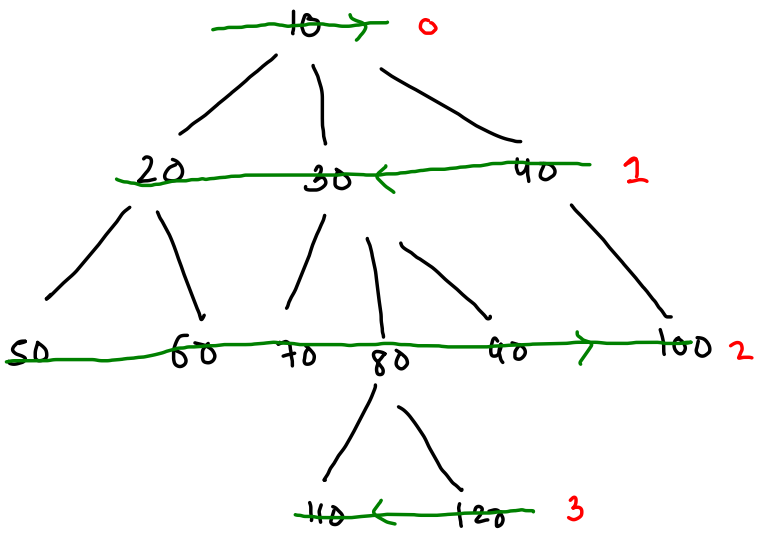
Levelorder Linewise Zig Zag

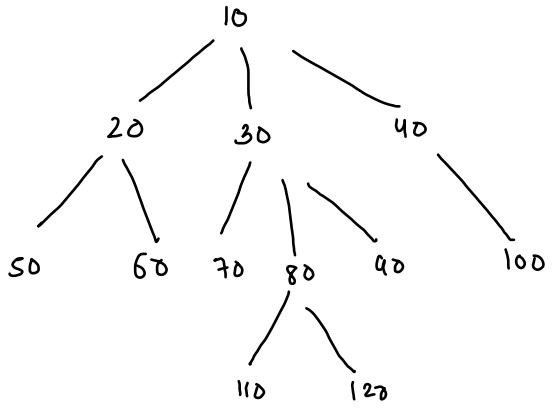


LOLW

0	10
1	40 30 20
2	50 60 70 80 90 100
3	120 110

even levels : L to R

odd levels : R to L



Lw ->

even : L to R

odd : R to L

level order line wise

(count way)

10

40 30 20

50 60 70 80 90 100

120 110

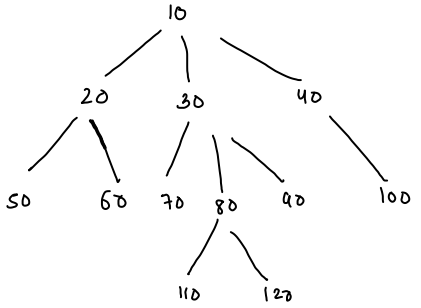
c = 2 d = ~~0~~ ~~1~~ ~~2~~ 3

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

0 1 2 3

- Count times
  - remove
  - work
  - add children

```java
while(q.size() > 0) {
    int count = q.size();

    ArrayList<Integer>list = new ArrayList<>();
    for(int k = 0; k < count;k++) {
        //remove
        Node rem = q.remove();

        //work
        list.add(rem.data);

        //add children
        for(int i=0; i < rem.children.size();i++) {
            Node child = rem.children.get(i);
            q.add(child);
        }
    }

    if(lev % 2 == 0) {
        //level is even -> L to R
        for(int i=0; i < list.size();i++) {
            System.out.print(list.get(i) + " ");
        }
        System.out.println();
    }
    else {
        //level is odd -> R to L
        for(int i=list.size()-1; i >= 0;i--) {
            System.out.print(list.get(i) + " ");
        }
        System.out.println();
    }

    lev++;
}
```



