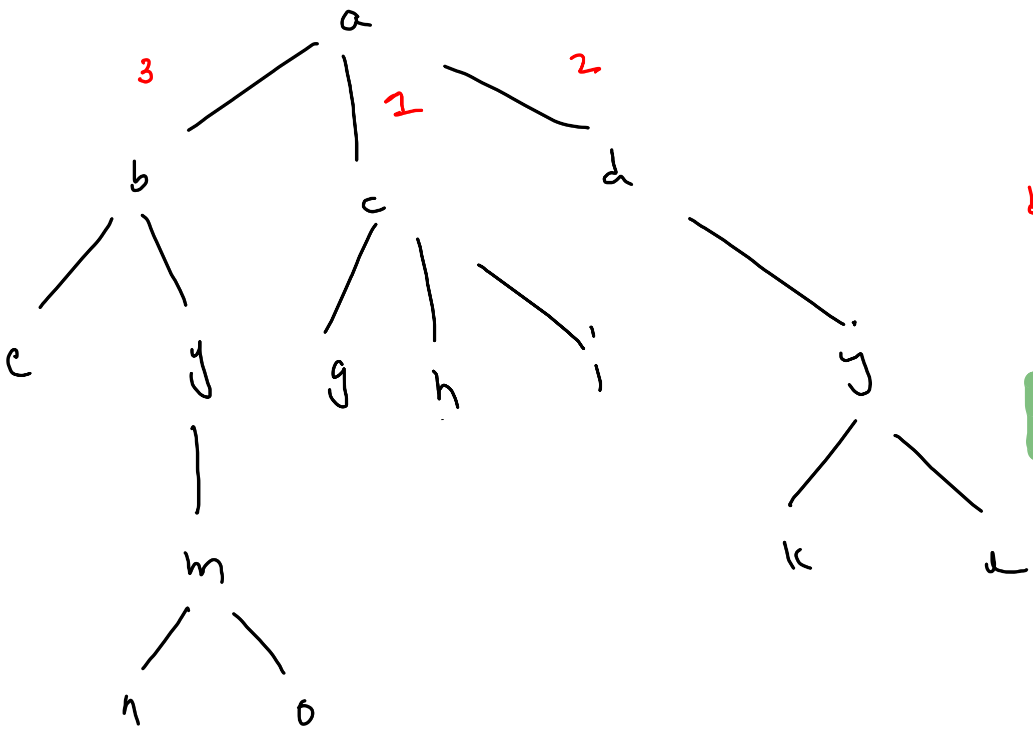
diameter

max distance between any two nodes

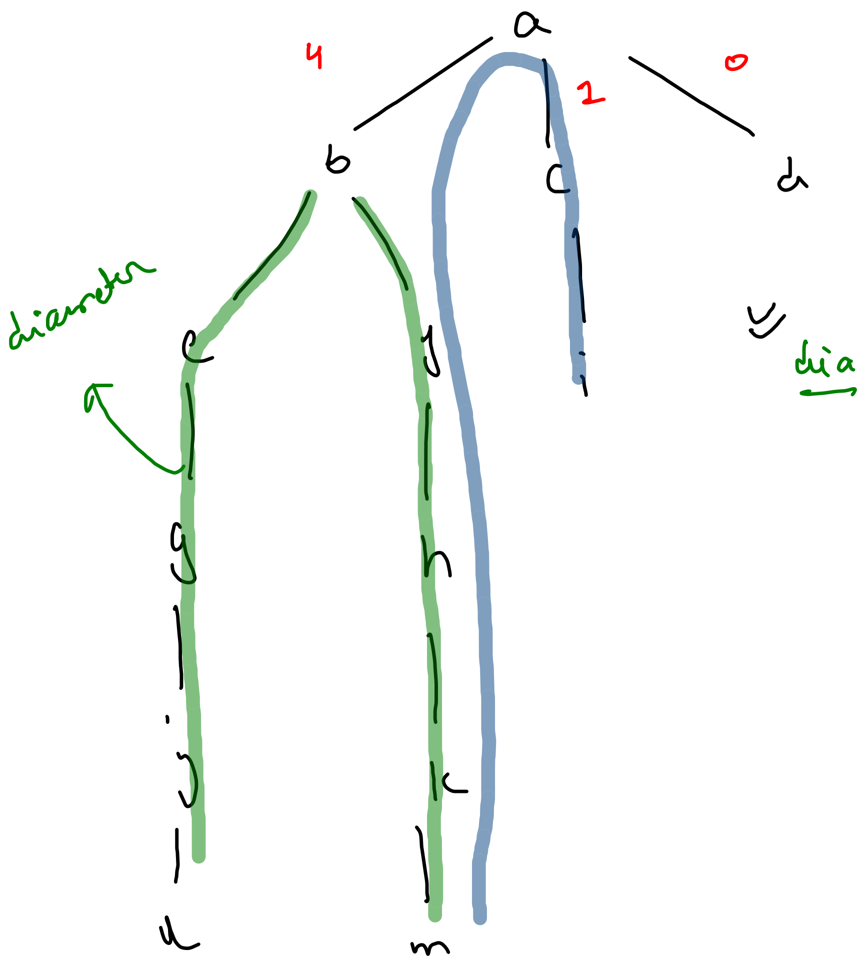


bht = 3

sbht = 2

dist = bht + sbht + 2.

= 7.



✓ dia (static)

