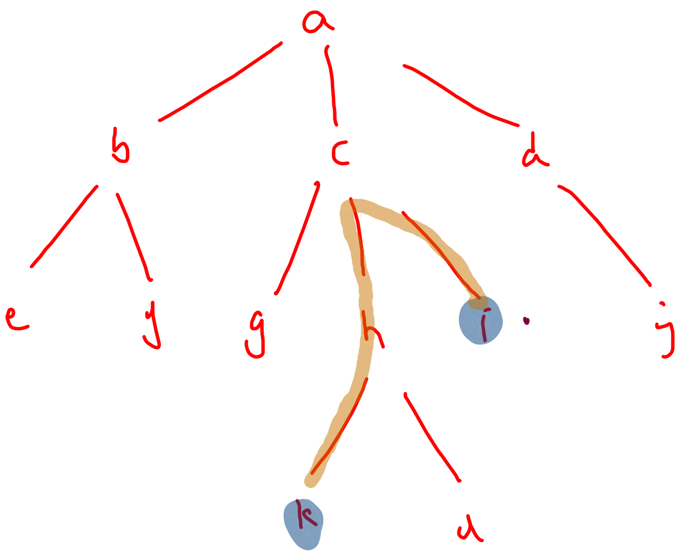
Distance Between Two Nodes In A Generic Tree



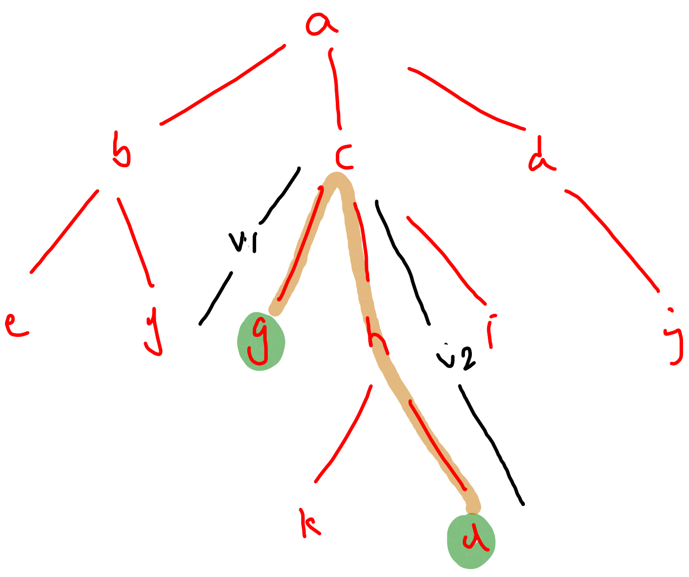
p1 : [i , c , a]
i ~~j~~ ~~j~~

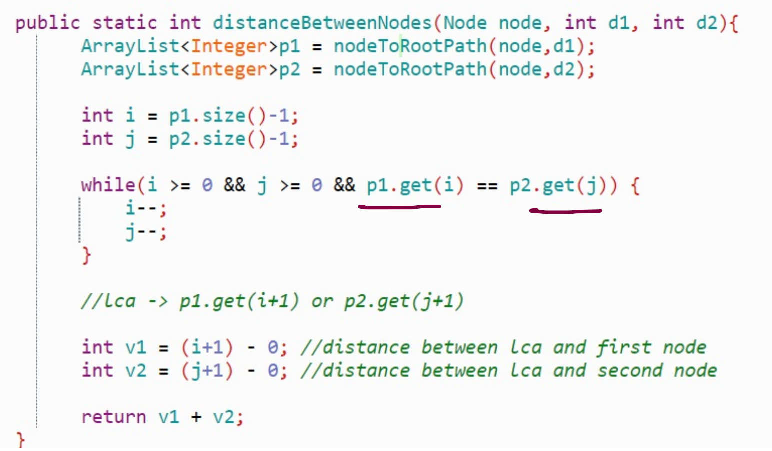
p2 : [k , h , c , a]
j ~~j~~ ~~j~~

d1 : i+1 → 1

d2 : j+1 → 2

ans: d1+d2





```
public static int distanceBetweenNodes(Node node, int d1, int d2){
        ArrayList<Integer>p1 = nodeToRootPath(node,d1);
        ArrayList<Integer>p2 = nodeToRootPath(node,d2);

        int i = p1.size()-1;
        int j = p2.size()-1;

        while(i >= 0 && j >= 0 && p1.get(i) == p2.get(j)) {
            i--;
            j--;
        }

        //lca -> p1.get(i+1) or p2.get(j+1)

        int v1 = (i+1) - 0; //distance between lca and first node
        int v2 = (j+1) - 0; //distance between lca and second node

        return v1 + v2;
}
```

→ lca

p1 = [g , c , a]

0 1 2

i ~~i~~ ~~i~~

v1 = 1

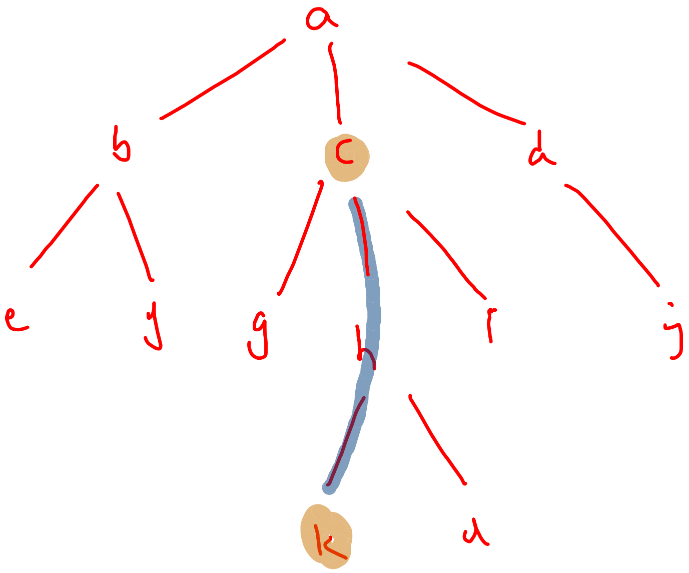
v2 = 2

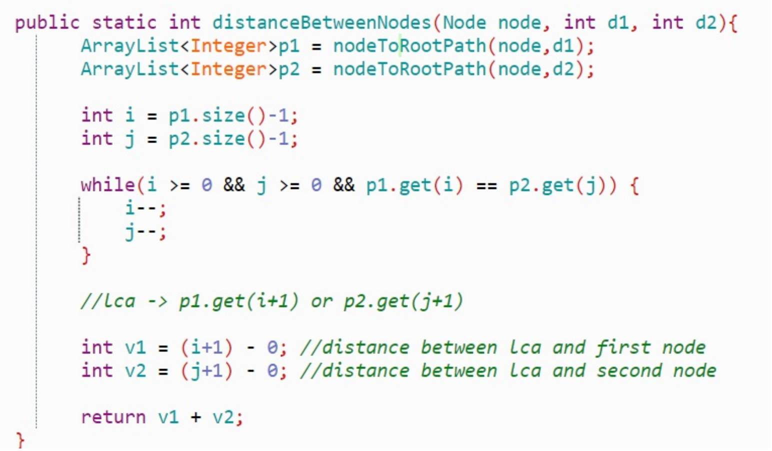
p2 = [l , h , c , a]

