            }
```

```
            return tokenizer.nextToken();
        }

        public int nextInt() {
            return Integer.parseInt(next());
        }

        public long nextLong() {
            return Long.parseLong(next());
        }

        public double nextDouble() {
            return Double.parseDouble(next());
        }

        public char[] nextCharArray() {
            return next().toCharArray();
        }

        public boolean hasNext() {
            try {
                String string = reader.readLine();
                if (string == null) {
                    return false;
                }
                tokenizer = new StringTokenizer(string);
                return tokenizer.hasMoreTokens();
            } catch (IOException e) {
                return false;
            }
        }

        public BigInteger nextBigInteger() {
            return new BigInteger(next());
        }

        public BigDecimal nextBigDecinal() {
            return new BigDecimal(next());
        }
    }

}
```

栈

常用方法

```
import java.util.Stack; //引用栈
//1.创建一个字符型的栈
Stack<Character> stack=new Stack<>();
System.out.println(stack);
//2.测试栈是否为空
System.out.println(stack.empty());
//3.入栈
stack.push('a');
stack.push('b');
```

```
stack.push('c');
System.out.println(stack);
//4.查看栈顶元素
System.out.println(stack.peek());
System.out.println(stack);
//5.出栈
stack.pop();
System.out.println(stack);
//6.返回对象在栈中的位置
System.out.println(stack.search('b'));
System.out.println(stack.search('a'));

/*

输出结果
//////////////////////////////////////////
[]
true
[a, b, c]
c
[a, b, c]
[a, b]
1
2
//////////////////////////////////////////
*/
```

队列（双向队列）

常用方法

add 增加一个元素 如果队列已满，则抛出一个IllegalSlabEepeplian异常

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

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

offer 添加一个元素并返回true，如果队列已满，则返回false

poll 移除并返问队列头部的元素，如果队列为空，则返回null

peek 返回队列头部的元素，如果队列为空，则返回null

put 添加一个元素，如果队列满，则阻塞

take 移除并返回队列头部的元素，如果队列为空，则阻塞

	Throws Exception抛出异常	Returns special value返回特殊值
Insert 插入	add(e)	offer(e)
Remove 删除	remove()	poll()
Examine 校验	element()	peek()

声明

```
Queue<Node> Q = new LinkedList<>();
*ArrayBlockingQueue ：一个由数组支持的有界队列。
*LinkedBlockingQueue ：一个由链接节点支持的可选有界队列。
*PriorityBlockingQueue ：一个由优先级堆支持的无界优先级队列。
```

堆（优先队列）

优先队列PriorityQueue是Queue接口的实现，可以对其中元素进行排序，

可以放基本数据类型的包装类（如：Integer，Long等）或自定义的类

对于基本数据类型的包装器类，优先队列中元素默认排列顺序是升序排列

但对于自己定义的类来说，需要自己定义比较器

常用方法

```
peek()//返回队首元素
poll()//返回队首元素，队首元素出队列
add()//添加元素
size()//返回队列元素个数
isEmpty()//判断队列是否为空，为空返回true,不空返回false
```

优先队列的使用

基本数据类型

```
//自定义比较器，降序排列
static Comparator<Integer> cmp = new Comparator<Integer>() {
      public int compare(Integer e1, Integer e2) {
        return e2 - e1;
      }
};

public static void main(String[] args) {

    //不用比较器，默认小顶堆
    Queue<Integer> q = new PriorityQueue<>();
    q.add(3);
    q.add(2);
```

```
    q.add(4);
    while(!q.isEmpty()){
        System.out.print(q.poll()+" ");
    }
        /**
         * 输出结果
         * 2 3 4
         */


    //大顶堆
    PriorityQueue<Integer> maxHeap = new PriorityQueue<Integer>(11,new
Comparator<Integer>(){
        @Override
        public int compare(Integer i1,Integer i2){
            return i2-i1;
        }
    });
}
```

自定义类（引用数据类型）

