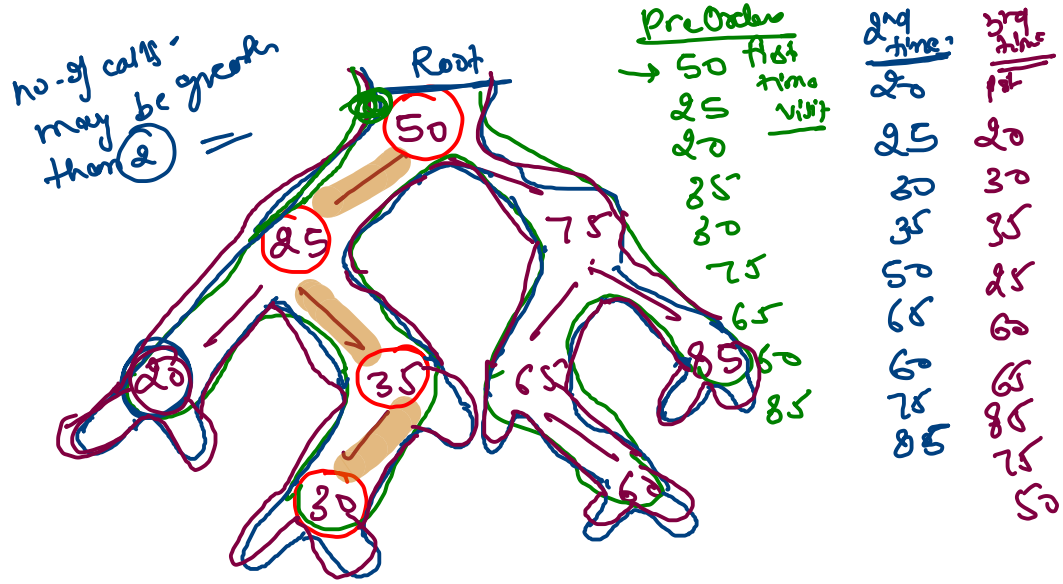
Find and Node to Root path.

Find - 30

If 30 is present so return node to root path.

30	35	25	50




```
public static void Traversal (Node root){
    if(root==null) return;
    //preArea
    Traversal(root.left);
    //InArea
    Traversal(root.right);
    //PostArea
}
```

```
Traversal==preOrder
  //preArea: Sys(node.data);
traversal==InOrder
  //InArea: Sys(node.data);
traversal==postOrder
  //postArea: Sys(node.data);
```

Find.

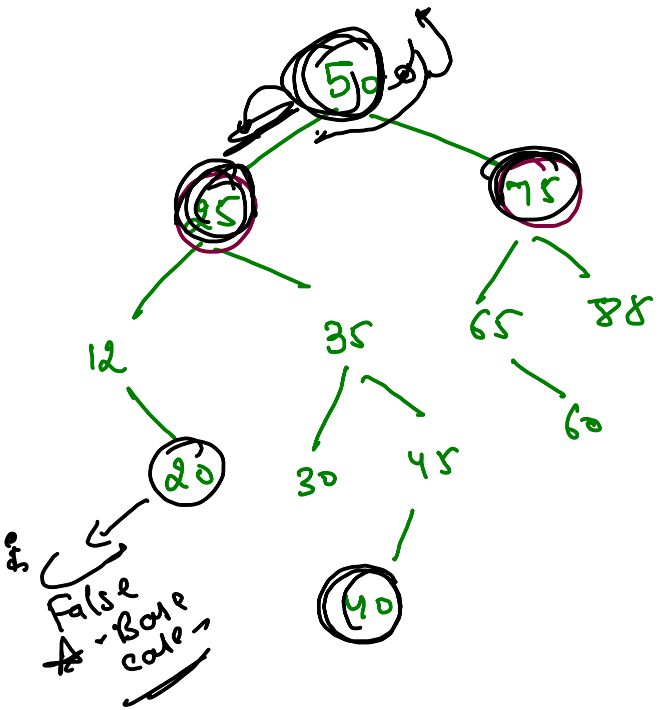
Root + Data to find.

↓ traversal — ↓ searching Element

Expectation, find(Root, data) → If exist → True; otherwise → False.

