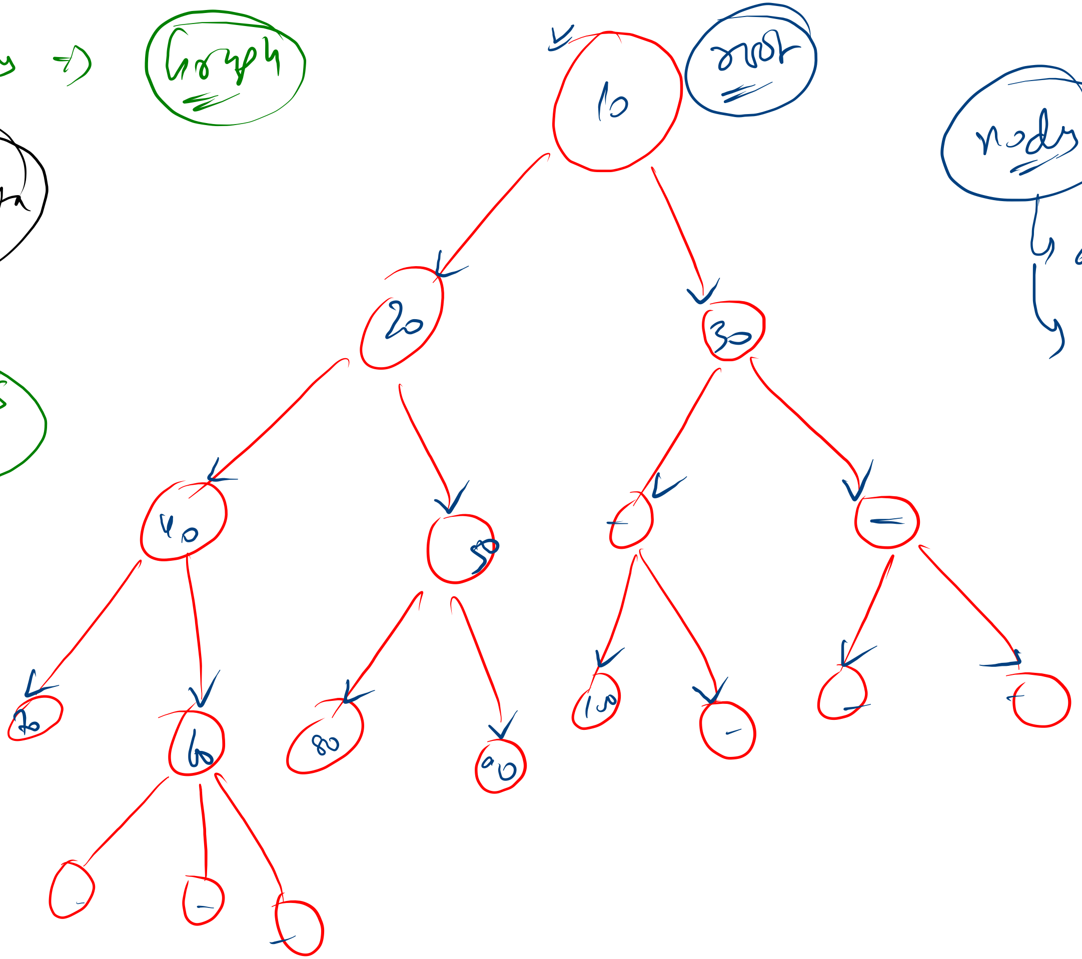
Trees + Cycle → Graph

hierarchical data

Level order → BFS

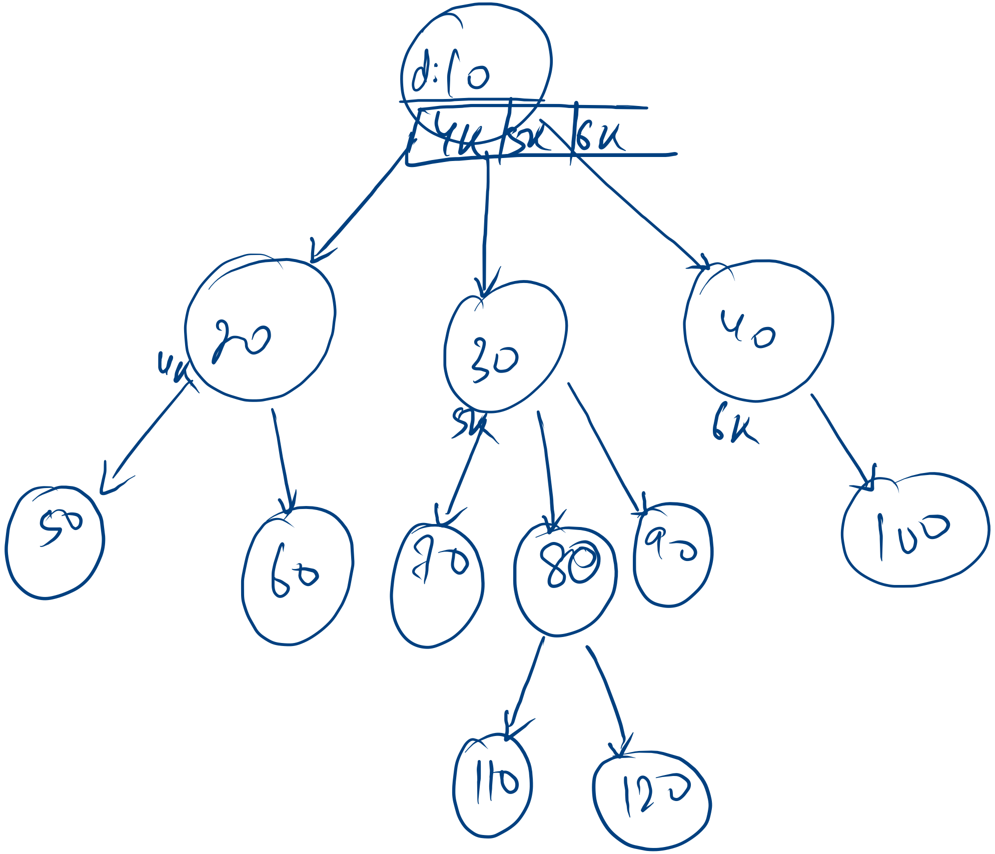


node
