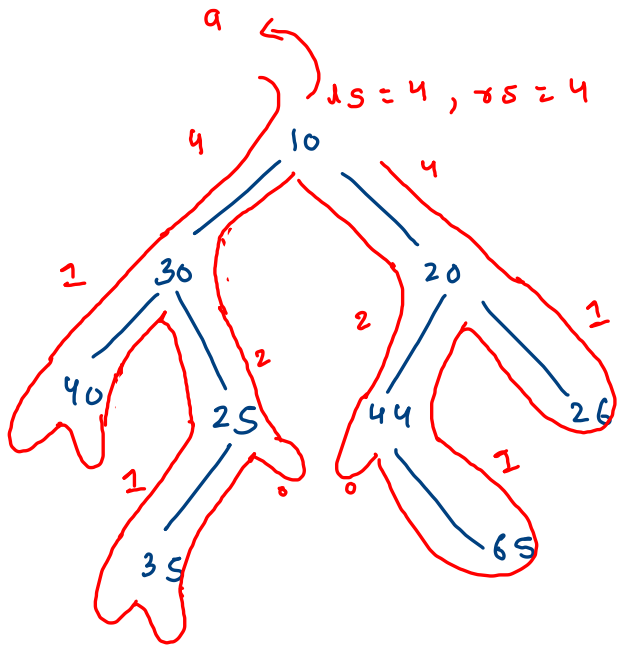
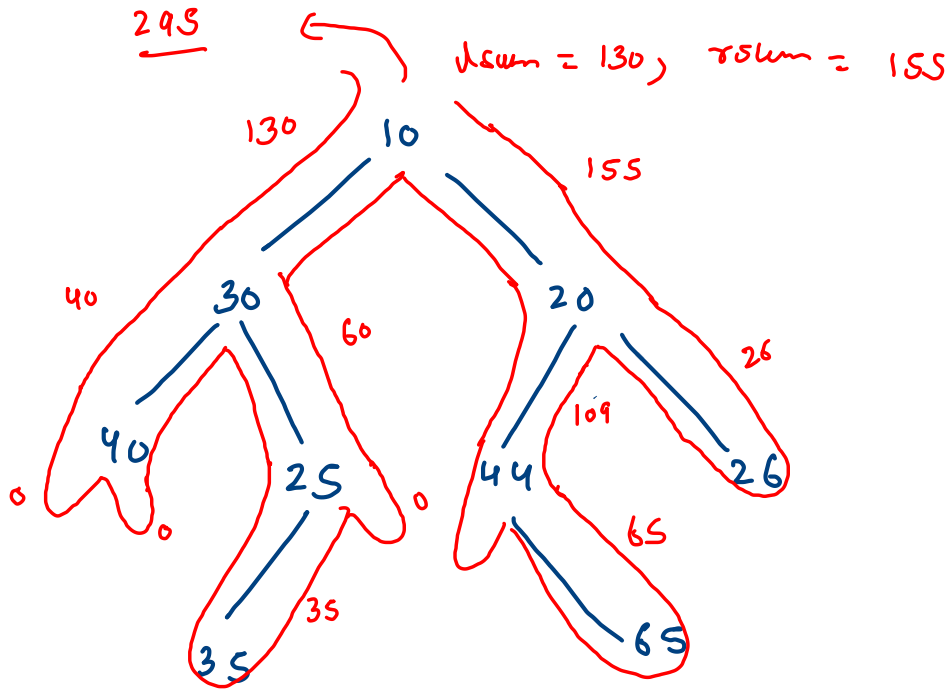
size, sum, max, height

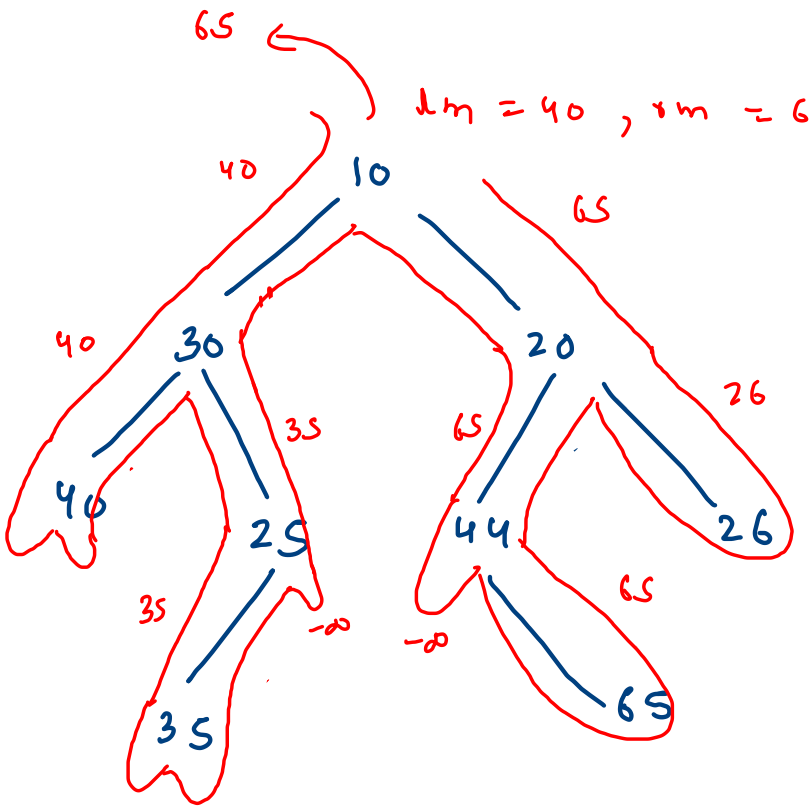


size

```
size {
    if (root == null) return 0;

    int ls = size(node.left);

    int rs = size(node.right);

    return ls + rs + 1;
}
```