bht = 4

sbht = 1

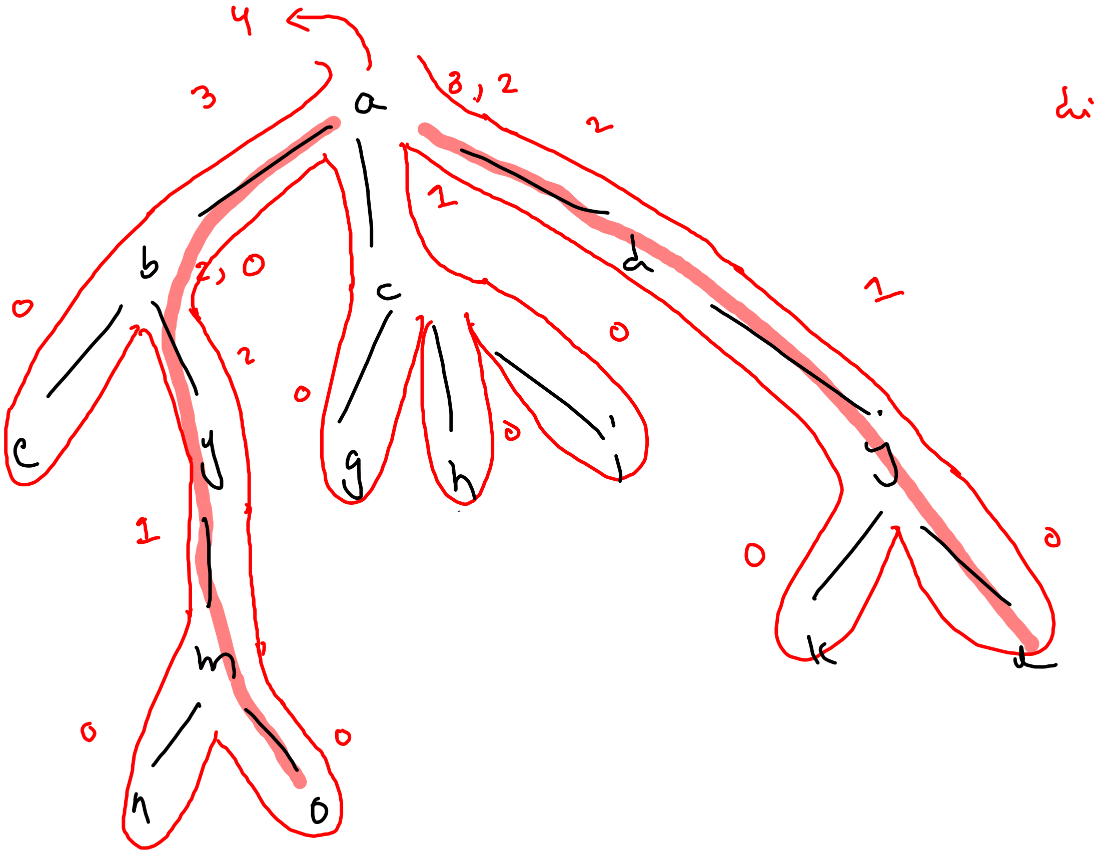
$\text{dist}_a = 4 + 1 + 2 = 7$

$\text{dist}_b = 3 + 3 + 2 = 8$

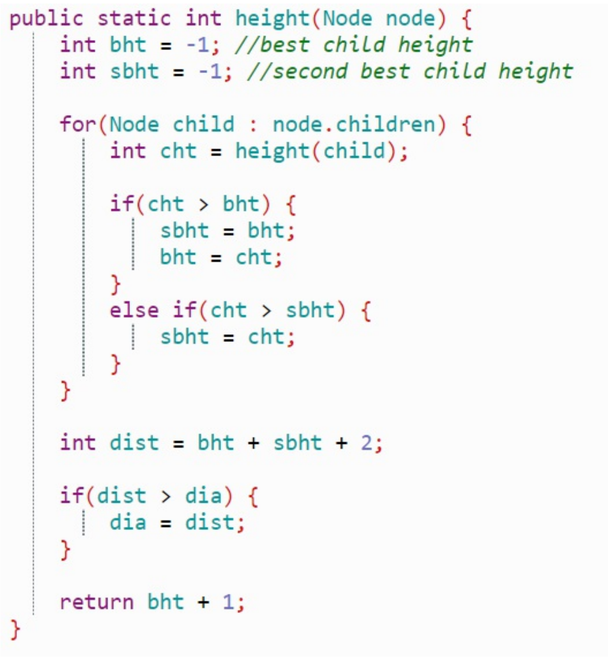
