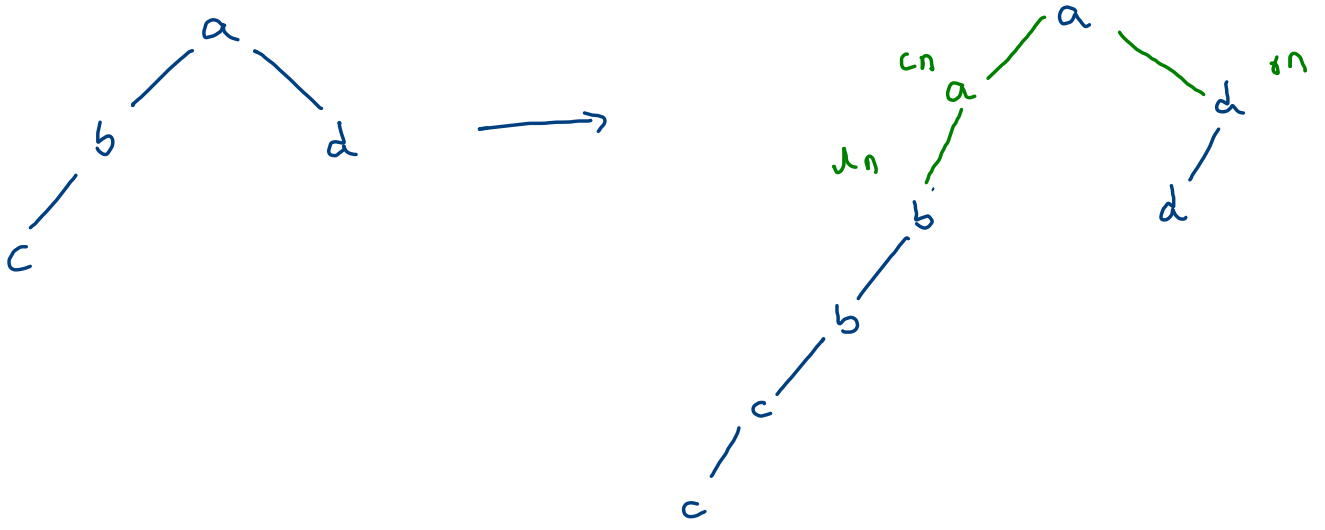
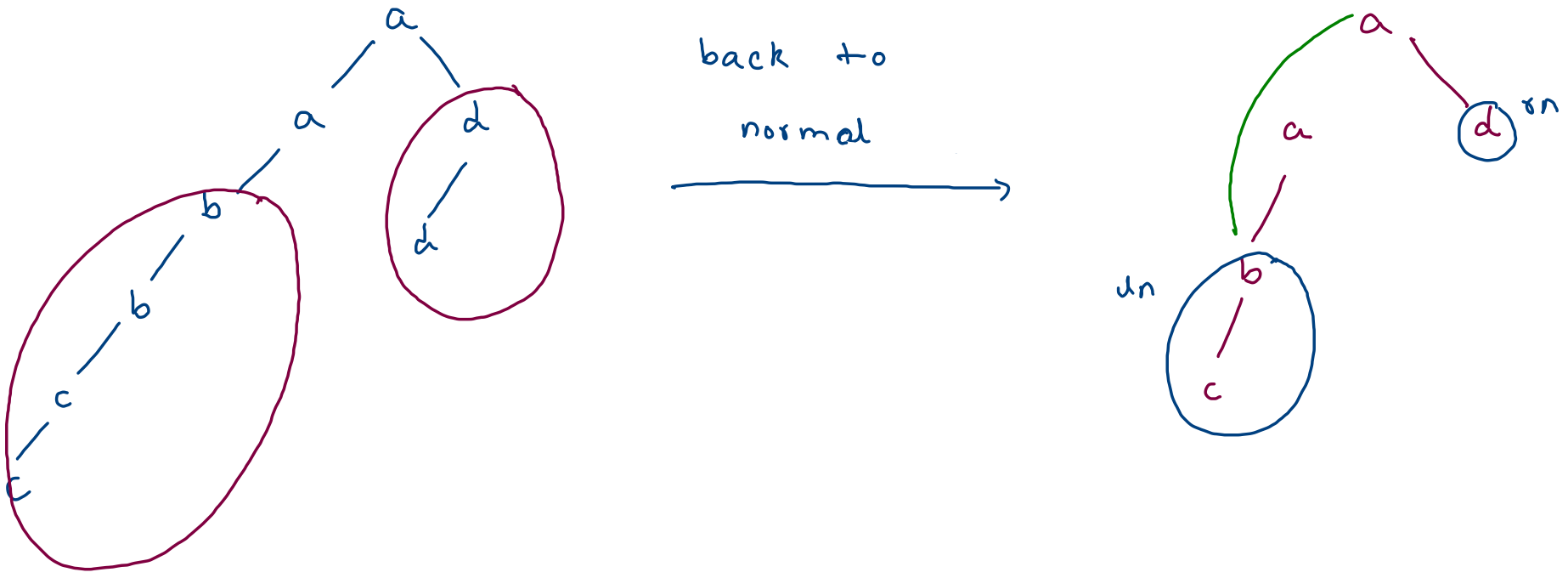
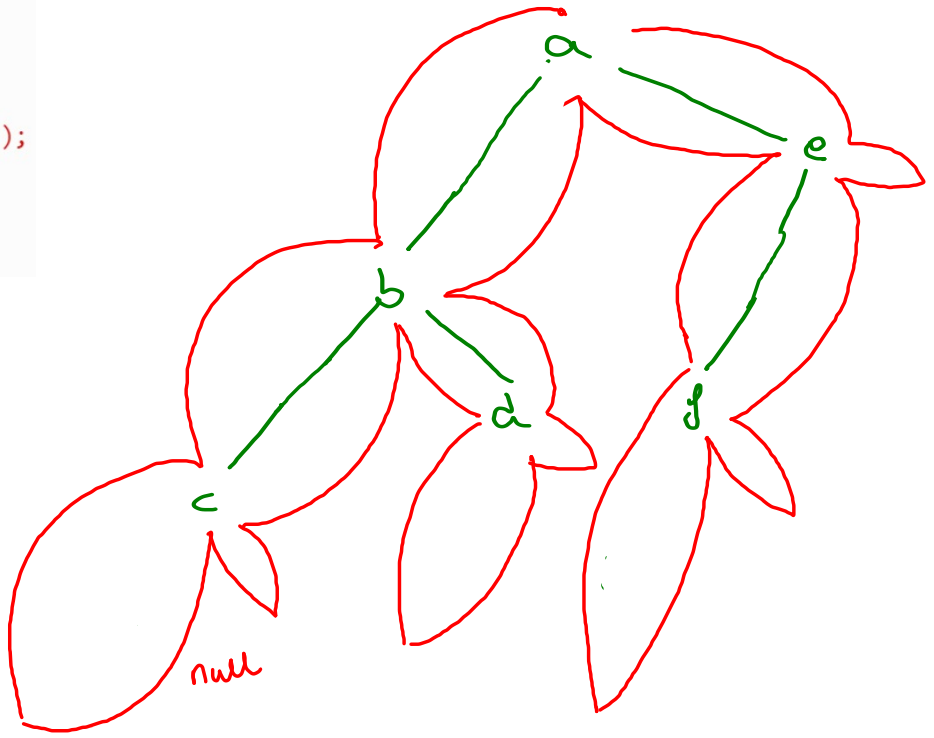
Transform To Left-cloned Tree