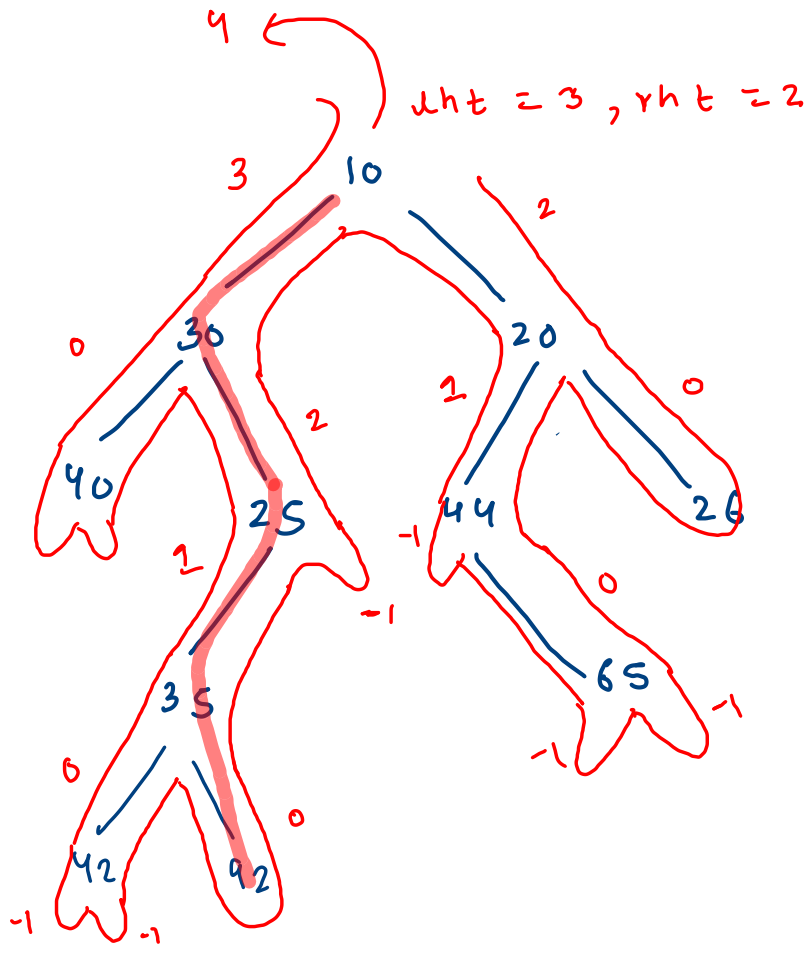


lsum + rsum + node.data

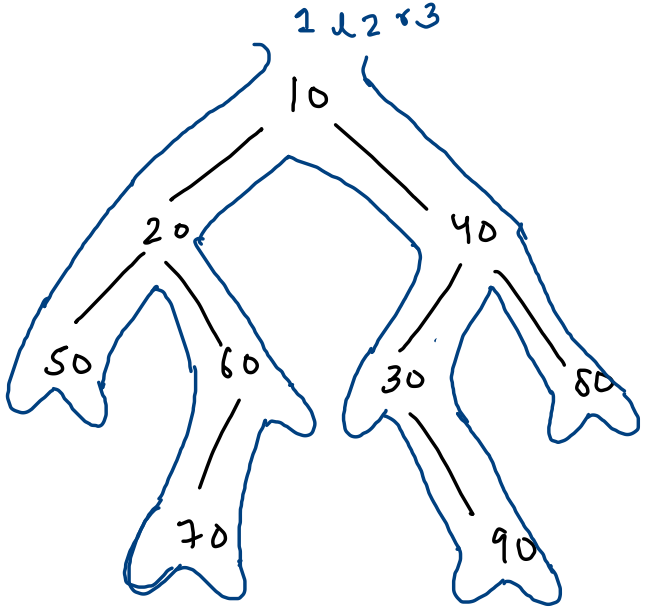


$$max = \begin{cases} max(left), \\ max(right), \\ node.data \end{cases}$$

Hight




```
if ( node == null ) {
    return -1;   // edges terms
}
lht = height(left);
rht = height(right);
int ht = max(lht, rht) + 1
```




```
if(node.==null)
    return;
3
1 [ pre += node.data;
left call  travel(node.left);
2 [ in += node.data
right call travel(node.right);
3 [ post += node.data;
```