↳ data
↳ next*

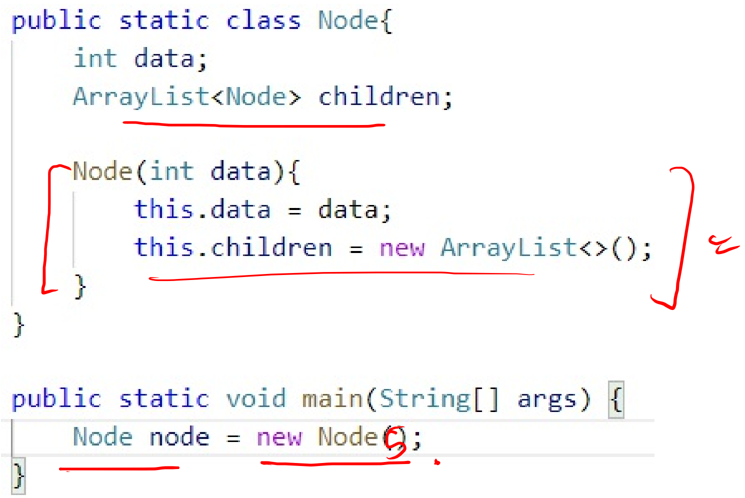
Trees

- Generic Tree
- Binary Tree
- BST
- AVL
- Red black
- Fenwik
- Segment
- B+
- B -Tree