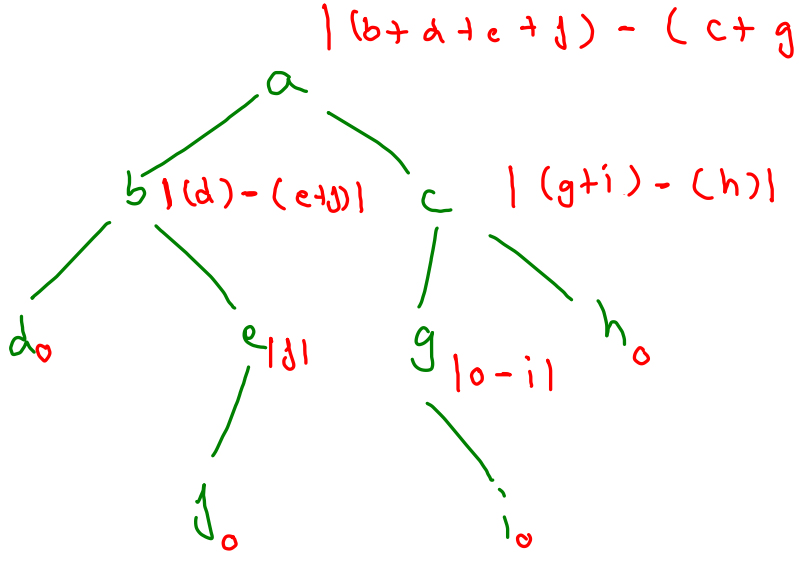
Transform To Normal From Left-cloned Tree



```java
public static Node transBackFromLeftClonedTree(Node node){
    if(node == null) {
        return null;
    }

    node.left = transBackFromLeftClonedTree(node.left.left);
    node.right = transBackFromLeftClonedTree(node.right);

    return node;
}
```



Tilt Of Binary Tree



tilt of a node :

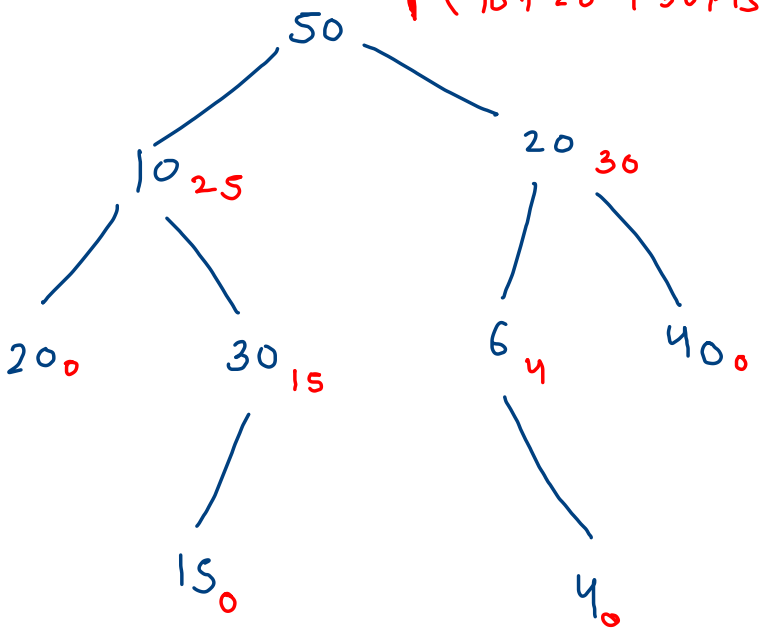
| left subtree sum − right subtree sum |

