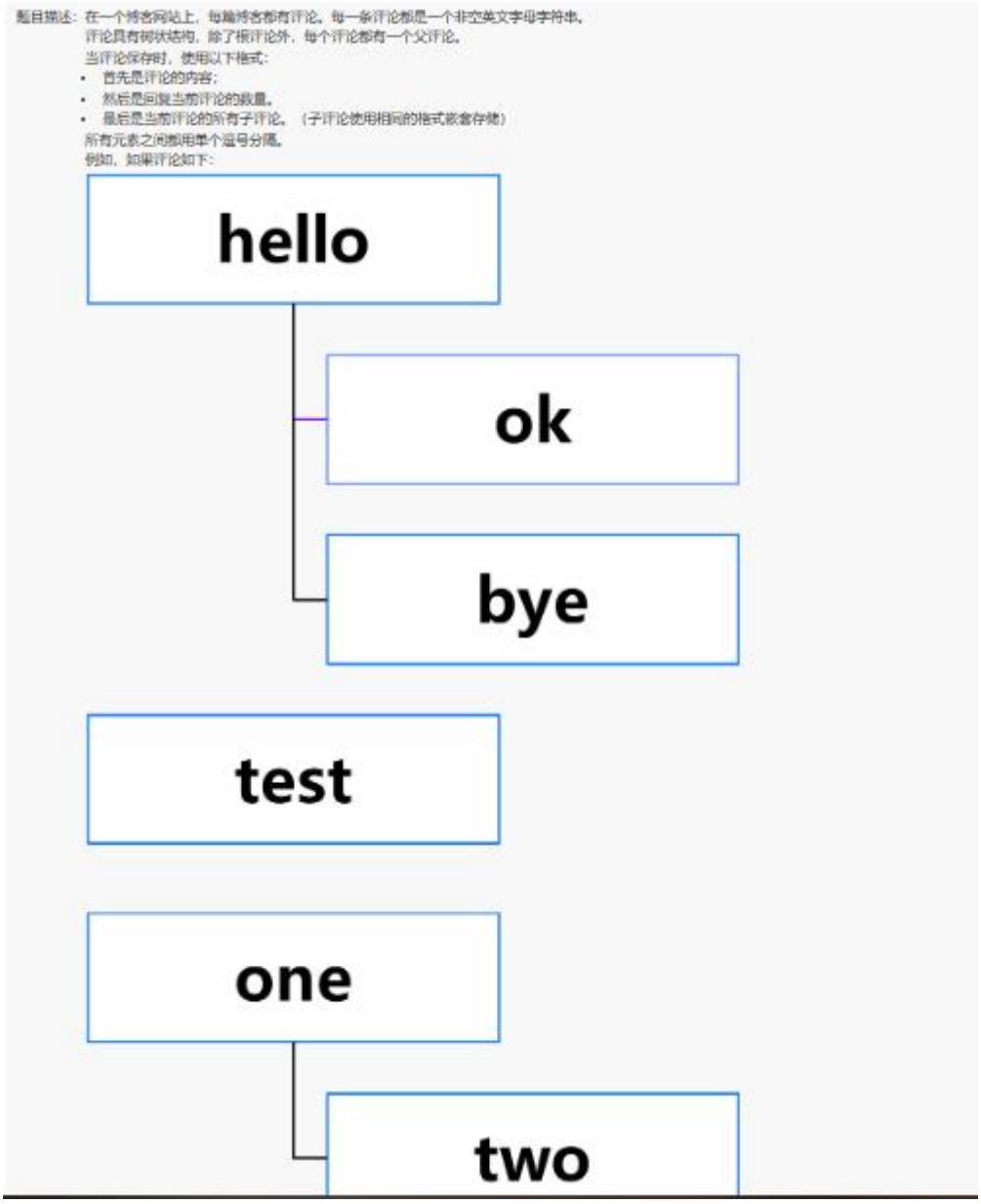
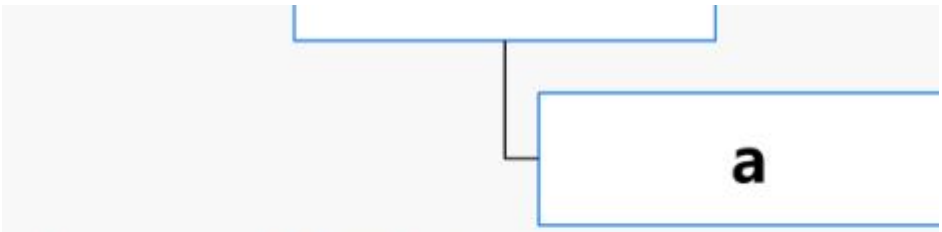
评论转换输出


