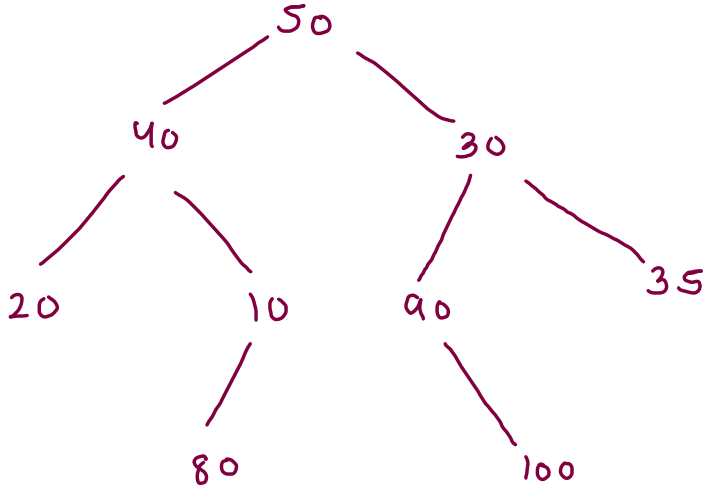
Path To Leaf From Root In Range

range lo : 112
hi : 250



root to leaf

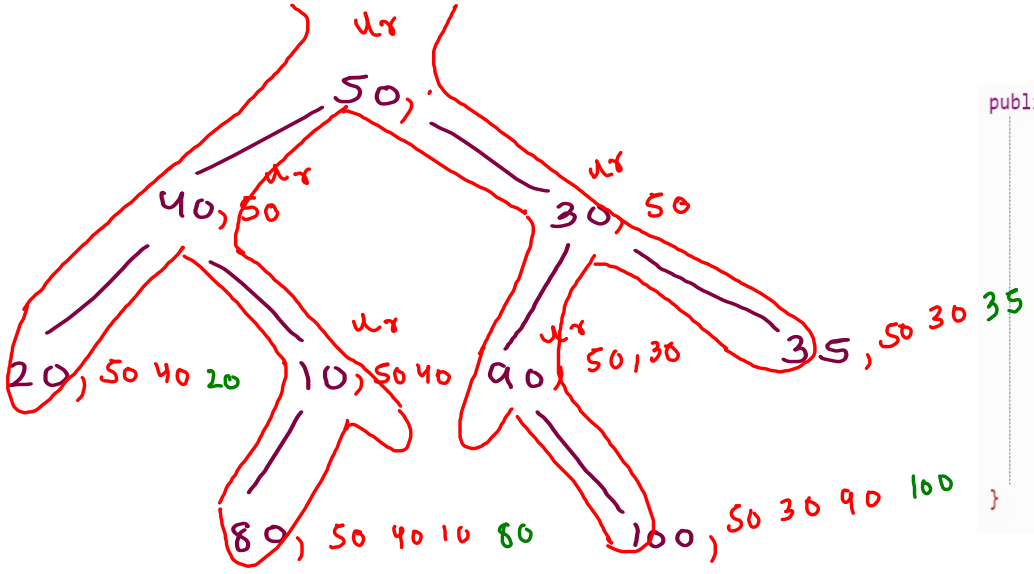
50 40 20 → 110

50 40 10 80 → 180

50 30 90 100 → 270

50 30 35 → 115

lo: 112 hi: 250



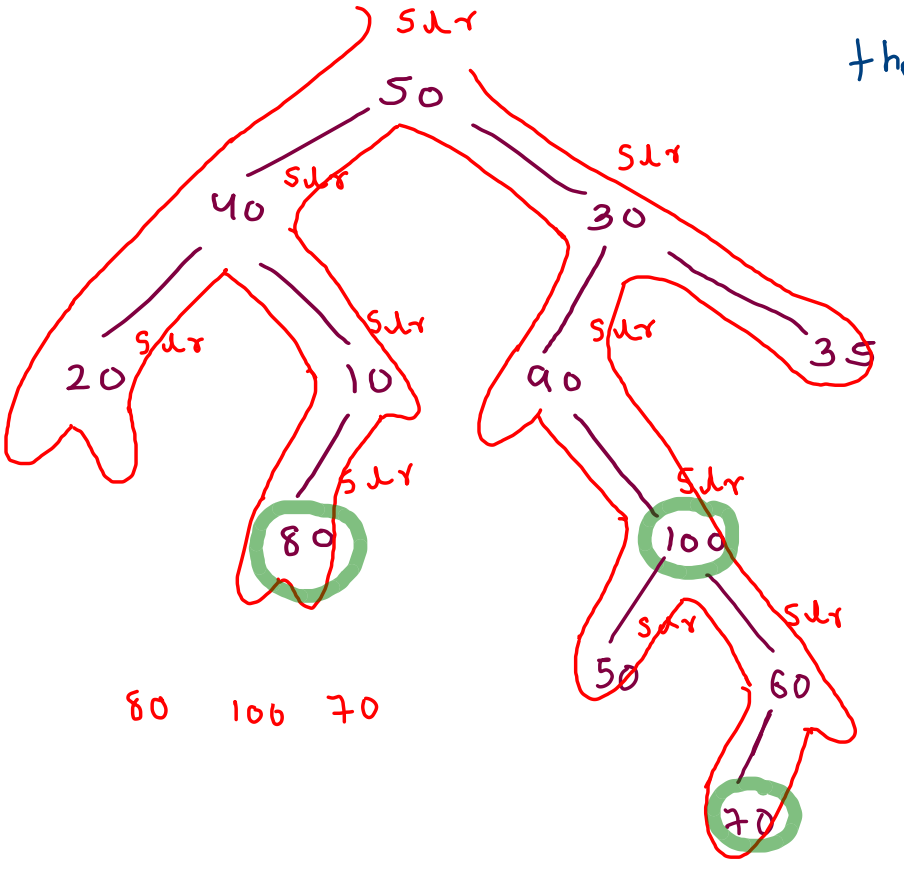

```java
public static void pathToLeafFromRoot(Node node, String path, int sum, int lo, int hi){
    if(node == null) {
        return;
    }
    else if(node.left == null && node.right == null) {
        path += node.data;
        sum += node.data;
        if(sum >= lo && sum <= hi) {
            System.out.println(path);
        }
        return;
    }

    pathToLeafFromRoot(node.left,path + node.data + " ",sum + node.data,lo,hi);
    pathToLeafFromRoot(node.right,path + node.data + " ",sum + node.data,lo,hi);
}
```

50 40 10 80

50 30 35

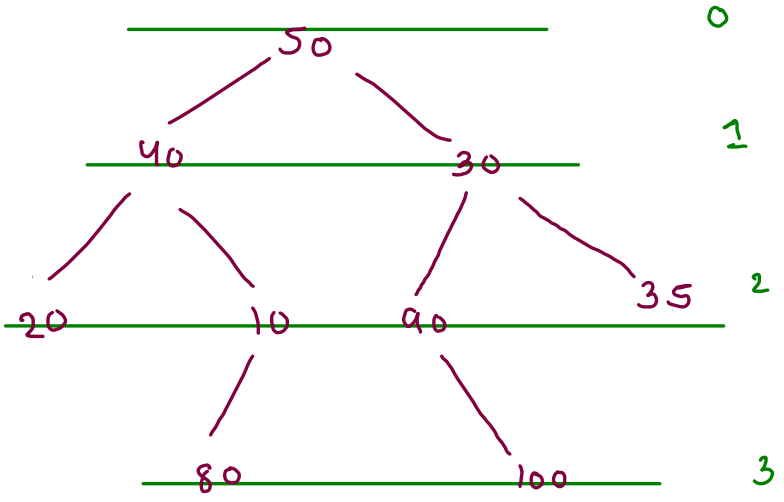
Print Single Child Nodes

print all nodes which one single child of this parents.




```java
public static void helper(Node node) {
    if(node == null) {
        return;
