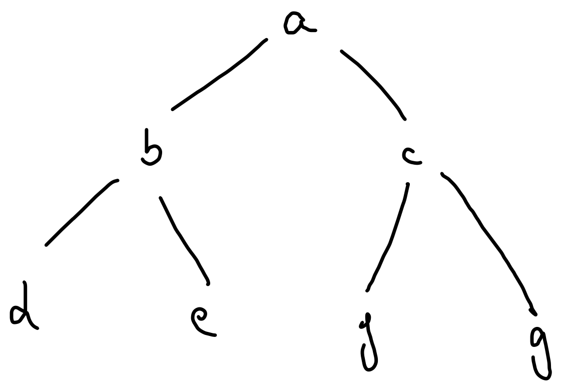
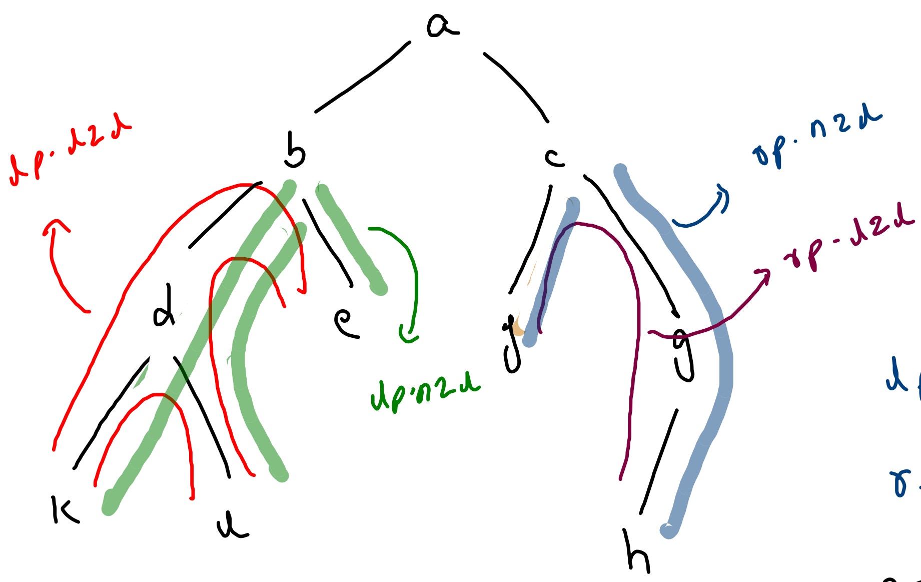
Maximum Path Sum In Between Two Leaves Of Binary Tree



f-g

d-e : d+b+e
d-f
d-g

e-f : e+b+a+c+f
e-g



Pair :

n2L

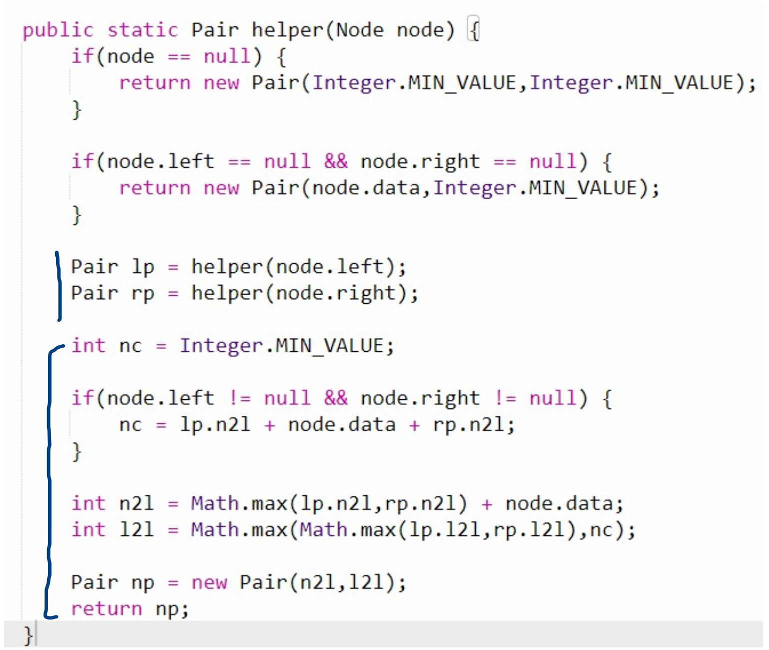
l2l

lp = helper(node.left);