tilt of a tree

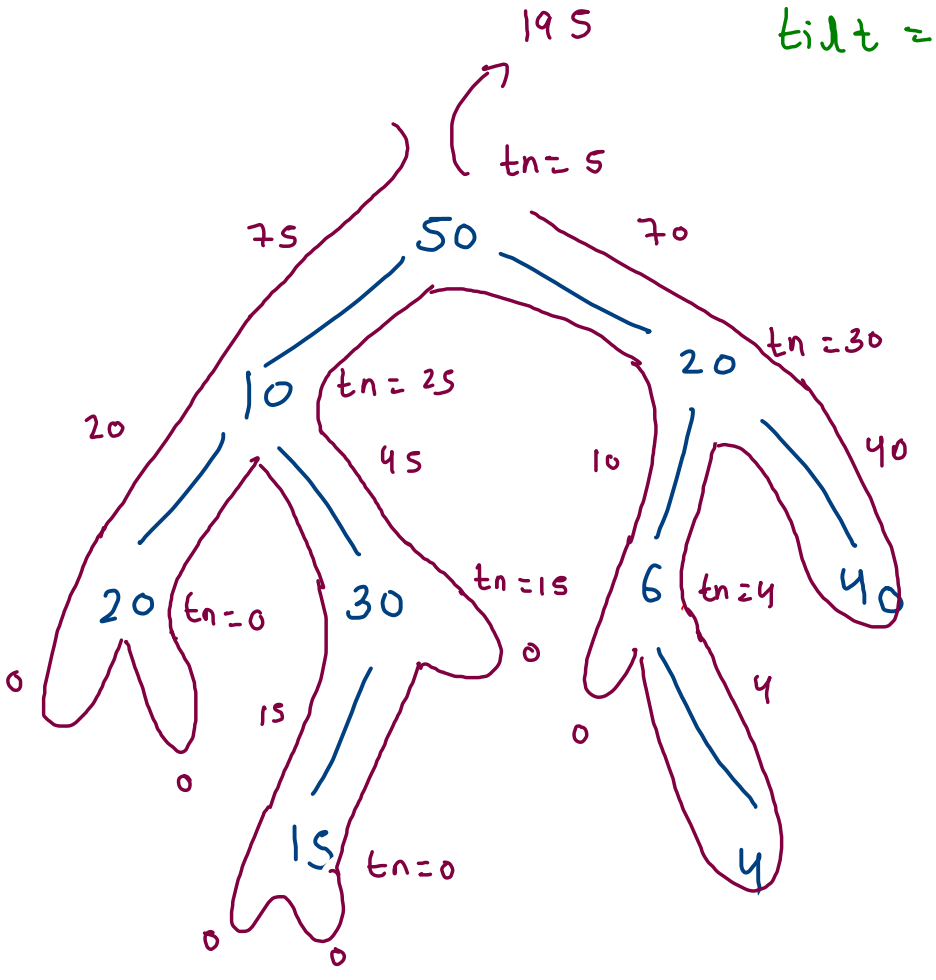
$\sum$ tilt

all nodes

$|(10+20+30+15) - (20+6+4+40)| = 5$



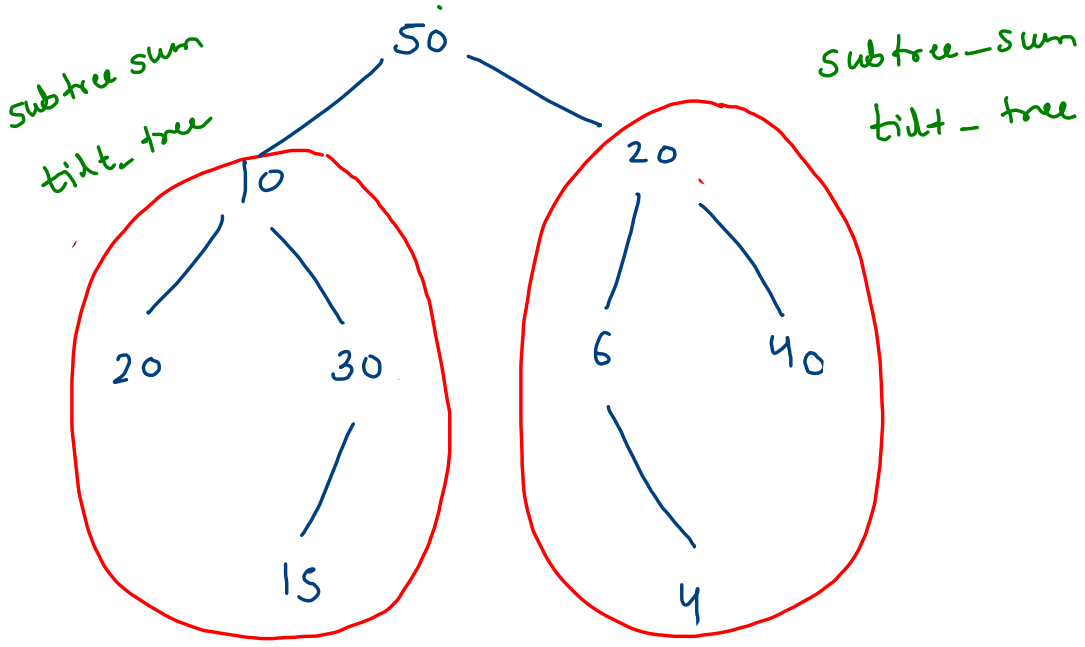
tilt of tree = 5+25+30+15+4



tilt = 15 + 25 + 4 + 30 + 5

```
static int tilt = 0;
public static int tilt(Node node){
    tilt = 0; //tile of binary tree

    sum(node);

    return tilt;
}
```

```
public static int sum(Node node) {
    if(node == null) {
        return 0;
    }

    int ls = sum(node.left); //left subtree sum
    int rs = sum(node.right); //right subtree sum

    int tilt_node = Math.abs(ls-rs); //tilt of a node

    tilt += tilt_node;

    return ls + rs + node.data;
}
```



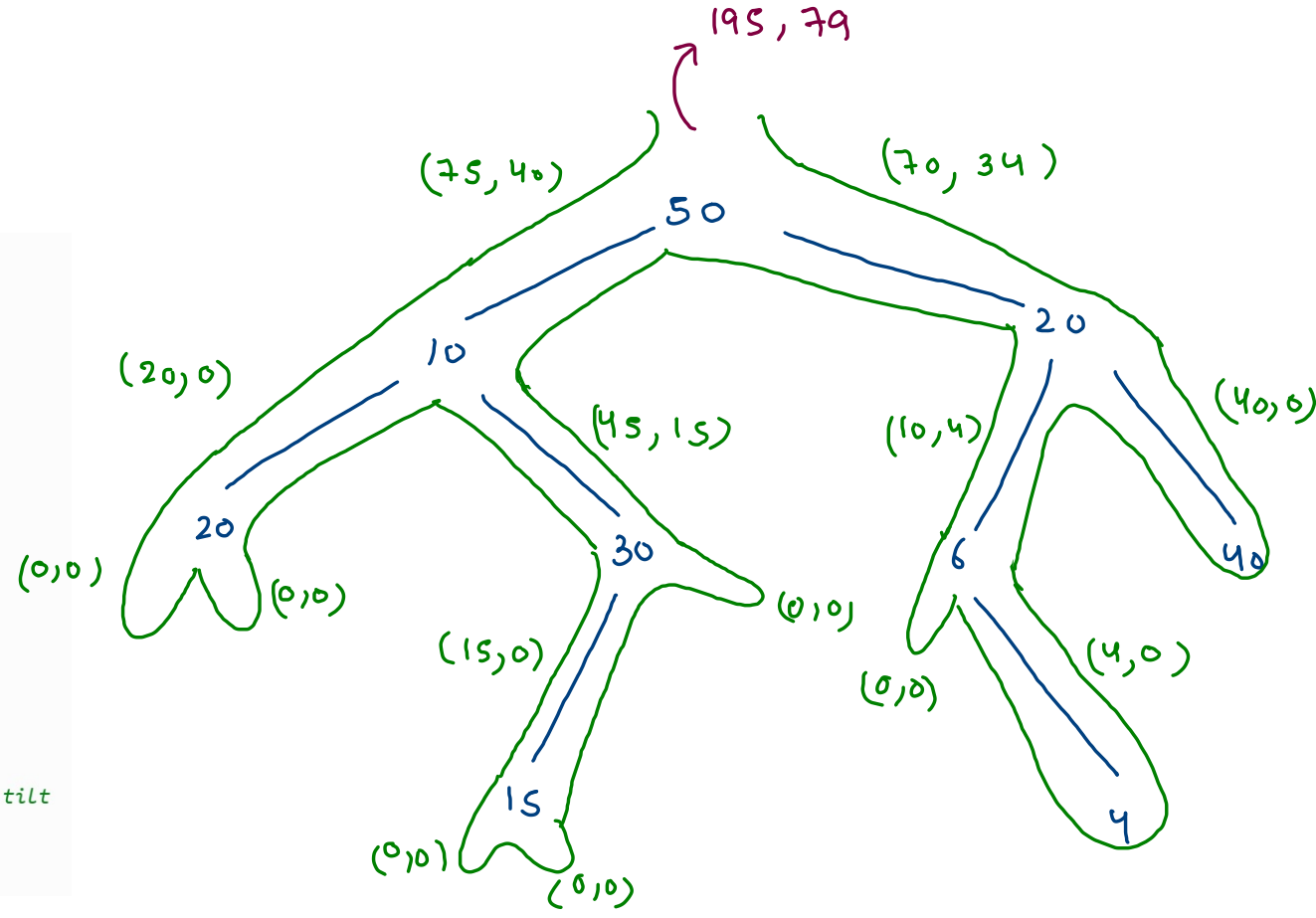
subtree_sum -> left.subtree_sum +
right.subtree_sum +
node.data;

tilt.node -> |lss - rss|

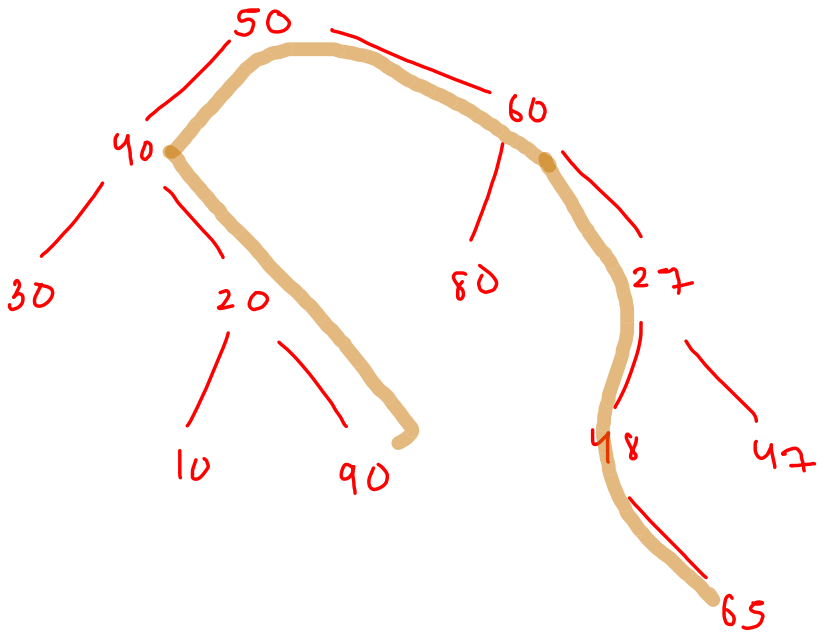
tilt_tree -> left.tilt_tree +
right.tilt_tree +
tilt_node;

(subtree sum, subtree tilt)