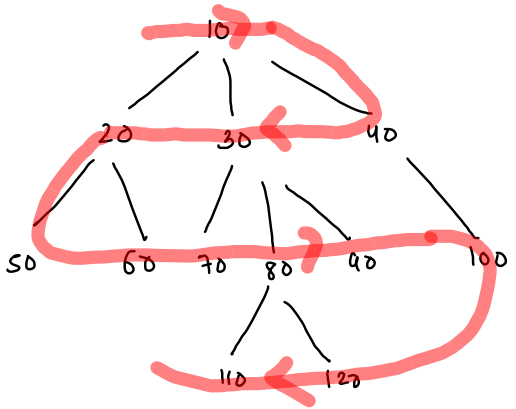
10

40 30 20

50 60 70 80 90 100

120 110

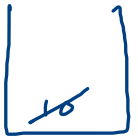
10	20	30	40	50	60	70	80	90	100	110	120

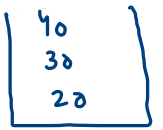

10
20
30
40
50
60
70
80
90
100
110
120

ms

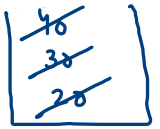
cs

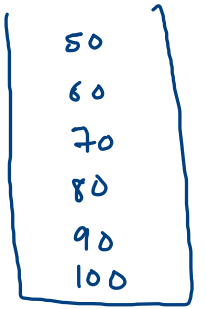
0


10

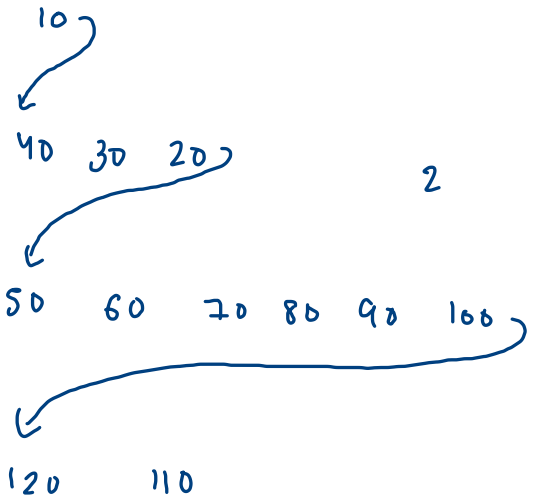

40
30
20

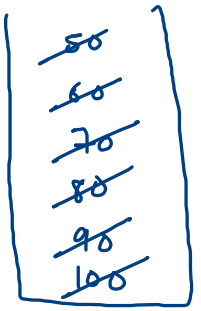
1

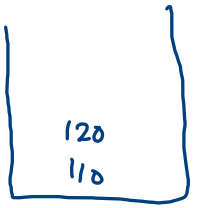

40
30
20

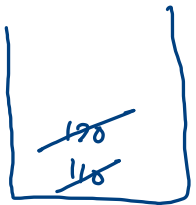

50
60
70
80
90
100

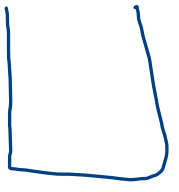
2

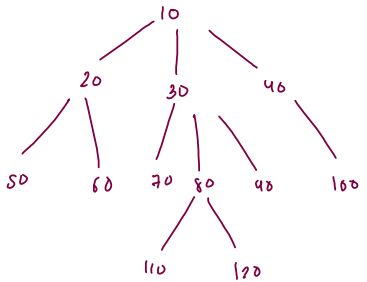

10
40 30 20
50 60 70 80 90 100
120 110


50
60
70
80
90
100


120
110


120
110





even lev : L to R

odd lev : R to L

```
while(ms.size() > 0) {
    //remove
    Node rem = ms.pop();

    //work
    System.out.print(rem.data+" ");


    //add children
    if(lev % 2 == 0) {
        for(int i=0; i < rem.children.size();i++) {
            Node child = rem.children.get(i);
            cs.push(child);
        }
    }
    else {
        for(int i=rem.children.size()-1; i >= 0;i--) {
            Node child = rem.children.get(i);
            cs.push(child);
        }
    }


    if(ms.size() == 0) {
        System.out.println();
        ms = cs;
        cs = new Stack<>();
        lev++;
    }
}
```