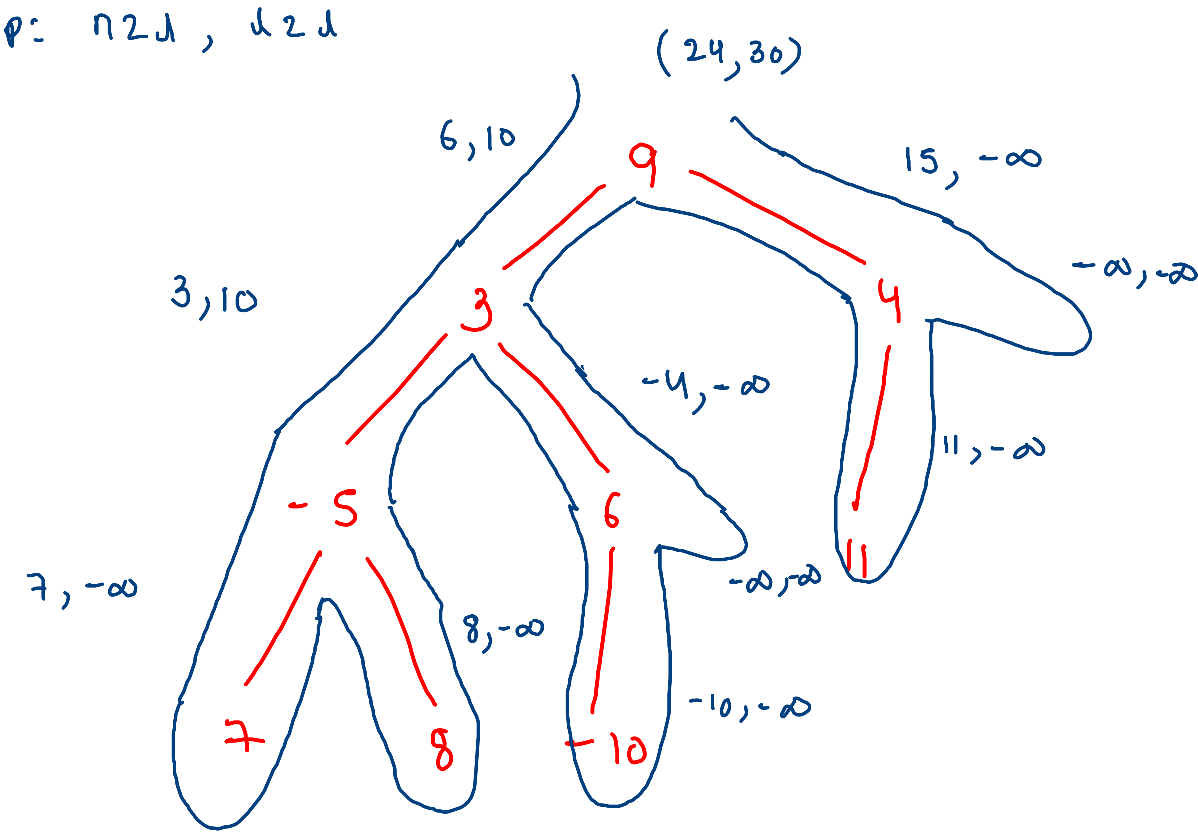
rp = helper(node.right);

