第十一章 其他

文章目录

基础知识

位运算

1. 补码

   用补码的形式表示负数：先按正数转换，再取反+1

```
要将十进制的-10用二进制表示，先将10用二进制表示：
0000 0000 0000 1010
取反：
1111 1111 1111 0101
加1：
1111 1111 1111 0110
所以，-10的二进制表示就是：1111 1111 1111 0110
```

2. 按位与&

   A & 1 = A

   A & 0 = 0

   只有当相应位上的数都是1时，该位才取1，否则该为为0。

```
将10与-10进行按位与(&)运算：
0000 0000 0000 1010
1111 1111 1111 0110
----------------------
0000 0000 0000 0010
所以：10 & -10 = 0000 0000 0000 0010
```

3. 按位或|

   只要相应位上存在1，那么该位就取1，均不为1，即为0。

```
将10与-10进行按位或(|)运算：
0000 0000 0000 1010
1111 1111 1111 0110
----------------------
1111 1111 1111 1110
所以：10 | -10 = 1111 1111 1111 1110
```

4. 按位异或^

   只有当相应位上的数字不相同时，该为才取1，若相同，即为0。

```
将10与-10进行按位异或(^)运算：
0000 0000 0000 1010
1111 1111 1111 0110
----------------------
1111 1111 1111 1100
所以：10 ^ -10 = 1111 1111 1111 1100
```

5. 取反~
   每个位上都取相反值。

```
对10进行取反(~)运算：
0000 0000 0000 1010
---------------------
1111 1111 1111 0101
所以：~10 = 1111 1111 1111 0101
```

6. 左移<<
   进行左移运算，用来将一个数各二进制位全部向左移动若干位。
   左移一位的结果就是原值乘2，左移两位的结果就是原值乘4。

```
对10左移2位(就相当于在右边加2个0)：
0000 0000 0000 1010
--------------------
0000 0000 0010 1000
所以：10 << 2 = 0000 0000 0010 1000 = 40
```

7. 右移>>
   进行右移运算，用来将一个数各二进制位全部向右移动若干位。
   右移一位的结果就是原值除2，左移两位的结果就是原值除4。

```
对10右移2位(就相当于在左边加2个0)：
0000 0000 0000 1010
--------------------
0000 0000 0000 0010
所以：10 >> 2 = 0000 0000 0000 0010 = 2
```

leetcode

例1：倒转二进制（190）【位运算】

题目描述
颠倒给定的 32 位无符号整数的二进制位。

```
示例 1：

输入：00000010100101000001111010011100
输出：00111001011110000010100101000000
解释：输入的二进制串 00000010100101000001111010011100 表示无符号整数 43261596，
      因此返回 964176192，其二进制表示形式为 00111001011110000010100101000000。
```

程序代码

```
// 颠倒给定的 32 位整数的二进制位
 public int reverseBits(int n) {
        // 代码讲解：https://www.bilibili.com/video/av3878878?from=search&seid=11865449822834851818
    // int n 为32位二进制数
        // 采用分治法，先16位反转，再8位反转，再4位反转……
        // 位运算思路：（以8位字符，4位反转为例0110 0001）
        // 先拿n二进制数与 0000 1111 做与& 操作，得到后四位二进制数a1 = 0000 0001
        // 再拿n二进制数右移四位与 0000 1111 做与& 操作，得到前四位二进制数字a2 = 0000 0110
        // 将a1左移四位 a1 << 4 + a2 得到反转后的值 0001 0110
        // 四位反转后进行二位反转 0100 1001，最后一位反转得出最终结果 1000 0110
        // 定义5个过滤器
        int m_16 = 0x0000ffff;  // 16位转置过滤器: 0000 0000 0000 0000 1111 1111 1111 1111
        int m_8 = 0x00ff00ff;   // 8位转置过滤器: 0000 0000 1111 1111 0000 0000 1111 1111
        int m_4 = 0x0f0f0f0f;   // 4位转置过滤器: 0000 1111 0000 1111 0000 1111 0000 1111
        int m_2 = 0x33333333;   // 2位转置过滤器: 0011 0011 0011 0011 0011 0011 0011 0011
        int m_1 = 0x55555555;   // 1位转置过滤器: 0101 0101 0101 0101 0101 0101 0101 0101
        // n与 16位过滤器m_16 进行与&运算 得后16位
        // n 向右移16位 并与 m_16 &运算 得 前16位
        // 将 后16位 左移16位 并与后16位相加 得 16位转置结果
        int reverse_16 = ((n & m_16) << 16) + ((n >> 16) & m_16);   // 16位转置结果
        int reverse_8 = ((reverse_16 & m_8) << 8) + ((reverse_16 >> 8) & m_8);  // 8位转置结果
        int reverse_4 = ((reverse_8 & m_4) << 4) + ((reverse_8 >> 4) & m_4);    // 4位转置结果
        int reverse_2 = ((reverse_4 & m_2) << 2) + ((reverse_4 >> 2) & m_2);    // 2位转置结果
        int reverse_1 = ((reverse_2 & m_1) << 1) + ((reverse_2 >> 1) & m_1);    // 1位转置结果
        return (int)reverse_1;
 }
```