    }

    //pre area self work
    if(node.left == null && node.right != null) {
        //node has only right child
        System.out.println(node.right.data);
    }
    else if(node.left != null && node.right == null) {
        //node has only one child
        System.out.println(node.left.data);
    }

    helper(node.left);
    helper(node.right);
}
```

Print K Levels Down



level order line wise.

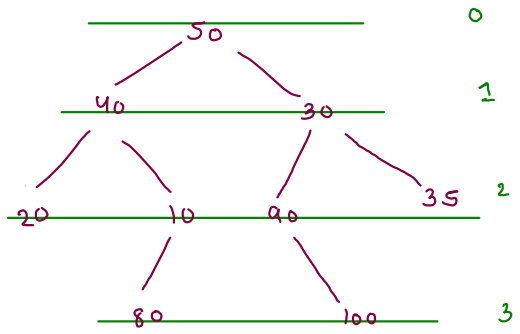
dev = ~~0~~ ~~1~~ 2

c = 2

count times
- remove
- work
- add children

dev++;

~~50~~	~~40~~	~~30~~	20	10	90	35



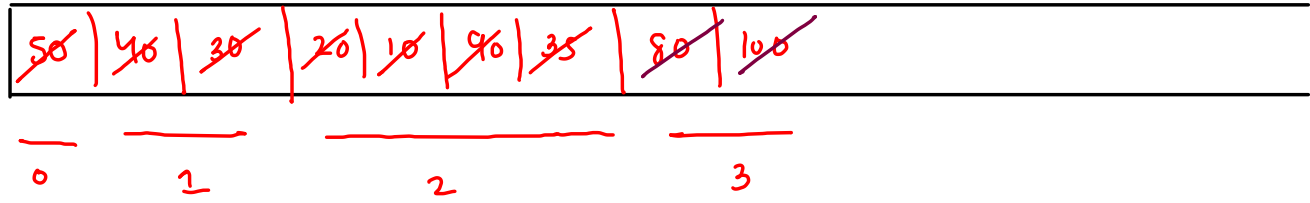

```
while(q.size() > 0) {
    if(lev == k) {
        break;
    }

    int count = q.size();

    for(int i=0; i < count;i++) {
        Node rem = q.remove();

        //add children
        if(rem.left != null) {
            q.add(rem.left);
        }
        if(rem.right != null) {
            q.add(rem.right);
        }
    }

    lev++;
}

//queue has k'th level nodes
while(q.size() > 0) {
    System.out.println(q.remove().data);
}
```