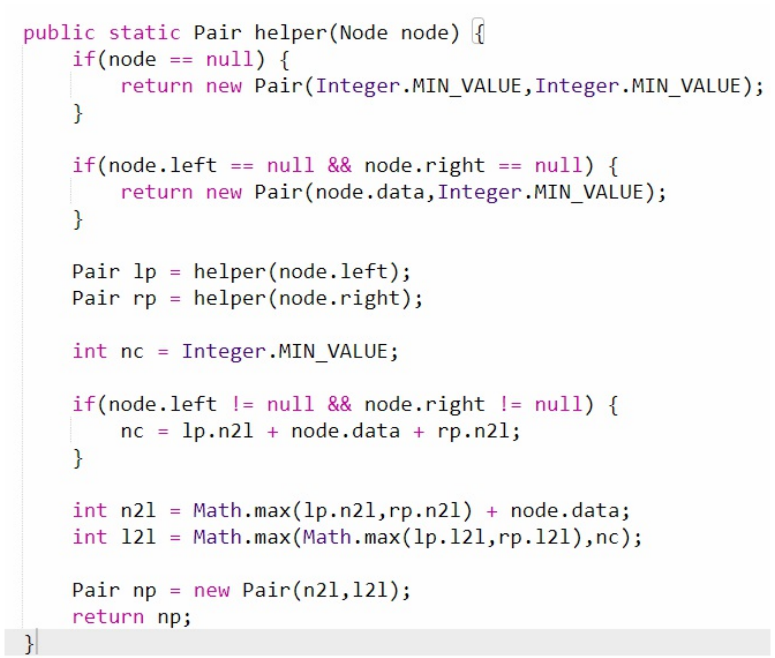
np.l2l = max(lp.n2L + node.val + rp.n2L),
lp.l2l , rp l2l

np.n2l = max (lp.n2l , rp.n2l) + node.val

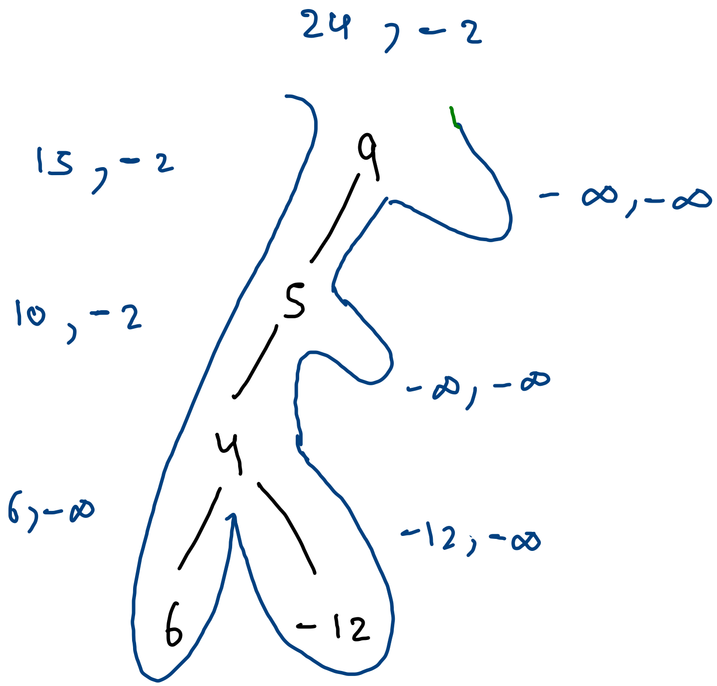


```java
public static Pair helper(Node node) {
    if(node == null) {
        return new Pair(Integer.MIN_VALUE,Integer.MIN_VALUE);
    }

    if(node.left == null && node.right == null) {
        return new Pair(node.data,Integer.MIN_VALUE);
    }

    Pair lp = helper(node.left);
    Pair rp = helper(node.right);

    int nc = Integer.MIN_VALUE;

    if(node.left != null && node.right != null) {
        nc = lp.n2l + node.data + rp.n2l;
    }

    int n2l = Math.max(lp.n2l,rp.n2l) + node.data;
    int l2l = Math.max(Math.max(lp.l2l,rp.l2l),nc);

    Pair np = new Pair(n2l,l2l);
    return np;
}
```

