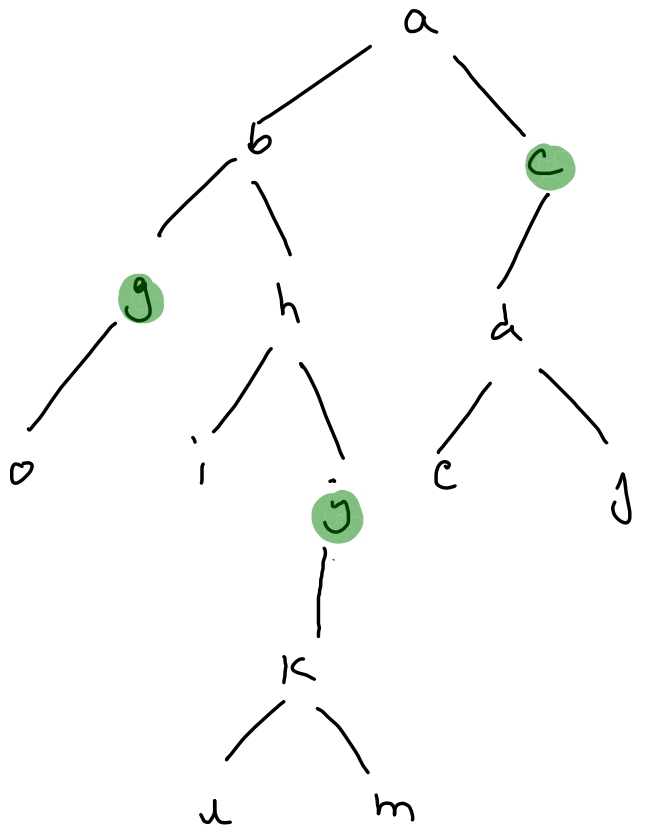
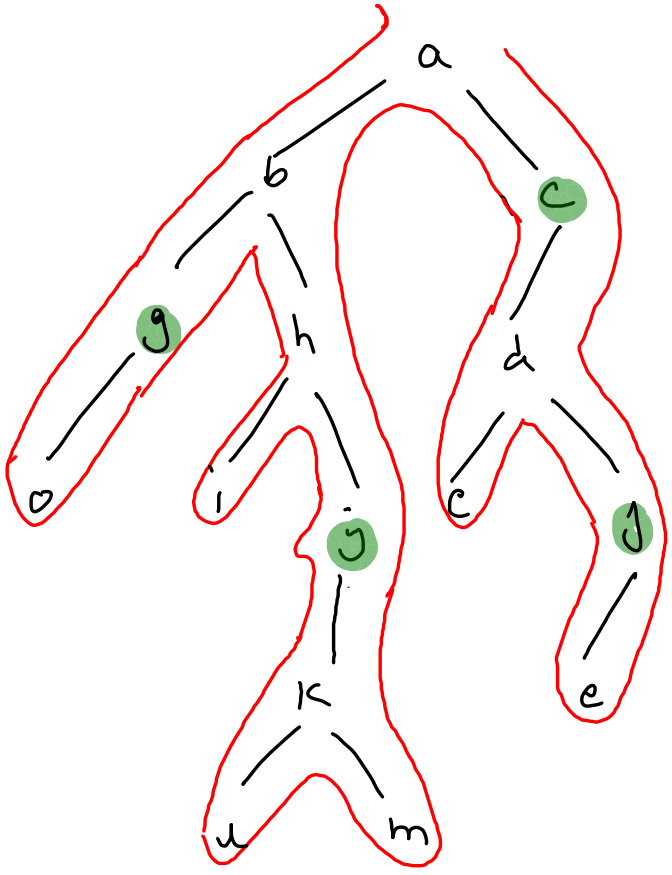
single child parent node