ms

lev : 0

lev : 1

lev : 2

lev : 3

lev : 4

cs

10

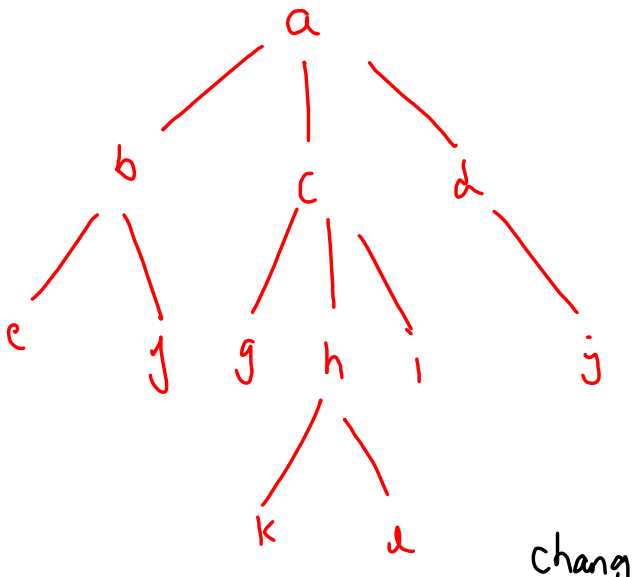
40 30 20

50 60 70 80 90 100

120 110

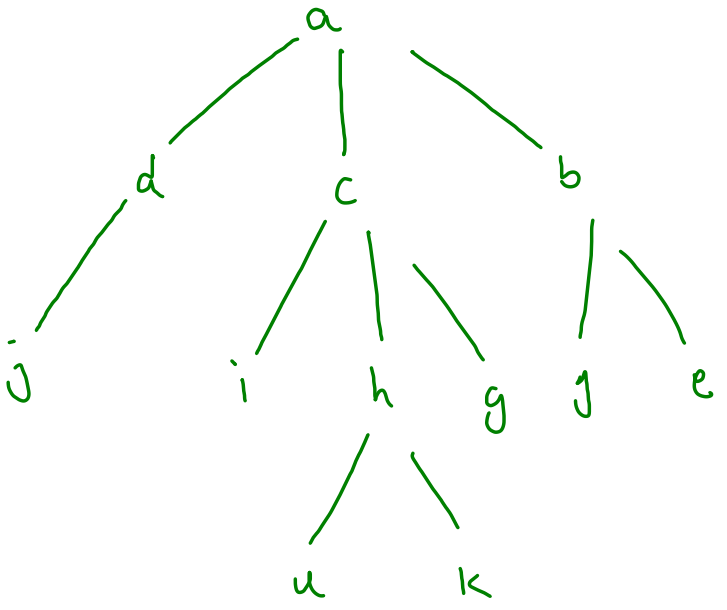
Mirror A Generic Tree

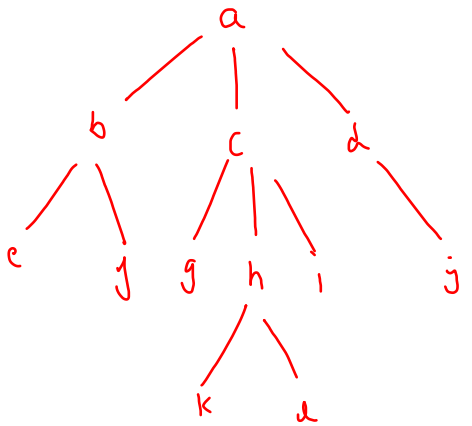
tree



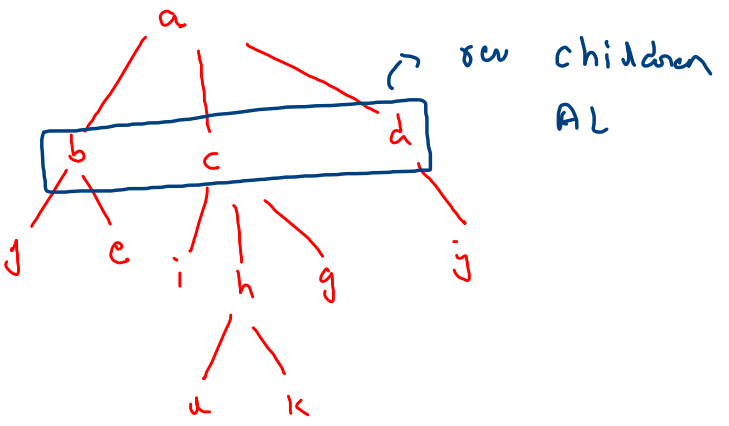
change this tree

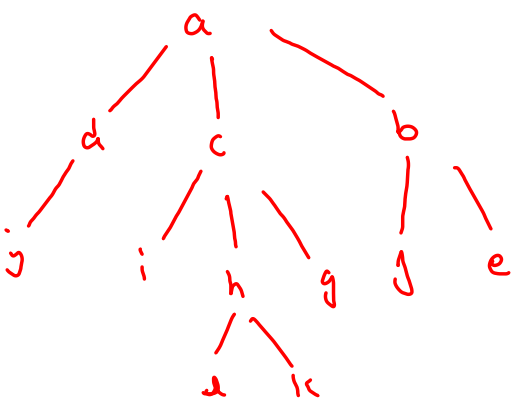
mirror image

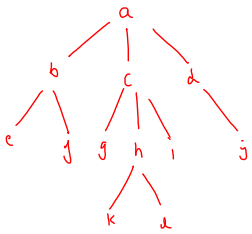


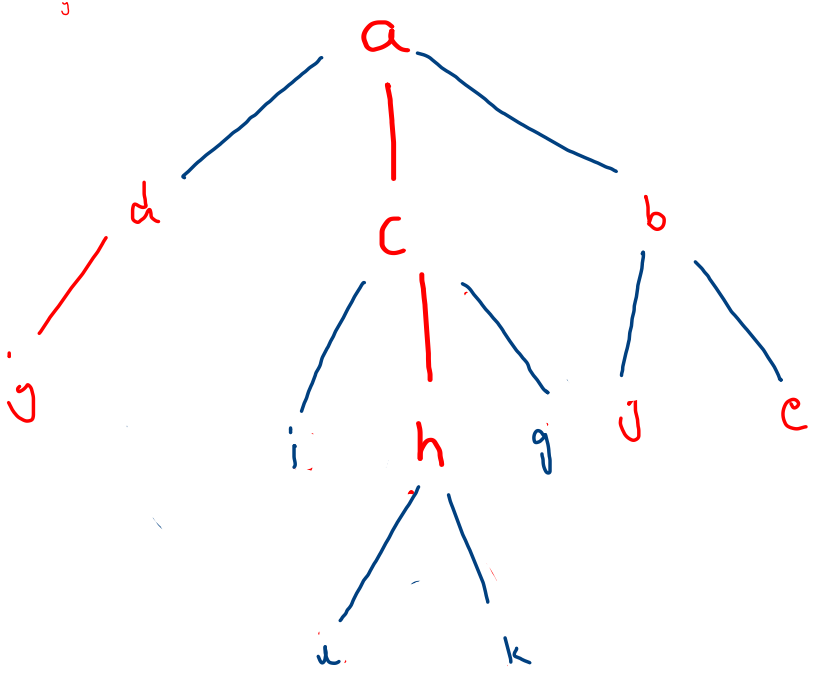


mirror (b),
mirror (c),
mirror (d)