dia = max [dist

for each node

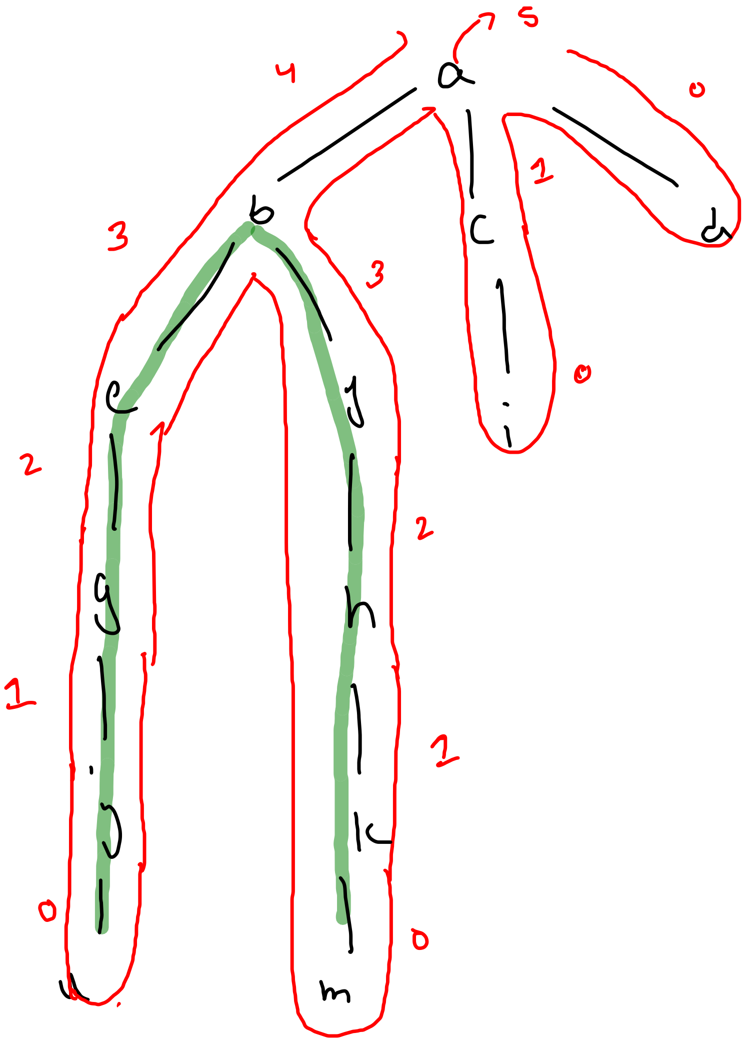
$\text{dist} = \underbrace{\text{bht} + \text{sbht}}_{\text{child}} + 2$