```
//矩形类
class Node{
    public Node(int chang,int kuan)
    {
        this.chang=chang;
        this.kuan=kuan;
    }
    int chang;
    int kuan;
}

public class Test {
       //自定义比较类，先比较长，长升序排列，若长相等再比较宽，宽降序
    static Comparator<Node> cNode=new Comparator<Node>() {
        public int compare(Node o1, Node o2) {
            if(o1.chang!=o2.chang)
                return o1.chang-o2.chang;
            else
                return o2.kuan-o1.kuan;
        }

    };
    public static void main(String[] args) {
        Queue<Node> q=new PriorityQueue<>(cNode);
        Node n1=new Node(1, 2);
        Node n2=new Node(2, 5);
        Node n3=new Node(2, 3);
        Node n4=new Node(1, 2);
        q.add(n1);
        q.add(n2);
        q.add(n3);
        Node n;
        while(!q.isEmpty())
        {
            n=q.poll();
```

```
            System.out.println("长: "+n.chang+" 宽: " +n.kuan);
        }
        /**
          * 输出结果
          * 长: 1 宽: 2
          * 长: 2 宽: 5
          * 长: 2 宽: 3
          */
    }
}
```

Collections类的常用方法

排序、反转、最大最小、二分查找

基本数据类型

```
public class Practice {
    public static void main(String[] args){
        ArrayList<Integer> c = new ArrayList<>();
        c.add(2);
        c.add(1);
        c.add(0);
        c.add(3);
        System.out.println(c);
        Collections.sort(c);//排序
        System.out.println(c);
        Collections.reverse(c);//反转
        System.out.println(c);
        int min = Collections.min(c);//返回ArrayList里的最小值
        int max = Collections.max(c);//返回ArrayList里的最大值
        Collections.rotate(c,1);//向右移动1位
        Collections.rotate(c,-1);//向左移动1位

    }
}
```

引用数据类型

```
class Node {
    int index;//加入顺序
    int value;//需要排序的值

    public Node(int index,int value){
        this.index = index;
        this.value = value;
    }
}


ArrayList<Node> nodes = new ArrayList<>();
Node i1 = new Node(1,1);
nodes.add(i1);
Node i2 = new Node(2,3);
nodes.add(i2);
Node i3 = new Node(3,2);
```

```
nodes.add(i3);
Node i4 = new Node(4,4);
nodes.add(i4);

//Node根据value升序排序
nodes.sort(new Comparator<Node>() {
    @Override
    public int compare(Node o1, Node o2) {
        //value相同的时候可以对index做判断，保证排序后相同value的仍按照原来的顺序
        int num = o1.value - o2.value;
        return (num == 0 ? o1.index - o2.index : num);
    }
});

//Node根据value降序排序
nodes.sort(new Comparator<Node>() {
    @Override
    public int compare(Node o1, Node o2) {
        //value相同的时候可以对index做判断，保证排序后相同value的仍按照原来的顺序
        int num = o2.value - o1.value;
        return (num == 0 ? o1.index - o2.index : num);
    }
});

//Node根据value找最大值
Node n1= Collections.max(nodes, new Comparator<Node>() {
    @Override
    public int compare(Node o1, Node o2) {
        int num = o1.value - o2.value;
        return (num == 0 ? o1.index - o2.index : num);
    }
});
out.println(n1.value);

//Node根据value找最小值
Node n1= Collections.min(nodes, new Comparator<Node>() {
    @Override
    public int compare(Node o1, Node o2) {
        int num = o1.value - o2.value;
        return (num == 0 ? o1.index - o2.index : num);
    }
});
out.println(n1.value);

//Node根据index二分查找
int index= Collections.binarySearch(nodes, i1, new Comparator<Node>() {
    @Override
    public int compare(Node o1, Node o2) {
        return o1.index - o2.index;
    }
});
```

字符串

```
//长度
String str = new String("asdfzxc");
int strlength = str.length();//strlength = 7
```