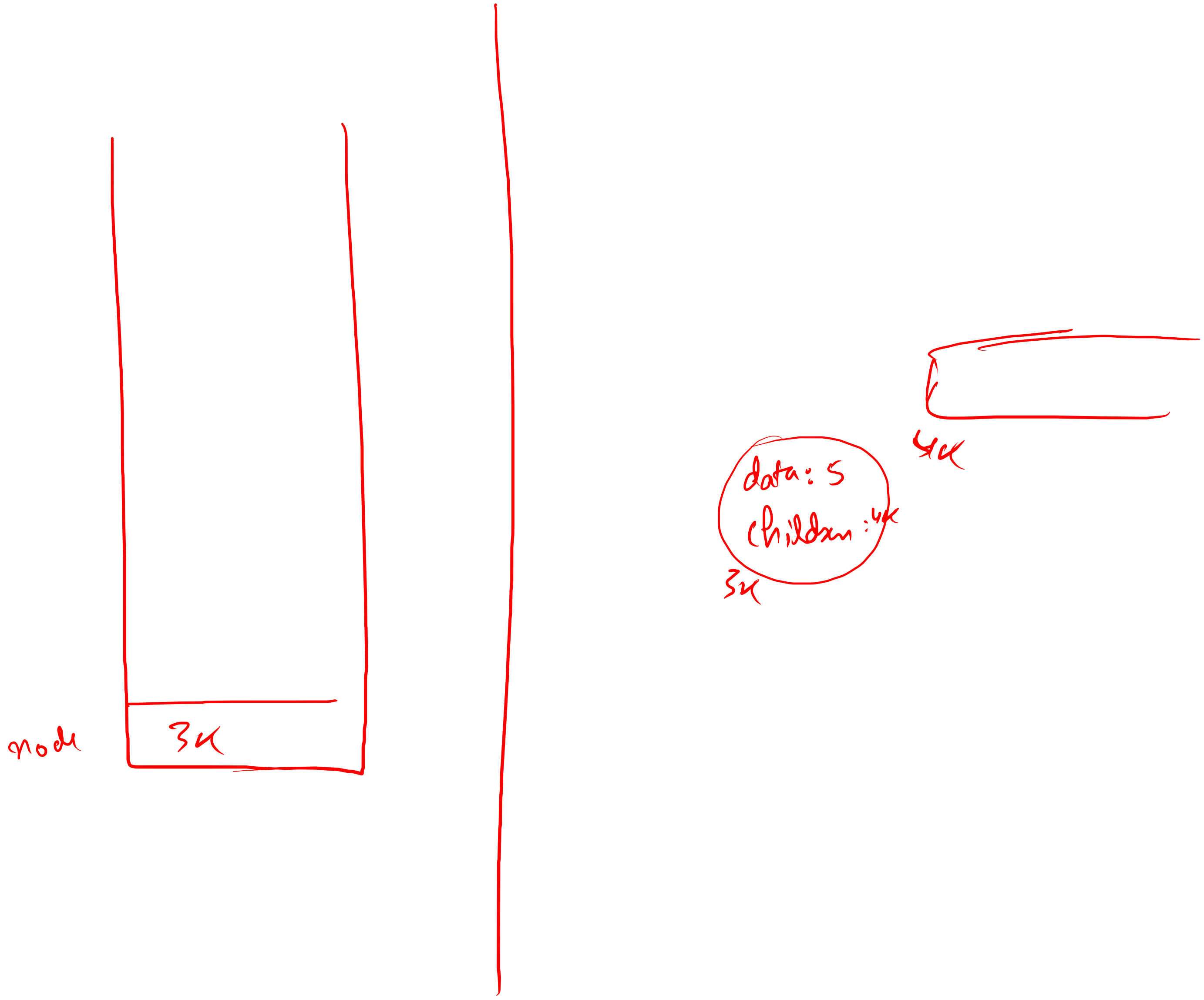
Generic Tree :- Every node has multiple child subtrees

