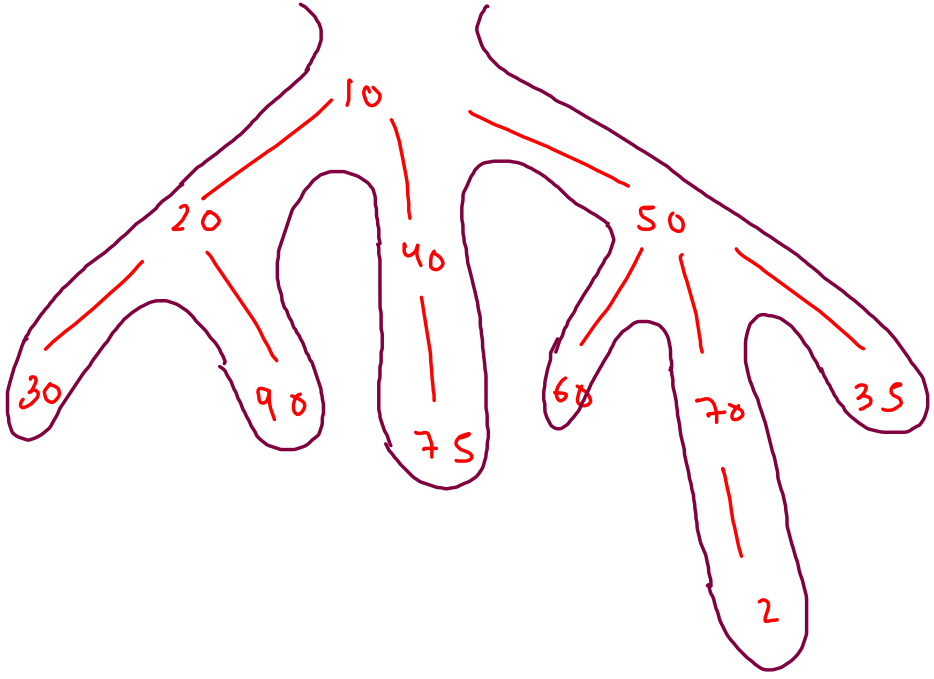
Multisolver For Generic Tree

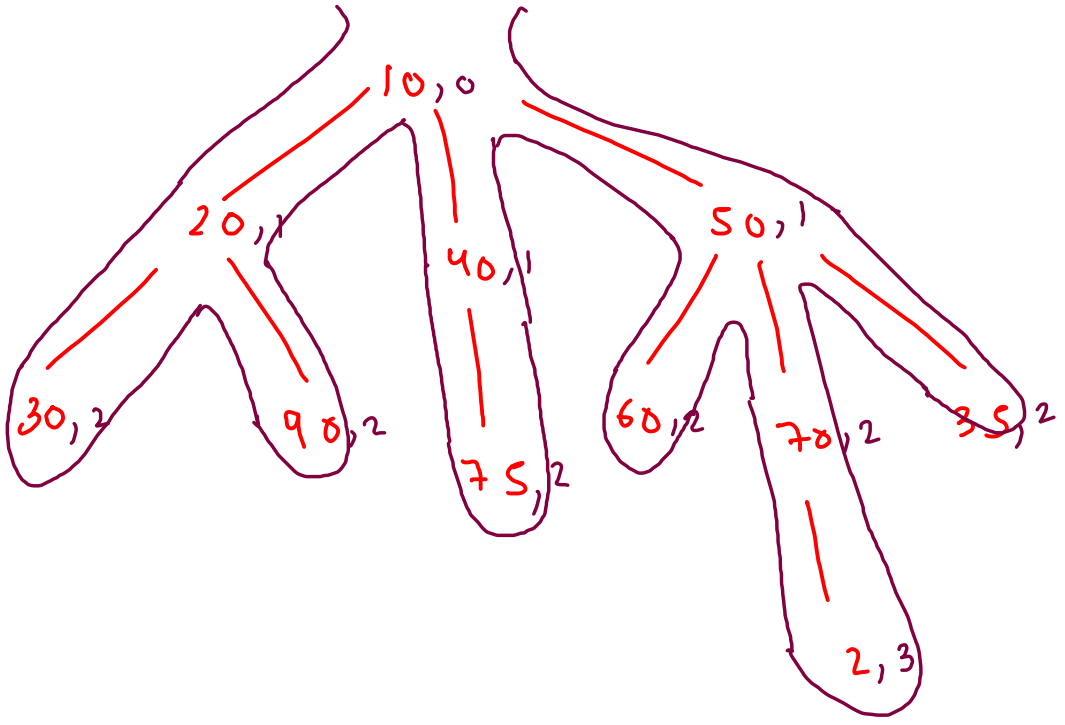


Size = ~~0 1 2 3 4 5 6 7 8 9 10~~ 11

max = ~~-∞ 10 20 30~~ 90

min = ~~∞ 10~~ 2

~~ht~~ =




```java
public static void multisolver(Node node,int lev) {
    //self work
    size++;
    max = Math.max(max,node.data);
    min = Math.min(min,node.data);
    ht = Math.max(ht,lev);

    for(int i=0; i < node.children.size();i++) {
        Node child = node.children.get(i);
        multisolver(child,lev+1);
    }

}
```