剑指offer

例1：二维数组的查找（1）【数组、智力】

题目描述
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。
算法思路
矩阵是有序的，从左下角开始遍历。向上数字递减，向右数字递增，因此从左下角开始查找（可以减少四种移动方式中两种可能性）
当要查找数字比左下角大的时候，右移;
当要查找数字比左下角小的时候，上移.
程序代码

```
    public boolean Find(int target, int [][] array) {
        // 矩阵是有序的，从左下角开始遍历。向上数字递减，向右数字递增
        // 因此从左下角开始查找（可以减少四种移动方式中两种可能性）
        // 当要查找数字比左下角大的时候，右移;
        // 当要查找数字比左下角小的时候，上移.
        if(array == null || array.length == 0)return false;
        int row = array.length-1;
        int col = array[0].length-1;
        int idx_x = row;
```

```
        int idx_y = 0;
        while(idx_x >=0 && idx_y <= col) {
            if(target == array[idx_x][idx_y])return true;   // 查找到target，返回true
            if(target < array[idx_x][idx_y]){
                idx_x--; // 查找数字较小，则上移
            }else {
                idx_y++;// 查找数字较大，则右移动
            }
        }
        return false;   // 遍历结束仍未查找到target，返回false
    }
```

例2：二进制中1的个数（11）【位运算】

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。

程序代码

```
    // 11.二进制中1的个数
    // 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。
    public int NumberOf1(int n) {
//          方式1:从n的2进制形式最右边开始右移判断是不是1（可能陷入死循环）
//          这种方式用于负数运算可能陷入死循环，因为负数右移的时候，最高位补的是1，本题求1的个数，此时会有无数个1
//          int count = 0;// 二进制表示中1的个数
//          while(n!=0) {
//              if((n & 1) == 1) count++;// 如果 n & 1 = 1(1和n进行位与运算),表示n的二进制表示数最后一位为 1，则二进制表示中 1 的个数++
//              n = n >> 1;// n 的二进制表示数 整体右移一位（相当于/2）
//          }
//          return count;
//          方式2:从1开始不断左移动判断是不是1
                int count = 0;
                int flag = 1;   //从1开始左移
                while(flag != 0) {
                    if((n & flag) != 0)count++; // 从右向左遍历n的每一位
                    flag = flag << 1;           // 位数指示器左移
                }
                return count;
    }
```

例3：数值的整数次方（12）【代码的完整性】

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

程序代码

```
    // 12.数值的整数次方
    // 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
    public double Power(double base, int exponent) {
        // 考虑指数运算的所有可能性
            // 即手写 Math.power(base, exponent) 实现
            // base = 0,exponent = 0 , 未定义
            // base = 0,exponent > 0 , result = 0
            // base = 0,exponent < 0 , 异常
            // base != 0,exponent = 0 , result = 1,
            // base != 0,exponent > 0 , result = base^exponent,
            // base != 0,exponent < 0 , result = 1/(base^(-exponent))
            if(base == 0) {
                if(exponent > 0)return 0;   // base = 0,exponent > 0 , result = 0
                else return 0;  // base = 0,exponent < 0 , 异常
            }else {
                double result = 1;
                if(exponent == 0)return 1;  // base != 0,exponent = 0 , result = 1,
                else if(exponent > 0) {     // base != 0,exponent > 0 , result = base^exponent,
                    for(int i = 0;i<exponent;i++)result *= base;
                    return result;
                }else {                     // base != 0,exponent < 0 , result = 1/(base^(-exponent))
                    for(int i = 0;i<-exponent;i++)result *= base;
                    return 1/result;
                }
            }
    }
```

例4：调整数组顺序使奇数位于偶数前面（13）【代码的完整性】

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。

程序代码

```
    // 13.调整数组顺序使奇数位于偶数前面
    // 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。
    public void reOrderArray(int [] array) {
        // 1. 遍历数组中的每个数字（数组前半段全是奇数，后半段全是偶数）
            // 2. 若遍历到奇数，则向前遍历插入到第一个遇到的奇数后面
            boolean isInsert = false;
            for (int i = 0;i<array.length;i++) {
                if(array[i]%2 == 1) {       //若遍历到奇数
                    int temp = array[i];
                    for(int j=i-1;j>=0;j--)
                        if(array[j]%2==1) { // 若遇到奇数，则停止遍历，并插入该奇数后
                            array[j+1] = temp;
                            isInsert = true;
                            break;
                        }else {             // 若遇到偶数，则将偶数向后移动
                            array[j+1] = array[j];
```

```
                    }
                if(!isInsert)array[0] = temp;
            }
        }
    }
```

例5：顺时针打印矩阵（19）【画图让抽象形象化】

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.

程序代码

```
static int[] dx = {0, 1, 0, -1};    // 水平方向偏移
    static int[] dy = {1, 0, -1, 0};    // 垂直方向偏移
    int di = 0;                     // 偏移指针
    int x = 0;                      // 当前位置x
    int y = 0;                      // 当前位置y

    public ArrayList<Integer> printMatrix(int [][] matrix) {
        // 按顺时针遍历的方向顺序：右->下->左->上
            // 定义边界与已经遍历的矩阵，如果到达边界或者下一个访问位置已遍历
            // 则变换方向
        if(matrix == null || matrix.length == 0)return null;

        ArrayList<Integer> result = new ArrayList<Integer>();
            int row = matrix.length;        // 数组行数
            int col = matrix[0].length; // 数组列数
            int sum = row * col;            // 数组中元素总数

            // 初始化访问数组,所有元素均未遍历，均为0
            int[][] visited = new int[row][col];
            for(int i=0; i<row; i++)
                for(int j=0; j<col; j++)
                    visited[i][j] = 0;

            // 顺时针访问matrix所有元素
            while(sum-- > 0) {
                result.add(matrix[x][y]);
                visited[x][y] = 1;
                nextStep(visited);
            }

            return result;
    }

    public void nextStep(int[][] visited) {
            // 继续前进下一步
            int row = visited.length;
            int col = visited[0].length;

            int px = x + dx[di];
            int py = y + dy[di];

            if(px<0 || px>=row || py<0 || py>=col || visited[px][py]==1){
                // 超出边界 || 该节点已访问 需要更换访问方向
                    if(di == 3)di = 0;
                    else di++;
            }
            x = x + dx[di];
            y = y + dy[di];
    }
```