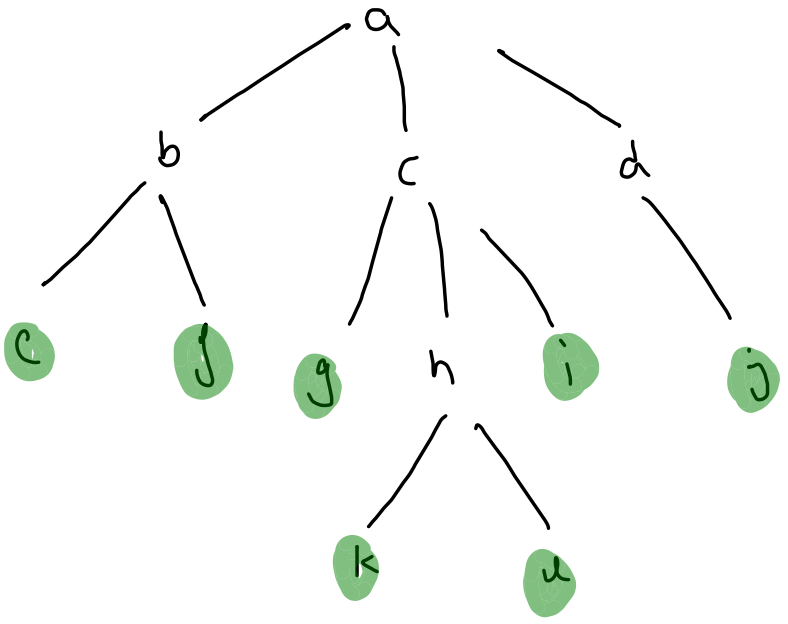


node

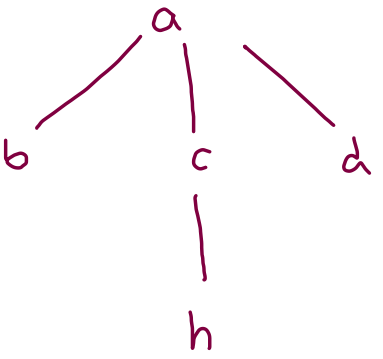
post

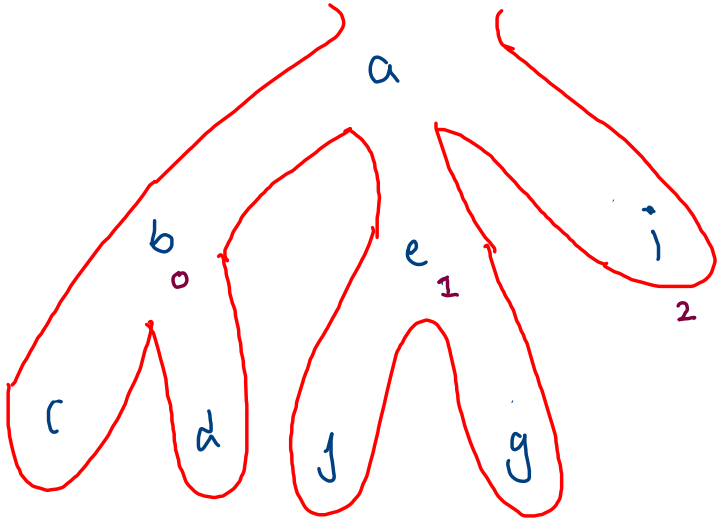
```java
public static void mirror(Node node){

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

    //reverse node's children arraylist
    int l = 0;
    int r = node.children.size()-1;

    while(l < r) {
        Node ln = node.children.get(l);
        Node rn = node.children.get(r);

        node.children.set(l,rn);
        node.children.set(r,ln);

        l++;
        r--;
    }
}
```

Remove Leaves In Generic Tree



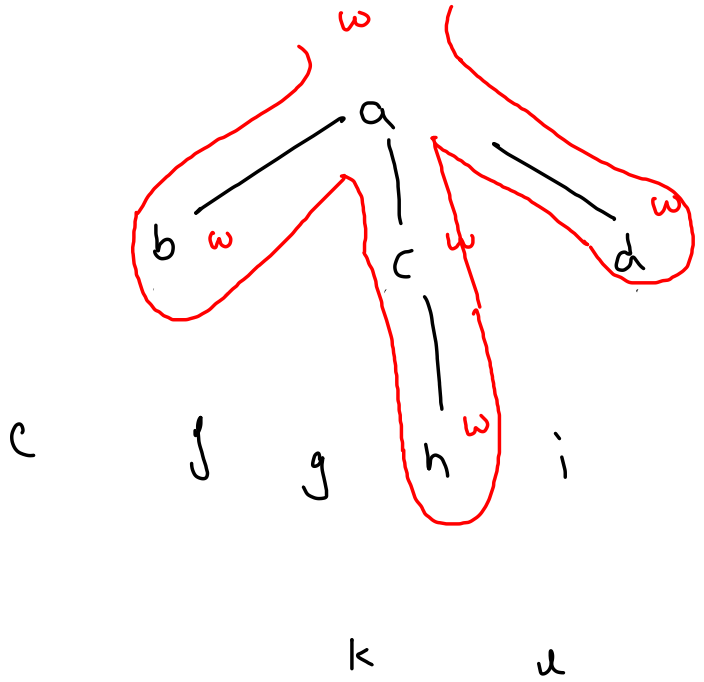
remove leaves →

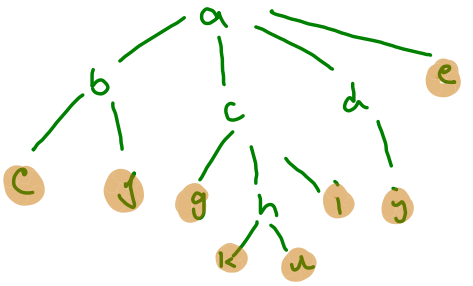





```java
public static void removeLeaves(Node node) {

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

    for(int i=node.chilren.size()-1; i >= 0;i--) {
        Node child = node.children.get(i);

        if(child.children.size() == 0) {
            node.children.remove(i);
        }
    }
}
```

wrong ; work (post-order)