11

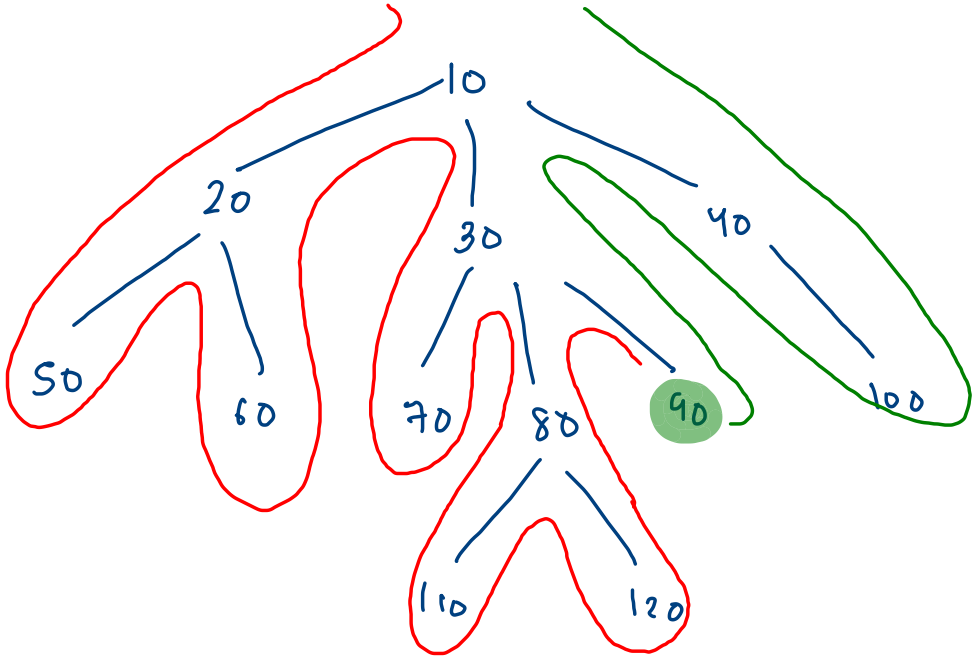
Size = ~~0 1 2 3 4 5 6 7 8 9 10~~

max = ~~-∞ 10 20 30~~ 90

min = ~~∞ 10~~ 2

ht = ~~0 1~~ 2 3

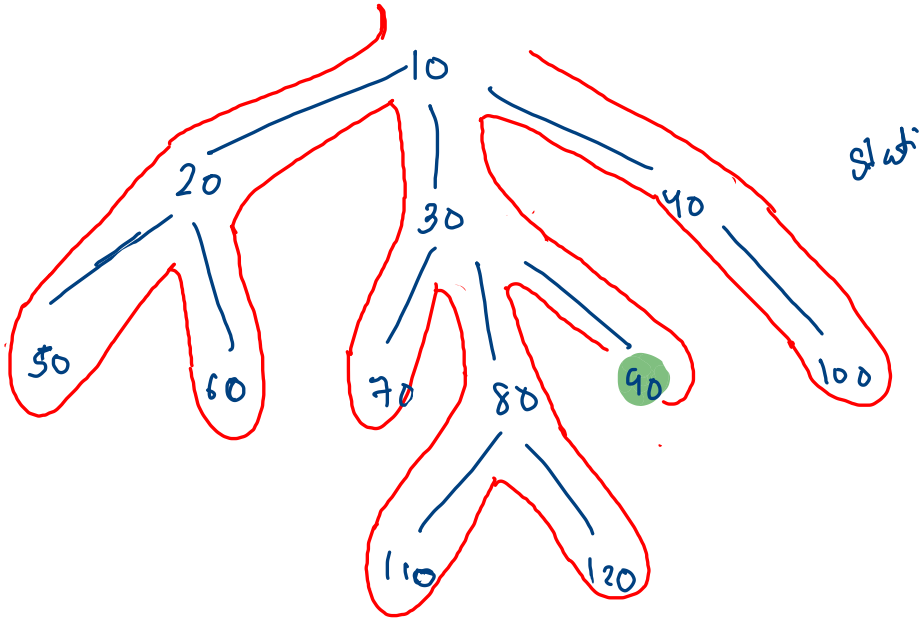
Predecessor And Successor Of An Element



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

— : before 90

— : after 90



static

prev = ~~n~~ 120 succ = ~~n~~ 40

prev = ~~n~~ ~~10~~ ~~20~~ ~~50~~ ~~60~~ ~~30~~ ~~70~~ ~~80~~ ~~40~~ ~~120~~ ~~90~~ 40