例6：整数中1出现的次数（31）

题目描述

求出1$_{13的整数中1出现的次数,并算出100}$1300的整数中1出现的次数？为此他特别数了一下1~13中包含1的数字有1、10、11、12、13因此共出现6次,但是对于后面问题他就没辙了。ACMer希望你们帮帮他,并把问题更加普遍化,可以很快的求出任意非负整数区间中1出现的次数（从1 到 n 中1出现的次数）。

程序代码

```
    // 31.整数中1出现的次数
    // 求出1~13的整数中1出现的次数,并算出100~1300的整数中1出现的次数？
        // 为此他特别数了一下1~13中包含1的数字有1、10、11、12、13因此共出现6次,但是对于后面问题他就没辙了。
    // ACMer希望你们帮帮他,并把问题更加普遍化,可以很快的求出任意非负整数区间中1出现的次数（从1 到 n 中1出现的次数）。
    public int NumberOf1Between1AndN_Solution(int n) {
        // 求整数中1的个数，判断整数的各位数字是否为1，若为1，则次数++
        int count = 0;
            if(n<1)return count;
        for(int i=1;i<=n;i++) count += NumberOf1InNumber(i);
            return count;
    }

    public int NumberOf1InNumber(int n) {
            int times = 0;
            while(n!=0) {
                if(n%10 == 1)times++;
                n = n/10;
            }
           return times;
    }
```

例7：丑数（33）

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。

程序代码

```
    Queue<Integer> multi_2 = new LinkedList<Integer>();
    Queue<Integer> multi_3 = new LinkedList<Integer>();
```

```
    Queue<Integer> multi_5 = new LinkedList<Integer>();
    List<Integer> min_array = new ArrayList<Integer>();// 存储1..index的丑数
    public int GetUglyNumber_Solution(int index) {
            // 暴力穷举
            // 定义3个队列,分别为*2队列*3队列*5队列
            // 丑数一定为 2^x*3^y*5^z
            // 即丑数均是从这3个队列中计算所得，即任一丑数通过*2,*3,*5计算所得
            // 丑数数组为逐一求得丑数的集合，将丑数最后一个丑数*2，*3，*5并放入队列（可能最小值队列）
            // 最小值为三个3队列中队首元素中最小值，逐一比较，若为最小值则将队列出队，并将最小值存入丑数数组
            min_array.add(1);                   // 第一个丑数为1
            if(index<1)return 0;
            for(int i=0;i<index;i++)
                putUglyNumberInArray(i);

            return min_array.get(index-1);
    }

    public void putUglyNumberInArray(int i) {
            // 将第 i 个丑数放入数组中
            int lastUglyNumber = min_array.get(min_array.size()-1);// 获取丑数数组最后一个数
            multi_2.offer(lastUglyNumber*2);
            multi_3.offer(lastUglyNumber*3);
            multi_5.offer(lastUglyNumber*5);
            min_array.add(chooseMinValueOfThreeQueue());
    }

    public Integer chooseMinValueOfThreeQueue() {
            int min_2 = multi_2.peek();
            int min_3 = multi_3.peek();
            int min_5 = multi_5.peek();
            int min_value = min_2<min_3?(min_2<min_5?min_2:min_5):(min_3<min_5?min_3:min_5);
            if(min_value == min_2)multi_2.poll();
            if(min_value == min_3)multi_3.poll();
            if(min_value == min_5)multi_5.poll();
            return min_value;
    }
```

例8：和为S的连续正数序列（40）

题目描述

小明很喜欢数学,有一天他在做数学作业时,要求计算出9~16的和,他马上就写出了正确答案是100。但是他并不满足于此,他在想究竟有多少种连续的正数序列的和为100(至少包括两个数)。没多久,他就得到另一组连续正数和为100的序列:18,19,20,21,22。现在把问题交给你,你能不能也很快的找出所有和为S的连续正数序列? Good Luck!