ω

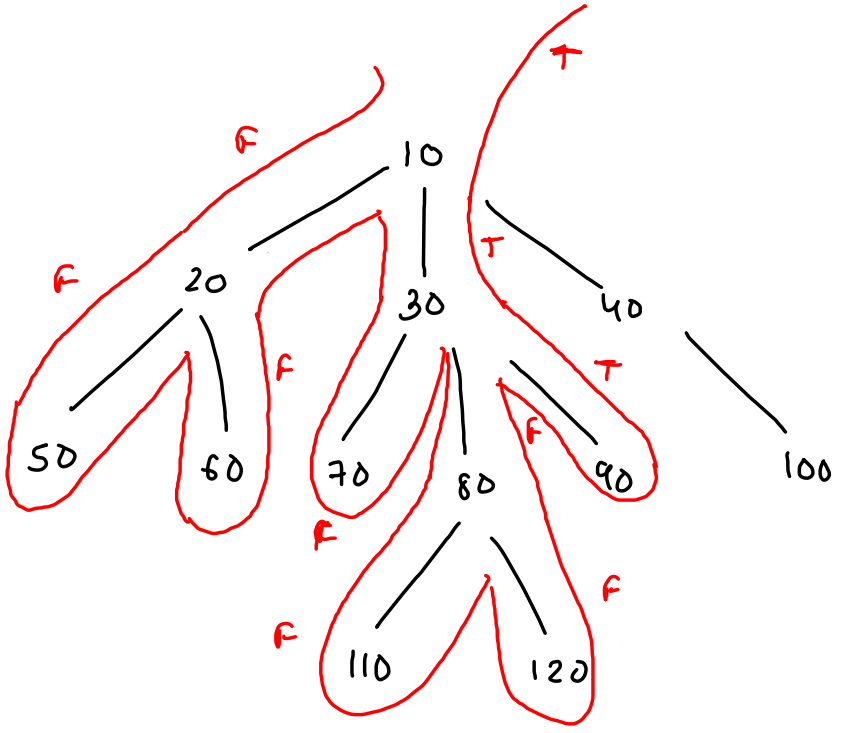
```
for(int i=node.children.size()-1; i >= 0;i--) {
    Node child = node.children.get(i);

    if(child.children.size() == 0) {
        node.children.remove(i);
    }
}
```

calls

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

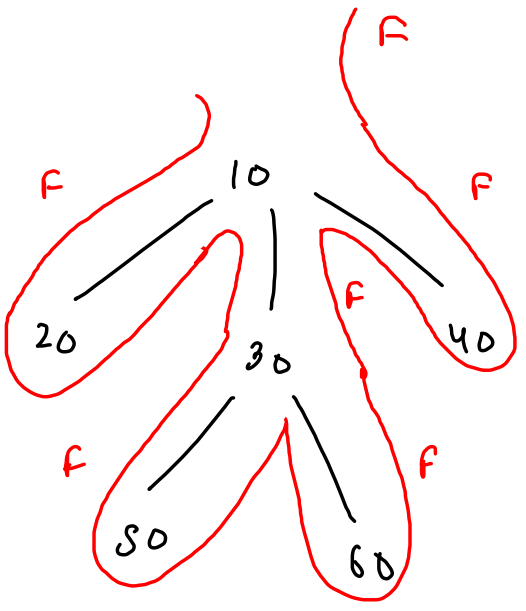
find

data = 90




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

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

        boolean fic = find(child,data);

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

    return false;
}
```

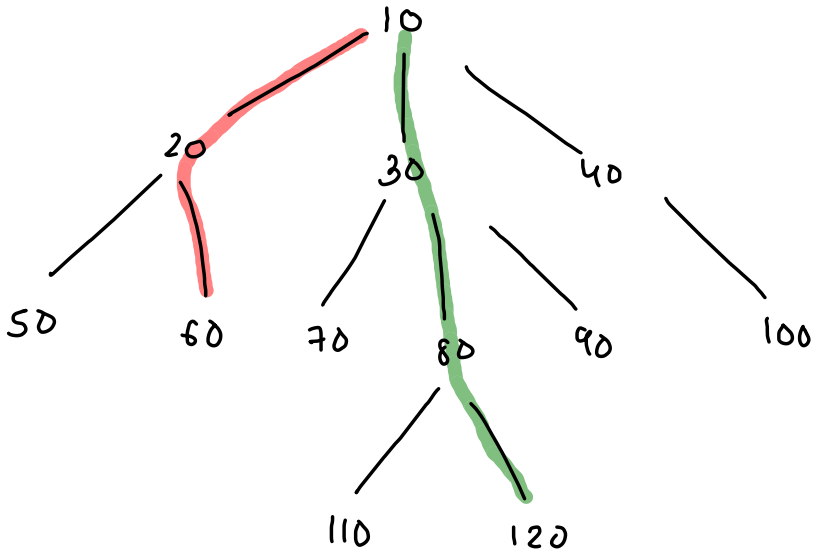
S



data = 100

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

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

        boolean fic = find(child,data);

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

    return false;
}
```