Faith, find (25, data) → How to find an Element

find (75, data) → "



Merging of faith & Expectation →

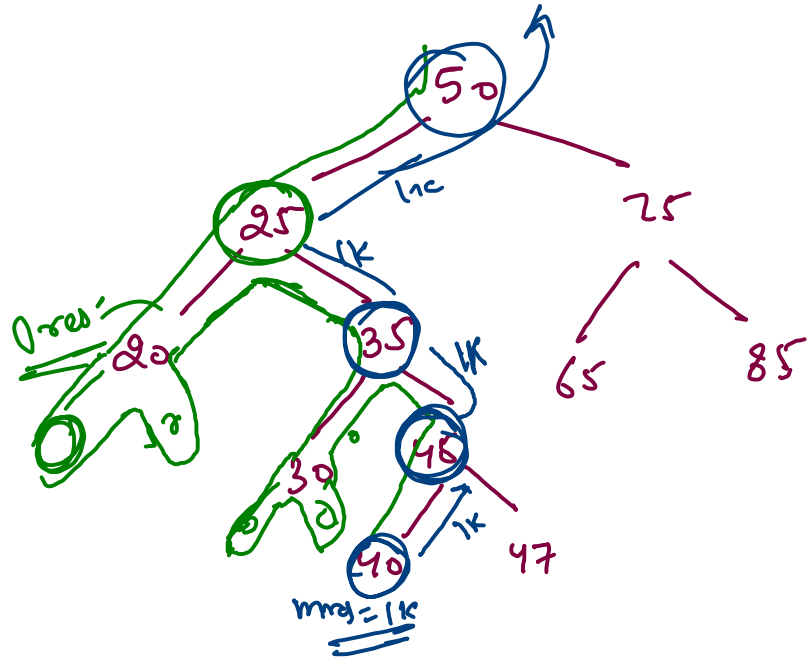
```
if(node.data == data) {
    return true;
}
boolean res = find(25, data);
if(res == true) return true;
boolean res = find(75, data);
if(res == true) return true;
return false;
```

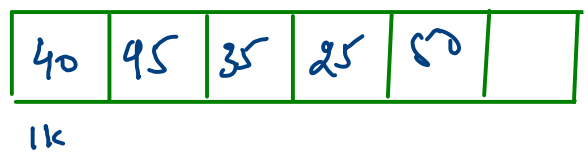
Node to Root path.

40 Node to Root path.

Node = 50

data = 40






```java
public static ArrayList<Integer> nodeToRootPath(Node node, int data){
    // write your code here
    if(node == null) {
        ArrayList<Integer> bres = new ArrayList<>();
        return bres;
    }

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

    if(node.data == data) {
        mres.add(data);
        return mres;
    }

    ArrayList<Integer> lres = nodeToRootPath(node.left, data);
    if(lres.size() > 0) {
        lres.add(node.data);
        return lres;
    }

    ArrayList<Integer> rres = nodeToRootPath(node.right, data);
    if(rres.size() > 0) {
        rres.add(node.data);
        return rres;
    }

    return mres;
}
```

Print k. level down.

k = 3;

k=3

Expectation: Kdown (node, k): ⟶ h i j k.