输出描述:

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

程序代码

```
// 40. 和为S的连续正数序列
 // 小明很喜欢数学,有一天他在做数学作业时,要求计算出9~16的和,他马上就写出了正确答案是100。
 // 但是他并不满足于此,他在想究竟有多少种连续的正数序列的和为100(至少包括两个数)。
 // 没多久,他就得到另一组连续正数和为100的序列:18,19,20,21,22。
 // 现在把问题交给你,你能不能也很快的找出所有和为S的连续正数序列? Good Luck!
 // 输出描述:
     // 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
 ArrayList<ArrayList<Integer>> resultList = new ArrayList<ArrayList<Integer>>();
 ArrayList<Integer> seqList = new ArrayList<Integer>();
 public ArrayList<ArrayList<Integer>> FindContinuousSequence(int sum) {
         // 对sum前一半数值进行遍历，对于每一个数字
         // 以该数字开始向后连加，若求和>=sum时
         // 若==100则存储到结果，否则开始处理下一个数字
         for(int i=1;i <= sum/2+1;i++)   // 只需要访问前一半数值
             getLongestSeqEqualsSum(i, sum);
         return resultList;
 }

 public void getLongestSeqEqualsSum(int i,int sum) {
         seqList.clear();
         int seqSum = 0;
         for(int j=i;j<= sum/2+1;j++) {
             seqList.add(j);
             seqSum += j;
             if(seqSum>=sum) {
                 if(seqSum == sum && seqList.size()>1)resultList.add(new ArrayList<Integer>(seqList));
                 break;
             }
         }
 }
```

例9：和为S的2个数字（41）

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。

程序代码

```
public ArrayList<Integer> FindNumbersWithSum(int [] array,int sum) {
           // 1. 定义2个指针start,end，最开始的时候start指向0，end指向数组最后一位array.length-1
           // 2. 若array[start]+array[end]==sum,则输出结果new List(){array[start],array[end]}
           //    （根据数学特性，当(start+end)相等时，max(|end-start|)时|start*end|最大）
           //    若array[start]+array[end]>sum，end--
           //    若array[start]+array[end]<sum,start++
           int start = 0;
           int end = array.length-1;
           ArrayList<Integer> result = new ArrayList<Integer>();
           while(start<end) {
               int curSum = array[start]+array[end];
               if(curSum == sum) {
                   result.add(array[start]);
                   result.add(array[end]);
                   return result;
               }
```

```
            else if(curSum < sum)start++;
            else end--;
        }
        return result;
    }
```

例10：扑克牌顺子（44）

题目描述

LL今天心情特别好,因为他去买了一副扑克牌,发现里面居然有2个大王,2个小王(一副牌原本是54张-)...他随机从中抽出了5张牌,想测测自己的手气,看看能不能抽到顺子,如果抽到的话,他决定去买体育彩票,嘿嘿！！“红心A,黑桃3,小王,大王,方片5”，“Oh My God!”不是顺子...LL不高兴了,他想了想,决定大\小 王可以看成任何数字,并且A看作1,J为11,Q为12,K为13。上面的5张牌就可以变成“1,2,3,4,5”(大小王分别看作2和4),“So Lucky!”。LL决定去买体育彩票啦。 现在,要求你使用这幅牌模拟上面的过程,然后告诉我们LL的运气如何， 如果牌能组成顺子就输出true，否则就输出false。为了方便起见,你可以认为大小王是0。

程序代码

```
public boolean isContinuous(int [] numbers) {
        // 1.对数组进行排序
        // 2.获取非0外的max和min
        // 3.若max-min < 5 && min~max中没有重复的值 => 顺子
        if(numbers == null || numbers.length == 0)return false;
        Arrays.sort(numbers);
        int min_idx = -1;
        for(int i=0;i<numbers.length;i++) {
            if(numbers[i]==0)min_idx = i;
            else if(i>0 && numbers[i] == numbers[i-1])return false;
        }
        int min = numbers[min_idx+1];
        int max = numbers[numbers.length-1];
        if(max-min<5)return true;
        return false;
    }
```

例11：1+2+3+...+n（46）【条件与&&短路原则】

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）。

程序代码

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

例12：不用加减乘除做加法（47）【位运算】

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。

程序代码

```
    // 47. 不用加减乘除做加法
    // 写一个函数，求两个整数之和，要求不得使用 +、-、*、/ 四则运算符号。
    public int Add(int a, int b) {
        // a ^ b 表示没有考虑进位的情况下两数的和，(a & b) << 1 就是进位。
        // 递归会终止的原因是 (a & b) << 1 最右边会多一个 0，那么继续递归，进位最右边的 0 会慢慢增多，最后进位会变为 0，递归终止。
        return b == 0 ? a : Add(a ^ b, (a & b) << 1);
    }
```

例13：构建乘积数组（50）【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。

程序代码

```
    // 50.构建乘积数组
    // 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。
    public int[] multiply(int[] A) {
        // 构造对角线为1的n*n二维数组
        // 计算二维数组中每一行i的乘积，即为B[i]的值
        if(A == null || A.length == 0)return A;
        int[][] multiArray = constructArray(A);
        int[] B = new int[A.length];
        for(int i = 0;i<A.length;i++)B[i] = 1;
        for(int i=0;i<A.length;i++) {
            for(int j=0;j<A.length;j++)
                B[i] *= multiArray[i][j];
        }
        return B;
    }

    public int[][] constructArray(int[] A){
        // 根据一维数组A[]构造对应二维乘积数组
        int[][] multiArray = new int[A.length][A.length];
        for(int i=0;i<A.length;i++) {
            for(int j=0;j<A.length;j++) {
                if(i==j)multiArray[i][j] = 1;
                else multiArray[i][j] = A[j];
            }
        }
        return multiArray;
    }
```