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

work

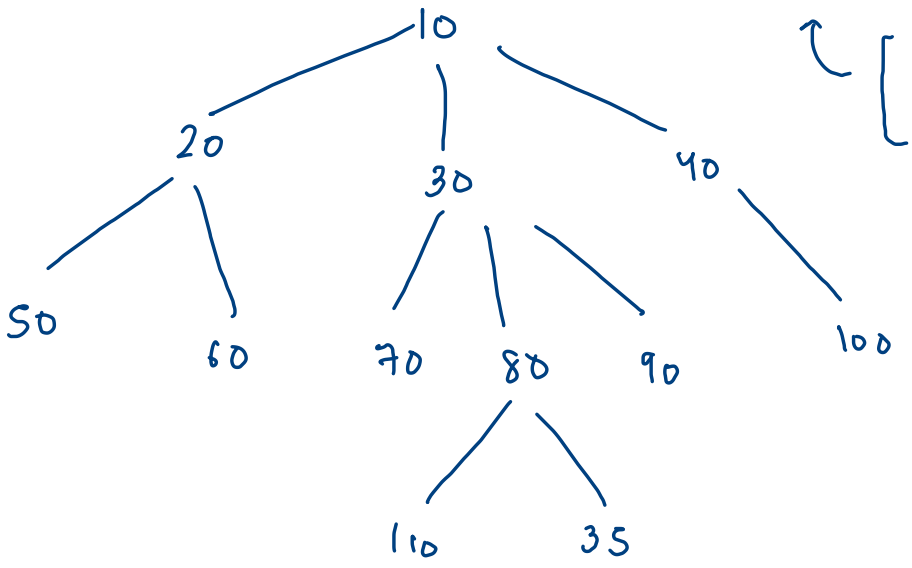
calls

```java
public static void predecessorAndSuccessor(Node node, int data) {
    prev = curr;
    curr = node;

    if(curr.data == data) {
        predecessor = prev;
    }
    else if(prev != null && prev.data == data) {
        successor = curr;
    }

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