single child parent

-> node.left == null &&
node.right != null

-> node.left != null &&
node.right == null

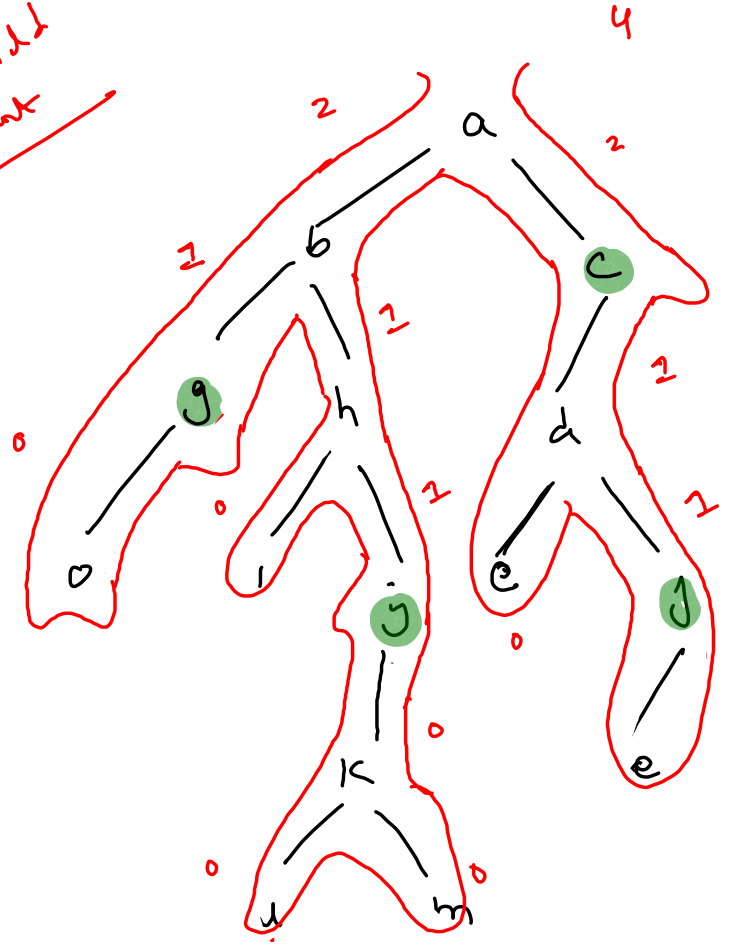



```java
public static void exactlyOneChild(TreeNode node,ArrayList<Integer>ans) {
    if(node == null) {
        return;
    }

    if((node.left != null && node.right == null) || (node.left == null && node.right != null)) {
        //single child parent node
        ans.add(node.val);
    }

    exactlyOneChild(node.left,ans);
    exactlyOneChild(node.right,ans);
}
public static ArrayList<Integer> exactlyOneChild(TreeNode root) {
```

ans. g, j, c, j

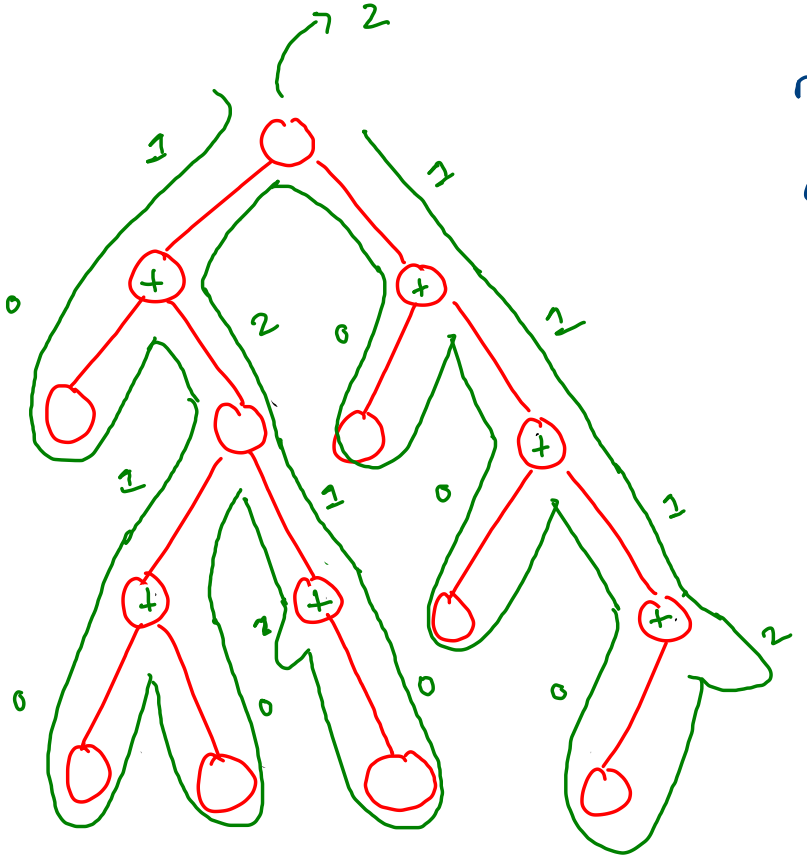
ul

Count of all single child parent




```java
public static int countExactlyOneChild(TreeNode node) {
    if(node == null) {
        return 0;
    }

    int lc = countExactlyOneChild(node.left);
    int rc = countExactlyOneChild(node.right);

    int ans = lc + rc;

    if((node.left != null && node.right == null)
    || (node.left == null && node.right != null)) {
        //node is a single child parent
        ans++;
    }

    return ans;
}
```