例14：正则表达式匹配（51）【模拟】

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

程序代码

```
    // 51.正则表达式匹配[模拟思想]
    // 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
         public boolean match(char[] str, char[] pattern) {
//                当模式中的第二个字符不是“*”时：
//                1、如果字符串第一个字符和模式中的第一个字符相匹配，那么字符串和模式都后移一个字符，然后匹配剩余的。
//                2、如果 字符串第一个字符和模式中的第一个字符相不匹配，直接返回false。
//
//                而当模式中的第二个字符是“*”时：
//                如果字符串第一个字符跟模式第一个字符不匹配，则模式后移2个字符，继续匹配。如果字符串第一个字符跟模式第一个字符匹配，可以有3种匹配方式：
//                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；
            if (str == null || pattern == null) {
                return false;
            }
            int strIndex = 0;
            int patternIndex = 0;
            return matchCore(str, strIndex, pattern, patternIndex);
        }

        public boolean matchCore(char[] str, int strIndex, char[] pattern, int patternIndex) {
            //有效性检验：str到尾，pattern到尾，匹配成功
            if (strIndex == str.length && patternIndex == pattern.length) {
                return true;
            }
            //pattern先到尾，匹配失败
            if (strIndex != str.length && patternIndex == pattern.length) {
                return false;
            }
            //模式第2个是*，且字符串第1个跟模式第1个匹配,分3种匹配模式；如不匹配，模式后移2位
            if (patternIndex + 1 < pattern.length && pattern[patternIndex + 1] == '*') {
                if ((strIndex != str.length && pattern[patternIndex] == str[strIndex]) || (pattern[patternIndex] == '.' && strIndex != str.length)) {
                    return matchCore(str, strIndex, pattern, patternIndex + 2)//模式后移2，视为x*匹配0个字符
                            || matchCore(str, strIndex + 1, pattern, patternIndex + 2)//视为模式匹配1个字符
                            || matchCore(str, strIndex + 1, pattern, patternIndex);//*匹配1个，再匹配str中的下一个
                } else {
                    return matchCore(str, strIndex, pattern, patternIndex + 2);
                }
            }
            //模式第2个不是*，且字符串第1个跟模式第1个匹配，则都后移1位，否则直接返回false
            if ((strIndex != str.length && pattern[patternIndex] == str[strIndex]) || (pattern[patternIndex] == '.' && strIndex != str.length)) {
                return matchCore(str, strIndex + 1, pattern, patternIndex + 1);
            }
            return false;
            }
```

例15：表示数值的字符串（52）【模拟】

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。

程序代码

```
        // 52.表示数值的字符串
            // 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。
        public boolean isNumeric(char[] str) {
             // 法一：直接采用正则表达式求解
                // [\\+\\-]?            -> 正或负符号出现与否
                // \\d*                 -> 整数部分是否出现，如-.34 或 +3.34均符合
                // (\\.\\d+)?           -> 如果出现小数点，那么小数点后面必须有数字；
                //                      否则一起不出现
                // ([eE][\\+\\-]?\\d+)? -> 如果存在指数部分，那么e或E肯定出现，+或-可以不出现，
                //                      紧接着必须跟着整数；或者整个部分都不出现
//                String string = String.valueOf(str);
//                return string.matches("[\\+\\-]?\\d*(\\.\\d+)?([eE][\\+\\-]?\\d+)?");
                // 法2：对字符串中的每个字符进行判断分析，基本格式：+/- A.B e(E) +/- C
                // e（E）后面只能接数字，并且不能出现2次
                // 对于+、-号，只能出现在第一个字符或者是e的后一位
                // 对于小数点，不能出现2次，e后面不能出现小数点
                boolean hasPoint = false;
                boolean hasE = false;
                for(int i=0;i<str.length;i++) {
                    if(i == 0) {    //首字符单独处理,必须为数字或者+/-
                        if(!(isInteger(str[i]) || str[i]=='+' || str[i] == '-'))return false;
                    }else if(str[i] == '.') {
                        if(hasPoint || hasE)return false;   // 小数点只能出现一次且只能出现在指数符号前面
                        hasPoint = true;
                    }else if(str[i] == 'E' || str[i] == 'e') {
                        i++;
                        if(hasE || i==str.length || !(isInteger(str[i]) || str[i]=='+' || str[i] == '-'))return false;
                        hasE = true;
                    }else if(!isInteger(str[i])) {
                        return false;
                    }
                }
                return true;
        }
```

```
        public boolean isInteger(Character c) {
                if(c >= '0' && c<='9')return true;
                else return false;
        }
```

例16：数据流中中位数（62）

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。

程序代码

```
        // 62.数据流中的中位数
        // 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。
        List<Double> sortList = new ArrayList<Double>();
        public void Insert(Integer num) {
            // 输入的时候插入排序，得到一个由小到大的排序序列
                sortList.add((double)num);
                int idx = sortList.size()-1;    // 最终插入位置
                for(int i=sortList.size()-2;i>=0;i--) {
                    // 从后向前遍历
                    // 若当前值小于遍历值，遍历值后移一位(交换当前值与遍历值)
                    if(num < sortList.get(i)) { sortList.set(i+1, sortList.get(i));idx = i;}
                    else break;
                }
                sortList.set(idx, (double)num);
        }

        public Double GetMedian() {
            // 若为奇数，取中间值
                // 若为偶数，取中间值求平均
                if(sortList == null || sortList.size() == 0)return null;
                else {
                    Integer length = sortList.size();
                    if(length == 1)return sortList.get(0);
                    if(length % 2 == 1) return sortList.get(length/2);
                    else return (sortList.get(length/2)+sortList.get(length/2+1))/2;
                }
        }