0 1 2 3

j ~~j~~ ~~j~~

ans : 3





```java
public static int distanceBetweenNodes(Node node, int d1, int d2){
        ArrayList<Integer>p1 = nodeToRootPath(node,d1);
        ArrayList<Integer>p2 = nodeToRootPath(node,d2);

        int i = p1.size()-1;
        int j = p2.size()-1;

        while(i >= 0 && j >= 0 && p1.get(i) == p2.get(j)) {
            i--;
            j--;
        }

        //lca -> p1.get(i+1) or p2.get(j+1)

        int v1 = (i+1) - 0; //distance between lca and first node
        int v2 = (j+1) - 0; //distance between lca and second node

        return v1 + v2;
}
```

i ~~i~~ ~~i~~

p1 : [c , a]

0 1

j ~~j~~ ~~j~~

p2 : [k , h , c , a]

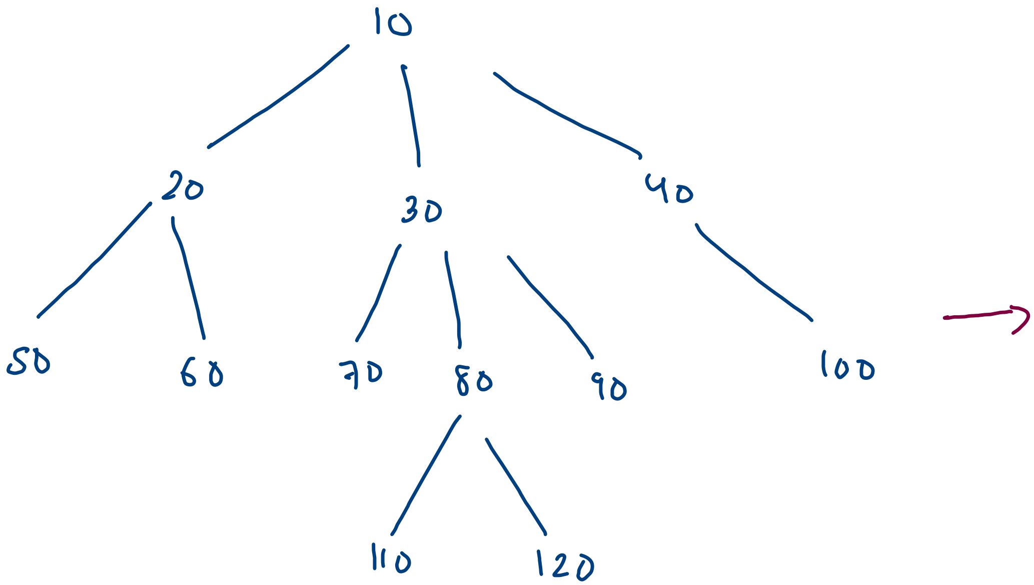
0 1 2 3

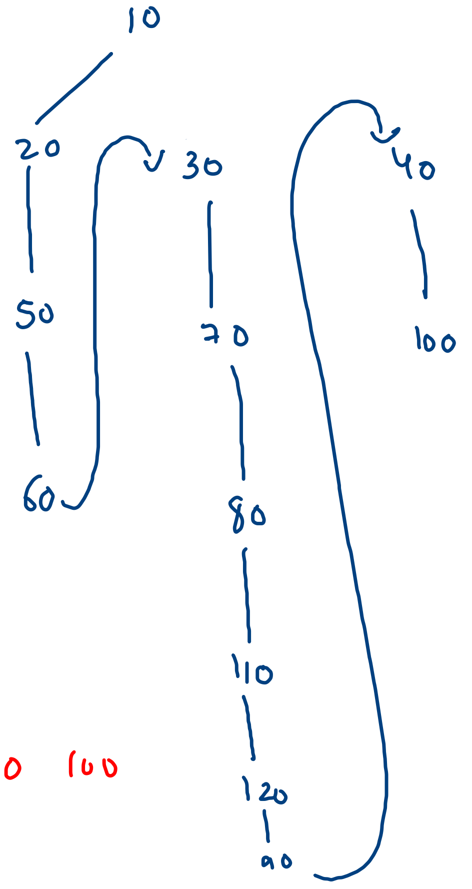
v1 = 0

v2 = 2

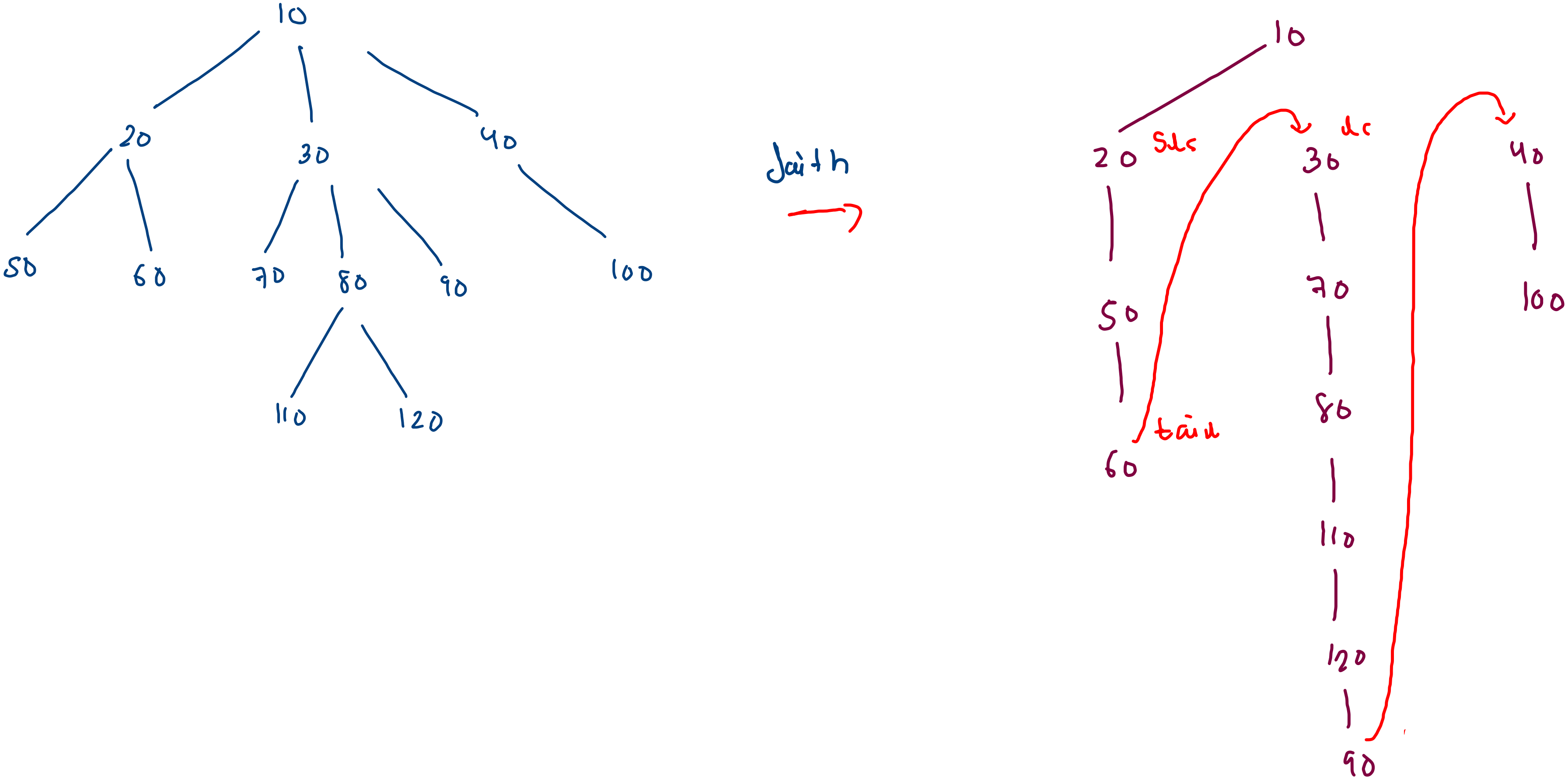
ans: 2

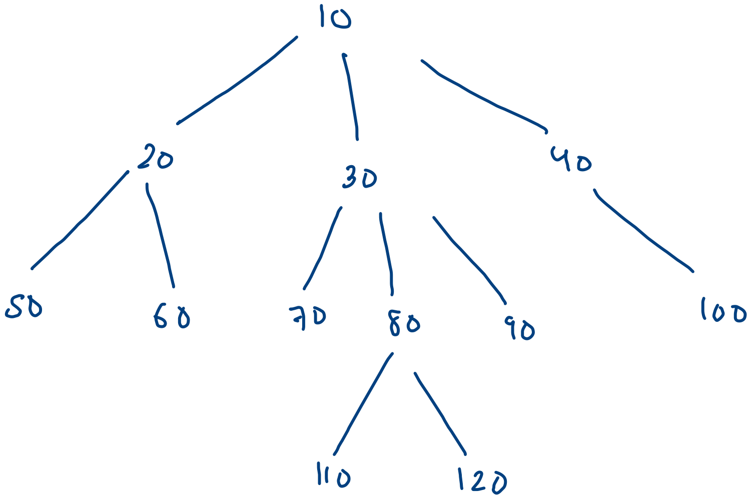
Linearize A Generic Tree

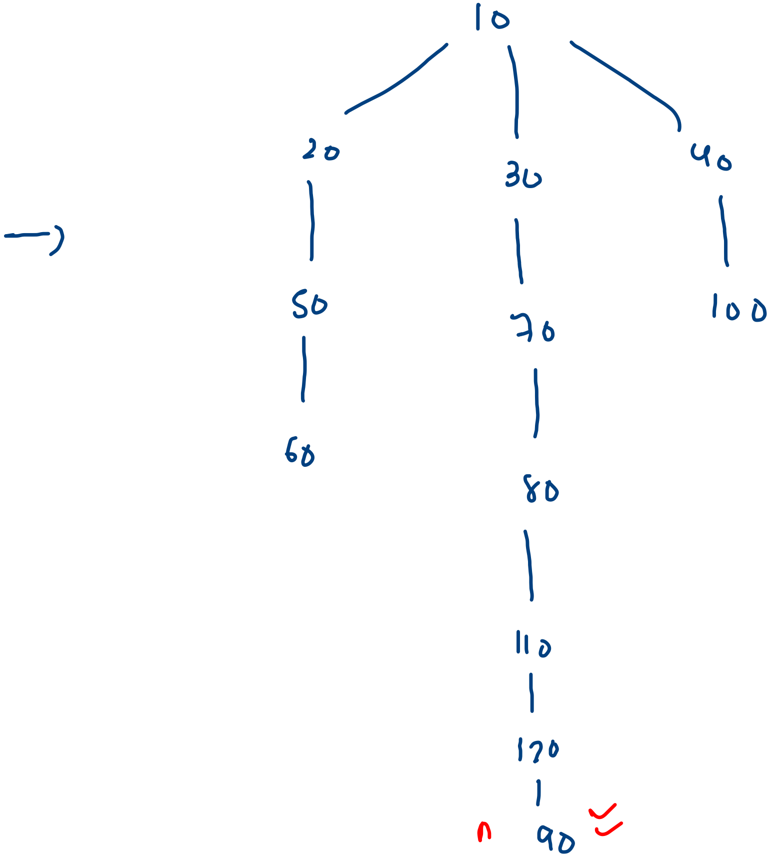


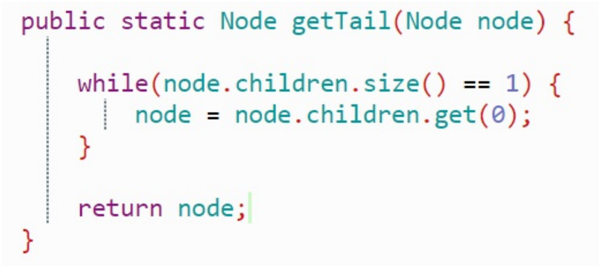


Pre: 10 20 50 60 30 70 80 110 120 90 40 100

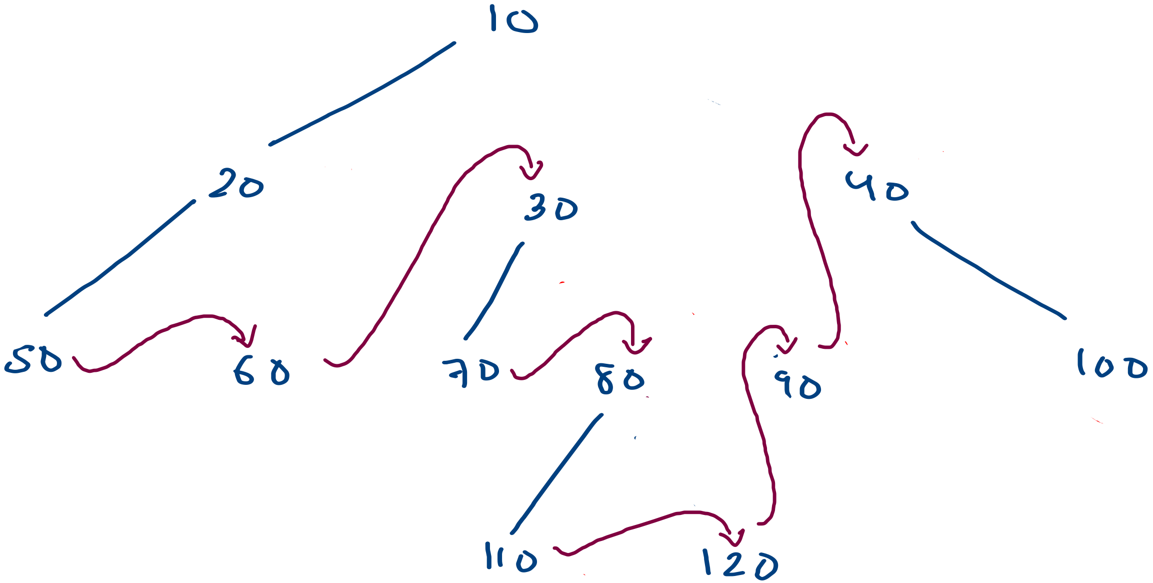

10
20
30
40
50
60
70
80
90
100
110
120
Jaith
10
20
Slc
30
dc
40
50
70
100
80
tail
60
110
120
90