Cameras

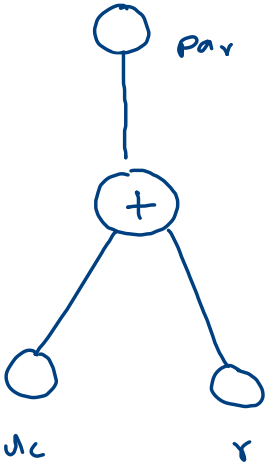


need -> 0

Camera -> 1

Covered -> 2

Min Cameras = ~~0~~ ~~1~~ ~~3~~ ~~2~~ 6 ~~4~~ ~~5~~

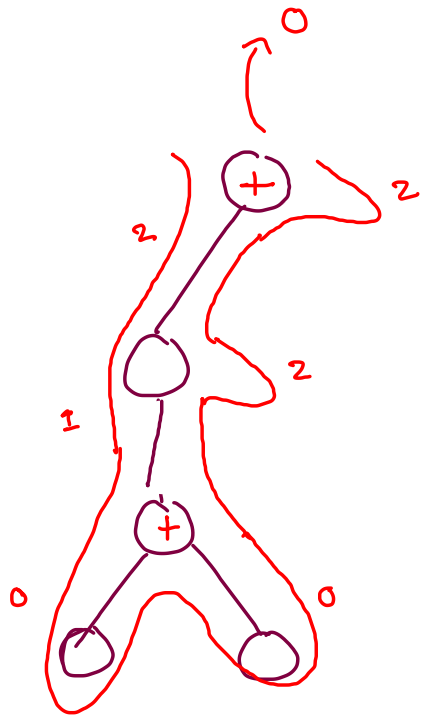


leaf nodes -> need

need -> 0

camera -> 1

Covered -> 2

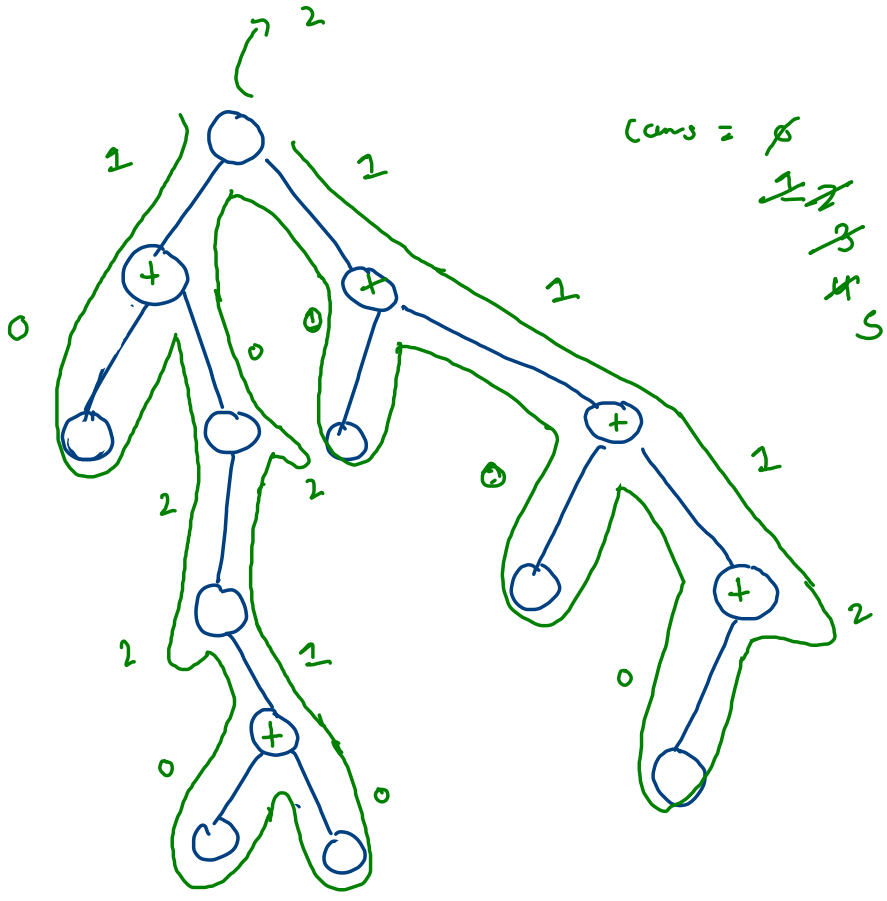


minCam

= ~~1~~

2

```
int lc = cameras(node.left);
int rc = cameras(node.right);

if( lc == 0 || rc == 0) {
        const t;
        return 1;
}
if ( lc == 1 || rc == 1 ) {
        return 2;
}
return 0;
```



