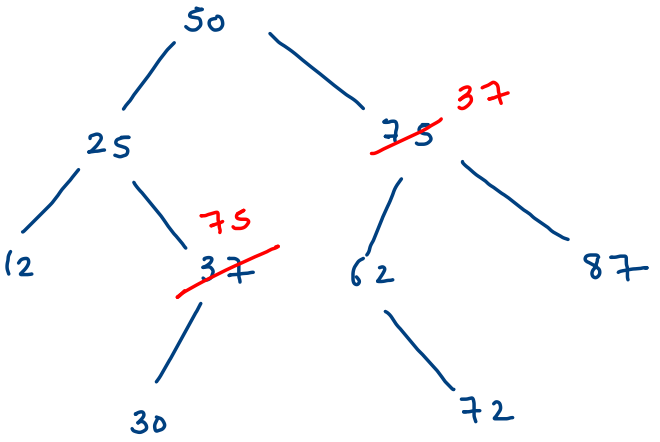
99. Recover Binary Search Tree

You are given the `root` of a binary search tree (BST), where the values of **exactly** two nodes of the tree were swapped by mistake. *Recover the tree without changing its structure.*

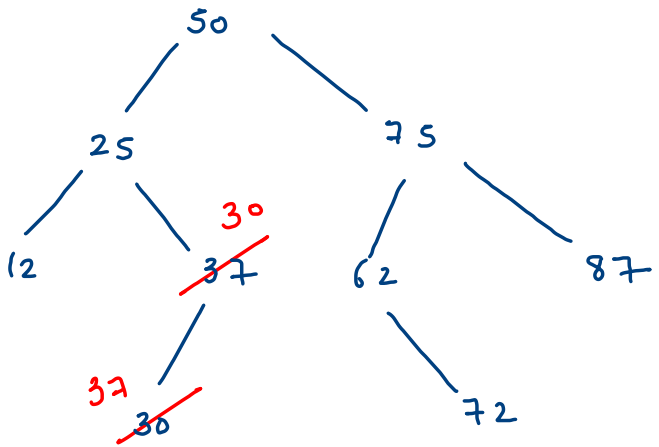
first = p1
second = c2



(after) In: 12 25 30 75 50 62 72 37 87

first mistake: 75 50 (p1 c1)

second mistake: 72 37 (p2 c2)



first mistake

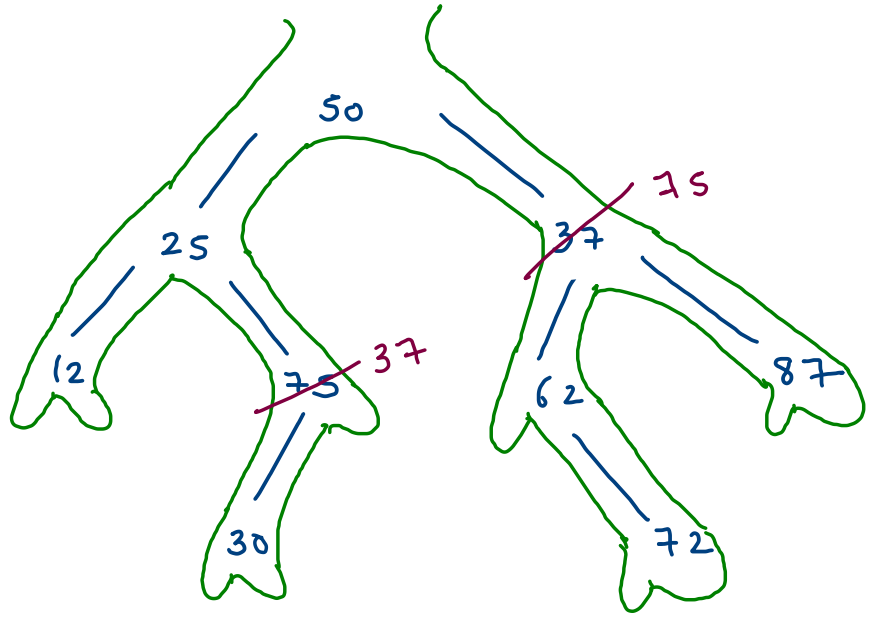
inorder: 12 25 37 30 50 62 72 75 87

p1 c1

first = p1

second = c1

```java
public void helper(TreeNode curr) {
    if(curr == null) {
        return;
    }

    helper(curr.left);

    //work
    if(prev != null && prev.val > curr.val) {
        if(first == null) {
            //first mistake
            first = prev;
            second = curr;
        }
        else {
            second = curr;
        }
    }
    prev = curr;

    helper(curr.right);
}
```