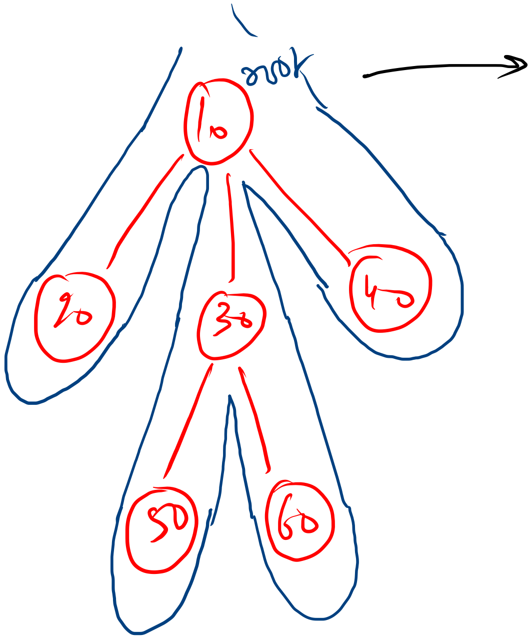


Node
- data
- Arraylist<Node> children



```
public static class Node{
    int data;
    ArrayList<Node> children;

    Node(int data){
        this.data = data;
        this.children = new ArrayList<>();
    }
}

public static void main(String[] args) {
    Node node = new Node(5);
}
```





10

20

-1

30

50

-1

60

-1

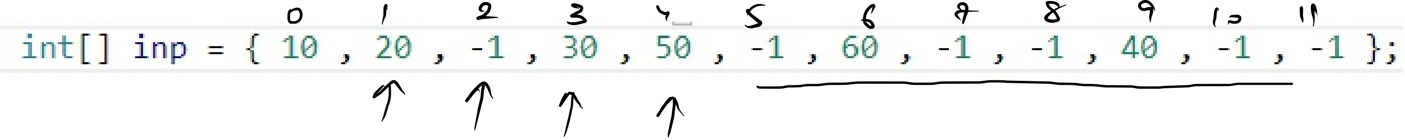
-1

40

-1

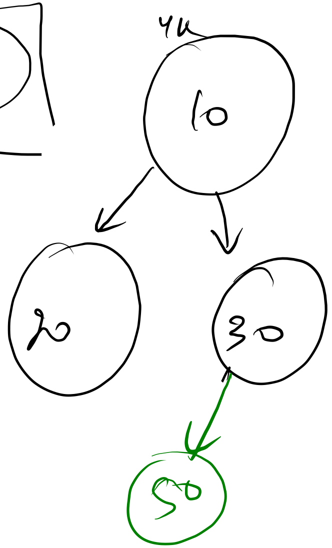
-1

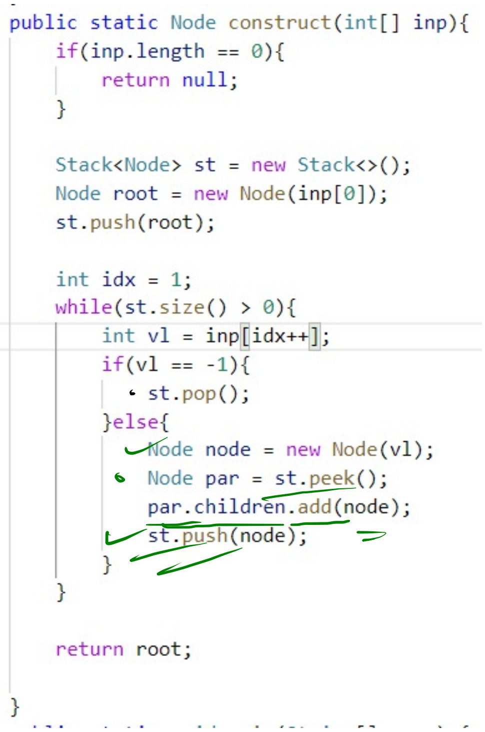
{ 10 , 20 , -1 , 30 , 50 , -1 , 60 , -1 , -1 , 40 , -1 , -1 }



