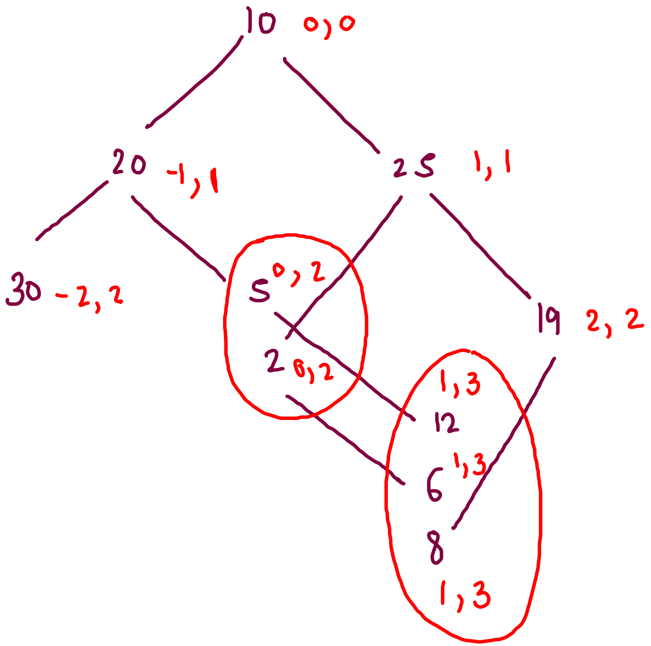
Vertical Order Traversal of a Binary Tree



30

20

10 2 5

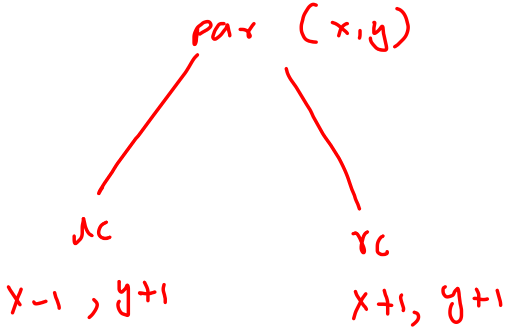
25 6 8 12

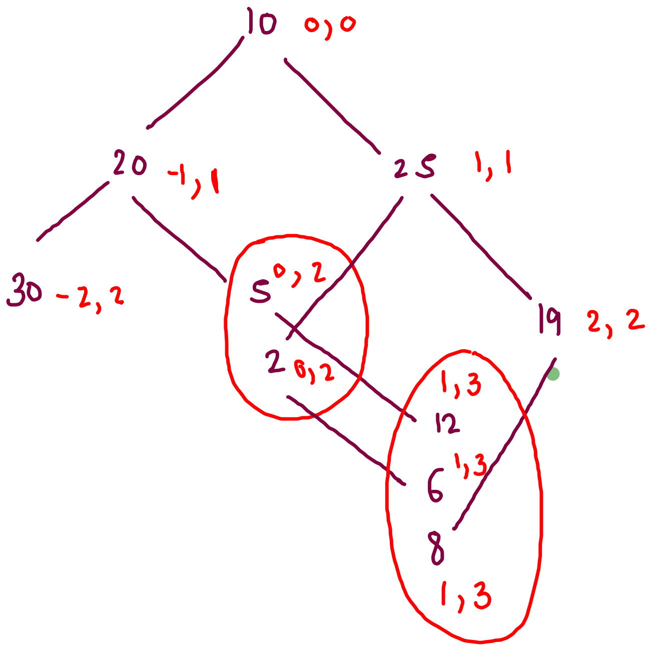
19

x, y

x -> vertical line no.

y -> horizontal line no.





