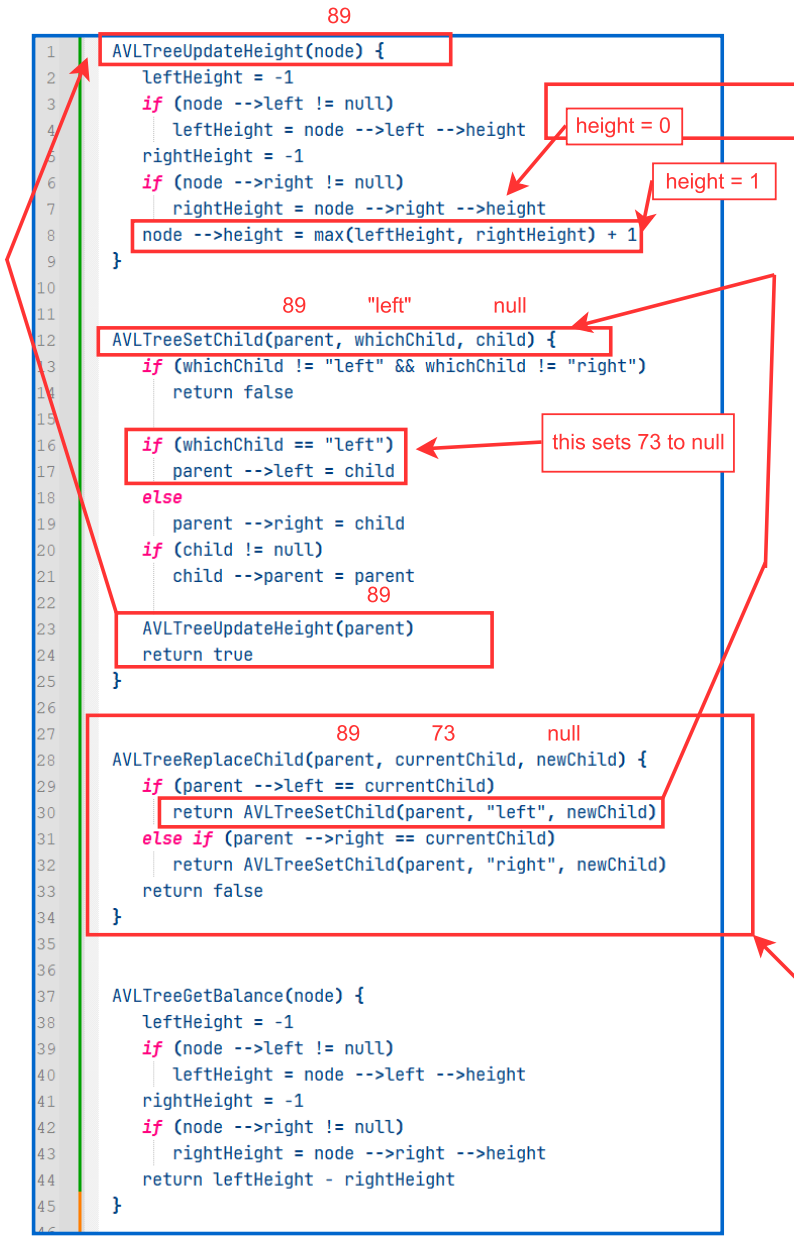

89
AVLTreeUpdateHeight(node) {
leftHeight = -1
if (node -->left != null)
leftHeight = node -->left -->height
rightHeight = -1
if (node -->right != null)
rightHeight = node -->right -->height
node -->height = max(leftHeight, rightHeight) + 1
}
height = 0
height = 1
89 "left" null
AVLTreeSetChild(parent, whichChild, child) {
if (whichChild != "left" && whichChild != "right")
return false
if (whichChild == "left")
parent -->left = child
this sets 73 to null
else
parent -->right = child
if (child != null)
child -->parent = parent
89
AVLTreeUpdateHeight(parent)
return true
}
89 73 null
AVLTreeReplaceChild(parent, currentChild, newChild) {
if (parent -->left == currentChild)
return AVLTreeSetChild(parent, "left", newChild)
else if (parent -->right == currentChild)
return AVLTreeSetChild(parent, "right", newChild)
return false
}
AVLTreeGetBalance(node) {
leftHeight = -1
if (node -->left != null)
leftHeight = node -->left -->height
rightHeight = -1
if (node -->right != null)
rightHeight = node -->right -->height
return leftHeight - rightHeight
}

```
AVLTreeRemoveNode(tree, node) {
    if (node == null) {
        return false
    }

    // Parent needed for rebalancing
    parent = node-->parent

    // Case 1: Internal node with 2 children
    if (node-->left != null && node-->right != null) {
        // Find successor
        succNode = node-->right
        while (succNode-->left != null) {
            succNode = succNode-->left
        }

        // Copy the key from the successor node
        node-->key = succNode-->key

        // Recursively remove successor
        AVLTreeRemoveNode(tree, succNode)

        // Nothing left to do since the recursive call will have rebalanced
        return true
    }

    // Case 2: Root node (with 1 or 0 children)
    else if (node == tree-->root) {
        if (node-->left != null) {
            tree-->root = node-->left
        }
        else {
            tree-->root = node-->right
        }
        if (tree-->root != null) {
            tree-->root-->parent = null
        }
        return true
    }

    // Case 3: Internal with left child only
    else if (node-->left != null) {
        AVLTreeReplaceChild(parent, node, node-->left)
    }
    // Case 4: Internal with right child only OR leaf
    else {
        AVLTreeReplaceChild(parent, node, node-->right)
    }

    // node is gone. Anything that was below node that has persisted is already correctly
    // balanced, but ancestors of node may need rebalancing.
    node = parent