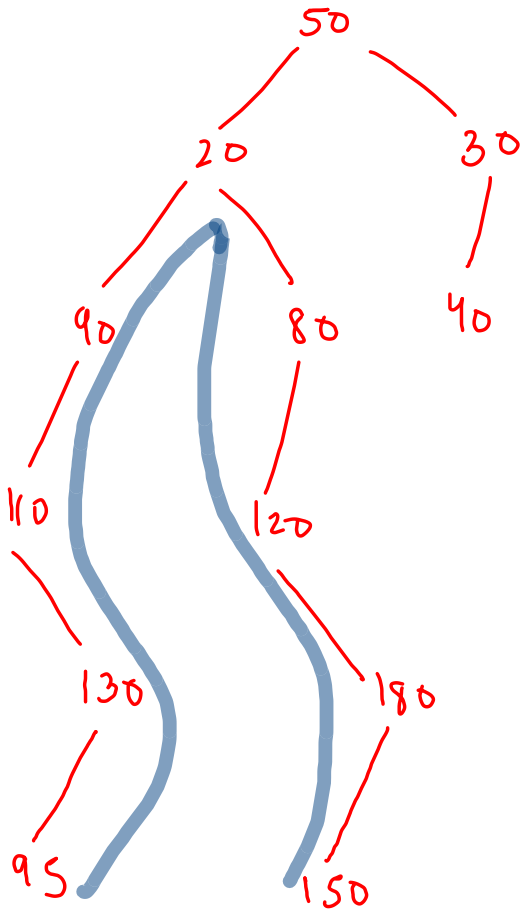
```
static class TPair {
    int ss; //subtree sum
    int tilt; //subtree tilt

    TPair(int ss,int tilt) {
        this.ss = ss;
        this.tilt = tilt;
    }
}


public static TPair tilt(Node node){
    if(node == null) {
        return new TPair(0,0);
    }

    TPair lp = tilt(node.left);
    TPair rp = tilt(node.right);

    int ss = lp.ss + rp.ss + node.data;
    int tilt_node = Math.abs(lp.ss - rp.ss);
    int tilt = lp.tilt + rp.tilt + tilt_node ; //subtree tilt

    return new TPair(ss,tilt);
}
```