Pair :

node

x

y

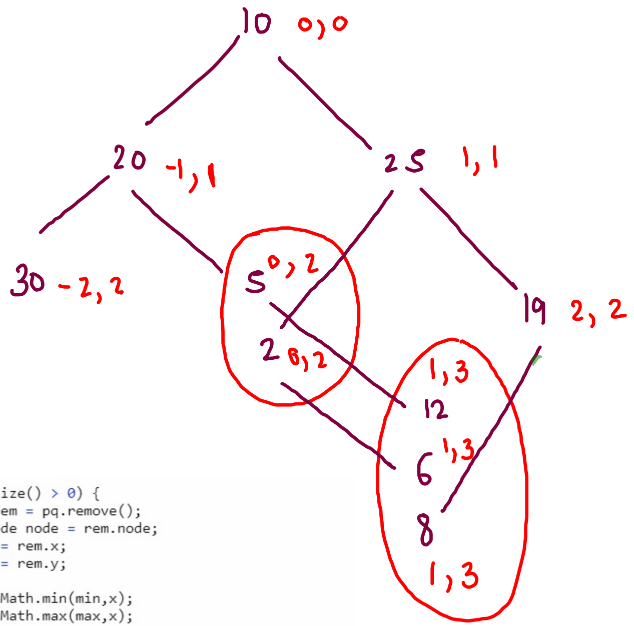
compare {

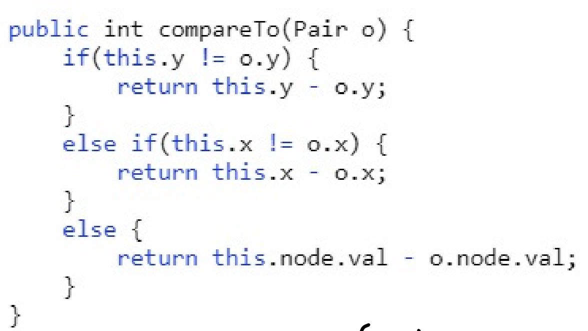
1st priority y

2nd priority x

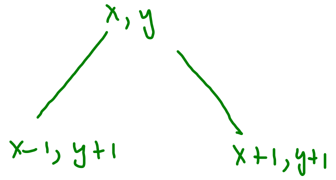
3rd priority value

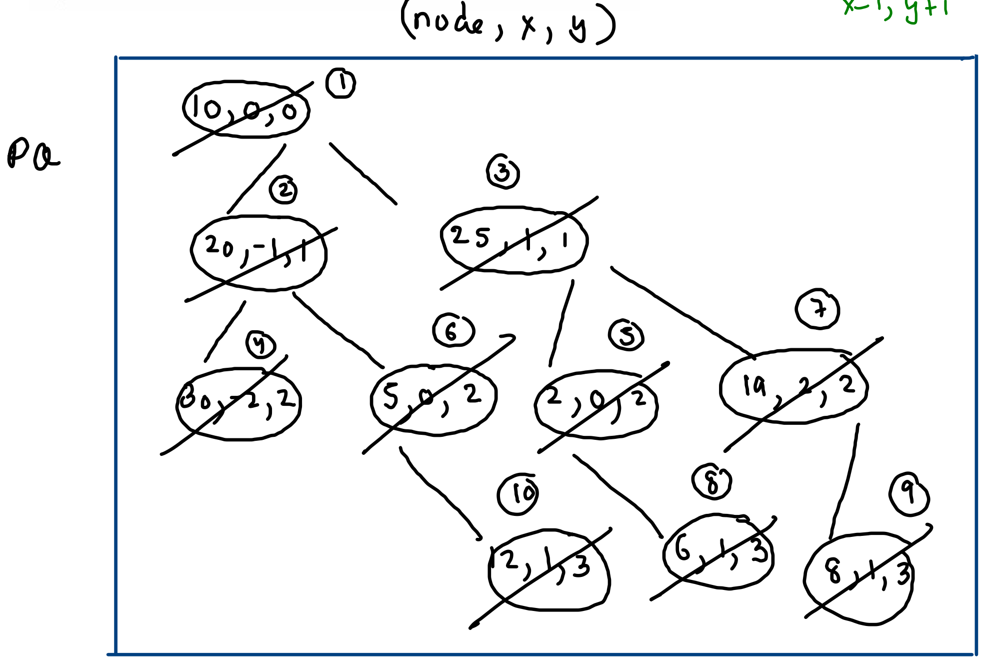
}

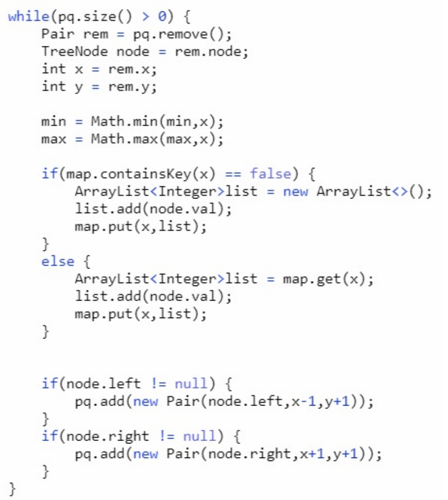




```
public int compareTo(Pair o) {
    if(this.y != o.y) {
        return this.y - o.y;
    }
    else if(this.x != o.x) {
        return this.x - o.x;
    }
    else {
        return this.node.val - o.node.val;
    }
}
```







```
while(pq.size() > 0) {
    Pair rem = pq.remove();
    TreeNode node = rem.node;
    int x = rem.x;