pre : 10 20 50 60 70 40 30 90 80

in : 50 20 70 60 10 30 90 40 80

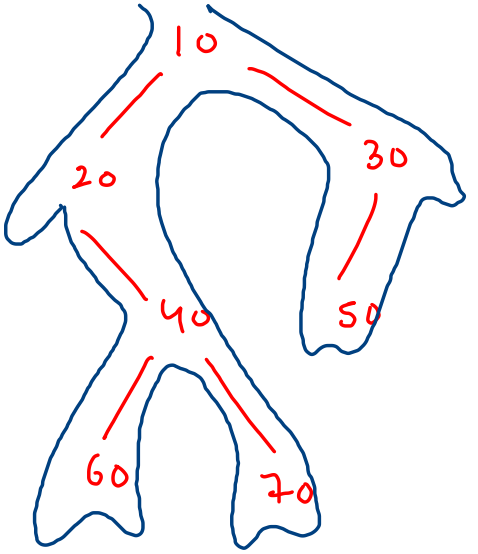
post : 50 70 60 20 90 30 80 40 10

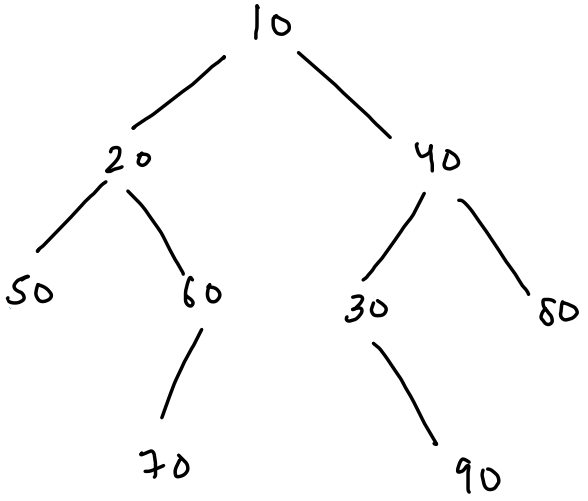
pre 10 20 40 60 70 30 50

in 20 60 40 70 10 50 30

post 60 70 40 20 50 30 10

```
int[]arr = {10,20,-1,40,60,-1,-1,70,-1,-1,30,50,-1,-1,-1};
```

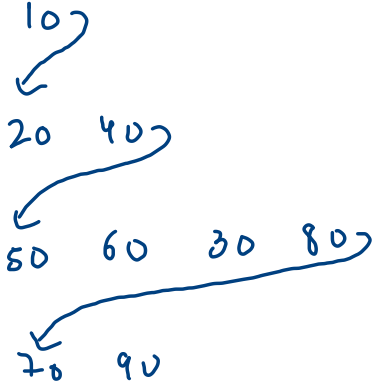




level - order way :