```
4
5  //某一位置的字符
6  String str = new String("asdfzxc");
7  char ch = str.charAt(4);//ch = z
8
9  //提取子串
10 String str1 = new String("asdfzxc");
11 String str2 = str1.substring(2);//str2 = "dfzxc"
12 String str3 = str1.substring(2,5);//str3 = "dfz"
13
14 //字符串连接
15 //相当于String str = “aa”+”bb”+”cc”
16 String str = "aa".concat("bb").concat("cc");
17
18 //字符串单个字符查找
19 String str = "I am a good student";
20 int a = str.indexOf('a');//a = 2
21 int b = str.indexOf("good");//b = 7
22 int c = str.indexOf("w",2);//c = -1
23 int d = str.lastIndexOf("a");//d = 5
24 int e = str.lastIndexOf("a",3);//e = 2
25
26 //字符串截去空格
27 String str = " a sd ";
28 String str1 = str.trim();
29
30 //字符串包含子串
31 String str = "student";
32 str.contains("stu");//true
33 str.contains("ok");//false
34
35 //字符串按照规定分成数组
36 String str = "asd!qwe|zxc#";
37 String[] str1 = str.split("!|#");//str1[0] = "asd";str1[1] = "qwe";str1[2] =
   "zxc";
```

集合类

HashSet

```
1  import java.util.Iterator;
2  import java.util.HashSet;
3
4
5  public class HashSetTest {
6
7      public static void main(String[] args) {
8          // HashSet常用API
9          testHashSetAPIs() ;
10     }
11
12     /*
13      * HashSet除了iterator()和add()之外的其它常用API
14      */
15     private static void testHashSetAPIs() {
16         // 新建HashSet
```

```
        HashSet set = new HashSet();

        // 将元素添加到Set中
        set.add("a");
        set.add("b");
        set.add("c");
        set.add("d");
        set.add("e");

        // 打印HashSet的实际大小
        System.out.printf("size : %d\n", set.size());

        // 判断HashSet是否包含某个值
        System.out.printf("HashSet contains a :%s\n", set.contains("a"));
        System.out.printf("HashSet contains g :%s\n", set.contains("g"));

        // 删除HashSet中的“e”
        set.remove("e");

        // 将Set转换为数组
        String[] arr = (String[])set.toArray(new String[0]);
        for (String str:arr)
            System.out.printf("for each : %s\n", str);

        // 新建一个包含b、c、f的HashSet
        HashSet otherset = new HashSet();
        otherset.add("b");
        otherset.add("c");
        otherset.add("f");

        // 克隆一个removeset，内容和set一模一样
        HashSet removeset = (HashSet)set.clone();
        // 删除“removeset中，属于otherSet的元素”
        removeset.removeAll(otherset);
        // 打印removeset
        System.out.printf("removeset : %s\n", removeset);

        // 克隆一个retainset，内容和set一模一样
        HashSet retainset = (HashSet)set.clone();
        // 保留“retainset中，属于otherSet的元素”
        retainset.retainAll(otherset);
        // 打印retainset
        System.out.printf("retainset : %s\n", retainset);


        // 遍历HashSet
        for(Iterator iterator = set.iterator();
                iterator.hasNext(); )
            System.out.printf("iterator : %s\n", iterator.next());

        // 清空HashSet
        set.clear();

        // 输出HashSet是否为空
        System.out.printf("%s\n", set.isEmpty()?"set is empty":"set is not
empty");
    }

```

```
}
```

HashMap

```
1.import java.util.HashMap;//导入;

2.HashMap<K, V> map=new HashMap<K, V>();//定义map，K和V是类，不允许基本类型;

3.void clear();//清空

4.put(K,V);//设置K键的值为V

5.V get(K);//获取K键的值

6.boolean isEmpty();//判空

7.int size();//获取map的大小

8.V remove(K);//删除K键的值，返回的是V，可以不接收

9.boolean containsKey(K);//判断是否有K键的值

10.boolean containsValue(V);//判断是否有值是V

11.Object clone();//浅克隆，类型需要强转；如HashMap<String , Integer> map2=
(HashMap<String, Integer>) map.clone();
```

进制转换

```
//数字转字符串
int i = 0;
String ii = i + "";

//字符串转数字
String i = "213";
int ii = Integer.parseInt(i);
out.println(ii);

//字符转数字
char i = '1';
int ii = Integer.parseInt(String.valueOf(i));

//进制转换
//使用Long类中的方法得到整数之间的各种进制转换的字符串
Long.toBinaryString(long l)
Long.toOctalString(long l)
Long.toHexString(long l)
Long.toString(long l, int p)//p作为任意进制

int ii = 1209814;
out.println(Long.toBinaryString(ii));
```

BigInteger类的常用方法

基本方法