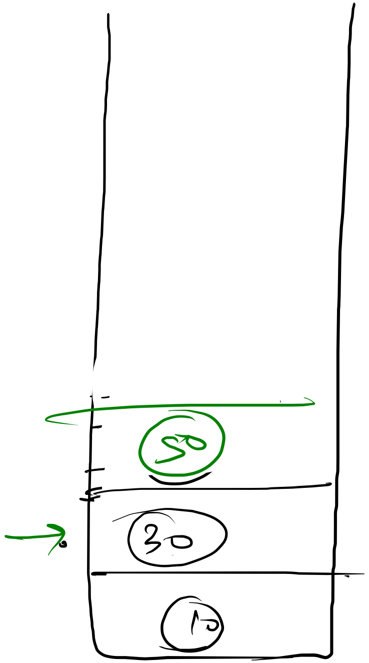
```
int[] inp = { 10 , 20 , -1 , 30 , 50 , -1 , 60 , -1 , -1 , 40 , -1 , -1 };
```

idx = ~~1~~ ~~2~~ ~~3~~ 4 5





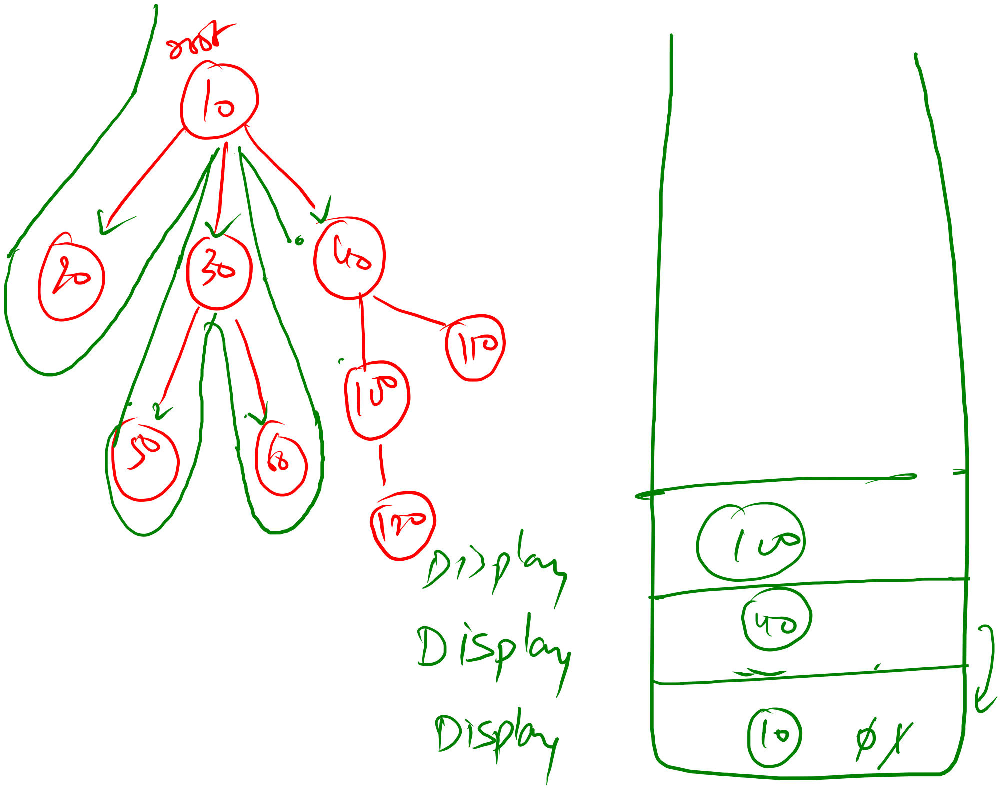
```
public static Node construct(int[] inp){
    if(inp.length == 0){
        return null;
    }

    Stack<Node> st = new Stack<>();
    Node root = new Node(inp[0]);
    st.push(root);

    int idx = 1;
    while(st.size() > 0){
        int vl = inp[idx++];
        if(vl == -1){
            st.pop();
        }else{
            Node node = new Node(vl);
            Node par = st.peek();
            par.children.add(node);
            st.push(node);
        }
    }

    return root;

}
```