k = 3

lev = ~~0~~ ~~1~~ ~~2~~ 3

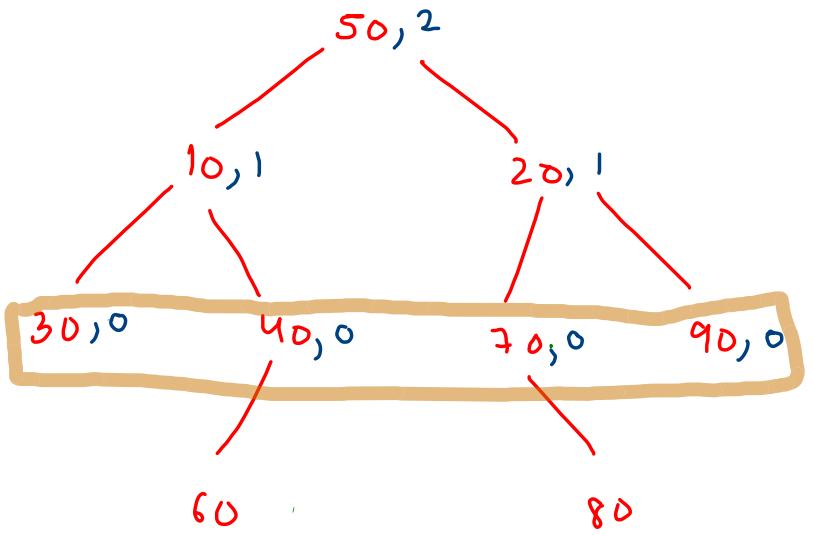
c = 4

80 100

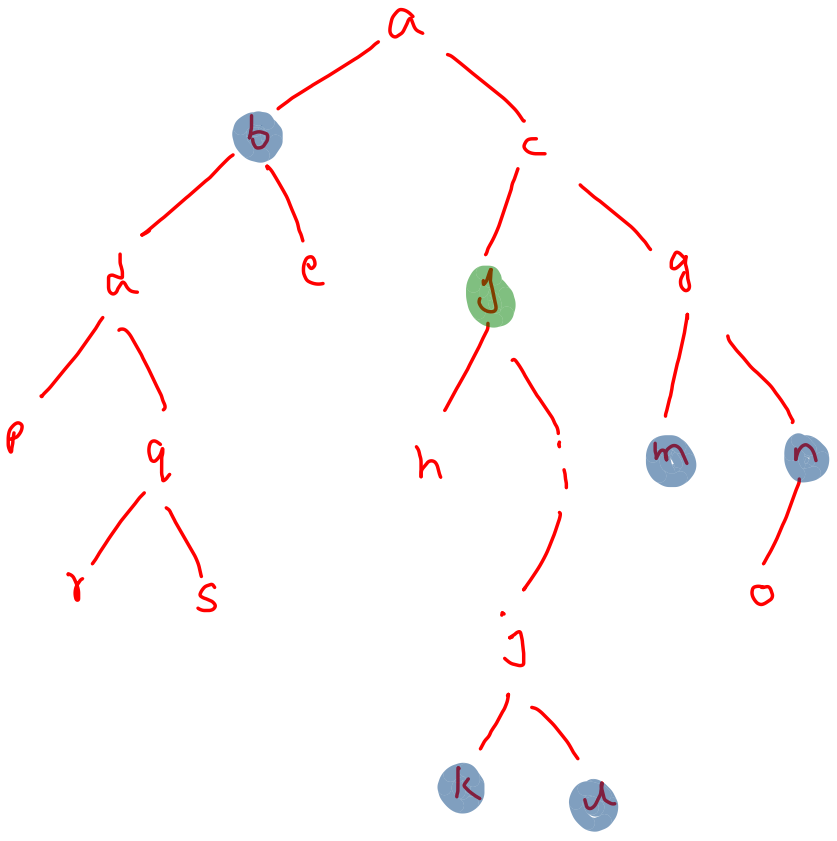


recursion

k = 2



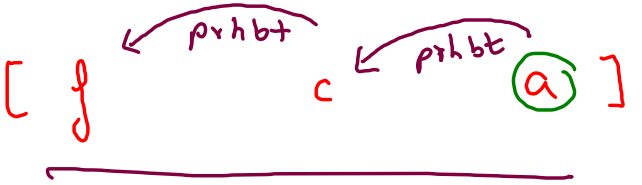
Print Nodes K Distance Away



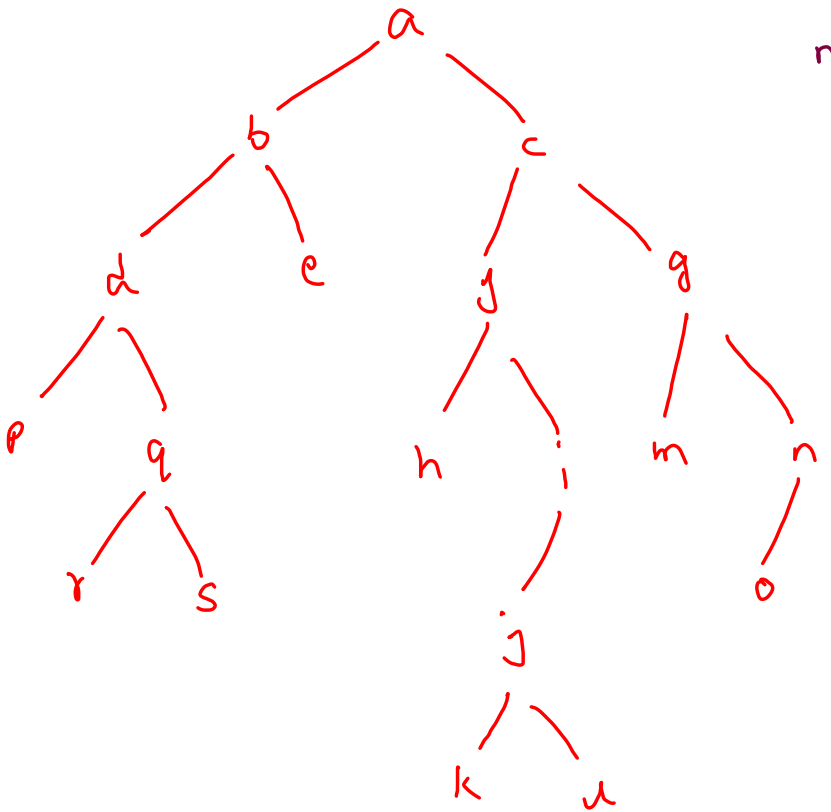
node : f

k = 3

nodeToRootPath , print k levels down



k-level downs	k	k-1	k-2
	—	—	—
	k, l	~~h~~, i, m, n	~~b~~, c



node f

k = 3

```
if(node == null || node == prhbt) {
    return;
}

if(k == 0) {
    System.out.println(node.data);
    return;
}

printKLevelsDown(node.left,k-1,prhbt);
printKLevelsDown(node.right,k-1,prhbt);
```

```
ArrayList<Node>path = nodeToRootPath(node,data);
Node prhbt = null;

for(int i=0; i < path.size();i++) {
    printKLevelsDown(path.get(i), k-i , prhbt);
    prhbt = path.get(i);
}
```

[f c a]