    int y = rem.y;

    min = Math.min(min,x);
    max = Math.max(max,x);

    if(map.containsKey(x) == false) {
        ArrayList<Integer>list = new ArrayList<>();
        list.add(node.val);
        map.put(x,list);
    }
    else {
        ArrayList<Integer>list = map.get(x);
        list.add(node.val);
        map.put(x,list);
    }


    if(node.left != null) {
        pq.add(new Pair(node.left,x-1,y+1));
    }
    if(node.right != null) {
        pq.add(new Pair(node.right,x+1,y+1));
    }
}
```

min = -2

max = 2

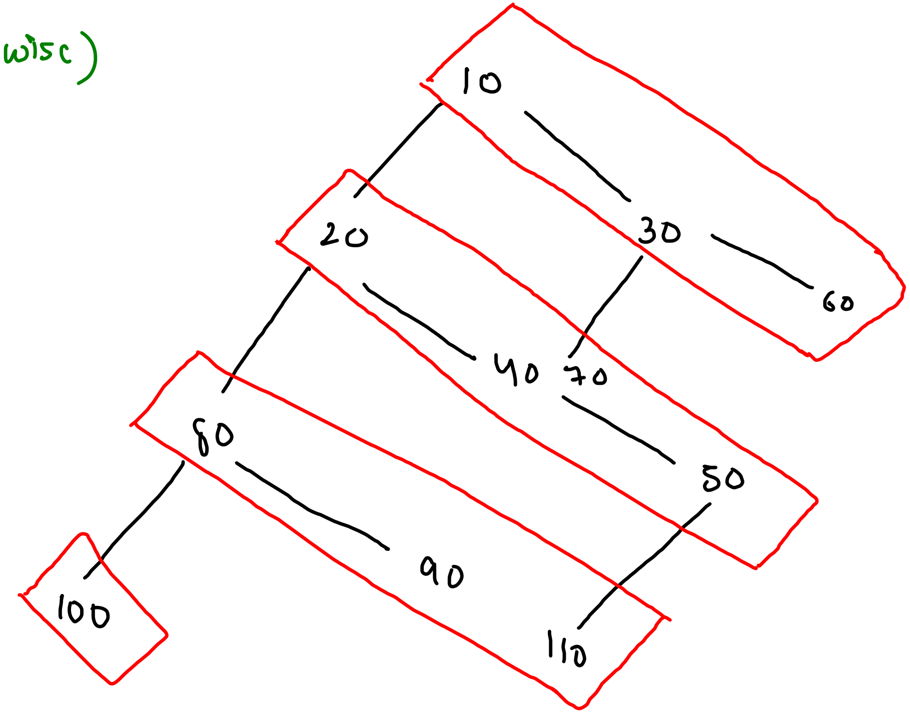
0 -> 10, 2, 5
-1 -> 20
1 -> 25, 6, 8, 12
-2 -> 30
2 -> 19