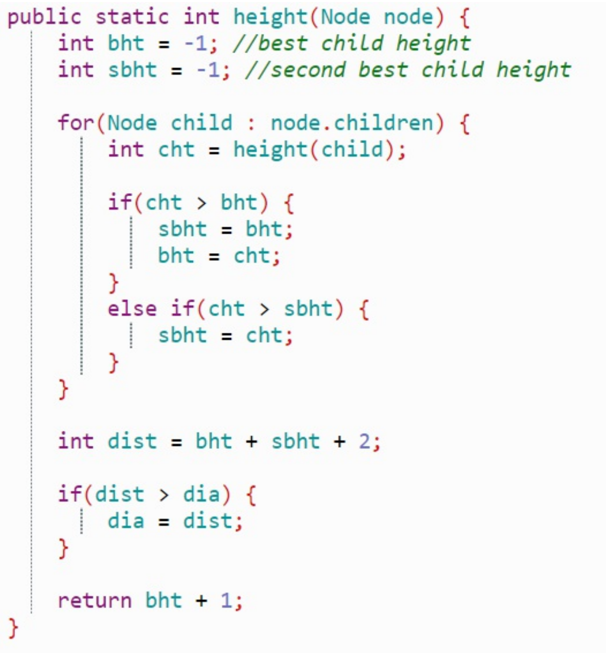
dia = ~~0~~ ~~2~~ ~~4~~ 7



```
public static int height(Node node) {
    int bht = -1; //best child height
    int sbht = -1; //second best child height

    for(Node child : node.children) {
        int cht = height(child);

        if(cht > bht) {
            sbht = bht;
            bht = cht;
        }
        else if(cht > sbht) {
            sbht = cht;
        }
    }

    int dist = bht + sbht + 2;

    if(dist > dia) {
        dia = dist;
    }

    return bht + 1;
}
```



dia = ~~0~~ ~~1~~ ~~2~~ ~~3~~ 8



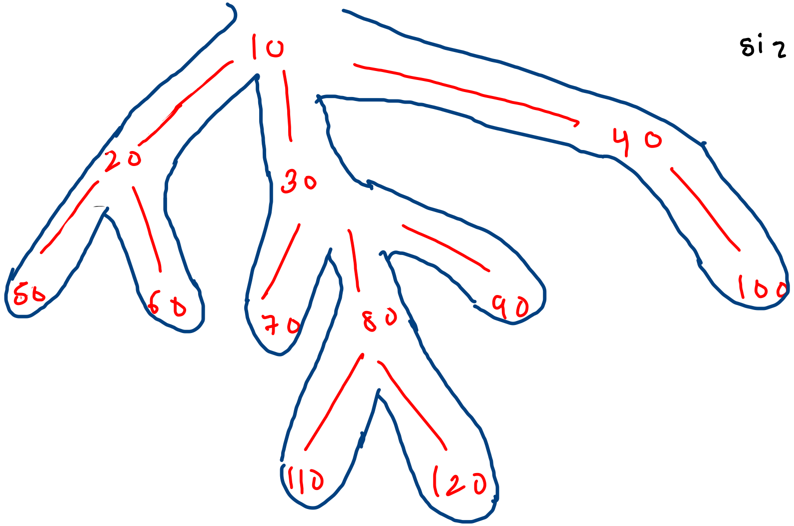
```java
public static int height(Node node) {
    int bht = -1; //best child height
    int sbht = -1; //second best child height

    for(Node child : node.children) {
        int cht = height(child);

        if(cht > bht) {
            sbht = bht;
            bht = cht;
        }
        else if(cht > sbht) {
            sbht = cht;
        }
    }

    int dist = bht + sbht + 2;

    if(dist > dia) {
        dia = dist;
    }

    return bht + 1;
}
```