need -> 0

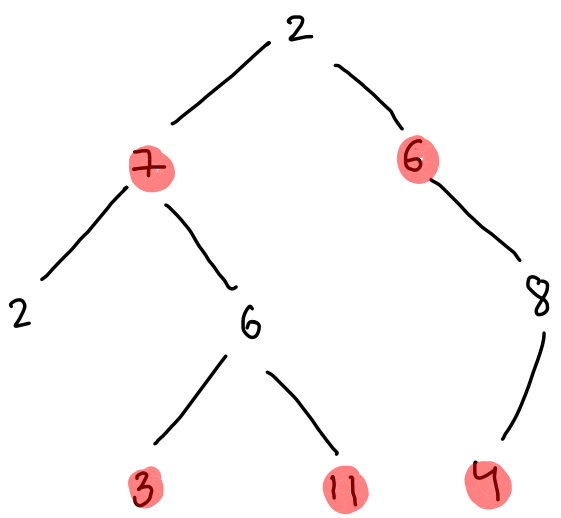
camera -> 1

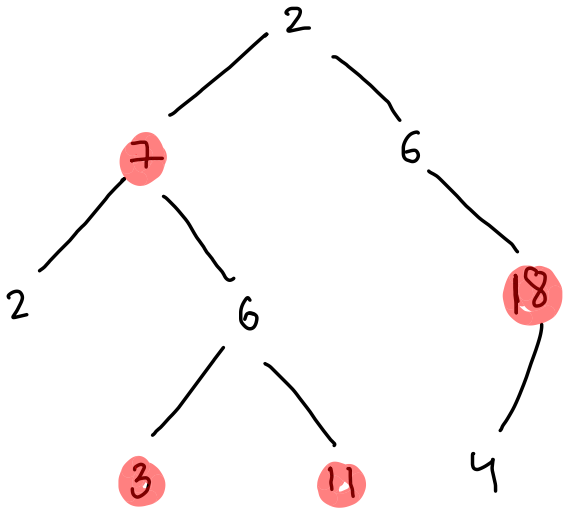
Covered -> 2

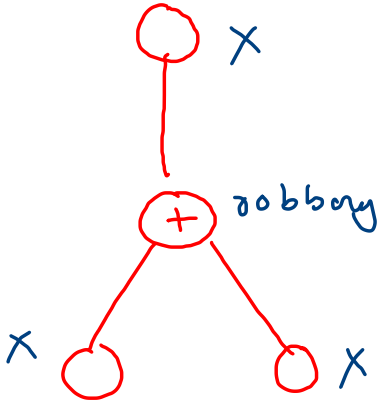
node -> null, return 2.

```
int lc = cameras(node.left);
int rc = cameras(node.right);

if(lc == 0 || rc == 0) {
    const++;
    return 1;
}
if(lc == 1 || rc == 1) {
    return 2;
}
return 0;
```

house robber



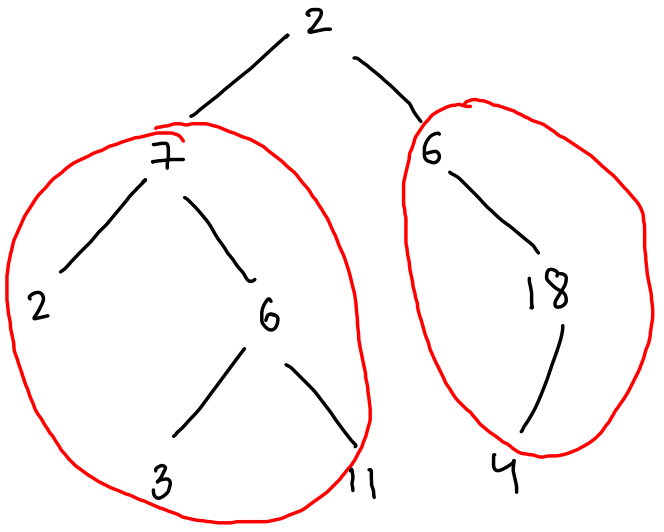




dc

p1 -> dc include

p2 -> dc exclude



rc

q1 -> rc include

q2 -> rc exclude

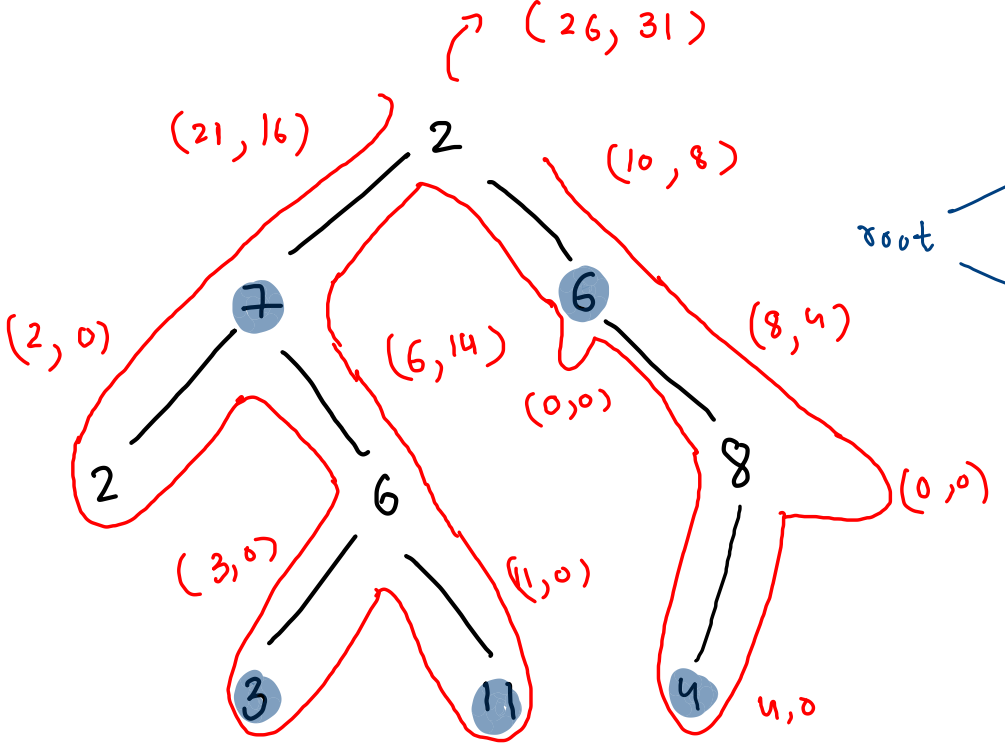
include -> included in robbery

exclude -> exclude from robbery.