```

2019校招

例1：被3整除（2）【数学】

题目描述

小Q得到一个神奇的数列: 1, 12, 123,...12345678910,1234567891011...。

并且小Q对于能否被3整除这个性质很感兴趣。

小Q现在希望你能帮他计算一下从数列的第l个到第r个(包含端点)有多少个数可以被3整除。

输入描述:

输入包括两个整数l和r(1 <= l <= r <= 1e9), 表示要求解的区间两端。

输出描述:

输出一个整数, 表示区间内能被3整除的数字个数。

程序代码

```
    // 2. 被3整除（Math）
    // 数学：位数和可以被3整除 == 该数字可被3整除
    // 一个数所有位数的和相加如果等于3的倍数，则这个整数是3的倍数。
    // 这里第一个数是1，第二个是12，第三个是123……第n个数是123……(n-1)n，各个位之和可以算成(i+1)*i/2，
    // 这里如果是大于等于两位数，它算成一个数和把每一位分开计算对3取余的结果都是一样的，所以没关系。
    // 所以，直接遍历l到r，根据通项公式判断即可。
    public void divideThree() {
        // 输入
        // 大数用Long处理
        Scanner sc = new Scanner(System.in);
        long l = sc.nextLong();
        long r = sc.nextLong();

        int num = 0;

        for(long i=l;i<=r;i++) {
                long bitSum = (1+i)*i/2;
                if(bitSum % 3 == 0)num++;
        }

        System.out.print(num);
    }
```

例2：迷路的牛牛（4）【模拟】

题目描述

牛牛去犇犇老师家补课，出门的时候面向北方，但是现在他迷路了。虽然他手里有一张地图，但是他需要知道自己面向哪个方向，请你帮帮他。

输入描述:

每个输入包含一个测试用例。

每个测试用例的第一行包含一个正整数，表示转方向的次数N(N<=1000)。

接下来的一行包含一个长度为N的字符串，由L和R组成，L表示向左转，R表示向右转。

输出描述:

输出牛牛最后面向的方向，N表示北，S表示南，E表示东，W表示西。

程序代码

```
    // 4. 迷路的牛牛（模拟）
    char[] dir = {'N','E','S','W'}; // 方向数组，对应旋转次序
    public void loseWay() {
```

```
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();   // 旋转方向的次数
        String s = sc.next();   // 旋转字符串
        int cur_dir = 0;            // 初始方向为北方0
        for(int i=0;i<n;i++) {
                Character rot = s.charAt(i);
                switch (rot) {
                    case 'L':cur_dir = cur_dir==0?3:cur_dir-1;break;    // 向左--
                    case 'R':cur_dir = cur_dir==3?0:cur_dir+1;break;    // 向右++
                }
        }
        System.out.println(dir[cur_dir]);
    }
```

例3：数对（5）【数学】

题目描述

牛牛以前在老师那里得到了一个正整数数对(x, y), 牛牛忘记他们具体是多少了。

但是牛牛记得老师告诉过他x和y均不大于n, 并且x除以y的余数大于等于k。

牛牛希望你能帮他计算一共有多少个可能的数对。

输入描述:

输入包括两个正整数n,k(1 <= n <= 10^5, 0 <= k <= n - 1)。

输出描述:

对于每个测试用例, 输出一个正整数表示可能的数对数量。

程序代码

```
// 5. 数对（数学）
// 牛牛以前在老师那里得到了一个正整数数对(x, y), 牛牛忘记他们具体是多少了。
// 但是牛牛记得老师告诉过他x和y均不大于n, 并且x除以y的余数大于等于k。
// 牛牛希望你能帮他计算一共有多少个可能的数对。
public void numberPair() {
    Scanner sc = new Scanner(System.in);
    long n = sc.nextLong(); // x、y均不大于n
    long k = sc.nextLong(); // x%y>=k
    long count = 0;
    if(k==0) {
            count = n*n;
    }
    else {
        // 利用数学规律进行计算：
        // 因为余数大于等于k，因此对于除数y而言，必须大于K（y从K+1开始遍历）
        // 所得余数为0...y-1，故对于除法每一完整周期（即余数从0...y-1），均有(y-k)个余数 >= k
        // 因此完整的周期数即对应余数共有 (n/y) * (y-k)
        // 对于最后一个周期，可能并不完整，故对最后一个周期单独讨论
        // 若最后一个周期的余数>=k，则最后一个周期对应>=k的余数个数为 n%y-k+1
    // 否则最后一个周期符合的余数为0
            for(long y=k+1;y<=n;y++)
                count += (n/y)*(y-k) + (n%y>=k?(n%y-k+1):0);
    }
    System.out.println(count);
}
```

例4：重叠矩阵（6）【数学】

题目描述

平面内有n个矩形, 第i个矩形的左下角坐标为(x1[i], y1[i]), 右上角坐标为(x2[i], y2[i])。

如果两个或者多个矩形有公共区域则认为它们是相互重叠的(不考虑边界和角落)。

请你计算出平面内重叠矩形数量最多的地方,有多少个矩形相互重叠。

输入描述:

输入包括五行。

第一行包括一个整数n(2 <= n <= 50), 表示矩形的个数。

第二行包括n个整数x1[i](-10^9 <= x1[i] <= 10^9),表示左下角的横坐标。

第三行包括n个整数y1[i](-10^9 <= y1[i] <= 10^9),表示左下角的纵坐标。

第四行包括n个整数x2[i](-10^9 <= x2[i] <= 10^9),表示右上角的横坐标。

第五行包括n个整数y2[i](-10^9 <= y2[i] <= 10^9),表示右上角的纵坐标。

输出描述:

输出一个正整数, 表示最多的地方有多少个矩形相互重叠,如果矩形都不互相重叠,输出1。

程序代码

```
// 6.重叠矩阵（数学）
// 平面内有n个矩形，第i个矩形的左下角坐标为(x1[i]，y1[i])，右上角坐标为(x2[i]，y2[i])。
// 如果两个或者多个矩形有公共区域则认为它们是相互重叠的(不考虑边界和角落)。
// 请你计算出平面内重叠矩形数量最多的地方,有多少个矩形相互重叠。
public void repeatRec() {
    // 无论何种情况，重叠区域也是四条边组成。
    // 而且是取自n个矩形中的四条。
    // 所以遍历边的交点,查看该交点包含几个矩阵即可。
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();   // 表示矩形个数
    int[] x1 = new int[n];  // 矩形左下坐标横坐标x
    int[] y1 = new int[n];  // 矩形左下坐标纵坐标y
    int[] x2 = new int[n];  // 矩形右上坐标横坐标x
    int[] y2 = new int[n];  // 矩形右上坐标纵坐标y
    for(int i=0;i<n;i++)x1[i] = sc.nextInt();
    for(int i=0;i<n;i++)y1[i] = sc.nextInt();
    for(int i=0;i<n;i++)x2[i] = sc.nextInt();
    for(int i=0;i<n;i++)y2[i] = sc.nextInt();
    int ans = 0;
    int cnt = 0;
    for(int i=0;i<x1.length;i++)
        for(int j=0;j<y1.length;j++) {
            // 对于某个可能的交点 (i,j)包含的矩阵数量
            // 考虑到若将平面按照所有矩形的的底边坐标值横向划分，每个划分中的最大重合情况总是出现在该划分底部
            // 重叠矩阵的左下交点一定由某个矩阵的左下x和某个矩阵的左下y所决定
            cnt = 0;
            for(int k=0;k<n;k++) {
```

```
                    if(x1[k]<=x1[i] && y1[k]<=y1[j] && x2[k]>x1[i] && y2[k]>y1[j])cnt++;
                }
                if(cnt>ans)ans = cnt;
            }
        System.out.println(ans);
    }
```

例5：牛牛的闹钟（7）【日期】

题目描述

牛牛总是睡过头，所以他定了很多闹钟，只有在闹钟响的时候他才会醒过来并且决定起不起床。从他起床算起他需要X分钟到达教室，上课时间为当天的A时B分，请问他最晚可以什么时间起床

输入描述:

每个输入包含一个测试用例。

每个测试用例的第一行包含一个正整数，表示闹钟的数量N(N<=100)。

接下来的N行每行包含两个整数，表示这个闹钟响起的时间为Hi(0<=A<24)时Mi(0<=B<60)分。

接下来的一行包含一个整数，表示从起床算起他需要X(0<=X<=100)分钟到达教室。

接下来的一行包含两个整数，表示上课时间为A(0<=A<24)时B(0<=B<60)分。

数据保证至少有一个闹钟可以让牛牛及时到达教室。

输出描述:

输出两个整数表示牛牛最晚起床时间。

程序代码

```
// 7. 牛牛的闹钟
// 牛牛总是睡过头，所以他定了很多闹钟，只有在闹钟响的时候他才会醒过来并且决定起不起床。
// 从他起床算起他需要X分钟到达教室，上课时间为当天的A时B分，请问他最晚可以什么时间起床
public class Alarm{
    public Integer hour;    // 小时
    public Integer minute;  // 分钟

    public Alarm(Integer _hour,Integer _minute) {
        hour = _hour;
        minute = _minute;
    }
}

public void getUpAlarm() {
    // 本题重点在于时间的转换
    // 理论最晚起床时间 = 上课时间 - 到达教室需要的时间
    // 再从大到小遍历闹钟时间，第一个小于 理论最晚起床时间 的即为闹铃时间
    ArrayList<Alarm> alarmList = new ArrayList<Alarm>();    //闹铃列表
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();   // 表示闹钟的数量
    for(int i=0;i<n;i++)        // 输入闹铃
            alarmList.add(new Alarm(sc.nextInt(),sc.nextInt()));
    int x  = sc.nextInt();  // 到达教室需要的时间
    int a = sc.nextInt();   // 上课时间（时）
    int b = sc.nextInt();   // 上课时间（分）

    Alarm theoryAlarm = declineDate(a,b,x);
    alarmList.add(theoryAlarm);
    Collections.sort(alarmList, new AlarmCmp());
    int i = 0;
    for(;i<alarmList.size();i++) {
            if(alarmList.get(i).hour == theoryAlarm.hour && alarmList.get(i).minute == theoryAlarm.minute)break;
    }
    System.out.println(alarmList.get(i+1).hour + " " + alarmList.get(i+1).minute);
}

public Alarm declineDate(Integer hour,Integer min,Integer time) {
    if(min >= time) {
        return new Alarm(hour,min-time);
    }else {
        return new Alarm(hour-1,min-time+60);
    }
}

// 闹铃的降序排序
public class AlarmCmp implements Comparator<Alarm>{
    @Override
    public int compare(Alarm a1, Alarm a2) {
        if(a2.hour != a1.hour)return a2.hour - a1.hour;
        else return a2.minute - a1.minute;
    }
}
```