i: 0 1 2

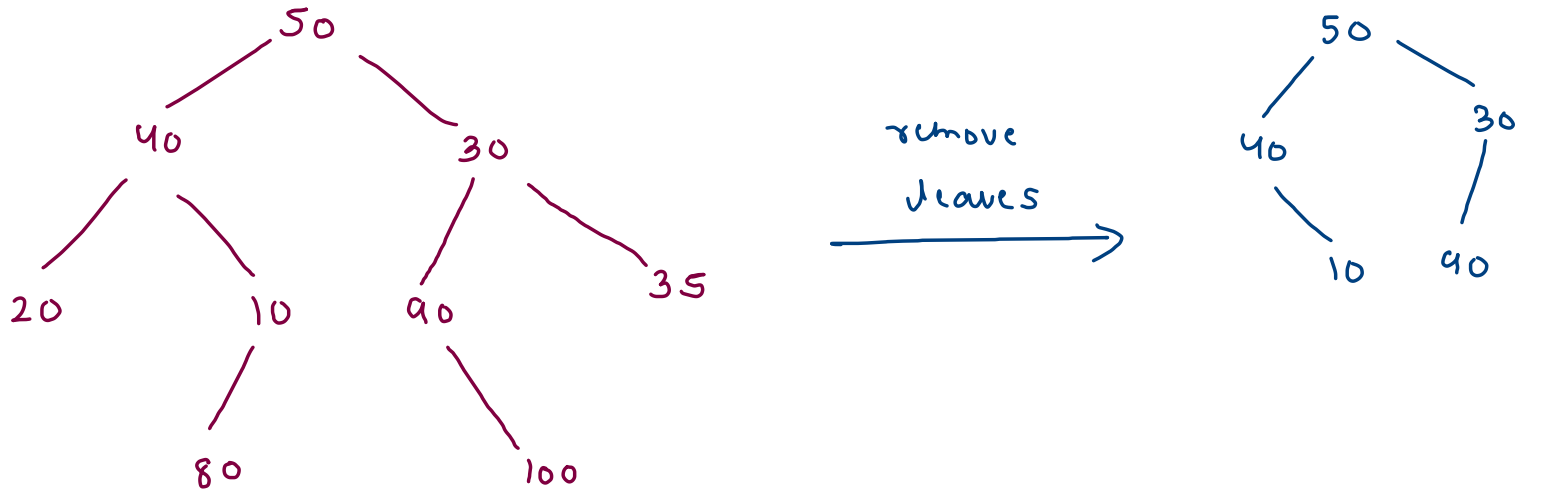
prhbt: null f c

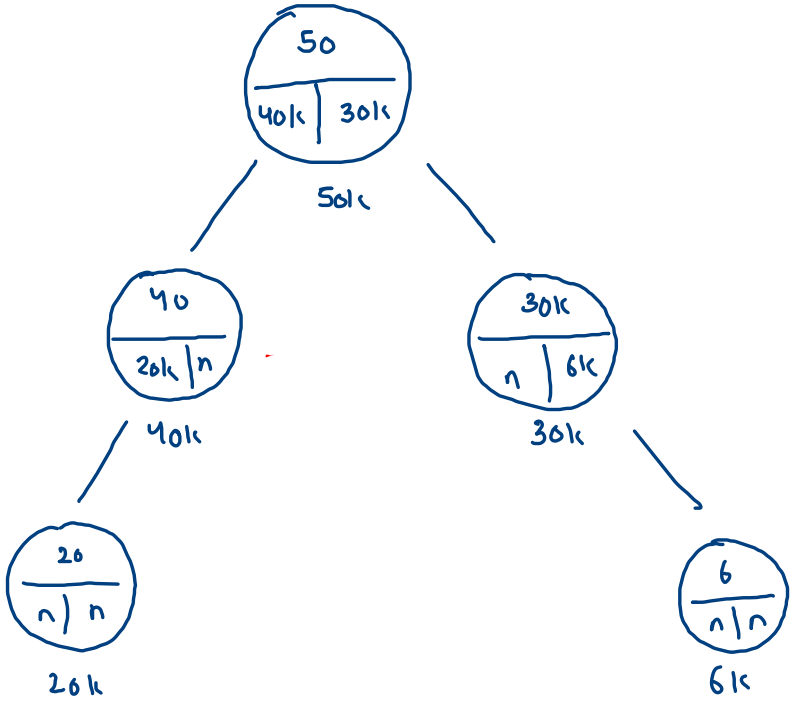
k down: 3 2 1

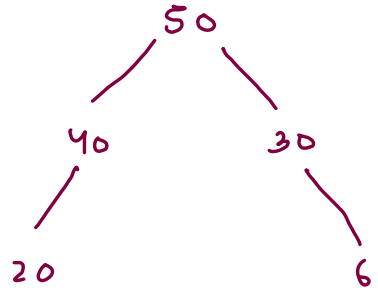
↓ ↓ ↓

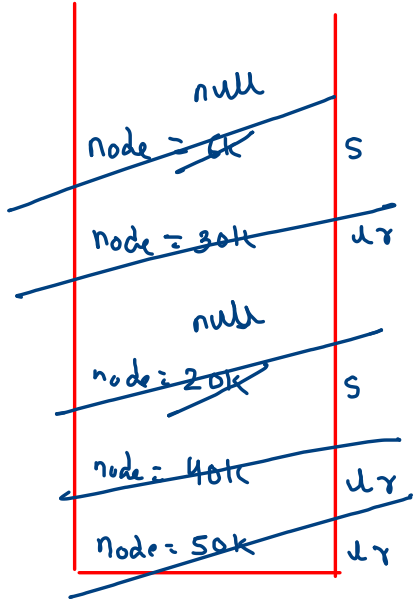
k, l m, n b

Remove Leaves In Binary Tree



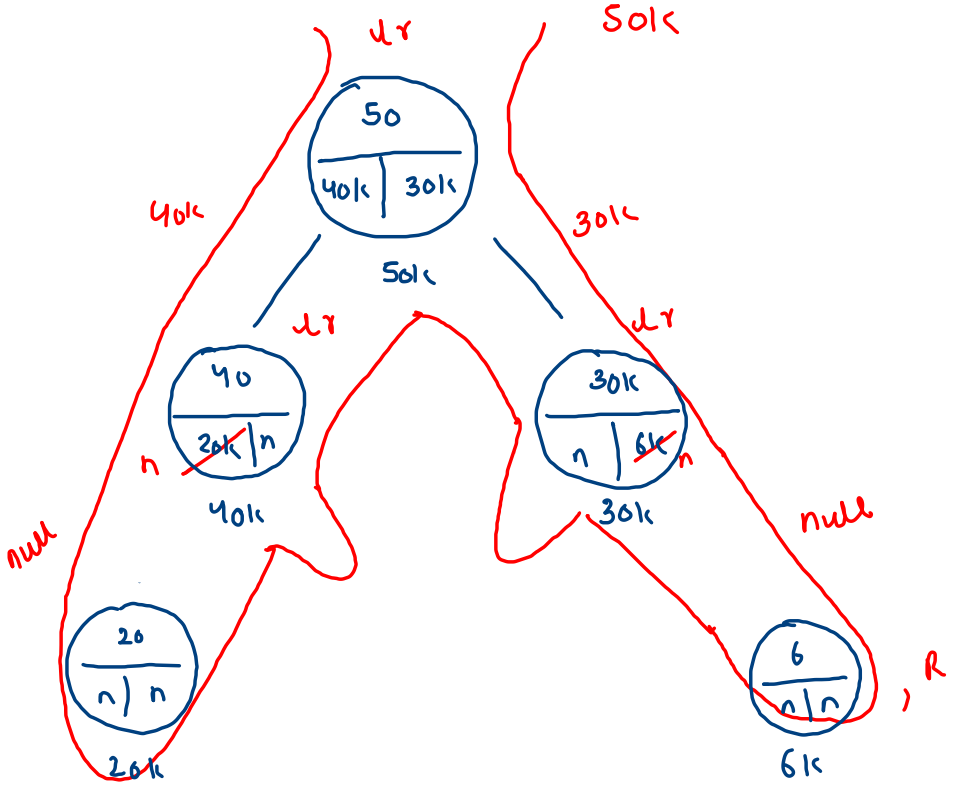


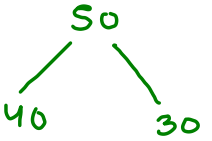




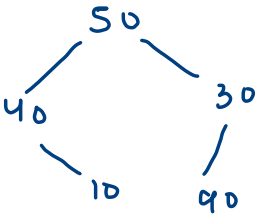

```java
public static void helper(Node node) {
    if(node == null) {
        return;
    }