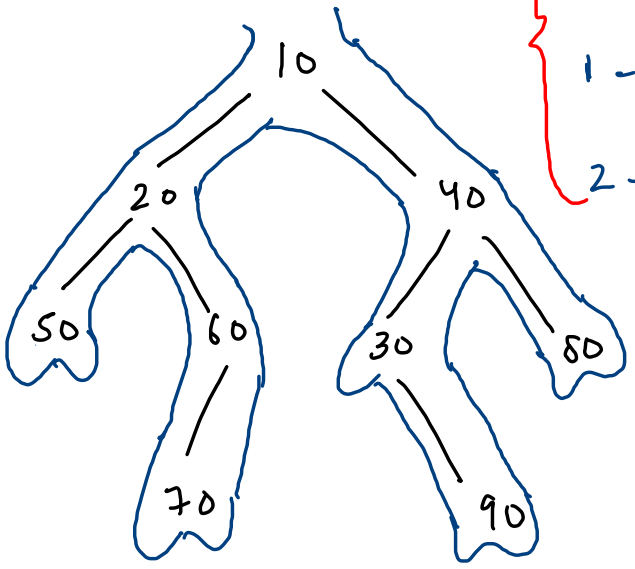
count = 2

~~10~~	~~20~~	~~40~~	~~50~~	~~60~~	~~30~~	~~80~~	~~70~~	~~90~~



count times

- remove
- work
- add children



0 -> pre, push left child, state++

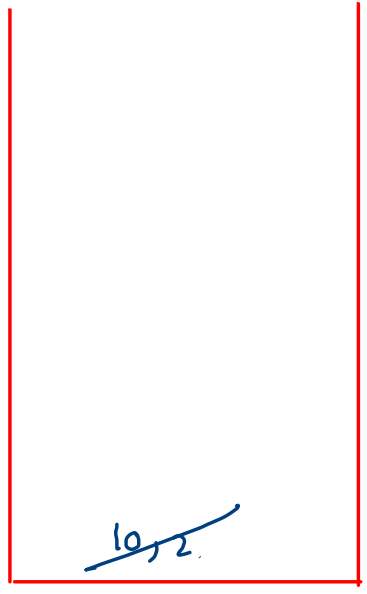
1 -> in, push right child, state++

2 -> post, pop

pre : 10 20 50 60 70 40 30 90 80

in : 50 20 70 60 10 30 90 40 80

post: 50 70 60 20 90 30 80 40 10

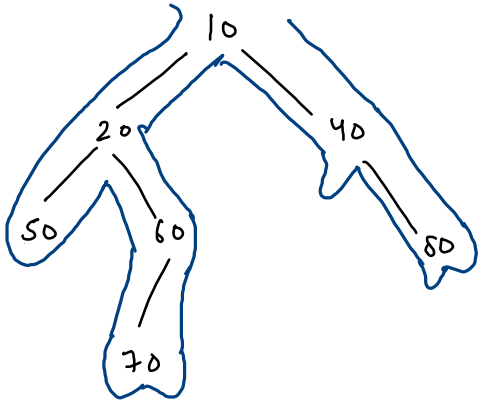


Pair {

node,

state

}

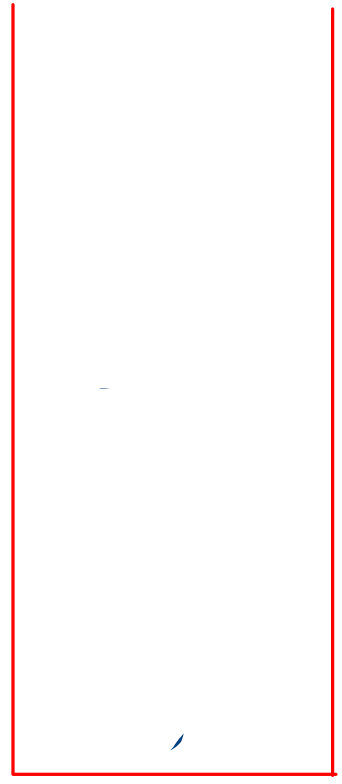



```
while(st.size() > 0) {
    Pair top = st.peek();

    if(top.state == 0) {
       //pre,push left child,state++
       pre += (top.node.data + " ");

       if(top.node.left != null) {
            Pair lcp = new Pair(top.node.left,0);
            st.push(lcp);
       }

       top.state++;
    }
    else if(top.state == 1) {
        //in, push right child,state++
        in += (top.node.data + " ");

        if(top.node.right != null) {
            Pair rcp = new Pair(top.node.right,0);
            st.push(rcp);
        }

        top.state++;
    }
    else if(top.state == 2) {
        //post, pop
        post += (top.node.data + " ");
        st.pop();
    }
}
```