Diameter Of A Binary Tree

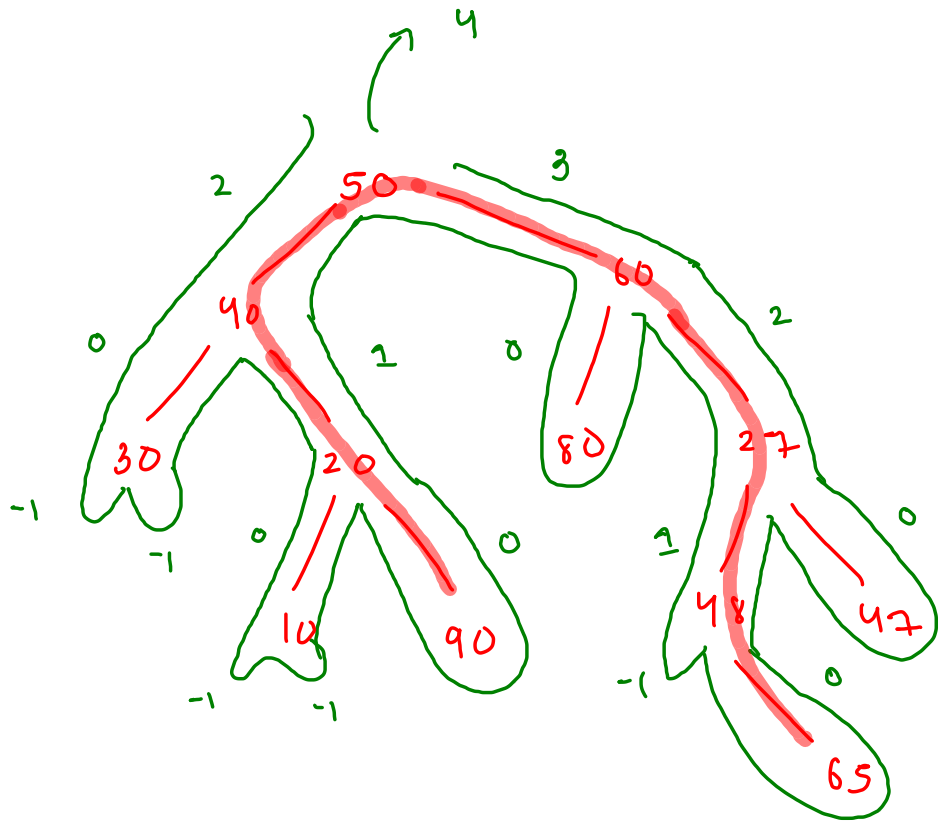


dist = lht + rht + 2

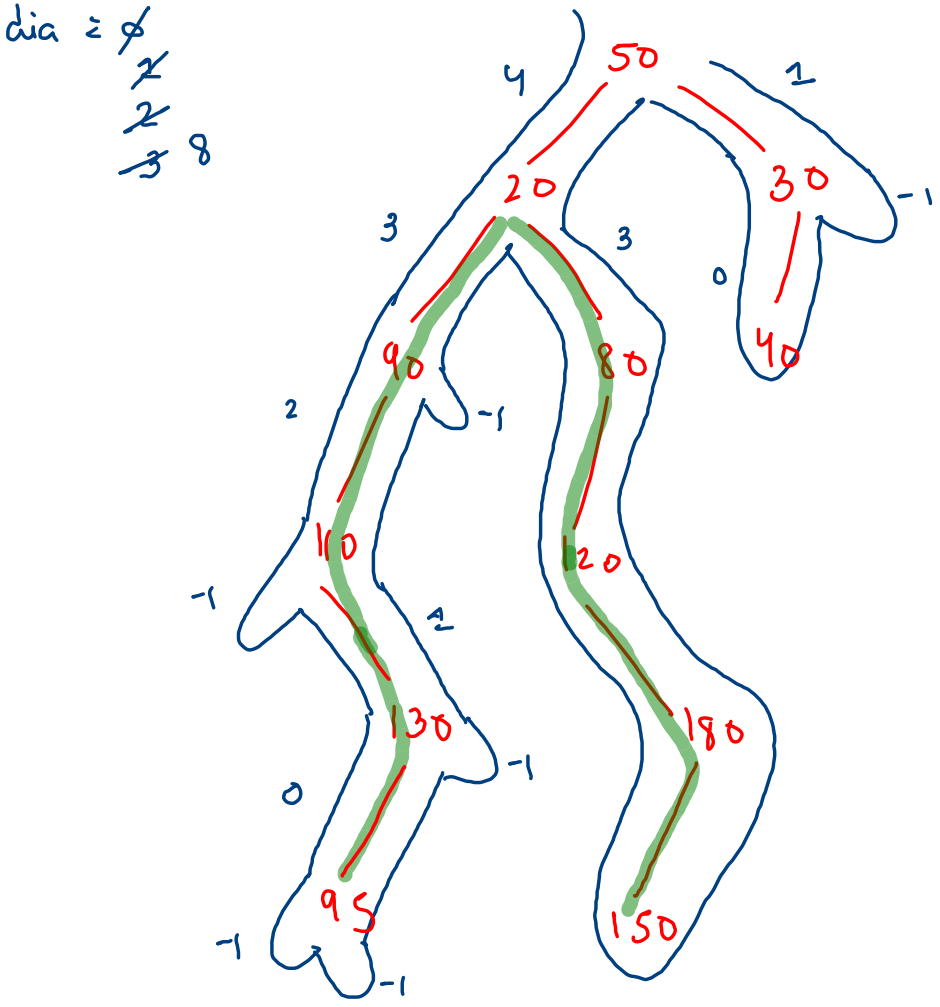

50
20
30
90
80
40
110
120
130
180
95
150

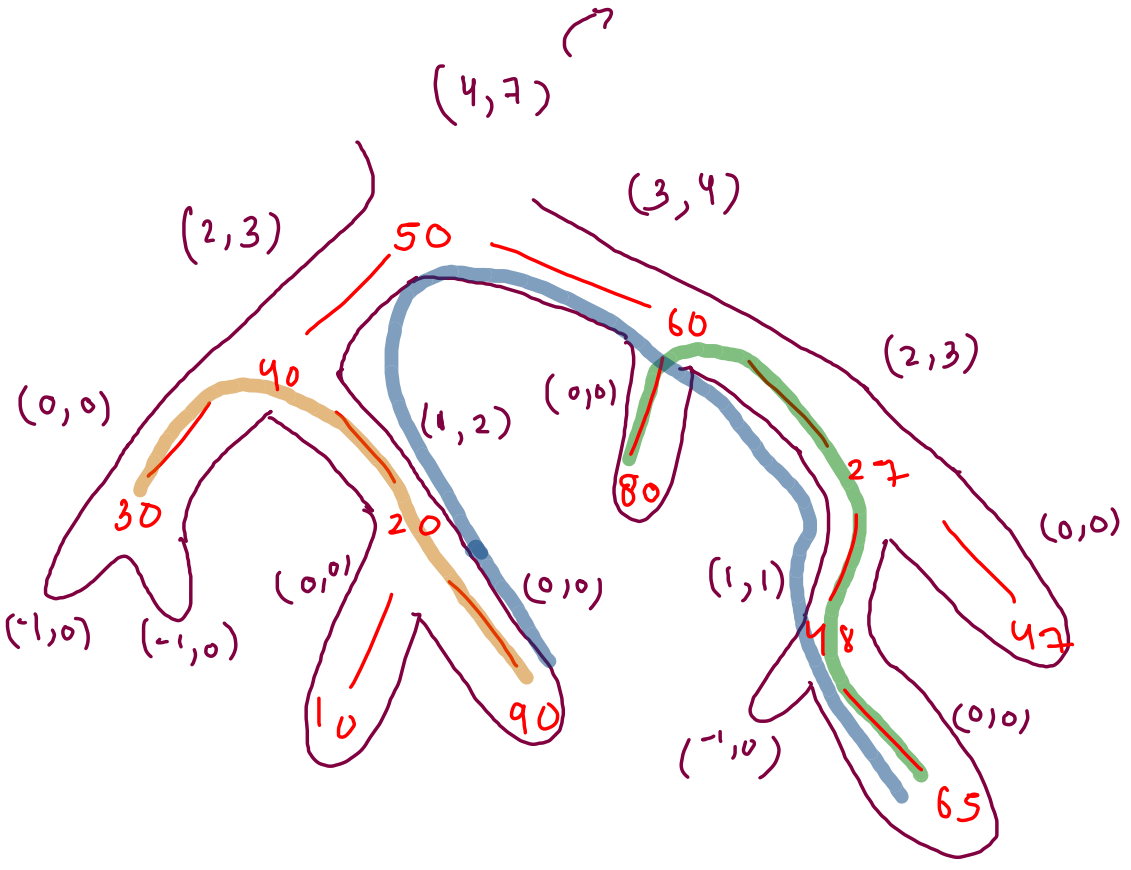
dia = ~~0~~ ~~2~~ ~~3~~ ~~4~~ 7

```java
public static int height(Node node) {
  if (node == null) {
    return -1;
  }

  int lh = height(node.left);
  int rh = height(node.right);

  int dist = lh + rh + 2;

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

  int th = Math.max(lh, rh) + 1;
  return th;
}

static int dia;
public static int diameter1(Node node) {
  dia = 0;

  height(node);

  return dia;
}
```



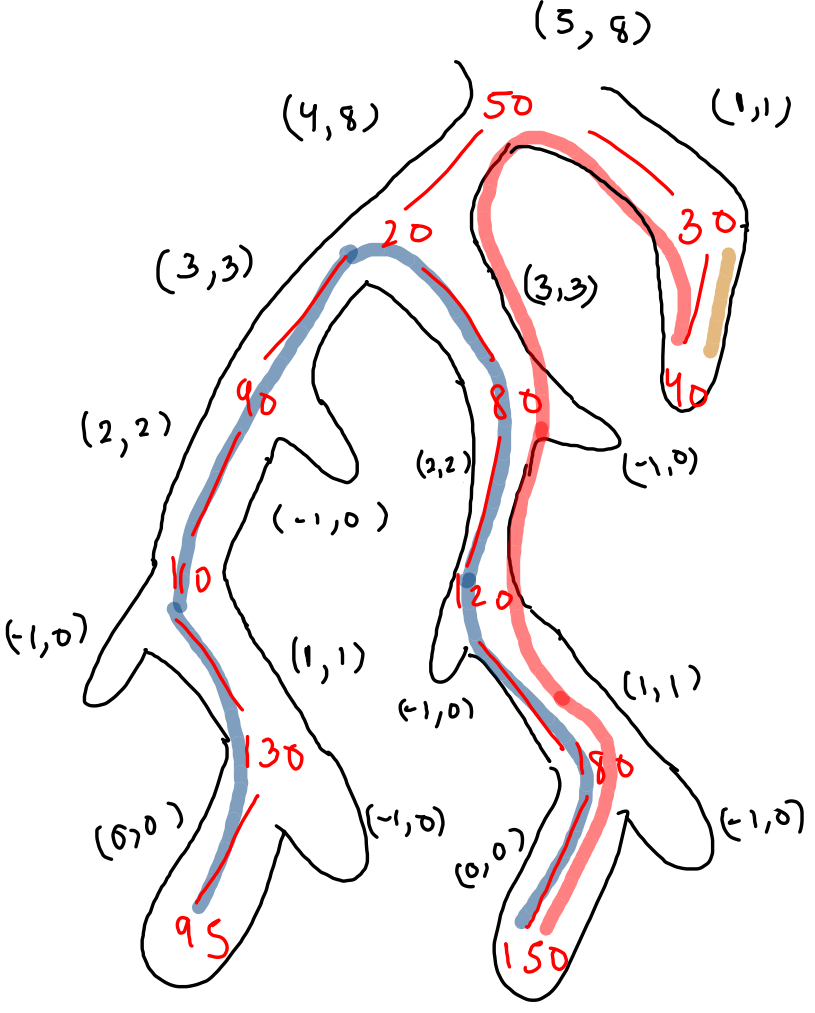
```java
public static int height(Node node) {
  if (node == null) {
    return -1;
  }

  int lh = height(node.left);
  int rh = height(node.right);

  int dist = lh + rh + 2;

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

  int th = Math.max(lh, rh) + 1;
  return th;
}

static int dia;
public static int diameter1(Node node) {
  dia = 0;

  height(node);

  return dia;
}
```





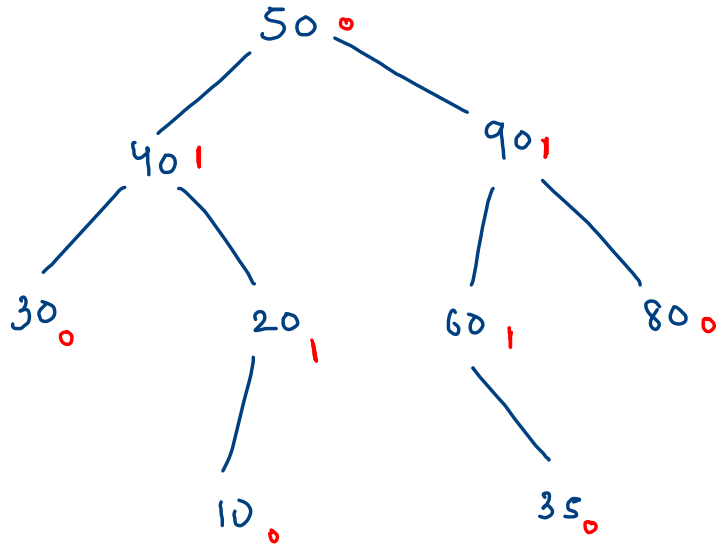
ht , dia

```
static class DPair {
    int ht; //subtree height
    int dia; //subtree diameter

    DPair(int ht,int dia) {
        this.ht = ht;
        this.dia = dia;
    }
}

public static DPair diameter(Node node) {
    if(node == null) {
       return new DPair(-1,0);
    }

    DPair lp = diameter(node.left);
    DPair rp = diameter(node.right);

    int dist = lp.ht + rp.ht + 2;
    int ht = Math.max(lp.ht,rp.ht) + 1;
    int dia = Math.max(Math.max(lp.dia,rp.dia),dist);

    return new DPair(ht,dia);
}
```




```java
static class DPair {
    int ht; //subtree height
    int dia; //subtree diameter

    DPair(int ht,int dia) {
        this.ht = ht;
        this.dia = dia;
    }
}

public static DPair diameter(Node node) {
    if(node == null) {
        return new DPair(-1,0);
    }

    DPair lp = diameter(node.left);
    DPair rp = diameter(node.right);

    int dist = lp.ht + rp.ht + 2;
    int ht = Math.max(lp.ht,rp.ht) + 1;
    int dia = Math.max(Math.max(lp.dia,rp.dia),dist);

    return new DPair(ht,dia);
}
```