Ceil And Floor In Generic Tree



initialisation

ceil = ∞

floor = -∞

data = 38

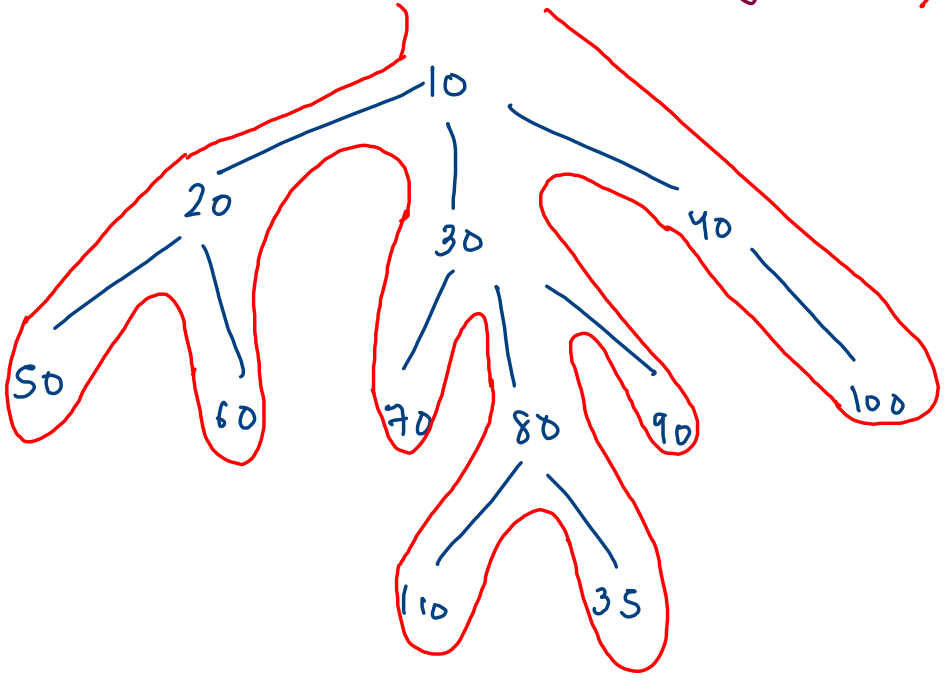
ceil -> just larger / smallest among larger / qualified min

floor -> just smaller / largest among smaller / qualified max

ceil = ~~∞~~ ~~50~~ 40

data = 38

floor = ~~-∞~~ ~~10~~ ~~20~~ ~~30~~ 35

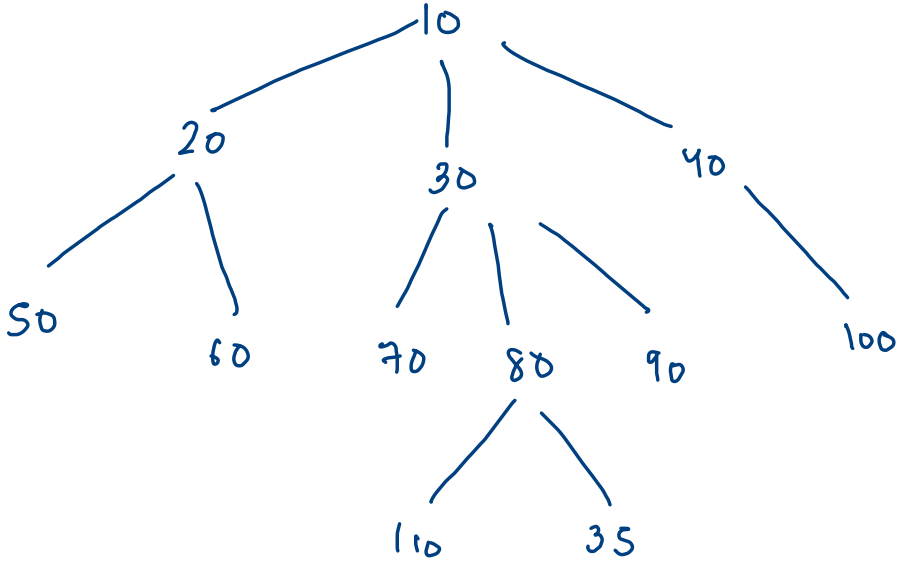