## States

-1 : pre, state++

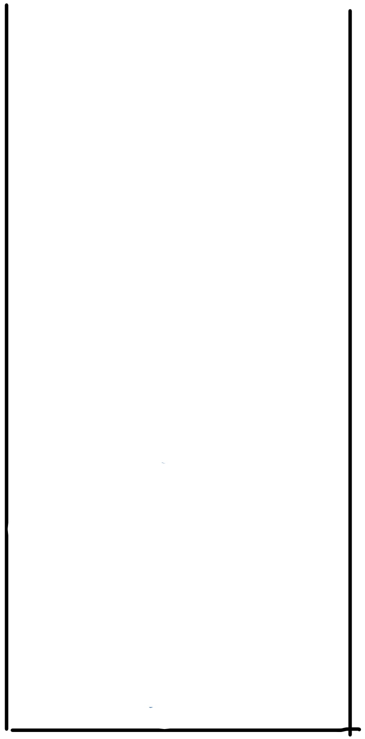
0 to size-1 : child push, state

size : post, pop

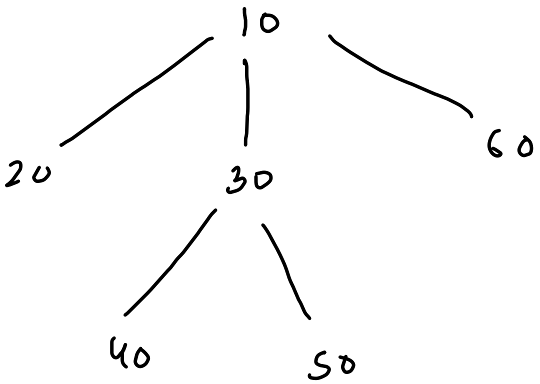


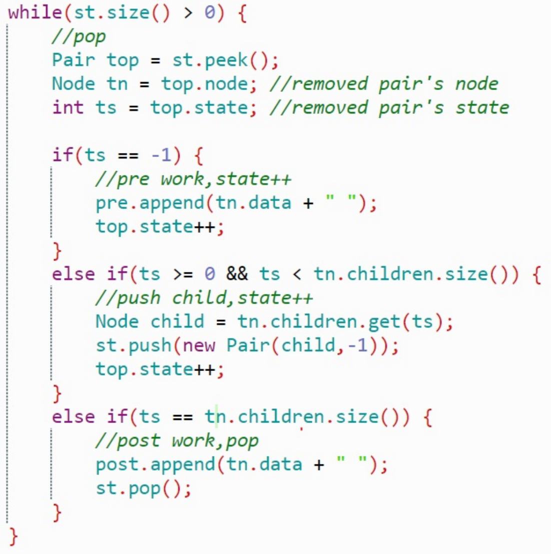
pre : 10 20 50 60 30 70 80 110 120 90 40 100

post : 50 60 20 70 110 120 80 90 30 100 40 10



Pair {
node,
state.
}

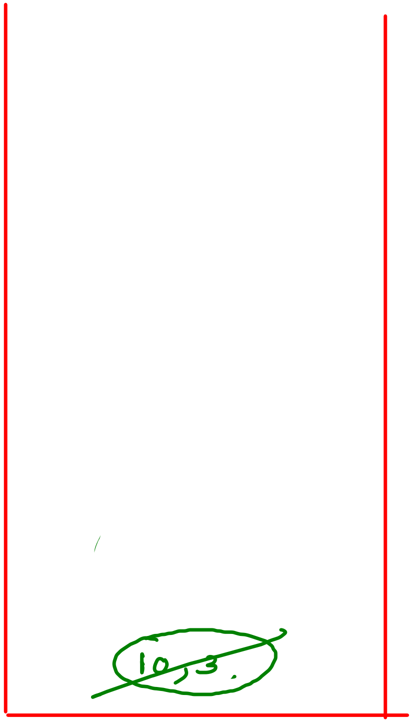




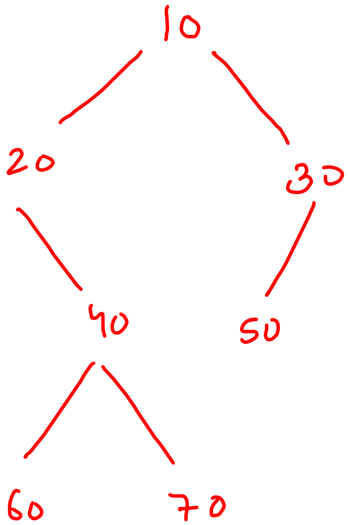
```
while(st.size() > 0) {
    //pop
    Pair top = st.peek();
    Node tn = top.node; //removed pair's node
    int ts = top.state; //removed pair's state

    if(ts == -1) {
        //pre work,state++
        pre.append(tn.data + " ");
        top.state++;
    }
    else if(ts >= 0 && ts < tn.children.size()) {
        //push child,state++
        Node child = tn.children.get(ts);
        st.push(new Pair(child,-1));
        top.state++;
    }
    else if(ts == tn.children.size()) {
        //post work,pop
        post.append(tn.data + " ");
        st.pop();
    }
}
```