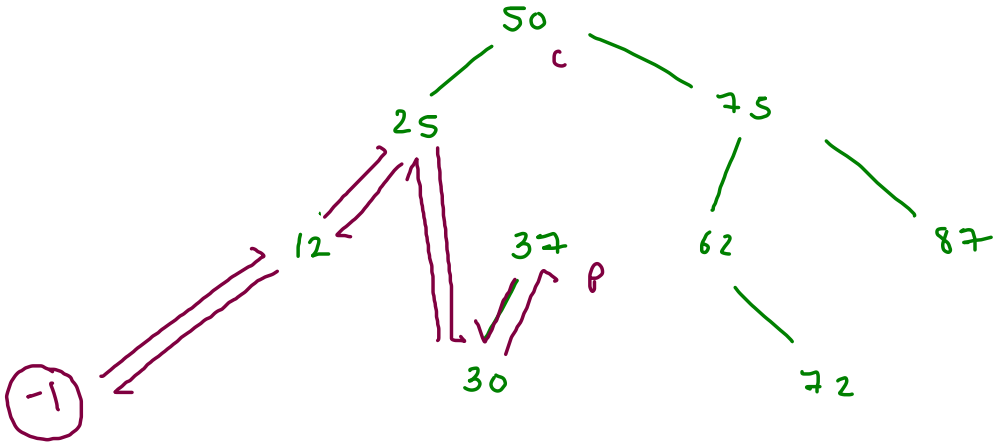


prev = ~~null~~ ~~12~~ ~~25~~ ~~30~~ ~~75~~ ~~50~~ ~~62~~ ~~72~~ ~~37~~ ~~87~~

first = ~~x~~ 75

second = ~~x~~ ~~50~~ 37

1534 · Convert Binary Search Tree to Sorted Doubly Linked List




```
p.right = c;
c.left = p;
```

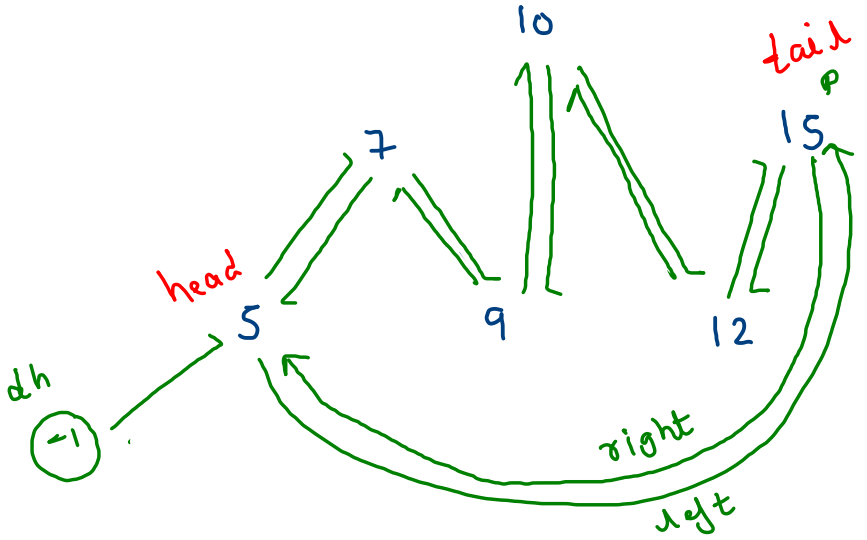
```
ll Node {
    int val;
    prev;
    next;
}
```

```
TreeNode {
    int val;
    left;
    right;
}
```

