597. Subtree with Maximum Average ☆

Description | Notes | Testcase | Judge

Given a binary tree, find the subtree with maximum average. Return the root of the subtree.

Notice

LintCode will print the subtree which root is your return node.
It's guaranteed that there is only one subtree with maximum average.

Have you met this question in a real interview? Yes

Example

Given a binary tree:

return the node `11`.

```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
```

```
 * }
 */


public class Solution {
    /*
     * @param root: the root of binary tree
     * @return: the root of the maximum average of subtree
     */
    private class ResultType {
        private int sum;
        private int size;
        private ResultType (int sum, int size) {
            this.sum = sum;
            this.size = size;
        }
    }
    TreeNode subtree = null;
    float maxAverage = Integer.MIN_VALUE;
    public TreeNode findSubtree2(TreeNode root) {
        // write your code here
        helper(root);
        return subtree;
    }
    private ResultType helper(TreeNode root) {
        if (root == null) {
            return new ResultType(0, 0);
        }

        ResultType left = helper(root.left);
        ResultType right = helper(root.right);
        int sum = left.sum + right.sum + root.val;
        ResultType result = new ResultType(sum, left.size + right.size + 1);
        float average = (float) sum / result.size;
        if (maxAverage < average) {
            maxAverage = average;
            subtree = root;
        }
        return result;

    }
}
```

596. Minimum Subtree ☆

Description | Notes | Testcase | Judge

Given a binary tree, find the subtree with minimum sum. Return the root of the subtree.

Notice

LintCode will print the subtree which root is your return node.
It's guaranteed that there is only one subtree with minimum sum and the given binary tree is not an empty tree.

Have you met this question in a real interview? Yes

Example

Given a binary tree:

```
     1
   /   \
 -5     2
 / \   /  \
0   2 -4  -5
```

return the node `1`.

```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */


public class Solution {
    /*
     * @param root: the root of binary tree
     * @return: the root of the minimum subtree
     */
    int minSum = Integer.MAX_VALUE;
    TreeNode subtree = null;
    public TreeNode findSubtree(TreeNode root) {
        // write your code here
        helper(root);
        return subtree;
    }
    private int helper(TreeNode root) {
        if (root == null) {
            return 0;
        }
        int leftSum = helper(root.left);
        int rightSum = helper(root.right);
        int sum = leftSum + rightSum + root.val;
        if (minSum > sum) {
            minSum = sum;
            subtree = root;
        }
        return sum;
    }
}
```

480. Binary Tree Paths ☆

Description | Notes | >_ Testcase | Judge

Given a binary tree, return all root-to-leaf paths.

Have you met this question in a real interview? Yes

Example

Given the following binary tree:

```
   1
 /   \
2     3
 \
  5
```

All root-to-leaf paths are:

```
[
  "1->2->5",
  "1->3"
]
```

```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */


public class Solution {
    /*
     * @param root: the root of the binary tree
     * @return: all root-to-leaf paths
     */
    public List<String> binaryTreePaths(TreeNode root) {
        // write your code here
        List<String> paths = new ArrayList<>();
        if (root == null) {
            return paths;
        }
        if (root.left == null && root.right == null) {
            paths.add(""+root.val);
            return paths;
        }
        List<String> leftPaths = binaryTreePaths(root.left);
        List<String> rightPaths = binaryTreePaths(root.right);
        // if a node only has one child, then it just have one path
        for (String path:leftPaths) {
            paths.add(root.val + "->" + path);
        }

         for (String path:rightPaths) {
            paths.add(root.val + "->" + path);

         }

        return paths;
    }
}
```

453. Flatten Binary Tree to Linked List ☆

Description | Notes | >_ Testcase | Judge

Flatten a binary tree to a fake "linked list" in pre-order traversal.