pre : 10 20 30 40 50 60

post : 20 40 50 30 60 10

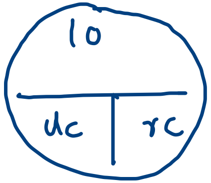


# Binary trees



child count -> atmost 2

0, 1, 2

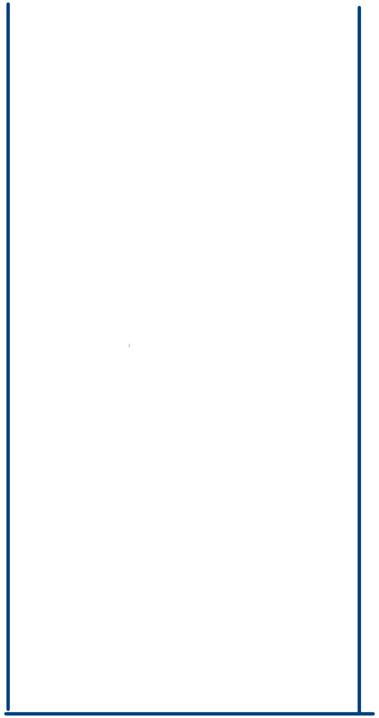


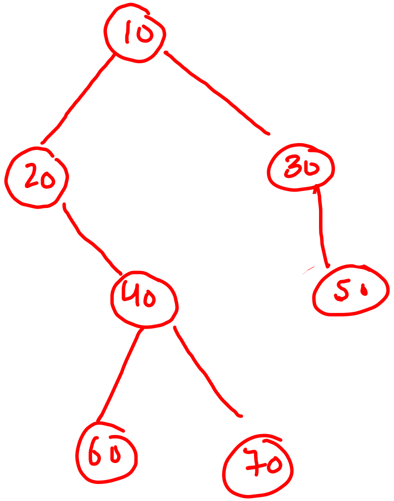
```
node {
      int data;
      Node left;
}     Node right;
```

## Construction

data : 10 20 -1 40 60 -1 -1 70 -1

-1 30 50 -1 -1 -1 .





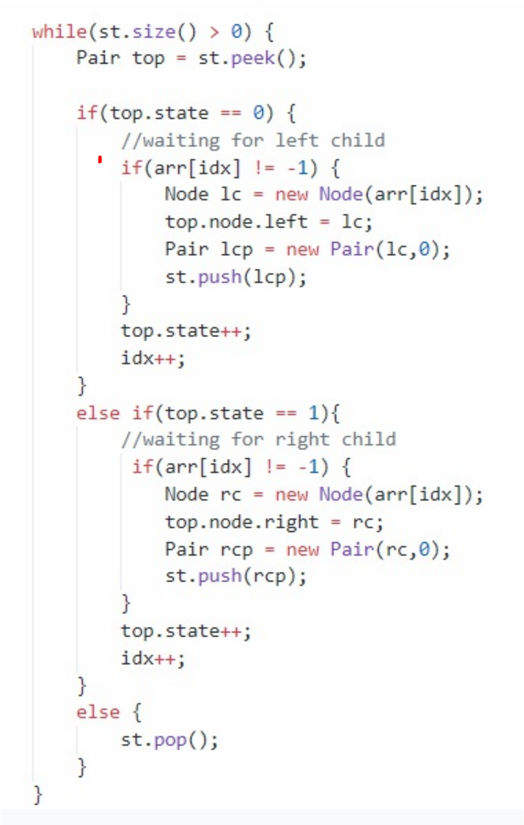
root = 10

0 -> waiting for left chil

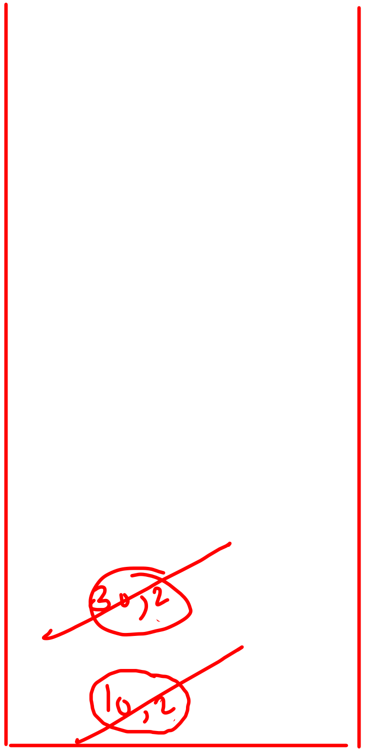
1 -> waiting for right child

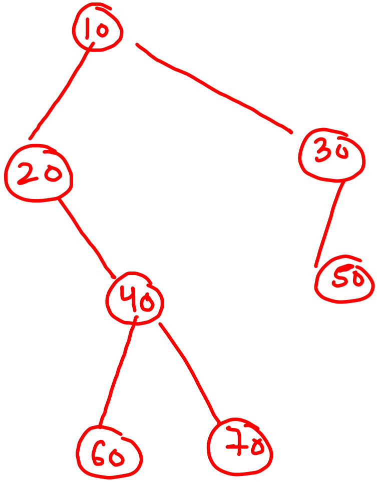
Pair {
node,
State
}

10 20 -1 40 60 -1 -1 70 -1 -1 30 50 -1 -1 -1 .