left :: prev

right :: next

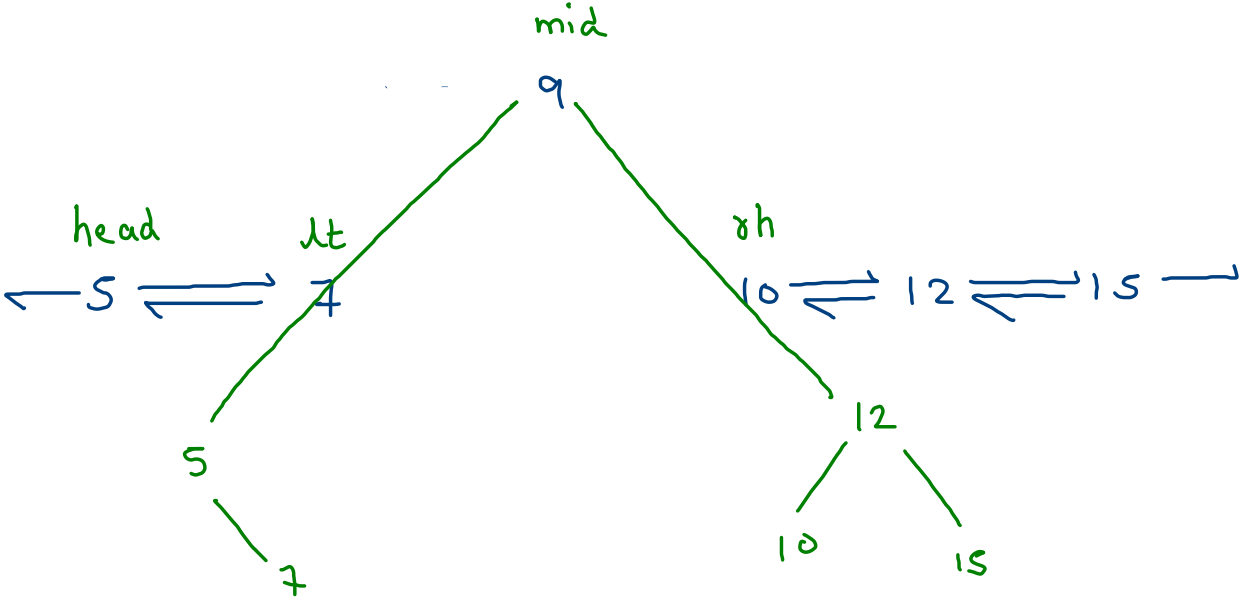
```java
public void helper(TreeNode curr) {
    if(curr == null) {
        return;
    }

    helper(curr.left);

    //work
    prev.right = curr;
    curr.left = prev;
    prev = curr;

    helper(curr.right);
}
```



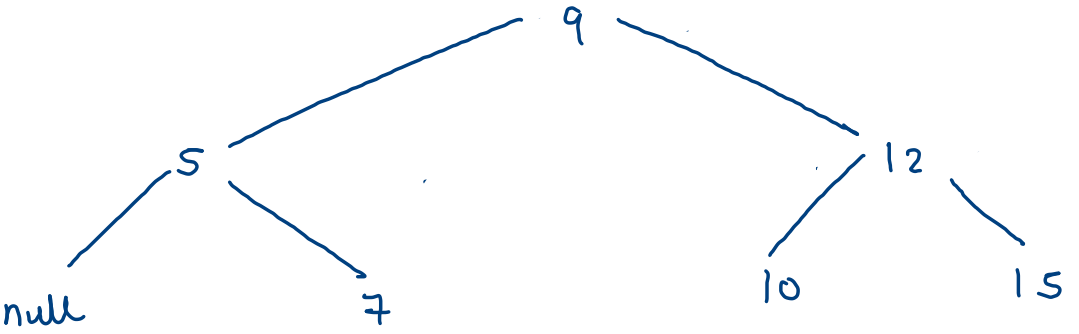

```java
TreeNode head = dh.right;
TreeNode tail = prev;

//dll -> cdll
head.left = tail;
tail.right = head;
```

Convert Sorted Doubly Linked List To Binary Search Tree

left :: prev
right :: next



```java
// left : prev, right : next
public static Node SortedDLLToBST(Node head) {
    if(head == null || head.right == null) {
        return head;
    }

    Node mid = mid(head);

    Node lt = mid.left;
    Node rh = mid.right;

    //break connections
    if(lt != null) {
        lt.right = mid.left = null; //break connection between lt & mid
    }
    mid.right = rh.left = null; //break connection mid & rh

    mid.left = SortedDLLToBST(lt != null ? head : null);
    mid.right = SortedDLLToBST(rh);

    return mid;
}
```