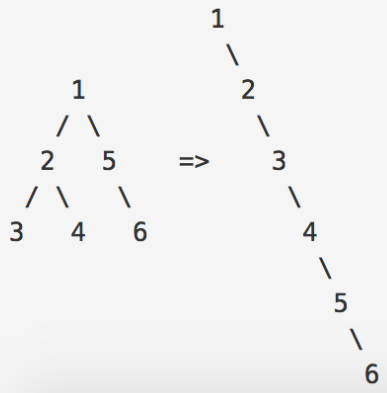
Here we use the *right* pointer in TreeNode as the *next* pointer in ListNode.

Notice

Don't forget to mark the left child of each node to null. Or you will get Time Limit Exceeded or Memory Limit Exceeded.

Have you met this question in a real interview? Yes

Example



```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */


public class Solution {
    /*
     * @param root: a TreeNode, the root of the binary tree
     * @return:
     */
    public void flatten(TreeNode root) {
      if (root == null) {
            return;
        }
        Stack<TreeNode> stack = new Stack<>();
        stack.push(root);
        while(!stack.empty()) {
            TreeNode node = stack.pop();
            if(node.right != null) {
                stack.push(node.right);
            }
            if (node.left != null) {
                stack.push(node.left);
            }
            // connect
            node.left = null;
            if (stack.empty()) {
                node.right = null;
            } else {
                node.right = stack.peek();
            }
        }

    }
}

// traverse version
// public class Solution {
```

```
//     /*
//      * @param root: a TreeNode, the root of the binary tree
//      * @return:
//      */
//      // flatten(root): lastNode+root+flatten(left subtree) + flatten(rightsubtree)
//      // flatten(2) 要让2先连1
//     private TreeNode lastNode = null;
//     public void flatten(TreeNode root) {
//         // write your code here
//         if (root == null) {
//             return;
//         }
//         if (lastNode != null) {
//             lastNode.left = null;
//             lastNode.right = root;
//         }
//         if (lastNode == null) {
//             // do nothing
//         }
//         lastNode = root;
//         // IMPORTANT!
//         TreeNode right = root.right;
//         // the root.right node will change after flatten(root.left);
//         flatten(root.left);
//         flatten(right);
//     }
// }
```

97. Maximum Depth of Binary Tree ☆

Description | Notes | >_ Testcase | Judge

Given a binary tree, find its maximum depth.

The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

Have you met this question in a real interview? Yes

Example

Given a binary tree as follow:

```
  1
 / \
2   3
   / \
  4   5
```

The maximum depth is `3`.

```java
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */
public class Solution {
    /**
     * @param root: The root of binary tree.
     * @return: An integer.
     */
    public int maxDepth(TreeNode root) {
        // write your code here
        if (root == null) {
            return 0;
        }
        int left = maxDepth(root.left);
        int right = maxDepth(root.right);
        return Math.max(left,right) + 1;
    }
}
```

93. Balanced Binary Tree ☆

Description | Notes | >_ Testcase | Judge

Given a binary tree, determine if it is height-balanced.

For this problem, a height-balanced binary tree is defined as a binary tree in which the depth of the two subtrees of every node never differ by more than 1.

Have you met this question in a real interview? Yes

Example

Given binary tree A = `{3,9,20,#,#,15,7}`, B = `{3,#,20,15,7}`

```
A)  3            B)    3
   / \                  \
  9  20                 20
    /  \                / \
   15   7              15  7
```

The binary tree A is a height-balanced binary tree, but B is not.

```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */


public class Solution {
    /*
     * @param root: The root of binary tree.
     * @return: True if this Binary tree is Balanced, or false.
     */
     boolean isBalanced = true;
    public boolean isBalanced(TreeNode root) {
        // write your code here
        depth(root);
        return isBalanced;
    }
    private int depth(TreeNode root) {
        if (root == null) {
            return 0;
        }
        int left = depth(root.left);
        int right = depth(root.right);
        if (Math.abs(left-right)>1) {
            isBalanced = false;
        }
        return Math.max(left,right) + 1;
    }

}
```

67. Binary Tree Inorder Traversal ☆

Description | Notes | >_ Testcase | Judge

Given a binary tree, return the *inorder* traversal of its nodes' values.