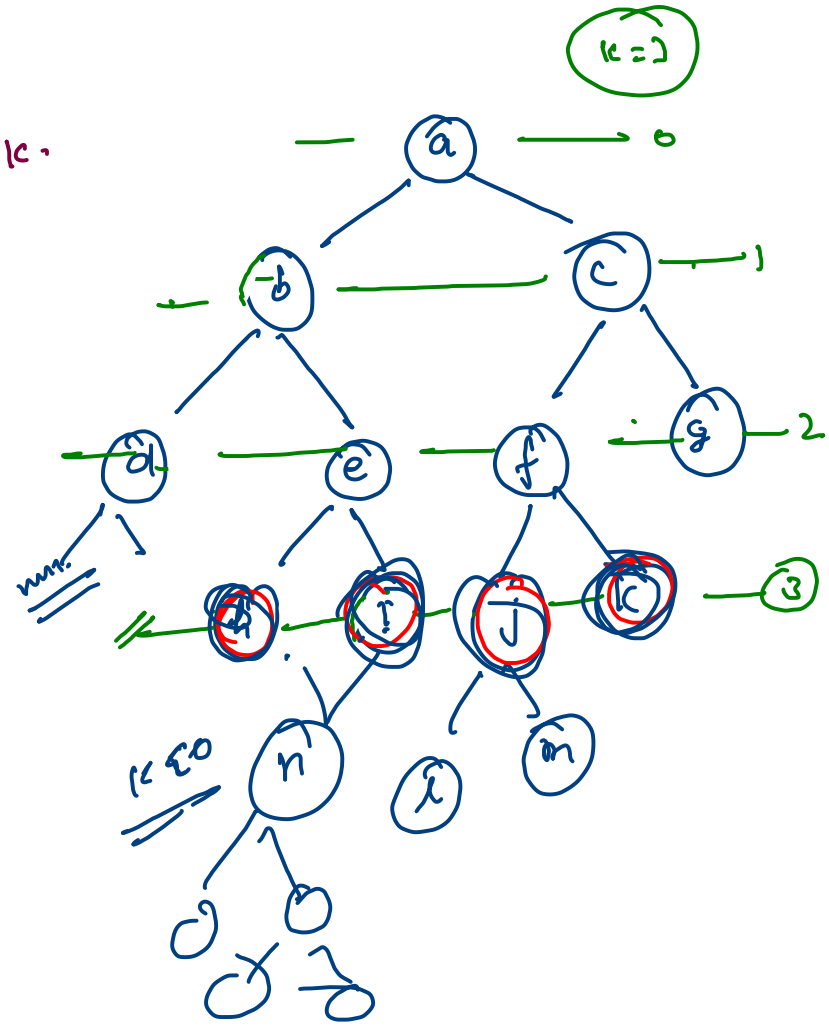
faith: = Kdown (node.left, k-1);
Kdown (node.right, k-1);

Base: if(k<0 || node == null) { return; }

Merging: if (k==0) {
Sys o (node.data); now

}

Kdown (node.left, k-1);
Kdown (node.right, k-1);