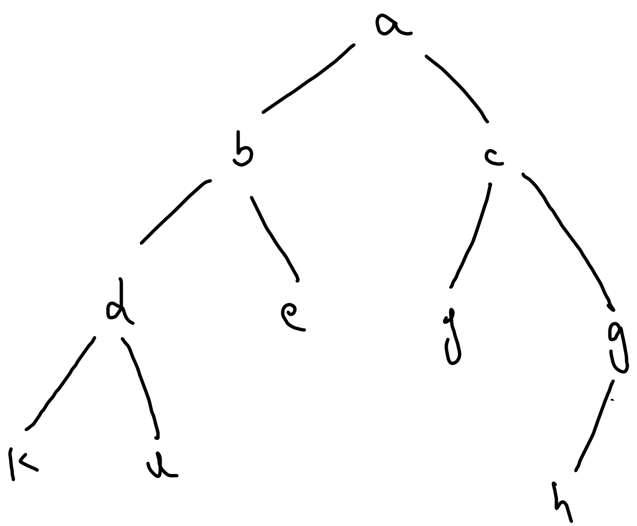




```java
public static Pair helper(Node node) {
    if(node == null) {
        return new Pair(Integer.MIN_VALUE,Integer.MIN_VALUE);
    }

    if(node.left == null && node.right == null) {
        return new Pair(node.data,Integer.MIN_VALUE);
    }

    Pair lp = helper(node.left);
    Pair rp = helper(node.right);

    int nc = Integer.MIN_VALUE;

    if(node.left != null && node.right != null) {
        nc = lp.n2l + node.data + rp.n2l;
    }

    int n2l = Math.max(lp.n2l,rp.n2l) + node.data;
    int l2l = Math.max(Math.max(lp.l2l,rp.l2l),nc);

    Pair np = new Pair(n2l,l2l);
    return np;
}
```



124. Binary Tree Maximum Path Sum



node to node max path sum

Pair :

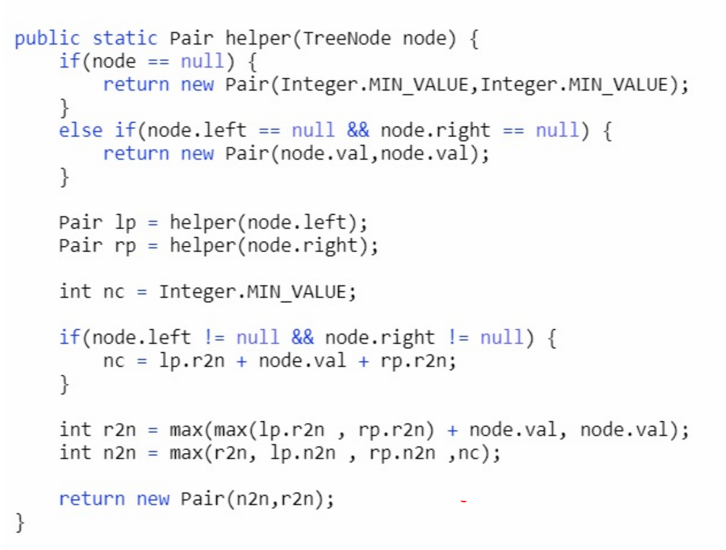
n2n → node to node

r2n → root to node

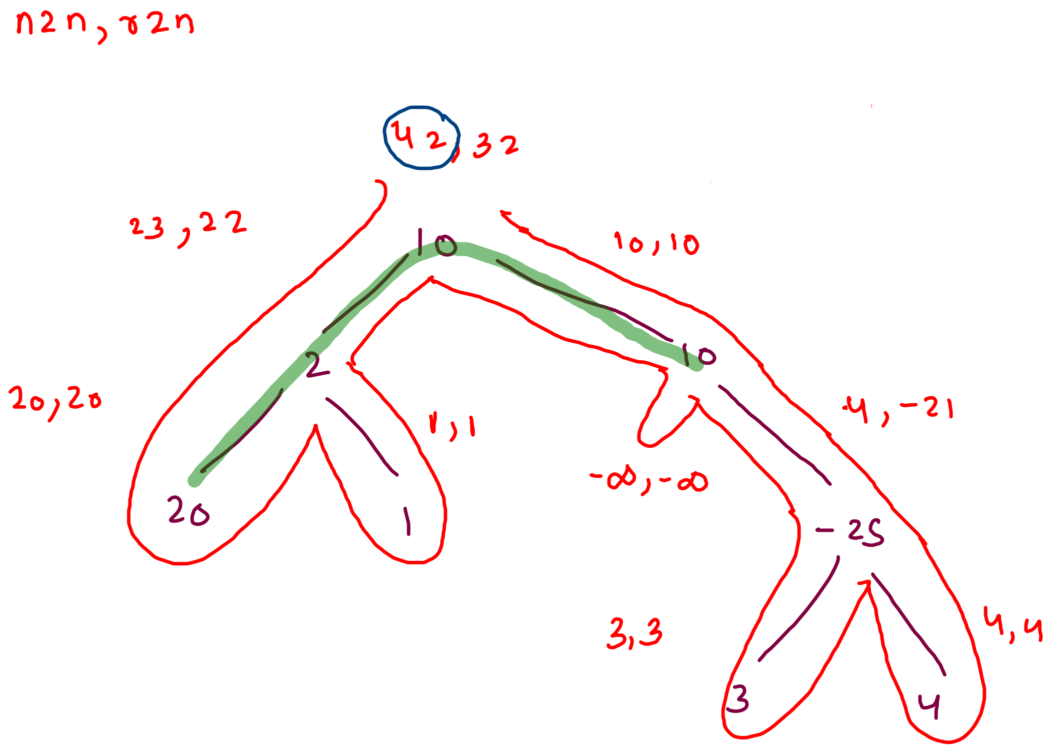
```
lp = helper (node.left);
rp = helper (node.right);
```