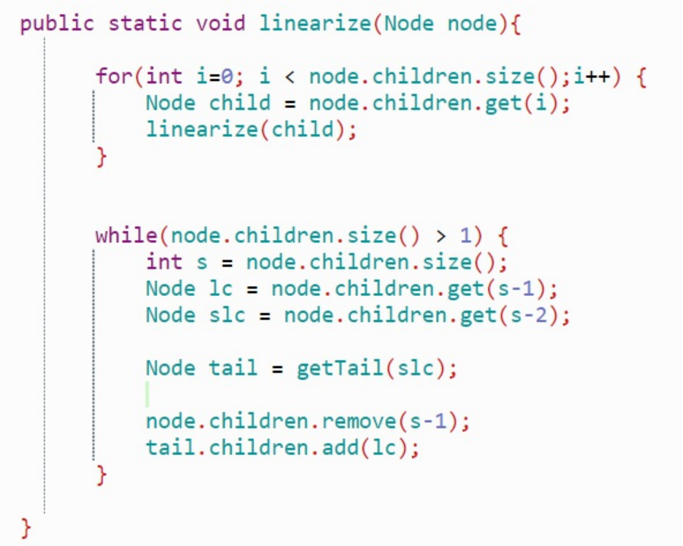




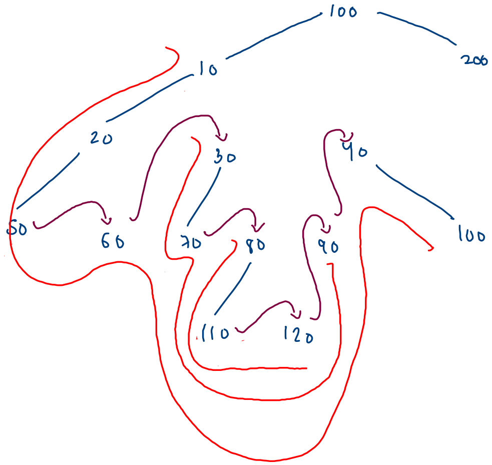


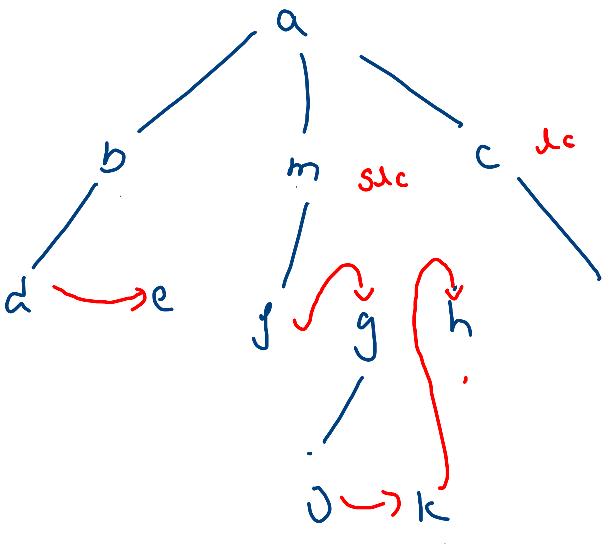
```
public static Node getTail(Node node) {

    while(node.children.size() == 1) {
        node = node.children.get(0);
    }

    return node;
}
```

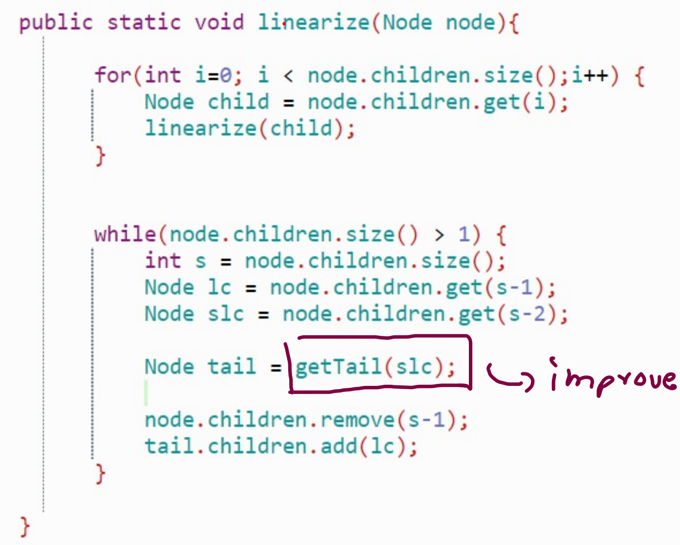




```
public static void linearize(Node node){

    for(int i=0; i < node.children.size();i++) {
        Node child = node.children.get(i);
        linearize(child);
    }


    while(node.children.size() > 1) {
        int s = node.children.size();
        Node lc = node.children.get(s-1);
        Node slc = node.children.get(s-2);

        Node tail = getTail(slc);

        node.children.remove(s-1);
        tail.children.add(lc);
    }

}
```





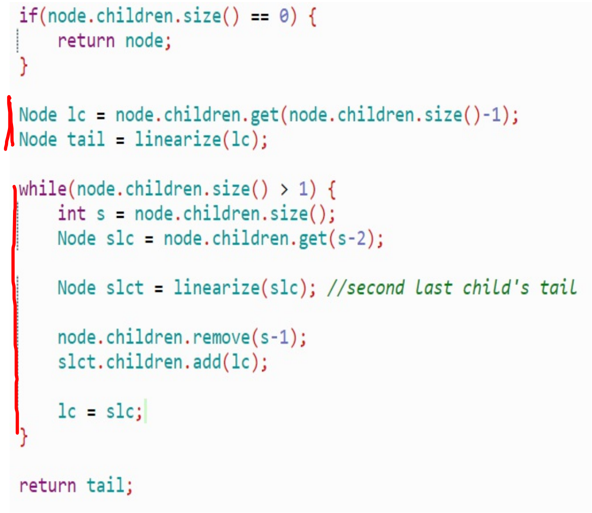


```
public static void linearize(Node node){

    for(int i=0; i < node.children.size();i++) {
        Node child = node.children.get(i);
        linearize(child);
    }


    while(node.children.size() > 1) {
        int s = node.children.size();
        Node lc = node.children.get(s-1);
        Node slc = node.children.get(s-2);

        Node tail = getTail(slc);

        node.children.remove(s-1);
        tail.children.add(lc);
    }
}
```