Have you met this question in a real interview? Yes

Example

Given binary tree `{1,#,2,3}`,

```
   1
    \
     2
    /
   3
```

return `[1,3,2]`.

```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */


public class Solution {
    /*
     * @param root: A Tree
     * @return: Inorder in ArrayList which contains node values.
     */
     // left root right
    List<Integer> result = new ArrayList<>();

    public List<Integer> inorderTraversal(TreeNode root) {
        // write your code here
        if (root == null) {
            return result;
        }
        helper(root);
        return result;
    }
    private void helper(TreeNode root) {
        if (root.left != null) {
            helper(root.left);
        }
        result.add(root.val);
        if (root.right != null) {
            helper(root.right);
        }
    }

}



// public class Solution {
//     /*
//      * @param root: A Tree
//      * @return: Inorder in ArrayList which contains node values.
//      */
```

```
//      // left root right
//     public List<Integer> inorderTraversal(TreeNode root) {
//         // write your code here
//         Stack<TreeNode> stack = new Stack<>();
//         ArrayList<Integer> result = new ArrayList<Integer>();
//         TreeNode curt = root;
//         while(curt != null || !stack.empty()) {
//             while (curt != null) {
//                 stack.push(curt);
//                 curt = curt.left;
//             }
//             curt = stack.pop();
//             result.add(curt.val);
//             curt = curt.right;

//         }
//         return result;
//     }
// }

// 1. curt = root
// while（curt！=null || stack！= empty）：
// 2. while（）：顺着curt迭代到最左边的点，全都加进stack
// 3. curt = stack.pop() which is 最左边的点
// 4. result.add(curt)
// 5. curt=curt.right
```

66. Binary Tree Preorder Traversal ☆

Description | Notes | >_ Testcase | Judge

Given a binary tree, return the preorder traversal of its nodes' values.

Have you met this question in a real interview? Yes

Example

Given:

```
    1
   / \
  2   3
 / \
4   5
```

return `[1,2,4,5,3]`.

Challenge

```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */


public class Solution {
```

```
    /*
     * @param root: A Tree
     * @return: Preorder in ArrayList which contains node values.
     */

     // root left right
     List<Integer> result = new ArrayList<>();
    public List<Integer> preorderTraversal(TreeNode root) {
        // write your code here
        Stack<TreeNode> stack = new Stack<TreeNode>();
        List<Integer> result = new ArrayList<Integer>();
        if (root == null) {
            return result;
        }
        stack.push(root);
        while(!stack.empty()) {
            TreeNode node = stack.pop();
            result.add(node.val);
            if (node.right != null) {
                stack.push(node.right);
            }
            if (node.left != null) {
                stack.push(node.left);
            }
        }
        return result;
        /* stack 过程
        1. 把root加进stack
        while：
        2. node = pop 出来最上面的一个
        3. add node to list
        4. push右节点进stack
        5. 加左节点进stack
        */
    }

}
// public class Solution {
//     /*
//      * @param root: A Tree
//      * @return: Preorder in ArrayList which contains node values.
//      */

//      // root left right
//      List<Integer> result = new ArrayList<>();
//     public List<Integer> preorderTraversal(TreeNode root) {
//         // write your code here
//         if(root != null) {
```

```
//             helper(root);
//         }
//         return result;
//     }
//     private void helper(TreeNode root) {
//         result.add(root.val);
//         if (root.left != null) {
//             helper(root.left);
//         }
//         if (root.right != null) {
//             helper(root.right);
//         }
//     }
// }
```

578. Lowest Common Ancestor III ☆

Description | Notes | Testcase | Judge

Given the root and two nodes in a Binary Tree. Find the lowest common ancestor(LCA) of the two nodes.
The lowest common ancestor is the node with largest depth which is the ancestor of both nodes.
Return `null` if LCA does not exist.

Notice

node A or node B may not exist in tree.