    while (node != null) {
        AVLTreeRebalance(tree, node)
        node = node-->parent
    }
    return true
}
```

```
AVLTreeRebalance(tree, node) {
    AVLTreeUpdateHeight(node)
    if (AVLTreeGetBalance(node) == -2) {
        if (AVLTreeGetBalance(node-->right) == 1) {
            // Double rotation case.
            AVLTreeRotateRight(tree, node-->right)
        }
        return AVLTreeRotateLeft(tree, node)
    }
    else if (AVLTreeGetBalance(node) == 2) {
        if (AVLTreeGetBalance(node-->left) == -1) {
            // Double rotation case.
            AVLTreeRotateLeft(tree, node-->left)
        }
        return AVLTreeRotateRight(tree, node)
    }
    return node
}
```

70

```
AVLTreeRemoveNode(tree, node) {
    if (node == null) {
        return false
    }

    // Parent needed for rebalancing
    parent = node-->parent

    // Case 1: Internal node with 2 children
    if (node-->left != null && node-->right != null) {
        // Find successor
        succNode = node-->right
        while (succNode-->left != null) {
            succNode = succNode-->left
        }

        // Copy the key from the successor node
        node-->key = succNode-->key

        // Recursively remove successor
        AVLTreeRemoveNode(tree, succNode)

        // Nothing left to do since the recursive call will have rebalanced
        return true
    }

    // Case 2: Root node (with 1 or 0 children)
    else if (node == tree-->root) {
        if (node-->left != null) {
            tree-->root = node-->left
        }
        else {
            tree-->root = node-->right
        }
        if (tree-->root != null) {
            tree-->root-->parent = null
        }
        return true
    }

    // Case 3: Internal with left child only
    else if (node-->left != null) {
        AVLTreeReplaceChild(parent, node, node-->left)
    }
    // Case 4: Internal with right child only OR leaf
    else {
        AVLTreeReplaceChild(parent, node, node-->right)
    }

    // node is gone. Anything that was below node that has persisted is already correctly
    // balanced, but ancestors of node may need rebalancing.
    node = parent
    while (node != null) {
        AVLTreeRebalance(tree, node)
        node = node-->parent
    }
    return true
}
```

null

node : 70
node --> right : 89
node --> left : 31

89

73

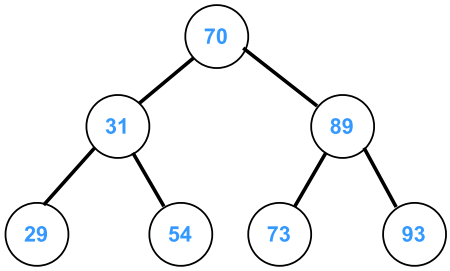
succNode : 89
node --> right : 89
if succNode --> left != null then

73

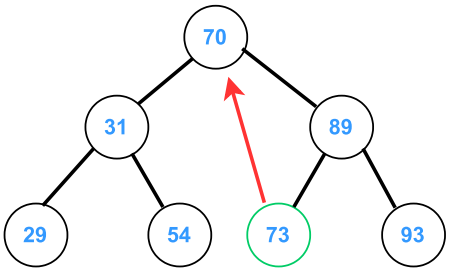
73

b

Remove 70



Find 73



Find 73