comparison incomplete

```
np.n2n = max(lp.r2n + node.val + rp.r2n),
             lp.n2n, rp.n2n

np.r2n = max (lp.r2n , rp.r2n) + root.data
```

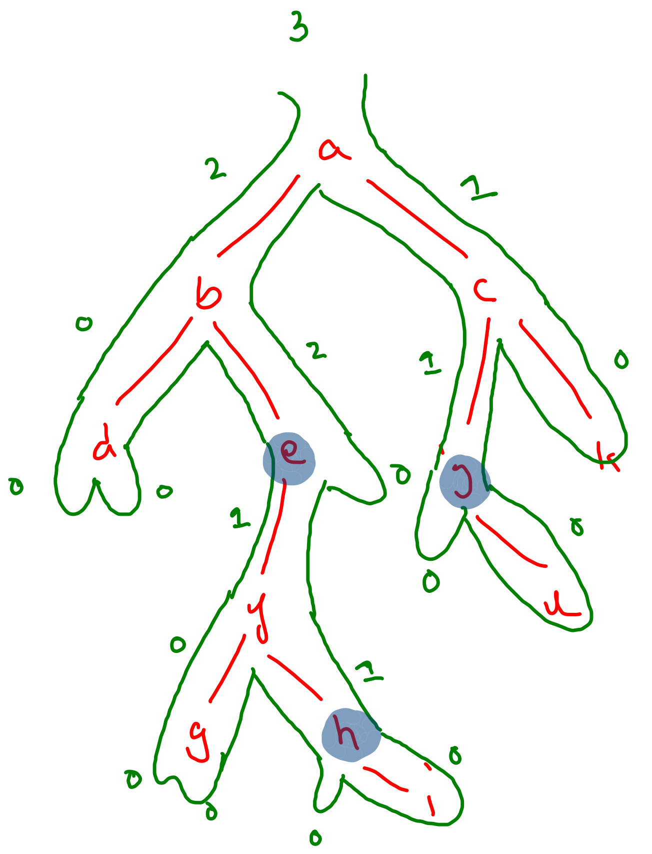


```java
public static Pair helper(TreeNode node) {
    if(node == null) {
        return new Pair(Integer.MIN_VALUE,Integer.MIN_VALUE);
    }
    else if(node.left == null && node.right == null) {
        return new Pair(node.val,node.val);
    }

    Pair lp = helper(node.left);
    Pair rp = helper(node.right);

    int nc = Integer.MIN_VALUE;

    if(node.left != null && node.right != null) {
        nc = lp.r2n + node.val + rp.r2n;
    }

    int r2n = max(max(lp.r2n , rp.r2n) + node.val, node.val);
    int n2n = max(r2n, lp.n2n , rp.n2n ,nc);

    return new Pair(n2n,r2n);
}
```