root

include : ans -> p2 + q2 + root.val

exclude : ans -> max(p1, p2) + max(q1, q2)



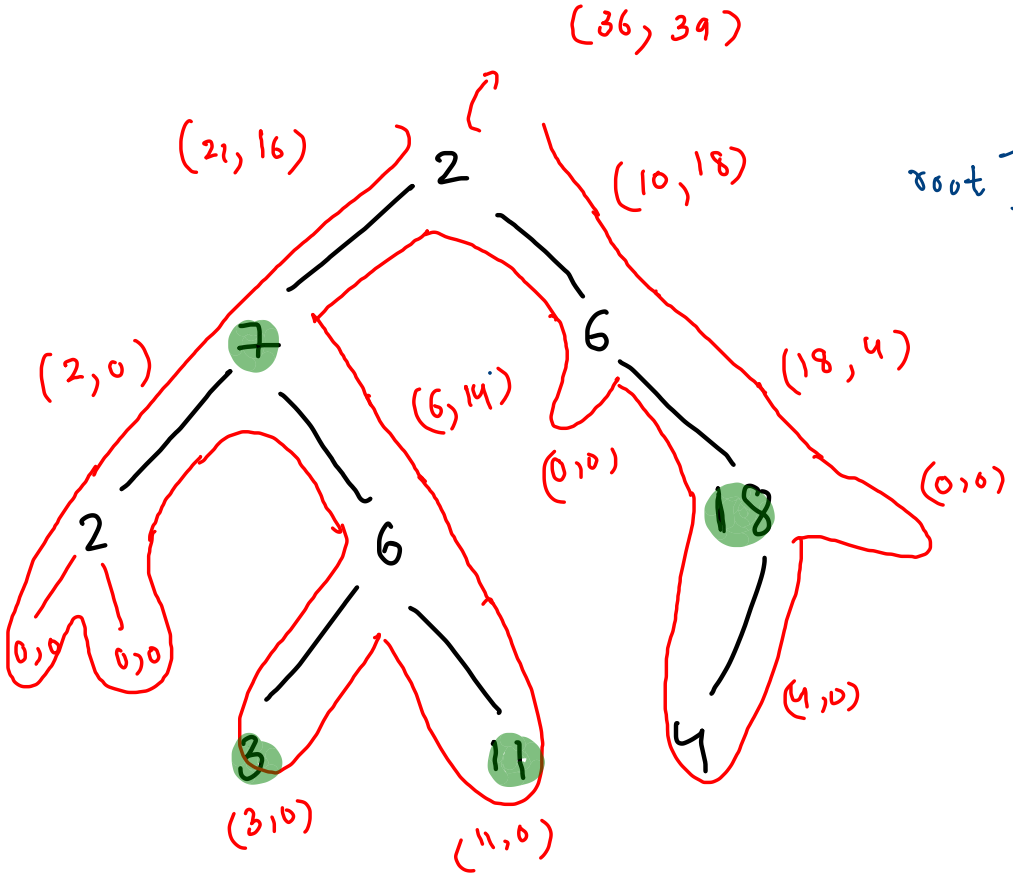
root

include : ans -> p2 + q2 + root.val

exclude : ans -> max (p1, p2) + max (q1, q2)

lc -> (p1, p2)

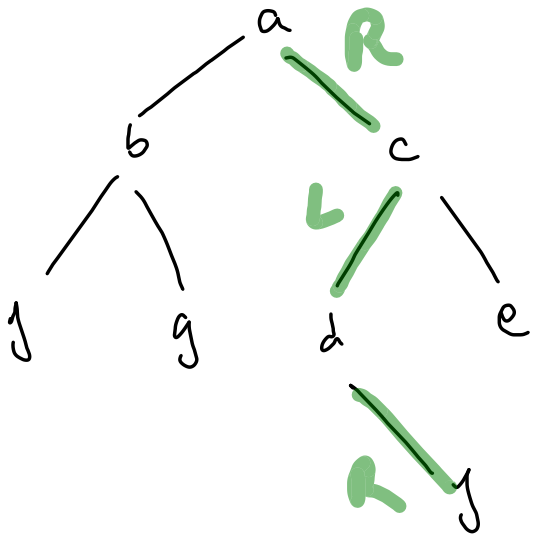
rc -> (q1, q2)