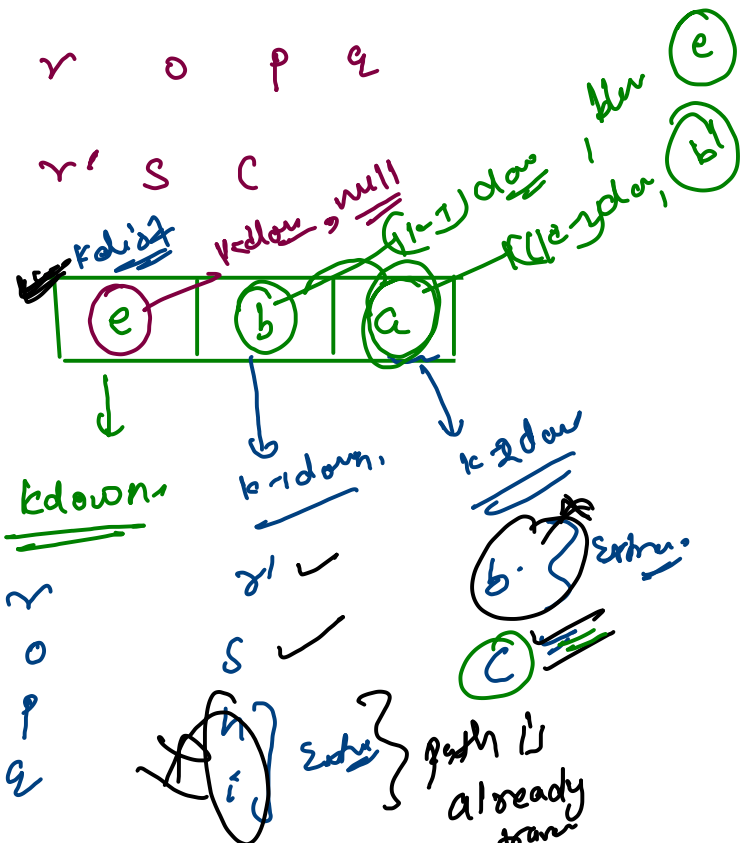


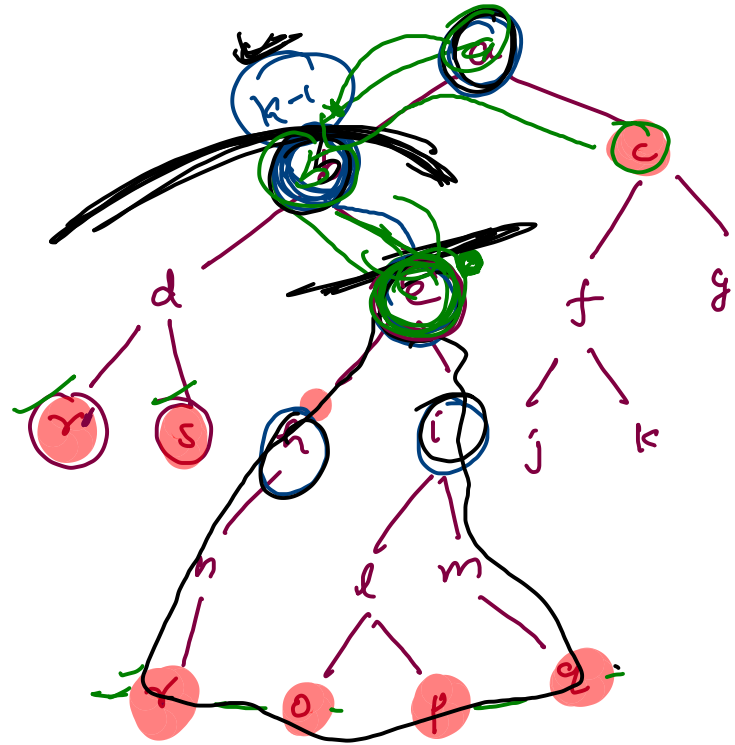
k far.

data = kdistance.

ydata = e

k = 3.





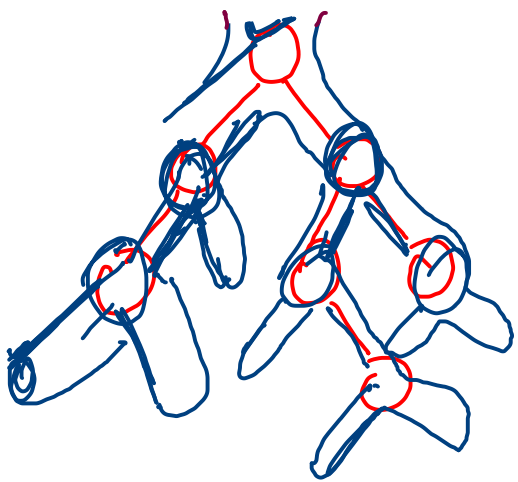
print In Range: (1)

call → Stupid

Base Case → Smart

```java
// method-1, calls -> stupid, base case -> smart
public static void traversal(Node node) { }
    if(node == null) return;

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

call



(2)

Base Case - Stupid

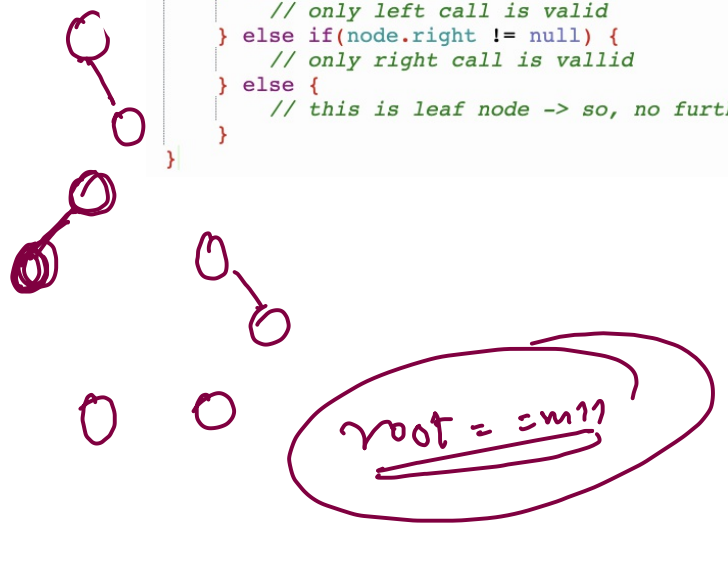
call - Smart

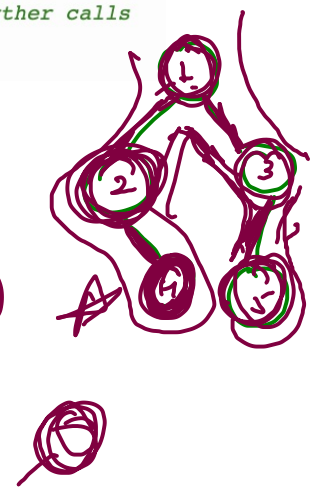
No of != nodes