```
//ceil -> qualified min
if(node.data > data && ceil > node.data) {
    ceil = node.data;
}

//floor -> qualified max
if(node.data < data && floor < node.data) {
    floor = node.data;
}

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

Kth Largest Element In Tree

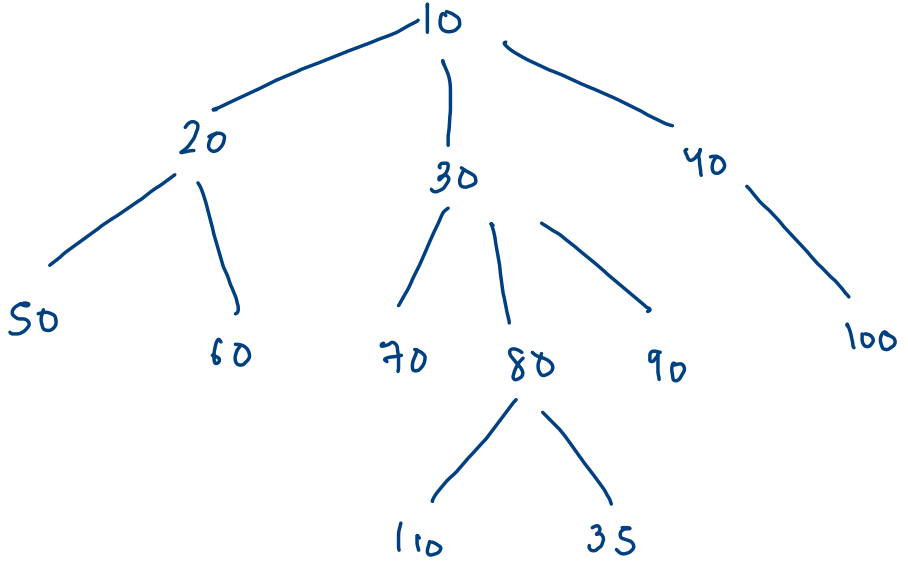


k = 3

∞ —floor→ (1st max)

(1st max) —floor→ (2nd max)

(2nd max) —floor→ (3rd max)




```java
public static int kthLargest(Node node, int k){
    int key = Integer.MAX_VALUE;

    for(int i=0; i < k;i++) {
        floor = Integer.MIN_VALUE;
        ceilAndFloor(node,key); -> n
        key = floor;
    }

    return floor;
}
```