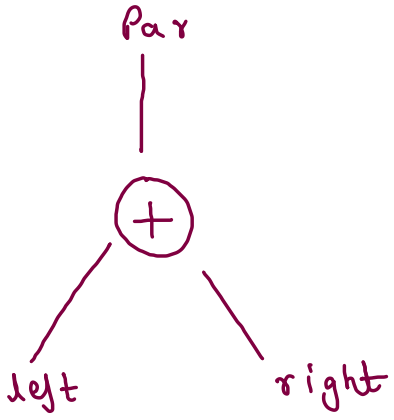


968. Binary Tree Cameras

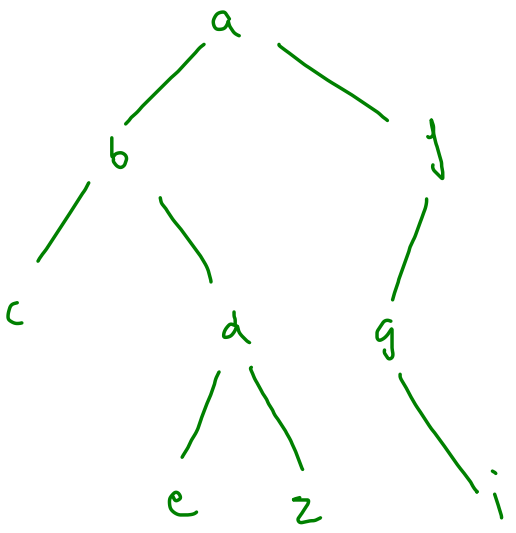
Hard 2560 33 Add to List Share

You are given the `root` of a binary tree. We install cameras on the tree nodes where each camera at a node can monitor its parent, itself, and its immediate children.

Return *the minimum number of cameras needed to monitor all nodes of the tree.*



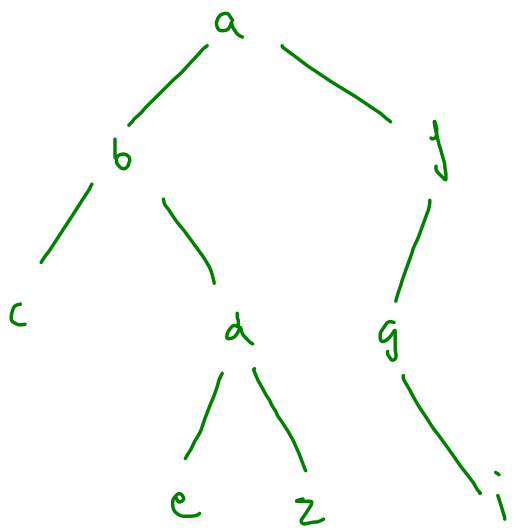
0 → needs coverage
1 → camera
2 → covered



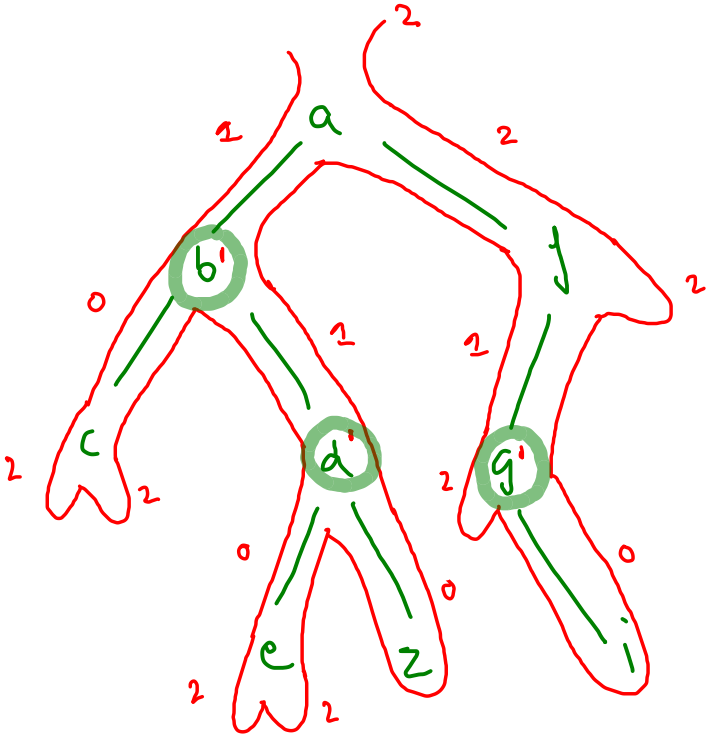
0 → needs coverage

1 → camera

2 → covered




```
int lci = helper(node.left)
int rci = helper(node.right);

if (lci == 0 || rci == 0) {
    camt++;
    return 1;
}
else if (lci == 1 || rci == 1) {
    return 2;
}
else {
    return 0;
}
```



cam = 0 + 1 + 1 + 1

```java
public int helper(TreeNode node) {
    if(node == null) {
        return 2;
    }

    int lci = helper(node.left);
    int rci = helper(node.right);

    if(lci == 0 || rci == 0) {
        cam++; //put a camera on node
        return 1;
    }
    else if(lci == 1 || rci == 1) {
        return 2; //node is covered
    }
    else {
        return 0;
    }
}
```