Is Balanced Tree

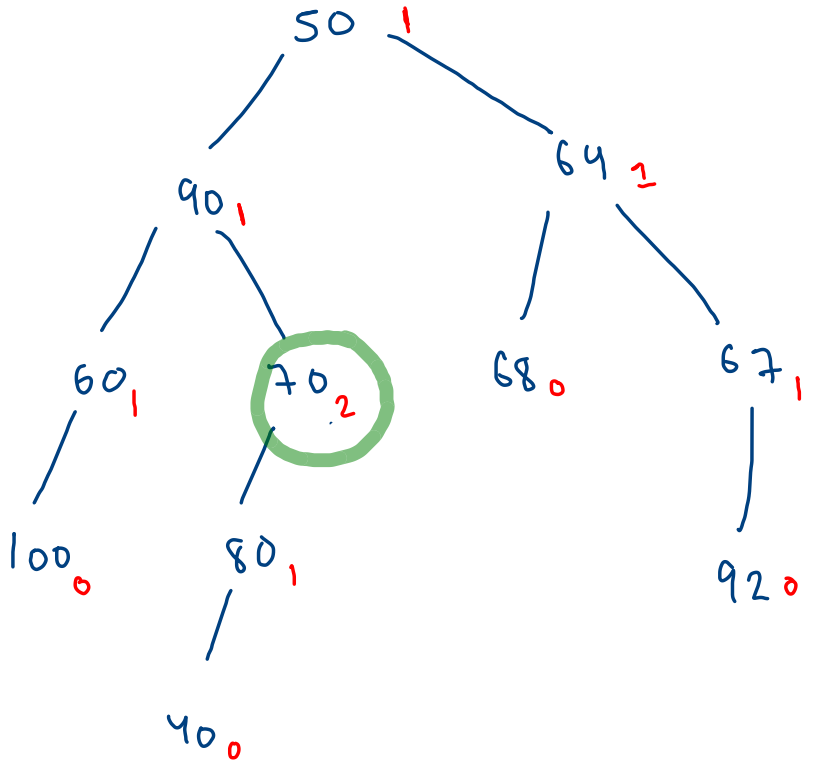


balancing factor

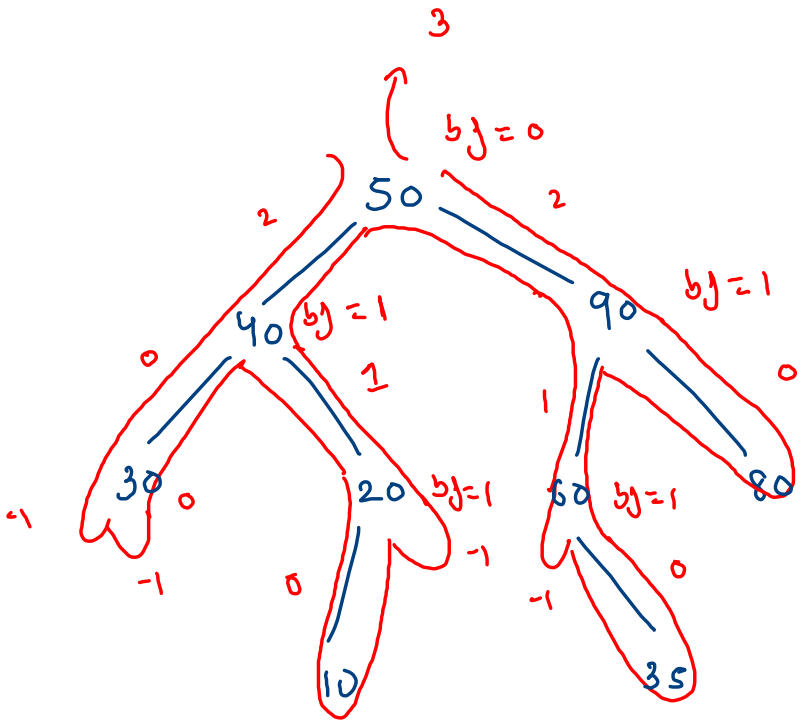
bf = |lh - rh|

A node is not balanced when bf ≥ 2

A tree is balanced when all of its one balanced.

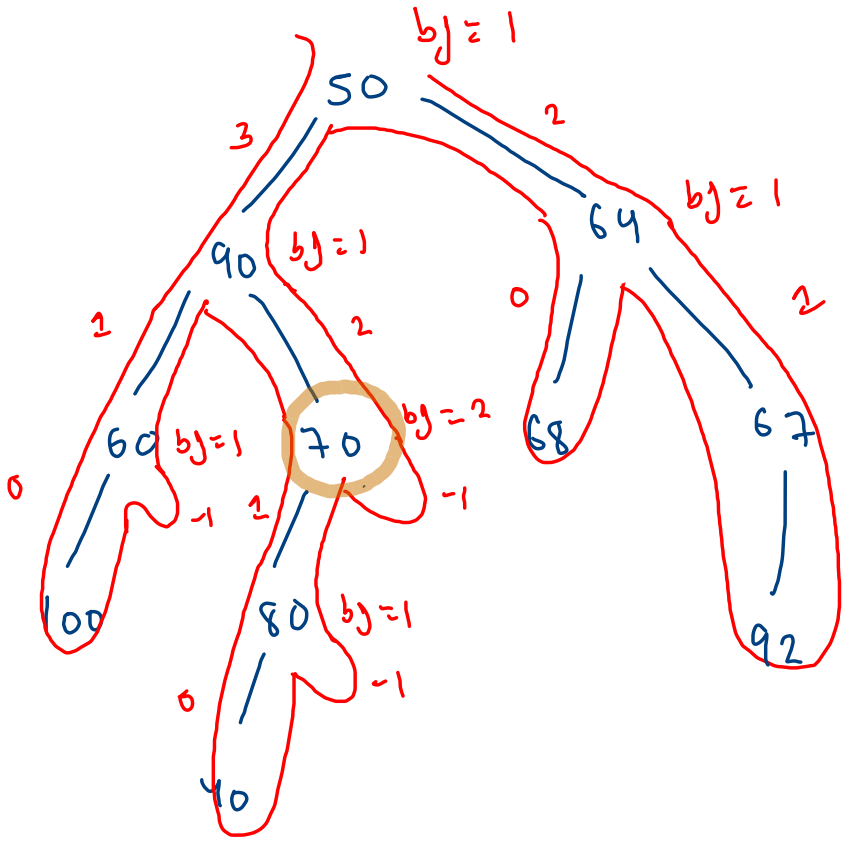


isBal = T;