题目描述：在一个博客网站上，每篇博客都有评论。每一条评论都是一个非空英文字母字符串。
评论具有树状结构，除了根评论外，每个评论都有一个父评论。
当评论保存时，使用以下格式：
首先是评论的内容；
然后是回复当前评论的数量。
最后是当前评论的所有子评论。（子评论使用相同的格式嵌套存储）
所有元素之间都用单个逗号分隔。
例如，如果评论如下：
hello
ok
bye
test
one
two



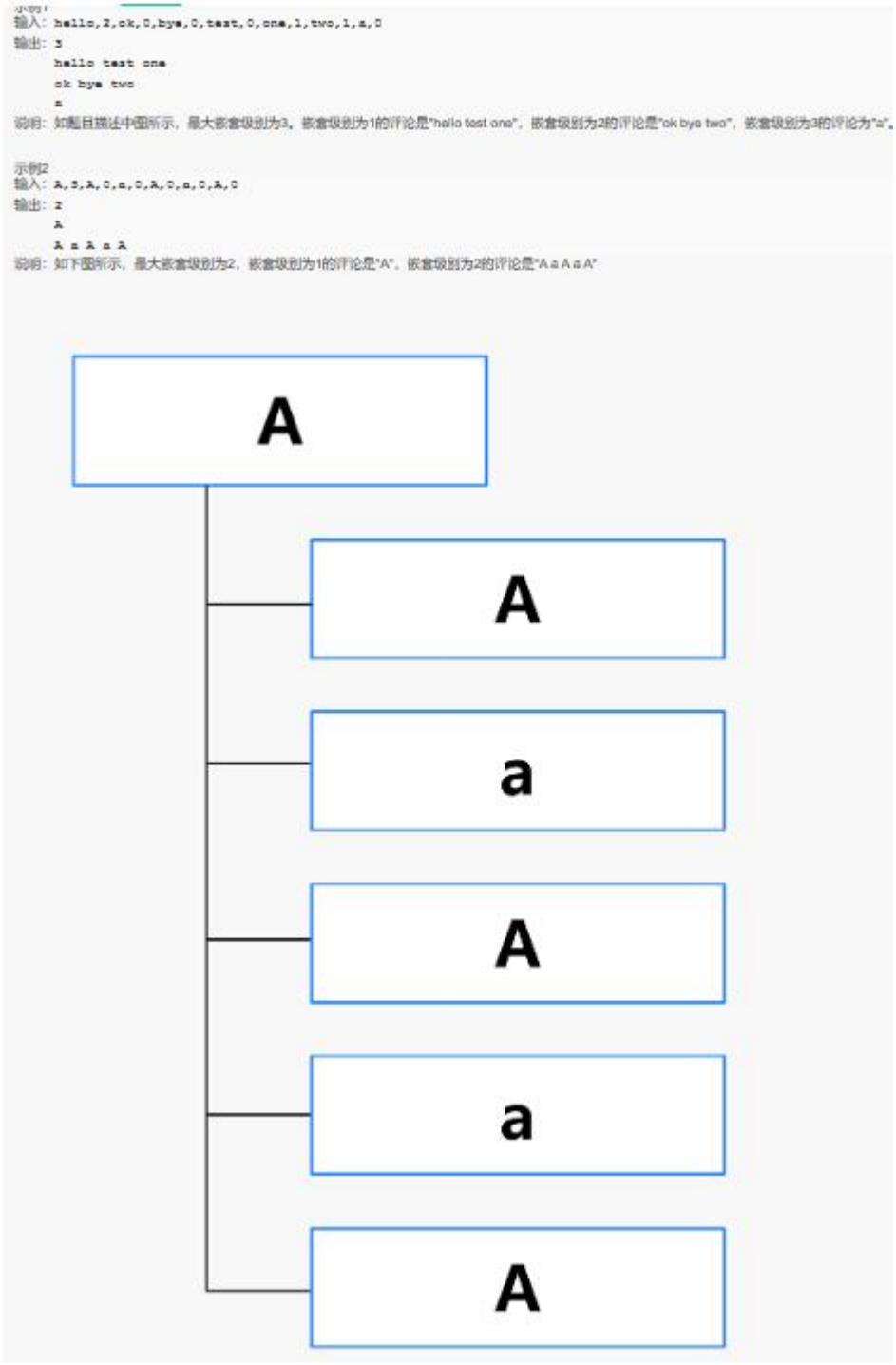
第一条评论是"hello,2,ok,0,bye,0"，第二条评论是"test,0"，第三条评论是"one,1,two,1,a,0"。
所有评论被保存成"hello,2,ok,0,bye,0,test,0,one,1,two,1,a,0"。