Node To Root Path In Generic Tree



data = 120

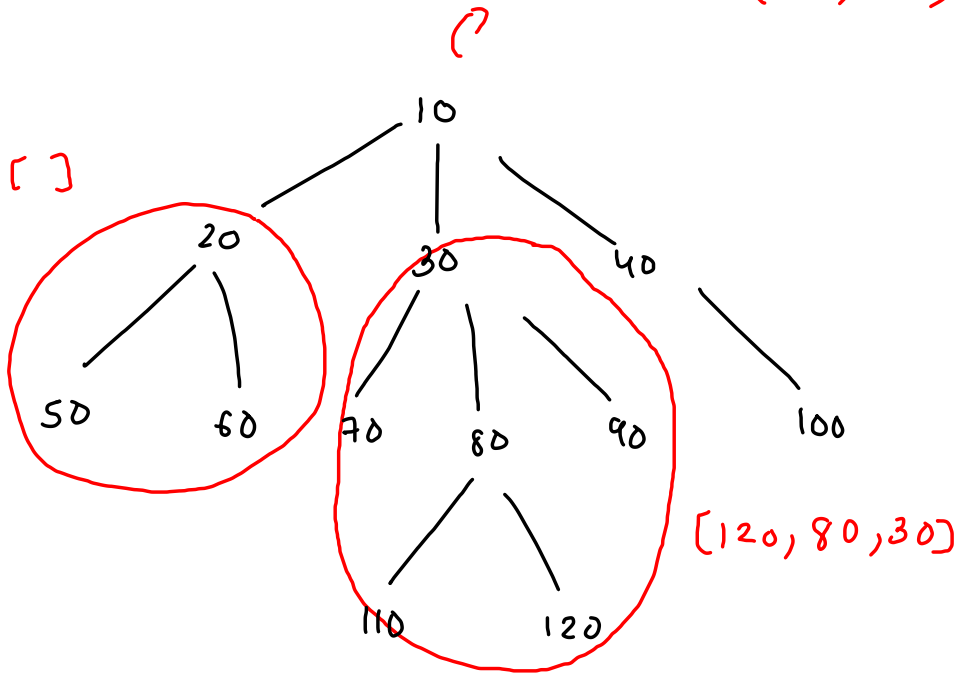
n2rP -> [120, 80, 30, 10]

(120)

n2rP -> [60, 20, 10]

(60)

n2r ans : [120, 80, 30, 10]



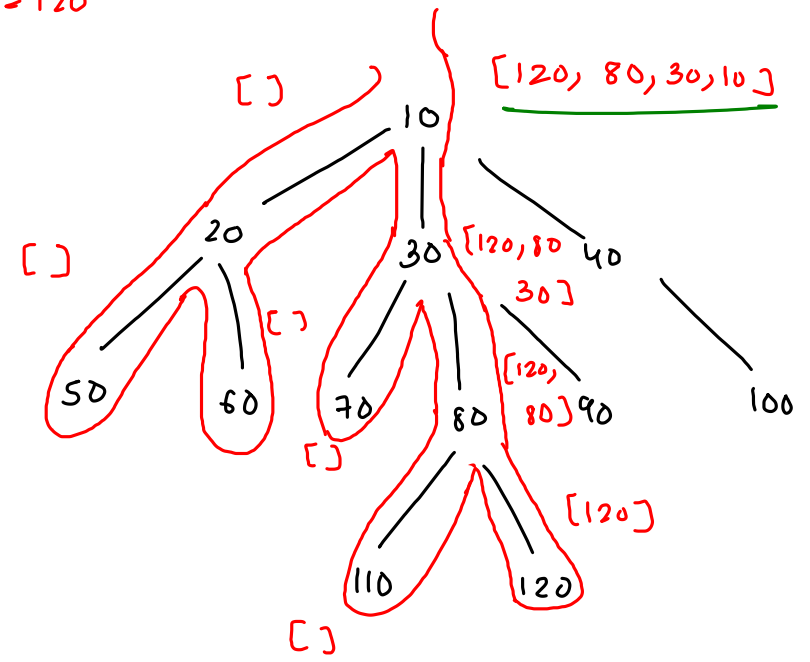
[ ]

[120, 80, 30]