例6：俄罗斯方块（9）【模拟】

题目描述

小易有一个古老的游戏机，上面有着经典的游戏俄罗斯方块。因为它比较古老，所以规则和一般的俄罗斯方块不同。

荧幕上一共有 n 列，每次都会有一个 1 x 1 的方块随机落下，在同一列中，后落下的方块会叠在先前的方块之上，当一整行方块都被占满时，这一行会被消去，并得到1分。

有一天，小易又开了一局游戏，当玩到第 m 个方块落下时他觉得太无聊就关掉了，小易希望你告诉他这局游戏他获得的分数。

输入描述:

第一行两个数 n, m

第二行 m 个数，c1, c2, ... , cm ， ci 表示第 i 个方块落在第几列

其中 1 <= n, m <= 1000, 1 <= ci <= n

输出描述:

小易这局游戏获得的分数

程序代码

```
// 9. 俄罗斯方块（模拟）
// 小易有一个古老的游戏机，上面有着经典的游戏俄罗斯方块。因为它比较古老，所以规则和一般的俄罗斯方块不同。
// 荧幕上一共有 n 列，每次都会有一个 1 x 1 的方块随机落下，在同一列中，后落下的方块会叠在先前的方块之上，当一整行方块都被占满时，这一行会被消去，并得到1分。
// 有一天，小易又开了一局游戏，当玩到第 m 个方块落下时他觉得太无聊就关掉了，小易希望你告诉他这局游戏他获得的分数。
public void Tetris() {
    // 记录每列落下的方块数目
    // 取所有列中落下方块的最小数目，即为消除的行数 = 小易获得分数
```

```java
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();   // 荧幕一共有n列
        int m = sc.nextInt();   // 一共掉落m个方块
        int[] game = new int[n+1];  // 定义列数组，game[i]表示第i列上有game[i]个方块
        // 初始化,每一列上方块数目为0
        for(int i=1;i<n+1;i++)game[i] = 0;
        for(int i=0;i<m;i++) {
            int idx = sc.nextInt();
            game[idx]++;
        }
        int min = Integer.MAX_VALUE;
        for(int i=1;i<n+1;i++)
            if(game[i]<min)min = game[i];
        System.out.println(min);
    }
```

例7：瞌睡（10）【模拟】

题目描述

小易觉得高数课太无聊了，决定睡觉。不过他对课上的一些内容挺感兴趣，所以希望你在老师讲到有趣的部分的时候叫醒他一下。你知道了小易对一堂课每分钟知识点的感兴趣程度，并以分数量化，以及他在这堂课上每分钟是否会睡着，你可以叫醒他一次，这会使得他在接下来的k分钟内保持清醒。你需要选择一种方案最大化小易这堂课听到的知识点分值。

输入描述:

第一行 n, k (1 <= n, k <= 105) ，表示这堂课持续多少分钟，以及叫醒小易一次使他能够保持清醒的时间。

第二行 n 个数，a1, a2, ... , an(1 <= ai <= 104) 表示小易对每分钟知识点的感兴趣评分。

第三行 n 个数，t1, t2, ... , tn 表示每分钟小易是否清醒, 1表示清醒。

输出描述:

小易这堂课听到的知识点的最大兴趣值。

程序代码

```java
    // 10.瞌睡（模拟）
    // 小易觉得高数课太无聊了，决定睡觉。
    // 不过他对课上的一些内容挺感兴趣，所以希望你在老师讲到有趣的部分的时候叫醒他一下。
    // 你知道了小易对一堂课每分钟知识点的感兴趣程度，并以分数量化，以及他在这堂课上每分钟是否会睡着。
    // 你可以叫醒他一次，这会使得他在接下来的k分钟内保持清醒。你需要选择一种方案最大化小易这堂课听到的知识点分值。
    int[] a;            // 每分钟知识点分值
    int[] t;            // 每分钟是否清醒
    int[] maxList;  // 记录在每个位置叫醒时可获得的最大分值
    public void WakeYiUp() {
        // 计算两部分，固定分值为保持清醒的分值
        // 继续遍历，计算连续k个中0对应的最大和， 才是叫醒额外获取的分值
        // 小易所获取最大总分值 = 固定分值 + 额外分值
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();   // 课持续时间
        int k = sc.nextInt();   // 叫醒一次保持清醒的时间
        a = new int[n];
        t = new int[n];
        maxList = new int[n-k+1];
        int cur_sum = 0;
        for(int i=0;i<n;i++)a[i] = sc.nextInt();
        for(int i=0;i<n;i++) {t[i] = sc.nextInt();cur_sum += t[i]==1?a[i]:0;}

        constructMaxList(n,k);
        Arrays.sort(maxList);
        System.out.println(maxList[n-k]+cur_sum);
    }

    public void constructMaxList(int n,int k) {
        // 在数组a[]中找到长为k的和最大的连续子串
        for(int i=0;i<n-k+1;i++) {
            int sum = 0;
            for(int j=0;j<k;j++)sum += (t[i+j]==0?a[i+j]:0);
            maxList[i] = sum;
        }
    }
```