001 【5 键键盘的输出】

有一个特殊的 5 键键盘，上面有 a，ctrl-c，ctrl-x，ctrl-v，ctrl-a 五个键。a 键在屏幕上输出一个字母 a；ctrl-c 将当前选择的字母复制到剪贴板；ctrl-x 将当前选择的字母复制到剪贴板，并清空选择的字母；ctrl-v 将当前剪贴板里的字母输出到屏幕；ctrl-a 选择当前屏幕上的所有字母。注意：

1 剪贴板初始为空，新的内容被复制到剪贴板时会覆盖原来的内容

2 当屏幕上没有字母时，ctrl-a 无效

3 当没有选择字母时，ctrl-c 和 ctrl-x 无效

4 当有字母被选择时，a 和 ctrl-v 这两个有输出功能的键会先清空选择的字母，再进行输出

给定一系列键盘输入，输出最终屏幕上字母的数量。

输入描述:

输入为一行，为简化解析，用数字 1 2 3 4 5 代表 a，ctrl-c，ctrl-x，ctrl-v，ctrl-a 五个键的输入，数字用空格分隔 输出描述:

输出一个数字，为最终屏幕上字母的数量

示例 1：

输入 1 1 1

输出 3

说明 连续键入 3 个 a，故屏幕上字母的长度为 3

示例 2：

输入 1 1 5 1 5 2 4 4

输出 2

说明 输入两个 a 后 ctrl-a 选择这两个 a，再输入 a 时选择的两个 a 先被清空，所以此时屏幕只有一个 a，后续的 ctrl-a，ctrl-c 选择并复制了这一个 a，最后两个 ctrl-v 在屏幕上输出两个 a，故屏幕上字母的长度为 2（第一个 ctrl-v 清空了屏幕上的那个 a）

```
public class ZT01 {
    private static String screen = "";
    private static String choose = "";
    private static String jianQie = "";
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] input = sc.nextLine().split(" ");
        for (int i = 0; i < input.length; i++) {
            operate(Integer.parseInt(input[i]));
        }
        System.out.println(screen.length());
    }//a

    public static void operate(int num){
        if (num == 1){//a screen 输入一个 a
            if (choose.equals("")){
                screen += "a";
            }else {
                choose = "";
                screen = "a";
            }
            return;
```

```
        }
        if (num == 2){//ctrl-c
            jianQie = choose;
            return;
        }
        if (num == 3){//ctrl-x
            jianQie = choose;
            choose = "";
            screen = "";
            return;
        }
        if (num ==4){//ctrl-v
            if (choose.equals("")){
                screen += jianQie;
            }else {
                screen = jianQie;
                choose = "";
            }
            return;
        }
        if (num == 5){//ctrl-a
            choose = screen;
        }
    }
}
```

【N进制减法】

实现一个基于字符串的N机制的减法。

需要对输入的两个字符串按照给定的N进制进行减法操作，输出正负符号和表示结果的字符串。

输入描述:

输入：三个参数。

第一个参数是整数形式的进制N值，N值范围为大于等于2、小于等于35。

第二个参数为被减数字符串；

第三个参数为减数字符串。有效的字符包括09以及小写字母az，字符串有效字符个数最大为100个字符，另外还有结尾的\0。

限制：

输入的被减数和减数，除了单独的0以外，不能是以0开头的字符串。

如果输入有异常或计算过程中有异常，此时应当输出-1表示错误。

输出描述:

输出：2个。

其一为减法计算的结果，-1表示出错，0表示结果为整数，1表示结果为负数。

其二为表示结果的字符串。

示例1:

输入

2 11 1

输出

0 10

说明

按二进制计算 11 -1 ，计算正常，0 表示符号为正数，结果为 10

示例 2:

输入

8 07 1

输出

-1

说明

按 8 进制，检查到减数不符合非 0 前导的要求，返回结果为-1，没有其他结果内容。

```
public class ZT02 {//2 11 1
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] input = sc.nextLine().split(" ");
        int jin = Integer.parseInt(input[0]);
        String numStr = input[1];
        String jianStr = input[2];
        //检查开头
        if (numStr.length() != 1 && jianStr.length() != 1 && (numStr.startsWith("0") || jianStr.startsWith("0"))){
            System.out.println(-1);
            return;
        }
        //检查结尾
        if (numStr.endsWith("/0")){
            numStr = numStr.substring(0,numStr.length()-2);
        }
        if (jianStr.endsWith("/0")){
            jianStr = jianStr.substring(0,jianStr.length()-2);
        }
        int no1 = 0;
        int no2 = 0;
        try {
            no1 = Integer.parseInt(numStr, jin);
            no2 = Integer.parseInt(jianStr, jin);
        }catch (Exception e){
            System.out.println(-1);
            return;
        }

        int res = no1 - no2;
        if (no1 - no2 > 0){
            System.out.print(0+" ");
```

```
        }else {
            System.out.print(1+" ");
        }
        System.out.println(Integer.toString(res, jin));
    }
}
```

003 【题目解析】

TLV 编码是按[Tag Length Value]格式进行编码的，一段码流中的信元用 Tag 标识，Tag 在码流中唯一不重复，Length 表示信元 Value 的长度，Value 表示信元的值。

码流以某信元的 Tag 开头，Tag 固定占一个字节，Length 固定占两个字节，字节序为小端序。

现给定 TLV 格式编码的码流，以及需要解码的信元 Tag，请输出该信元的 Value。

输入码流的 16 机制字符中，不包括小写字母，且要求输出的 16 进制字符串中也不要包含小写字母；码流字符串的最大长度不超过 50000 个字节。

输入描述:

输入的第一行为一个字符串，表示待解码信元的 Tag；

输入的第二行为一个字符串，表示待解码的 16 进制码流，字节之间用空格分隔。

输出描述:

输出一个字符串，表示待解码信元以 16 进制表示的 Value。

示例 1：

输入

31

32 01 00 AE 90 02 00 01 02 30 03 00 AB 32 31 31 02 00 32 33 33 01 00 CC

输出

32 33

说明

需要解析的信元的 Tag 是 31，从码流的起始处开始匹配，Tag 为 32 的信元长度为 1（01 00，小端序表示为 1）；第二个信元的 Tag 是 90，其长度为 2；第三个信元的 Tag 是 30，其长度为 3；第四个信元的 Tag 是 31，其长度为 2（02 00），所以返回长度后面的两个字节即可，即 32 33。

```
/**
 * TLV 解码
 * tag 一个字节 //第一行
 * length 占 2 个字节
 * 字节为小端序
 * 1 各字节占 8 位 length 16 位
 * 1 个 16 进制数 占 2 进制多少位？ 4 位 所以 length 占 4 个数
 *  * //32 00 01 AE
 * //90 00 02 01 02
 * //30 00 03 AB 32 31
 * //31 00 02 32 33
 * //33 0 01 CC
 *  */
public class ZT03 {
    public static void main(String[] args) {
```

```
        Scanner sc = new Scanner(System.in);
        String tag = sc.nextLine();//需要解析的 tag
        String stream = sc.nextLine();//所有的 tag 内容
        Map<String,Inner> map = new HashMap<>();
        //分割所有码流数据，塞到 map 中
        String[] input = stream.split(" ");
        int idx = 0;
        while (idx < input.length){
            String tagTemp = input[idx];
            String lengthStr = input[idx+2] + input[idx+1];
            idx += 3;
            //计算内容的长度
            int length = Integer.parseInt(lengthStr,16);
            StringBuilder content = new StringBuilder();
            for (int i = idx; i < idx + length; i++) {
                if (i == idx+length -1){
                    content.append(input[i]);
                }else {
                    content.append(input[i]).append(" ");
                }
            }
            map.put(tagTemp,new Inner(length,content.toString()));
            idx += length;
        }

        Inner inner = map.get(tag);
        System.out.println(inner.content);
    }

    static class Inner{
        private int length;
        private String content;

        public Inner(int length, String content) {
            this.length = length;
            this.content = content;
        }
    }
}
```

3038 【VLAN 资源池】

VLAN 是一种对局域网设备进行逻辑划分的技术，为了标识不同的 VLAN，引入 VLAN ID(1-4094 之间的整数)的概念。定义一个 VLAN ID 的资源池(下称 VLAN 资源池)，资源池中连续的 VLAN 用开始 VLAN-结束 VLAN 表示，不连续的用单个整数表示，所有的 VLAN

用英文逗号连接起来。现在有一个 VLAN 资源池，业务需要从资源池中申请一个 VLAN，需要你输出从 VLAN 资源池中移除申请的 VLAN 后的资源池。

输入描述:

第一行为字符串格式的 VLAN 资源池，第二行为业务要申请的 VLAN，VLAN 的取值范围为[1,4094]之间的整数。

输出描述:

从输入 VLAN 资源池中移除申请的 VLAN 后字符串格式的 VLAN 资源池，输出要求满足题目描述中的格式，并且按照 VLAN 从小到大升序输出。

如果申请的 VLAN 不在原 VLAN 资源池内，输出原 VLAN 资源池升序排序后的字符串即可。

示例 1：

输入

1-5

2

输出

1,3-5

说明

原 VLAN 资源池中有 VLAN 1、2、3、4、5，从资源池中移除 2 后，剩下 VLAN 1、3、4、5，按照题目描述格式并升序后的结果为 1,3-5。

示例 2：

输入

20-21,15,18,30,5-10

15

输出

5-10,18,20-21,30

说明

原 VLAN 资源池中有 VLAN 5、6、7、8、9、10、15、18、20、21、30，从资源池中移除 15 后，资源池中剩下的 VLAN 为 5、6、7、8、9、10、18、20、21、30，按照题目描述格式并升序后的结果为 5-10,18,20-21,30。

示例 3：

输入

5,1-3

10

输出

1-3,5

说明

原 VLAN 资源池中有 VLAN 1、2、3，5，申请的 VLAN 10 不在原资源池中，将原资源池按照题目描述格式并按升序排序后输出的结果为 1-3,5。

备注:

输入 VLAN 资源池中 VLAN 的数量取值范围为[2-4094]间的整数，资源池中 VLAN 不重复且合法([1,4094]之间的整数)，输入是乱序的。

```
public class ZT04 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] input = sc.nextLine().split(",");
        int search = sc.nextInt();
```

```
List<Integer> list = new ArrayList<>();
for (int i = 0; i < input.length; i++) {
    if (input[i].contains("-")){
        String[] num = input[i].split("-");
        int start = Integer.parseInt(num[0]);
        int end = Integer.parseInt(num[1]);
        for (int j = start; j <end+1 ; j++) {
            list.add(j);
        }
    }else {
        list.add(Integer.parseInt(input[i]));
    }
}

Collections.sort(list);
list.remove((Object) search);
//输出
int idx = 0;
int start = 0;//1 3 4 5
int tem = 1;
StringBuffer sb = new StringBuffer();
while (idx < list.size()){
    if (start == 0){
        start = list.get(idx);
    }
    if (idx+1 == list.size()){//保证下一位一定存在
        sb.append(start);
        break;
    }
    if (list.get(idx+1) == start+tem){//下一位与当前相等 继续后推
        idx ++;
        tem ++;
        continue;
    }else if (start == list.get(idx)){//输出当前值加,
        sb.append(start).append(",");
        start = 0;
        tem = 1;
    }else {
        sb.append(start).append("-").append(list.get(idx));
        start = 0;
        tem = 1;
        if (idx +1 < list.size()){//当前不是最后一个数据
            sb.append(",");
        }
```

```
            }
            idx ++;
        }
        System.out.println(sb);
    }
}
```

【按身高和体重排队】

某学校举行运动会，学生们按编号(1、2、3…n)进行标识，现需要按照身高由低到高排列，对身高相同的人，按体重由轻到重排列；对于身高体重都相同的人，维持原有的编号顺序关系。请输出排列后的学生编号。

输入描述:

两个序列，每个序列由 n 个正整数组成（0 < n <= 100）。第一个序列中的数值代表身高，第二个序列中的数值代表体重。

输出描述:

排列结果，每个数值都是原始序列中的学生编号，编号从 1 开始

示例 1：

输入

4

100 100 120 130

40 30 60 50

输出

2 1 3 4

说明

输出的第一个数字 2 表示此人原始编号为 2，即身高为 100，体重为 30 的这个人。由于他和编号为 1 的人身高一样，但体重更轻，因此要排在 1 前面。

示例 2：

输入

3

90 110 90

45 60 45

输出

1 3 2

说明

1 和 3 的身高体重都相同，需要按照原有位置关系让 1 排在 3 前面，而不是 3 1 2

```
public class ZT05 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int count = Integer.parseInt(sc.nextLine());
        String[] high = sc.nextLine().split(" ");
        String[] weight = sc.nextLine().split(" ");
        List<Player> playerList = new ArrayList<>();
        for (int i = 0; i < count; i++) {
            playerList.add(new Player(i+1,Integer.parseInt(high[i]),Integer.parseInt(weight[i])));
        }
```

```
        Collections.sort(playerList);
        for (int i = 0; i < playerList.size(); i++) {
            System.out.print(playerList.get(i).idx+" ");
        }
    }

    static class Player implements Comparable<Player>{
        private int idx;
        private int high;
        private int weight;

        public Player(int idx, int high, int weight) {
            this.idx = idx;
            this.high = high;
            this.weight = weight;
        }

        @Override
        public int compareTo(Player ply) {
            if (ply.high > this.high){
                return -1;
            }
            if (ply.weight > this.weight){
                return -1;
            }
            if (ply.idx > this.idx){
                return -1;
            }
            return 0;
        }
    }
}
```

【检索引范围翻转文章片段】

输入一个英文文章片段，翻转指定区间的单词顺序，标点符号和普通字母一样处理。例如输入字符串"I am a developer. “，区间[0,3]，则输出"developer. a am I"

String reverseWords(String s, int start, int end)

输入描述:

使用换行隔开三个参数，第一个参数为英文文章内容即英文字符串，第二个参数为翻转起始单词下标(下标从 0 开始)，第三个参数为结束单词下标。

输出描述:

翻转后的英文文章片段所有单词之间以一个半角空格分隔进行输出

示例 1：

输入

I am a developer.

1

2

输出

I a am developer.

示例 2：

输入

hello world!

0

1

输出

world! hello

说明

输入字符串可以在前面或者后面包含多余的空格，但是反转后的字符不能包括。

示例 3：

输入

I am a developer.

0

3

输出

developer. a am I

说明

如果两个单词间有多余的空格，将反转后单词间的空格减少到只含一个。

示例 4：

输入

Hello!

0

3

输出

EMPTY

说明

指定翻转区间只有一个单词或无有效单词，则统一输出"EMPTY"

```
public class ZT06 {
    private static String[] content;
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String input = sc.nextLine();
        String[] num = sc.nextLine().split(" ");
        System.out.println(reverseWords(input, Integer.parseInt(num[0]), Integer.parseInt(num[1])));
    }
    private static String reverseWords(String str,int start,int end){
        String[] content = str.split(" ");
        while (start < end){
```

```
            String temp = "";
            temp = content[end];
            content[end] = content[start];
            content[start] = temp;
            start++;
            end--;
        }
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < content.length; i++) {
            sb.append(content[i]);
            if (i+1 < content.length){
                sb.append(" ");
            }
        }
        return sb.toString();
    }
}
```

308 〖报数游戏〗

100 个人围成一圈，每个人有一个编码，编号从 1 开始到 100。他们从 1 开始依次报数，报到为 M 的人自动退出圈圈，然后下一个人接着从 1 开始报数，直到剩余的人数小于 M。请问最后剩余的人在原先的编号为多少？

输入描述:

输入一个整数参数 M

输出描述:

如果输入参数 M 小于等于 1 或者大于等于 100，输出“ERROR!”；否则按照原先的编号从小到大的顺序，以英文逗号分割输出编号字符串

示例 1：

输入

3

输出

58,91

说明

输入 M 为 3，最后剩下两个人

示例 2：

输入

4

输出

34,45,97

说明

输入 M 为 4，最后剩下三个人

```
public class ZT07 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
```

```
        int num = sc.nextInt();
        if (num <= 1 || num>= 100){
            System.out.println("ERROR!");
            return;
        }
        Map<Integer,Integer> map = new HashMap<>();
        for (int i = 1; i < 101; i++) {
            map.put(i,i);
        }
        int start = 1 ;
        while (map.size() >= num){
            Iterator<Map.Entry<Integer, Integer>> iterator = map.entrySet().iterator();
            while (iterator.hasNext()){
                Map.Entry<Integer, Integer> next = iterator.next();
                Integer key = next.getKey();
                map.put(key,start++);
                Integer value = next.getValue();
                if (value == num){
                    iterator.remove();
                    start = 1;
                }
            }
        }
        StringBuilder sb = new StringBuilder();
        for (Map.Entry<Integer, Integer> cou : map.entrySet()) {
            sb.append(cou.getKey()).append(",");
        }
        sb.deleteCharAt(sb.lastIndexOf(","));
        System.out.println(sb);
    }
}
```

〖比赛评分〗

一个有 N 个选手参加比赛，选手编号为 1~N（3<=N<=100），有 M（3<=M<=10）个评委对选手进行打分。打分规则为每个评委对选手打分，最高分 10 分，最低分 1 分。

请计算得分最多的 3 位选手的编号。如果得分相同，则得分高分值最多的选手排名靠前(10 分数量相同，则比较 9 分的数量，以此类推，用例中不会出现多个选手得分完全相同的情况)。

输入描述:

第一行为半角逗号分割的两个正整数，第一个数字表示 M（3<=M<=10）个评委，第二个数字表示 N（3<=N<=100）个选手。

第 2 到 M+1 行是半角逗号分割的整数序列，表示评委为每个选手的打分，0 号下标数字表示 1 号选手分数，1 号下标数字表示 2 号选手分数，依次类推。

输出描述:

选手前 3 名的编号。

注：若输入为异常，输出-1，如 M、N、打分不在范围内。

示例 1：

输入

4,5

10,6,9,7,6

9,10,6,7,5

8,10,6,5,10

9,10,8,4,9

输出

2,1,5

说明

第一行代表有 4 个评委，5 个选手参加比赛

矩阵代表是 4*5，每个数字是选手的编号，每一行代表一个评委对选手的打分排序，

2 号选手得分 36 分排第 1，1 号选手 36 分排第 2，5 号选手 30 分(2 号 10 分值有 3 个，1 号 10 分值只有 1 个，所以 2 号排第一)

示例 2：

输入

2,5

7,3,5,4,2

8,5,4,4,3

输出

-1

说明：只有 2 个评委，要求最少为 3 个评委

示例 3：

输入

4,2

8,5

5,6

10,4

8,9

输出

-1

说明：只有 2 名选手参加，要求最少为 3 名

示例 4：

输入

4,5

11,6,9,7,8

9,10,6,7,8

8,10,6,9,7

9,10,8,6,7

输出

-1

说明：第一个评委给第一个选手打分 11，无效分数

```
public class ZT08 {
    public static void main(String[] args) {
```

```
    Scanner sc = new Scanner(System.in);
    String[] num = sc.nextLine().split(",");
    int ma = Integer.parseInt(num[0]);//教练
    int no = Integer.parseInt(num[1]);//选手
    if (ma>10 || ma<3 || no>100 || no<3){
        System.out.println(-1);
        return;
    }
    List<String[]> list = new ArrayList<>();
    for (int i = 0; i < ma; i++) {
        list.add(sc.nextLine().split(","));
    }
    //收集选手信息
    List<Player> players = new ArrayList<>();
    for (int i = 0; i < no; i++) {//第 i 个选手
        int total = 0;
        List<Integer> listScore = new ArrayList<>();
        for (int j = 0; j < ma; j++) {//第 j 个裁判
            String[] strings = list.get(j);
            int score = Integer.parseInt(strings[i]);
            if (score<0 || score>10){
                System.out.println(-1);
                return;
            }
            listScore.add(score);
            total+= score;
        }
        players.add(new Player(i,total,listScore));
    }
    //比较选手分数
    Collections.sort(players);
    for (int i = 0; i < 3; i++) {
        if (i == 2){
            System.out.println(players.get(i).idx+1);
        }else {
            System.out.print(players.get(i).idx+1 + ",");
        }
    }

}
static class Player implements Comparable<Player>{
    private int idx;
    private int total;
    private List<Integer> scores;
```

```
        public Player(int idx,int total, List<Integer> scores) {
            this.idx = idx;
            this.total = total;
            this.scores = scores;
        }

        private int checkCount(List<Integer> list,int count){
            int cou = 0;
            for (int i = 0; i < list.size(); i++) {
                if (list.get(i) == count) {
                    cou++;
                }
            }
            return cou;
        }

        @Override
        public int compareTo(Player ply) {
            //先比较总分
            if (ply.total < this.total){
                return -1;
            }else if (ply.total > this.total){
                return 1;
            }else {
                //后比较最高分的数量
                List<Integer> scPly = ply.scores;
                List<Integer> scores = this.scores;
                for (int i = 10; i > 0; i--) {
                    int ipl = checkCount(scPly, i);
                    int ith = checkCount(scores, i);
                    if (ipl < ith){
                        return -1;
                    }
                }
            }
            return 0;
        }
    }
}
```

XXC 【查找接口成功率最优时间段】

服务之间交换的接口成功率作为服务调用关键质量特性，某个时间段内的接口失败率使用一个数组表示，数组中每个元素都是单位时间内失败率数值，数组中的数值为 0~100 的整数，给定一个数值(minAverageLost)表示某个时间段内平均失败率容忍值，即平均失败率小于等于 minAverageLost，找出数组中最长时间段，如果未找到则直接返回 NULL。

输入描述:

输入有两行内容，第一行为{minAverageLost}，第二行为{数组}，数组元素通过空格(" ")分隔，minAverageLost 及数组中元素取值范围为 0~100 的整数，数组元素的个数不会超过 100 个。

输出描述:

找出平均值小于等于 minAverageLost 的最长时间段，输出数组下标对，格式{beginIndex}-{endIndx}(下标从 0 开始)，如果同时存在多个最长时间段，则输出多个下标对且下标对之间使用空格(" ")拼接，多个下标对按下标从小到大排序。

示例 1：

输入

1

0 1 2 3 4

输出

0-2

说明

A、输入解释：minAverageLost=1，数组[0, 1, 2, 3, 4]

B、前 3 个元素的平均值为 1，因此数组第一个至第三个数组下标，即 0-2

示例 2：

输入

2

0 0 100 2 2 99 0 2

输出

0-1 3-4 6-7

说明

A、输入解释：minAverageLost=2，数组[0, 0, 100, 2, 2, 99, 0, 2]

B、通过计算小于等于 2 的最长时间段为：数组下标为 0-1 即[0, 0]，数组下标为 3-4 即[2, 2]，数组下标为 6-7 即[0, 2]，这三个部分都满足平均值小于等 2 的要求，因此输出 0-1 3-4 6-7

```
public class ZT09 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int num = Integer.parseInt(sc.nextLine());
        String[] arr0 = sc.nextLine().split(" ");
        int[] arr = new int[arr0.length];
        for (int i = 0; i < arr0.length; i++) {
            arr[i] = Integer.parseInt(arr0[i]);
        }
        int left = 0;
        int right = 0;
        StringBuilder sb = new StringBuilder();
        while (right < arr.length){
            if (left == right){
                right++;
```

```
            }
            if (checkAvMin(arr,left,right,num)){//需要查找到最大的数组？
                while (right <arr.length && checkAvMin(arr,left,right,num)){
                    right++;
                }
                right--;
                sb.append(left).append("-").append(right).append(" ");
                right++;
            }else {
                left++;
            }
        }
        System.out.println(sb);

    }

    //给出数组，求平均值是否小于等于某个期望值
    private static boolean checkAvMin(int[] arr,int start,int end,int target){
        double total = 0;
        for (int i = start; i < end+1; i++) {
            total += arr[i];
        }
        double res = total/(end - start +1);
        return res <= target;
    }
}
```

————————————————————

011 【查找众数及中位数】
012 【单词接龙】
013 【第k个排列】
014 【斗地主之顺子】
015 【非严格递增连续数字序列】
016 【分班】
017 【分糖果】
018 【高矮个子排队】
019 【工号不够用了怎么办】
020 【勾股数元组】

011 【查找众数及中位数】

磁盘容量排序

磁盘的容量单位常用的有 M，G，T 这三个等级，它们之间的换算关系为 1T = 1024G，1G = 1024M，现在给定 n 块磁盘的容量，请对它们按从小到大的顺序进行稳定排序，例如给定 5 块盘的容量，1T，20M，3G，10G6T，3M12G9M 排序后的结果为 20M，3G，3M12G9M，1T，10G6T。注意单位可以重复出现，上述 3M12G9M 表示的容量即为 3M+12G+9M，和 12M12G 相等。

输入描述:

输入第一行包含一个整数 n(2 <= n <= 100)，表示磁盘的个数，接下的 n 行，每行一个字符串(长度大于 2，小于 30)，表示磁盘的容量，由一个或多个格式为 mv 的子串组成，其中 m 表示容量大小，v 表示容量单位，例如 20M，1T，30G，10G6T，3M12G9M。磁盘容量 m 的范围为 1 到 1024 的正整数，容量单位 v 的范围只包含题目中提到的 M，G，T 三种，换算关系如题目描述。

输出描述:

输出 n 行，表示 n 块磁盘容量排序后的结果。

示例 1：

输入

3

1G

2G

1024M

输出

1G

1024M

2G

说明

1G 和 1024M 容量相等，稳定排序要求保留它们原来的相对位置，故 1G 在 1024M 之前

示例 2：

输入

3

2G4M

3M2G

1T

输出

3M2G

2G4M

1T

说明

1T 的容量大于 2G4M，2G4M 的容量大于 3M2G

在这里插入代码片

1

〖单词接龙〗

单词接龙的规则是：可用于接龙的单词首字母必须要前一个单词的尾字母相同；当存在多个首字母相同的单词时，取长度最长的单词，如果长度也相等，则取字典序最小的单词；已经参与接龙的单词不能重复使用。

现给定一组全部由小写字母组成单词数组，并指定其中的一个单词作为起始单词，进行单词接龙，请输出最长的单词串，单词串是单词拼接而成，中间没有空格。

输入描述:

输入的第一行为一个非负整数，表示起始单词在数组中的索引 K，0 <= K < N ；

输入的第二行为一个非负整数，表示单词的个数 N；

接下来的 N 行，分别表示单词数组中的单词。

输出描述:

输出一个字符串，表示最终拼接的单词串。

示例 1：

输入

0

6

word

dd

da

dc

dword

d

输出

worddworddа

说明

先确定起始单词 word，再接以 d 开头的且长度最长的单词 dword，剩余以 d 开头且长度最长的有 dd、da、dc，则取字典序最小的 da，所以最后输出 worddworddа。

示例 2：

输入

4

6

word

dd

da

dc

dword

d

输出

dworddа

说明

先确定起始单词 dword，剩余以 d 开头且长度最长的有 dd、da、dc，则取字典序最小的 da，所以最后输出 dworddа。

备注:

单词个数 N 的取值范围为[1, 20]；

单个单词的长度的取值范围为[1, 30]；

在这里插入代码片

1

目录

给定参数 n，从 1 到 n 会有 n 个整数：1,2,3,…,n，这 n 个数字共有 n! 种排列。

按大小顺序升序列出所有排列情况，并一一标记，当 n = 3 时，所有排列如下：

“123”

