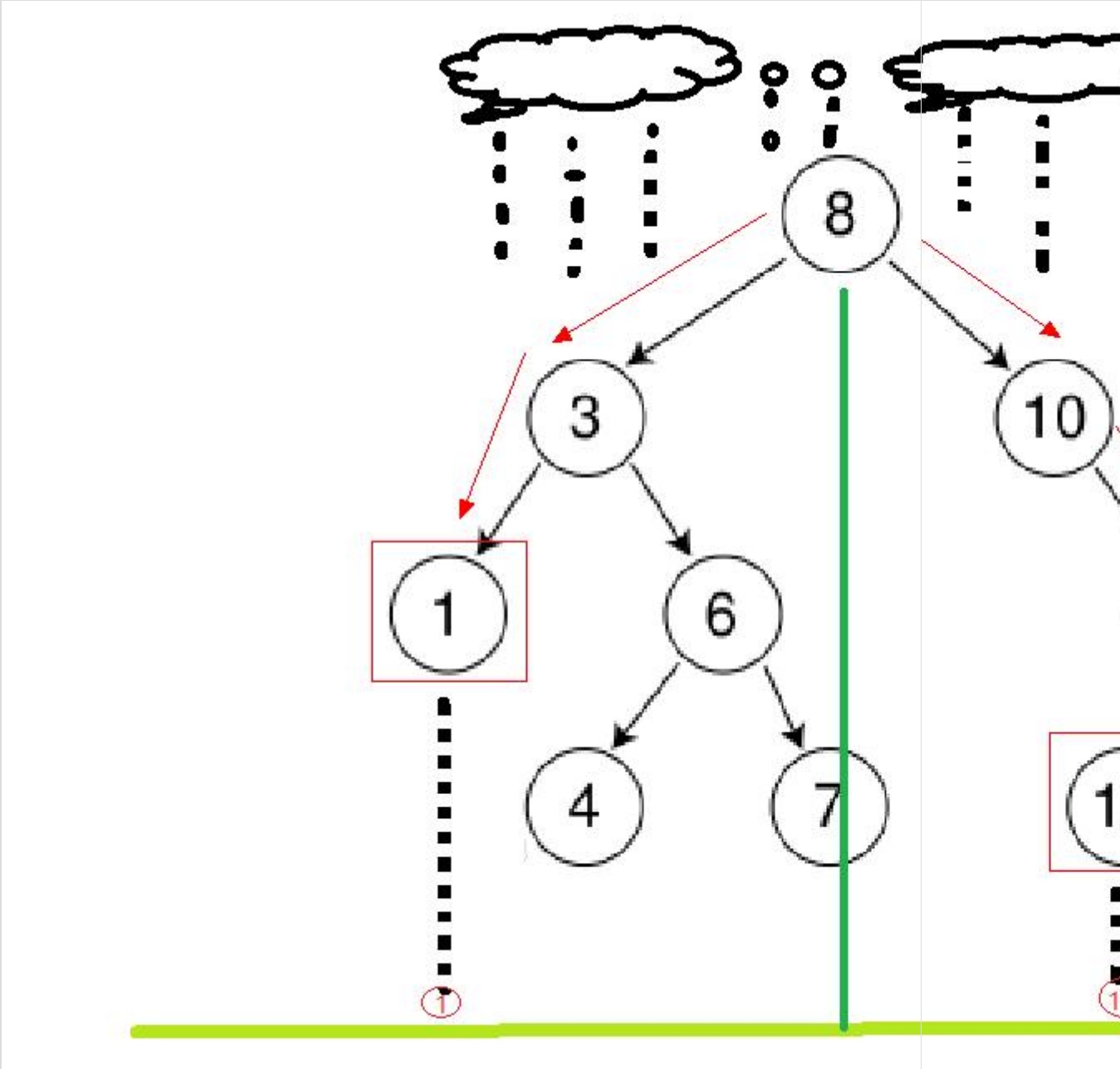
题目描述：

普通的伞在二维平面世界中，左右两侧均有一条边，而两侧伞边最下面各有一个伞坠子，雨滴落到伞面，逐步流到伞坠处，会将伞坠的信息携带并落到地面，随着日积月累，地面会呈现伞坠的信息。

1、为了模拟伞状雨滴效应，用二叉树来模拟二维平面伞（如下图所示），现在输入一串正整数数组序列（不含 0，数组成员至少是 1 个），若此数组序列是二叉搜索树的前序遍历的结果，那么请输出一个返回值 1，否则输出 0.