isBal = T

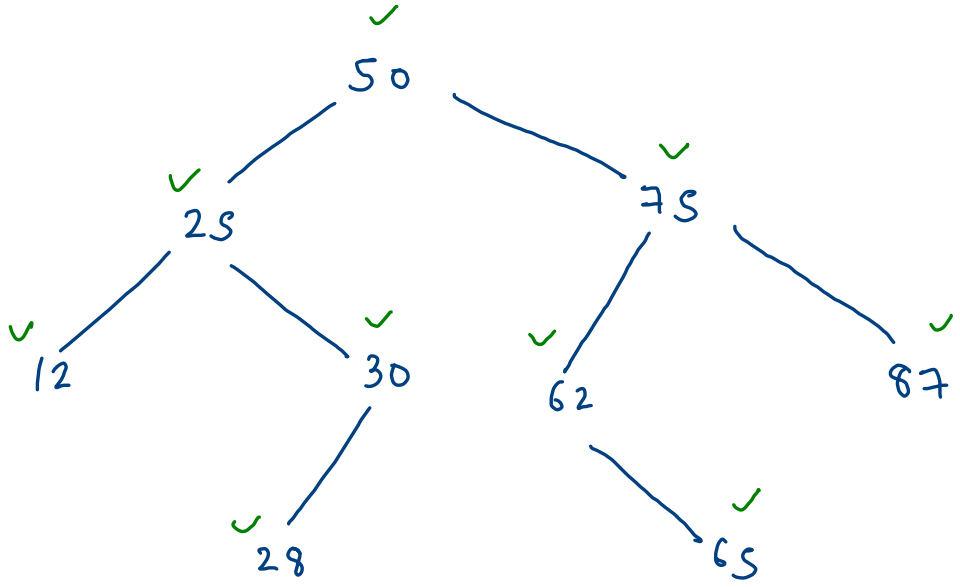
```java
public static int height(Node node) {
    if(node == null) {
        return -1;
    }

    int lh = height(node.left);
    int rh = height(node.right);

    int bf = Math.abs(lh - rh); //balancing factor of this node

    if(bf >= 2) {
        //this node is not balanced
        isBal = false;
    }

    return Math.max(lh,rh) + 1;
}
```



isBal = T / F

```java
public static int height(Node node) {
    if(node == null) {
        return -1;
    }

    int lh = height(node.left);
    int rh = height(node.right);

    int bf = Math.abs(lh - rh); //balancing factor of this node

    if(bf >= 2) {
        //this node is not balanced
        isBal = false;
    }

    return Math.max(lh,rh) + 1;
}
```

Is A Binary Search Tree

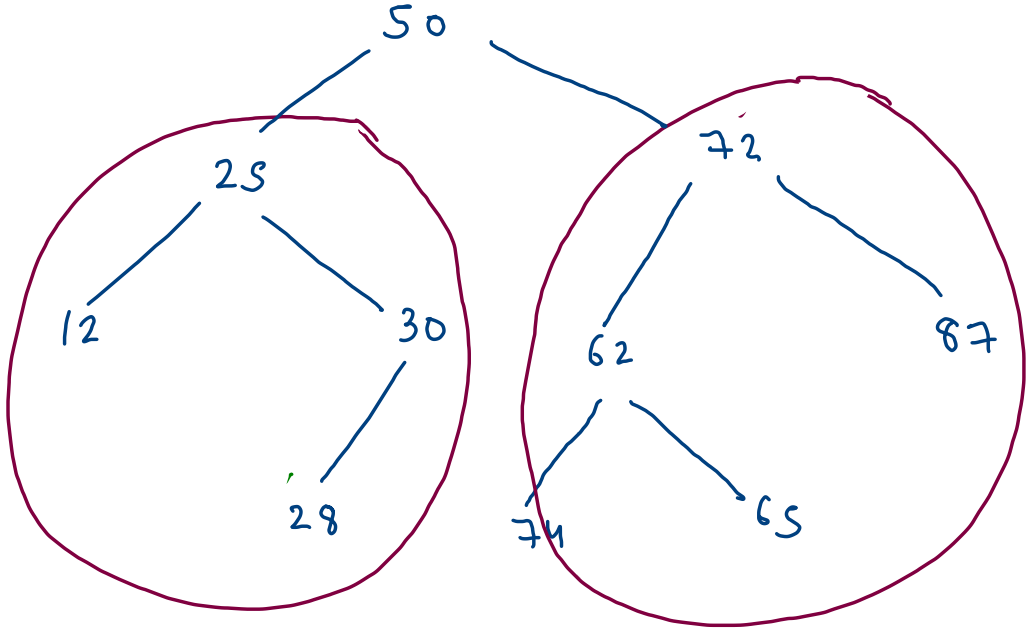


→ Node property

all nodes < node.data < all nodes
in its in its
left subtree right subtree

when a tree is BST :

when all nodes follows the above property.

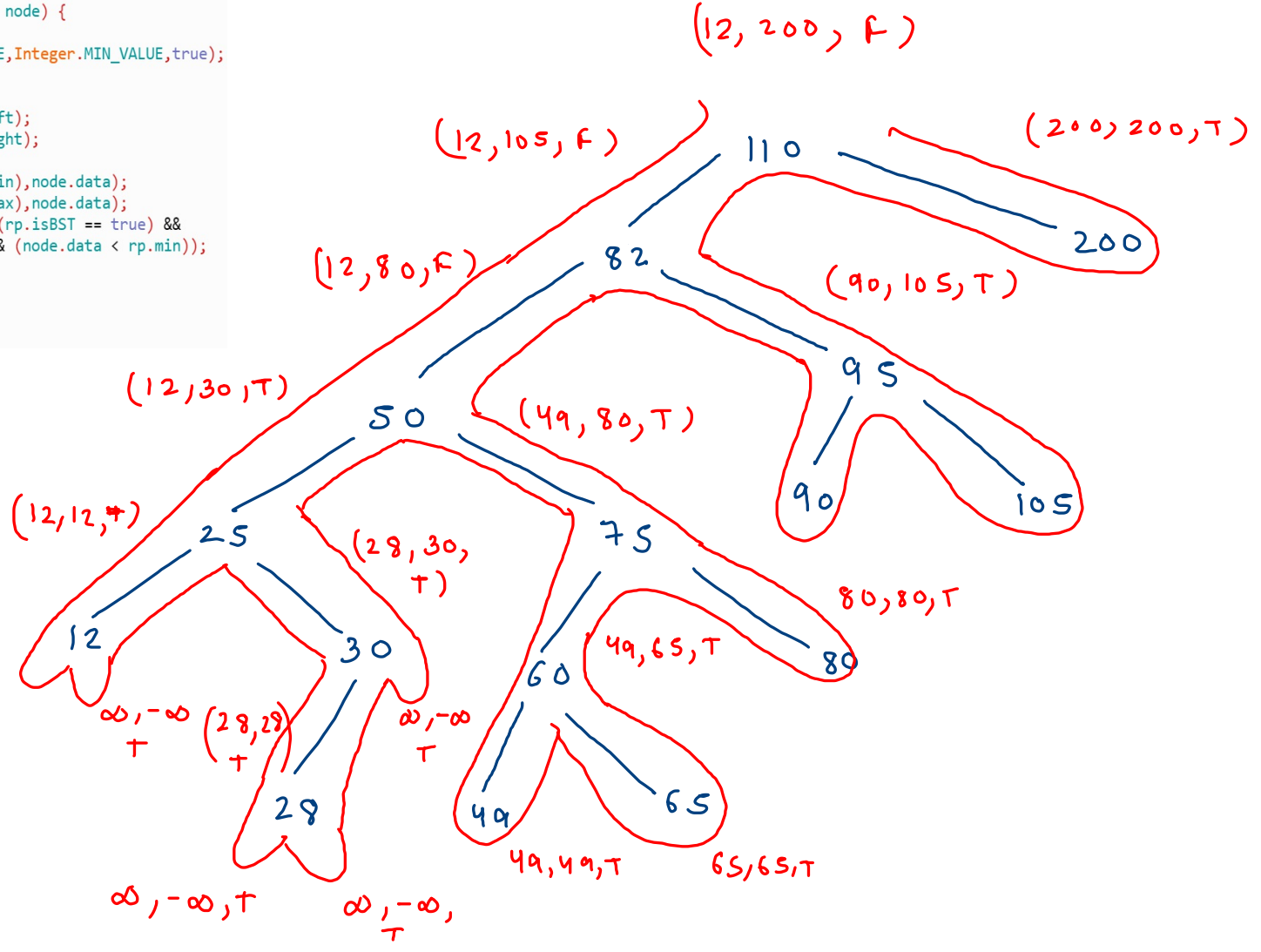



```
Pair {
    int min;
    int max;
    bool isBST
}
```