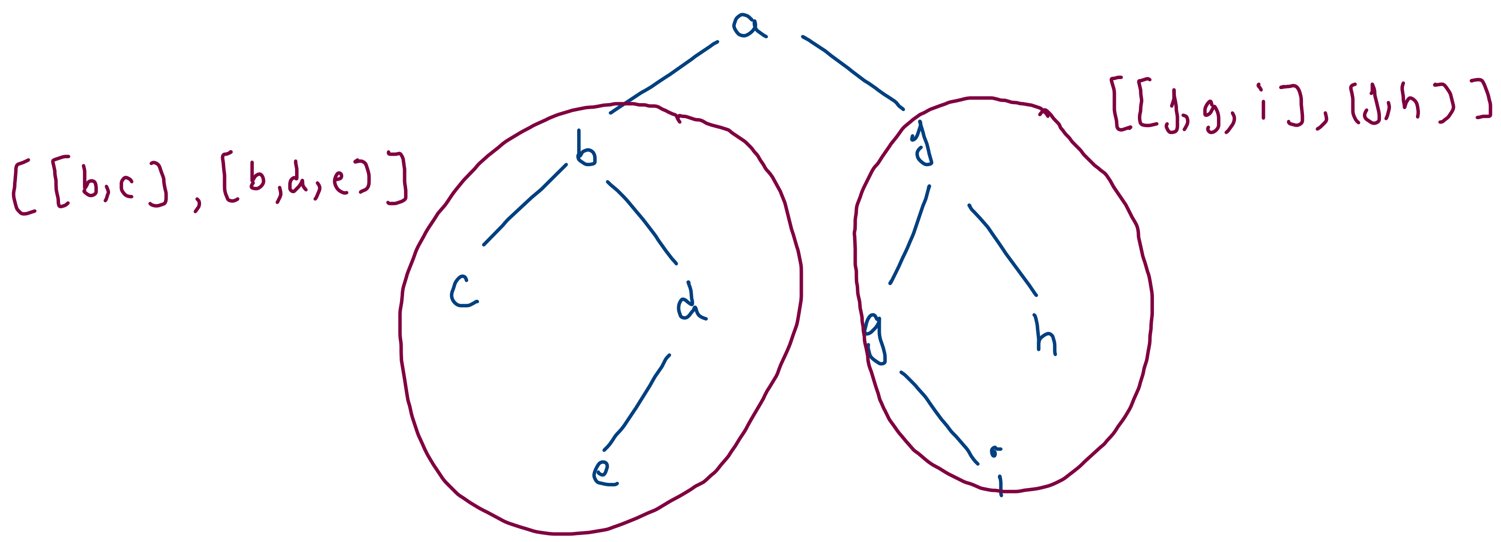


Count All Single Child Parent In Binary Tree

```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 is single child parent
        if((node.left == null && node.right != null) || (node.left != null && node.right == null)) {
            ans += 1;
        }

        return ans;
}
```



Root To All Leaf Path In Binary Tree



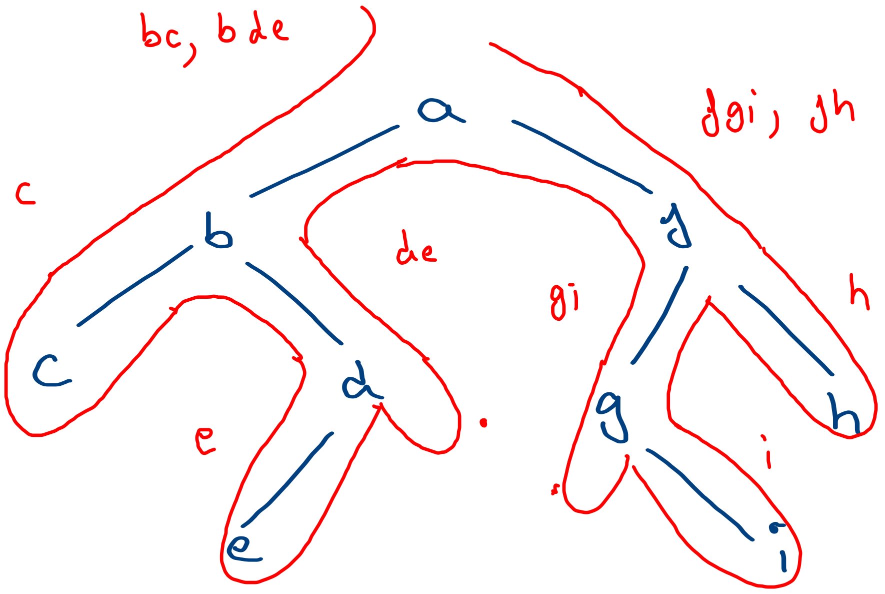
[[a b c]