“132”

“213”

“231”

“312”

“321”

给定 n 和 k，返回第 k 个排列。

输入描述:

输入两行，第一行为 n，第二行为 k，给定 n 的范围是 [1,9]，给定 k 的范围是[1,n!]。

输出描述:

输出排在第 k 位置的数字。

示例 1：

输入

3

3

输出

213

说明

3 的排列有 123 132 213…，那么第 3 位置的为 213

示例 2：

输入

2

2

输出

21

说明

2 的排列有 12 21，那么第 2 位置的为 21

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int ni = sc.nextInt();//[1,9]
        int kLine = sc.nextInt();//[1,n!]
        List<Integer> list = new ArrayList<>();
        for (int i = 1; i <= ni; i++) {
            list.add(i);
        }
        int idx = ni;
```

```
        StringBuilder sb = new StringBuilder();
        kLine--;//数组中，存在下标0，顾--
        while (list.size()>0){
            int total = 1;
            for (int i = idx-1; i > 0; i--) {
                total *= i;
            }
            int index = kLine/total;
            kLine = Math.max(kLine - total, 0);
            int res = list.get(index);
            sb.append(res);
            list.remove(index);
            idx--;
        }
        System.out.println(sb);
    }
}
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
```

〖斗地主之顺子〗

在斗地主扑克牌游戏中， 扑克牌由小到大的顺序为：3,4,5,6,7,8,9,10,J,Q,K,A,2，玩家可以出的扑克牌阵型有：单张、对子、顺子、飞机、炸弹等。

其中顺子的出牌规则为：由至少 5 张由小到大连续递增的扑克牌组成，且不能包含 2。

例如：{3,4,5,6,7}、{3,4,5,6,7,8,9,10,J,Q,K,A}都是有效的顺子；而{J,Q,K,A,2}、 {2,3,4,5,6}、{3,4,5,6}、{3,4,5,6,8}等都不是顺子。

给定一个包含 13 张牌的数组，如果有满足出牌规则的顺子，请输出顺子。

如果存在多个顺子，请每行输出一个顺子，且需要按顺子的第一张牌的大小（必须从小到大）依次输出。

如果没有满足出牌规则的顺子，请输出 No。

输入描述:

13 张任意顺序的扑克牌，每张扑克牌数字用空格隔开，每张扑克牌的数字都是合法的，并且不包括大小王：

2 9 J 2 3 4 K A 7 9 A 5 6

不需要考虑输入为异常字符的情况

输出描述:

组成的顺子，每张扑克牌数字用空格隔开：

3 4 5 6 7

示例 1：

输入

2 9 J 2 3 4 K A 7 9 A 5 6

输出

3 4 5 6 7

说明

13 张牌中，可以组成的顺子只有 1 组：3 4 5 6 7

示例 2：

输入

2 9 J 10 3 4 K A 7 Q A 5 6

输出

3 4 5 6 7

9 10 J Q K A

说明

13 张牌中，可以组成 2 组顺子，从小到大分别为：3 4 5 6 7 和 9 10 J Q K A

示例 3：

输入

2 9 9 9 3 4 K A 10 Q A 5 6

输出

No

说明

13 张牌中，无法组成顺子

```
public class ZT13 {
    private static boolean flag = false;
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
```

```
String[] input = sc.nextLine().split(" ");
int[] arr = new int[input.length];
for (int i = 0; i < arr.length; i++) {
    String tem = input[i];
    if (tem.equals("A")){
        arr[i] = 14;
    }else if (tem.equals("J")){
        arr[i] = 11;
    }else if (tem.equals("Q")){
        arr[i] = 12;
    }else if (tem.equals("K")){
        arr[i] = 13;
    }else {
        arr[i] = Integer.parseInt(tem);
    }
}
Arrays.sort(arr);
//查找可以连成5张的顺子
int start = 0;
int temp = 0;
for (int i = 0; i < arr.length; i++) {
    if (arr[i] == 2){//2不能顺子
        continue;
    }
    if (start == 0){//初始化
        start = arr[i];
        temp = arr[i];
        continue;
    }
    if (arr[i] == temp){//对子等
        continue;
    }
    if (arr[i] == temp+1){
        temp = arr[i];
    }else {//不能连上了
        if (temp - start >= 4){
            print(start,temp);
        }
        start = 0;
        temp = 0;
    }
}
if (temp - start >= 4){
    print(start,temp);
```

```
        }
        if (!flag){
            System.out.println("NO");
        }
    }

    private static void print(int start,int end){
        flag = true;
        for (int j = start; j <= end ; j++) {
            if (j == 14){
                System.out.print("A ");
            }else if (j == 11){
                System.out.print("J ");
            }else if (j == 12){
                System.out.print("Q ");
            }else if (j == 13){
                System.out.print("K ");
            }else {
                System.out.print(j + " ");
            }
        }
        System.out.println();
    }
}
```

038【非严格递增连续数字序列】

输入一个字符串仅包含大小写字母和数字，求字符串中包含的最长的非严格递增连续数字序列的长度（比如 12234 属于非严格递增连续数字序列）。

输入描述:

输入一个字符串仅包含大小写字母和数字，输入的字符串最大不超过 255 个字符。

输出描述:

最长的非严格递增连续数字序列的长度

示例 1：

输入

abc2234019A334bc

输出

4

说明

2234 为最长的非严格递增连续数字序列，所以长度为 4。

```
public class ZT15 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String input = sc.nextLine();
        char[] chars = input.toCharArray();
```

```
        int start = 0;
        int end = 0;
        int max = 0;
        char tem = ' ';
        for (int i = 0; i < chars.length; i++) {
            char ch = chars[i];
            if (ch>= '0' && ch<='9'){
                if (start == 0){
                    start = i;
                    end = i;
                    tem = ch;
                    continue;
                }
                if (ch == tem + 1 || ch == tem){
                    tem = ch;
                    end++;
                }else {
                    max = Math.max(max,end - start+1);
                    start = 0;
                    end = 0;
                    tem = ' ';
                }
            }else {
                max = Math.max(max,end - start+1);
                start = 0;
                end = 0;
                tem = ' ';
            }
        }
        System.out.println(max);
    }
}
```

388【分糖】

幼儿园两个班的小朋友在排队时混在了一起，每位小朋友都知道自己是否与前面一位小朋友是否同班，请你帮忙把同班的小朋友找出来。

小朋友的编号为整数，与前一位小朋友同班用 Y 表示，不同班用 N 表示。

输入描述:

输入为空格分开的小朋友编号和是否同班标志。

比如：6/N 2/Y 3/N 4/Y，表示共 4 位小朋友，2 和 6 同班，3 和 2 不同班，4 和 3 同班。

其中，小朋友总数不超过 999，每个小朋友编号大于 0，小于等于 999。

不考虑输入格式错误问题。

输出描述:

输出为两行，每一行记录一个班小朋友的编号，编号用空格分开。且：

1、编号需要按照大小升序排列，分班记录中第一个编号小的排在第一行。

2、若只有一个班的小朋友，第二行为空行。

3、若输入不符合要求，则直接输出字符串 ERROR。

示例 1：

输入

1/N 2/Y 3/N 4/Y

输出

1 2

3 4

说明

2 的同班标记为 Y，因此和 1 同班。

3 的同班标记为 N，因此和 1、2 不同班。

4 的同班标记为 Y，因此和 3 同班。

所以 1、2 同班，3、4 同班，输出为

1 2

3 4

```
public class ZT16 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] input = sc.nextLine().split(" ");
        List<Integer> list1 = new ArrayList<>();
        List<Integer> list2 = new ArrayList<>();
        //0 号是 1 班 剩下是 2 班
        int preCla = 1;
        for (int i = 0; i < input.length; i++) {
            String[] num = input[i].split("/");
            if (i == 0){
                list1.add(Integer.parseInt(num[0]));
                preCla = 1;
                continue;
            }
            if (preCla == 1){
                if (num[1].equals("Y")){
                    list1.add(Integer.parseInt(num[0]));
                }else {
                    list2.add(Integer.parseInt(num[0]));
                    preCla = 2;
                }
            }else {//N
                if (num[1].equals("Y")){
                    list2.add(Integer.parseInt(num[0]));
                }else {
                    list1.add(Integer.parseInt(num[0]));
                    preCla = 1;
```

```
                }
            }
        }
        Collections.sort(list1);
        Collections.sort(list2);
        for (int i = 0; i < list1.size(); i++) {
            System.out.print(list1.get(i) + " ");
        }
        System.out.println();
        for (int i = 0; i < list2.size(); i++) {
            System.out.print(list2.get(i) + " ");
        }
    }
}
```

088【分糖果】

小明从糖果盒中随意抓一把糖果，每次小明会取出一半的糖果分给同学们。

当糖果不能平均分配时，小明可以选择从糖果盒中（假设盒中糖果足够）取出一个糖果或放回一个糖果。

小明最少需要多少次（取出、放回和平均分配均记一次），能将手中糖果分至只剩一颗

输入描述:

抓取的糖果数（<10000000000）：

15

输出描述:

最少分至一颗糖果的次数：

5

示例 1：

输入

15

输出

5

备注:

解释：(1)15+1=16;(2)16/2=8;(3)8/2=4;(4)4/2=2;(5)2/2=1;

```
public class ZT17 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int num = sc.nextInt();
        System.out.println(handle(num));

    }
    private static int handle(int tang){
        if (tang == 1){
            return 0;
        }
        if (tang % 2 == 0){
```

```
            tang /= 2;
            //次数+1
            return handle(tang)+1;
        }
        int plus = handle(tang +1) +1;//增加一个
        int sub = handle(tang -1) +1;//减少一个
        return Math.min(plus,sub);
    }
}
```

088【高矮个子排队】

现在有一队小朋友，他们高矮不同，我们以正整数数组表示这一队小朋友的身高，如数组{5,3,1,2,3}。

我们现在希望小朋友排队，以“高”“矮”“高”“矮”顺序排列，每一个“高”位置的小朋友要比相邻的位置高或者相等；

每一个“矮”位置的小朋友要比相邻的位置矮或者相等；

要求小朋友们移动的距离和最小，第一个从“高”位开始排，输出最小移动距离即可。

例如，在示范小队{5,3,1,2,3}中，{5, 1, 3, 2, 3}是排序结果。{5, 2, 3, 1, 3} 虽然也满足“高”“矮”“高”“矮”顺序排列，但小朋友们的移动距离大，所以不是最优结果。

移动距离的定义如下所示：

第二位小朋友移到第三位小朋友后面，移动距离为 1，若移动到第四位小朋友后面，移动距离为 2；

输入描述:

排序前的小朋友，以英文空格的正整数：

4 3 5 7 8

注：小朋友<100 个

输出描述:

排序后的小朋友，以英文空格分割的正整数：

4 3 7 5 8

示例 1：

输入

4 1 3 5 2

输出

4 1 5 2 3

示例 2：

输入

1 1 1 1 1

输出

1 1 1 1 1

说明

相邻位置可以相等

示例 3：

输入

xxx

输出

[ ]

说明：

出现非法参数情况， 返回空数组

备注:

4（高）3（矮）7（高）5（矮）8（高）， 输出结果为最小移动距离，只有 5 和 7 交换了位置，移动距离都是 1。

```
public class ZT18 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String input = sc.nextLine();
        String[] boys = input.split(" ");
        int[] arr = new int[boys.length];
        for (int i = 0; i < boys.length; i++) {
            arr[i] = Integer.parseInt(boys[i]);
        }
        int idx = 0;
        while (idx +1 < arr.length){
            //原则奇数位大 偶数位小
            if ((idx+1) % 2 == 0){//1 算奇数位 idx 要小
                if (arr[idx] > arr[idx+1]){
                    int tem = arr[idx+1];
                    arr[idx +1] = arr[idx];
                    arr[idx] = tem;
                }
            }else {//0 算奇数位 idx 要大
                if (arr[idx] < arr[idx+1]){
                    int tem = arr[idx+1];
                    arr[idx +1] = arr[idx];
                    arr[idx] = tem;
                }
            }
            idx++;
        }
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i]+" ");
        }
    }
}
```

088【工号不够用了怎么办】

3020 年，空间通信集团的员工人数突破 20 亿人，即将遇到现有工号不够用的窘境。

现在，请你负责调研新工号系统。继承历史传统，新的工号系统由小写英文字母（a-z）和数字（0-9）两部分构成。新工号由一段英文字母开头，之后跟随一段数字，比如"aaahw0001"，“a12345”，“abcd1”，“a00”。注意新工号不能全为字母或者数字，允许数字部分有前导 0 或者全为 0。

但是过长的工号会增加同事们的记忆成本，现在给出新工号至少需要分配的人数 X 和新工号中字母的长度 Y，求新工号中数字的最短长度 Z。

输入描述:

一行两个非负整数 X Y，用数字用单个空格分隔。

0< X <=2^50 - 1

0< Y <=5

输出描述:

输出新工号中数字的最短长度 Z

示例 1

输入

260 1

输出

1

示例 2

输入

26 1

输出

1

说明

数字长度不能为 0

示例 3

输入

2600 1

输出

2

090 【勾股数元组】

如果 3 个正整数(a,b,c)满足 a2 + b2 = c2 的关系，则称(a,b,c)为勾股数（著名的勾三股四弦五），为了探索勾股数的规律，我们定义如果勾股数(a,b,c)之间两两互质（即 a 与 b，a 与 c，b 与 c 之间均互质，没有公约数），则其为勾股数元祖（例如(3,4,5)是勾股数元祖，(6,8,10)则不是勾股数元祖）。请求出给定范围[N,M]内，所有的勾股数元祖。

输入描述:

起始范围 N，1 <= N <= 10000

结束范围 M，N < M <= 10000

输出描述:

a,b,c 请保证 a < b < c,输出格式：a b c；

多组勾股数元祖请按照 a 升序，b 升序，最后 c 升序的方式排序输出；

给定范围中如果找不到勾股数元祖时，输出”NA”。

示例 1：

输入

1

20

输出

3 4 5

5 12 13

8 15 17

说明

[1, 20]范围内勾股数有：(3 4 5)，(5 12 13)，(6 8 10)，(8 15 17)，(9 12 15)，(12 16 20)；其中，满足(a,b,c)之间两两互质的勾股数元祖有：(3 4 5)，(5 12 13)，(8 15 17)；按输出描述中顺序要求输出结果。

示例 2：

输入

5

10

输出

NA

说明

[5, 10]范围内勾股数有：(6 8 10)；其中，没有满足(a,b,c)之间两两互质的勾股数元祖； 给定范围中找不到勾股数元祖，输出”NA”。

```
public class ZT20 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int i1 = sc.nextInt();//[1,10000]
        int i2 = sc.nextInt();//[n,10000]
        List<List<Integer>> lists = new ArrayList<>();
        for (int i = i1; i <= i2 ; i++) {
            for (int j = i+1; j <= i2 ; j++) {
                if (checkFlag(i,j,i2)) {
                    List<Integer> list = new ArrayList<>();
                    list.add(i);
                    list.add(j);
                    list.add((int)Math.sqrt(i*i + j*j));
                    Collections.sort(list);
                    lists.add(list);
                }
            }
        }
        //所有的勾股数都在 lists 里,逐个排除存在公约数的数组
        int idx = 0;
        while (idx < lists.size()){
            List<Integer> list = lists.get(idx);
            if (checkYue(list.get(0),list.get(1),list.get(2))) {
                lists.remove(idx);
                idx = 0;
            }
            idx++;
        }
        for (int i = 0; i < lists.size(); i++) {
            List<Integer> list = lists.get(i);
            System.out.println(list.get(0) + " " + list.get(1) + " "+ list.get(2));
        }
    }
    private static boolean checkFlag(int a1,int a2,int max){
```

```
        int mul = a1 *a1 + a2 * a2;
        double res = Math.sqrt(mul);
        int sub = (int) res;
        return res * res  == sub * sub && sub<= max && sub>a2;
    }

    //369 246
    private static boolean checkYue(int a1,int a2,int a3){
        for (int i = 2; i < a3; i++) {
            if (a1 % i ==0 && a2 % i == 0){
                return true;
            }else if (a1 % i == 0 && a3 % i == 0){
                return true;
            }else if (a2 % i == 0 && a3 % i == 0){
                return true;
            }
        }
        return false;
    }
}
```

————————————————————

021 【喊7的次数重排】
022 【猴子爬山】
023 【滑动窗口最大和】
024 【火星文计算】
025 【计算面积】
026 【计算最大乘积】
027 【检查是否存在满足条件的数字组合】
028 【矩阵扩散】
029 【矩阵最大值】
030 【考勤信息】

022 【喊7的次数重排】

喊7是一个传统的聚会游戏，N个人围成一圈，按顺时针从1到N编号。编号为1的人从1开始喊数，下一个人喊的数字为上一个人的数字加1，但是当将要喊出来的数字是7的倍数或者数字本身含有7的话，不能把这个数字直接喊出来，而是要喊"过"。
假定玩这个游戏的N个人都没有失误地在正确的时机喊了"过"，当喊到数字K时，可以统计每个人喊"过"的次数。
现给定一个长度为N的数组，存储了打乱顺序的每个人喊"过"的次数，请把它还原成正确的顺序，即数组的第i个元素存储编号i的人喊"过"的次数。
输入描述:
输入为一行，为空格分隔的喊"过"的次数，注意K并不提供，K不超过200，而数字的个数即为N。
输出描述:
输出为一行，为顺序正确的喊"过"的次数，也由空格分隔。

示例 1：

输入

0 1 0

输出

1 0 0

说明

一共只有一次喊"过"，那只会发生在需要喊 7 时，按顺序，编号为 1 的人会遇到 7，故输出 1 0 0。注意，结束时的 K 不一定是 7，也可以是 8、9 等，喊过的次数都是 1 0 0。

示例 2：

输入

0 0 0 2 1

输出

0 2 0 1 0

说明

一共有三次喊"过"，发生在 7 14 17，按顺序，编号为 2 的人会遇到 7 17，编号为 4 的人会遇到 14，故输出 0 2 0 1 0。

```
public class ZT21 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] input = sc.nextLine().split(" ");
        int[] arr = new int[input.length+1];
        int total = 0;
        for (int i = 0; i < input.length; i++) {
            total += Integer.parseInt(input[i]);
        }
        int temp = 0;
        for (int i = 1; i < 200; i++) {
            if (checkSeven(i)){
                arr[i% input.length]++;
                temp++;
            }
            if (temp>= total){
                break;
            }
        }
        for (int i = 1; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
        //7 14 17 21 27 28
    }

    private static boolean checkSeven(int num){
        //1、是不是 7 的倍数
        if (num % 7 == 0){
```

```
            return true;
        }
        //2、是不是数字包含 7 -> 拆解所有位
        while (num > 0){
            int wi = num % 10;
            if (wi == 7){
                return true;
            }
            num /= 10;
        }
        return false;
    }
}
```

022【猴子爬山】

一天一只顽猴想去从山脚爬到山顶，途中经过一个有个 N 个台阶的阶梯，但是这猴子有一个习惯： 每一次只能跳 1 步或跳 3 步，试问猴子通过这个阶梯有多少种不同的跳跃方式？

输入描述:

输入只有一个整数 N（0<N<=50）此阶梯有多少个阶梯

输出描述:

输出有多少种跳跃方式（解决方案数）

示例 1：

输入

50

输出

122106097

示例 2：

输入

3

输出

2

```
public class ZT22 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int input = sc.nextInt();
        System.out.println(calcCount(input));
    }
    private static int calcCount(int step){
        //只剩一个台阶了
        if (step < 3){
            return 1;
        }
        //剩下的场景都需要分成当前跳 1 步或者跳 3 步
        return calcCount(step -1) + calcCount( step -3);
```

```
    }
}
```

038【滑动窗口最大和】

有一个 N 个整数的数组，和一个长度为 M 的窗口，窗口从数组内的第一个数开始滑动直到窗口不能滑动为止，每次窗口滑动产生一个窗口和（窗口内所有数的和），求窗口滑动产生的所有窗口和的最大值。

输入描述:

第一行输入一个正整数 N，表示整数个数。(0<N<100000)

第二行输入 N 个整数，整数的取值范围为[-100,100]。

第三行输入一个正整数 M，M 代表窗口大小，M<=100000，且 M<=N。

输出描述:

窗口滑动产生的所有窗口和的最大值。

示例 1：

输入

6

10 20 30 15 23 12

3

输出

68

说明

窗口长度为 3，窗口滑动产生的窗口和分别为 10+20+30=60，20+30+15=65，30+15+23=68，15+23+12=50，所以窗口滑动产生的所有窗口和的最大值为 68。

```
public static void main(String[] args) {
    int len = 7;
    int step = 3;
    int[] arr = new int[]{10, 20, 30, 15, 23, 12, 100};
    int max = 0;
    for (int i = 0; i + 2 < len; i++) {
        int temp = arr[i];
        step = 3;
        while (step > 1) {
            step--;
            temp += arr[i + step];
        }
        if (temp > max) {
            max = temp;
        }
    }
    System.out.println(max);
}
```

```
public class ZT23 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int count = Integer.parseInt(sc.nextLine());
```

```
        String[] input = sc.nextLine().split(" ");
        int windows = Integer.parseInt(sc.nextLine());
        int[] arr = new int[count];
        for (int i = 0; i < count; i++) {
            arr[i] = Integer.parseInt(input[i]);
        }
        int left = 0;
        int right = 0;
        int total = 0;
        int max = 0;
        while (right<count){
            total += arr[right];
            if (right - left < windows){
                right++;
            }else {
                total -= arr[left];
                left++;
                right++;
            }
            max = Math.max(max,total);
        }
        System.out.println(max);
    }
}
```

【火星文计算】

已知火星人使用的运算符为#、$，其与地球人的等价公式如下:

x#y=2*x+3*y+4

x$y=3x+y+2

1、其中 x、y 是无符号整数

2、地球人公式按 C 语言规则计算

3、火星人公式中，KaTeX parse error: Expected 'EOF', got '#' at position 7: 的优先级高于#，相同的运算符，按从左到右的顺…）组成的计算表达式。例如：123#4$5#67$78。

1、用例保证字符串中，操作数与操作符之间没有任何分隔符。

2、用例保证操作数取值范围为 32 位无符号整数。

3、保证输入以及计算结果不会出现整型溢出。

4、保证输入的字符串为合法的求值报文，例如：123#4$5#67$78

5、保证不会出现非法的求值报文，例如类似这样字符串：

#4$5 //缺少操作数

4$5# //缺少操作数

4#$5 //缺少操作数

4 $5 //有空格

3+4-56/7 //有其它操作符

123456789876543 21$54321 //32 位整数计算溢出

输出描述:

根据输入的火星人字符串输出计算结果（结尾不带回车换行）

示例 1

输入

7#6$5#12

输出

226

说明

示例：

7#6$5#12

=7#(18+5+2)#12

=7#25#12

=(14+75+4)#12

=93#12

=293+312+4

=226

```
public class Test1 {
  public static void main(String[] args) {
//       Scanner sc = new Scanner(System.in);
//       while (sc.hasNext()) {
//          String input = sc.nextLine();
//          System.out.println(func(input));
//       }
//       sc.close();
    System.out.print(func("7#6$5#12"));
  }

  public static String func(String input) {
    String temp = input;
    String regex = "\\d{1,}[$]\\d{1,}";
    Pattern pattern = Pattern.compile(regex);
    Matcher matcher = pattern.matcher(temp);
    while (matcher.find()) {
      String group = matcher.group();
      String[] split = group.split("\\$");
      String value = String.valueOf(getNum(
        false, Integer.parseInt(split[0]), Integer.parseInt(split[1])));
      temp = temp.replace(group, value);
    }
    regex = "\\d{1,}[#]\\d{1,}";
    pattern = Pattern.compile(regex);
    matcher = pattern.matcher(temp);
    while (matcher.find()) {
      String group = matcher.group();
      String[] split = group.split("#");
```

```
      String value = String.valueOf(getNum(
        true, Integer.parseInt(split[0]), Integer.parseInt(split[1])));
      if (temp.equalsIgnoreCase(group)) {
        temp = value;
      } else {
        temp = temp.replace(group, value);
      }
    }
    if (temp.contains("#")) {
      String[] split = temp.split("#");
      String value = String.valueOf(getNum(
        true, Integer.parseInt(split[0]), Integer.parseInt(split[1])));
      return value;
    }
    return temp;
  }

  public static int getNum(boolean isSharp, int num1, int num2) {
    if (isSharp) {
      return 2 * num1 + 3 * num2 + 4;
    } else {
      return 3 * num1 + num2 + 2;
    }
  }
}
```

```
public class ZT24 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String input = sc.nextLine();
        while (input.contains("$") || input.contains("#")){
            if (input.contains("$")) {//字符串左取右不取 7#6$5#12
                int idx = input.lastIndexOf("$");
                int leftStart = changeInput(input, idx, -1);
                int op1 = Integer.parseInt(input.substring(leftStart,idx));
                int rightEnd = changeInput(input, idx, +1);
                int op2 = Integer.parseInt(input.substring(idx +1,rightEnd));
                int res = calcStar('$', op1, op2);
                input = input.substring(0,leftStart) + res + input.substring(rightEnd);
            }else {
                if (input.contains("#")) {
                    int idx = input.indexOf("#",0);
                    int leftStart = changeInput(input, idx, -1);
```

```
                    int op1 = Integer.parseInt(input.substring(leftStart,idx));
                    int rightEnd = changeInput(input, idx, +1);
                    int op2 = Integer.parseInt(input.substring(idx +1,rightEnd));
                    int res = calcStar('#', op1, op2);
                    input = input.substring(0,leftStart) + res + input.substring(rightEnd);
                }
            }
        }
        System.out.println(input);
    }

    //opt -1 表示左边的整数 1 表示右边的整数 7#6$5#12 leftStart  rightEnd
    private static int changeInput(String input,int idx,int opt){
        char ch = input.charAt(idx + opt);
        while (ch >= '0' && ch<= '9'){
            if (opt>0){
                opt++;
            }else {
                opt--;
            }
            if (idx + opt >=0 && idx + opt < input.length()){
                ch = input.charAt(idx + opt);
            }else {
                break;
            }
        }
        if (opt<0){
            return opt + idx +1;
        }else {
            return opt + idx;
        }
    }

    private static int calcStar(char ch, int op1,int op2){
        if (ch == '#'){
            return op1 * 2 + op2 * 3 + 4;
        }else {
            return 3 * op1 + op2 + 2;
        }
    }
}
```

088 【计算面积】

绘图机器的绘图笔初始位 i 在原点(0.0)。 机器启动后其绘图笔按下面规则绘制直线:

1)尝试沿着横向坐标轴正向绘制直线，直到给定的终点值 E,

2)期间可通过指令在纵坐标轴方向进行偏移。并同时绘制直线，偏移后按规则1绘制直线;指令的格式为X offsetY。表示在横坐标X沿纵坐标方向偏移，offsetY为正数表示正向偏移,为负数表示负向偏移。

给定了横坐标终点值E.以及若干条绘制指令。请计算给制的直线和横坐标轴。以及X-E的直线组成图形的面积。

输入模述:

首行为两个整数NE。表示有N条指令。机器运行的横坐标終点值E.

接下来N行。每行两个整数表示-条给制指令x osorr。用例保证横坐标X以递增排序方式出现。且不会出现相同横坐标义。取值范围:0<N<= 10000, 0<X<= E<=20000, -10000<=offsetY<=10000.

输出描述:

一个整数，表示计算得到的面积。用例保证.结果范围在0-4294967295内

示例1:

输入

4 10

1 1

2 1

3 1

4 -2

输出

12