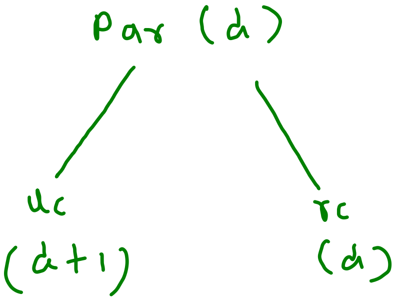
map

[[30], [20], [10, 2, 5], [25, 6, 8, 12], [19]]

Diagonal Order Of A Binarytree

(clockwise)





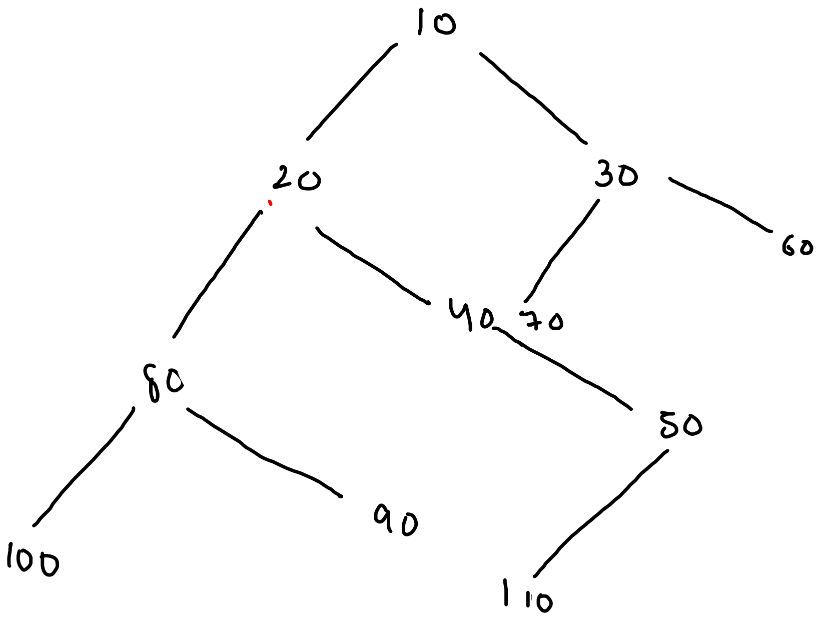
AL

0 -> 10 30 60

1 -> 20 40 70 50

2 -> 80 90 110

3 -> 100



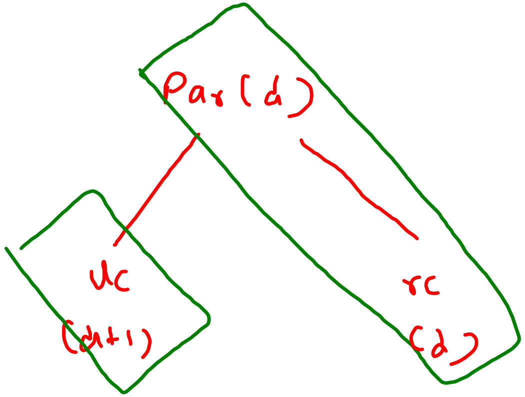
do

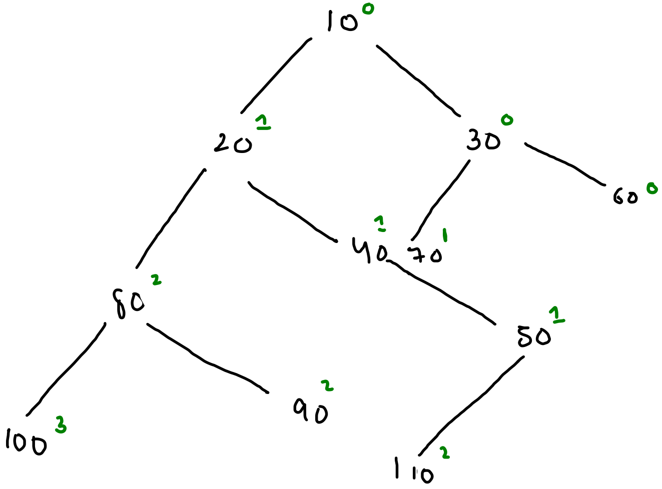
0 -> 10, 30, 60

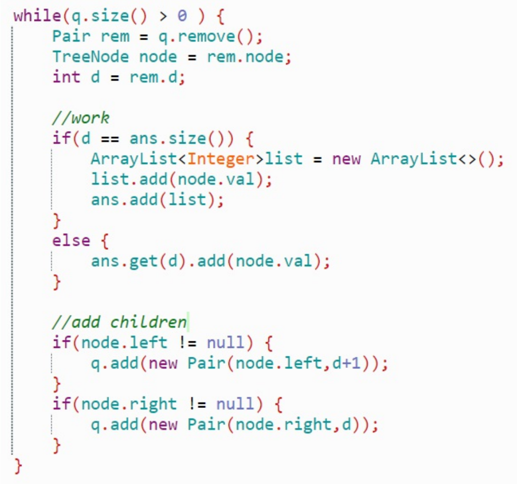
1 -> 20, 40, 70, 50

2 -> 80, 90, 110

3 -> 100







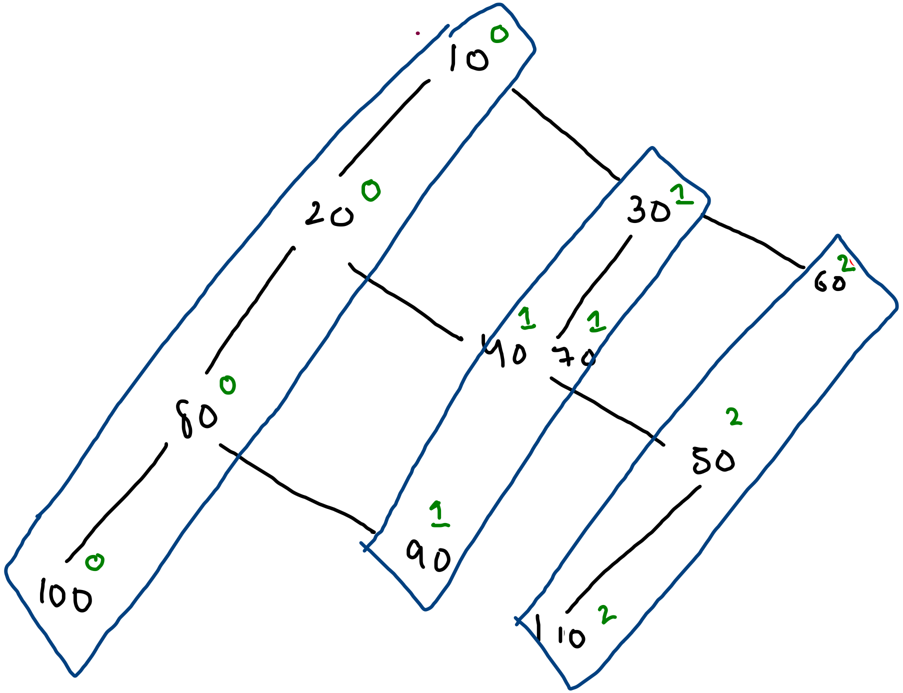
```
while(q.size() > 0 ) {
    Pair rem = q.remove();
    TreeNode node = rem.node;
    int d = rem.d;

    //work
    if(d == ans.size()) {
        ArrayList<Integer>list = new ArrayList<>();
        list.add(node.val);
        ans.add(list);
    }
    else {
        ans.get(d).add(node.val);
    }

    //add children
    if(node.left != null) {
        q.add(new Pair(node.left,d+1));
    }
    if(node.right != null) {
        q.add(new Pair(node.right,d));
    }
}
```