```java
// method-2, calls -> smart, base case -> stupid
public static void traversal(Node node) {
    if(node == null)  return;

    if(node.left != null && node.right != null) {
        // both calls are valid
    } else if(node.left != null) {
        // only left call is valid
    } else if(node.right != null) {
        // only right call is vallid
    } else {
        // this is leaf node -> so, no further calls
    }
}
```

AND → &&

OR → ||



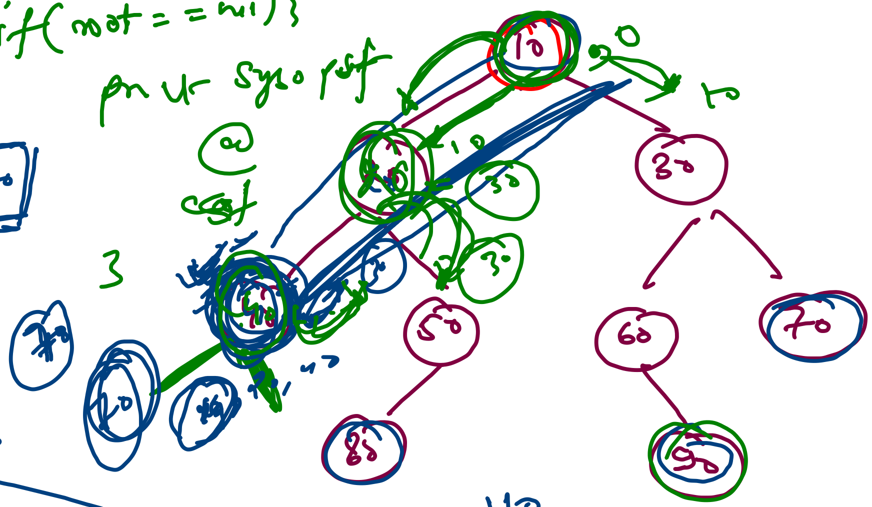


Node, Path so far — String

cost so far — Integer

Global

if(root==null){
print sysopsf @ csf



10 20 40

10, 10, 10

20, 10→20, 30

30, 10→30, 40

40, 10→20→40, 70

sum is in range.

60, 10→30→60, 100

70, 10→30→70, 110

90, 10→30→60→90, 190

50, 10→20→50, 80

80, 10→20→50→80, 160

10 → 20 → 40

10 → 20 → 50 → 80

10 → 30 → 60 → 90

10 → 30 → 70

Pathsum

70

160

190

110

if(csf >= lo && csf <= hi){

// in range.

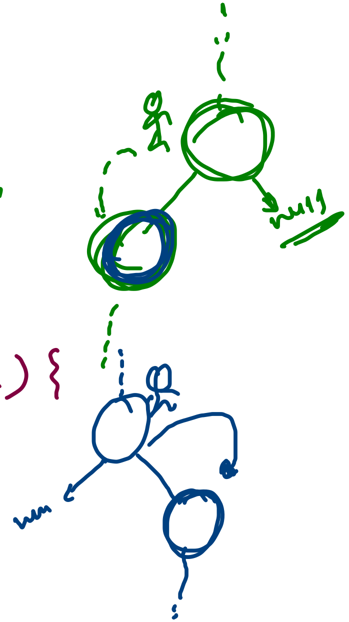
syso (psf + @ + csf)

lo = 100

hi = 180

Method - 1