```
public class ZT25 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] input = sc.nextLine().split(" ");
        int count = Integer.parseInt(input[0]);
        int maxX1 = Integer.parseInt(input[1]);
        int sqa = 0;
        int tempX1 = 0;
        int tempY1 = 0;
        for (int i = 0; i < count; i++) {
            String[] str = sc.nextLine().split(" ");
            int x1New = Integer.parseInt(str[0]);
            sqa += (x1New - tempX1) * Math.abs(tempY1);
            tempY1 += Integer.parseInt(str[1]);
            tempX1 = x1New;
        }
        sqa += (maxX1 - tempX1) * tempY1;
        System.out.println(sqa);
    }
}
```

088【计算最大乘积】

给定一个元素类型为小写字符串的数组，请计算两个没有相同字符的元素 长度乘积的最大值，如果没有符合条件的两个元素，返回0。

输入描述:

输入为一个半角逗号分隔的小写字符串的数组，2 <= 数组长度<=100，0 < 字符串长度<= 50。

输出描述:

两个没有相同字符的元素 长度乘积的最大值。

示例 1

输入

iwdvpbn,hk,iuop,iikd,kadgpf

输出

14

说明

数组中有 5 个元素。

iwdvpbn 与 hk 无相同的字符，满足条件，iwdvpbn 的长度为 7，hk 的长度为 2，乘积为 14（7*2）。

iwdvpbn 与 iuop、iikd、kadgpf 均有相同的字符，不满足条件。

iuop 与 iikd、kadgpf 均有相同的字符，不满足条件。

iikd 与 kadgpf 有相同的字符，不满足条件。

因此，输出为 14。

```
public class ZT26 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] split = sc.nextLine().split(",");
        int max = 0;
        for (int i = 0; i < split.length; i++) {
            for (int j = i+1; j < split.length; j++) {
                if (!checkHaveSame(split[i], split[j])){
                    max = Math.max(max,split[i].length() * split[j].length());
                }
            }
        }
        System.out.println(max);
    }

    private static boolean checkHaveSame(String str1,String str2){
        //字符串排序
        char[] chars1 = str1.toCharArray();
        char[] chars2 = str2.toCharArray();
        Arrays.sort(chars1);
        Arrays.sort(chars2);
        int idx1 = 0;
        int idx2 = 0;
        while (idx1<str1.length() && idx2< str2.length()){
            if (chars1[idx1] == chars2[idx2]){
                return true;
            }else if (chars1[idx1] > chars2[idx2]){
                idx2++;
```

```
            }else if (chars1[idx1] < chars2[idx2]){
                idx1++;
            }
        }
        return false;
    }
}
```

098【检查是否存在满足条件的数字组合】

给定一个正整数数组，检查数组中是否存在满足规则的数字组合

规则：

A = B + 2C

输入描述:

第一行输入数组的元素个数。

接下来一行输入所有数组元素，用空格隔开。

输出描述:

如果存在满足要求的数，在同一行里依次输出规则里 A/B/C 的取值，用空格隔开。

如果不存在，输出 0。

示例 1：

输入

4

2 7 3 0

输出

7 3 2

说明

7 = 3 + 2 * 2

示例 2：

输入

3

1 1 1

输出

0

说明

找不到满足条件的组合

备注:

数组长度在 3-100 之间。

数组成员为 0-65535，数组成员可以重复，但每个成员只能在结果算式中使用一次。如：数组成员为[0, 0, 1, 5]，0 出现 2 次是允许的，但结果 0 = 0 + 2 * 0 是不允许的，因为算式中使用了 3 个 0。

用例保证每组数字里最多只有一组符合要求的解。

```
public class ZT27 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int count = Integer.parseInt(sc.nextLine());
        int[] arr = new int[count];
```

```
        String[] input = sc.nextLine().split(" ");
        for (int i = 0; i < count; i++) {
            arr[i] = Integer.parseInt(input[i]);
        }
        Arrays.sort(arr);
        for (int i = 0; i < count; i++) {
            for (int j = i + 1; j < count; j++) {
                if (checkAdd(arr,i,j)) {
                    return;
                }
            }
        }
        System.out.println(0);
    }

    private static boolean checkAdd(int[] arr,int idx1,int idx2){//a
        int sub1 = 2 * arr[idx1] + arr[idx2];
        int sub2 = arr[idx1] + 2 * arr[idx2];
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == sub1 && i> idx1 && i> idx2) {
                System.out.println(sub1 + " " + arr[idx2] + " " + arr[idx1]);
                return true;
            }
            if (i> idx1 && i> idx2 && arr[i] == sub2) {
                System.out.println(sub1 + " " + arr[idx1] + " " + arr[idx2]);
                return true;
            }
        }
        return false;
    }
}
1
2
3
4
5
6
7
8
9
10
11
12
13
```

14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36

028 【矩阵扩散】

存在一个 mn 的二维数组，其成员取值范围为 0 或 1。其中值为 1 的成员具备扩散性，每经过 1S，将上下左右值为 0 的成员同化为 1。二维数组的成员初始值都为 0，将第[i,j]和[k,l]两个个位置上元素修改成 1 后，求矩阵的所有元素变为 1 需要多长时间。

输入描述:

输出数据中的前 2 个数字表示这是一个 mn 的矩阵，m 和 n 不会超过 1024 大小；中间两个数字表示一个初始扩散点位置为 i,j；最后 2 个数字表示另一个扩散点位置为 k,l。

输出描述:

输出矩阵的所有元素变为 1 所需要秒数。

示例 1：

输入

4,4,0,0,3,3

输出

3

说明

输出数据中的前 2 个数字表示这是一个 4*4 的矩阵；中间两个数字表示一个初始扩散点位置为 0,0；最后 2 个数字表示另一个扩散点位置为 3,3。

给出的样例是一个很简单模型，初始点在对角线上，达到中间的位置分别为 3 次迭代，即 3 秒。所以输出为 3。

```
public class ZT28 {
    private static int ml;
    private static int nl;
```

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    String[] split = sc.nextLine().split(",");
    m1 = Integer.parseInt(split[0]);
    n1 = Integer.parseInt(split[1]);
    int[][] arr = new int[m1][n1];
    int x1 = Integer.parseInt(split[2]);
    int y1 = Integer.parseInt(split[3]);
    int x2 = Integer.parseInt(split[4]);
    int y2 = Integer.parseInt(split[5]);
    arr[x1][y1] = 1;
    arr[x2][y2] = 1;
    System.out.println(calcSecond(arr));
}
private static int calcSecond(int[][] arr){
    int used = 0;
    while (checkHasZero(arr)){
        int[][] arrTemp = new int[m1][n1];
        for (int i = 0; i < m1; i++) {
            for (int j = 0; j < n1; j++) {
                if (arr[i][j] == 1) {
                    arrTemp[i][j] = 1;
                    //上下左右
                    if (i-1 >= 0){
                        arrTemp[i-1][j] = 1;
                    }
                    if (i+1 < m1){
                        arrTemp[i+1][j] = 1;
                    }
                    if (j-1 >= 0){
                        arrTemp[i][j-1] = 1;
                    }
                    if (j+1 < n1){
                        arrTemp[i][j+1] = 1;
                    }
                }
            }
        }
        arr = arrTemp;
        used++;
    }
    return used;
}
```

```
    private static boolean checkHasZero(int[][] arr){
        for (int i = 0; i < m1; i++) {
            for (int j = 0; j < n1; j++) {
                if (arr[i][j] == 0) {
                    return true;
                }
            }
        }
        return false;
    }
}
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
```

34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58

029 【矩阵最大值】

给定一个仅包含 0 和 1 的 N*N 二维矩阵，请计算二维矩阵的最大值，计算规则如下：

1、 每行元素按下标顺序组成一个二进制数（下标越大越排在低位），二进制数的值就是该行的值。矩阵各行值之和为矩阵的值。

2、允许通过向左或向右整体循环移动每行元素来改变各元素在行中的位置。

比如： [1,0,1,1,1]向右整体循环移动 2 位变为[1,1,1,0,1]，二进制数为 11101，值为 29。

[1,0,1,1,1]向左整体循环移动 2 位变为[1,1,1,1,0]，二进制数为 11110，值为 30。

输入描述:

1、输入的第一行为正整数，记录了 N 的大小，0 < N <= 20。

2、输入的第 2 到 N+1 行为二维矩阵信息，行内元素边角逗号分隔。

输出描述:

矩阵的最大值。

示例 1：

输入

5

1,0,0,0,1

0,0,0,1,1

0,1,0,1,0

1,0,0,1,1

1,0,1,0,1

输出

122

说明

第一行向右整体循环移动 1 位，得到本行的最大值[1,1,0,0,0]，二进制值为 11000，十进制值为 24。

第二行向右整体循环移动 2 位，得到本行的最大值[1,1,0,0,0]，二进制值为 11000，十进制值为 24。

第三行向左整体循环移动 1 位，得到本行的最大值[1,0,1,0,0]，二进制值为 10100，十进制值为 20。

第四行向右整体循环移动 2 位，得到本行的最大值[1,1,1,0,0]，二进制值为 11100，十进制值为 28。

第五行向右整体循环移动 1 位，得到本行的最大值[1,1,0,1,0]，二进制值为 11010，十进制值为 26。

因此，矩阵的最大值为 122。

```
public class ZT29 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int num = Integer.parseInt(sc.nextLine());
        //如何把 1 全部移动到高位 0 到低位
        int total = 0;
        for (int i = 0; i < num; i++) {
            String[] split = sc.nextLine().split(",");
            StringBuilder sb = new StringBuilder();
            for (int j = 0; j < num; j++) {
                sb.append(split[j]);
            }
            total += calcMax(sb.toString(), num);
        }
        System.out.println(total);
    }

    //最多移动 num 次
    private static int calcMax(String str,int num){
        int max = Integer.parseInt(str,2);
        while (num>1){
            String tem = str.substring(1) + str.charAt(0);
            max = Math.max(max,Integer.parseInt(tem,2));
            str = tem;
            num--;
        }
        return max;
    }
}
1
2
```

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

030 【考勤信息】

公司用一个字符串来表示员工的出勤信息：

absent：缺勤

late：迟到

leaveearly：早退

present：正常上班

现需根据员工出勤信息，判断本次是否能获得出勤奖，能获得出勤奖的条件如下：

缺勤不超过一次；没有连续的迟到/早退；任意连续 7 次考勤，缺勤/迟到/早退不超过 3 次

输入描述:

用户的考勤数据字符串，记录条数 >= 1；输入字符串长度<10000；不存在非法输入

如：

2

present

present absent present present leaveearly present absent

输出描述:

根据考勤数据字符串，如果能得到考勤奖，输出"true"；否则输出"false"，对于输入示例的结果应为：

true false

示例 1：

输入

2

present

present present

输出

true true

示例 2：

输入

2

present

present absent present present leaveearly present absent

输出

true false

————————————————————

031 【靠谱的车】

032 【快递运输】

033 【连续字母长度】

034 【两数之和绝对值最小】

035 【流水线】

036 【内存资源分配】

037 【判断一组不等式是否满足约束并输出最大差】

038 【判断字符串子序列】

039 【拼接 URL】

040 【求符合要求的结对方式】

031 【靠谱的车】

程序员小明打了一辆出租车去上班。出于职业敏感，他注意到这辆出租车的计费表有点问题，总是偏大。

出租车司机解释说他不喜欢数字 4，所以改装了计费表，任何数字位置遇到数字 4 就直接跳过，其余功能都正常。

比如：

23 再多一块钱就变为 25；

39 再多一块钱变为 50；

399 再多一块钱变为 500；

小明识破了司机的伎俩，准备利用自己的学识打败司机的阴谋。

给出计费表的表面读数，返回实际产生的费用。

输入描述:

只有一行，数字 N，表示里程表的读数。

(1<=N<=888888888)。

输出描述:

一个数字，表示实际产生的费用。以回车结束。

示例 1：

输入

5

输出

4

说明

5 表示计费表的表面读数。

4 表示实际产生的费用其实只有 4 块钱。

示例 2：

输入

17

输出

15

说明

17 表示计费表的表面读数。

15 表示实际产生的费用其实只有 15 块钱。

示例 3：

输入

100

输出

81

说明

100 表示计费表的表面读数。

81 表示实际产生的费用其实只有 81 块钱。

```
public class ZT31 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int input = sc.nextInt();
        int reduce = 0;
        int idx = 0;
        while (idx < input){
            int next = idx +1 ;
            String temp = String.valueOf(next);
            String newTemp = "";
            if (temp.contains("4")) {
                int fourIdx = temp.indexOf("4");
                if (fourIdx == temp.length()-1){
                    newTemp = temp.substring(0,fourIdx) + 5;
                }else {
                    newTemp = temp.substring(0,fourIdx) + 5 + temp.substring(fourIdx+1);
                }
                reduce += Integer.parseInt(newTemp) - idx -1;
                idx = Integer.parseInt(newTemp);
            }else {
                idx++;
```

```
            }
        }
        System.out.println(input - reduce);
    }
```

上述算法遇到大位数据会超时，优化如下

```
public class ZT3102 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int input = in.nextInt();
       //求司机多算了多少[0,10][10,100][100,1000]
        int temp = 0;
        int total = input;
        int k = 0;//记录当前位
        int j = 1;//记录个位？
        //13
        while (total > 0){
            if (total % 10 >4) {//当前位大于4
                temp += (total % 10 - 1) * k + j ;
```

```
            }else {//当前位小于4
                temp += (total % 10) * k;
            }
            k = k * 9 + j;
            j *= 10;
            total = total/10;
        }
        System.out.println(input - temp);
    }
}
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23

032 【快递运输】

一辆运送快递的货车，运送的快递均放在大小不等的长方体快递盒中，为了能够装载更多的快递，同时不能让货车超载，需要计算最多能装多少个快递。

注：快递的体积不受限制，快递数最多 1000 个，货车载重最大 50000。

输入描述:

第一行输入每个快递的重量，用英文逗号分隔，如：5,10,2,11

第二行输入货车的载重量，如：20

不需要考虑异常输入。

输出描述:

输出最多能装多少个快递，如：3

示例 1：

输入

5,10,2,11

20

输出

3

说明

货车的载重量为20，最多只能放三个快递5、10、2，因此输出3

```
public class ZT32 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] input = sc.nextLine().split(",");//2 5 10 11 2510
        int max = Integer.parseInt(sc.nextLine());//20
        int[] arr = new int[input.length];
        for (int i = 0; i < input.length; i++) {
            arr[i] = Integer.parseInt(input[i]);
        }
        Arrays.sort(arr);
        System.out.println(calc(arr,0,0,0,max));
    }

    private static int calc(int[] arr,int idx,int count,int total,int max){
        if (total > max || idx>= arr.length){
            return count-1;
        }
        int cl1 = calc(arr, idx+1, count+1,total + arr[idx],max);
        int cl2 = calc(arr, idx+1, count, total,max);
        return Math.max(cl1,cl2);
    }
}
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

15

16

17

18

19

20

21

22

033 【连续字母长度】

给定一个字符串，只包含大写字母，求在包含同一字母的子串中，长度第 k 长的子串的长度，相同字母只取最长的那个子串。

输入描述:

第一行有一个子串(1<长度<=100)，只包含大写字母。

第二行为 k 的值

输出描述:

输出连续出现次数第 k 多的字母的次数。

示例 1

输入

AAAAHHHBBCDHHHH

3

输出

2

说明

同一字母连续出现的最多的是 A 和 H，四次；第二多的是 H，3 次，但是 H 已经存在 4 个连续的，故不考虑；下个最长子串是 BB，所以最终答案应该输出 2。

示例 2

输入

AABAAA

2

输出

1

说明

同一字母连续出现的最多的是 A，三次；第二多的还是 A，两次，但 A 已经存在最大连续次数三次，故不考虑；下个最长子串是 B，所以输出 1

示例 3

输入

ABC

4

输出

-1

说明

只含有 3 个包含同一字母的子串，小于 k，输出-1

示例 4

输入

ABC

2

输出

1

说明

三个子串长度均为1，所以此时 k = 1，k=2，k=3 这三种情况均输出 1。特此说明，避免歧义。

备注:

若子串中只包含同一字母的子串数小于k，则输出-1

```
public class ZT33 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String input = sc.nextLine();
        int k1 = Integer.parseInt(sc.nextLine());
        int[] arr = new int[26];
        int temp = 0;
        char pre = ' ';
        for (int i = 0; i < input.length(); i++) {
            if (i == 0){
                temp++;
                pre = input.charAt(i);
            }else if (input.charAt(i) == pre){
                temp++;
                int count = arr[pre - 'A'];
                arr[pre - 'A'] = Math.max(temp, count);
            }else {//换代
                temp = 1;
                pre = input.charAt(i);
                arr[input.charAt(i) - 'A'] = Math.max(temp, arr[input.charAt(i) - 'A'] );;
            }
        }
        List<Integer> list = new ArrayList<>();
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] != 0){
                list.add(arr[i]);
            }
        }
        list.sort((a1,b1) -> b1 - a1);
        if (list.size()<k1){
            System.out.println(-1);
        }else {
            System.out.println(list.get(k1-1));
        }
    }
}
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36

034 【两数之和绝对值最小】

给定一个从小到大的有序整数序列（存在正整数和负整数）数组 nums ，请你在该数组中找出两个数，其和的绝对值(|nums[x]+nums[y]|)为最小值，并返回这个绝对值。

每种输入只会对应一个答案。但是，数组中同一个元素不能使用两遍。

输入描述:

一个通过空格分割的有序整数序列字符串，最多 1000 个整数，且整数数值范围是 -65535~65535。

输出描述:

两数之和绝对值最小值

示例 1

输入

-3 -1 5 7 11 15

输出

2

说明

因为 |nums[0] + nums[2]| = |-3 + 5| = 2 最小，所以返回 2

```
public class ZT34 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] input = sc.nextLine().split(" ");
        int[] arr = new int[input.length];
        for (int i = 0; i < input.length; i++) {
            arr[i] = Integer.parseInt(input[i]);
        }
        int min = Integer.MAX_VALUE;
        for (int i = 0; i < arr.length; i++) {
            for (int j = 0; j < arr.length; j++) {
                min = Math.min(min,Math.abs(arr[i]+arr[j]));
            }
        }
        System.out.println(min);
    }
}
```

035 【流水线】

一个工厂有 m 条流水线，来并行完成 n 个独立的作业，该工厂设置了一个调度系统，在安排作业时，总是优先执行处理时间最短的作业。

现给定流水线个数 m，需要完成的作业数 n，每个作业的处理时间分别为 t1,t2…tn。请你编程计算处理完所有作业的耗时为多少？

当 n>m 时，首先处理时间短的 m 个作业进入流水线，其他的等待，当某个作业完成时，依次从剩余作业中取处理时间最短的进入处理。

输入描述:

第一行为 2 个整数（采用空格分隔），分别表示流水线个数 m 和作业数 n；

第二行输入 n 个整数（采用空格分隔），表示每个作业的处理时长 t1,t2…tn。

0< m,n<100，0<t1,t2…tn<100。

注：保证输入都是合法的。

输出描述:

输出处理完所有作业的总时长

示例 1

输入

3 5

8 4 3 2 10

输出

13

说明

1、先安排时间为 2、3、4 的 3 个作业。

2、第一条流水线先完成作业，然后调度剩余时间最短的作业 8。

3、第二条流水线完成作业，然后调度剩余时间最短的作业 10。

4、总工耗时就是第二条流水线完成作业的时间 13（3+10）。

```
public class ZT35 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] input = sc.nextLine().split(" ");
        String[] job = sc.nextLine().split(" ");
        int med = Integer.parseInt(input[0]);
        int[] arr = new int[job.length];
        for (int i = 0; i < job.length; i++) {
            arr[i] = Integer.parseInt(job[i]);
        }
        Arrays.sort(arr);
        List<Medic> list = new ArrayList<>();
        for (int i = 0; i < arr.length; i++) {
            if (list.size() <med){
                list.add(new Medic(arr[i],arr[i]));
            }else {
                //找到数组中 total 最小的 medic 加进去
                Collections.sort(list);
                Medic medic = list.get(0);
```

```
                medic.total += arr[i];
            }
        }
        Collections.sort(list);
        System.out.println(list.get(list.size()-1).total);
    }
    static class Medic implements Comparable{
        private int end;
        private int total;

        public Medic(int end, int total) {
            this.end = end;
            this.total = total;
        }

        @Override
        public int compareTo(Object obj) {
            Medic medic= (Medic)obj;
            return this.total - medic.total;
        }
    }
}
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
```

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

036 【内存资源分配】

有一个简易内存池，内存按照大小粒度分类，每个粒度有若干个可用内存资源，用户会进行一系列内存申请，需要按需分配内存池中的资源，返回申请结果成功失败列表。分配规则如下：

1、分配的内存要大于等于内存申请量，存在满足需求的内存就必须分配，优先分配粒度小的，但内存不能拆分使用。

2、需要按申请顺序分配，先申请的先分配。

3、有可用内存分配则申请结果为 true，没有可用内存分配则返回 false。

注：不考虑内存释放。

输入描述:

输入为两行字符串：

第一行为内存池资源列表，包含内存粒度数据信息，粒度数据间用逗号分割，一个粒度信息内部用冒号分割，冒号前为内存粒度大小，冒号后为数量。资源列表不大于 1024，每个粒度的数量不大于 4096

第二行为申请列表，申请的内存大小间用逗号分隔。申请列表不大于 100000

如：

64:2,128:1,32:4,1:128

50,36,64,128,127

输出描述:

输出为内存池分配结果。

如：

true,true,true,false,false

示例 1

输入

64:2,128:1,32:4,1:128

50,36,64,128,127

输出

true,true,true,false,false

说明

内存池资源包含：64K 共 2 个、128K 共 1 个、32K 共 4 个、1K 共 128 个的内存资源；

针对 50,36,64,128,127 的内存申请序列，分配的内存依次是：64,64,128,NULL,NULL,第三次申请内存时已经将 128 分配出去，因此输出结果是：true,true,true,false,false

```
public class ZT36 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] input = sc.nextLine().split(",");
        String[] apply = sc.nextLine().split(",");
        List<Integer> exist = new ArrayList<>();
        Map<Integer,Integer> map = new HashMap<>();
        for (int i = 0; i < input.length; i++) {
            String[] typeCount = input[i].split(":");
            int type = Integer.parseInt(typeCount[0]);
            exist.add(type);
            map.put(type,Integer.parseInt(typeCount[1]));
        }
        Collections.sort(exist);
        for (int i = 0; i < apply.length; i++) {
            boolean flag = false;
            int need = Integer.parseInt(apply[i]);
            for (int j = 0; j < exist.size(); j++) {
                if (need<= exist.get(j)){
                    //拿出来一个
                    int pool = map.get(exist.get(j));
                    flag = true;
                    if (--pool == 0){
                        map.remove(exist.get(j));
                        exist.remove(j);
                    }else {
                        map.put(exist.get(j),pool);
                    }
                    break;
                }
            }
            System.out.print(flag + " ");
        }
        System.out.println();
    }
}
```

1

2

3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37

037 【判断一组不等式是否满足约束并输出最大差】

给定一组不等式，判断是否成立并输出不等式的最大差(输出浮点数的整数部分)，要求：1）不等式系数为 double 类型，是一个二维数组；2）不等式的变量为 int 类型，是一维数组；3）不等式的目标值为 double 类型，是一维数组；4）不等式约束为字符串数组，只能是：“>”，“>=”，“<”，“<=”，“=”，例如,不等式组：

a11x1+a12x2+a13x3+a14x4+a15x5<=b1;

a21x1+a22x2+a23x3+a24x4+a25x5<=b2;

a31x1+a32x2+a33x3+a34x4+a35x5<=b3;

最大差=max{ (a11x1+a12x2+a13x3+a14x4+a15x5-b1), (a21x1+a22x2+a23x3+a24x4+a25x5-b2),

(a31x1+a32x2+a33x3+a34x4+a35x5-b3) }，类型为整数(输出浮点数的整数部分)

输入描述:

1）不等式组系数(double 类型)：

a11,a12,a13,a14,a15

a21,a22,a23,a24,a25

a31,a32,a33,a34,a35

2）不等式变量(int 类型)：

x1,x2,x3,x4,x5

3）不等式目标值(double 类型)：b1,b2,b3

4)不等式约束(字符串类型):<=,<=,<=

输入：a11,a12,a13,a14,a15;a21,a22,a23,a24,a25;a31,a32,a33,a34,a35;x1,x2,x3,x4,x5;b1,b2,b3;<=,<=,<=

输出描述:

true 或者 false, 最大差

示例 1

输入

2.3,3,5.6,7,6;11,3,8.6,25,1;0.3,9,5.3,66,7.8;1,3,2,7,5;340,670,80.6;<=,<=,<=

输出

false 458

示例 2

输入

2.36,3,6,7.1,6;1,30,8.6,2.5,21;0.3,69,5.3,6.6,7.8;1,13,2,17,5;340,67,300.6;<=,>=,<=

输出

false 758

038 【判断字符串子序列】

给定字符串 target 和 source, 判断 target 是否为 source 的子序列。

你可以认为 target 和 source 中仅包含英文小写字母。字符串 source 可能会很长（长度 ~= 500,000），而 target 是个短字符串（长度 <=100）。

字符串的一个子序列是原始字符串删除一些（也可以不删除）字符而不改变剩余字符相对位置形成的新字符串。（例如，"abc"是"aebycd"的一个子序列，而"ayb"不是）。

请找出最后一个子序列的起始位置。

输入描述:

第一行为 target，短字符串（长度 <=100）

第二行为 source，长字符串（长度 ~= 500,000）

输出描述:

最后一个子序列的起始位置， 即最后一个子序列首字母的下标

示例 1

输入

abc

abcaybec

输出

3

说明

这里有两个 abc 的子序列满足，取下标较大的，故返回 3

备注:

若在 source 中找不到 target，则输出-1

```
public class ZT38 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String target = sc.nextLine();//[0,100]
        String source = sc.nextLine();//[0,50w]
        int idx = source.length();
        for (int i = target.length()-1; i >= 0; i--) {
            char ta = target.charAt(i);
            if (source.contains(String.valueOf(ta))) {
                int idxLast = source.lastIndexOf(String.valueOf(ta));
                if (idxLast < idx){
                    idx = idxLast;
                }else {
                    System.out.println(-1);
                    return;
                }
            }else {
                System.out.println(-1);
                return;
            }
        }
        System.out.println(idx);
    }
}
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
```

19

20

21

22

23

24

039 【拼接 URL】

给定一个 URL 前缀和 URL 后缀，通过","分割，需要将其连接为一个完整的 URL，如果前缀结尾和后缀开头都没有“/”，需自动补上“/”连接符，如果前缀结尾和后缀开头都为“/”，需自动去重。

约束：不用考虑前后缀 URL 不合法情况。

输入描述:

URL 前缀（一个长度小于 100 的字符串),URL 后缀（一个长度小于 100 的字符串）。

输出描述:

拼接后的 URL。

示例 1

输入

/acm,/bb

输出

/acm/bb

示例 2

输入

/abc/,/bcd

输出

/abc/bcd

示例 3

输入

/acd,bef

输出

/acd/bef

示例 4

输入

,

输出

/

040 【求符合要求的结对方式】

用一个数组 A 代表程序员的工作能力，公司想通过结对编程的方式提高员工的能力，假设结对后的能力为两个员工的能力之和，求一共有多少种结对方式使结对后能力为 N。

输入描述:

5

1 2 2 2 3

4

第一行为员工的总人数，取值范围[1,1000]

第二行为数组 A 的元素，每个元素的取值范围[1,1000]

第三行为N的值，取值范围[1,1000]

输出描述:

4

满足结对后能力为N的结对方式总数

示例1

输入

5

1 2 2 2 3

4

输出

4

说明

满足要求的结对方式为：A[0]和A[4]，A[1]和A[2]，A[1]和A[3]，A[2]和A[3]