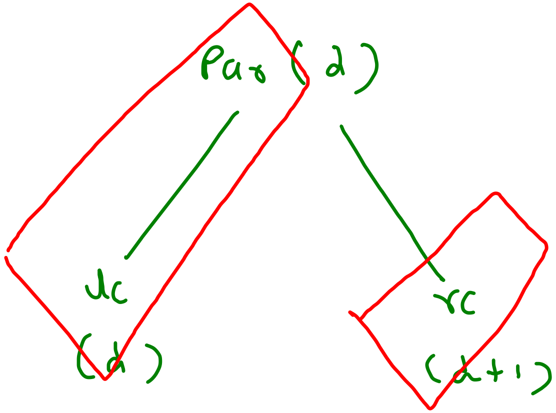
ans [[10, 30, 60], [20, 40, 70, 50]
0 1

[80, 90, 110], [100]]
2 3

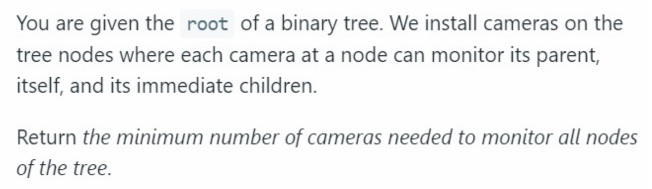
~~10,0~~	~~20,1~~	~~30,0~~	~~80,2~~	~~40,1~~	~~70,1~~	~~60,0~~	~~100,3~~	~~90,2~~	~~50,1~~	~~110,2~~

Anticlockwise :