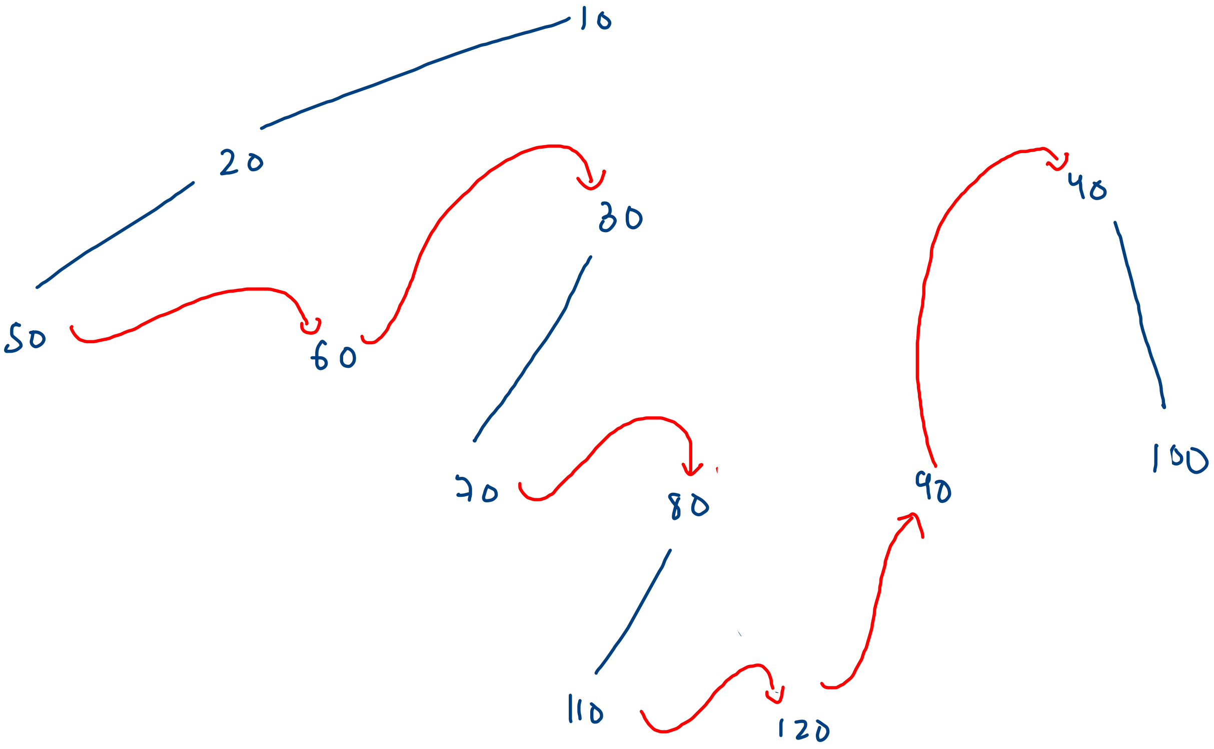
improve

100

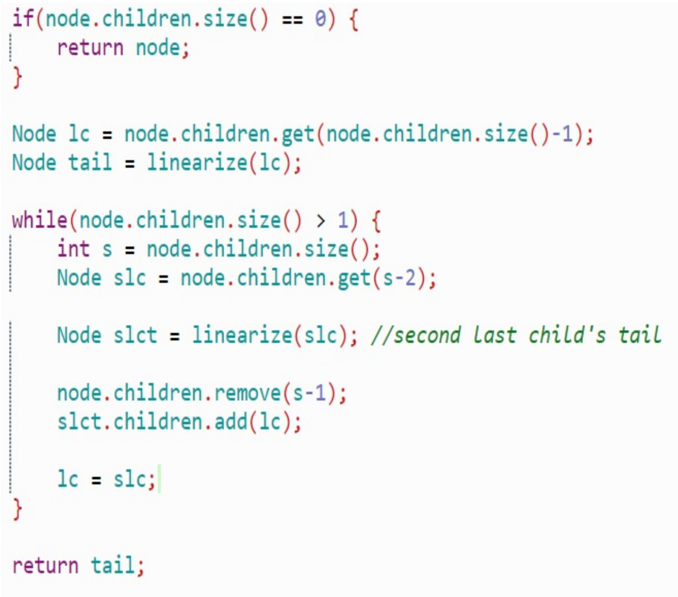
improvised



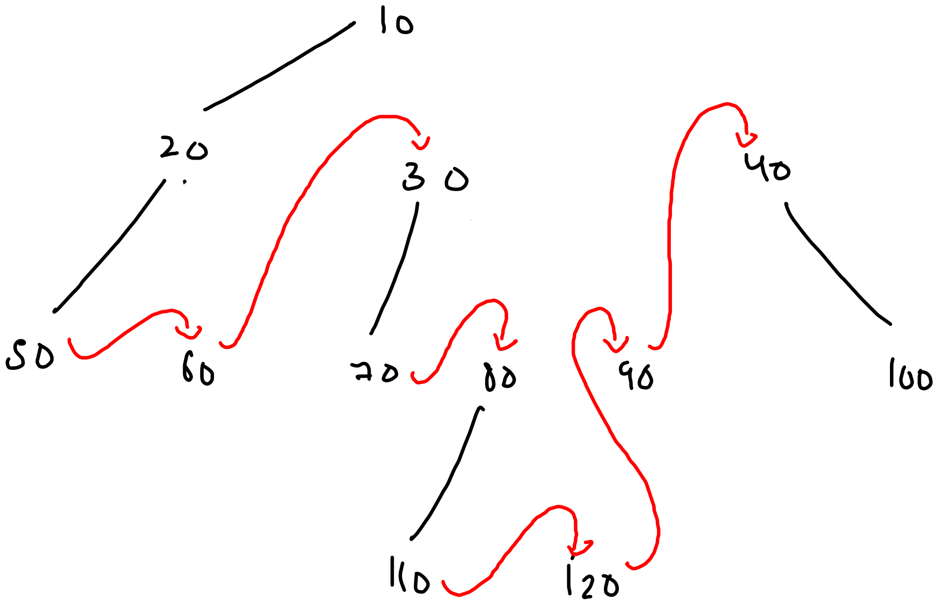
```
if(node.children.size() == 0) {
    return node;
}

Node lc = node.children.get(node.children.size()-1);
Node tail = linearize(lc);

while(node.children.size() > 1) {
    int s = node.children.size();
    Node slc = node.children.get(s-2);

    Node slct = linearize(slc); //second last child's tail

    node.children.remove(s-1);
    slct.children.add(lc);

    lc = slc;
}

return tail;
```



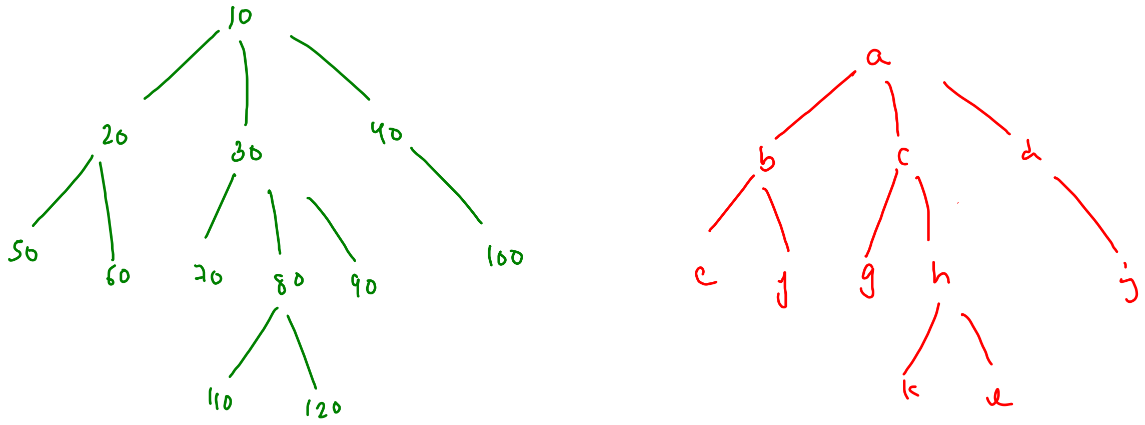
o(n)