```
public class ZT40 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int people = Integer.parseInt(sc.nextLine());
        String[] input = sc.nextLine().split(" ");
        int target = Integer.parseInt(sc.nextLine());
        int[] arr = new int[people];
        for (int i = 0; i < people; i++) {
            arr[i] = Integer.parseInt(input[i]);
        }
        int count = 0;
        for (int i = 0; i < people; i++) {
            for (int j = i + 1; j < people; j++) {
                if (target == arr[i] + arr[j]){
                    count++;
                }
            }
        }
        System.out.println(count);
    }
}
```

————————————————————

041 【求解连续数列】

042 【求字符串中所有整数的最小和】

043 【求最多可以派出多少支团队】

044 【删除字符串中字符最少字符】

045 【数据分类】

046 【数列描述】

047 【数字涂色】

048 【数组二叉树】

049 【数组拼接】

050 【数组去重和排序】

041 【求解连续数列】

已知连续正整数数列{K}=K1,K2,K3…Ki 的各个数相加之和为 S，i=N（0<S<100000，0<N<100000），求此数列 K。

输入描述:

输入包含两个参数，1）连续正整数数列和 S，2）数列里数的个数 N。

输出描述:

如果有解输出数列 K，如果无解输出-1

示例 1

输入

525 6

输出

85 86 87 88 89 90

示例 2

输入

3 5

输出

-1

```
public class ZT41 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] input = sc.nextLine().split(" ");
        int result = Integer.parseInt(input[0]);
        int count = Integer.parseInt(input[1]);
        int middle = 0;
        if (result/count == 0){
            System.out.println(-1);
            return;
        }
        int[] arr = new int[count];
        int start = 0;
        if (result % count == 0){//count 是奇数
            //正好的中间的奇数位
            middle = result/count;
            start = middle - count/2;
        }else {//向下取整了 ，middle 是中间的前一个数
            middle = result/count;
```

```
            start = middle - count/2 +1;
        }
        int idx = 0;
        while (idx<count){
            arr[idx] = start;
            start++;
            idx++;
        }
        int sum = 0;
        for (int i = 0; i < arr.length; i++) {
            sum += arr[i];
        }
        if (sum == result){
            for (int i = 0; i < arr.length; i++) {
                System.out.print(arr[i] + " ");
            }
        }else {
            System.out.println(-1);
        }
    }
}
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
```

24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
用数学表达式求和后，直接计算出起始位置更优

```
public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int sum = in.nextInt();
        int n = in.nextInt();
        // sum(x x+1 x+2 ... x+n-1) = sum
        // n*x + n*(n-1)/2 = sum
        // x= [sum - n*(n-1)/2 ]/n
        int temp = sum - n*(n-1)/2;
        if (temp <=0 || temp%n!=0){
            System.out.println(-1);
            return;
        }
        int begin = temp/n;
        for (int i = 0; i < n; i++) {
            System.out.print(begin+i);
            System.out.print(' ');
        }
    }
}
```

1
2
3
4
5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

042 【求字符串中所有整数的最小和】

输入字符串 s，输出 s 中包含所有整数的最小和

说明

字符串 s，只包含 a-z A-Z ± ；

合法的整数包括

1） 正整数 一个或者多个 0-9 组成，如 0 2 3 002 102

2）负整数 负号 - 开头，数字部分由一个或者多个 0-9 组成，如 -0 -012 -23 -00023

输入描述:

包含数字的字符串

输出描述:

所有整数的最小和

示例 1

输入

bb1234aa

输出

10

示例 2

输入

bb12-34aa

输出

-31

说明

1+2+（-34） = 31

```
public class ZT42 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String input = sc.nextLine();
        //原则 1、正数所有都拆分
```

```
        //2、负数保留最大
        boolean fuShu = false;
        String tempNum = "-";
        List<Integer> list = new ArrayList<>();
        for (int i = 0; i < input.length(); i++) {
            char tem = input.charAt(i);
            if (tem == '-'){
                fuShu = true;
                continue;
            }
            if (tem >= '0' && tem <= '9'){
                if (fuShu){ //只要不是字符就一直往下记录
                    tempNum += tem;
                }else {
                    list.add(Integer.parseInt(String.valueOf(tem)));
                }
            }else {
                if (tempNum != "-"){
                    list.add(Integer.parseInt(tempNum));
                    tempNum = "-";
                }
                fuShu = false;
            }
        }
        int total = 0;
        for (int i = 0; i < list.size(); i++) {
            total += list.get(i);
        }
        System.out.println(total);
    }
}
1
2
3
4
5
6
7
8
9
10
11
12
13
```

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

043 【求最多可以派出多少支团队】

用数组代表每个人的能力，一个比赛活动要求参赛团队的最低能力值为N，每个团队可以由1人或2人组成，且1个人只能参加1个团队，请计算出最多可以派出多少支符合要求的团队？

输入描述:

5

3 1 5 7 9

8

第一行数组代表总人数，范围[1,500000]

第二行数组代表每个人的能力，每个元素的取值范围[1, 500000]，数组的大小范围[1,500000]

第三行数值为团队要求的最低能力值，范围[1, 500000]

输出描述:

3

最多可以派出的团队数量

示例1

输入

5

3 1 5 7 9

8

输出

3

说明

3,5 组成一队，1,7 组成一队，9 自己一个队，故输出 3

```
public class ZT43 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int count = Integer.parseInt(sc.nextLine());
        String[] input = sc.nextLine().split(" ");
        int limit = Integer.parseInt(sc.nextLine());
        int maxTeam = 0;
        int[] arr = new int[count];
        for (int i = 0; i < count; i++) {
            int tem = Integer.parseInt(input[i]);
            if (tem >= limit){
                maxTeam++;
            }else {
                arr[i] = tem;
            }
        }
        Arrays.sort(arr);//1 2 3 5 6
        int left = 0;
        int right = arr.length-1;
        while (left< right){
            if (arr[left] + arr[right] < limit){//最小+最大 < min
                left++;
                continue;
            }else {//min + max >= limit
                maxTeam++;
                left++;
                right--;
            }
        }
        System.out.println(maxTeam);
    }
}
1
2
3
4
5
6
7
8
9
10
```

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

044 【删除字符串中字符最少字符】

删除字符串中出现次数最少的字符，如果有多个字符出现次数一样，则都删除。

输入描述:

输入 abcdd

字符串中只包含小写英文字母。

输出描述:

dd

示例 1

输入

abcdd

输出

dd

示例 2

输入

aabbccdd

输出

empty

说明

如果字符串的字符都被删除，则范围 empty

```
public class ZT44 {
    public static void main(String[] args) {
```

```
        Scanner sc = new Scanner(System.in);
        String input = sc.nextLine();
        Map<String,Integer> map = new HashMap<>();
        for (int i = 0; i < input.length(); i++) {
            int orDefault = map.getOrDefault(String.valueOf(input.charAt(i)), 0);
            map.put(String.valueOf(input.charAt(i)),orDefault+1);
        }
        int less = Collections.min(map.values());
        StringBuilder res = new StringBuilder();
        Set<String> set = new HashSet<>();
        for (Map.Entry<String,Integer> mp:map.entrySet()) {
            int times = mp.getValue();
            if (times == less) {
                set.add(mp.getKey());
            }
        }
        for (int i = 0; i < input.length(); i++) {
            if (!set.contains(String.valueOf(input.charAt(i)))) {
                res.append(input.charAt(i));
            }
        }
        if (res.length() == 0){
            System.out.println("empty");
        }else {
            System.out.println(res);
        }
    }
}
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
```

17

18

19

20

21

22

23

24

25

26

27

28

29

30

045 【数据分类】

对一个数据 a 进行分类，分类方法为：此数据 a（四个字节大小）的四个字节相加对一个给定的值 b 取模，如果得到的结果小于一个给定的值 c，则数据 a 为有效类型，其类型为取模的值；如果得到的结果大于或者等于 c，则数据 a 为无效类型。

比如一个数据 a=0x01010101，b=3，按照分类方法计算（0x01+0x01+0x01+0x01）%3=1，所以如果 c=2，则此 a 为有效类型，其类型为 1，如果 c=1，则此 a 为无效类型；

又比如一个数据 a=0x01010103，b=3，按照分类方法计算（0x01+0x01+0x01+0x03）%3=0，所以如果 c=2，则此 a 为有效类型，其类型为 0，如果 c=0，则此 a 为无效类型。

输入 12 个数据，第一个数据为 c，第二个数据为 b，剩余 10 个数据为需要分类的数据，请找到有效类型中包含数据最多的类型，并输出该类型含有多少个数据。

输入描述:

输入 12 个数据，用空格分隔，第一个数据为 c，第二个数据为 b，剩余 10 个数据为需要分类的数据。

输出描述:

输出最多数据的有效类型有多少个数据。

示例 1

输入

3 4 256 257 258 259 260 261 262 263 264 265

输出

3

说明

10 个数据 4 个字节相加后的结果分别为 1 2 3 4 5 6 7 8 9 10，故对 4 取模的结果为 1 2 3 0 1 2 3 0 1 2，c 为 3，所以 0 1 2 都是有效类型，类型为 1 和 2 的有 3 个数据，类型为 0 的只有 2 个数据，故输出 3

示例 2

输入

1 4 256 257 258 259 260 261 262 263 264 265

输出

2

说明

10 个数据 4 个字节相加后的结果分别为 1 2 3 4 5 6 7 8 9 10，故对 4 取模的结果为 1 2 3 0 1 2 3 0 1 2，c 为 1，所以只有 0 是有效类型，类型为 0 的有 2 个数据，故输出 2

```
public class ZT45 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] input = sc.nextLine().split(" ");
        int c1 = Integer.parseInt(input[0]);
        int b1 = Integer.parseInt(input[1]);
        int[] arr = new int[b1-1];
        for (int i = 2; i < input.length; i++) {
            int res = calcAdd(Integer.parseInt(input[i])) % b1;
            if (res < c1){//有效
                arr[res]++;
            }
        }
        int max = -1;
        for (int i = 0; i < arr.length ; i++) {
            max = Math.max(max,arr[i]);
        }
        System.out.println(max);
    }

    private static int calcAdd(int num){
        String str = Integer.toBinaryString(num);
        //100000000
        int total = 0;
        while (str.length() > 8){
            String temp = str.substring(str.length()-8);
            total += Integer.parseInt(temp);
            str = str.substring(0,str.length()-8);
        }
       total += Integer.parseInt(str);
       return total;
    }
}
1
2
3
4
5
6
7
8
9
10
11
```

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

046 【数列描述】

有一个数列 a[N] (N=60)，从 a[0]开始，每一项都是一个数字。数列中 a[n+1]都是 a[n]的描述。其中 a[0]=1。

规则如下：

a[0]:1

a[1]:11(含义：其前一项 a[0]=1 是 1 个 1，即“11”。表示 a[0]从左到右，连续出现了 1 次“1”)

a[2]:21(含义：其前一项 a[1]=11，从左到右：是由两个 1 组成，即“21”。表示 a[1]从左到右，连续出现了两次“1”)

a[3]:1211(含义：其前一项 a[2]=21，从左到右：是由一个 2 和一个 1 组成，即“1211”。表示 a[2]从左到右，连续出现了 1 次“2”，然后又连续出现了 1 次“1”)

a[4]:111221(含义：其前一项 a[3]=1211，从左到右：是由一个 1、一个 2、两个 1 组成，即“111221”。表示 a[3]从左到右，连续出现了 1 次“1”，连续出现了 1 次“2”，连续出现了两次“1”)

请输出这个数列的第 n 项结果（a[n]，0≤n≤59）。

输入描述:

数列的第 n 项(0≤n≤59)：

4

输出描述:

数列的内容：

111221

示例 1

输入

4

输出

111221

```
public class ZT46 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int input = sc.nextInt();
        String temp = "1";
        for (int i = 1; i <= input; i++) {
            int idx = 0;
            char pre = ' ';
            StringBuilder sb = new StringBuilder();
            for (int j = 0; j < temp.length(); j++) {
                if (j == 0){
                    pre = temp.charAt(j);
                }
                char tem = temp.charAt(j);
                if (tem == pre){
                    idx++;
                }else {
                    sb.append(idx).append(pre);
                    idx = 1;
                    pre = tem;
                }
            }
            sb.append(idx).append(pre);
            temp = sb.toString();
        }
        System.out.println(temp);
    }
}
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

16

17

18

19

20

21

22

23

24

25

26

27

28

047 【数字涂色】

疫情过后，希望小学终于又重新开学了，三年二班开学第一天的任务是将后面的黑板报重新制作。黑板上已经写上了 N 个正整数，同学们需要给这每个数分别上一种颜色。为了让黑板报既美观又有学习意义，老师要求同种颜色的所有数都可以被这种颜色中最小的那个数整除。现在请你帮帮小朋友们，算算最少需要多少种颜色才能给这 N 个数进行上色。

输入描述:

第一行有一个正整数 N，其中 1 \leq N \leq 1001≤N≤100。

第二行有 N 个 int 型数(保证输入数据在[1,100]范围中)，表示黑板上各个正整数的值。

输出描述:

输出只有一个整数，为最少需要的颜色种数。

示例 1

输入

3

2 4 6

输出

1

说明

所有数都能被 2 整除

示例 2

输入

4

2 3 4 9

输出

2

说明

2 与 4 涂一种颜色，4 能被 2 整除；3 与 9 涂另一种颜色，9 能被 3 整除。不能 4 个数涂同一个颜色，因为 3 与 9 不能被 2 整除。所以最少的颜色是两种。

```
public class ZT47 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int count = Integer.parseInt(sc.nextLine());
```

```
        String[] input = sc.nextLine().split(" ");
        int[] arr = new int[count];
        for (int i = 0; i < count; i++) {
            arr[i] = Integer.parseInt(input[i]);
        }
        Arrays.sort(arr);
        List<Integer> list = new ArrayList<>();
        for (int i = 0; i < arr.length; i++) {
            //如果 list 的数能整除 arr[i] 就过 否则要加进 list
            int temp = arr[i];
            boolean flag = false;
            for (int j = 0; j < list.size(); j++) {
                int num = list.get(j);
                if (temp % num == 0){
                    flag = true;
                    break;
                }
            }
            if (!flag){
                list.add(temp);
            }
        }
        System.out.println(list.size());
    }
}
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

20

21

22

23

24

25

26

27

28

29

048 【数组二叉树】

二叉树也可以用数组来存储，给定一个数组，树的根节点的值存储在下标 1，对于存储在下标 N 的节点，它的左子节点和右子节点分别存储在下标 2N 和 2N+1，并且我们用值-1 代表一个节点为空。

给定一个数组存储的二叉树，试求从根节点到最小的叶子节点的路径，路径由节点的值组成。

输入描述:

输入一行为数组的内容，数组的每个元素都是正整数，元素间用空格分隔。注意第一个元素即为根节点的值，即数组的第 N 个元素对应下标 N，下标 0 在树的表示中没有使用，所以我们省略了。输入的树最多为 7 层。

输出描述:

输出从根节点到最小叶子节点的路径上，各个节点的值，由空格分隔，用例保证最小叶子节点只有一个。

示例 1

输入

3 5 7 -1 -1 2 4

输出

3 7 2

说明

数组存储的二叉树如图，故到最小叶子节点的路径为 3 7 2

示例 2

输入

5 9 8 -1 -1 7 -1 -1 -1 -1 -1 6

输出

5 8 7 6

说明

数组存储的二叉树如图，故到最小叶子节点的路径为 10 8 7 6，注意数组仅存储至最后一个非空节点，故不包含节点“7”右子节点的-1

```
public class ZT48 {
    public static void main(String[] args) {//1 2 4 8 16
        Scanner sc = new Scanner(System.in);
        String[] input = sc.nextLine().split(" ");
        //先找到最小叶子节点所在行  求行号
        int min = Integer.MAX_VALUE;//数组每个元素都是正整数
        int idx = 0;
        for (int i = 1; i < input.length; i++) {
            if (Integer.parseInt(input[i]) != -1 && Integer.parseInt(input[i])<min){
```

```
            idx = i;
            min = Integer.parseInt(input[i]);
        }
    }
    int line = calcLine(idx);
    int[] arr = new int[line];
    //line 表示行号 start 表示该行首个元素
    int start = calcStart(line);//arr[7]
    arr[line-1] = Integer.parseInt(input[idx]);
    //求该值在所在行的序号
    int turnInLine = idx - start + 1 ;
    //求剩余其他行
    line--;
    turnInLine = turnInLine%2 ==0 ? turnInLine/2 : turnInLine/2+1;//3
    while (line > 0){
        start = calcStart(line);//arr
        arr[line-1] = Integer.parseInt(input[turnInLine + start -1 ]);
        line--;
        turnInLine = turnInLine%2 ==0 ? turnInLine/2 : turnInLine/2+1;//2
    }
    for (int i = 0; i < arr.length; i++) {
        System.out.print(arr[i] + " ");
    }
    System.out.println();
}

private static int calcStart(int line){
    int total = 0;
    int tem = 0;
    while (line>1){
        total += 1<<tem;
        tem++;
        line--;
    }
    return total;
}

//计算在第几行
private static int calcLine(int line){
    int tem = 0;
    while (line>0){
        line -= 1<<tem;
        tem++;
    }
```

```
        return tem;
    }
}
```

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

049 【数组拼接】

现在有多组整数数组，需要将它们合并成一个新的数组。合并规则，从每个数组里按顺序取出固定长度的内容合并到新的数组中，取完的内容会删除掉，如果该行不足固定长度或者已经为空，则直接取出剩余部分的内容放到新的数组中，继续下一行。

输入描述:

第一行是每次读取的固定长度，0<长度<10

第二行是整数数组的数目，0<数目<1000

第 3-n 行是需要合并的数组，不同的数组用回车换行分隔，数组内部用逗号分隔，最大不超过 100 个元素。

输出描述:

输出一个新的数组，用逗号分隔。

示例 1

输入

3

2

2,5,6,7,9,5,7

1,7,4,3,4

输出

2,5,6,1,7,4,7,9,5,3,4,7

说明

1、获得长度 3 和数组数目 2。

2、先遍历第一行，获得 2,5,6；

3、再遍历第二行，获得 1,7,4；

4、再循环回到第一行，获得 7,9,5；

5、再遍历第二行，获得 3,4；

6、再回到第一行，获得 7，按顺序拼接成最终结果。

示例 2

输入

4

3

1,2,3,4,5,6

1,2,3

1,2,3,4

输出

1,2,3,4,1,2,3,1,2,3,4,5,6

```
public class ZT49 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int limit = Integer.parseInt(sc.nextLine());
        int num = Integer.parseInt(sc.nextLine());
        Map<Integer, List<Integer>> map = new HashMap<>();
        //存
        for (int i = 0; i < num; i++) {
            String[] split = sc.nextLine().split(",");
            List<Integer> list = new ArrayList<>();
            for (int j = 0; j < split.length; j++) {
                list.add(Integer.parseInt(split[j]));
            }
            map.put(i,list);
        }
        //取
        List<Integer> list = new ArrayList<>();
        while (!map.isEmpty()){
            Iterator<Map.Entry<Integer, List<Integer>>> iterator = map.entrySet().iterator();
            List<Integer> removeKey = new ArrayList<>();
            while (iterator.hasNext()){
                Map.Entry<Integer, List<Integer>> next = iterator.next();
                List<Integer> value = next.getValue();
                int start = 0;
                for (int i = 0; i < value.size(); i++) {
                    if (i<limit){
                        list.add(value.get(i));
                        start++;
                    }
                }
                value = value.subList(start,value.size());

                if (value.size() == 0){
                    removeKey.add(next.getKey());
                }else {
                    next.setValue(value);
                }
            }
            for (int i = 0; i < removeKey.size(); i++) {
```

```
                map.remove(removeKey.get(i));
            }
        }
        System.out.println(list);
    }
}
```

39

40

41

42

43

44

45

050 【数组去重和排序】

给定一个乱序的数组，删除所有的重复元素，使得每个元素只出现一次，并且按照出现的次数从高到低进行排序，相同出现次数按照第一次出现顺序进行先后排序。

输入描述:

一个数组

输出描述:

去重排序后的数组

示例 1

输入

1,3,3,3,2,4,4,4,5

输出

3,4,1,2,5

备注:

数组大小不超过 100

数组元素值大小不超过 100

```
public class ZT50 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] split = sc.nextLine().split(",");
        Map<Integer,Integer> map = new HashMap<>();
        List<Order> list = new ArrayList<>();
        int idxInList = 0;
        for (int i = 0; i < split.length; i++) {
            int value = Integer.parseInt(split[i]);
            Order order = new Order(i,value,1);
            if (list.contains(order)){
                int idx = map.get(value);
                list.get(idx).count++;
            }else {
                map.put(value,idxInList++);
                list.add(order);
            }
        }
        list.sort(null);//能把下标重排吗?
        for (int i = 0; i < list.size(); i++) {
            System.out.print(list.get(i).value + ",");
```

```
        }

    }
    private static class Order implements Comparable<Order>{
        private int idx;//出场顺序
        private int value;
        private int count;

        public Order(int idx ,int value, int count) {
            this.idx = idx;
            this.value = value;
            this.count = count;
        }

        @Override
        public boolean equals(Object obj) {
            Order order = (Order) obj;
            return order.value == this.value;
        }

        @Override
        public int compareTo(Order order) {
            if (order.count != this.count){
                return order.count - this.count;
            }else {
                return this.idx - order.idx ;
            }
        }
    }
}
```

————————————————————

051 【数组组成的最小数字】
052 【水仙花数】
053 【素数之积】
054 【太阳能板最大面积】
055 【停车场车辆统计】
056 【统计射击比赛成绩】
057 【完全二叉树非叶子部分后序遍历】
058 【玩牌高手】

059 【相对开音节】

060 【消消乐游戏】

051 【数组组成的最小数字】

给定一个整型数组，请从该数组中选择 3 个元素组成最小数字并输出（如果数组长度小于 3，则选择数组中所有元素来组成最小数字）。

输入描述:

一行用半角逗号分割的字符串记录的整型数组，0 < 数组长度 <= 100，0 < 整数的取值范围 <= 10000。

输出描述:

由 3 个元素组成的最小数字，如果数组长度小于 3，则选择数组中所有元素来组成最小数字。

示例 1

输入

21,30,62,5,31

输出

21305

说明

数组长度超过 3，需要选 3 个元素组成最小数字，21305 由 21,30,5 三个元素组成的数字，为所有组合中最小的数字

示例 2

输入

5,21

输出

215

说明

数组长度小于 3， 选择所有元素来主城最小值，215 为最小值。

//思路：

//1、大于 3 各数的数组，从其中选择 3 各最小的数，组合排序可以得到最小的数

//2、小于 3 各数，组合排序即可

//3、可以根据首位进行排序

1

2

3

4

052 【水仙花数】

所谓水仙花数，是指一个 n 位的正整数，其各位数字的 n 次方和等于该数本身。例如 153 是水仙花数，153 是一个 3 位数，并且 153 = 111 + 555 + 333

输入描述:

第一行输入一个整数 n，表示一个 n 位的正整数。n 在 3 到 7 之间，包含 3 和 7。[3,7]

第二行输入一个正整数 m，表示需要返回第 m 个水仙花数。

输出描述:

返回长度是 n 的第 m 个水仙花数。个数从 0 开始编号。

若 m 大于水仙花数的个数，返回最后一个水仙花数和 m 的乘积。

若输入不合法，返回-1。

示例 1

输入

3 0

输出

153

说明

153 是第一个水仙花数

示例 2

输入

9

1

输出

-1

说明

9 超出范围

```
public class ZT02 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int count = Integer.parseInt(sc.nextLine());//[3,7]
        int num = Integer.parseInt(sc.nextLine());
        if (count< 3 || count> 7){
            System.out.println(-1);
            return;
        }//3 100 4 1000
        int start = 1;
        for (int i = 1; i < count; i++) {
            start *= 10;
        }
        int end = start *10 -1;
        int times = 0;
        for (int i = start; i < end; i++) {
            if (checkFlower(i)){
                if (times++ == num) {
                    System.out.println(i);
                    return;
                }
            }
        }
        System.out.println(-1);
    }

    private static boolean checkFlower(int num){
        int temp = num;
        int total = 0;
        while (temp > 0){
```

```
            int wei = temp%10;
            total += wei *wei * wei;
            temp /=10;
        }
        return total == num;
    }
}
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
```

053 【素数之积】

RSA 加密算法在网络安全世界中无处不在，它利用了极大整数因数分解的困难度，数据越大，安全系数越高，给定一个 32 位正整数，请对其进行因数分解，找出是哪两个素数的乘积。

输入描述:

一个正整数 num

0 < num <= 2147483647

输出描述:

如果成功找到，以单个空格分割，从小到大输出两个素数，分解失败，请输出-1 -1

示例 1

输入

15

输出

3 5

说明

因数分解后，找到两个素数 3 和 5，使得 3*5=15，按从小到大排列后，输出 3 5

示例 2

输入

27

输出

-1 -1

说明

通过因数分解，找不到任何素数，使得他们的乘积为 27，输出-1 -1

```
public class ZT53 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int target = sc.nextInt();//[0,2147483647]
        int max = target;
        for (int i = 3; i <= max; i++) {
            //先判断能不能被 target 整除
            if (target % i == 0){
                max = target / i;
                if (checkSu(i) && checkSu(target/i)){
                    System.out.println(i + " " + target/i);
                    return;
                }
            }
        }
        System.out.println(-1 + " " + -1);
    }
    //素数是指除了 1 和本身不能被其他所有数整除
    private static boolean checkSu(int num){
        for (int i = 2; i < num; i++) {
            if (num % i == 0){
```

```
                return false;
            }
        }
        return true;
    }
}
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27

054 【太阳能板最大面积】

给航天器一侧加装长方形或正方形的太阳能板（图中的红色斜线区域），需要先安装两个支柱（图中的黑色竖条），再在支柱的中间部分固定太阳能板。但航天器不同位置的支柱长度不同，太阳能板的安装面积受限于最短一侧的那根支柱长度。如图：现提供一组整形数组的支柱高度数据，假设每根支柱间距离相等为 1 个单位长度，计算如何选择两根支柱可以使太阳能板的面积最大。

输入描述:

10,9,8,7,6,5,4,3,2,1

注：支柱至少有 2 根，最多 10000 根，能支持的高度范围 1~10^9 的整数。柱子的高度是无序的，例子中递减只是巧合。

输出描述:

可以支持的最大太阳能板面积：（10 米高支柱和 5 米高支柱之间）

25

示例 1

输入

10,9,8,7,6,5,4,3,2,1

输出

25

备注:

10 米高支柱和 5 米高支柱之间宽度为 5，高度取小的支柱高也是 5，面积为 25。任取其他两根支柱所能获得的面积都小于 25。所以最大的太阳能板面积为 25。

055 【停车场车辆统计】

特定大小的停车场，数组 cars[]表示，其中 1 表示有车，0 表示没车。车辆大小不一，小车占一个车位（长度 1），货车占两个车位（长度 2），卡车占三个车位（长度 3），统计停车场最少可以停多少辆车，返回具体的数目。

输入描述:

整型字符串数组 cars[]，其中 1 表示有车，0 表示没车，数组长度小于 1000。

输出描述:

整型数字字符串，表示最少停车数目。

示例 1

输入

1,0,1

输出

2

说明

1 个小车占第 1 个车位

第二个车位空

1 个小车占第 3 个车位

最少有两辆车

示例 2

输入

1,1,0,0,1,1,1,0,1

输出

3

说明

1 个货车占第 1、2 个车位

第 3、4 个车位空

1 个卡车占第 5、6、7 个车位

第 8 个车位空

1 个小车占第 9 个车位

最少 3 辆车

```
public class ZT55 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] split = sc.nextLine().split(",");
        //连续的 2 个 1 可以是 1 个 连续三个 1 可以是 1 个
```

```
        int total = 0;
        int tem = 0;
        for (int i = 0; i < split.length; i++) {
            if (Integer.parseInt(split[i]) == 1){
                tem++;
            }else {
                if (tem != 0){
                    total += calcMin(tem);
                }
                tem = 0;
            }
        }
        total += calcMin(tem);
        System.out.println(total);
    }

    private static int calcMin(int num){
        //6 -> 2
        int total = 0;
        while (num >= 3){//有多少 3 除掉多少 3
            num -= 3;
            total++;
        }
        while (num >= 2){//有多少 3 除掉多少 3
            num -= 2;
            total++;
        }
        total += num;
        return total;
    }
}
1
2
3
4
5
6
7
8
9
10
11
12
13
```

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

056 【统计射击比赛成绩】

给定一个射击比赛成绩单，包含多个选手若干次射击的成绩分数，请对每个选手按其最高 3 个分数之和进行降序排名，输出降序排名后的选手 ID 序列。条件如下：

1、一个选手可以有多个射击成绩的分数，且次序不固定。

2、如果一个选手成绩少于 3 个，则认为选手的所有成绩无效，排名忽略该选手。

3、如果选手的成绩之和相等，则成绩之和相等的选手按照其 ID 降序排列。

输入描述:

输入第一行，一个整数 N，表示该场比赛总共进行了 N 次射击，产生 N 个成绩分数（2<=N<=100）。

输入第二行，一个长度为 N 整数序列，表示参与每次射击的选手 ID（0<=ID<=99）。

输入第三行，一个长度为 N 整数序列，表示参与每次射击的选手对应的成绩（0<=成绩<=100）。

输出描述:

符合题设条件的降序排名后的选手 ID 序列。

示例 1

输入

13

3,3,7,4,4,4,4,7,7,3,5,5,5

53,80,68,24,39,76,66,16,100,55,53,80,55

输出

5,3,7,4

说明

该场射击比赛进行了 13 次，参赛的选手为{3,4,5,7}。

3 号选手成绩：53,80,55，最高 3 个成绩的和为：80+55+53=188。

4 号选手成绩：24,39,76,66，最高 3 个成绩的和为：76+66+39=181。

5 号选手成绩：53,80,55，最高 3 个成绩的和为：80+55+53=188。

7 号选手成绩：68,16,100，最高 3 个成绩的和为：100+68+16=184。

比较各个选手最高 3 个成绩的和，有 3 号=5 号>7 号>4 号，由于 3 号和 5 号成绩相等且 ID 号 5>3，所以输出为：5,3,7,4

```
public class ZT56 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int count = Integer.parseInt(sc.nextLine());
        String[] turnList = sc.nextLine().split(",");
        String[] scores = sc.nextLine().split(",");
        List<Player> plays = new ArrayList<>();
        Map<Integer,Integer> map= new HashMap<>();
        int tempCount = 0;
        //获取成绩
        for (int i = 0; i < count; i++) {
            int idx = Integer.parseInt(turnList[i]);
            List<Integer> li = new ArrayList<>();
            li.add(Integer.parseInt(scores[i]));
            Player pls = new Player(idx,li);
            if (plays.contains(pls)) {
                plays.get(map.get(idx)).list.add(Integer.parseInt(scores[i]));
            }else{
                map.put(idx,tempCount++);
                plays.add(pls);
            }
        }
        //整理成绩
        for (int i = 0; i < plays.size(); i++) {
            Player player = plays.get(i);
            List<Integer> list = player.list;
            list.sort((a0,b0) -> b0 -a0);//逆序
            int total = 0;
            for (int j = 0; j < 3; j++) {
                total += list.get(j);
            }
            player.setScore(total);
        }
        plays.sort(null);
        for (int i = 0; i < plays.size(); i++) {
            if (i == plays.size() -1){
                System.out.print(plays.get(i).idx);
            }else {
```

```
                System.out.print(plays.get(i).idx + ",");
            }
        }
    }
    static class Player implements Comparable<Player>{
        private int idx;
        private List<Integer> list;
        private int score;

        public void setScore(int score) {
            this.score = score;
        }

        public Player(int idx, List<Integer> list) {
            this.idx = idx;
            this.list = list;
        }

        @Override
        public boolean equals(Object obj) {
            Player ply = (Player)obj;
            return ply.idx == this.idx;
        }

        @Override
        public int compareTo(Player ply) {
            if (ply.score != this.score){
                return ply.score - this.score;
            }else {
                return ply.idx - this.idx;
            }
        }
    }
}
1
2
3
4
5
6
7
8
9
10
```

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

057 【完全二叉树非叶子部分后序遍历】

给定一个以顺序储存结构存储整数值的完全二叉树序列（最多 1000 个整数），请找出此完全二叉树的所有非叶子节点部分，然后采用后序遍历方式将此部分树（不包含叶子）输出。

1、只有一个节点的树，此节点认定为根节点（非叶子）。

2、此完全二叉树并非满二叉树，可能存在倒数第二层出现叶子或者无右叶子的情况

其他说明：二叉树的后序遍历是基于根来说的，遍历顺序为：左-右-根 根输出

输入描述:

一个通过空格分割的整数序列字符串

输出描述:

非叶子部分树结构

示例 1

输入

1 2 3 4 5 6 7

输出

2 3 1

说明

找到非叶子部分树结构，然后采用后续遍历输出

备注:

输出数字以空格分隔

```
public class ZT57Tree {
    private static List<TreeNode> nodes = new ArrayList<>();

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] input = sc.nextLine().split(" ");
```

```
    int[] arr = new int[input.length+1];
    for (int i = 1; i <= input.length; i++) {
        arr[i] = Integer.parseInt(input[i-1]);
    }
    buildTree(arr);
    //检查叶子节点
    for (int i = 0; i < nodes.size(); i++) {
        int degree = 0;
        if (nodes.get(i).left != null){
            degree++;
        }
        if (nodes.get(i).right != null){
            degree++;
        }
        nodes.get(i).degree = degree;
    }
    afterPrint(nodes.get(1));
}

private static void buildTree(int[] arr){
    nodes.add(new TreeNode(0,null,null));
    //先将所有的节点放到数组中
    for (int i = 1; i < arr.length; i++) {
        nodes.add(new TreeNode(arr[i],null,null));
    }
    for (int i = 1; i < nodes.size(); i++) {
        if (2 * i < nodes.size()){
            nodes.get(i).left = nodes.get(2 * i);
        }
        if (2 * i +1 < nodes.size()){
            nodes.get(i).right = nodes.get(2 * i +1);
        }
    }
}

private static void print(TreeNode node){
    //只输出非叶子节点
    if (node.degree != 0){
        System.out.println(node.val);
    }
}
//前序遍历 根[根输出] 左 右
private static void prePrint(TreeNode node){
    print(node);
```

```
        if (node.left != null){
            prePrint(node.left);
        }
        if (node.right != null){
            prePrint(node.right);
        }
    }

    //中序遍历  左 根[根输出] 右
    private static void middlePrint(TreeNode node){
        if (node.left != null){
            middlePrint(node.left);
        }
        print(node);
        if (node.right != null){
            middlePrint(node.right);
        }
    }

    //后续遍历 左-右-根[根输出]
    private static void afterPrint(TreeNode node){
        if (node.left != null){
            afterPrint(node.left);
        }
        if (node.right != null){
            afterPrint(node.right);
        }
        print(node);
    }

    private static class TreeNode{
        private int degree;
        private int val;
        private TreeNode left;
        private TreeNode right;

        public TreeNode(int val, TreeNode left, TreeNode right) {
            this.val = val;
            this.left = left;
            this.right = right;
        }
    }
}
1
```

90

91

92

93

058 【玩牌高手】

给定一个长度为 n 的整型数组，表示一个选手在 n 轮内可选择的牌面分数。选手基于规则选牌，请计算所有轮结束后其可以获得的最高总分数。选择规则如下：

1、在每轮里选手可以选择获取该轮牌面，则其总分数加上该轮牌面分数，为其新的总分数。

2、选手也可不选择本轮牌面直接跳到下一轮，此时将当前总分数还原为 3 轮前的总分数，若当前轮次小于等于 3（即在第 1、2、3 轮选择跳过轮次），则总分数置为 0。

3、选手的初始总分数为 0，且必须依次参加每一轮。

输入描述:

第一行为一个小写逗号分割的字符串，表示 n 轮的牌面分数，1<= n <=20。

分数值为整数，-100 <= 分数值 <= 100。

不考虑格式问题。

输出描述:

所有轮结束后选手获得的最高总分数。

示例 1

输入

1,-5,-6,4,3,6,-2

输出

11

说明

总共有 7 轮牌面。

第一轮选择该轮牌面，总分数为 1。

第二轮不选择该轮牌面，总分数还原为 0。

第三轮不选择该轮牌面，总分数还原为 0。

第四轮选择该轮牌面，总分数为 4。

第五轮选择该轮牌面，总分数为 7。

第六轮选择该轮牌面，总分数为 13。

第七轮如果不选择该轮牌面，则总分数还原到 3 轮 1 前分数，即第四轮的总分数 4，如果选择该轮牌面，总分数为 11，所以选择该轮牌面。

因此，最终的最高总分为 11。

059 【相对开音节】

相对开音节构成的结构为辅音+元音（aeiou）+辅音(r 除外)+e，常见的单词有 bike、cake 等。

给定一个字符串，以空格为分隔符，反转每个单词中的字母，若单词中包含如数字等其他非字母时不进行反转。

反转后计算其中含有相对开音节结构的子串个数（连续的子串中部分字符可以重复）。

输入描述:

字符串，以空格分割的多个单词，字符串长度<10000，字母只考虑小写

输出描述:

含有相对开音节结构的子串个数，注：个数<10000

示例 1

输入

ekam a ekac

输出

2

说明

反转后为 make a cake 其中 make、cake 为相对开音节子串，返回 2

示例 2

输入

!ekam a ekekac

输出

2

说明

反转后为!ekam a cakeke 因!ekam 含非英文字符所以未反转，其中 cake、keke 为相对开音节子串，返回 2

```
public class ZT59 {
    public static void main(String[] args) {
        Scanner sc =new Scanner(System.in);
        String[] input = sc.nextLine().split(" ");
        int total = 0;
        for (int i = 0; i < input.length; i++) {
            String content = input[i];
            boolean flag = true;
            for (int j = 0; j < content.length(); j++) {
                if (content.charAt(j) < 'a' ||  content.charAt(j)> 'z'){
                    flag = false;
                    break;
                }
            }
            if (flag){
                total += checkKai(content);
            }
        }
        System.out.println(total);
    }

    private static int checkKai(String str){
        String strTemp = "";
        for (int i = str.length()-1; i >=0 ; i--) {
            strTemp += str.charAt(i);
        }

        int left = 0;
        int right = 0;
        int total = 0;
        String tem = "";
```

```
        while (right<=str.length()){
            tem = strTemp.substring(left,right);
            if (tem.length() < 4 ){
                right++;
                continue;
            }
            if (checkTrue(tem)) {
                total++;
            }
            left++;
            right++;

        }
        return total;
    }

    private static boolean checkTrue(String str){
        //辅音+元音（aeiou）+辅音(r 除外)+e blame
        if (!yuanyin.contains(str.charAt(0)) && 'e' == str.charAt(str.length()-1)){//首位是辅音 末尾是元音
            int temYuan = 0;
            for (int i = 1; i < str.length()-1; i++) {//中间 2 位或 3 位
                if (yuanyin.contains(str.charAt(i))){
                    temYuan = i;
                }
                if (temYuan!= 0 && !yuanyin.contains(str.charAt(i)) && str.charAt(i) != 'r' && i> temYuan){
                    return true;
                }
            }
        }
        return false;
    }

    private static List<Character> yuanyin = new ArrayList<>();
    static {
        char[] f1 = {'a','e','i','o','u'};
        for (int i = 0; i < f1.length; i++) {
            yuanyin.add(f1[i]);
        }
    }
}
1
2
3
4
```

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

060 【消消乐游戏】

游戏规则：输入一个只包含英文字母的字符串，字符串中的两个字母如果相邻且相同，就可以消除。

在字符串上反复执行消除的动作，直到无法继续消除为止，此时游戏结束。

输出最终得到的字符串长度。

输入描述:

输入原始字符串 str ，只能包含大小写英文字母，字母的大小写敏感， str 长度不超过 100。

输出描述:

输出游戏结束后，最终得到的字符串长度

示例 1

输入

gg

输出

0

说明

gg 可以直接消除，得到空串，长度为 0

示例 2

输入

mMbccbc

输出

3

说明

在 mMbccbc 中，可以先消除 cc ；此时字符串变成 mMbbc ，可以再消除 bb ；此时字符串变成 mMc ，此时没有相邻且相同的字符，无法继续消除。最终得到的字符串为 mMc ，长度为 3

备注:

输入中包含 非大小写英文字母 时，均为异常输入，直接返回 0

```
public class ZT60 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String input = sc.nextLine();
        while (true){
            char pre = input.charAt(0);
            boolean flag = false;
            for (int i = 1; i < input.length(); i++) {
                if (input.charAt(i) == pre){
                    input = input.substring(0,i-1) + input.substring(i+1);
                    flag = true;
                    break;
                }else {
                    pre = input.charAt(i);
                }
            }
            if (!flag || input.length() == 0){
                System.out.println(input.length());
                return;
            }
        }
    }
}
```

————————————————————

061 【寻找身高相近的小朋友】

062 【寻找相同子串】

063 【一种字符串压缩表示的解压】

064 【英文输入法】

065 【用户调度问题】

066 【用连续自然数之和来表达整数】

067 【找车位】

068 【找出符合要求的字符串子串】

069 【找朋友】

070 【找终点】

061 【寻找身高相近的小朋友】

小明今年升学到小学一年级，来到新班级后发现其他小朋友们身高参差不齐，然后就想基于各小朋友和自己的身高差对他们进行排序，请帮他实现排序。

输入描述:

第一行为正整数 H 和 N，0<H<200，为小明的身高，0<N<50，为新班级其他小朋友个数。

第二行为 N 个正整数 H1-HN，分别是其他小朋友的身高，取值范围 0<Hi<200（1<=i<=N），且 N 个正整数各不相同。

输出描述:

输出排序结果，各正整数以空格分割。和小明身高差绝对值最小的小朋友排在前面，和小明身高差绝对值最大的小朋友排在最后，如果两个小朋友和小明身高差一样，则个子较小的小朋友排在前面。

示例 1

输入

100 10

95 96 97 98 99 101 102 103 104 105

输出

99 101 98 102 97 103 96 104 95 105

说明

小明身高 100，班级学生 10 个，身高分别为 95 96 97 98 99 101 102 103 104 105，按身高差排序后结果为：99 101 98 102 97 103 96 104 95 105。

```
public class ZT61 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] input = sc.nextLine().split(" ");
        int myHigh = Integer.parseInt(input[0]);
        int count = Integer.parseInt(input[1]);
        String[] students = sc.nextLine().split(" ");
        List<Student> list = new ArrayList<>();
        for (int i = 0; i < count; i++) {
            int highTemp = Integer.parseInt(students[i]);
            list.add(new Student(highTemp, Math.abs(myHigh - highTemp)));
        }
        list.sort(null);
        for (int i = 0; i < list.size(); i++) {
            System.out.print(list.get(i).high + " ");
        }
        System.out.println();
    }
    private static class Student implements Comparable<Student>{
        private int high;
        private int gap;

        public Student(int high, int gap) {
            this.high = high;
            this.gap = gap;
        }
```

```
        @Override
        public int compareTo(Student stu) {
            if (stu.gap != this.gap){
                return this.gap - stu.gap ;
            }else {
                return this.high - stu.high ;
            }
        }
    }
}
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
```

34

35

36

37

062 【寻找相同子串】

给你两个字符串 t 和 p，要求从 t 中找到一个和 p 相同的连续子串，并输出该字串第一个字符的下标。

输入描述:

输入文件包括两行，分别表示字符串 t 和 p，保证 t 的长度不小于 p，且 t 的长度不超过 1000000，p 的长度不超过 10000。

输出描述:

如果能从 t 中找到一个和 p 相等的连续子串，则输出该子串第一个字符在 t 中的下标（下标从左到右依次为 1,2,3,…）；如果不能则输出”No”；如果含有多个这样的子串，则输出第一个字符下标最小的。

示例 1

输入

AVERDXIVYERDIAN

RDXI

输出

4

```
public class ZT62 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String search = sc.nextLine();
        String target = sc.nextLine();
        if (!search.contains(target)){
            System.out.println("no");
            return;
        }
        int i = search.indexOf(target);
        System.out.println(i+1);
    }
}
```

1

2

3

4

5

6

7

8

9

10

11

12

13

063 【一种字符串压缩表示的解压】

有一种简易压缩算法：针对全部由小写英文字母组成的字符串，将其中连续超过两个相同字母的部分压缩为连续个数加该字母，其他部分保持原样不变。例如：字符串“aaabbccccd”经过压缩成为字符串“3abb4cd”。 请您编写解压函数，根据输入的字符串，判断其是否为合法压缩过的字符串，若输入合法则输出解压缩后的字符串，否则输出字符串“!error”来报告错误。

输入描述:

输入一行，为一个 ASCII 字符串，长度不会超过 100 字符，用例保证输出的字符串长度也不会超过 100 字符

输出描述:

若判断输入为合法的经过压缩后的字符串，则输出压缩前的字符串；若输入不合法，则输出字符串“!error”。

示例 1

输入

4dff

输出

ddddff

说明

4d 扩展为 dddd，故解压后的字符串为 ddddff

示例 2

输入

2dff

输出

!error

说明

两个 d 不需要压缩，故输入不合法

示例 3

输入

4d@A

输出

!error

说明

全部由小写英文字母组成的字符串压缩后不会出现特殊字符@和大写字母 A，故输入不合法

```
public class ZT63 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String input = sc.nextLine();
        StringBuilder res = new StringBuilder();
        for (int i = 0; i < input.length(); i++) {
            char ch = input.charAt(i);
            int times = 1;
            StringBuilder count = new StringBuilder();
            if ((ch>= '0' && ch<= '9') ){//34a
                for (int j = i; j < input.length(); j++) {
                    ch = input.charAt(j);
                    if (ch>= '0' && ch<= '9'){
                        count.append(input.charAt(j));
```

```
                    i++;
                    continue;
                }
                break;
            }
            times = Integer.parseInt(count.toString());
            if (times<= 2){
                System.out.println("!error");
                return;
            }
        }
        ch = input.charAt(i);//下一个元素
        if (ch>= 'a' && ch<= 'z'){
            while (times > 0 ){
                res.append(ch);
                times--;
            }
        }else {
            System.out.println("!error");
            return;
        }
    }
    System.out.println(res);
  }
}
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

064 【英文输入法】

主管期望你来实现英文输入法单词联想功能。需求如下：

依据用户输入的单词前缀，从已输入的英文语句中联想出用户想输入的单词，按字典序输出联想到的单词序列，如果联想不到，请输出用户输入的单词前缀。

注意：

英文单词联想时，区分大小写

缩略形式如”don’t”，判定为两个单词，”don”和”t”

输出的单词序列，不能有重复单词，且只能是英文单词，不能有标点符号

输入描述:

输入为两行。

首行输入一段由英文单词 word 和标点符号组成的语句 str；

接下来一行为一个英文单词前缀 pre。

0 < word.length() <= 20

0 < str.length <= 10000

0 < pre <= 20

输出描述:

输出符合要求的单词序列或单词前缀，存在多个时，单词之间以单个空格分割

示例 1

输入

I love you

He

输出

He

说明

从用户已输入英文语句”I love you”中提炼出“I”、“love”、“you”三个单词，接下来用户输入“He”，从已输入信息中无法联想到任何符合要求的单词，因此输出用户输入的单词前缀。

示例 2

输入

The furthest distance in the world, Is not between life and death, But when I stand in front of you, Yet you don’t know that I love you.

f

输出

front furthest

说明

从用户已输入英文语句”The furthest distance in the world, Is not between life and death, But when I stand in frontof you, Yet you dont know that I love you.”中提炼出的单词，符合“f”作为前缀的，有“furthest”和“front”，按字典序排序并在单词间添加空格后输出，结果为“front furthest”。

```
public class ZT64 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String input = sc.nextLine();
        String target = sc.nextLine();
        List<String> list = new ArrayList<>();
        StringBuilder word = new StringBuilder();
        for (int i = 0; i < input.length(); i++) {
            char ch = input.charAt(i);
            if ((ch >= 'a' && ch <= 'z') || ( ch>= 'A' && ch<= 'Z') ){
                word.append(ch);
            }else {
                list.add(word.toString());
                word = new StringBuilder();
            }
        }
        List<String> res = new ArrayList<>();
        for (int i = 0; i < list.size(); i++) {
            if (list.get(i).startsWith(target)) {
                res.add(list.get(i));
            }
        }
        Collections.sort(res);
        if (res.size() > 0){
            for (int i = 0; i < res.size(); i++) {
                System.out.print(res.get(i) + " ");
            }
            System.out.println();
        }else {
            System.out.println(target);
```

```
        }
    }
}
```

065 【用户调度问题】

在通信系统中，一个常见的问题是对用户进行不同策略的调度，会得到不同的系统消耗和性能。

假设当前有 n 个待串行调度用户，每个用户可以使用 A/B/C 三种不同的调度策略，不同的策略会消耗不同的系统资源。请你根据如下规则进行用户调度，并返回总的消耗资源数。

规则：

1、相邻的用户不能使用相同的调度策略，例如，第 1 个用户使用了 A 策略，则第 2 个用户只能使用 B 或者 C 策略。

2、对单个用户而言，不同的调度策略对系统资源的消耗可以归一化后抽象为数值。例如，某用户分别使用 A/B/C 策略的系统消耗分别为 15/8/17。

3、每个用户依次选择当前所能选择的对系统资源消耗最少的策略（局部最优），如果有多个满足要求的策略，选最后一个。

输入描述:

第一行表示用户个数 n

接下来每一行表示一个用户分别使用三个策略的系统消耗 resA resB resC

输出描述:

最优策略组合下的总的系统资源消耗数

示例 1

输入

3

15 8 17

12 20 9

11 7 5

输出

24

说明

1 号用户使用 B 策略，2 号用户使用 C 策略，3 号用户使用 B 策略。系统资源消耗: 8 + 9 + 7 = 24。

备注:

所有策略对系统的资源消耗均为正整数，n < 1000

```
public class ZT65 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int count = Integer.parseInt(sc.nextLine());
        List<List<Integer>> lists = new ArrayList<>();
        for (int i = 0; i < count; i++) {
            String[] input = sc.nextLine().split(" ");
            List<Integer> list = new ArrayList<>();
            for (int j = 0; j < 3; j++) {
                list.add(Integer.parseInt(input[j]));
            }
            lists.add(list);
        }
        //执行策略
        int pre = -1;
        int total = 0;
        for (int i = 0; i < lists.size(); i++) {
            List<Integer> list = lists.get(i);
            int min = Integer.MAX_VALUE;
            if (i == 0){
                for (int j = 0; j < 3; j++) {
                    if (min >= list.get(j)){
                        min = list.get(j);
                        pre = j;
                    }
```

```
                }
            }else {
                int tem = 0;
                for (int j = 0; j < 3; j++) {
                    if (j != pre && min >= list.get(j)){
                        min = list.get(j);
                        tem = j;
                    }
                }
                pre = tem;
            }
            total += min;
        }
        System.out.println(total);
    }
}
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
```

29

30

31

32

33

34

35

36

37

38

39

40

41

066 【用连续自然数之和来表达整数】

一个整数可以由连续的自然数之和来表示。给定一个整数，计算该整数有几种连续自然数之和的表达式，且打印出每种表达式。

输入描述:

一个目标整数 T (1 <=T<= 1000)

输出描述:

该整数的所有表达式和表达式的个数。如果有多种表达式，输出要求为：

1.自然数个数最少的表达式优先输出

2.每个表达式中按自然数递增的顺序输出，具体的格式参见样例。在每个测试数据结束时，输出一行”Result:X”，其中 X 是最终的表达式个数。

示例 1

输入

9

输出

9=9

9=4+5

9=2+3+4

Result:3

说明

整数 9 有三种表示方法，第 1 个表达式只有 1 个自然数，最先输出，第 2 个表达式有 2 个自然数，第 2 次序输出，第 3 个表达式有 3 个自然数，最后输出。每个表达式中的自然数都是按递增次序输出的。

数字与符号之间无空格

示例 2

输入

10

输出

10=10

10=1+2+3+4

Result:2

```
public class ZT66 {
    public static void main(String[] args) {
```

```
        Scanner sc = new Scanner(System.in);
        int sum = sc.nextInt();
        int n = 1;
        while (n< sum/2){
            if (2* sum % n == 0 && (2* sum/n -n +1) %2 == 0 && (2* sum/n -n +1)/2 > 0){
                int x1 = (2* sum/n -n +1)/2;
                System.out.print(sum + "=");
                for (int i = 0; i <n ; i++) {
                    if (i == n-1){
                        System.out.print(x1++);
                    }else {
                        System.out.print(x1++ + "+");
                    }
                }
                System.out.println();
            }
            n++;
        }
    }
}
//输入
//2*sum = n(2x + n-1) n 取[1,sum]
```

23

24

067 【找车位】

停车场有一横排车位，0 代表没有停车，1 代表有车。至少停了一辆车在车位上，也至少有一个空位没有停车。

为了防剐蹭，需为停车人找到一个车位，使得距停车人的车最近的车辆的距离是最大的，返回此时的最大距离。

输入描述:

1、一个用半角逗号分割的停车标识字符串，停车标识为 0 或 1，0 为空位，1 为已停车。

2、停车位最多 100 个。

输出描述:

输出一个整数记录最大距离。

示例 1

输入

1,0,0,0,0,1,0,0,1,0,1

输出

2

说明

当车停在第 3 个位置上时，离其最近的的车距离为 2（1 到 3）。

当车停在第 4 个位置上时，离其最近的的车距离为 2（4 到 6）。

其他位置距离为 1。

因此最大距离为 2。