    if(node.left == null && node.right == null) {
        node = null;
        return;
    }

    helper(node.left);
    helper(node.right);

}
```

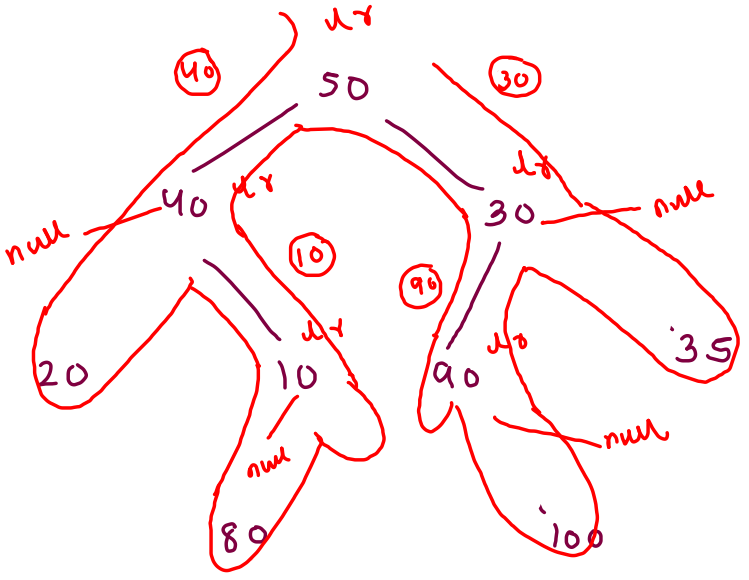
wrong






```java
public static Node removeLeaves(Node node){
    if(node == null) {
        return null;
    }
    //leaf node
    else if(node.left == null && node.right == null) {
        return null;
    }

    node.left = removeLeaves(node.left);
    node.right = removeLeaves(node.right);

    return node;
}
```