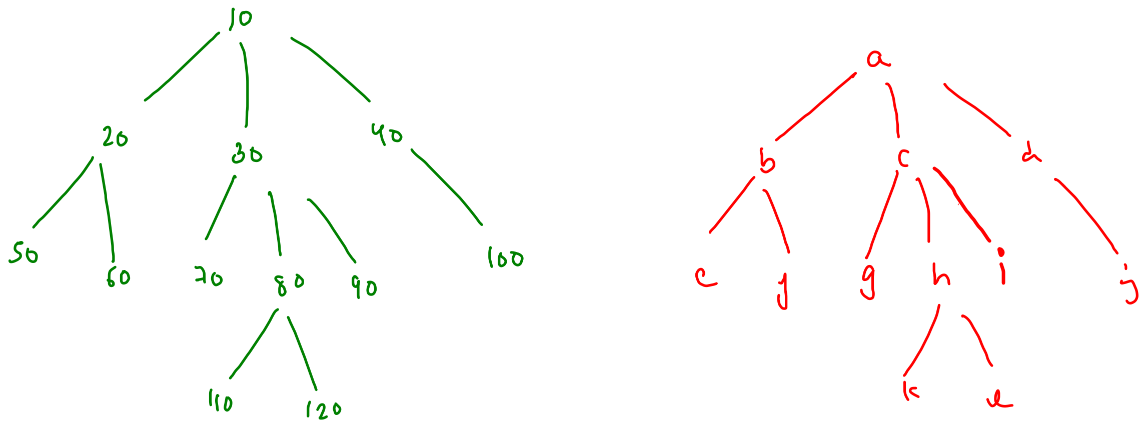
```
if(node.children.size() == 0) {
    return node;
}

Node lc = node.children.get(node.children.size()-1);
Node tail = linearize(lc);

while(node.children.size() > 1) {
    int s = node.children.size();
    Node slc = node.children.get(s-2);

    Node slct = linearize(slc); //second last child's tail

    node.children.remove(s-1);
    slct.children.add(lc);

    lc = slc;
}

return tail;
```



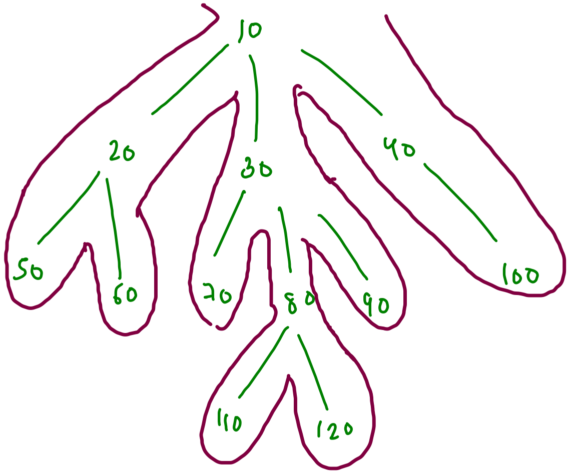
Are Trees Similar In Shape

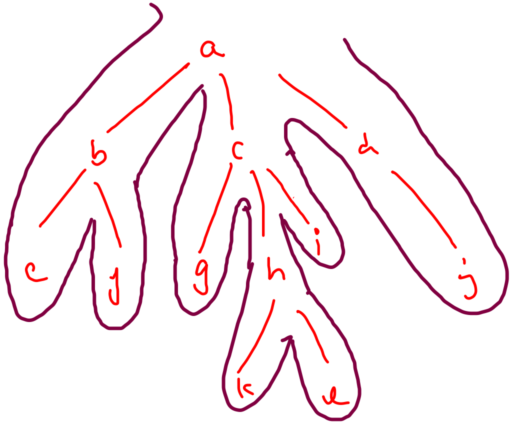


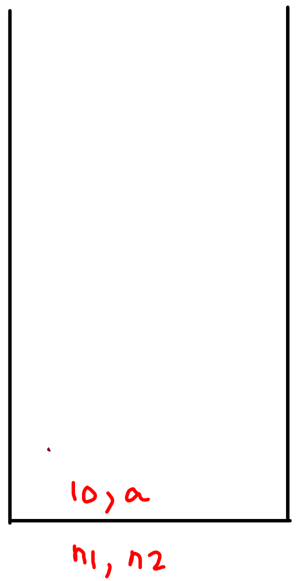
→ false

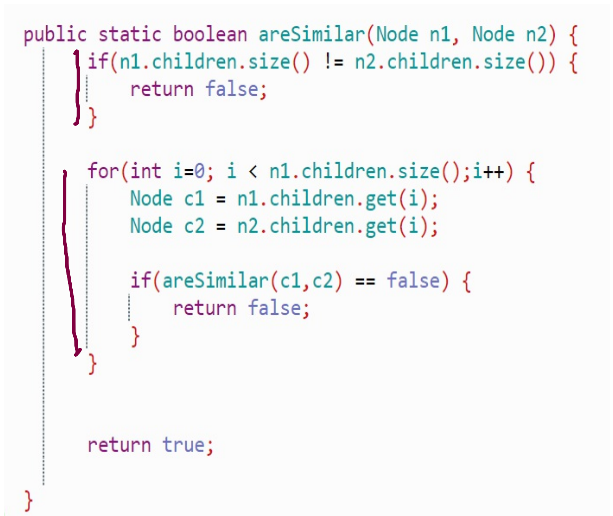


→ true



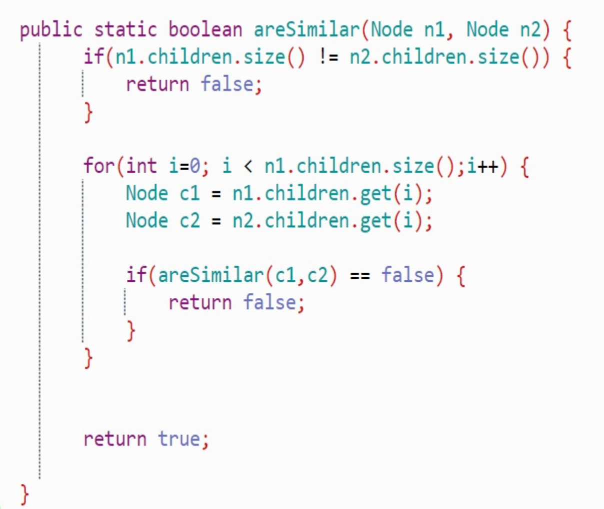




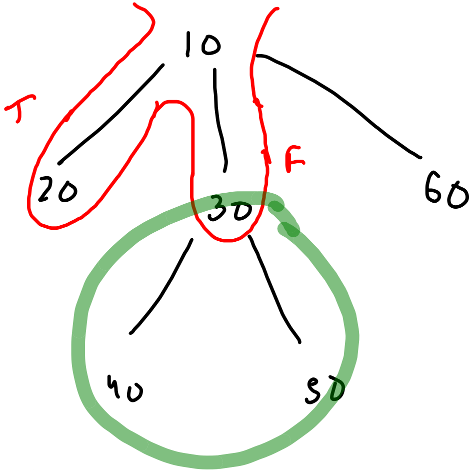


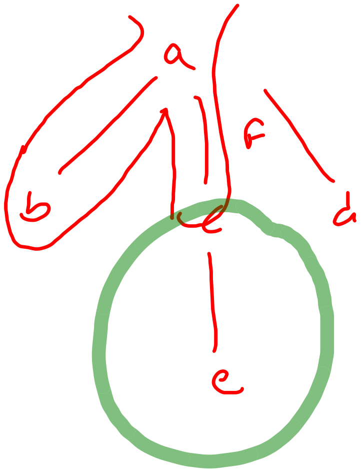
```
public static boolean areSimilar(Node n1, Node n2) {
    if(n1.children.size() != n2.children.size()) {
        return false;
    }

    for(int i=0; i < n1.children.size();i++) {
        Node c1 = n1.children.get(i);
        Node c2 = n2.children.get(i);

        if(areSimilar(c1,c2) == false) {
            return false;
        }
    }


    return true;

}
```