```
public class ZT67 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] input = sc.nextLine().split(",");
        int[] arr = new int[input.length];
        for (int i = 0; i < input.length; i++) {
            arr[i] = Integer.parseInt(input[i]);
        }
        int max = 0;
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == 1){
                continue;
            }
            max = Math.max(max,calcDis(arr,i));
        }
        System.out.println(max);
    }
    //1,0,0,0,1,0,0,1
    private static int calcDis(int[] arr,int idx){
        int left = 0;
        int right = 0;
        for (int i = 0; i < idx; i++) {
            if (arr[i] == 1){
```

```
                left = i;
            }
        }
        for (int i = idx; i < arr.length; i++) {
            if (arr[i] == 1){
                right =i;
                break;
            }
        }
        return Math.min(idx - left,right - idx);
    }
}
```

33

34

35

068 【找出符合要求的字符串子串】

给定两个字符串，从字符串 2 中找出字符串 1 中的所有字符，去重并按照 ASCII 值从小到大排序

输入字符串 1：长度不超过 1024

输入字符串 2：长度不超过 1000000

字符范围满足 ASCII 编码要求，按照 ASCII 的值由小到大排序

输入描述:

bach

bbaaccedfg

输出描述:

abc

示例 1

输入

fach

bbaaccedfg

输出

acf

说明

备注:

输入字符串 1 为给定字符串 bach，输入字符串 2 bbaaccedfg

从字符串 2 中找出字符串 1 的字符，去除重复的字符，并且按照 ASCII 值从小到大排序，得到输出的结果为 abc。

字符串 1 中的字符 h 在字符串 2 中找不到不输出。

```
public class ZT68 {
    public static void main(String[] args) {
        Scanner sc =new Scanner(System.in);
        String target = sc.nextLine();
        String search = sc.nextLine();
        List<Character> list = new ArrayList<>();
        for (int i = 0; i < target.length(); i++) {
            Character tem = target.charAt(i);
            if (search.contains(String.valueOf(tem))) {
                list.add(tem);
            }
        }
        Collections.sort(list);
        for (int i = 0; i < list.size(); i++) {
            System.out.print(list.get(i));
        }
        System.out.println();
    }
}
```

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

069 【找朋友】

在学校中，N 个小朋友站成一队， 第 i 个小朋友的身高为 height[i]，

第 i 个小朋友可以看到的第一个比自己身高更高的小朋友 j，那么 j 是 i 的好朋友(要求 j > i)。

请重新生成一个列表，对应位置的输出是每个小朋友的好朋友位置，如果没有看到好朋友，请在该位置用 0 代替。

小朋友人数范围是 [0, 40000]。

输入描述:

第一行输入 N，N 表示有 N 个小朋友

第二行输入 N 个小朋友的身高 height[i]，都是整数

输出描述:

输出 N 个小朋友的好朋友的位置

示例 1

输入

2

100 95

输出

0 0

说明

第一个小朋友身高 100，站在队尾位置，向队首看，没有比他身高高的小朋友，所以输出第一个值为 0。

第二个小朋友站在队首，前面也没有比他身高高的小朋友，所以输出第二个值为 0。

示例 2

输入

8

123 124 125 121 119 122 126 123

输出

1 2 6 5 5 6 0 0

说明

123 的好朋友是 1 位置上的 124

124 的好朋友是 2 位置上的 125

125 的好朋友是 6 位置上的 126

以此类推

```
public class ZT69 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int count = Integer.parseInt(sc.nextLine());
        String[] input = sc.nextLine().split(" ");
        int[] arr = new int[count];
        for (int i = 0; i < input.length; i++) {
           //右边第一个比自己身高高的人
            int idx = 0;
            for (int j = i+1; j < input.length; j++) {
                if (Integer.parseInt(input[i]) < Integer.parseInt(input[j])){
                    idx = j;
                    break;
                }
            }
            arr[i] = idx;
        }
        for (int i = 0; i < arr.length; i++) {
            if (i!= arr.length-1){
                System.out.print(arr[i] + " ");
            }else {
                System.out.print(arr[i]);
            }
        }
    }
}
1
2
3
4
5
6
7
8
9
10
11
12
```

13

14

15

16

17

18

19

20

21

22

23

24

25

26

070 【找终点】

给定一个正整数数组，设为 nums，最大为 100 个成员，求从第一个成员开始，正好走到数组最后一个成员，所使用的最少步骤数。

要求：

1、第一步必须从第一元素开始，且 1<=第一步的步长<len/2;（len 为数组的长度，需要自行解析）。

2、从第二步开始，只能以所在成员的数字走相应的步数，不能多也不能少，如果目标不可达返回-1，只输出最少的步骤数量。

3、只能向数组的尾部走，不能往回走。

输入描述:

由正整数组成的数组，以空格分隔，数组长度小于 100，请自行解析数据数量。

输出描述:

正整数，表示最少的步数，如果不存在输出-1

示例 1

输入

7 5 9 4 2 6 8 3 5 4 3 9

输出

2

说明

第一步： 第一个可选步长选择 2，从第一个成员 7 开始走 2 步，到达 9；第二步： 从 9 开始，经过自身数字 9 对应的 9 个成员到最后。

示例 2

输入

1 2 3 7 1 5 9 3 2 1

输出

-1

```
public class ZT70 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] input = sc.nextLine().split(" ");
        int firstStepMin = 2;
```

```
        int firstStepMax = input.length/2;
        int minStep = Integer.MAX_VALUE;
        int cur = 0;
        int tempStep = 0;
        for (int step = firstStepMin; step < firstStepMax; step++) {
            int target = input.length - 1;
            cur = step;
            tempStep++;
            while (cur < target){
                cur += Integer.parseInt(input[cur]);
                tempStep++;
                if (cur == target){
                    minStep = Math.min(minStep,tempStep);
                    break;
                }
                if (cur> target){
                    tempStep = 0;
                    cur = 0;
                    break;
                }
            }
        }
        if (minStep > input.length){
            System.out.println(-1);
        }else {
            System.out.println(minStep);
        }
    }
}
```

————————————————————

原文链接：https://blog.csdn.net/csfun1/article/details/124551079

071 【整数编码】

072 【整数对最小和】

073 【整型数组按个位值排序】

074 【执行时长】

075 【字符串变换最小字符串】

076 【字符串分割】

077 【字符串加密】

078 【字符串筛选排序】

079 【字符串统计】

080 【字符串序列判定】

071 【整数编码】

实现一种整数编码方法，使得待编码的数字越小，编码后所占用的字节数越小。

编码规则如下：

1、编码时 7 位一组，每个字节的低 7 位用于存储待编码数字的补码。

2、字节的最高位表示后续是否还有字节，置 1 表示后面还有更多的字节，置 0 表示当前字节为最后一个字节。

3、采用小端序编码，低位和低字节放在低地址上。

3、编码结果按 16 进制数的字符格式输出，小写字母需转换为大写字母。

输入描述:

输入的为一个字符串表示的非负整数

输出描述:

输出一个字符串，表示整数编码的 16 进制码流

示例 1

输入

0

输出

00

说明

输出的 16 进制字符，不足两位的前面补 0，如 00、01、02。

示例 2

输入

100

输出

64

说明

100 的二进制表示为 0110 0100，只需要一个字节进行编码；

字节的最高位置 0，剩余 7 位存储数字 100 的低 7 位（110 0100），所以编码后的输出为 64。

示例 3

输入

1000

输出

E807

说明

1000 的二进制表示为 0011 1110 1000，至少需要两个字节进行编码；

第一个字节最高位置 1，剩余的 7 位存储数字 1000 的第一个低 7 位（110 1000），所以第一个字节的二进制为 1110 1000，即 E8；

第二个字节最高位置 0，剩余的 7 位存储数字 1000 的第二个低 7 位（000 0111），所以第一个字节的二进制为 0000 0111，即 07；

采用小端序编码，所以低字节 E8 输出在前，高字节 07 输出在后。

备注:

待编码的数字取值范围为[0, 1<<64 - 1]

```
public class ZT71 {
    public static void main(String[] args) {
        char[] arr = {'A','B','C','D','E','F','G','H','I','J','K','L',
            'M','N','O','P','K','R','S','T','U','V','W','X','Y','Z'};
```

```
        Scanner sc = new Scanner(System.in);
        int input = sc.nextInt();//1000
        String str = Integer.toBinaryString(input);//11 1110 1000
        boolean flag = false;
        StringBuilder sb = new StringBuilder();
        while (str.length() > 0){//先找到低 7 位
            if (str.length() > 7){
                flag = true;
            }else {//2 进制不足 7 位时补 0
                flag = false;
                int mid = 7 - str.length();
                for (int i = 0; i < mid; i++) {
                    str = "0" + str;
                }
            }
            String sub = str.substring(str.length() - 7);
            str = str.substring(0,str.length() - 7);
            if (flag){
                sub = "1" + sub;
            }else {
                sub = "0" + sub;
            }
            String res = Integer.toHexString(Integer.parseInt(sub, 2));
            if (res.length()<2){//16 进制不足 2 位补 0
                res = "0" + res;
            }
            for (int i = 0; i < res.length(); i++) {
                char ch = res.charAt(i);
                if (ch>= 'a' && ch<= 'z'){
                    ch = arr[ch - 'a'];
                }
                sb.append(ch);
            }
        }
        //每个字符如果是小写要转成大写
        System.out.println(sb);
    }
}
1
2
3
4
5
6
```

7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
善用 API 之：string.toUpperCase() 直接将小写改成大写

072 【整数对最小和】

给定两个整数数组 array1、array2，数组元素按升序排列。假设从 array1、array2 中分别取出一个元素可构成一对元素，现在需要取出 k 对元素，并对取出的所有元素求和，计算和的最小值

注意：两对元素如果对应于 array1、array2 中的两个下标均相同，则视为同一对元素。

输入描述:

输入两行数组 array1、array2，每行首个数字为数组大小 size(0 < size <= 100);

0 < array1[i] <= 1000

0 < array2[i] <= 1000

接下来一行为正整数 k

0 < k <= array1.size() * array2.size()

输出描述:

满足要求的最小和

示例 1

输入

3 1 1 2

3 1 2 3

2

输出

4

说明

用例中，需要取 2 对元素

取第一个数组第 0 个元素与第二个数组第 0 个元素组成 1 对元素[1,1];

取第一个数组第 1 个元素与第二个数组第 0 个元素组成 1 对元素[1,1];

求和为 1+1+1+1=4，为满足要求的最小和

```
public class ZT72 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] str1 = sc.nextLine().split(" ");
        String[] str2 = sc.nextLine().split(" ");
        int count = Integer.parseInt(sc.nextLine());
        List<Integer> list = new ArrayList<>();
        for (int i = 0; i < str1.length; i++) {
            for (int j = 0; j < str2.length; j++) {
                list.add(Integer.parseInt(str1[i]) + Integer.parseInt(str2[j]));
            }
        }
        Collections.sort(list);
        int res = 0;
        for (int i = 0; i < count; i++) {
            res += list.get(i);
        }
        System.out.println(res);
    }
}
```

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

073 【整型数组按个位值排序】

给定一个非空数组（列表），其元素数据类型为整型，请按照数组元素十进制最低位从小到大进行排序，十进制最低位相同的元素，相对位置保持不变。

当数组元素为负值时，十进制最低位等同于去除符号位后对应十进制值最低位。

输入描述:

给定一个非空数组，其元素数据类型为32位有符号整数，数组长度[1, 1000]

输出描述:

输出排序后的数组

示例1

输入

1,2,5,-21,22,11,55,-101,42,8,7,32

输出

1,-21,11,-101,2,22,42,32,5,55,7,8

```
public class ZT73 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] splits = sc.nextLine().split(",");
        Map<Integer, List<Integer>> map = new HashMap<>();
        for (int i = 0; i < splits.length; i++) {
            int abs = Math.abs(Integer.parseInt(splits[i]));
            int res = abs % 10;
            List<Integer> list = map.getOrDefault(res, new ArrayList<>());
            list.add(Integer.parseInt(splits[i]));
            map.put(res,list);
        }
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < 10; i++) {
            List<Integer> list = map.get(i);
```

```
            if (list!= null){
                for (int j = 0; j < list.size(); j++) {
                    sb.append(list.get(j)).append(",");
                }
            }
        }
        sb.deleteCharAt(sb.length()-1);
        System.out.println(sb);
    }
}
```

074 【执行时长】

为了充分发挥 GPU 算力，需要尽可能多的将任务交给 GPU 执行，现在有一个任务数组，数组元素表示在这 1 秒内新增的任务个数且每秒都有新增任务，假设 GPU 最多一次执行 n 个任务，一次执行耗时 1 秒，在保证 GPU 不空闲情况下，最少需要多长时间执行完成

输入描述:

第一个参数为 GPU 一次最多执行的任务个数，取值范围[1, 10000]

第二个参数为任务数组长度，取值范围[1, 10000]

第三个参数为任务数组，数字范围[1, 10000]

输出描述:

执行完所有任务最少需要多少秒

示例 1

输入

3

5

1 2 3 4 5

输出

6

说明

一次最多执行 3 个任务，最少耗时 6s

示例 2

输入

4

5

5 4 1 1 1

输出

5

说明

一次最多执行 4 个任务，最少耗时 5s

```
public class ZT74 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int bing = Integer.parseInt(sc.nextLine());
        int times = Integer.parseInt(sc.nextLine());
        String[] inputs = sc.nextLine().split(" ");
        int left = 0;
        for (int i = 0; i < times; i++) {
            int th = Integer.parseInt(inputs[i]);
            left = Math.max(0,left + th - bing);

        }
        int res = times;
        if (left % bing == 0){
            res += left / bing;
        }else {
            res += left / bing + 1;
        }
        System.out.println(res);
    }
}
1
2
3
```

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

075 【字符串变换最小字符串】

给定一个字符串 s，最多只能进行一次变换，返回变换后能得到的最小字符串（按照字典序进行比较）。

变换规则：交换字符串中任意两个不同位置的字符。

输入描述:

一串小写字母组成的字符串 s

输出描述:

按照要求进行变换得到的最小字符串

示例 1

输入

abcdef

输出

abcdef

说明

abcdef 已经是最小字符串，不需要交换

示例 2

输入

bcdefa

bacdefa

输出

acdefb

aacdefb[abcdefa 排序更大]

说明

a 和 b 进行位置交换，可以等到最小字符串

备注:

s 是都是小写字符组成

1<=s.length<=1000

```
public class ZT75 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String input = sc.nextLine();
        char[] chars = input.toCharArray();
        Arrays.sort(chars);//最优解
        String str = input;
        for (int i = 0; i < input.length(); i++) {
            if (chars[i] == input.charAt(i)) {
                continue;
            }else {
                char aChar = chars[i];
                int i1 = input.lastIndexOf(String.valueOf(aChar));
                //i1 和 i 对调顺序即可
                str = input.substring(0,i) + aChar + input.substring(i+1,i1) + input.charAt(i) +
input.substring(i1+1);
                break;
            }
        }
        System.out.println(str);
    }
}
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
```

076 【字符串分割】

给定一个非空字符串 S，其被 N 个‘-’分隔成 N+1 的子串，给定正整数 K，要求除第一个子串外，其余的子串每 K 个字符组成新的子串，并用‘-’分隔。对于新组成的每一个子串，如果它含有的小写字母比大写字母多，则将这个子串的所有大写字母转换为小写字母；反之，如果它含有的大写字母比小写字母多，则将这个子串的所有小写字母转换为大写字母；大小写字母的数量相等时，不做转换。

输入描述:

输入为两行，第一行为参数 K，第二行为字符串 S。

输出描述:

输出转换后的字符串。

示例 1

输入

3

12abc-abCABc-4aB@

输出

12abc-abc-ABC-4aB-@

说明

子串为 12abc、abCABc、4aB@，第一个子串保留，后面的子串每 3 个字符一组为 abC、ABc、4aB、@，abC 中小写字母较多，转换为 abc，ABc 中大写字母较多，转换为 ABC，4aB 中大小写字母都为 1 个，不做转换，@中没有字母，连起来即 12abc-abc-ABC-4aB-@

示例 2

输入

12

12abc-abCABc-4aB@

输出

12abc-abCABc4aB@

说明

子串为 12abc、abCABc、4aB@，第一个子串保留，后面的子串每 12 个字符一组为 abCABc4aB@，这个子串中大小写字母都为 4 个，不做转换，连起来即 12abc-abCABc4aB@

```
public class ZT76 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int count = Integer.parseInt(sc.nextLine());
        String[] split = sc.nextLine().split("-");
        List<String> listStr = new ArrayList<>();
        StringBuilder temp = new StringBuilder();
        for (int i = 1; i < split.length; i++) {
            temp.append(split[i]);
        }
        String tem = temp.toString();
        while (tem.length()>0){
            if (tem.length() >= count){//截取
                listStr.add(tem.substring(0,count));
                tem = tem.substring(count);
            }else {
```

```
                listStr.add(tem);
                tem = "";
            }
        }
        StringBuilder sb= new StringBuilder();
        sb.append(split[0]).append("-");
        for (int i = 0; i < listStr.size(); i++) {
            String s1 = listStr.get(i);
            sb.append(reBalance(s1)).append("-");
        }
        sb.deleteCharAt(sb.length() -1 );
        System.out.println(sb);
    }
    private static String reBalance(String str){
        int xiao = 0;
        int da = 0;
        for (int i = 0; i < str.length(); i++) {
            char ch = str.charAt(i);
            if (ch>= 'a' && ch<= 'z'){
                xiao++;
            }else if (ch >= 'A' && ch <= 'Z'){
                da++;
            }
        }
        if (xiao > da ){
            return str.toLowerCase();
        }else if (da > xiao){
            return str.toUpperCase();
        }else {
            return str;
        }
    }
}
1
2
3
4
5
6
7
8
9
10
11
```

077 【字符串加密】

给你一串未加密的字符串 str，通过对字符串的每一个字母进行改变来实现加密，加密方式是在每一个字母 str[i]偏移特定数组元素 a[i]的量，数组 a 前三位已经赋值：a[0]=1,a[1]=2,a[2]=4。当 i>=3 时，数组元素 a[i]=a[i-1]+a[i-2]+a[i-3]，

例如：原文 abcde 加密后 bdgkr，其中偏移量分别是 1,2,4,7,13。

输入描述:

第一行为一个整数 n（1<=n<=1000），表示有 n 组测试数据，每组数据包含一行，原文 str（只含有小写字母，0<长度<=50）。

输出描述:

每组测试数据输出一行，表示字符串的密文

示例 1

输入

1

xy

输出

ya

说明

第一个字符 x 偏移量是 1，即为 y，第二个字符 y 偏移量是 2，即为 a

示例 2

输入

2

xy

abcde

输出

ya

bdgkr

说明

第二行输出字符偏移量分别为 1、2、4、7、13

备注:

解答要求

时间限制：2000ms,内存限制：64MB

```
public class ZT77 {
    private static int[] arrOff = new int[50];

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int count = Integer.parseInt(sc.nextLine());
        char[] arr = {'a','b','c','d','e','f','g','h','i','j','k','l','m',
            'n','o','p','q','r','s','t','u','v','w','x','y','z'};
        arrOff[0] = 1;
        arrOff[1] = 2;
        arrOff[2] = 4;
        for (int i = 3; i < 50; i++) {
            arrOff[i] = arrOff[i-1] + arrOff[i-2] + arrOff[i-3];
        }
        for (int i = 0; i < count; i++) {
            System.out.println(handleStr(sc.nextLine(),arr));
        }
    }

    public static String handleStr(String str,char[] arr){
```

```
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < str.length(); i++) {
            char ch = str.charAt(i);
            int offset = arrOff[i];
            ch = arr[(ch -'a' +  offset) % 26];
            sb.append(ch);
        }
        return sb.toString();
    }
}
```

078 【字符串筛选排序】

输入一个由 n 个大小写字母组成的字符串，按照 Ascii 码值从小到大的排序规则，查找字符串中第 k 个最小 ascii 码值的字母（k>=1），输出该字母所在字符串的位置索引(字符串的第一个字符位置索引为 0）。

k 如果大于字符串长度，则输出最大 ascii 值的字母所在字符串的位置索引，如果有重复的字母，则输出字母的最小位置索引。

输入描述:

第一行输入一个由大小写字母组成的字符串

第二行输入 k，k 必须大于 0，k 可以大于输入字符串的长度

输出描述:

输出字符串中第 k 个最小 ascii 码值的字母所在字符串的位置索引。k 如果大于字符串长度，则输出最大 ascii 值的字母所在字符串的位置索引，如果第 k 个最小 ascii 码值的字母存在重复，则输出该字母的最小位置索引。

示例 1

输入

AbCdeFG

3

输出

5

说明

根据 ascii 码值排序，第 3 个最小 ascii 码值的字母为 F，F 在字符串中的位置索引为 5（0 为字符串的第一个字母位置索引）

示例 2

输入

fAdDAkBbBq

4

输出

6

说明

根据 ascii 码值排序，前 4 个字母为 AABB ，由于 B 重复，则只取 B 的（第一个）最小位置索引 6 ，而不是第二个 B 的位置索引 8

```
public class ZT78 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String input = sc.nextLine();
        int turn = Integer.parseInt(sc.nextLine());
        char[] chars = input.toCharArray();
        Arrays.sort(chars);
        char ch = ' ';
        if (turn > chars.length){
            ch = chars[chars.length-1];
            System.out.println(input.lastIndexOf(String.valueOf(ch)));
        }else {
            ch = chars[turn-1];
            System.out.println(input.indexOf(ch));
        }
    }
}
```

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

079 【字符串统计】

给定两个字符集合，一个为全量字符集，一个为已占用字符集。已占用的字符集中的字符不能再使用，要求输出剩余可用字符集。

输入描述:

1、输入为一个字符串，一定包含@符号。@前的为全量字符集，@后的字为已占用字符集。

2、已占用字符集中的字符一定是全量字符集中的字符。字符集中的字符跟字符之间使用英文逗号分隔。

3、每个字符都表示为字符加数字的形式，用英文冒号分隔，比如 a:1，表示 1 个 a 字符。

4、字符只考虑英文字母，区分大小写，数字只考虑正整形，数量不超过 100。

5、如果一个字符都没被占用，@标识仍然存在，例如 a:3,b:5,c:2@

输出描述:

输出可用字符集，不同的输出字符集之间回车换行。

注意，输出的字符顺序要跟输入一致。不能输出 b:3,a:2,c:2

如果某个字符已全被占用，不需要再输出。

示例 1

输入

a:3,b:5,c:2@a:1,b:2

输出

a:2,b:3,c:2

说明

全量字符集为 3 个 a，5 个 b，2 个 c。

已占用字符集为 1 个 a，2 个 b。

由于已占用字符不能再使用，因此，剩余可用字符为 2 个 a，3 个 b，2 个 c。

因此输出 a:2,b:3,c:2

```
public class ZT79 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] split = sc.nextLine().split("@");
        if (split.length<2){
            System.out.println(split[0]);
```

```
            return;
        }
        List<String> list = new ArrayList<>();
        Map<String,Integer> map = new HashMap<>();
        String[] full = split[0].split(",");
        for (int i = 0; i < full.length; i++) {
            String[] dan = full[i].split(":");
            list.add(dan[0]);
            map.put(dan[0],Integer.parseInt(dan[1]));
        }
        String[] used = split[1].split(",");
        for (int i = 0; i < used.length; i++) {
            String[] dan = used[i].split(":");
            int fullCount = map.get(dan[0]);
            if (fullCount > Integer.parseInt(dan[1])){
                map.put(dan[0],fullCount - Integer.parseInt(dan[1]));
            }else {
                map.remove(dan[0]);
                list.remove(dan[0]);
            }
        }
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < list.size(); i++) {
            String tem = list.get(i);
            int left = map.get(tem);
            sb.append(tem).append(":").append(left).append(",");
        }
        sb.deleteCharAt(sb.length()-1);
        System.out.println(sb);
    }
}
1
2
3
4
5
6
7
8
9
10
11
12
13
```

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

080 【字符串序列判定】

输入两个字符串 S 和 L，都只包含英文小写字母。S 长度<=100，L 长度<=500,000。判定 S 是否是 L 的有效字串。

判定规则：S 中的每个字符在 L 中都能找到（可以不连续），且 S 在 L 中字符的前后顺序与 S 中顺序要保持一致。（例如，S="ace"是 L="abcde"的一个子序列且有效字符是 a、c、e，而"aec"不是有效子序列，且有效字符只有 a、e）

输入描述:

输入两个字符串 S 和 L，都只包含英文小写字母。S 长度<=100，L 长度<=500,000。

先输入 S，再输入 L，每个字符串占一行。

输出描述:

S 串最后一个有效字符在 L 中的位置。（首位从 0 开始计算，无有效字符返回-1）

示例 1

输入

ace

babcde

输出

5

示例 2

输入

fgh

abcde

输出

-1

```
public class ZT80 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String target = sc.nextLine();
        String search = sc.nextLine();
        int idx = 0;
        for (int i = 0; i < target.length(); i++) {
            char ch = target.charAt(i);
            if (search.contains(String.valueOf(ch))){
                int i1 = search.indexOf(String.valueOf(ch));
                idx += i1;
                search = search.substring(i1+1);
            }else {
                System.out.println(-1);
                return;
            }
        }
        System.out.println(idx + target.length() - 1);
    }
}
```

————————————————————

081 【字符统计及重排】
082 【组成最大数】
083 【最大N个数与最小N个数的和】
084 【最大花费金额】
085【最大矩阵和】
086【最大括号深度】
087【最远足迹】
088【最长连续子序列】
089【最长元音子串的长度】
090【最长子字符串的长度】

081 【字符统计及重排】

给出一个仅包含字母的字符串，不包含空格，统计字符串中各个字母（区分大小写）出现的次数，并按照字母出现次数从大到小的顺序输出各个字母及其出现次数。如果次数相同，按照自然顺序进行排序，且小写字母在大写字母之前。

输入描述:

输入一行，为一个仅包含字母的字符串。

输出描述:

按照字母出现次数从大到小的顺序输出各个字母和字母次数，用英文分号分隔，注意末尾的分号；字母和次数间用英文冒号分隔。

示例 1

输入

xyxyXX

输出

x:2;y:2;X:2;

说明

每个字符出现的个数都是 2，故 x 排在 y 之前，而小写字母 x 在 X 之前

示例 2

输入

abababb

输出

b:4;a:3;

说明

b 的出现个数比 a 多，故 b 排在 a 之前

```
public class ZT81 {
    public static void main(String[] args) {
        List<Word> list = new ArrayList<>();
        Scanner sc = new Scanner(System.in);
        String input = sc.nextLine();
        //存
        Map<Character,Integer> map = new HashMap<>();
        int idx = 0;
        for (int i = 0; i < input.length(); i++) {
            char ch = input.charAt(i);
            Word word = new Word(1,ch);
            if (map.containsKey(ch)) {
                int index = map.get(ch);
                list.get(index).count++;
            }else {
                map.put(ch,idx++);
                list.add(word);
            }
        }
        //取
        Collections.sort(list);
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < list.size(); i++) {
            Word word = list.get(i);
            sb.append(word.ch).append(":").append(word.count).append(",");
        }
        sb.deleteCharAt(sb.length() -1).append(";");
```