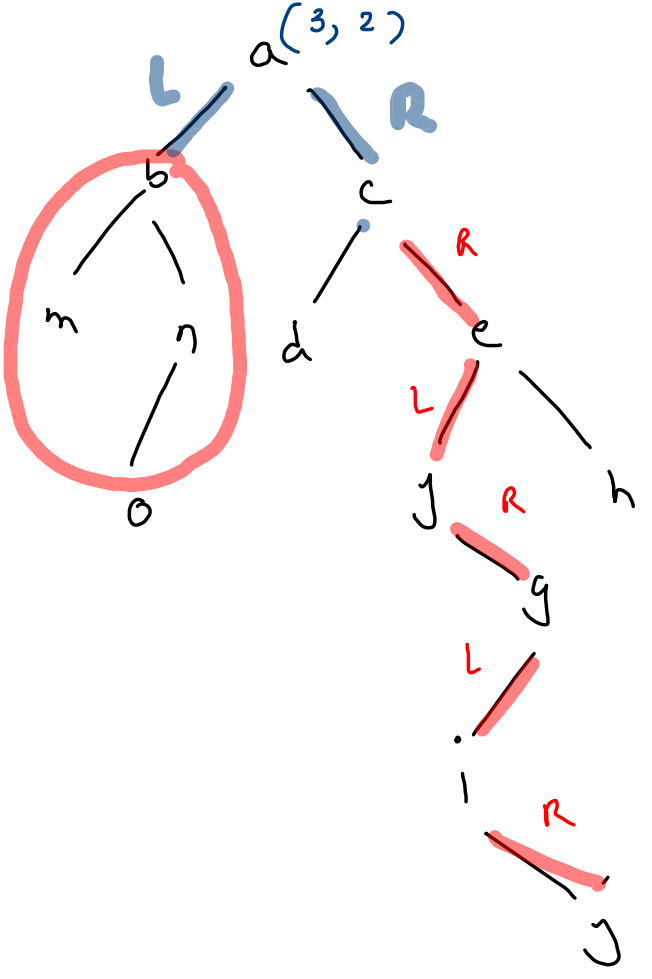


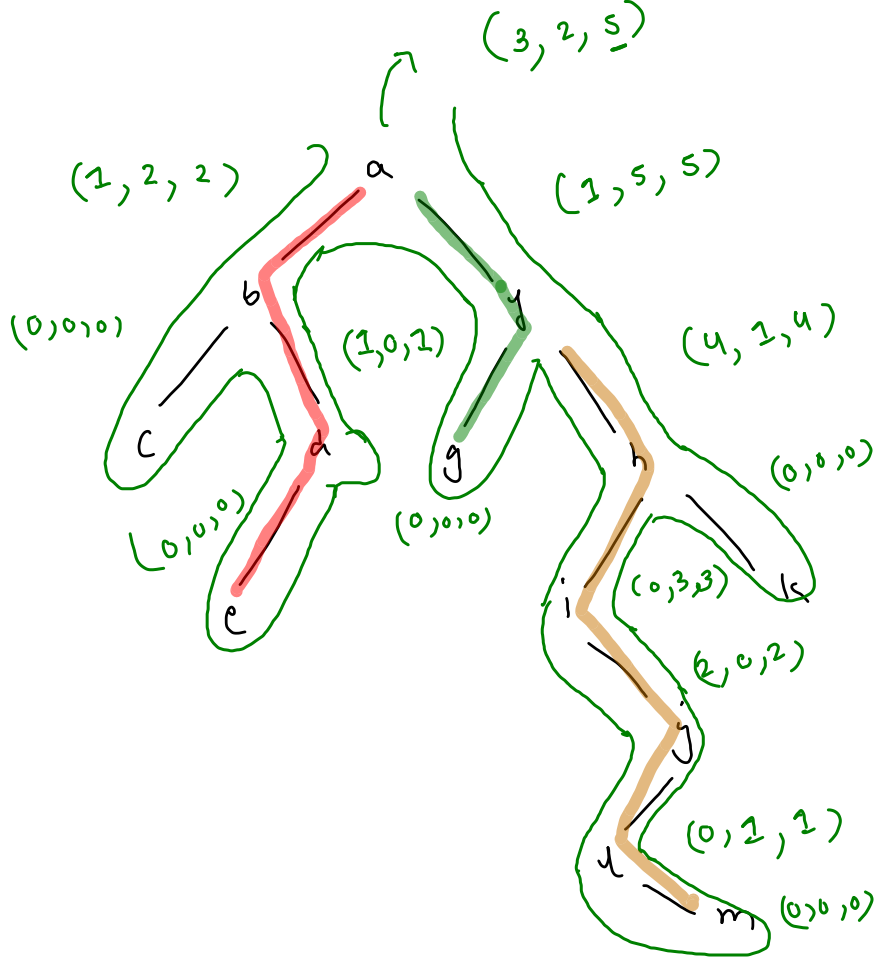
root → include : ans → p2 + q2 + root.val

root → exclude : ans → max(p1, p2) + max(q1, q2)

root → null (0, 0)