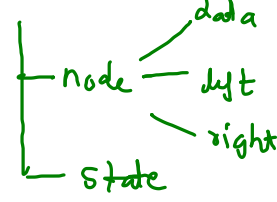
pre : 10 20 50 60 70 40 80

in : 50 20 70 60 10 40 80

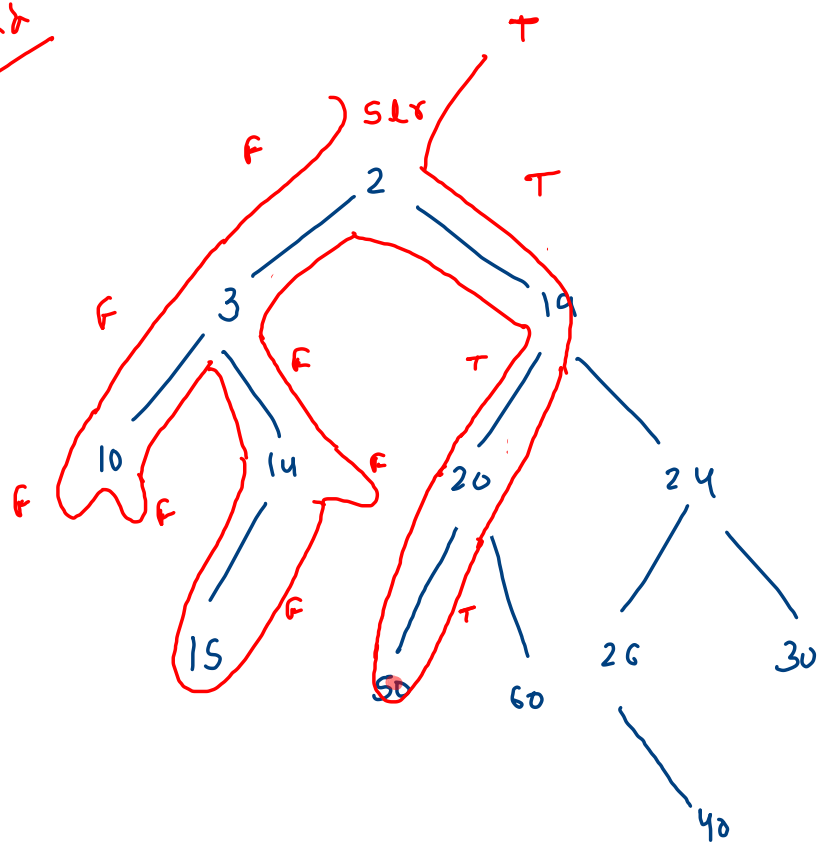
post : 50 70 60 20 80 40 10



Stack -> Pair



Find



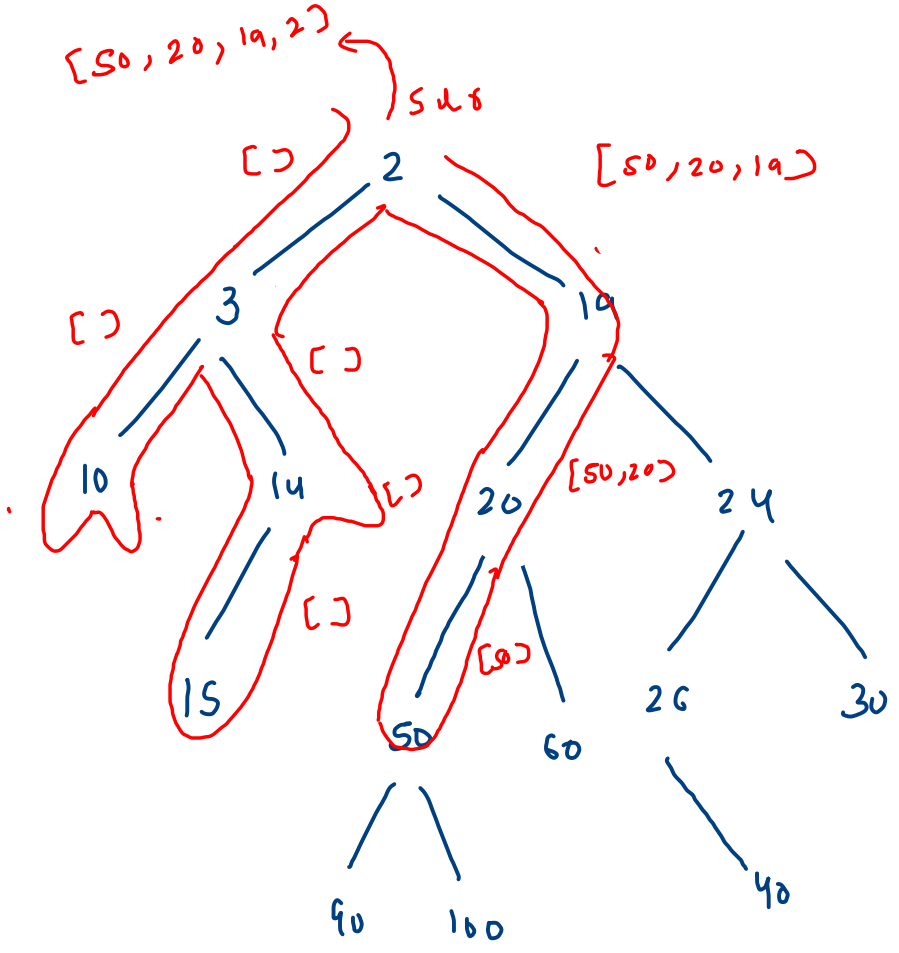
data = 50

```
public static boolean find(Node node, int data){
    if(node == null) {
        return false;
    }

    if(node.data == data) {
        return true;
    }

    boolean lans = find(node.left,data);

    if(lans == true) {
        return true;
    }

    boolean rans = find(node.right,data);

    if(rans == true) {
        return true;
    }

    return false;
}
```

S

L

R

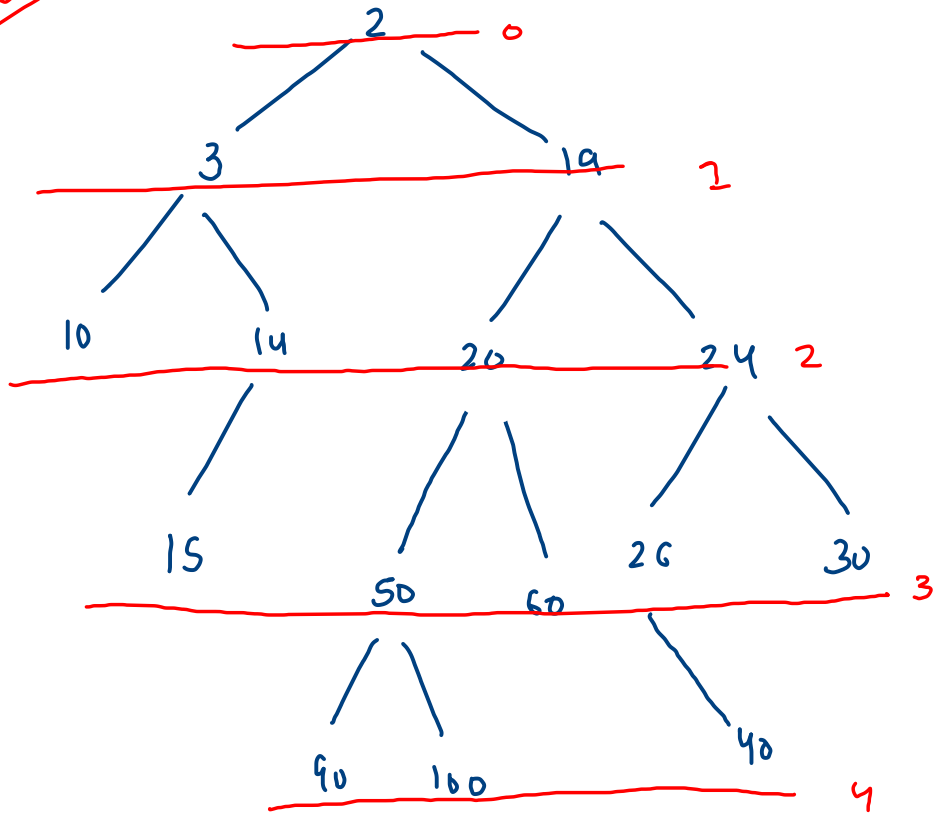


data = 50

[50, 20, 19, 2]