```
        System.out.println(sb);

    }

    private static class Word implements Comparable<Word>{
        private int count;
        private char ch;

        public Word(int count, Character ch) {
            this.count = count;
            this.ch = ch;
        }

        @Override
        public int compareTo(Word wo) {
            char o1 = wo.ch;
            char o2 = this.ch;
            if (wo.count!= this.count){
                return wo.count - this.count;
            }
            if ((o1 >= 'a' && o2>= 'a') || (o1 <= 'Z' && o2<= 'Z')){//相同时正序排
                return o2 - o1;
            }else {//逆序排
                return o1 - o2;
            }
        }

        @Override
        public boolean equals(Object obj) {
            Word wo = (Word) obj;
            return this.ch == wo.ch;
        }
    }
}
1
2
3
4
5
6
7
8
9
10
```

55

56

57

58

59

60

61

082 【组成最大数】

小组中每位都有一张卡片，卡片上是 6 位内的正整数，将卡片连起来可以组成多种数字，计算组成的最大数字。

输入描述:

“,”号分割的多个正整数字符串，不需要考虑非数字异常情况，小组最多 25 个人

输出描述:

最大的数字字符串

示例 1

输入

22,221

输出

22221

示例 2

输入

4589,101,41425,9999，9

输出

9999458941425101

```
public class ZT82 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] split = sc.nextLine().split(",");
        //首位排序后，在对次位进行排序
        Map<Integer, List<String>> map = new HashMap<>();
        for (int i = 0; i < split.length; i++) {
            int ch = Integer.parseInt(String.valueOf(split[i].charAt(0)));
            List<String> def = map.getOrDefault(ch, new ArrayList<>());
            def.add(split[i]);
            map.put(ch,def);
        }
        //到过来输出
        StringBuilder sb = new StringBuilder();
        for (int i = 9; i >0 ; i--) {
            if (map.containsKey((Object) i )) {
                List<String> strings = map.get(i);
                sb.append(getChangeList(strings));
            }
        }
```

```
        System.out.println(sb);
    }
    //9,99,91,98,991,905,9
    //9,99,991,98,91,905
    //思路转为拿出一个数，比较后面有没有比他大的数，有则当前数放到最后，没有则直接拼接
    private static String getChangeList(List<String> list){
        LinkedList<String> linkList = new LinkedList<>(list);
        StringBuilder sb = new StringBuilder();
        while (linkList.size() > 0){
            String tem = linkList.get(0);
            boolean flag = true;
            if (tem.length() != 1){
                for (int i = 1; i < linkList.size(); i++) {
                    if (compareMax(tem,linkList.get(i))){//tem 不是当前最大值
                        linkList.remove(0);
                        linkList.addLast(tem);
                        flag = false;
                        break;
                    }
                }
            }//没有比当前值更大的，直接删掉当前值
            if (flag){
                sb.append(tem);
                linkList.remove(0);
            }
        }
        return sb.toString();
    }
    //98,99,901,902,998
    private static boolean compareMax(String idx0,String idx1){
        int idx1Length = idx0.length();
        int idx2Length = idx1.length();
        int length =Math.min(idx1Length,idx2Length);
        for (int i = 1; i < length; i++) {
            int i1 = Integer.parseInt(String.valueOf(idx0.charAt(i)));
            int i2 = Integer.parseInt(String.valueOf(idx1.charAt(i)));
            if (i1 != i2){
                return i2 - i1 > 0;
            }
        }
        return idx2Length == length;
    }
}
```

1

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

083 【最大 N 个数与最小 N 个数的和】

给定一个数组，编写一个函数来计算它的最大 N 个数与最小 N 个数的和。你需要对数组进行去重。

说明：

*数组中数字范围[0, 1000]

*最大 N 个数与最小 N 个数不能有重叠，如有重叠，输入非法返回-1

*输入非法返回-1

输入描述:

第一行输入 M， M 标识数组大小

第二行输入 M 个数，标识数组内容

第三行输入 N，N 表达需要计算的最大、最小 N 个数

输出描述:

输出最大 N 个数与最小 N 个数的和。

示例 1

输入

5

95 88 83 64 100

2

输出

342

说明

最大 2 个数[100,95],最小 2 个数[83,64], 输出为 342

示例 2

输入

5

3 2 3 4 2

2

输出

-1

说明

最大 2 个数[4,3],最小 2 个数[3,2], 有重叠输出为-1

```
public class ZT83 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int count = Integer.parseInt(sc.nextLine());
        String[] input = sc.nextLine().split(" ");
        int max = Integer.parseInt(sc.nextLine());
        Set<Integer> set = new HashSet<>();
        for (int i = 0; i < input.length; i++) {
            set.add(Integer.parseInt(input[i]));
        }
        List<Integer> list = new ArrayList<>(set);
        Collections.sort(list);
        //找到最大 N 个数的最小值 ma < mi
        //最小 N 个数的最大值 mi
        List<Integer> listMin = list.subList(0,max);
        List<Integer> listMax = list.subList(list.size() - max,list.size());
        if (listMin.get(max-1) > listMax.get(0)){
            System.out.println(-1);
        }else {
            int total = 0;
            for (int i = 0; i < max; i++) {
                total += listMin.get(i) + listMax.get(i);
            }
            System.out.println(total);
        }
    }
}
```

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

084 【最大花费金额】

双十一众多商品进行打折销售，小明想购买自己心仪的一些物品，但由于受购买资金限制，所以他决定从众多心仪商品中购买三件，而且想尽可能的花完资金，现在请你设计一个程序帮助小明计算尽可能花费的最大资金数额。

输入描述:

输入第一行为一维整型数组 M，数组长度小于 100，数组元素记录单个商品的价格，单个商品价格小于 1000。

输入第二行为购买资金的额度 R，R 小于 100000。

输出描述:

输出为满足上述条件的最大花费额度。

注意：如果不存在满足上述条件的商品，请返回-1。

示例 1

输入

23,26,36,27

78

输出

76

说明

金额 23、26 和 27 相加得到 76，而且最接近且小于输入金额 78

示例 2

输入

23,30,40

26

输出

-1

说明

因为输入的商品，无法组合出来满足三件之和小于 26.故返回-1

备注:

输入格式是正确的，无需考虑格式错误的情况。

```
public class ZT84 {
```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] split = sc.nextLine().split(",");
        int max = Integer.parseInt(sc.nextLine());
        int[] arr = new int[split.length];
        for (int i = 0; i < arr.length; i++) {
            arr[i] = Integer.parseInt(split[i]);
        }
        int total = -1;
        for (int i = 0; i < split.length; i++) {
            for (int j = i+1; j < split.length; j++) {
                for (int k = j+1; k < split.length; k++) {
                    int tem = arr[i] + arr[j] + arr[k];
                    if (tem <= max){
                        total = Math.max(tem,total);
                    }
                }
            }
        }
        System.out.println(total);
    }
}
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
```

23

085【最大矩阵和】

给定一个二维整数矩阵，要在这个矩阵中选出一个子矩阵，使得这个子矩阵内所有的数字和尽量大，我们把这个子矩阵称为和最大子矩阵，子矩阵的选取原则是原矩阵中一块相互连续的矩形区域。

输入描述:

输入的第一行包含 2 个整数 n, m(1 <= n, m <= 10)，表示一个 n 行 m 列的矩阵，下面有 n 行，每行有 m 个整数，同一行中，每 2 个数字之间有 1 个空格，最后一个数字后面没有空格，所有的数字的在[-1000, 1000]之间。

输出描述:

输出一行一个数字，表示选出的和最大子矩阵内所有的数字和。

示例 1

输入

3 4

-3 5 -1 5

2 4 -2 4

-1 3 -1 3

输出

20

说明

一个 3*4 的矩阵中，后面 3 列的子矩阵求和加起来等于 20，和最大。

```
/**
 * 最大矩阵和
 */
public class Main {
    public static Map<Node04,Integer> usedMap = new HashMap<Node04, Integer>();

    public static void main(String[] args) {
        Scanner sc =new Scanner(System.in);
        String[] nums = sc.nextLine().split(" ");
        int n1 = Integer.parseInt(nums[0]);//行
        int m1 = Integer.parseInt(nums[1]);//列
        int[][] arr = new int[n1][m1];
        for (int i = 0; i < n1; i++) {
            String[] input = sc.nextLine().split(" ");
            for (int j = 0; j < m1; j++) {
                arr[i][j] = Integer.parseInt(input[j]);
            }
        }
        System.out.println(calcMax(arr,0,n1,0,m1));
    }

    public static int calcMax(int[][] arr,int startX,int endX,int startY,int endY){
        Node04 node = new Node04(startX,endX,startY,endY);
```

```
    int max = 0;
    if (usedMap.containsKey(node)) {
        return usedMap.get(node);
    }else {
        //计算当前范围的node结果值
        //分成4个方向 上下左右各切一刀
        max = calcRes(arr,startX,endX,startY,endY);
        if (startX + 1 <= endX){
            int res1 = calcMax(arr, startX + 1, endX, startY, endY);
            int res2 = calcMax(arr, startX, endX-1, startY, endY);
            max = Math.max(max,Math.max(res1,res2));
        }
        if (startY+1 <= endY){
            int res3 = calcMax(arr, startX , endX, startY+1, endY);
            int res4 = calcMax(arr, startX, endX, startY, endY-1);
            max = Math.max(max,Math.max(res3,res4));
        }
        Integer put = usedMap.put(node, max);
    }
    return max;
}

public static int calcRes(int[][] arr,int x1,int x2,int y1,int y2){
    int total = 0;
    for (int i = x1; i < x2; i++) {
        for (int j = y1; j < y2; j++) {
            total += arr[i][j];
        }
    }
    return total;
}

public static class Node04{
    private int startX;
    private int endX;
    private int startY;
    private int endY;

    public Node04(int startX, int endX, int startY, int endY) {
        this.startX = startX;
        this.endX = endX;
        this.startY = startY;
        this.endY = endY;
    }
```

```
        @Override
        public boolean equals(Object obj) {
            Node04 node = (Node04) obj;
            return node.startX == this.startX && node.endX == this.endX &&
                node.startY == this.startY && node.endY == this.endY;
        }
    }
}
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
```

36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77

086【最大括号深度】

现有一字符串仅由 ‘(’，’)’，’{’，’}’，’[’，’]’六种括号组成。

若字符串满足以下条件之一，则为无效字符串：

①任一类型的左右括号数量不相等；

②存在未按正确顺序（先左后右）闭合的括号。

输出括号的最大嵌套深度，若字符串无效则输出 0。

0≤字符串长度≤100000

输入描述:

一个只包括 ‘(’，’)’，’{’，’}’，’[’，’]’的字符串

输出描述:

一个整数，最大的括号深度

示例 1

输入

[]

输出

1

说明

有效字符串，最大嵌套深度为 1

示例 2

输入

([]{()})

输出

3

说明

有效字符串，最大嵌套深度为 3

示例 3

输入

(]

输出

0

说明

无效字符串，有两种类型的左右括号数量不相等

示例 4

输入

([)]

输出

0

说明

无效字符串，存在未按正确顺序闭合的括号

示例 5

输入

)(

输出

0

说明

无效字符串，存在未按正确顺序闭合的括号

```
/**
 * 栈分割括号
 */
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String input = sc.nextLine();
        char[] chars = input.toCharArray();
        MyStack<Character> stack = new MyStack<Character>(input.length());
        Character pop;
        int totalDeep = 0;
        int maxDeep = 0;
        for (char ch: chars) {
            switch (ch){
                case '{':
                case '[':
                case '(':
                    totalDeep++;
                    stack.pull(ch);
                    break;
                case ']':
                    pop = stack.pop();
                    if (pop != null &&  pop == '['){
                        totalDeep--;
                        break;
                    }else {
                        System.out.println(0);
                        return;
                    }
                case '}':
                    pop = stack.pop();
                    if (pop!= null &&  pop == '{'){
                        totalDeep--;
                        break;
                    }else {
                        System.out.println(0);
                        return;
                    }
                case ')':
                    pop = stack.pop();
                    if (pop != null && pop == '('){
                        totalDeep--;
                        break;
```

```
                }else {
                    System.out.println(0);
                    return;
                }
            default:
                throw new IllegalStateException("Unexpected value: " + ch);
        }
        maxDeep = Math.max(maxDeep,totalDeep);
    }
    if (stack.isEmpty()) {
        System.out.println(maxDeep);
    }else {
        System.out.println(0);
    }



}
private static class MyStack<Item>{
    public Item[] arr;
    private int size;

    public MyStack(int cap) {//由用户来初始化长度
        arr = (Item[]) new Object[cap];
    }

    //弹出栈顶
    public Item pop(){
        if (isEmpty()) {
            return null;
        }
        Item item = arr[--size];
        arr[size] = null;
        return item;

    }
    //入栈 长度由初始化保证，不考虑扩容
    public void pull(Item item){
        arr[size++] = item;
    }

    public boolean isEmpty(){
        return size == 0;
    }
}
```

```
}
//([]{()})
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
```

87

88

89

087【最远足迹】

某探险队负责对地下洞穴进行探险。探险队成员在进行探险任务时，随身携带的记录器会不定期地记录自身的坐标，但在记录的间隙中也会记录其他数据。探索工作结束后，探险队需要获取到某成员在探险过程中相对于探险队总部的最远的足迹位置。

仪器记录坐标时，坐标的数据格式为(x,y)，如(1,2)、(100,200)，其中0<x<1000，0<y<1000。同时存在非法坐标，如(01,1)、(1,01)，(0,100)属于非法坐标。

设定探险队总部的坐标为(0,0)，某位置相对总部的距离为：xx+yy。

若两个座标的相对总部的距离相同，则第一次到达的坐标为最远的足迹。

若记录仪中的坐标都不合法，输出总部坐标（0,0）。

备注：不需要考虑双层括号嵌套的情况，比如sfsdfsd((1,2))。

输入描述:

字符串，表示记录仪中的数据。

如：ferga13fdsf3(100,200)f2r3rfasf(300,400)

输出描述:

字符串，表示最远足迹到达的坐标。

如： (300,400)

示例1

输入

ferg(3,10)a13fdsf3(3,4)f2r3rfasf(5,10)

输出

(5,10)

说明

记录仪中的合法坐标有3个： (3,10)， (3,4)， (5,10)，其中(5,10)是相距总部最远的坐标， 输出(5,10)。

示例2

输入

asfefaweawfaw(0,1)fe

输出

(0,0)

说明

记录仪中的坐标都不合法，输出总部坐标（0,0）

```
/**
 * 【最远足迹】
 */
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String input = sc.nextLine();
        List<String> list = new ArrayList<>();
        boolean flag = input.contains("(");
        while (flag){
            int i1 = input.indexOf("(");
```

```
            int i2 = input.indexOf(")");
            list.add(input.substring(i1+1,i2));
            input = input.substring(i2+1);
            flag = input.contains("(");
        }
        //分割数据
        int xi = 0 ;
        int yi = 0;
        int maxi = 0;
        for (int i = 0; i < list.size(); i++) {//0<x<1000, 0<y<1000
            //数据不能是开头
            String[] split = list.get(i).split(",");
            if (split[0].startsWith("0") || split[1].startsWith("0")){//数据忽略
                continue;
            }
            int x1 = Integer.parseInt(split[0]);
            int y1 = Integer.parseInt(split[1]);
            if (x1 <= 0 || x1 >=1000 || y1 <= 0  || y1 >= 1000){
                continue;
            }
            if (x1 * x1 + y1 * y1 > maxi){
                maxi = x1 * x1 + y1 * y1;
                xi = x1;
                yi = y1;
            }
        }
        System.out.println("("+ xi + ","  +yi +")");
    }
}
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

088【最长连续子序列】

有 N 个正整数组成的一个序列。给定整数 sum，求长度最长的连续子序列，使他们的和等于 sum，返回此子序列的长度，如果没有满足要求的序列，返回-1。

输入描述:

序列：1,2,3,4,2

sum：6

输出描述:

序列长度：3

输入描述:

序列：1,2,3,4,2

sum：6

输出描述:

序列长度：3

示例 1

输入

1,2,3,4,2

6

输出

3

说明

解释：1,2,3 和 4,2 两个序列均能满足要求，所以最长的连续序列为 1,2,3，因此结果为 3

示例 2

输入

1,2,3,4,2

20

输出

-1

说明

解释：没有满足要求的子序列，返回-1

备注:

输入序列仅由数字和英文逗号构成，数字之间采用英文逗号分隔；

序列长度：1 <= N <= 200；

输入序列不考虑异常情况，由题目保证输入序列满足要求。

```
/**
 * 【最长连续子序列】
 */
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] split = sc.nextLine().split(",");
        int[] arr = new int[split.length];
        int sum = Integer.parseInt(sc.nextLine());
        int total = 0;
        for (int i = 0; i < arr.length; i++) {
            arr[i] = Integer.parseInt(split[i]);
            total += arr[i];
        }
        if (total< sum){
            System.out.println(-1);
        }else if (total == sum){
            System.out.println(split.length);
        }else {
            //双指针
            int left = 0;
            int right = 0;
            int temp = 0;
            while (left < arr.length && right< arr.length){
                if (temp < sum){
                    temp += arr[right];
                    right++;
                }else if (temp != sum){
                    temp -= arr[left];
```

```
                    left++;
                }
                if (temp == sum) {
                    System.out.println(right - left);
                    left++;
                    right = left;
                    temp = 0;
                }
            }
        }
    }
}
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
```

33

34

35

36

37

38

39

40

41

089【最长元音子串的长度】

定义：当一个字符串只有元音字母（aeiouAEIOU）组成，称为元音字符串。

现给定一个字符串，请找出其中最长的元音字符子串，并返回其长度；如果找不到，则返回 0。

子串：字符串中任意个连续的字符组成的子序列称为该字符串的子串。

输入描述:

一个字符串，其长度范围：0 < length <= 65535。

字符串仅由字符 a-z 和 A-Z 组成。

输出描述:

一个整数，表示最长的元音字符子串的长度。

示例 1

输入

asdbuiodevauufgh

输出

3

说明

样例 1 解释：最长元音子串为 “uio” 或 “auu”，其长度都为 3，因此输出 3

```
/**
 * 89 最长元音子串的长度
 * 定义：当一个字符串只有元音字母（aeiouAEIOU）组成，称为元音字符串。
 * 现给定一个字符串，请找出其中最长的元音字符子串，并返回其长度；如果找不到，则返回 0。
 *
 * 子串：字符串中任意个连续的字符组成的子序列称为该字符串的子串。
 *
 * 输入描述:
 * 一个字符串，其长度范围：0 < length <= 65535。
 * 字符串仅由字符 a-z 和 A-Z 组成。
 *
 * 输出描述:
 * 一个整数，表示最长的元音字符子串的长度。
 *
 * 示例 1
 * 输入
 * asdbuiodevauufgh
 * 输出
```

```
 * 3
 * 说明
 * 样例1解释：最长元音子串为 “uio” 或 “auu”，其长度都为3，因此输出3
 */
public class Main {
    public static void main(String[] args) {
        char[] arr = {'a','e','i','o','u'};
        List<Character> list = new ArrayList<>();
        for (int i = 0; i < arr.length; i++) {
            list.add(arr[i]);
        }
        Scanner sc = new Scanner(System.in);
        String input = sc.nextLine();
        int idx = 0;
        int max = 0;
        int tem = 0;

        while (idx < input.length()){
            char ch = input.charAt(idx);
            if (list.contains(ch)) {
                tem++;
            }else {
                if (tem != 0) {
                    max = Math.max(max,tem);
                    tem = 0;
                }
            }
            idx++;
        }
        System.out.println(max);
    }
}
1
2
3
4
5
6
7
8
9
10
11
12
```

13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50

3303【最长子字符串的长度】

给你一个字符串 s，字符串 s 首尾相连成一个环形 ，请你在环中找出 ‘o’ 字符出现了偶数次最长子字符串的长度。

输入描述:

输入是一串小写字母组成的字符串

输出描述:

输出是一个整数

示例 1

输入

alolobo

输出

6

说明

最长子字符串之一是 “alolob”，它包含’ o’ 2 个。

示例 2

输入

looxdolx looxdolxlooxdolxlooxdolxlooxdolx

输出

7

说明

最长子字符串是 “oxdolxl”，由于是首尾连接在一起的，所以最后一个 ‘x’ 和开头的 ‘l’ 是连接在一起的，此字符串包含 2 个’ o’ 。

示例 3

输入

bcbcbc

输出

6

说明

这个示例中，字符串 “bcbcbc” 本身就是最长的，因为 ‘o’ 都出现了 0 次。

备注:

1 <= s.length <= 5 x 10^5

s 只包含小写英文字母。

```
/**
 * 【最长子字符串的长度】
 */
public class ZT90 {
    1
}
```

————————————————————

原文链接：https://blog.csdn.net/csfun1/article/details/124579505

091 【迷宫问题】[图-dfs]

092 【机器人走迷宫】[图-dfs]

093 【最长广播响应】(图-迪杰斯特拉算法)

094 【求满足条件的最长子串的长度】

095 【We Are A Team】

096 【招聘】

097 【乱序整数序列两数之和绝对值最小】

098 【电信号】

099 【任务最优调度】

100 【猜密码】

101 【分积木】

091 【迷宫问题】[图-dfs]

定义一个二维数组 N*M ，如 5 × 5 数组下所示：

```
int maze[5][5] = {
0, 1, 0, 0, 0,
0, 1, 1, 1, 0,
0, 0, 0, 0, 0,
0, 1, 1, 1, 0,
0, 0, 0, 1, 0,
};
```

它表示一个迷宫，其中的 1 表示墙壁，0 表示可以走的路，只能横着走或竖着走，不能斜着走，要求编程序找出从左上角到右下角的路线。入口点为[0,0],既第一格是可以走的路。

数据范围： 2≤n,m≤10 ， 输入的内容只包含 0≤val≤1

输入描述：

输入两个整数，分别表示二维数组的行数，列数。再输入相应的数组，其中的 1 表示墙壁，0 表示可以走的路。数据保证有唯一解,不考虑有多解的情况，即迷宫只有一条通道。

输出描述：

左上角到右下角的最短路径，格式如样例所示。

示例 1

输入：

```
5 5
0 1 0 0 0
0 1 1 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0
```

输出：

```
(0,0)
(1,0)
(2,0)
(2,1)
(2,2)
(2,3)
(2,4)
(3,4)
(4,4)
```

示例 2

输入：

```
5 5
0 1 0 0 0
```

0 1 0 1 0

0 0 0 0 1

0 1 1 1 0

0 0 0 0 0

输出：

(0,0)

(1,0)

(2,0)

(3,0)

(4,0)

(4,1)

(4,2)

(4,3)

(4,4)

说明：注意：不能斜着走！！

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Scanner;

/**
 * DFS 找最短路线
 int maze[5][5] = {
0,1,0,0,0
0,1,1,1,0
0,0,0,0,0
0,1,1,1,0
0,0,0,1,0
 };
 */
public class Main {
    private static int n1;
    private static int m1;
    private static int[][] arr;//邻接表
    private static boolean[][] mark;
    private static int endX1;
    private static int endY1;
    private static List<Point> list = new ArrayList<>();
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] sa = sc.nextLine().split(" ");
        n1 = Integer.parseInt(sa[0]);
```

```
    m1 = Integer.parseInt(sa[1]);
    endX1 = n1 -1;
    endY1 = m1 -1;
    arr = new int[n1][m1];
    mark = new boolean[n1][m1];
    for (int i = 0; i < n1; i++) {
        String[] split = sc.nextLine().split(" ");
        for (int j = 0; j < m1; j++) {
            arr[i][j] = Integer.parseInt(split[j]);
        }
    }
    dfs(0, 0, 0);
    Collections.reverse(list);
    for (int i = 0; i < list.size(); i++) {
        Point point = list.get(i);
        System.out.println("(" + point.x1 + "," +point.y1+ ")");
    }
}

//dfs 就是递归求最小值
private static boolean dfs(int x1,int y1,int step){
    if (x1 == endX1 && y1 == endY1){
        list.add(new Point(endX1,endY1));
        return true;
    }
    int[][] next = {{1,0},{0,1},{0,-1},{-1,0}};
    boolean flag = false;
    for (int i = 0; i < 4; i++) {
        int nextX = x1 + next[i][0];
        int nextY = y1 + next[i][1];
        if (nextX < 0 || nextX >= n1 || nextY<0 || nextY>= m1){
            continue;
        }
        if (arr[nextX][nextY] == 0 && !mark[nextX][nextY]){//下个点是 0 且未做标记
            mark[nextX][nextY] = true;
            if (dfs(nextX, nextY, step + 1)) {
                flag = true;
                list.add(new Point(x1,y1));
            }
            mark[nextX][nextY] = false;
        }
    }
    return flag;
}
```

```
    static class Point{
        int x1;
        int y1;

        public Point(int x1, int y1) {
            this.x1 = x1;
            this.y1 = y1;
        }
    }
}
```

79

80

81

82

092 【机器人走迷宫】[图-dfs]

1、 房间由 XY 的方格组成，例如下图为 64 的大小。每一个方格以坐标(x，y)描述。

2、 机器人固定从方格(0，0)出发，只能向东或者向北前进。出口固定为房间的最东北角，如下图的方格(5，3)。用例保证机器人可以从入口走到出口。

3、 房间有些方格是墙壁，如(4，1)，机器人不能经过那儿。

4、 有些地方是一旦到达就无法走到出口的，如标记为 B 的方格，称之为陷阱方格。

5、 有些地方是机器人无法到达的的，如标记为 A 的方格，称之为不可达方格，不可达方格不包括墙壁所在的位置。

6、 如下示例图中，陷阱方格有 2 个，不可达方格有 3 个。

7、 请为该机器人实现路径规划功能：给定房间大小、墙壁位置，请计算出陷阱方格与不可达方格分别有多少个。

```
/**
 * 机器人走迷宫
 * BFS 解决在不在一起的问题
 * DFS 解决路径问题
 */
public class ZT92 {
    private static int n1;
    private static int m1;
    private static int endX1;
    private static int endY1;
    private static int[][] arr;
    private static boolean[][] mark;
    private static int trip =0;

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] nms = sc.nextLine().split(" ");
        n1 = Integer.parseInt(nms[1]);
        m1 = Integer.parseInt(nms[0]);
        arr = new int[n1][m1];
        mark = new boolean[n1][m1];
        int x1 = n1 - 1;
        int y1 = 0;
        endX1 = 0;
        endY1 = m1 - 1;
        int wellCount = Integer.parseInt(sc.nextLine());
        for (int i = 0; i < wellCount; i++) {
            String[] wall = sc.nextLine().split(" ");
            int weX1 = n1 -1 - Integer.parseInt(wall[1]);
            int weY1 = Integer.parseInt(wall[0]);
```

```
            arr[weX1][weY1] = 1;
        }
        arr[x1][y1] = 2;//初始化点位
        dfs(x1, y1);
        //遍历所有坐标 结果是0的就是A
        int unReach = 0;
        for (int i = 0; i < n1; i++) {
            for (int j = 0; j < m1; j++) {
                if (arr[i][j] == 0){
                    unReach++;
                    System.out.println(i + ":" + j);
                }
            }
        }
        System.out.println(trip + " " + unReach);
    }

    private static boolean dfs(int x1,int y1){
        boolean flag = false;
        if (x1 == endX1 && y1 == endY1){
            return true;
        }
        int[][] next = {{-1,0},{0,1}};
        int nextX1 = 0;
        int nextY1 = 0;
        for (int i = 0; i < 2; i++) {
            nextX1 = x1 + next[i][0];
            nextY1 = y1 + next[i][1];
            if (nextX1 < 0 || nextY1 >=m1 ){
                continue;
            }
            if (arr[nextX1][nextY1] != 1 && !mark[nextX1][nextY1]){
                mark[nextX1][nextY1] = true;
                arr[nextX1][nextY1] = 2;//为了区别出未曾经过的坐标A
                boolean dfs = dfs(nextX1, nextY1);
                if (flag || dfs){
                    flag = true;
                }
                mark[nextX1][nextY1] = false;
            }
        }
        if (!flag){//坐标点B
            trip++;
            System.out.println(x1 + ":" + y1);
```

```
        }
        return flag;
    }
}
```