```
import java.math.BigInteger;

//BigInteger 对象的创建
BigInteger a = new BigInteger("123"); // 这里是字符串

//改变 BigInteger 的值
String str = "123";
BigInteger a = BigInteger.valueOf(str);
int num = 456;
BigInteger a = BigInteger.valueOf(num);

//基本常量
a = BigInteger.ONE // 1
b = BigInteger.TEN // 10
c = BigInteger.ZERO // 0

//输出
BigInteger a;
a = in.nextBigInteger();
System.out.print(a.toString());
```

进制转换

```
//BigInteger 转化成十进制表示的 String
System.out.print(a.toString());

//BigInteger 转化成 p 进制表示的 String
int p = 2;
System.out.print(a.toString(p)); // 输出a的二进制

//BigInteger 二进制下的长度
BigInteger n = new BigInteger("12");
System.out.println(n.bitLength()); // 4


//进制转换
String str = "1011100111";
int radix = 2;
BigInteger interNum1 = new BigInteger(str,radix);   //743

//我们通常不写，则是默认成10进制转换，如下：
BigInteger interNum2 = new BigInteger(str);         //1011100111
```

运算

```
@Test
    public void testBasic() {
        BigInteger a = new BigInteger("13");
        BigInteger b = new BigInteger("4");
        int n = 3;

        //1.加
        BigInteger bigNum1 = a.add(b);          //17
```

```
        //2.减
        BigInteger bigNum2 = a.subtract(b);      //9
        //3.乘
        BigInteger bigNum3 = a.multiply(b);      //52
        //4.除
        BigInteger bigNum4 = a.divide(b);        //3
        //5.取模(需 b > 0，否则出现异常：ArithmeticException("BigInteger: modulus not positive"))
        BigInteger bigNum5 = a.mod(b);           //1
        //6.求余
        BigInteger bigNum6 = a.remainder(b);     //1
        //7.平方(需 n >= 0，否则出现异常：ArithmeticException("Negative exponent"))
        BigInteger bigNum7 = a.pow(n);           //2197
        //8.取绝对值
        BigInteger bigNum8 = a.abs();            //13
        //9.取相反数
        BigInteger bigNum9 = a.negate();         //-13
    }


//除法取余
BigInteger a = new BigInteger("123");
BigInteger b = new BigInteger("456");
BigInteger result[] = b.divideAndRemainder(a); // 该函数返回的是数组
System.out.println("商是：" + result[0] + "；余数是：" + result[1]);

//幂
BigInteger a = new BigInteger("2");
System.out.println(a.pow(3)); // 8

//最大公约数
BigInteger a = new BigInteger("12");
BigInteger b = new BigInteger("56");
System.out.println(a.gcd(b)); // 4

//保留两位小数
double d = 3.1415926;
String result = String .format("%.2f",d);
```

比较

```
//比较
BigInteger a = new BigInteger("123");
BigInteger b = new BigInteger("456");
if(a.compareTo(b) == 0) System.out.println("a == b"); // a == b
else if(a.compareTo(b) > 0) System.out.println("a > b"); // a > b
else if(a.compareTo(b) < 0) System.out.println("a < b"); // a < b

//相等
BigInteger a = new BigInteger("123");
BigInteger b = new BigInteger("456");
System.out.println(a.equals(b)); // a == b 时为 true 否则为 false
```

数据类型转换

```
    //类型转换(返回类型如下)
    @Test
    public void testToAnother() {
        BigInteger bigNum = new BigInteger("52");
        int radix = 2;

        //1.转换为bigNum的二进制补码形式
        byte[] num1 = bigNum.toByteArray();
        //2.转换为bigNum的十进制字符串形式
        String num2 = bigNum.toString();        //52
        //3.转换为bigNum的radix进制字符串形式
        String num3 = bigNum.toString(radix);   //110100
        //4.将bigNum转换为int
        int num4 = bigNum.intValue();
        //5.将bigNum转换为long
        long num5 = bigNum.longValue();
        //6.将bigNum转换为float
        float num6 = bigNum.floatValue();
        //7.将bigNum转换为double
        double num7 = bigNum.doubleValue();
    }
```