

思路1：

recursion:

二叉搜索可以拆解成子问题，如果左边也是二叉树，右边也是二叉树，且左边都小于当前点，右边都大于当前点，return true

可以设置上下界，左边的recursion将当前点设为上界，就能满足左边点必然小于当前点，

同理右边recursion将当前点设为下界

class Solution {

    public boolean isValidBST(TreeNode root) {

        return bstRecursive(root, null, null);

    }

    public boolean bstRecursive(TreeNode node, Integer lower, Integer upper) {

        if (node == null) {

            return true;

        } //到底，可以return true了

        int val = node.val;

        if (lower != null && val <= lower) {

            return false;

        } //违反了上下界，false

        if (upper != null && val >= upper) {

            return false;

        }

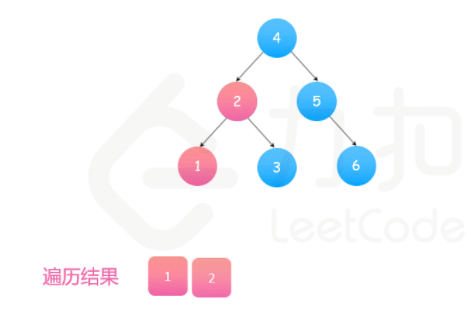
        return (bstRecursive(node.right, val, upper)&&bstRecursive(node.left, lower, val));

    } //左右都return true

}

思路2：中序遍历

二叉搜索树有一个性质，那就是按照中序遍历，必然是从小到大排列的



那我们只要中序遍历，用一个变量存储上一个node.var，如果大于等于了，return false

class Solution {

    long pre = Long.MIN\_VALUE; //其实最好应该加个 static

    public boolean isValidBST(TreeNode root) {

        if (root == null) {

            return true; //base case,如果一个点到底了，那么说明他左子树右子树为空，必然满足bST

        }

        // 访问左子树

        if (!isValidBST(root.left)) {

            return false;

        } //中序遍历，先遍历左边，在处理当前node，最后遍历右边，这里我们用if来处理左边recursion，因为如果是true我们就可以继续循环

        // 访问当前节点：如果当前节点小于等于中序遍历的前一个节点，说明不满足BST，返回 false；否则继续遍历。

        if (root.val <= pre) {

            return false;

        }

        pre = root.val;

        // 访问右子树

        return isValidBST(root.right); //右边

    }

}