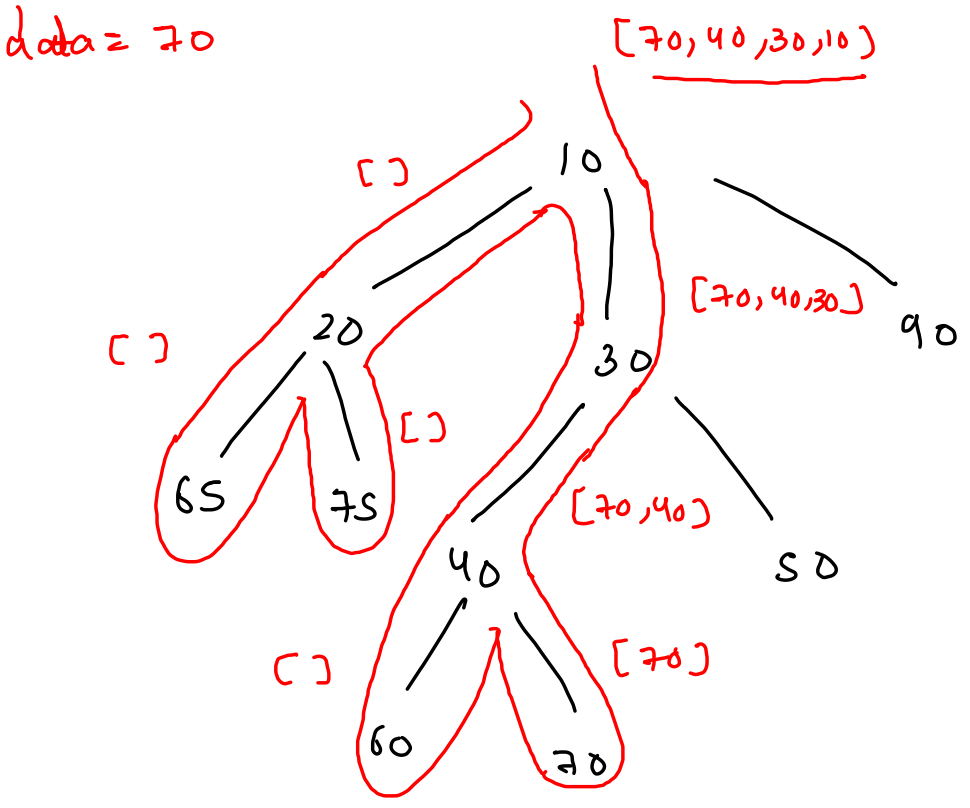
data → 120

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

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

        ArrayList<Integer>n2cp = nodeToRootPath(child,data);

        if(n2cp.size() > 0) {
            n2cp.add(node.data); //converting n2cp(node to child pa
            return n2cp;
        }
    }

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

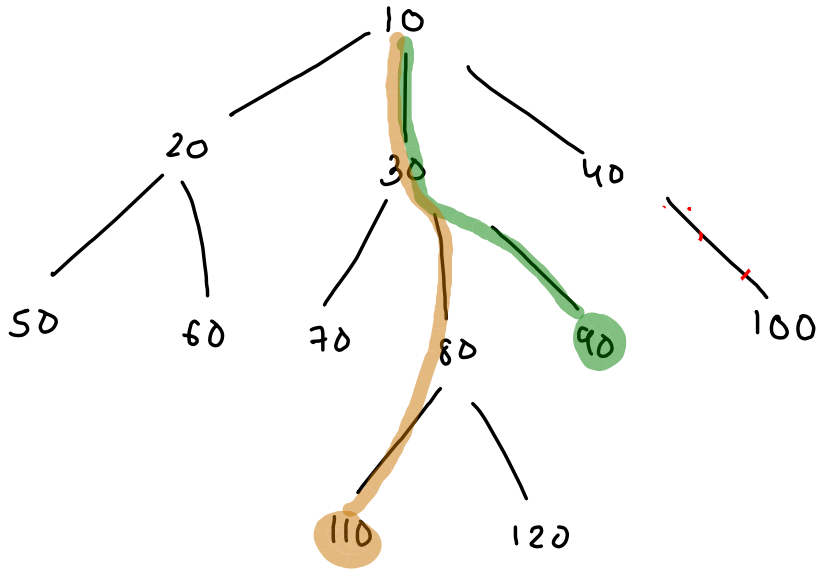
data = 120



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

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

        ArrayList<Integer>n2cp = nodeToRootPath(child,data);

        if(n2cp.size() > 0) {
            n2cp.add(node.data); //converting n2cp(node to child pc
            return n2cp;
        }
    }

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



Lowest Common Ancestor (generic Tree)