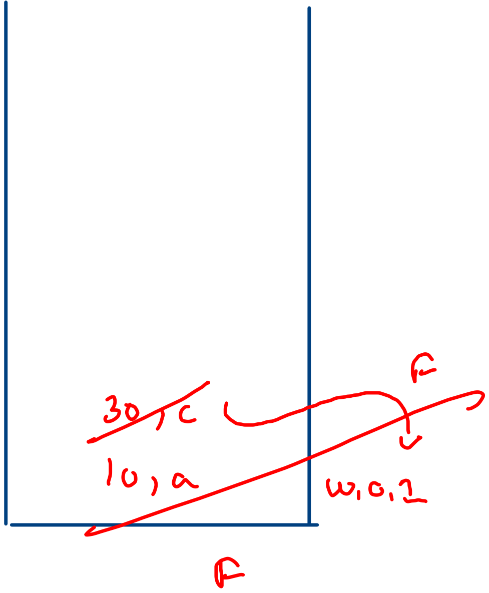
10, a → 20, b   30 , c   40, d



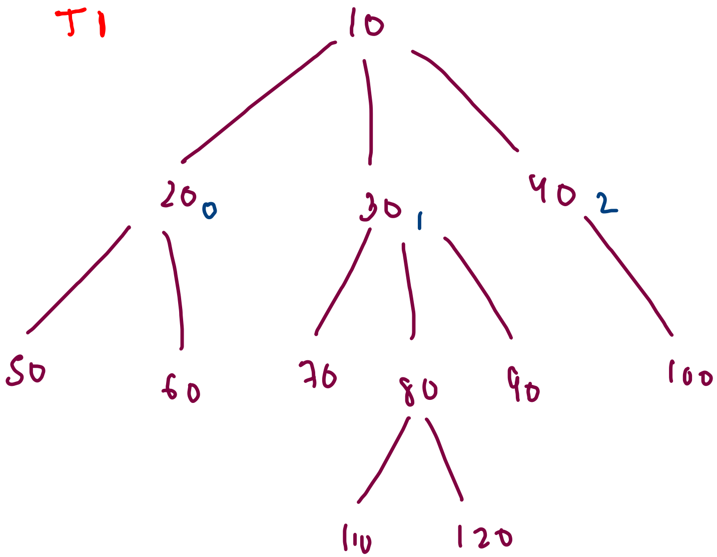
```
public static boolean areSimilar(Node n1, Node n2) {
    if(n1.children.size() != n2.children.size()) {
        return false;
    }

    for(int i=0; i < n1.children.size();i++) {
        Node c1 = n1.children.get(i);
        Node c2 = n2.children.get(i);

        if(areSimilar(c1,c2) == false) {
            return false;
        }
    }


    return true;
}
```

