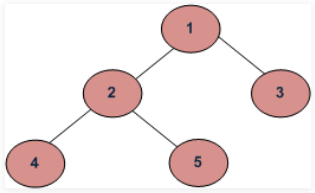
99. Recover Binary Search Tree

link

Morris Traversal方法遍历二叉树（非递归，不用栈，O(1)空间）

这个问题是O(1)所以只考虑morris遍历, 理解题目本质是找到两个顺序不同的, 节点, 然后把他们交换.



Example Tree

Depth First Traversals:
(a) Inorder (Left, Root, Right) : 4 2 5 1 3
(b) Preorder (Root, Left, Right) : 1 2 4 5 3
(c) Postorder (Left, Right, Root) : 4 5 2 3 1

代码: 题目本质是数组排序换两个数可以达到正常升序, 找到两个数使他们交换可以满足, 找到第一个pre > current的pre和第二个pre > current的after, 注意first == null条件, 忘了加

```
public void recoverTree(TreeNode root) {
    if(root != null){
        TreeNode current = root;
        TreeNode rightNode = null;
        TreeNode pre = null;
        TreeNode first = null, second = null;
        while(current != null){
            if(current.left != null){
                rightNode = current.left;
                while(rightNode.right != null && rightNode.right != current){
                    rightNode = rightNode.right;
                }
                if(rightNode.right == null){
                    rightNode.right = current;
                    current = current.left;
                    continue;
                }else{
                    rightNode.right = null;
                }
            }
```

```
//这里得注意是first==null否则第二次会把第一次复写
                if(first == null && pre != null && pre.val > current.val){
                    first = pre;
                }
                if(first != null && pre.val > current.val){
                    second = current;
                }

                pre = current;
                current = current.right;
            }
            if(first != null && second != null){
                int tmp = first.val;
                first.val = second.val;
                second.val = tmp;
            }
        }
    }
```

link翻转二叉树, 线索话 morris, inorder

```
    public void morrisHelper(TreeNode head, List<Integer>res){
        TreeNode rightNode = null;
        TreeNode current = head;
        while(current != null){
            if(current.left != null){
                rightNode = current.left;
                while(rightNode.right != null && rightNode.right != current){
                    rightNode = rightNode.right;
                }
                if(rightNode.right == null){
//这里是第一次访问父节点
                    rightNode.right = current;
                    current = current.left;
                    continue;
                }else{
                    rightNode.right = null;
                }
            }
//这里的res能下来每次都是第二次为rightNode.right!=null的时候.
            res.add(current.val);
            current = current.right;
        }
```

preorder

```
public static void morrisPre(Node head) {
    if(head == null){
        return;
    }
    Node cur = head;
    Node mostRight = null;
    while (cur != null){
        mostRight = cur.left;
        if(mostRight != null){
            while (mostRight.right !=null && mostRight.right != cur){
```

```
                mostRight = mostRight.right;
            }
            if(mostRight.right == null){
                mostRight.right = cur;
                System.out.print(cur.value+" ");
                cur = cur.left;
                continue;
            }else {
                mostRight.right = null;
            }
        }else {
//上面加了一次, 没有else这里会重复, 因为上面已经加了每个中间节点.
            System.out.print(cur.value + " ");
        }
        cur = cur.right;
    }
    System.out.println();
}
```

中序解法 遍历二叉树, 找到第一个比之前大的节点,与第二个比之前大的节点, 然后交换他们(这里只改变了val).

```
public class Solution {

    TreeNode firstElement = null;
    TreeNode secondElement = null;
    // The reason for this initialization is to avoid null pointer exception in the
    first comparison when prevElement has not been initialized
    TreeNode prevElement = new TreeNode(Integer.MIN_VALUE);

    public void recoverTree(TreeNode root) {

        // In order traversal to find the two elements
        traverse(root);

        // Swap the values of the two nodes
        int temp = firstElement.val;
        firstElement.val = secondElement.val;
        secondElement.val = temp;
    }

    private void traverse(TreeNode root) {

        if (root == null)
            return;

        traverse(root.left);

        // Start of "do some business",
        // If first element has not been found, assign it to prevElement (refer to 6
    in the example above)
        if (firstElement == null && prevElement.val >= root.val) {
            firstElement = prevElement;
        }

        // If first element is found, assign the second element to the root (refer
    to 2 in the example above)
        if (firstElement != null && prevElement.val >= root.val) {
```

```
            secondElement = root;
        }
        prevElement = root;

        // End of "do some business"

        traverse(root.right);
}
```

代码: 题目本质是数组排序换两个数可以达到正常升序, 找到两个数使他们交换可以满足, 找到第一个pre > current的pre和第二个pre > current的after, 注意first == null条件, 忘了加

```
    public void recoverTree(TreeNode root) {
        if(root != null){
            TreeNode current = root;
            TreeNode rightNode = null;
            TreeNode pre = null;
            TreeNode first = null, second = null;
            while(current != null){
                if(current.left != null){
                    rightNode = current.left;
                    while(rightNode.right != null && rightNode.right != current){
                        rightNode = rightNode.right;
                    }
                    if(rightNode.right == null){
                        rightNode.right = current;
                        current = current.left;
                        continue;
                    }else{
                        rightNode.right = null;
                    }
                }
//这里得注意是first==null否则第二次会把第一次复写
                if(first == null && pre != null && pre.val > current.val){
                    first = pre;
                }
                if(first != null && pre.val > current.val){
                    second = current;
                }

                pre = current;
                current = current.right;
            }
            if(first != null && second != null){
                int tmp = first.val;
                first.val = second.val;
                second.val = tmp;
            }
        }
    }
```