SlidesPractice_L2_BinaryTree_Diagram

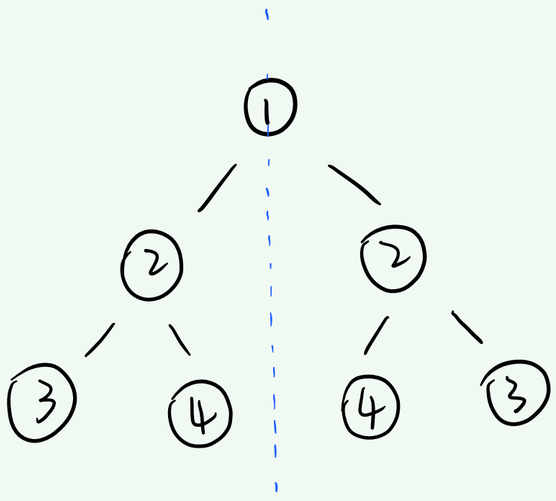
4/5/17 2:58 PM

preorder : root - left - right

inorder : left - root - right

postorder : left - right - root

101_Symmetric Tree



Compare 2 nodes per time

↓

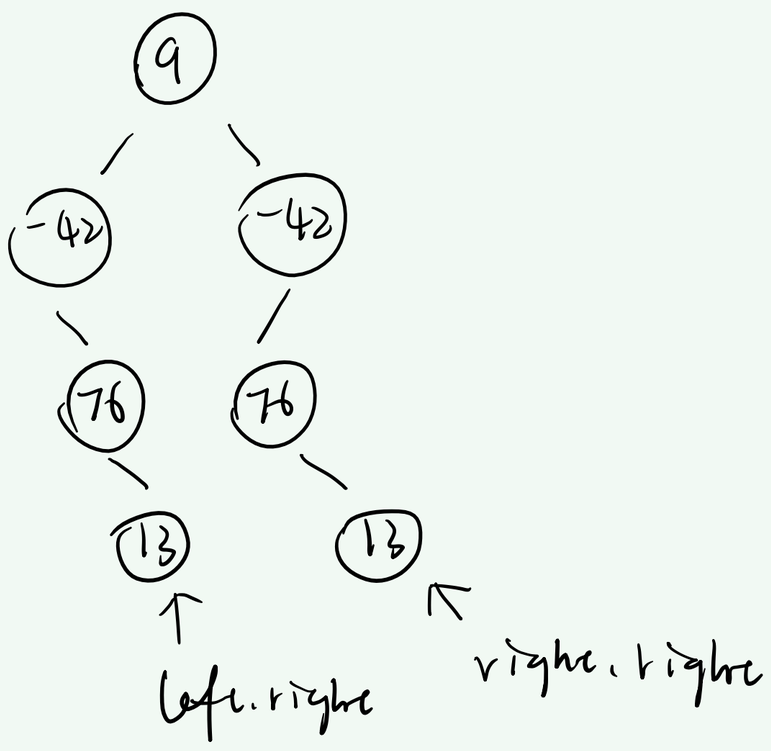
Currently compare left and right

↓

next compare (left.left) with (right.right)

AND (left.right) with (right.left)

Check wrong answer:



queue : | r | l |
| --- | --- |
| -42 | -42 |

left: -42

right: -42

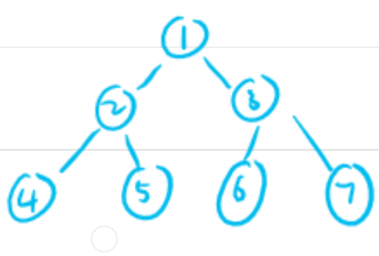
queue : | r.l | l.r | r.r | l.l |
| --- | --- | --- | --- |
| 76 | 76 | null | null |

left: null

right: null → Should continue

not "return true"!

226_Invert Binary Tree



loop 0: stack: 1
result: empty

loop 1: temp = 1
stack = 3, 2
result = 1

loop 2: temp = 3
stack = 7 6 2
result = 3, 1

loop 3: temp = 7
stack = 6 2
result = 7, 3, 1

loop 4: temp = 6
stack = 2
result = 6, 7, 3, 1

loop 5: temp = 2
stack = 5, 4
result = 2, 6, 7, 3, 1

loop 6: temp = 5
stack = 4
result = 5, 2, 6, 7, 3, 1

loop 7: temp = 4
stack = empty
result = 4, 5, 2, 6, 7, 3, 1

Java

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<Integer> postorderTraversal(TreeNode root) {
        LinkedList<Integer> result = new LinkedList<>();
        // (1) List<Integer> result = new ArrayList<>();
        //     result cannot use methods from Queue.

        // Queue<Integer> result = new LinkedList<>();
        if (root == null) {
            return result;
        }
        Stack<TreeNode> stack = new Stack<>();
        stack.push(root);
        while (!stack.isEmpty()) {
            TreeNode temp = stack.pop();
            result.addFirst(temp.val);
            if (temp.left != null) {
                stack.push(temp.left);
            }
            if (temp.right != null) {
                stack.push(temp.right);
            }
        }
        return result;

    }
}
```

Custom Testcase