a
R
b
c
L
f
g
d
e
R
y


a
(3, 2)
L
R
b
c
m
n
d
R
e
L
o
f
h
R
g
L
i
R
y

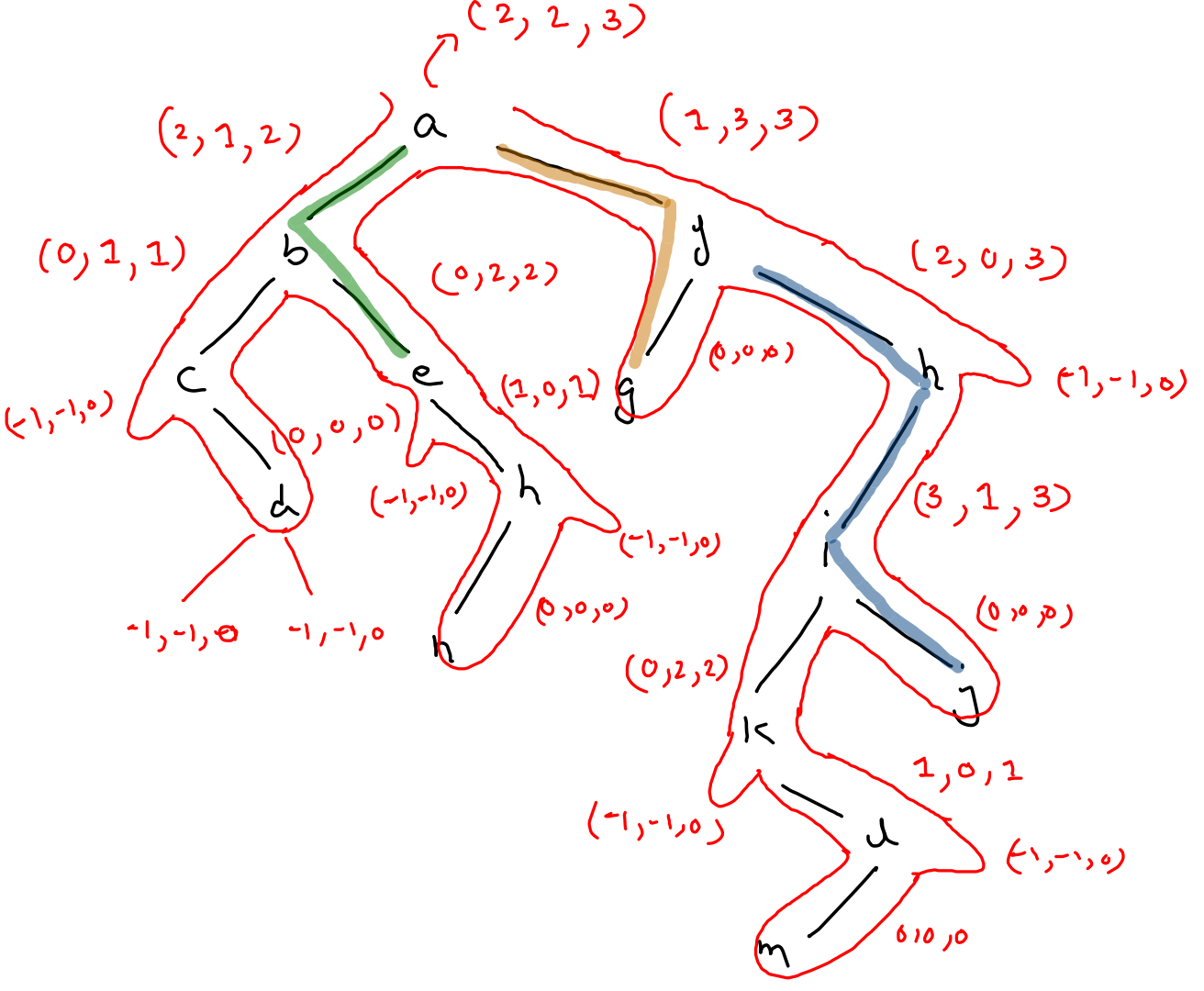


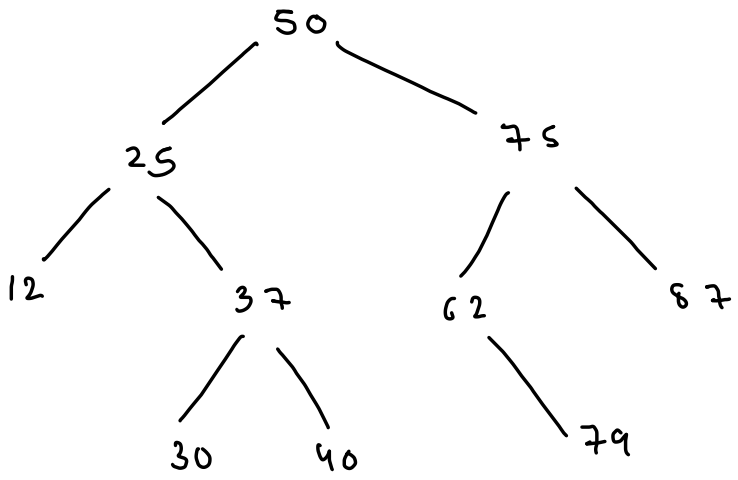

```
Pair {
    int lzzp;
    int rzzp;      } node included

    int mzzp;
}
```

root { dcp , rcp.

```
root.dzzp = dcp.rzzp + 1;
root.rzzp = rcp.dzzp + 1;
root.mzzp = max(dzzp, rzzp,
                dcp.mzzp,
                rcp.mzzp)
```

```java
public Pair helper(TreeNode node) {
    if(node == null) {
        return new Pair(-1,-1,0);
    }

    Pair lcp = helper(node.left);
    Pair rcp = helper(node.right);

    Pair np = new Pair();

    np.lzzp = lcp.rzzp + 1;
    np.rzzp = rcp.lzzp + 1;


    np.mzzp = max(np.lzzp,np.rzzp,lcp.mzzp,rcp.mzzp);

    return np;
}
```