2、同时请将此序列构成的伞状效应携带到地面的数字信息输出来(左边伞坠信息，右边伞坠信息，详细参考示例图地面上数字)，若此树不存在左或右扇坠，则对应位置返回 0。同时若非二叉排序树那么左右伞坠信息也返回 0。



输入描述：

一个通过空格分割的整数序列字符串，数组不含 0，数组成员至少 1 个，输入的数组的任

意两个数字都互不相同，最多 1000 个正整数，正整数值范围 1~655350

输出描述：

输出如下三个值，以空格分隔：是否二叉排序树，左侧地面呈现的伞坠数字值，右侧地面呈现的伞坠数字值。

若是二叉排序树，则输出 1，否则输出 0（其左右伞坠值也直接赋值 0）。

若不存存在左侧或者右侧伞坠值，那么对应伞坠值直接赋值 0。

示例 1

输入：

```
8 3 1 6 4 7 10 14 13
```

输出：

```
1 1 13
```

说明：

1 表示是二叉搜索树前序遍历结果，1 表示左侧地面呈现的伞坠数字值，13 表示右侧地面呈现的伞坠数字值

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        List<Integer> treeList = new ArrayList<>();
        while(in.hasNextLine()){
            String line = in.nextLine();
            String[] split = line.split(" ");
            for (int i = 0; i < split.length; i++) {
                treeList.add(Integer.valueOf(split[i]));
            }
            int ifTree = 1;
            TreeNode head = new TreeNode(0);
            if(treeList.size() <= 0){
                ifTree = 0;
            }else{
                head = Main.buildTree(treeList, 0, treeList.size() - 1);
                if(head == null || head.val == -1){
                    ifTree = 0;
                }
            }
            if(ifTree == 0){
                System.out.println("0 0 0");
                return;
            }else{
                int leftMost = 0;
                TreeNode cur = head;
                if(cur.left == null){
                    leftMost = 0;
                }else{
                    while (cur.left != null || cur.right != null) {
```

```
                        if(cur.left!=null){
                            cur = cur.left;
                            continue;
                        }
                        cur = cur.right;
                    }
                    leftMost = cur.val;
                }

                cur = head;
                int rightMost = 0;
                if(cur.right != null){
                    while(cur.left !=null ||cur.right != null){
                        if(cur.right != null){
                            cur = cur.right;
                            continue;
                        }
                        cur = cur.left;
                    }
                    rightMost = cur.val;
                }

                System.out.println("1 "+leftMost +" "+rightMost);
            }
        }

}

public    static TreeNode buildTree(List<Integer> treeList, int left, int right){
        if(right <left){
            return null;
        }
        TreeNode head = new TreeNode(treeList.get(left));
        int i = left +1;
        while(i< treeList.size() && treeList.get(i) < treeList.get(left)){
            i++;
        }
        for (int j = i; j <= right; j++) {
            if(treeList.get(j) < treeList.get(left)){
                return new TreeNode(-1);
            }
        }
        head.left = buildTree(treeList, left +1, i - 1);
        head.right = buildTree(treeList, i, right);
```

```
            if((head.left != null &&head.left.val == -1) || (head.right != null && head.right.val ==
-1)){
                  return new TreeNode(-1);
            }
            return head;
      }
}
class TreeNode{
      public TreeNode(int val){
            this.val = val;
      }
      public TreeNode left;
      public TreeNode right;
      public int val;
}
```