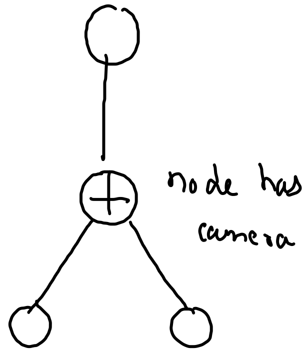


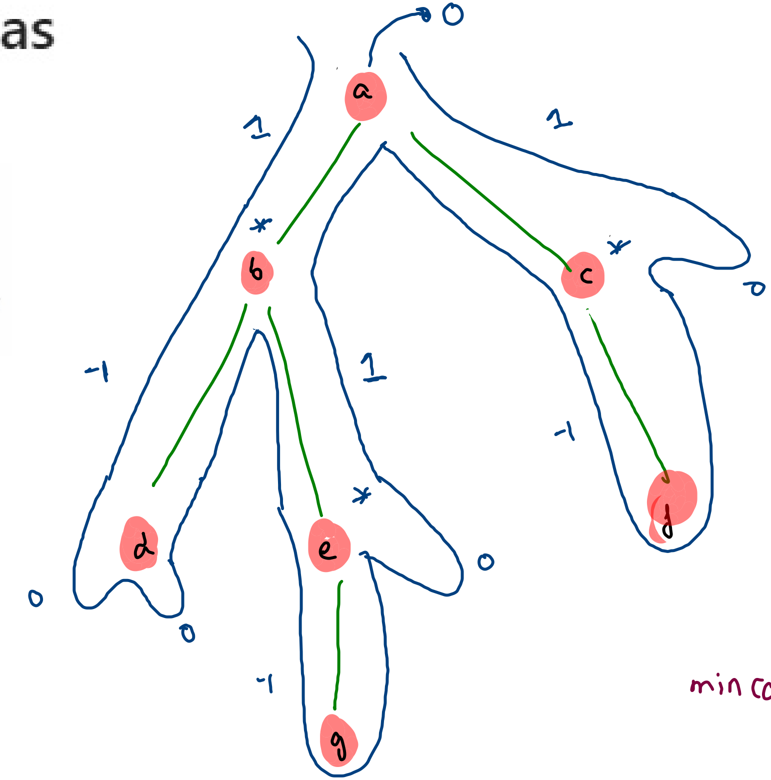
968. Binary Tree Cameras



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.*





Situations

(i) camera at it. → 1

(ii) covered by someone's camera → 0

(iii) need coverage → -1

null → 0 (covered)

leaf → -1 (need)