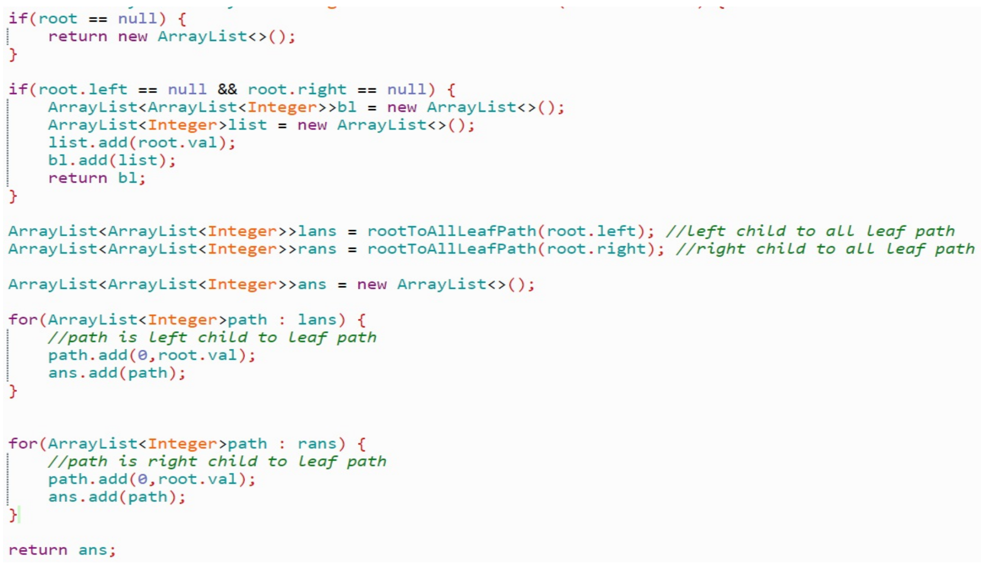
[a b d e]

[a f g i]

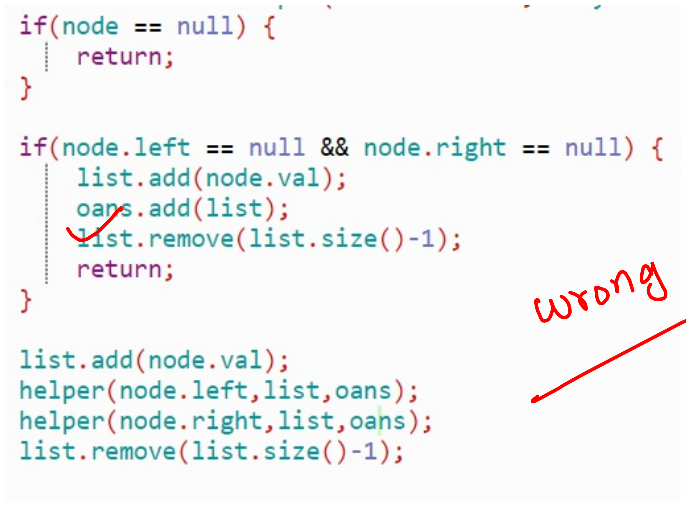
[a f h i]]

[[a, b, c], [a, b, d, e], [a, f, g, i], [a, f, h]]



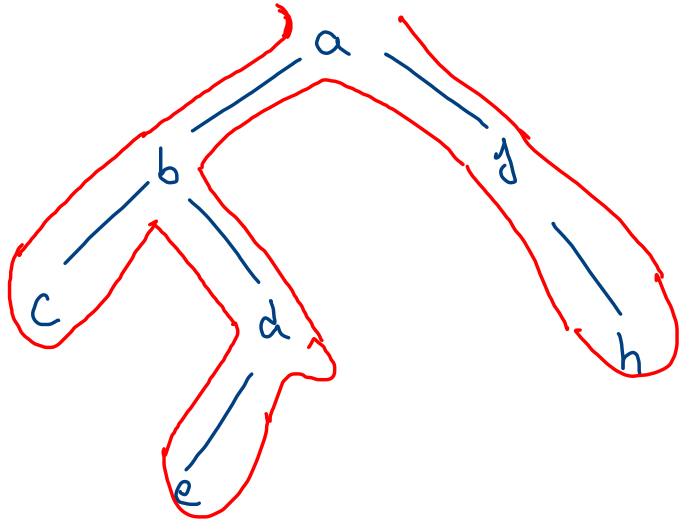


```
if(root == null) {
    return new ArrayList<>();
}

if(root.left == null && root.right == null) {
    ArrayList<ArrayList<Integer>>bl = new ArrayList<>();
    ArrayList<Integer>list = new ArrayList<>();
    list.add(root.val);
    bl.add(list);
    return bl;
}

ArrayList<ArrayList<Integer>>lans = rootToAllLeafPath(root.left); //left child to all leaf path
ArrayList<ArrayList<Integer>>rans = rootToAllLeafPath(root.right); //right child to all leaf path

ArrayList<ArrayList<Integer>>ans = new ArrayList<>();

for(ArrayList<Integer>path : lans) {
    //path is left child to leaf path
    path.add(0,root.val);
    ans.add(path);
}


for(ArrayList<Integer>path : rans) {
    //path is right child to leaf path
    path.add(0,root.val);
    ans.add(path);
}

return ans;
```