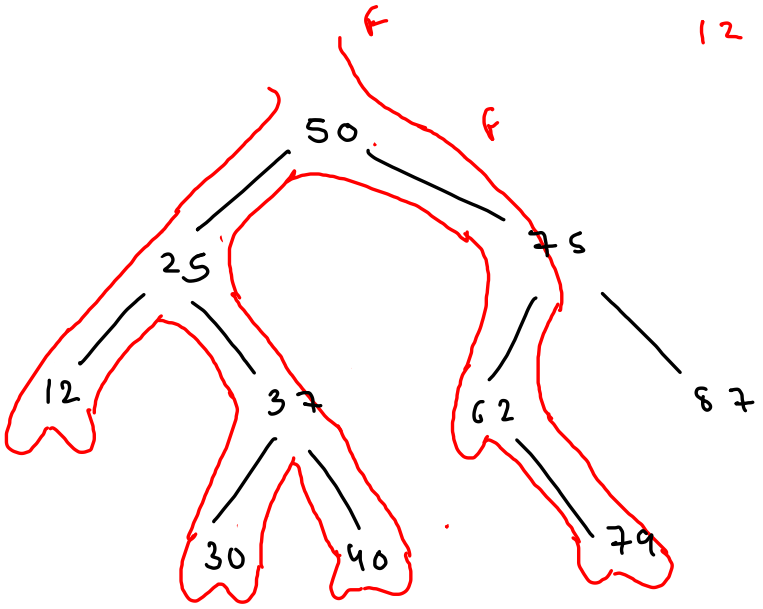


inorder:

12 25 30 37 40 50 62 79 75 87

∀ nodes in left < node.val < ∀ nodes in right subtree

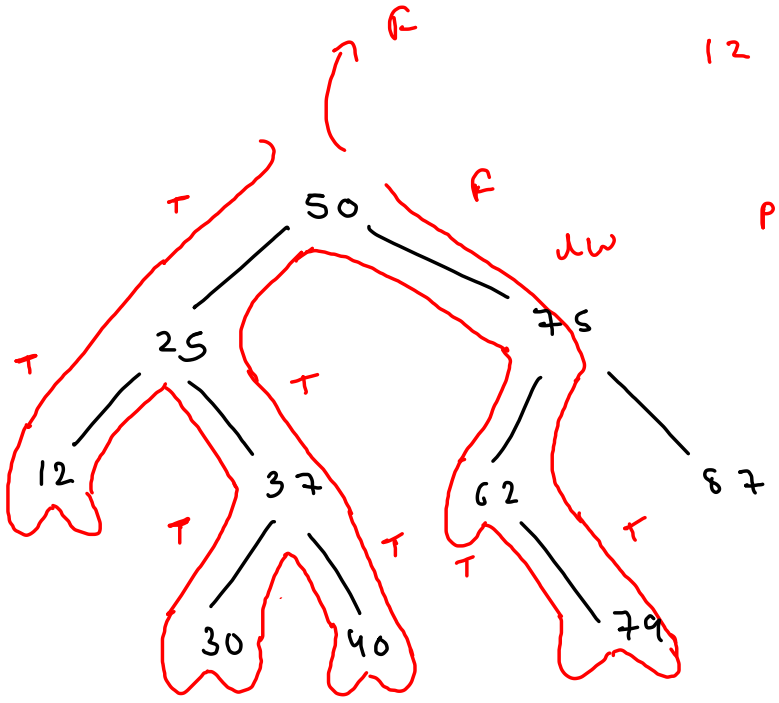
subtree



12 25 30 37 40 50 62 79 75 87

prev = ~~∅~~ ~~12~~ ~~25~~ ~~30~~ ~~37~~ ~~40~~ ~~50~~ ~~62~~ 79

```
left call(node.left);
if (prev != null && prev.val > node.val) {
        return false;
}
prev = node;
right call(node.right);
```



12 25 30 37 40 50 62 79 75 87

p = ~~9~~ ~~12~~ ~~25~~ ~~30~~ ~~37~~ ~~40~~ ~~50~~ ~~62~~ 79

```java
public static boolean helper(TreeNode node) {
    if(node == null) {
        return true;
    }

    boolean lans = helper(node.left);

    //work
    if(prev != null && prev.val > node.val) {
        //inorder is not sorted
        return false;
    }

    prev = node;

    boolean rans = helper(node.right);

    return lans && rans;
}
```