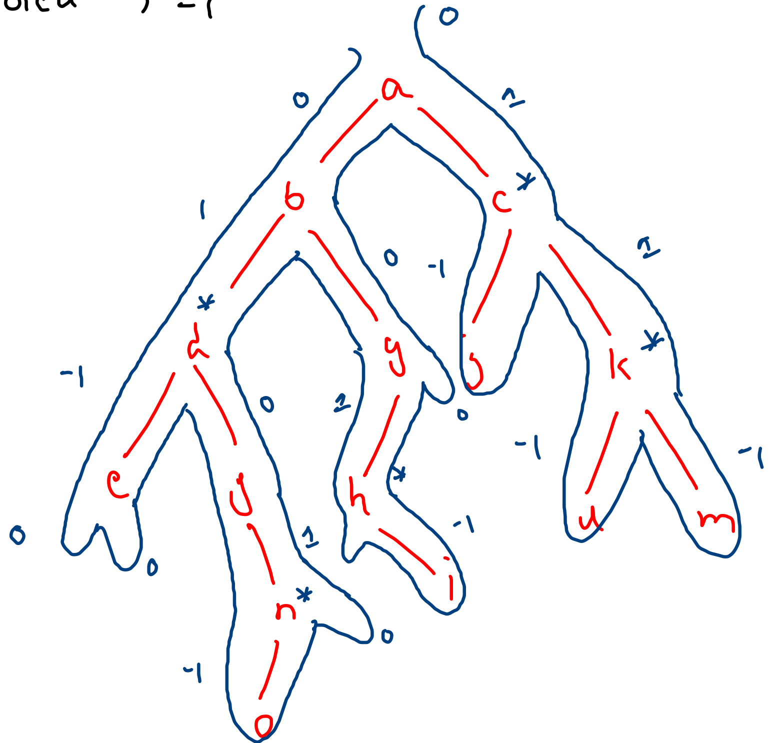
min cam = 3

Node situation

(i) camera at it → 1

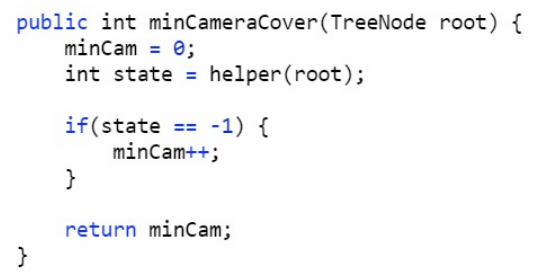
(ii) covered by someone's cam → 0

(iii) not covered → -1

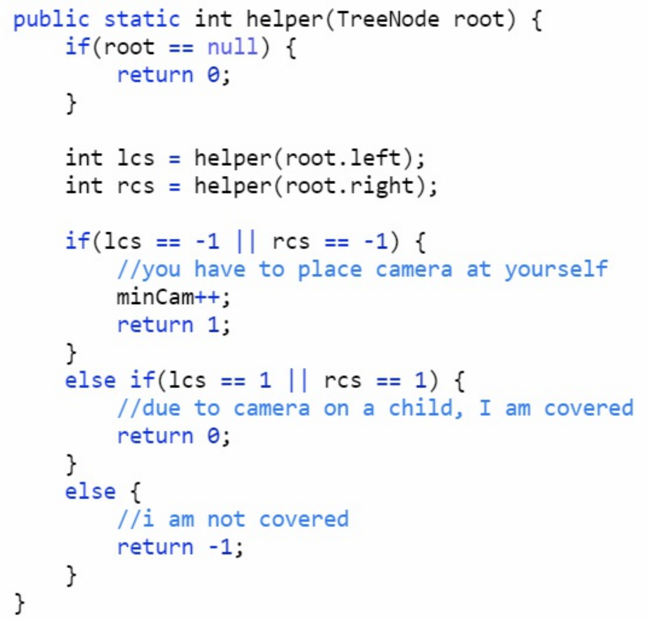


```
if (lc == -1    || rc == -1) {
        mincam++;
        return 1;
}
else if (lc == 1   || rc == 1) {
        return 0;
}
else {
        return -1;
}
```

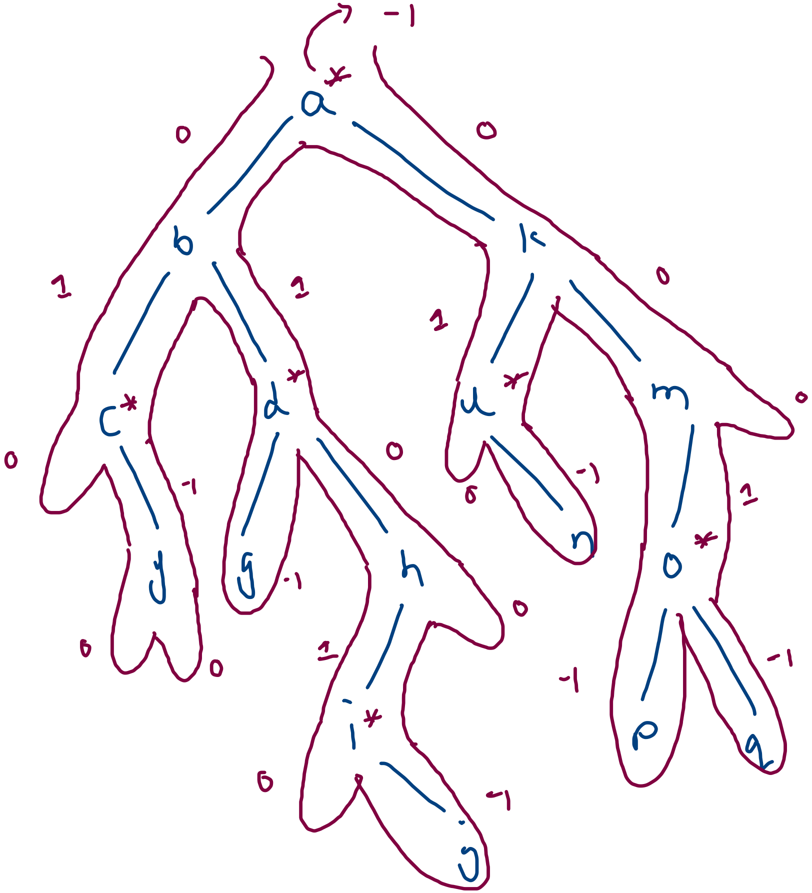
min cam = 4



```
public int minCameraCover(TreeNode root) {
    minCam = 0;
    int state = helper(root);

    if(state == -1) {
        minCam++;
    }

    return minCam;
}
```



```
public static int helper(TreeNode root) {
    if(root == null) {
        return 0;
    }

    int lcs = helper(root.left);
    int rcs = helper(root.right);

    if(lcs == -1 || rcs == -1) {
        //you have to place camera at yourself
        minCam++;
        return 1;
    }
    else if(lcs == 1 || rcs == 1) {
        //due to camera on a child, I am covered
        return 0;
    }
    else {
        //i am not covered
        return -1;
    }
}
```



mincam = ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ 6

-1 -> not covered (need)

0 -> covered

1 -> camera