Have you met this question in a real interview? Yes

Example

For the following binary tree:

```
  4
 / \
3   7
   / \
  5   6
```

LCA(3, 5) = `4`

```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */


public class Solution {
    /*
     * @param root: The root of the binary tree.
     * @param A: A TreeNode
     * @param B: A TreeNode
     * @return: Return the LCA of the two nodes.
     */
     // 和LCA1的区别是，输入的两个子节点有可能不属于这个tree
    //  1. 如果节点root的左子树有个A, 右子树有个B，那么root就是LCA。return root
    //  2. 如果root左子树有target，返回target
    //  3. 如果root右自树有target，返回target
class ResultType {
    public boolean a_exist, b_exist;
    public TreeNode node;
    ResultType(boolean a, boolean b, TreeNode n) {
        a_exist = a;
        b_exist = b;
        node = n;
    }
}
    public TreeNode lowestCommonAncestor3(TreeNode root, TreeNode A, TreeNode B) {
        // write your code here

        ResultType rt = helper(root, A, B);
        if (rt.a_exist && rt.b_exist)
            return rt.node;
        else
            return null;

    }
    public ResultType helper(TreeNode root, TreeNode A, TreeNode B) {
        if (root == null)
            return new ResultType(false, false, null);
```

```
        ResultType left_rt = helper(root.left, A, B);
        ResultType right_rt = helper(root.right, A, B);

        boolean a_exist = left_rt.a_exist || right_rt.a_exist || root == A;
        boolean b_exist = left_rt.b_exist || right_rt.b_exist || root == B;

        if (root == A || root == B)
            return new ResultType(a_exist, b_exist, root);

        if (left_rt.node != null && right_rt.node != null)
            return new ResultType(a_exist, b_exist, root);
        if (left_rt.node != null)
            return new ResultType(a_exist, b_exist, left_rt.node);
        if (right_rt.node != null)
            return new ResultType(a_exist, b_exist, right_rt.node);

        return new ResultType(a_exist, b_exist, null);
    }

}
```

95. Validate Binary Search Tree ☆

Description | Notes | >_ Testcase | Judge

Given a binary tree, determine if it is a valid binary search tree (BST).

Assume a BST is defined as follows:

- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.
- A single node tree is a BST

Have you met this question in a real interview? Yes

Example

An example:

```
  2
 / \
1   4
   / \
  3   5
```

The above binary tree is serialized as `{2,1,4,#,#,3,5}` (in level order).

```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */


public class Solution {
    /*
     * @param root: The root of binary tree.
     * @return: True if the binary tree is BST, or false
     */
     // root 大于 左子树最大的点
     // root 小于 右子树最小的点
    public boolean isValidBST(TreeNode root) {
        // write your code here
        if (root == null) {
            return true;
        }
        return dfs(root).isBST;

    }
    public class ReturnType {
        int min;
        int max;
        boolean isBST;
        public ReturnType(int min, int max, boolean isBST) {
            this.min = min;
            this.max = max;
            this.isBST = isBST;
        }
    }
    // BST:
    // 1. Left tree is BST;
    // 2. Right tree is BST;
    // 3. root value is bigger than the max value of left tree and
    // smaller than the min value of the right tree.
    private ReturnType dfs(TreeNode root) {
```

```
        // initial: min:Integer.MAX_VALUE max:Integer.MIN_VALUE
        ReturnType ret = new ReturnType(Integer.MAX_VALUE, Integer.MIN_VALUE, true);
        if (root == null) {
            return ret;
        }
        ReturnType left = dfs(root.left);
        ReturnType right = dfs(root.right);
        if (!left.isBST || !right.isBST) {
            ret.isBST = false;
            return ret;
        }
// 判断Root.left != null是有必要的，如果root.val是MAX 或是MIN value,判断会失误
        if (root.left != null && root.val <= left.max) {
            ret.isBST = false;
            return ret;
        }
        if (root.right != null && root.val >= right.min) {
            ret.isBST = false;
            return ret;
        }

        // root 这个树是BST
        // 如果只有一个节点最小值是root.val，  否则left.min
        ret.min = Math.min(root.val,left.min);
        //
        ret.max = Math.max(root.val,right.max);
        ret.isBST = true;
        return ret;


    }

}
```