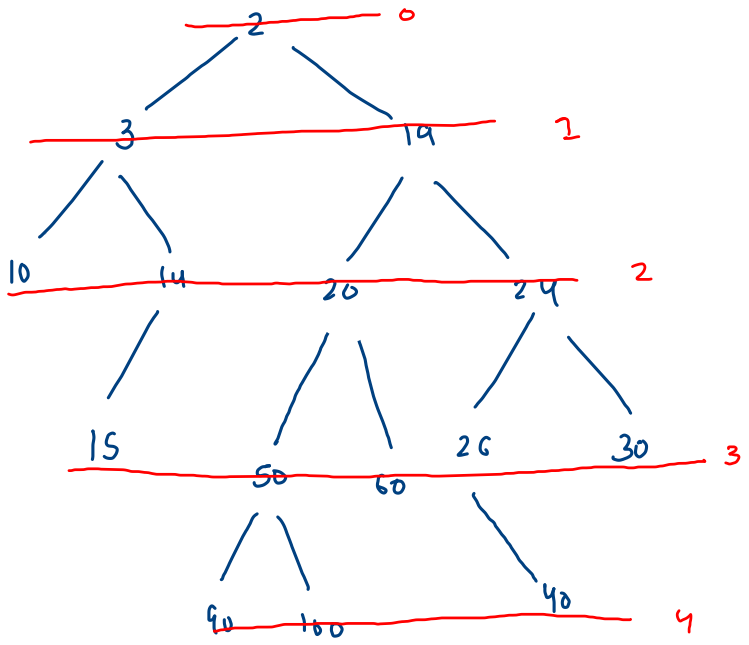
```java
public static ArrayList<Integer> nodeToRootPath(Node node, int data){
    if(node == null) {
        return new ArrayList<>();
    }

    if(node.data == data) {
        ArrayList<Integer>list = new ArrayList<>();
        list.add(node.data);
        return list;
    }

    ArrayList<Integer>n2lcp = nodeToRootPath(node.left,data); //node to left child path

    if(n2lcp.size() > 0) {
        n2lcp.add(node.data); //converting n2lcp to node to root path
        return n2lcp;
    }

    ArrayList<Integer>n2rcp = nodeToRootPath(node.right,data); //node to right child path

    if(n2rcp.size() > 0) {
        n2rcp.add(node.data); //converting n2rcp to node to root path
        return n2rcp;
    }

    return new ArrayList<>();
}
```