```
if( left != null  && right != null) {
    call (l)
    call (r)
} else if (left != null) {
    syso node.left.data
    call (l)
} else if ( right != null) {
    Syso( node.right.data);
    call(r);
} else {
    Return.
}
```



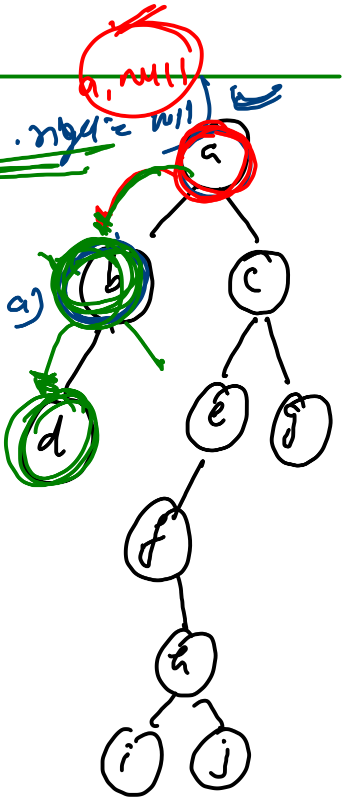
Method 2 -

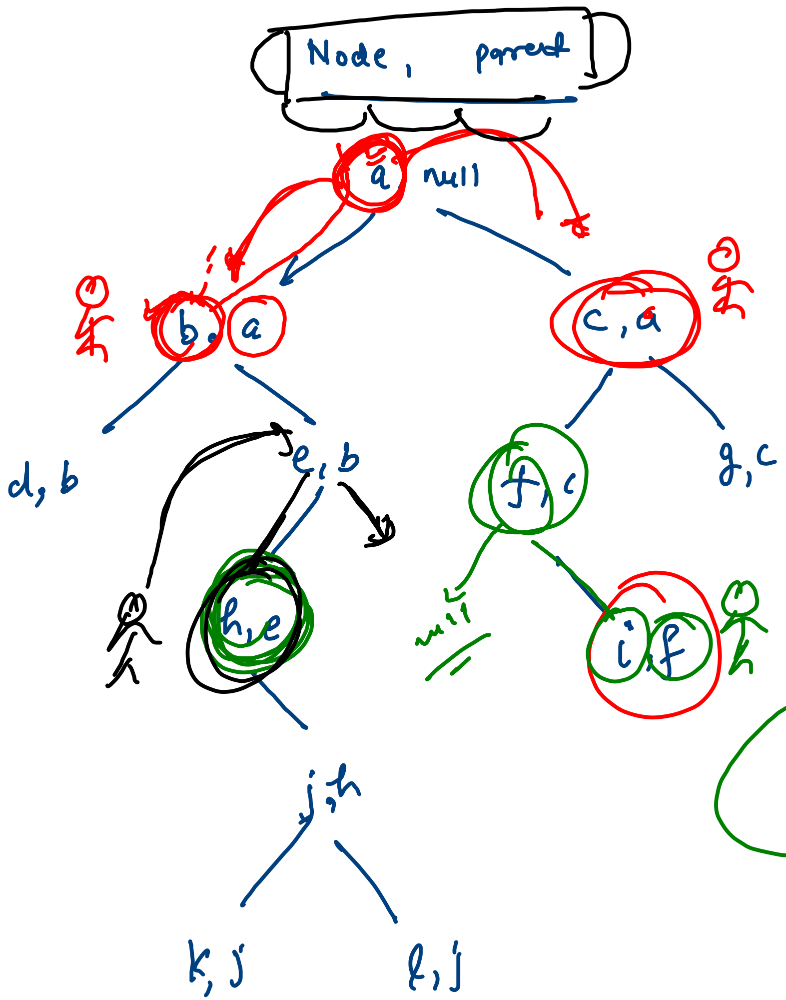
```
if (parent != null &&
    (parent.left == node && parent.right == null)
    syso (node.data);
else if (parent != null &&
    (parent.right == node &&
     parent.left == null))
    syso (node.data)

call( node.left, node);
call( node.right, node);
```

(b, a)

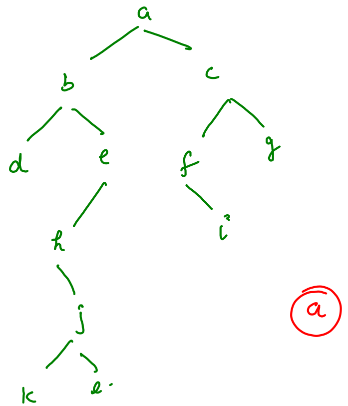
d, b

b, a





parent != null




```
if (parent.left == node && parent.right == null){
   // g in single left child :
}else if (parent.right == node && parent.left == null){
   // g in single Right child.
}
```

Sygo i

call(l)
call(r);