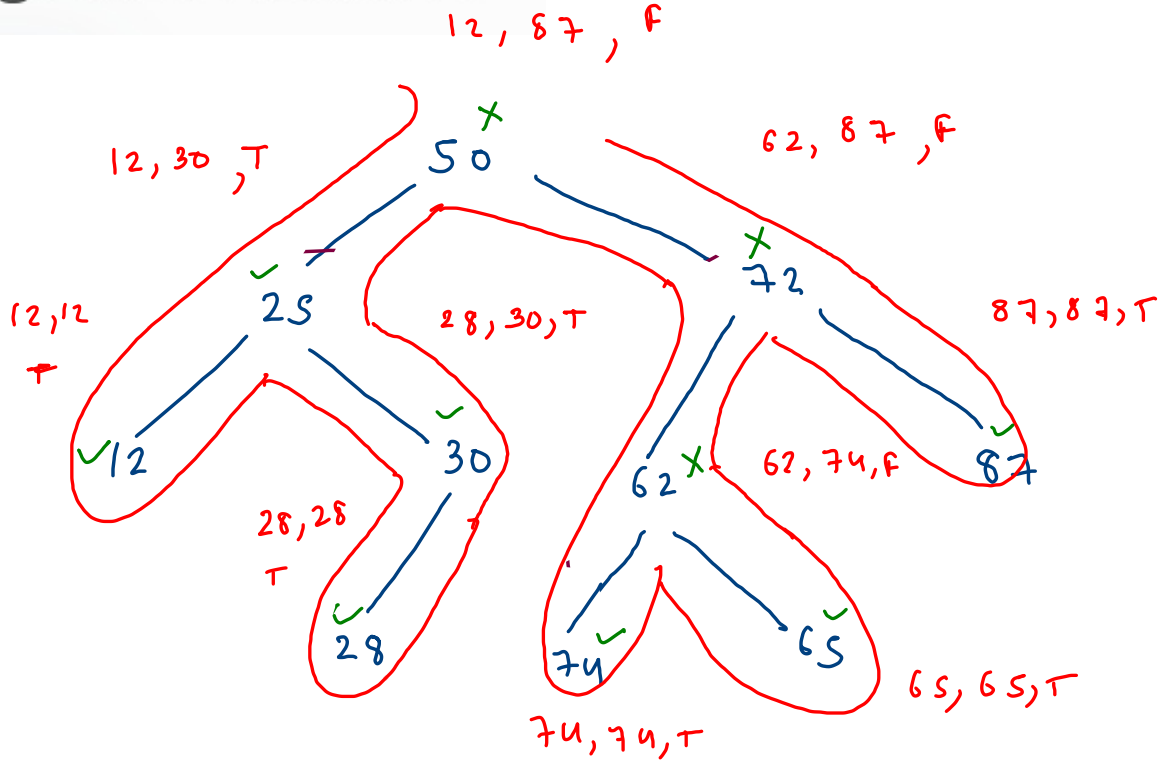
```java
public static BSTPair isBinarySearchTree(Node node) {
    if(node == null) {
        return new BSTPair(Integer.MAX_VALUE,Integer.MIN_VALUE,true);
    }

    BSTPair lp = isBinarySearchTree(node.left);
    BSTPair rp = isBinarySearchTree(node.right);

    int min = Math.min(Math.min(lp.min,rp.min),node.data);
    int max = Math.max(Math.max(lp.max,rp.max),node.data);
    boolean isBST = (lp.isBST == true)  && (rp.isBST == true) &&
                    ((lp.max < node.data) && (node.data < rp.min));

    return new BSTPair(min,max,isBST);
}
```

min, max, isBST

```java
public static class BSTPair {
    int min;
    int max;
    boolean isBST;

    BSTPair(int min,int max,boolean isBST) {
        this.min = min;
        this.max = max;
        this.isBST = isBST;
    }
}
```



Largest Bst Subtree



25 @ 4

Pair {

```
int min;
int max;
boolean isBST;
Node lbstsn
int lbstsz;
```

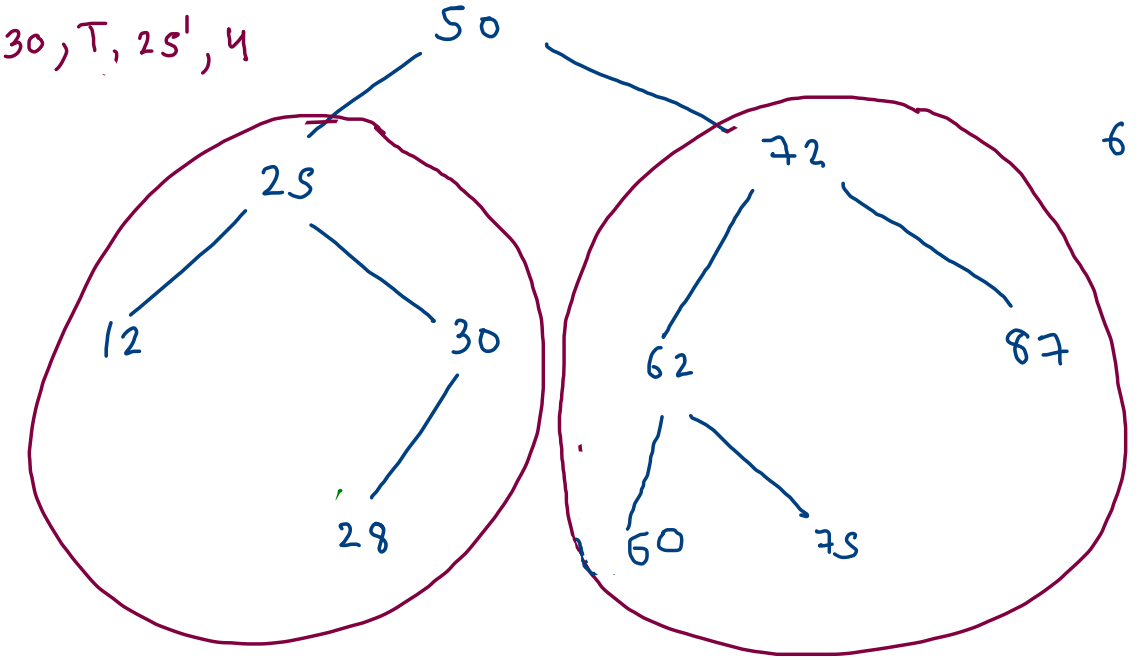
}

lbstsn → largest bst subtree node

lbstsz → largest bst subtree size

12, 87, F, 25', 4

12, 30, T, 25', 4



60, 87, F, 62', 3

```java
int min = Math.min(Math.min(lp.min,rp.min),node.data);
int max = Math.max(Math.max(lp.max,rp.max),node.data);
boolean isBST = (lp.isBST == true) && (rp.isBST == true) &&
                ((lp.max < node.data) && (node.data < rp.min));

Node lbstnode = null; //Largest bst subtree node
int  size = 0; //Largest bst subtree size

if(isBST == true) {
    //the subtree rooted at node is a bst
    lbstnode = node;
    size = lp.size + rp.size + 1;
}
else {
    if(lp.size > rp.size) {
        lbstnode = lp.node;
        size = lp.size;
    }
    else {
        lbstnode = rp.node;
        size = rp.size;
    }
}
```