k

kn

k = 3

90

0

~~110~~ ~~100~~

1

key = ~~∞~~

2

floor : ~~-∞~~ 90

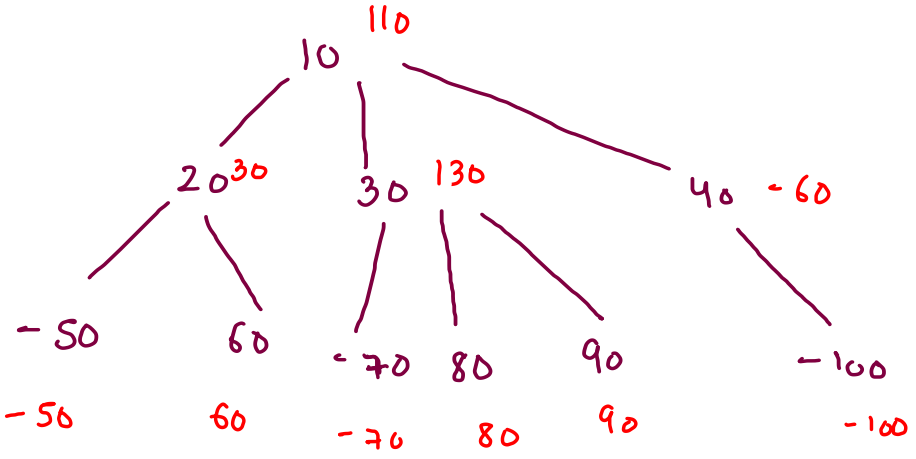
Node With Maximum Subtree Sum

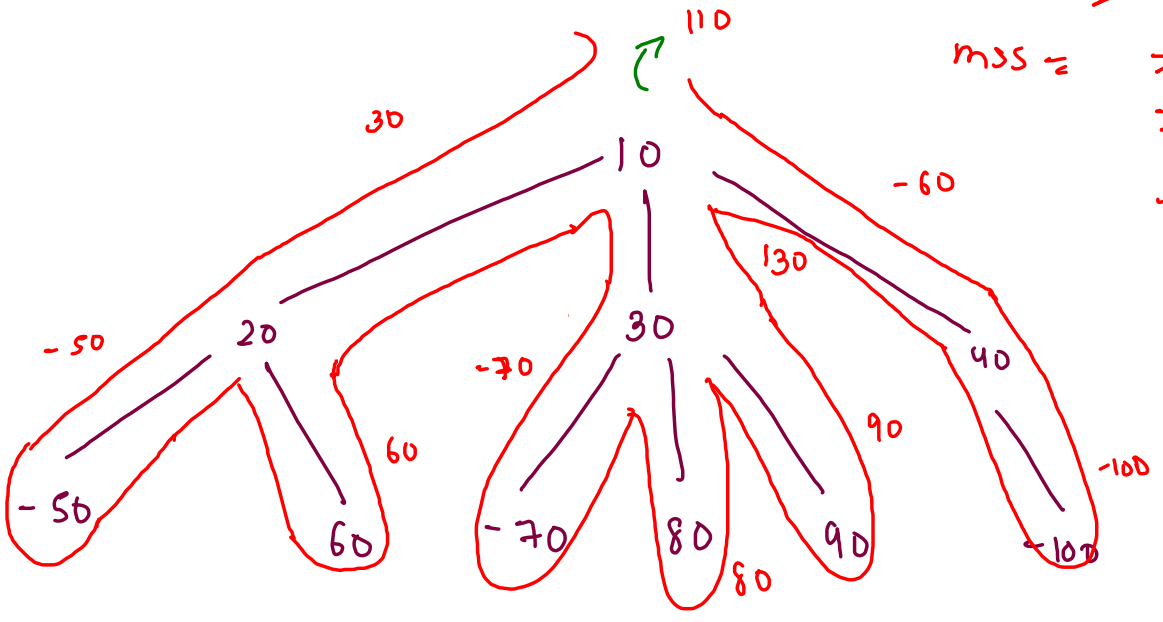
```
20
10 20 -50 -1 60 -1 -1 30 -70 -1 80 -1 90 -1 -1 40 -100 -1 -1 -1
```

30 @ 130

mss = -∞

mssn = null





mssn = ~~null~~ ~~-50~~ ~~60~~ ~~80~~ ~~90~~ 30

mss = ~~-∞~~ ~~-50~~ ~~60~~ ~~80~~ ~~90~~ 130

30 @ 130

```java
public static void maxSubtreeSum(Node root) {
    mssn = null; //max subtree sum node
    mss = Integer.MIN_VALUE; //max subtree sum
    sum(root);

    System.out.println(mssn.data + "@" + mss);
}

public static int sum(Node node) {
    int sum = node.data;

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

        sum += cfs;
    }

    if(sum > mss) {
        mss = sum;
        mssn = node;
    }

    return sum;
}
```