对于上述格式的评论，请以另外一种格式打印：
首先打印评论嵌套的最大深度。
然后是打印n行，第i（1<=i<=n）行对应于嵌套级别为i的评论（根评论的嵌套级别为1）。
对于第i行，嵌套级别为i的评论按照它们出现的顺序打印，用空格分隔开。

描述：一行评论，由英文字母、数字和英文逗号组成。
保证每个评论都是由英文字符组成的非空字符串。
每个评论的数量都是整数（至少由一个数字组成）。
整个字符串的长度不超过10^6。
给定的评论结构保证是合法的。

描述：按照给定的格式打印评论。对于每一级嵌套，评论应该按照输入中的顺序打印。

说明：




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

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

```
        boolean isNum = false;
        MyTree tree = new MyTree();
        List<MyTree> allTree = new LinkedList<MyTree>();
        tree.setChilds(allTree);
        TempInt temp = new TempInt();
        initTree(tree, temp, all);
        System.out.println(temp.getMaxDeep());
        Queue<MyTree> q1 = new LinkedList<>();
        Queue<MyTree> q2 = new LinkedList<>();
        for (MyTree t : tree.getChilds()) {
            q1.add(t);
        }
        while (!q1.isEmpty() || !q2.isEmpty()) {
            if (!q1.isEmpty()) {
                while (!q1.isEmpty()) {
                    MyTree outTree = q1.poll();
                    System.out.print(outTree.getContent()+" ");
                    for (MyTree t : outTree.getChilds()) {
                        q2.add(t);
                    }
                }
                System.out.print("\n");
            } else {
                while (!q2.isEmpty()) {
                    MyTree outTree = q2.poll();
                    System.out.print(outTree.getContent()+" ");
                    for (MyTree t : outTree.getChilds()) {
                        q1.add(t);
                    }
                }
                System.out.print("\n");
            }
        }
    }
    public static void initTree(MyTree tree, TempInt temp, String[] all) {
        List<MyTree> childs = tree.getChilds();
        while                    (tree.getDeep()!=0&&tree.getNum()                    >
childs.size()||(tree.getDeep()==0&&temp.getIndex()<all.length)) {
            MyTree t = new MyTree();
            t.setContent(all[temp.getIndex()]);
            t.setDeep(tree.getDeep() + 1);
            t.setNum(Integer.parseInt(all[temp.getIndex() + 1]));
            t.setChilds(new LinkedList<MyTree>());
            temp.setIndex(temp.getIndex() + 2);
```

```
                temp.setMaxDeep(Math.max(temp.getMaxDeep(), t.getDeep()));
                if (t.getNum() > 0) {
                    initTree(t, temp, all);
                }
                childs.add(t);
            }
            tree.setChilds(childs);
        }
    }
    class TempInt {
        private int index;
        private int maxDeep;
        public void setIndex(int n) {
            this.index = n;
        }
        public int getIndex() {
            return index;
        }
        public void setMaxDeep(int n) {
            this.maxDeep = n;
        }
        public int getMaxDeep() {
            return maxDeep;
        }
    }
    class MyTree {
        private int deep;
        private String content;
        private List<MyTree> childs;
        private int num;
        public void setChilds(List<MyTree> tree) {
            this.childs = tree;
        }
        public List<MyTree> getChilds() {
            return this.childs;
        }
        public void setNum(int n) {
            this.num = n;
        }
        public int getNum() {
            return num;
        }
        public void setDeep(int n) {
            this.deep = n;
```

```
    }
    public int getDeep() {
        return deep;
    }
    public void setContent(String n) {
        this.content = n;
    }
    public String getContent() {
        return content;
    }
}
```