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

if(node.left == null && node.right == null) {
    list.add(node.val);
    oans.add(list);
    list.remove(list.size()-1);
    return;
}

list.add(node.val);
helper(node.left,list,oans);
helper(node.right,list,oans);
list.remove(list.size()-1);
```

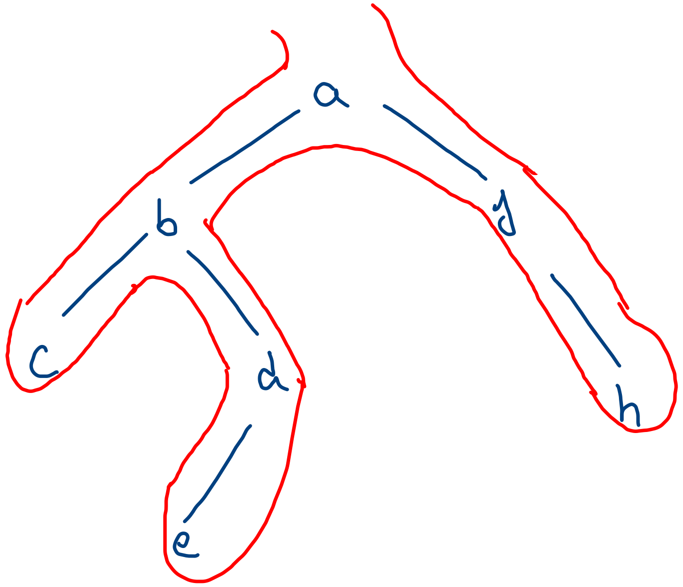
wrong

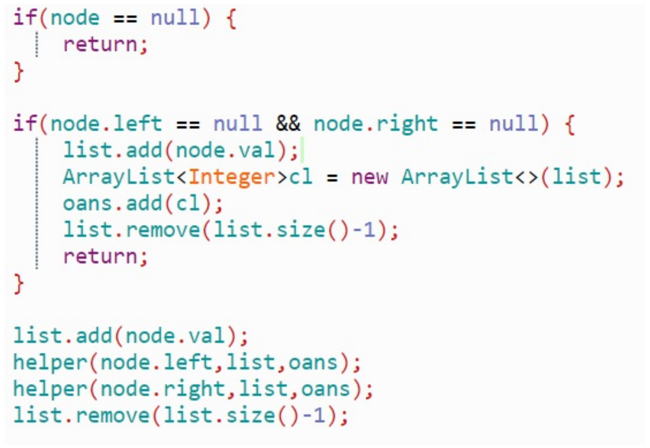


[~~a~~, ~~b~~, ~~d~~, ~~e~~, ~~g~~, ~~h~~

list = 9k

oans = 6k [9k, 9k , 9k]





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

if(node.left == null && node.right == null) {
    list.add(node.val);
    ArrayList<Integer>cl = new ArrayList<>(list);
    oans.add(cl);
    list.remove(list.size()-1);
    return;
}

list.add(node.val);
helper(node.left,list,oans);
helper(node.right,list,oans);
list.remove(list.size()-1);
```

[ ]

list = 9k

oans [[a,b,c], [a,b,d,e], [a,g,h]]

6k 10k 12k 13k