```
        }
        int len = pre.length;
        int[] pos = new int[len];
        HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
        for (int i = 0; i < len; i++) {
                map.put(in[i], i);
        }
        setPos(pre, 0, len - 1, in, 0, len - 1, pos, len - 1, map);
        return pos;
}

// 从右往左依次填好后序数组 s
// si 为后序数组 s 该填的位置
// 返回值为 s 该填的下一个位置
public int setPos(int[] p, int pi, int pj, int[] n, int ni, int nj,
                int[] s, int si, HashMap<Integer, Integer> map) {
        if (pi > pj) {
                return si;
        }
        s[si--] = p[pi];
        int i = map.get(p[pi]);
        si = setPos(p, pj - nj + i + 1, pj, n, i + 1, nj, s, si, map);
        return setPos(p, pi + 1, pi + i - ni, n, ni, i - 1, s, si, map);
}
```

统计和生成所有不同的二叉树

【题目】

给定一个整数 N，如果 $N<1$，代表空树结构，否则代表中序遍历的结果为{1,2,3,…，N}。请返回可能的二叉树结构有多少。

例如，$N=-1$ 时，代表空树结构，返回 1；$N=2$ 时，满足中序遍历为{1, 2}的二叉树结构只有如图 3-49 所示的两种，所以返回结果为 2。

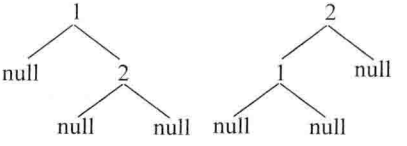

图 3-49

进阶：N 的含义不变，假设可能的二叉树结构有 M 种，请返回 M 个二叉树的头节点，每一棵二叉树代表一种可能的结构。

【难度】

尉 ★★☆☆

【解答】

如果中序遍历有序且无重复值，则二叉树必为搜索二叉树。假设 num(a)代表 a 个节点的搜索二叉树有多少种可能，再假设序列为{1, …, i, …, N}，如果以 1 作为头节点，1 不可能有左子树，故以 1 作为头节点有多少种可能的结构，完全取决于 1 的右子树有多少种可能结构，1 的右子树有 N-1 个节点，所以有 num(N-1)种可能。

如果以 i 作为头节点，i 的左子树有 i-1 个节点，所以可能的结构有 num(i-1)种，右子树有 N-i 个节点，所以有 num(N-i)种可能。故以 i 为头节点的可能结构有 num(i-1)×num(N-i)种。

如果以 N 作为头节点，N 不可能有右子树，故以 N 作为头节点有多少种可能，完全取决于 N 的左子树有多少种可能，N 的左子树有 N-1 个节点，所以有 num(N-1)种。

把从 1 到 N 分别作为头节点时，所有可能的结构加起来就是答案，可以利用动态规划来加速计算的过程，从而做到 $O(N^2)$的时间复杂度。

具体请参看如下代码中的 numTrees 方法。

```
public int numTrees(int n) {
        if (n < 2) {
                return 1;
        }
        int[] num = new int[n + 1];
        num[0] = 1;
        for (int i = 1; i < n + 1; i++) {
                for (int j = 1; j < i + 1; j++) {
                        num[i] += num[j - 1] * num[i - j];
                }
        }
        return num[n];
}
```

进阶问题与原问题的过程其实是很类似的。如果要生成中序遍历是{a…b}的所有结构，就从 a 开始一直到 b，枚举每一个值作为头节点，把每次生成的二叉树结构的头节点都保存下来即可。假设其中一次是以 i 值为头节点的($a \leqslant i \leqslant b$)，以 i 头节点的所有结构按如下步骤生成：

1．用{a…i-1}递归生成左子树的所有结构，假设所有结构的头节点保存在 listLeft 链表中。

2．用{a…i+1}递归生成右子树的所有结构，假设所有结构的头节点保存在 listRight 链表中。

3．在以 i 为头节点的前提下，listLeft 中的每一种结构都可以与 listRight 中的每一种结构构成单独的结构，且和其他任何结构都不同。为了保证所有的结构之间不互相交叉，所以对每一种结构都复制出新的树，并记录在总的链表 res 中。

具体过程请参看如下代码中的 generateTrees 方法。

```
public List<Node> generateTrees(int n) {
        return generate(1, n);
}

public List<Node> generate(int start, int end) {
        List<Node> res = new LinkedList<Node>();
        if (start > end) {
                res.add(null);
        }
        Node head = null;
        for (int i = start; i < end + 1; i++) {
                head = new Node(i);
                List<Node> lSubs = generate(start, i - 1);
                List<Node> rSubs = generate(i + 1, end);
                for (Node l : lSubs) {
                        for (Node r : rSubs) {
                                head.left = l;
                                head.right = r;
                                res.add(cloneTree(head));
                        }
                }
        }
        return res;
}

public Node cloneTree(Node head) {
        if (head == null) {
                return null;
        }
        Node res = new Node(head.value);
        res.left = cloneTree(head.left);
        res.right = cloneTree(head.right);
        return res;
}
```