```
public int getMaxLength(Node head, int sum) {
        HashMap<Integer, Integer> sumMap = new HashMap<Integer, Integer>();
        sumMap.put(0, 0); // 重要
        return preOrder(head, sum, 0, 1, 0, sumMap);
}

public int preOrder(Node head, int sum, int preSum, int level,
                int maxLen, HashMap<Integer, Integer> sumMap) {
        if (head == null) {
                return maxLen;
        }
        int curSum = preSum + head.value;
        if (!sumMap.containsKey(curSum)) {
                sumMap.put(curSum, level);
        }
        if (sumMap.containsKey(curSum - sum)) {
                maxLen = Math.max(level - sumMap.get(curSum - sum), maxLen);
        }
        maxLen = preOrder(head.left, sum, curSum, level + 1, maxLen, sumMap);
        maxLen = preOrder(head.right, sum, curSum, level + 1, maxLen, sumMap);
        if (level == sumMap.get(curSum)) {
                sumMap.remove(curSum);
        }
        return maxLen;
}
```

找到二叉树中的最大搜索二叉子树

【题目】

给定一棵二叉树的头节点 head，已知其中所有节点的值都不一样，找到含有节点最多的搜索二叉子树，并返回这棵子树的头节点。

例如，二叉树如图 3-16 所示。

这棵树中的最大搜索二叉子树如图 3-17 所示。

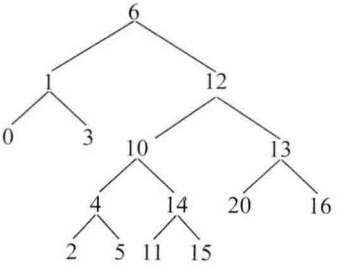

图 3-16

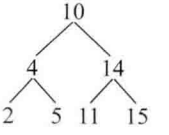

图 3-17

【要求】

如果节点数为 N，要求时间复杂度为 $O(N)$，额外空间复杂度为 $O(h)$，h 为二叉树的高度。

【难度】

尉 ★★☆☆

【解答】

以节点 node 为头的树中，最大的搜索二叉子树只可能来自以下两种情况。

第一种：如果来自 node 左子树上的最大搜索二叉子树是以 node.left 为头的；来自 node 右子树上的最大搜索二叉子树是以 node.right 为头的；node 左子树上的最大搜索二叉子树的最大值小于 node.value；node 右子树上的最大搜索二叉子树的最小值大于 node.value，那么以节点 node 为头的整棵树都是搜索二叉树。

第二种：如果不满足第一种情况，说明以节点 node 为头的树整体不能连成搜索二叉树。这种情况下，以 node 为头的树上的最大搜索二叉子树是来自 node 的左子树上的最大搜索二叉子树和来自 node 的右子树上的最大搜索二叉子树之间，节点数较多的那个。

通过以上分析，求解的具体过程如下：

1．整体过程是二叉树的后序遍历。

2．遍历到当前节点记为 cur 时，先遍历 cur 的左子树收集 4 个信息，分别是左子树上最大搜索二叉子树的头节点（lBST）、节点数（lSize）、最小值（lMin）和最大值（lMax）。再遍历 cur 的右子树收集 4 个信息，分别是右子树上最大搜索二叉子树的头节点（rBST）、节点数（rSize）、最小值（rMin）和最大值（rMax）。

3．根据步骤 2 所收集的信息，判断是否满足第一种情况，如果满足第一种情况，就返回 cur 节点，如果满足第二种情况，就返回 lBST 和 rBST 中较大的一个。

4．可以使用全局变量的方式实现步骤 2 中收集节点数、最小值和最大值的问题。

找到最大搜索二叉子树的具体过程请参看如下代码中的 biggestSubBST 方法。

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

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

```
public Node biggestSubBST(Node head) {
        int[] record = new int[3];
        return posOrder(head, record);
}

public Node posOrder(Node head, int[] record) {
        if (head == null) {
                record[0] = 0;
                record[1] = Integer.MAX_VALUE;
                record[2] = Integer.MIN_VALUE;
                return null;
        }
        int value = head.value;
        Node left = head.left;
        Node right = head.right;
        Node lBST = posOrder(left, record);
        int lSize = record[0];
        int lMin = record[1];
        int lMax = record[2];
        Node rBST = posOrder(right, record);
        int rSize = record[0];
        int rMin = record[1];
        int rMax = record[2];
        record[1] = Math.min(lMin, value);
        record[2] = Math.max(rMax, value);
        if (left == lBST && right == rBST && lMax < value && value < rMin) {
                record[0] = lSize + rSize + 1;
                return head;
        }
        record[0] = Math.max(lSize, rSize);
        return lSize > rSize ? lBST : rBST;
}
```

record[0]=节点数，[1]=最小值，[2]=最大值

lBST左子树上的最大搜索二叉树的头节点
rBST为右子树上的

判断是否符合情况1，即结合左右最大搜索二叉树

不符合则选择最大的一方

找到二叉树中符合搜索二叉树条件的最大拓扑结构

【题目】

给定一棵二叉树的头节点 head，已知所有节点的值都不一样，返回其中最大的且符合搜索二叉树条件的最大拓扑结构的大小。

例如，二叉树如图 3-18 所示。