088 【最长广播响应】图-迪杰斯特拉算法D

某通信网络中有 N 个网络结点，用 1 到 N 进行标识。网络中的结点互联互通，且结点之间的消息传递有时延，相连结点的时延均为一个时间单位。

现给定网络结点的连接关系 link[i]={u,v}，其中 u 和 v 表示网络结点。

当指定一个结点向其他结点进行广播，所有被广播结点收到消息后都会在原路径上回复一条响应消息，请计算发送结点至少需要等待几个时间单位才能收到所有被广播结点的响应消息。

注：

1、N 的取值范围为[1,100]；

2、连接关系 link 的长度不超过 3000，且 1 <= u,v <= N；

3、网络中任意结点间均是可达的；

输入描述:

输入的第一行为两个正整数，分别表示网络结点的个数 N，以及时延列表的长度 I；

接下来的 I 行输入，表示结点间的连接关系列表；

最后一行的输入为一个正整数，表示指定的广播结点序号；

输出描述:

输出一个整数，表示发送结点接收到所有响应消息至少需要等待的时长。

示例 1

输入

5 7

2 1

1 4

2 4

2 3

3 4

3 5

4 5

2

输出

4

说明

2 到 5 的最小时延是 2 个时间单位，而 2 到其他结点的最小时延是 1 个时间单位，所以 2 收到所有结点的最大响应时间为 2*2=4。

```
public class ZT91Djtls {
    private static int[][] arr;//邻接表
    private static int[] dis;//表示从某个点到其他点位的最小距离
    private static boolean[] mark;
    private static int from;
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] nms = sc.nextLine().split(" ");
```

```
    int count = Integer.parseInt(nms[0]);
    int line = Integer.parseInt(nms[1]);
    arr = new int[count+1][count+1];
    for (int i = 0; i < line; i++) {
        String[] dis = sc.nextLine().split(" ");
        int x1 = Integer.parseInt(dis[0]);
        int y1 = Integer.parseInt(dis[1]);
        arr[x1][y1] = 1;
    }
    from = Integer.parseInt(sc.nextLine());//从哪个点开始广播
    dis = new int[count+1];
    for (int i = 0; i <= count; i++) {
        if (i == from){
            dis[i] = 0;
        }else {
            dis[i] = Integer.MAX_VALUE;
        }
    }
    mark = new boolean[count+1];
    calcTls(from,count);
}

private static void calcTls(int local,int count){
    //广播的点初始化距离
    mark[local] = true;
    for (int i = 1; i <= count; i++) {
        if (arr[local][i] != 0){
            dis[i] = arr[local][i];
        }
    }

    int used = 0;
    while (used< count){
        //计算下一个加入进来的坐标
        int tempLocal = 0;
        int distMin = Integer.MAX_VALUE;
        for (int i = 1; i <= count; i++) {
            //距离最近的点先加进来
            if (!mark[i] && dis[i] < distMin){
                distMin = dis[i];
                tempLocal = i;
            }
        }
        local = tempLocal;
```

```
            //逐个点的加入到图中
            mark[local] = true;
            for (int i = 1; i <= count; i++) {
                if (arr[local][i] != 0){
                    dis[i] = Math.min(dis[local] + arr[local][i],dis[i]);
                }
            }
            used++;
        }
        int maxDis = Integer.MIN_VALUE;
        for (int i = 1; i <= count; i++) {
            System.out.println(from + "到" + i + "的距离" + dis[i]);
            maxDis = Math.max(maxDis,dis[i]);
        }
        System.out.println(maxDis * 2);//时间要花双倍
    }
}
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
```

094 【求满足条件的最长子串的长度】

给定一个字符串，只包含字母和数字，按要求找出字符串中的最长（连续）子串的长度，字符串本身是其最长的子串，子串要求：

1、 只包含 1 个字母(a~z, A~Z)，其余必须是数字；

2、 字母可以在子串中的任意位置；

如果找不到满足要求的子串，如全是字母或全是数字，则返回-1。

输入描述：字符串(只包含字母和数字)

输出描述：子串的长度

示例 1

输入

abC124ACb

输出

4

说明

满足条件的最长子串是 C124 或者 124A，长度都是 4

示例 2

输入

a5

输出

2

说明

字符串自身就是满足条件的子串，长度为 2

示例 3

输入

aBB9

输出

2

说明

满足条件的子串为 B9，长度为 2

示例 4

输入

abcdef

输出

-1

说明

没有满足要求的子串，返回-1

```
public class ZT94 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String input = sc.nextLine();

        //双指针滑动
        int left = 0;
```

```
        int right = 1;
        int maxLe = -1;
        while (left < input.length() && right < input.length()){
            right++;
            String str = input.substring(left,right);//左取右不取
            if (checkStr(str)){
                maxLe = Math.max(str.length(),maxLe);
            }else {
                left++;
            }
        }
        System.out.println(maxLe);
    }

    private static boolean checkStr(String str){
        char[] chars = str.toCharArray();
        int wordCount = 0;
        int numCount = 0;
        for (int i = 0; i < chars.length; i++) {
            if ((chars[i] >= 'a' && chars[i] <= 'z') || (chars[i] >= 'A' && chars[i] <= 'Z')){
                if (++wordCount >= 2){
                    return false;
                }
            }else {
                numCount++;
            }
        }
        return numCount > 0 && wordCount > 0;
    }
}
```

【[illegible]】

总共有 n 个人在机房，每个人有一个标号（1 <= 标号 <=n），他们分成了多个团队，需要你根据收到的 m 条消息判定指定的两个人是否在一个团队中，具体的：

1、消息构成为：a b c，整数 a、b 分别代表了两个人的标号，整数 c 代表指令。

2、c0 代表 a 和 b 在一个团队内。

3、c1 代表需要判定 a 和 b 的关系，如果 a 和 b 是一个团队，输出一行“we are a team”，如果不是，输出一行“we are not a team”。

4、c 为其它值，或当前行 a 或 b 超出 1~n 的范围，输出“da pian zi”。

输入描述:

1、第一行包含两个整数 n, m(1 <= n, m <= 100000)，分别表示有 n 个人和 m 条消息。

2、随后的 m 行，每行一条消息，消息格式为:a b c (1 <= a, b <= n, 0 <= c <= 1)。

输出描述:

1、c==1 时，根据 a 和 b 是否在一个团队中输出一行字符串，在一个团队中输出“we are a team”，不在一个团队中输出“we are not a team”。

2、c 为其他值，或当前行 a 或 b 的标号小于 1 或者大于 n 时，输出字符串“da pian zi”。

3、如果第一行 n 和 m 的值超出约定的范围时，输出字符串"NULL"。

示例 1

输入

5 6

1 2 0

1 2 1

1 5 0

2 3 1

2 5 1

1 3 2

输出

we are a team

we are not a team

we are a team

da pian zi

说明

第 2 行定义了 1 和 2 是一个团队

第 3 行要求进行判定，输出"we are a team"

第 4 行定义了 1 和 5 是一个团队，自然 2 和 5 也是一个团队

第 5 行要求进行判定，输出"we are not a team"

第 6 行要求进行判定，输出"we are a team"

第 7 行 c 为其它值，输出"da pian zi"

```
public class ZT95 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] nms = sc.nextLine().split(" ");
        int nums = Integer.parseInt(nms[0]);
        int mes = Integer.parseInt(nms[1]);
        List<Set<Integer>> lists = new ArrayList<>();
        List<String> waitForVerify = new ArrayList<>();
        for (int i = 0; i < mes; i++) {
            String real = sc.nextLine();
            String[] rels = real.split(" ");
            int i1 = Integer.parseInt(rels[0]);
            int i2 = Integer.parseInt(rels[1]);
            int i3 = Integer.parseInt(rels[2]);
            if (i3 != 0){
                waitForVerify.add(real);
            }else {
                boolean flag = false;
                for (int j = 0; j < lists.size(); j++) {
                    Set<Integer> set = lists.get(j);
```

```
                    if (set.contains(i1) || set.contains(i2)){
                        set.add(i1);
                        set.add(i2);
                        flag = true;
                        break;
                    }
                }
                if (!flag){
                    Set<Integer> set = new HashSet<>();
                    set.add(i1);
                    set.add(i2);
                    lists.add(set);
                }
            }
        }
        //判断非0的关系
        for (int i = 0; i < waitForVerify.size(); i++) {
            String[] reli = waitForVerify.get(i).split(" ");
            int i1 = Integer.parseInt(reli[0]);
            int i2 = Integer.parseInt(reli[1]);
            int i3 = Integer.parseInt(reli[2]);
            if (i3 != 1){
                System.out.println("da pian zi");
                continue;
            }
            for (int j = 0; j < lists.size(); j++) {
                Set<Integer> set = lists.get(j);
                if (set.contains(i1) && set.contains(i2)){
                    System.out.println("we are a team");
                }else {
                    System.out.println("we are not a team");
                }
            }
        }
    }
}
```

088 【难题】

某公司组织一场公开招聘活动，假设由于人数和场地的限制，每人每次面试的时长不等，并已经安排给定，用(S1,E1)、(S2,E2)、(Sj,Ej)…(Si < Ei，均为非负整数)表示每场面试的开始和结束时间。面试采用一对一的方式，即一名面试官同时只能面试一名应试者，一名面试官完成一次面试后可以立即进行下一场面试，且每个面试官的面试人次不超过 m。

为了支撑招聘活动高效顺利进行，请你计算至少需要多少名面试官。

输入描述:

输入的第一行为面试官的最多面试人次 m，第二行为当天总的面试场次 n，接下来的 n 行为每场面试的起始时间和结束时间，起始时间和结束时间用空格分隔。

其中，1 <= n, m <= 500

输出描述:

输出一个整数，表示至少需要的面试官数量。

示例 1

输入

2

5

1 2

2 3

3 4

4 5

5 6

输出

3

说明

总共有 5 场面试，且面试时间都不重叠，但每个面试官最多只能面试 2 人次，所以需要 3 名面试官。

示例 2

输入

3

3

1 2

2 3

3 4

输出

1

说明

总共有 3 场面试，面试时间都不重叠，每个面试官最多能面试 3 人次，所以只需要 1 名面试官。

示例 3

输入

3

3

8 35

5 10

1 3

输出

2

说明

总共有 3 场面试，[5,10]和[8,35]有重叠，所以至少需要 2 名面试官。

097 【乱序整数序列两数之和绝对值最小】

给定一个随机的整数（可能存在正整数和负整数）数组 nums ，请你在该数组中找出两个数，其和的绝对值(|nums[x]+nums[y]|)为最小值，并返回这个两个数（按从小到大返回）以及绝对值。

每种输入只会对应一个答案。但是，数组中同一个元素不能使用两遍。

输入描述:

一个通过空格分割的有序整数序列字符串，最多 1000 个整数，且整数数值范围是 [-65535, 65535]。

输出描述:

两数之和绝对值最小值

示例 1

输入

-1 -3 7 5 11 15

输出

-3 5 2

说明

因为 |nums[0] + nums[2]| = |-3 + 5| = 2 最小，所以返回 -3 5 2

```
public class ZT97 {
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        String[] nums = sc.nextLine().split(" ");
        int[] arr = new int[nums.length];
        for (int i = 0; i < nums.length; i++) {
            arr[i] = Integer.parseInt(nums[i]);
        }
        int res = Integer.MAX_VALUE;
        int l1 = Integer.MIN_VALUE;
        int l2 = Integer.MIN_VALUE;
        for (int i = 0; i < nums.length; i++) {
            for (int j = i+1; j < nums.length; j++) {
                int temp = Math.abs(arr[i] + arr[j]);
                if (temp < res){
                    l1 = arr[i];
                    l2 = arr[j];
                    res = temp;
                }
            }
        }
        if (l1>l2){
            int temp = l2;
            l2 = l1;
            l1 = temp;
        }
        System.out.println(l1 +" " + l2 + " " + res);
    }
}
1
2
3
4
```

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

【电信号】方波

输入一串方波信号，求取最长的完全连续交替方波信号，并将其输出，如果有相同长度的交替方波信号，输出任一即可，方波信号高位用 1 标识，低位用 0 标识，如图：

说明：

1） 一个完整的信号一定以 0 开始然后以 0 结尾，即 010 是一个完整信号，但 101，1010，0101 不是

2）输入的一串方波信号是由一个或多个完整信号组成

3） 两个相邻信号之间可能有 0 个或多个低位，如 0110010，011000010

4） 同一个信号中可以有连续的高位，如 01110101011110001010，前 14 位是一个具有连续高位的信号

5） 完全连续交替方波是指 10 交替，如 01010 是完全连续交替方波，0110 不是

输入描述:

输入信号字符串（长度>=3 且<=1024）：

00101010101100001010000010

注：输入总是合法的，不用考虑异常情况

输出描述:

输出最长的完全连续交替方波信号串：

01010

若不存在完全连续交替方波信号串，输出 -1

示例 1

输入

00101010101100001010010

输出

01010

备注:

输入信号串中有三个信号：0 010101010110(第一个信号段) 00 01010(第二个信号段) 010(第三个信号段)

第一个信号虽然有交替的方波信号段，但出现了 11 部分的连续高位，不算完全连续交替方波，在剩下的连续方波信号串中 01010 最长

```
public class ZT98 {
    private static int maxLength = Integer.MIN_VALUE;
    private static String res;
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String input = sc.nextLine();
        if (input.contains("00")) {//包含 2 各及以上的信号波
            String[] split = input.split("00");
            //0 和尾要在后面和前面各添加一个 0
            //其他的前后均要补 0
            for (int i = 0; i < split.length; i++) {
                String temp = split[i];
                if (i == 0){
                    check(temp + "0");
                }else if (i == split.length -1){
                    check("0"+temp);
                }else {
                    check("0" + temp + "0");
                }
            }
        }else {//单信号波 01010
            check(input);
        }
        if (maxLength == Integer.MIN_VALUE){
            System.out.println(-1);
        }else {
            System.out.println(res);
        }
    }
    private static void check(String str){//首位一定是以 0 开头的 00 /01
        char[] chars = str.toCharArray();
        boolean start = false;
        int startIdx = 0;
        int endIdx = -1;
```

```
        char pre = '1';
        for (int i = 0; i < chars.length; i++) {
            if (!start && i+1 < chars.length && chars[i] == '0' && chars[i+1] == '1'){
                start = true;
                startIdx = i;
                pre = '0';
                continue;
            }
            if (pre != chars[i]){
                pre = chars[i];
            }else if (pre =='1' && chars[i] == '1'){
                return;
            } else {
                endIdx = i;
                break;
            }
        }
        if (endIdx == -1){
            if (str.length() - startIdx > maxLength){
                maxLength = str.length() - startIdx;
                res = str.substring(startIdx);
            }
        }else {
            if (endIdx - startIdx > maxLength){
                maxLength = endIdx - startIdx;
                res = str.substring(startIdx,endIdx);
            }
        }
    }
}
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

59

60

61

62

63

64

099 【任务最优调度】

给定一个正整型数组表示待系统执行的任务列表，数组的每一个元素代表一个任务，元素的值表示该任务的类型。请计算执行完所有任务所需的最短时间。任务执行规则如下：

1、任务可以按任意顺序执行，且每个任务执行耗时间均为 1 个时间单位。

2、两个同类型的任务之间必须有长度为 N 个单位的冷却时间，比如：N 为 2 时，在时间 K 执行了类型 3 的任务，那么 K+1 和 K+2 两个时间不能执行类型 3 任务。

3、系统在任何一个单位时间内都可以执行一个任务，或者等待状态。

说明：数组最大长度为 1000,数组最大值 1000.

输入描述:

第一行记录一个用半角逗号分隔的数组，数组长度不超过 1000，数组元素的值不超过 1000

第二行记录任务冷却时间，N 为正整数，N<=100。

输出描述:

输出为执行完所有任务所需的最短时间。

示例 1

输入

2,2,2,3

2

输出

7

说明

时间 1：执行类型 2 任务。

时间 2：执行类型 3 的任务（因为冷却时间为 2，所以时间 2 不能执行类型 2 的任务）。

时间 3：系统等待（仍然在类型 2 的冷却时间）。

时间 4：执行类型 2 任务。

时间 5：系统等待。

时间 6：系统等待。

时间 7：执行类型 2 任务。

因此总共耗时 7。

100 【猜密码】

小杨申请了一个保密柜，但是他忘记了密码。只记得密码都是数字，而且所有数字都是不重复的。请你根据他记住的数字范围和密码的最小数字数量，帮他算下有哪些可能的组合，规则如下：

1、输出的组合都是从可选的数字范围中选取的，且不能重复；

2、输出的密码数字要按照从小到大的顺序排列，密码组合需要按照字母顺序，从小到大的顺序排序。

3、输出的每一个组合的数字的数量要大于等于密码最小数字数量；

4、如果可能的组合为空，则返回“None”

输入描述:

1、输入的第一行是可能的密码数字列表，数字间以半角逗号分隔

2、输入的第二行是密码最小数字数量

输出描述:

可能的密码组合，每种组合显示成一行，每个组合内部的数字以半角逗号分隔，从小到大的顺序排列。

输出的组合间需要按照字典序排序。

比如：

2,3,4 放到 2,4 的前面

示例 1

输入

2,3,4

2

输出

2,3

2,3,4

2,4

3,4

说明

最小密码数量是两个，可能有三种组合：

2,3

2,4

3,4

三个密码有一种：

2,3,4

示例 2

输入

2,0

1

输出

0

0,2

2

说明

可能的密码组合，一个的有两种 ：

0

2

两个的有一个：

0,2

备注:

字典序是指按照单词出现在字典的顺序进行排序的方法，比如：

a 排在 b 前

a 排在 ab 前

ab 排在 ac 前

ac 排在 aca 前

在这里插入代码片

1

101 【分积木】

Solo 和 koko 是两兄弟，妈妈给了他们一大堆积木，每块积木上都有自己的重量。现在他们想要将这些积木分成两堆。哥哥 Solo 负责分配，弟弟 koko 要求两个人获得的积木总重量“相等”（根据 Koko 的逻辑），个数可以不同，不然就会哭，但 koko 只会先将两个数转成二进制再进行加法，而且总会忘记进位（每个进位都忘记）。如当 25（11101）加 11（1011）时，koko 得到的计算结果是 18（10010）：

11001

+01011--------

10010

Solo 想要尽可能使自己得到的积木总重量最大，且不让 koko 哭。

输入描述:

3

3 5 6

第一行是一个整数 N(2≤N≤100)，表示有多少块积木；第二行为空格分开的 N 个整数 Ci(1≤Ci≤106)，表示第 i 块积木的重量。

输出描述:

11

让 koko 不哭，输出 Solo 所能获得积木的最大总重量；否则输出“NO”。

示例 1

输入

3

3 5 6

输出

11

备注:

如果能让 koko 不哭，输出 Solo 所能获得的积木的总重量，否则输出-1。

该样例输出为 11。

解释：Solo 能获得重量为 5 和 6 的两块积木，5 转成二级制为 101，6 转成二进制位 110，按照 koko 的计算方法（忘记进位），结果为 11(二进制)。Koko 获得重量为 3 的积木，转成二进制位 11(二进制)。Solo 和 koko 得到的积木的重量都是 11(二进制)。因此 Solo 可以获得的积木的总重量是 5+6=11（十进制）。

————————————————————

原文链接：https://blog.csdn.net/csfun1/article/details/124643796

题目详情

标题：跳格子游戏 | 时间限制：1 秒 | 内存限制：262144K | 语言限制：不限

地上共有 N 个格子，你需要跳完地上所有的格子，但是格子间是有强依赖关系的，跳完前一个格子后，后续的格子才会被开启，格子间的依赖关系由多组 steps 数组给出，steps[0]表示前一个格子,steps[1]表示 steps[0]可以开启的格子:

比如[0,1]表示从跳完第 0 个格子以后第 1 个格子就开启了，比如[2,1]，[2,3]表示跳完第 2 个格子后第 1 个格子和第 3 个格子就被开启了

请你计算是否能由给出的 steps 数组跳完所有的格子,如果可以输出 yes，否则输出 no

说明：

1.你可以从一个格子跳到任意一个开启的格子

2.没有前置依赖条件的格子默认就是开启的

3.如果总数是 N，则所有的格子编号为[0,1,2,3…N-1]连续的数组

输入描述:

输入一个整数 N 表示总共有多少个格子，接着输入多组二维数组 steps 表示所有格子之间的依赖关系

输出描述:

如果能按照 steps 给定的依赖顺序跳完所有的格子输出 yes

否则输出 no

示例 1

输入

3

0 1

0 2

输出

yes

说明

总共有三个格子[0,1,2]，跳完 0 个格子后第 1 个格子就开启了，跳到第 0 个格子后第 2 个格子也被开启了，按照 0->1->2 或者 0->2->1 的顺序都可以跳完所有的格子

示例 2

输入

2

1 0

0 1

输出

no

说明

总共有 2 个格子，第 1 个格子可以开启第 0 格子，但是第 1 个格子又需要第 0 个格子才能开启，相互依赖，因此无法完成

示例 3

输入

6

0 1

0 2

0 3

0 4

0 5

输出

yes

说明

总共有 6 个格子，第 0 个格子可以开启第 1,2,3,4,5 个格子，所以跳完第 0 个格子之后其他格子都被开启了，之后按任何顺序可以跳完剩余的格子

示例 4

输入

5

4 3

0 4

2 1

3 2

输出

yes

说明

跳完第 0 个格子可以开启格子 4，跳完格子 4 可以开启格子 3，跳完格子 3 可以开启格子 2，跳完格子 2 可以开启格子 1，按照 0->4->3->2->1 这样就跳完所有的格子

示例 5

输入

4

1 2

1 0

输出

yes

说明

总共 4 个格子[0,1,2,3]，格子 1 和格子 3 没有前置条件所以默认开启，格子 1 可以开启格子 0 和格子 2，所以跳到格子 1 之后就可以开启所有的格子，因此可以跳完所有格子

解题思路

很明显这是一道拓扑排序题目，不同的节点之间存在先后执行依赖关系，只有前一个节点完成后一个节点才能完成。比如以下面例子

6

1 2

1 3

1 4

0 2

0 3

3 1

为例子，有拓扑图如下所示：

edges 是一个长度 N 为的 List 集合，以 steps 格子的右边数据（即开启的格子）为索引，以依赖关系的左边数据作为 list 的相邻顶点集合。节点 5 是一个单独节点。

可以使用深度搜索或者广度搜索来遍历图中的每个节点以及其相邻节点，如果判断图中存在环，那么一定有两个格子相互依赖，也就无法跳完所有的格子。以深度搜索 dfs 为例说明，如何判别图中是否存在环呢？针对某个节点来说，遍历其有三种可能：

这个节点还未被遍历过，这种情况会将这个节点加入 dfs 里面递归遍历，直到某节点再无相邻节点就结束。图中 0 节点还没被遍历过，将节点 0 加入到 dfs 方法里面一直遍历到节点 4，节点 4 再无相邻节点，符合条件。

这个节点已经被遍历过了，但是在遍历其相邻节点时又遍历了一次这个节点。比如节点 0 进入 dfs 深度遍历，先是遍历 0 的相邻节点 3，再遍历相邻节点 1，接下来再次遍历节点 1 的相邻节点时是节点 3 而不是节点 4，此时发现节点 3 已经被遍历过了，那么图中一定存在环，跳格子失败。

这个节点被遍历过了而且相邻节点也被深度遍历过了，说明图中无环（单独一个节点必然没有环），可以跳完所有的格子。
想要对一个节点标记 3 中遍历状态，那么使用 boolean 数组是欠妥的，可以使用 int 数组，用整数 0，1，2 表示一个节点的不同的遍历状态。

Java 代码

```
public class Main {
    private static final int NOT_VISITED = 0;
    private static final int VISITED = 1;
    private static final int VISIT_FINISHED = 2;
    static int[] visitStatus;                          //节点遍历状态
    static List<List<Integer>> edges;                    //记录每个节点的相邻节点
    static boolean stepAllGrids;                       //是否能跳完所有格子的结果判断

    public static boolean step(int N,List<List<Integer>> edges) {
        visitStatus = new int[N];     //节点默认为 0 ~ N - 1

        for(int i = 0; i < N && stepAllGrids; i++) {   //总是从节点 0 开始进入
            if(visitStatus[i] == NOT_VISITED) {
```

```
            dfs(i);
        }
    }

    return stepAllGrids;
}

public static void dfs(int node) {
    visitStatus[node] = VISITED;

    for(int neighbor : edges.get(node)) {        //遍历相邻节点
        if(visitStatus[neighbor] == NOT_VISITED) {
            dfs(neighbor);

            if(!stepAllGrids) {  //如果退出循环时检测到环，直接退出
                return;
            }
        }else if(visitStatus[neighbor] == VISITED){   //再一次遍历到此节点，说明存在环，无法完成拓扑排序
            stepAllGrids = false;
            return;
        }
    }

    visitStatus[node] = VISIT_FINISHED;
}


public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    while(sc.hasNextLine()) {
        int N = sc.nextInt();
        if(N < 1 || N >= 500) {
            System.out.println("error:输入的格子个数应该大于等于 1 且小于 500");
            return;
        }
        edges = new ArrayList<>();
        for(int i = 0; i < N; i++) {     //默认节点在 0 ~ N - 1 之间
            edges.add(new ArrayList<>());
        }

        stepAllGrids = true;

        while(sc.hasNext()) {
```

```
                int a = sc.nextInt();
                if(a == -1) {             //题目缺失输入退出条件，不妨设置输入-1 就退出输入
                    break;
                }
                String line = sc.nextLine();
                String[] strs = line.split(" ");
                if(strs.length != 2) {
                    System.out.println("error:输入的格子数组列数不为 2");
                    return;
                }
                if(a < 0 || a >= N) {
                    System.out.println("error:输入的左侧格子号码应该大于等于 0 且小于" + N);
                    return;
                }
                int b = Integer.parseInt(strs[1]);
                if(b < 0 || b >= N) {
                    System.out.println("error:输入的右侧格子号码应该大于等于 0 且小于" + N);
                    return;
                }
                edges.get(b).add(a);
            }
            if(step(N, edges)) {
                System.out.println("yes");
            }else {
                System.out.println("no");
            }
        }
    }
}
```

————————————————————

原文链接：https://blog.csdn.net/JackSnack/article/details/124554285