数组左部分都比头节点的值小，用来生成头节点的左子树，剩下的部分用来生成右子树。

具体过程请参看如下代码中的 posArrayToBST 方法。

```
public class Node {
        public int value;
        public Node left;
        public Node right;

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

public Node posArrayToBST(int[] posArr) {
        if (posArr == null) {
                return null;
        }
        return posToBST(posArr, 0, posArr.length - 1);
}

public Node posToBST(int[] posArr, int start, int end) {
        if (start > end) {
                return null;
        }
        Node head = new Node(posArr[end]);
        int less = -1;
        int more = end;
        for (int i = start; i < end; i++) {
                if (posArr[end] > posArr[i]) {
                        less = i;
                } else {
                        more = more == end ? i : more;
                }
        }
        head.left = posToBST(posArr, start, less);
        head.right = posToBST(posArr, more, end - 1);
        return head;
}
```

判断一棵二叉树是否为搜索二叉树和完全二叉树

【题目】

给定一个二叉树的头节点 head，已知其中没有重复值的节点，实现两个函数分别判断这棵二叉树是否是搜索二叉树和完全二叉树。

【难度】

士 ★☆☆☆

【解答】

判断一棵二叉树是否是搜索二叉树，只要改写一个二叉树中序遍历，在遍历的过程中看节点值是否都是递增的即可。本书改写的是 Morris 中序遍历，所以时间复杂度为 $O(N)$，额外空间复杂度为 $O(1)$。有关 Morris 中序遍历的介绍，请读者阅读本书“遍历二叉树的神级方法”问题。需要注意的是，Morris 遍历分调整二叉树结构和恢复二叉树结构两个阶段，所以，当发现节点值降序时，不能直接返回 false，这么做可能会跳过恢复阶段，从而破坏二叉树的结构。

通过改写 Morris 中序遍历来判断搜索二叉树的过程请参看如下代码中的 isBST 方法。

```
public class Node {
        public int value;
        public Node left;
        public Node right;

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

public boolean isBST(Node head) {
        if (head == null) {
                return true;
        }
        boolean res = true;
        Node pre = null;
        Node cur1 = head;
        Node cur2 = null;
        while (cur1 != null) {
                cur2 = cur1.left;
                if (cur2 != null) {
                        while (cur2.right != null && cur2.right != cur1) {
                                cur2 = cur2.right;
                        }
                        if (cur2.right == null) {
                                cur2.right = cur1;
                                cur1 = cur1.left;
                                continue;
                        } else {
                                cur2.right = null;
                        }
                }
```

```
                if (pre != null && pre.value > cur1.value) {
                        res = false;
                }
                pre = cur1;
                cur1 = cur1.right;
        }
        return res;
}
```

判断一棵二叉树是否是完全二叉树，依据以下标准会使判断过程变得简单且易实现：

1. 按层遍历二叉树，从每层的左边向右边依次遍历所有的节点。
2. 如果当前节点有右孩子，但没有左孩子，直接返回 false。
3. 如果当前节点并不是左右孩子全有，那之后的节点必须都为叶节点，否则返回 false。
4. 遍历过程中如果不返回 false，遍历结束后返回 true。

判断是否是完全二叉树的全部过程请参看如下代码中的 isCBT 方法。

```
public boolean isCBT(Node head) {
        if (head == null) {
                return true;
        }
        Queue<Node> queue = new LinkedList<Node>();
        boolean leaf = false;
        Node l = null;
        Node r = null;
        queue.offer(head);
        while (!queue.isEmpty()) {
                head = queue.poll();
                l = head.left;
                r = head.right;
                if ((leaf&&(l!=null||r!=null)) || (l==null&&r!=null)) {
                        return false;
                }
                if (l != null) {
                        queue.offer(l);
                }
                if (r != null) {
                        queue.offer(r);
                } else {
                        leaf = true;
                }
        }
        return true;
}
```