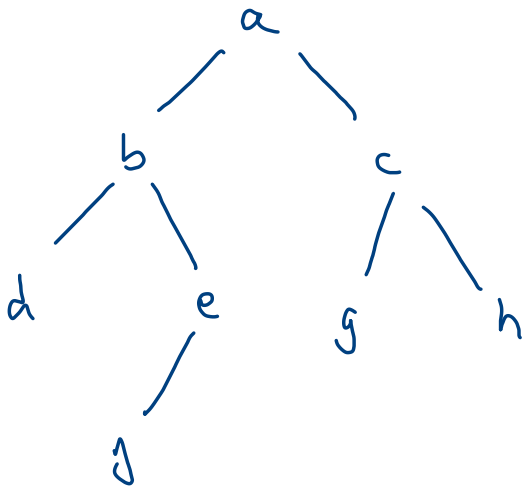




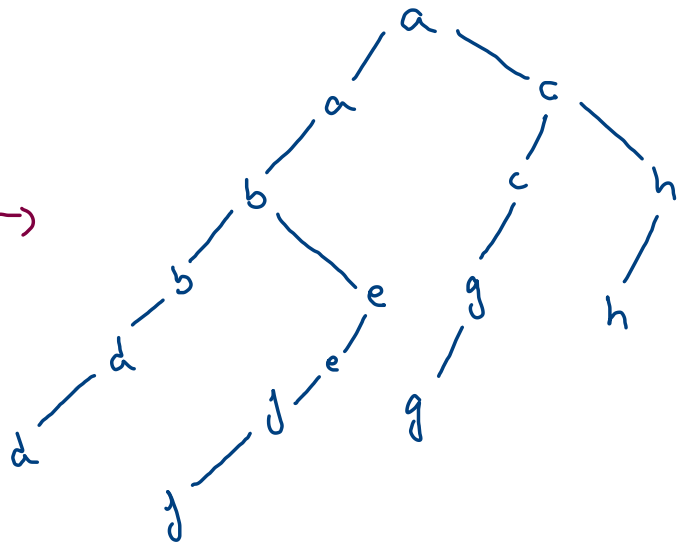

```java
public static Node removeLeaves(Node node){
    if(node == null) {
        return null;
    }
    //Leaf node
    else if(node.left == null && node.right == null) {
        return null;
    }

    node.left = removeLeaves(node.left);
    node.right = removeLeaves(node.right);

    return node;
}
```

Transform To Left-cloned Tree

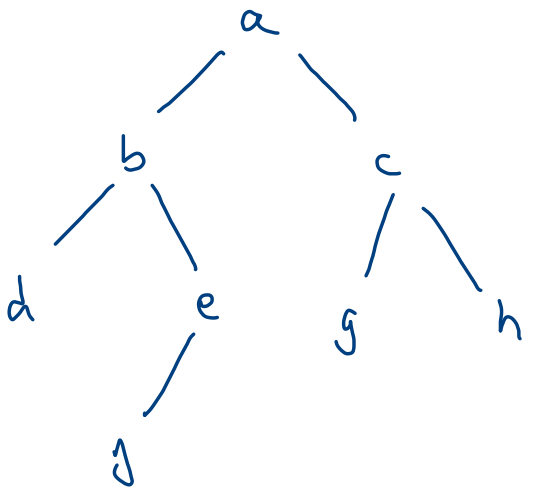
insert a node of value 'parent' between parent node and its left child.