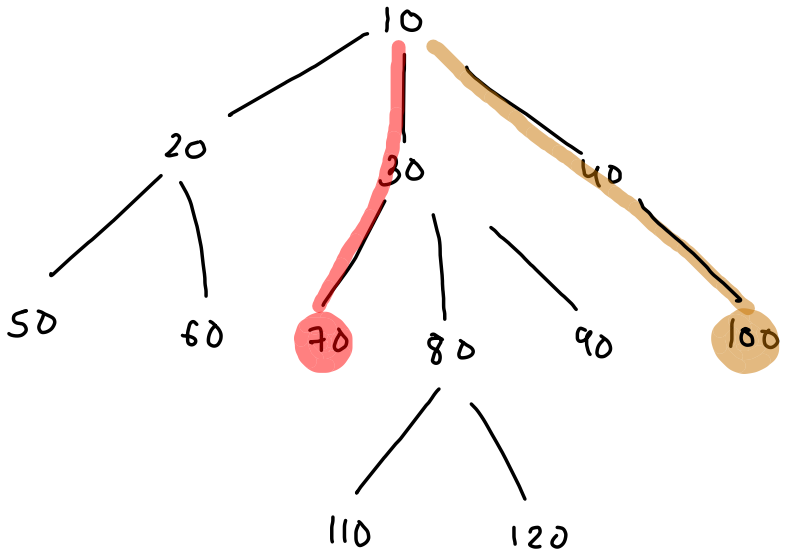


d1 = 110

d2 = 90

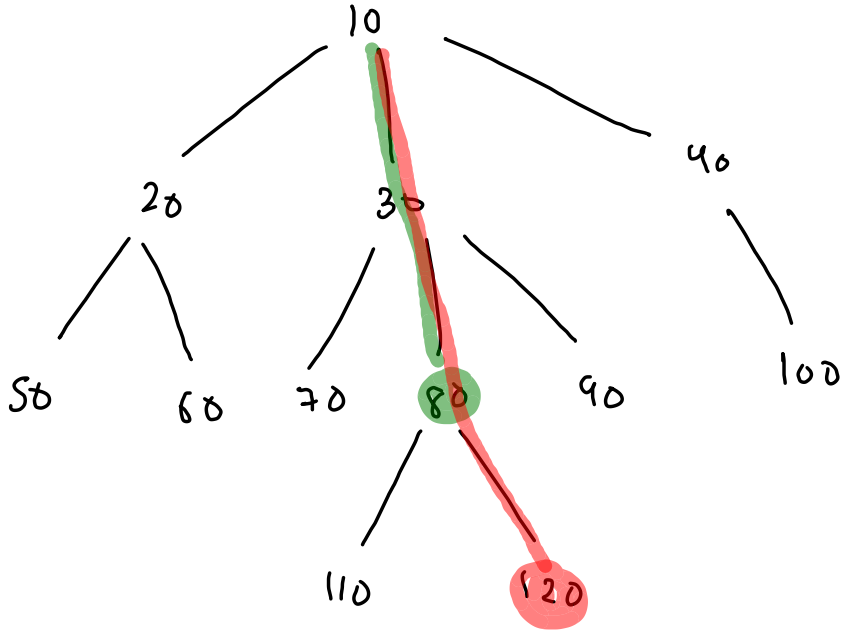
n2rp : [110, 80, 30, 10]

n2sp : [90, 30, 10]



path1 : 70, 30, 10
i i

path2 : 100, 40, 10
j j



d1 = 80

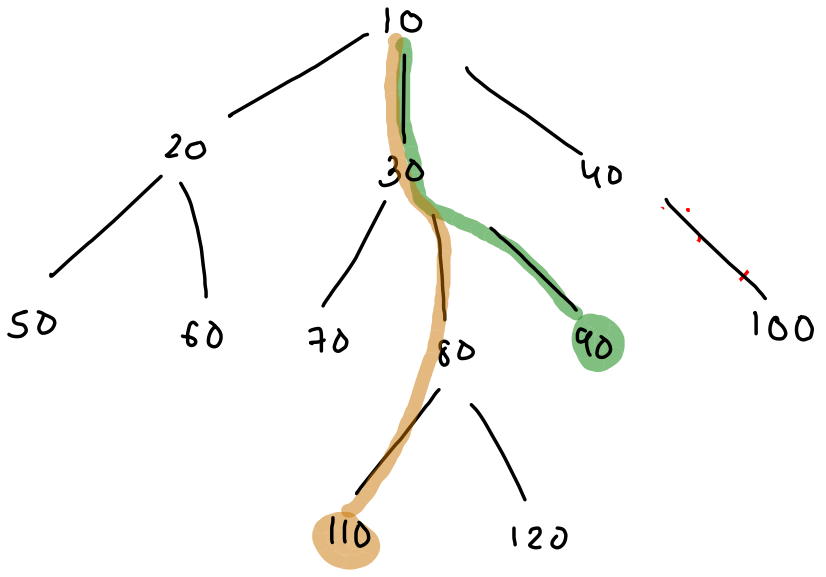
d2 = 120

p1 : [80, 30, 10]

i ~~i~~ ~~i~~ ~~i~~

p2 : [120, 80, 30, 10]

j ~~j~~ ~~j~~ ~~j~~




```java
public static int lca(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--;
    }

    return p1.get(i+1);
}
```

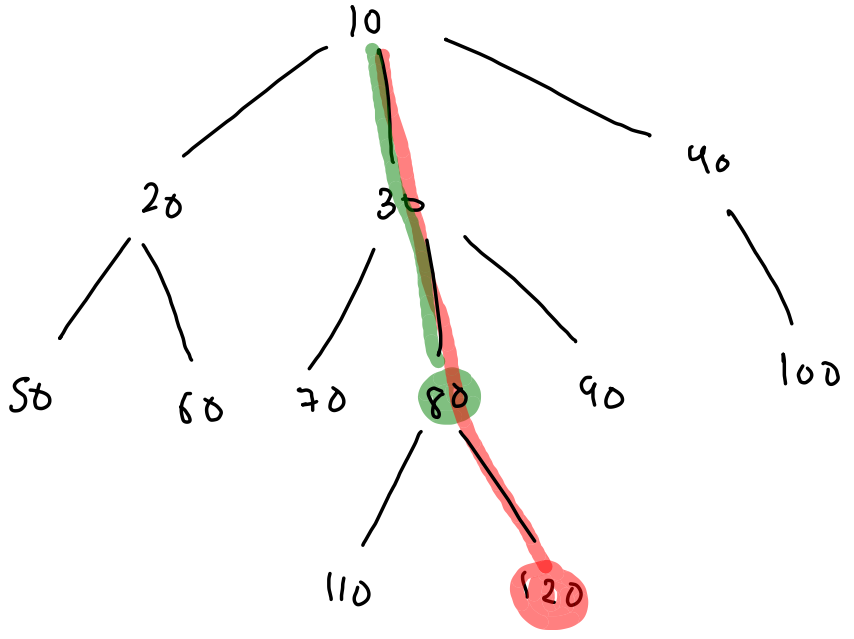
d1 = 90

d2 = 110

p1 = [90, 30, 10]

p2 = [110, 80, 30, 10]

dca -> 30




```java
public static int lca(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--;
    }

    return p1.get(i+1);
}
```

d1 = 120

d2 = 80

p1 : [120, 80, 30, 10]

p2 : [80, 30, 10]

lca = 80