print k levels down



k = 3

15 50 60 26 30



level order leo

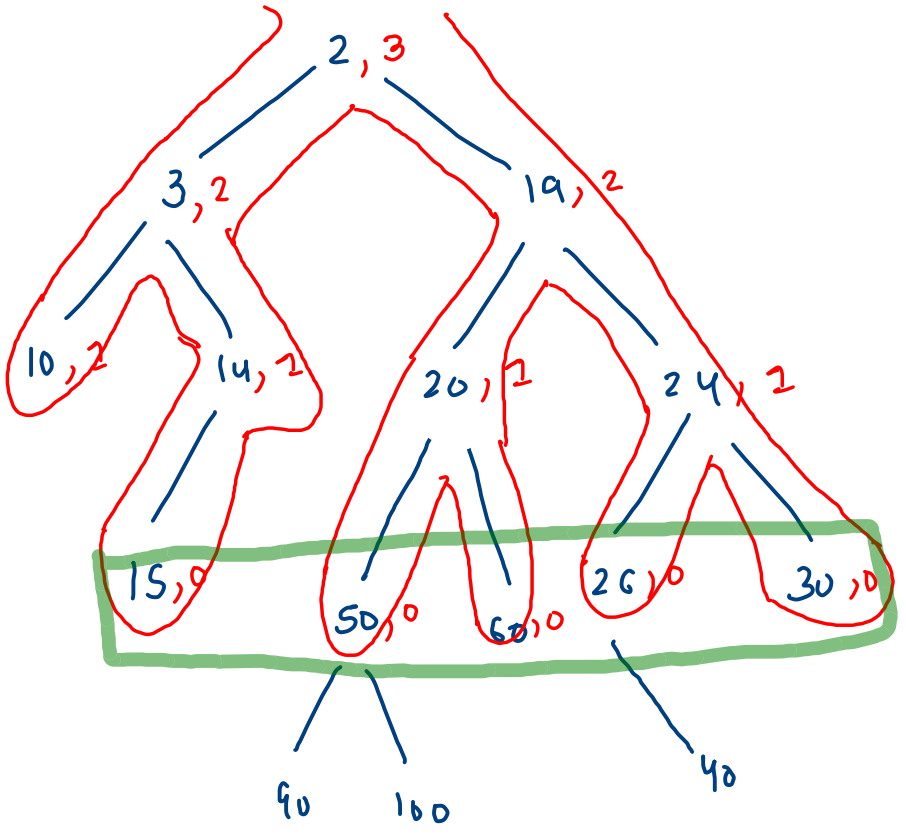
k = 3

leo : ∅ ~~1~~ ~~2~~ 3

count = 4

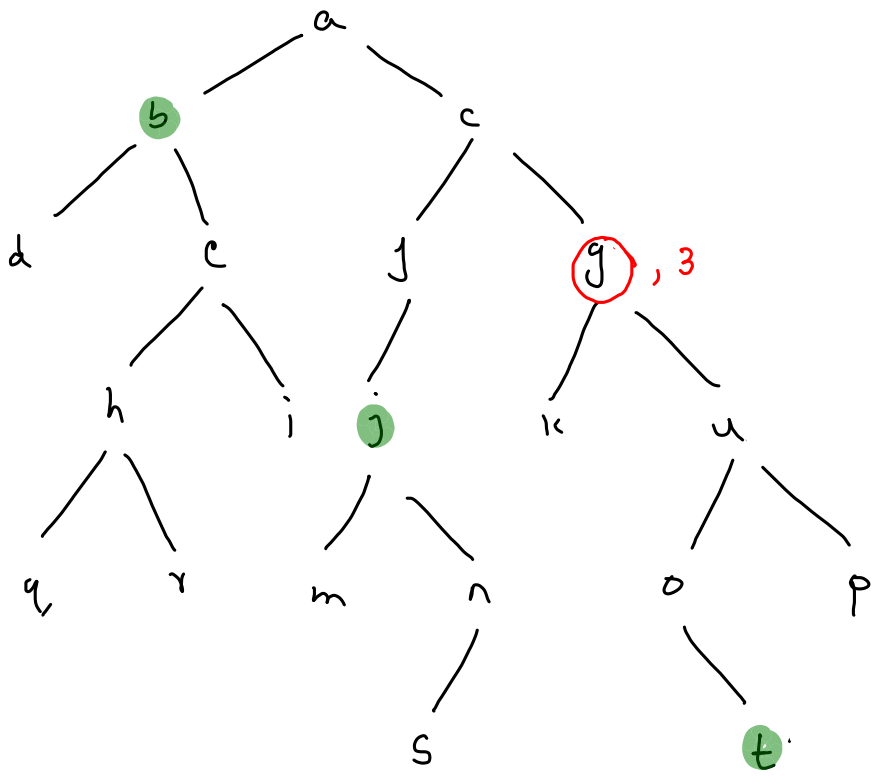
~~2~~	~~3~~	~~19~~	~~10~~	~~14~~	~~20~~	~~24~~	15	50	60	26	30