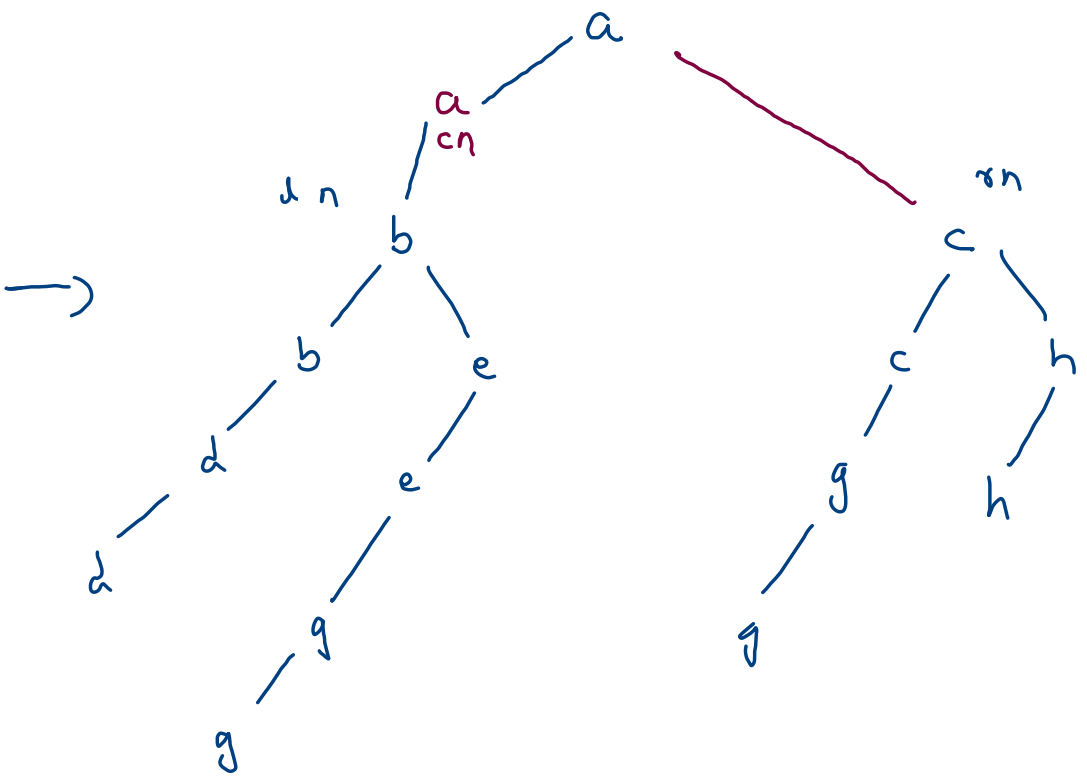


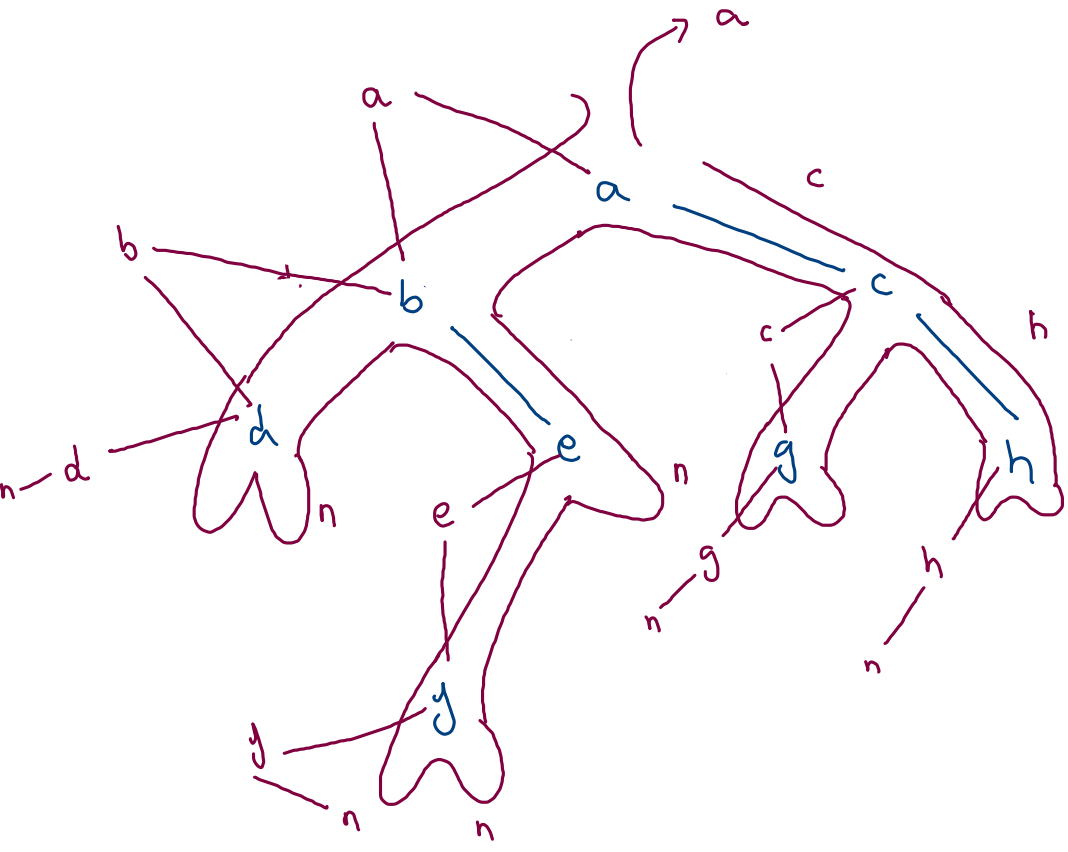
left cloned tree



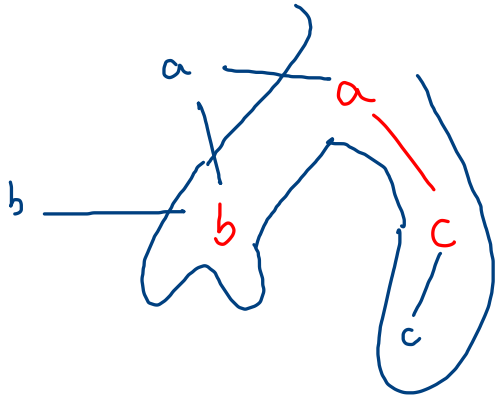
```
node.right = rn;
Node cn = new Node(node.data);
node.left = cn;
cn.left = ln;
```








```java
public static Node createLeftCloneTree(Node node){
    if(node == null) {
        return null;
    }

    Node ln = createLeftCloneTree(node.left); //left subtree root after cloning
    Node rn = createLeftCloneTree(node.right); //right subtree root after cloning

    node.right = rn;
    Node cn = new Node(node.data); //cloned
    node.left = cn;
    cn.left = ln;

    return node;
}
```




```java
public static Node createLeftCloneTree(Node node){
    if(node == null) {
        return null;
    }

    Node ln = createLeftCloneTree(node.left); //left subtree root after cloning
    Node rn = createLeftCloneTree(node.right); //right subtree root after cloning

    node.right = rn;
    Node cn = new Node(node.data); //cloned
    node.left = cn;
    cn.left = ln;

    return node;
}
```