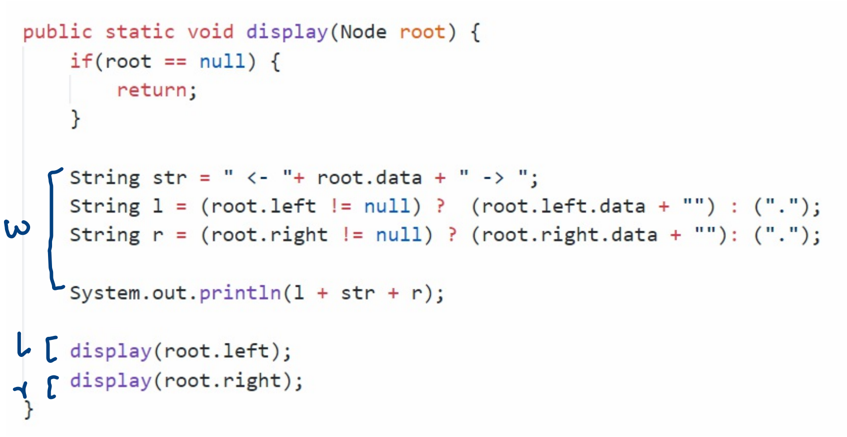


```
while(st.size() > 0) {
    Pair top = st.peek();

    if(top.state == 0) {
        //waiting for left child
        if(arr[idx] != -1) {
            Node lc = new Node(arr[idx]);
            top.node.left = lc;
            Pair lcp = new Pair(lc,0);
            st.push(lcp);
        }
        top.state++;
        idx++;
    }
    else if(top.state == 1){
        //waiting for right child
         if(arr[idx] != -1) {
            Node rc = new Node(arr[idx]);
            top.node.right = rc;
            Pair rcp = new Pair(rc,0);
            st.push(rcp);
        }
        top.state++;
        idx++;
    }
    else {
        st.pop();
    }
}
```

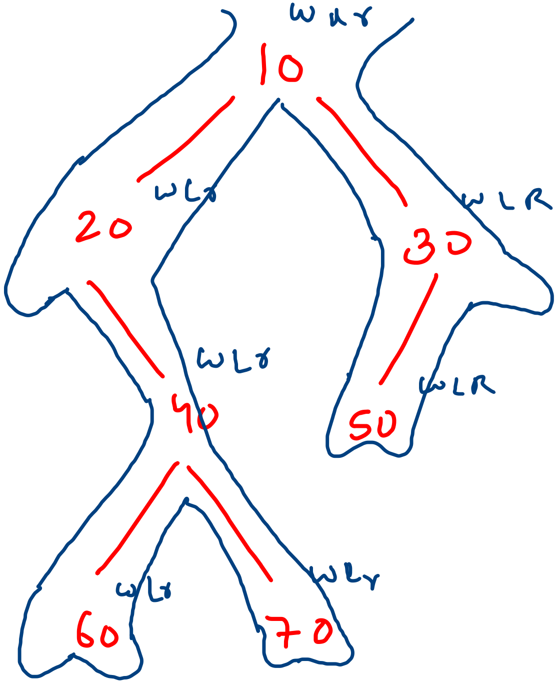




root = 10



```
public static void display(Node root) {
    if(root == null) {
        return;
    }

    String str = " <- "+ root.data + " -> ";
    String l = (root.left != null) ?  (root.left.data + "") : (".");
    String r = (root.right != null) ? (root.right.data + ""): (".");

    System.out.println(l + str + r);

    display(root.left);
    display(root.right);
}
```



20 ⟵ 10 ⟶ 30

. ⟵ 20 ⟶ 40

60 ⟵ 40 ⟶ 70

. ⟵ 60 ⟶ .

. ⟵ 70 ⟶ .

50 ⟵ 30 ⟶ .

. ⟵ 50 ⟶ .