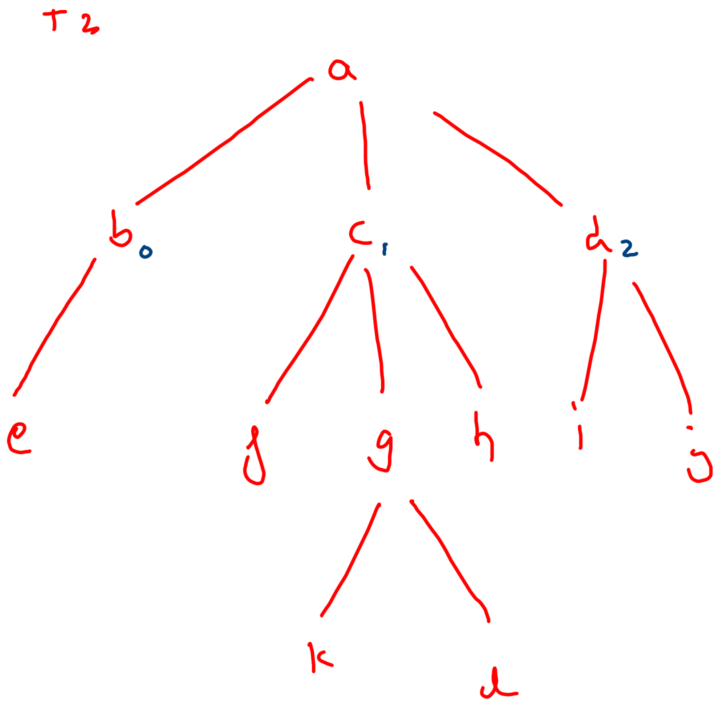


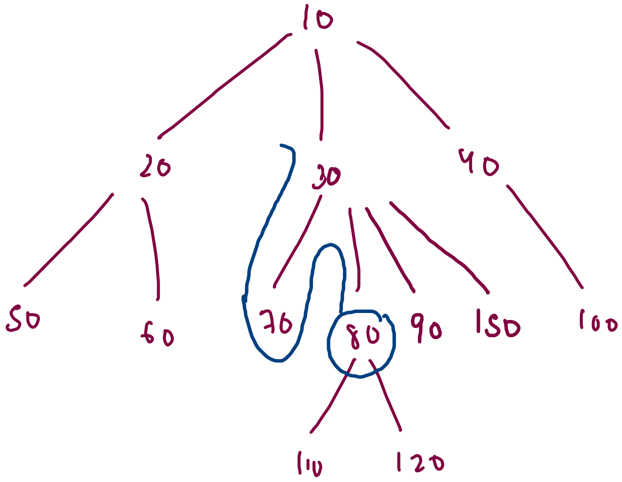


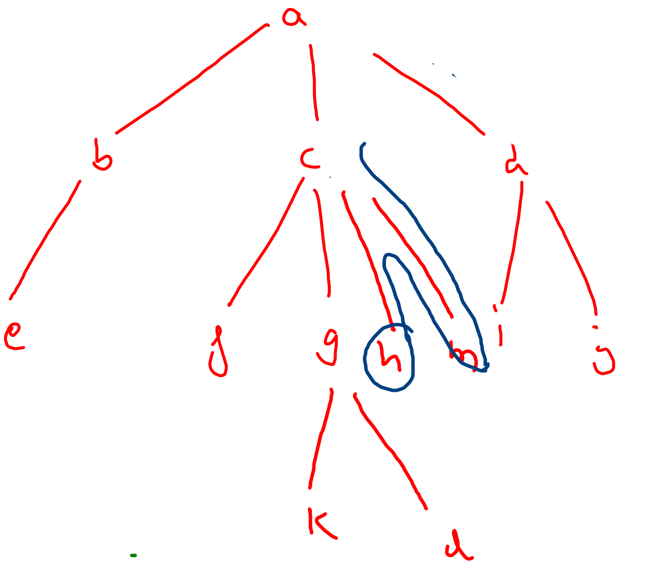


Are Trees Mirror In Shape

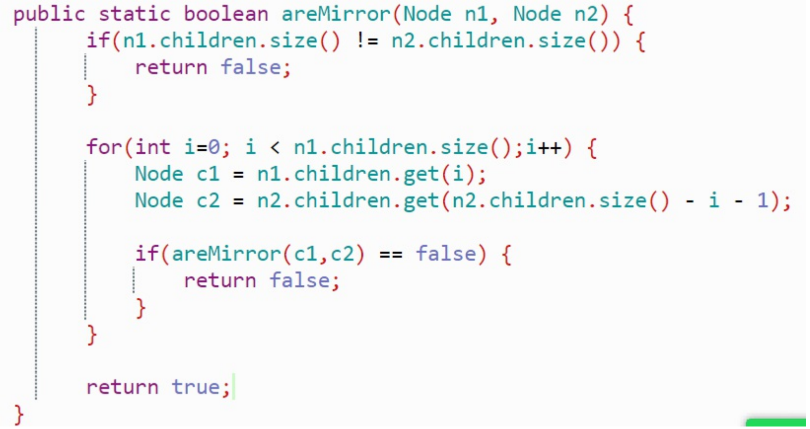








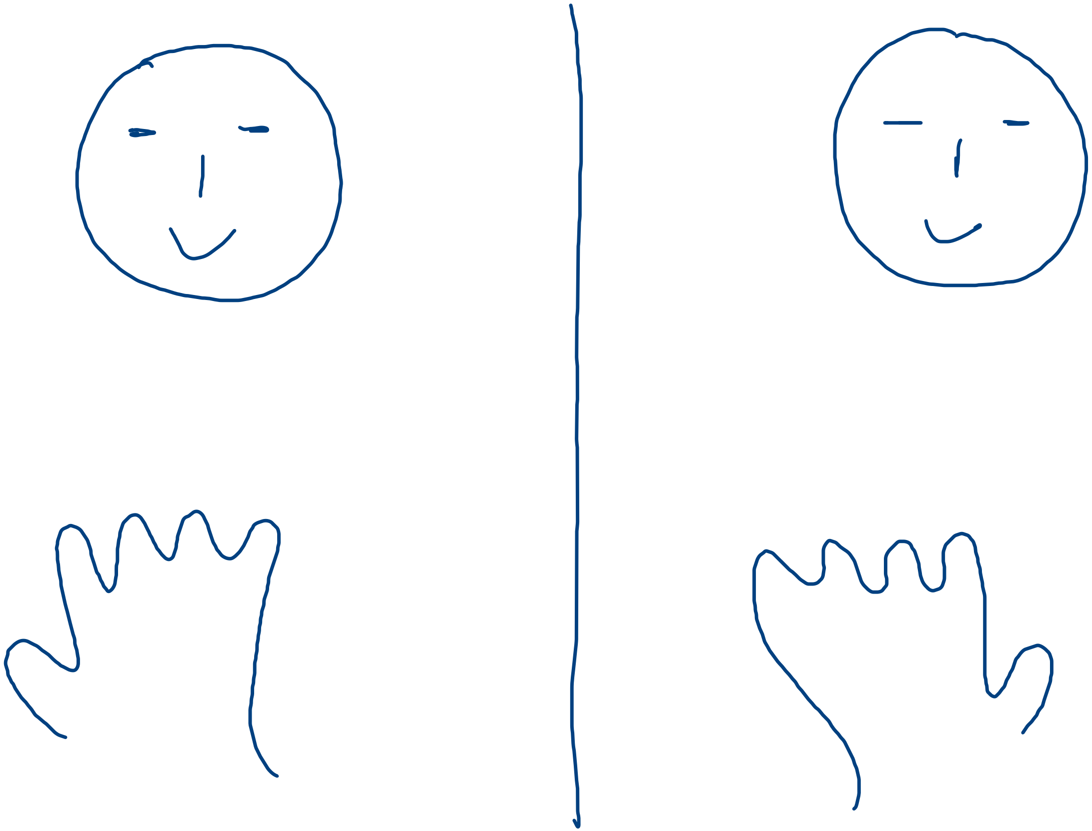
i=0



```
public static boolean areMirror(Node n1, Node n2) {
    if(n1.children.size() != n2.children.size()) {
        return false;
    }

    for(int i=0; i < n1.children.size();i++) {
        Node c1 = n1.children.get(i);
        Node c2 = n2.children.get(n2.children.size() - i - 1);

        if(areMirror(c1,c2) == false) {
            return false;
        }
    }

    return true;
}
```

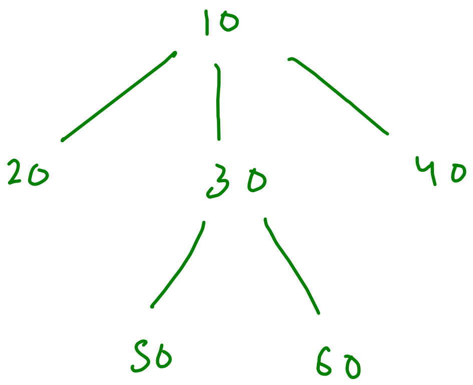
10, a → 20, d   30, c   40, b

Is Generic Tree Symmetric

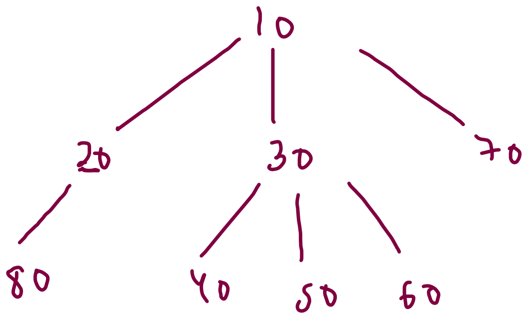


symmetric

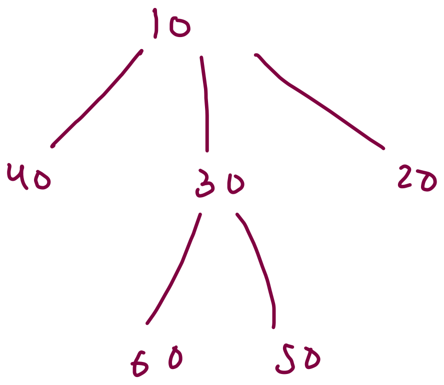
asymmetric

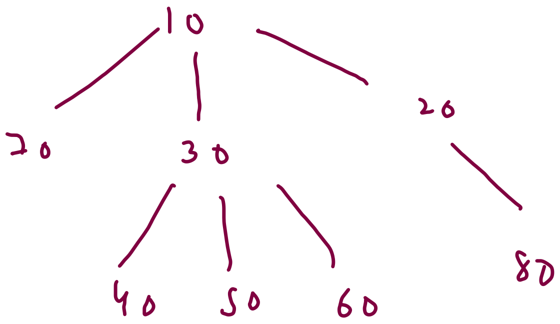

10
20
30
40
50
60

symm


10
20
30
70
80
40
50
60

assym


10
40
30
20
60
50


10
70
30
20
40
50
60
80

isSymmetric (node) → areMirror (node, node)

areMirror (T1, T2)

true : mirror (T1) = T2

areMirror (T1, T1)

true : mirror(T1) = T1

(proves T1 is a symmetric shape)