count { remove
work
add children



K = 3

15, 50, 60, 26, 30



data = g

k = 3

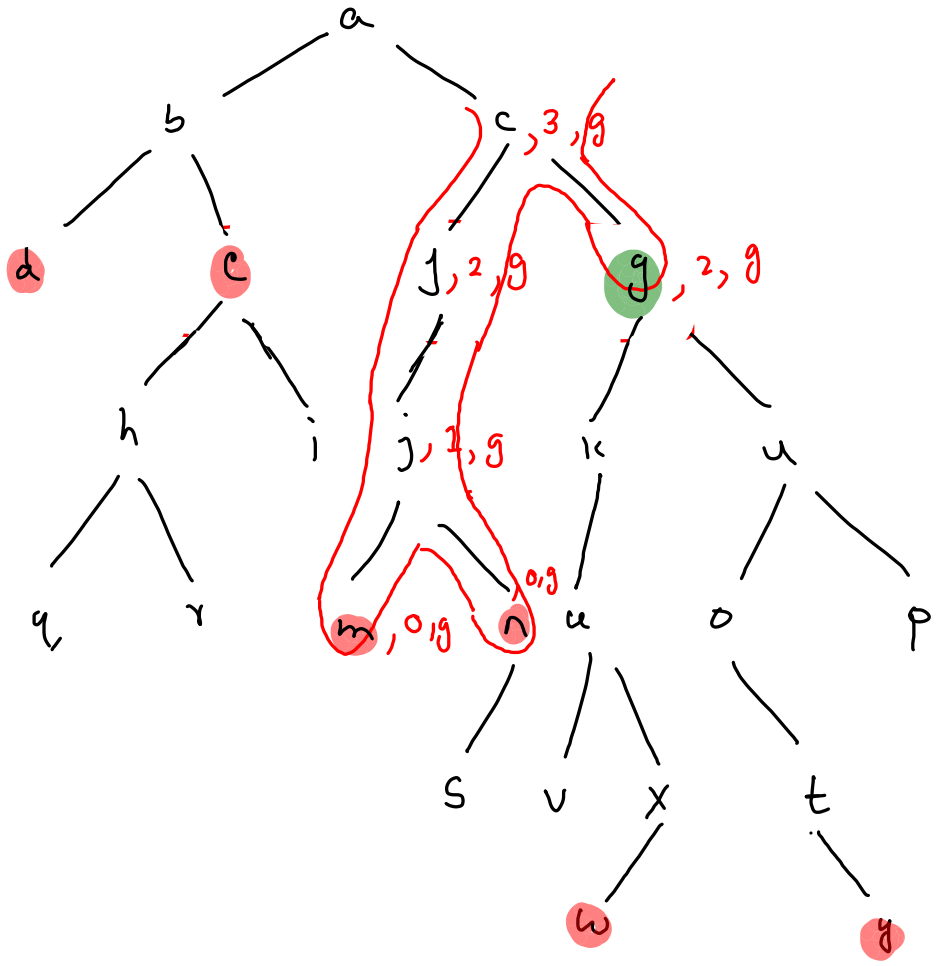
prb: null g c

n2sp: [g , c , a]

k down (k-1) down (k-2) down

(k away) (k away) (k away)

t j o



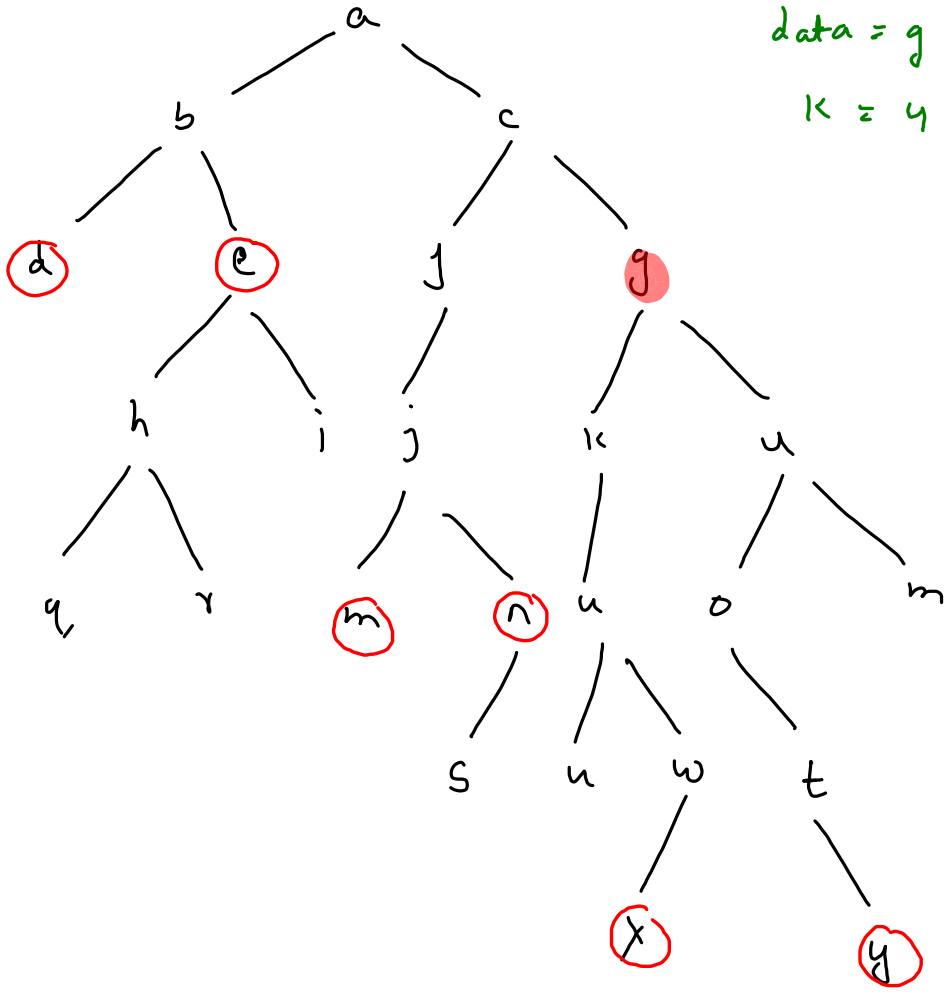
data = g

k = 4

[g c a]

g	c	a
g k down	c k-1 down	a k-2 down
g k away	g k away	g k away
prbt null	prbt g	prbt c
w, y	m, n	d, e

prbt null g c



data = g

k = 4

```
public static void printKNodesFar(Node root, int data, int k) {
  // write your code here
  ArrayList<Node>n2rp = nodeToRootPath(root,data);
  Node prhbt = null;

  for(int i=0; i < n2rp.size();i++) {
      Node node =  n2rp.get(i);
      printKLevelsDown(node,k-i,prhbt);
      prhbt = node;
  }
}
```

n2rp = [g (0), c (1), a (2)]

prhbt = ~~null~~ g → c → a

```
public static void printKLevelsDown(Node node,int k,Node prhbt) {
    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);
}
```

x, y, m, n, d, e