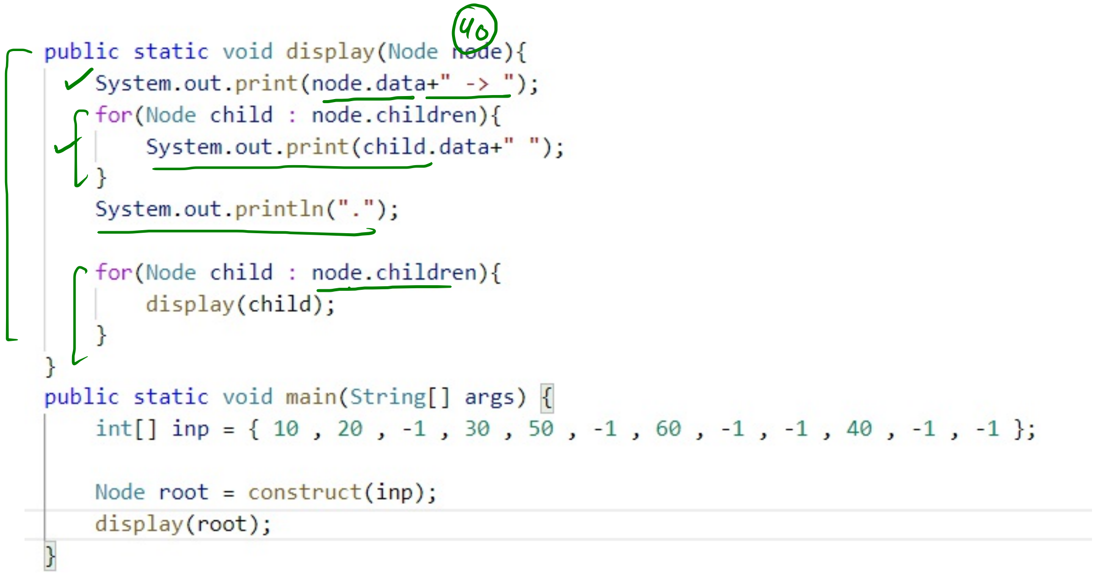


NOTES

val == -1 => no further child

val != -1 => valid input data





```
public static void display(Node node){
    System.out.print(node.data+" -> ");
    for(Node child : node.children){
        System.out.print(child.data+" ");
    }
    System.out.println(".");

    for(Node child : node.children){
        display(child);
    }
}
public static void main(String[] args) {
    int[] inp = { 10 , 20 , -1 , 30 , 50 , -1 , 60 , -1 , -1 , 40 , -1 , -1 };

    Node root = construct(inp);
    display(root);
}
```

10 → 20 30 40 .

20 → .

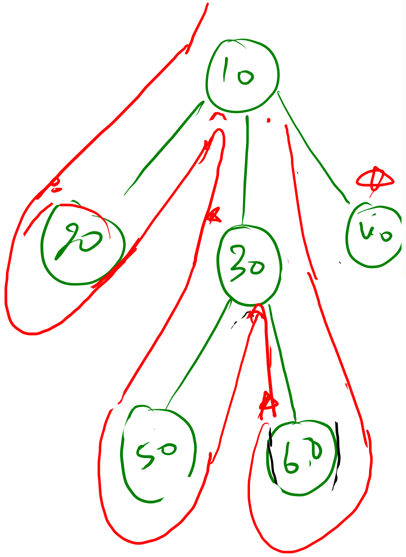
30 → 50 60 .

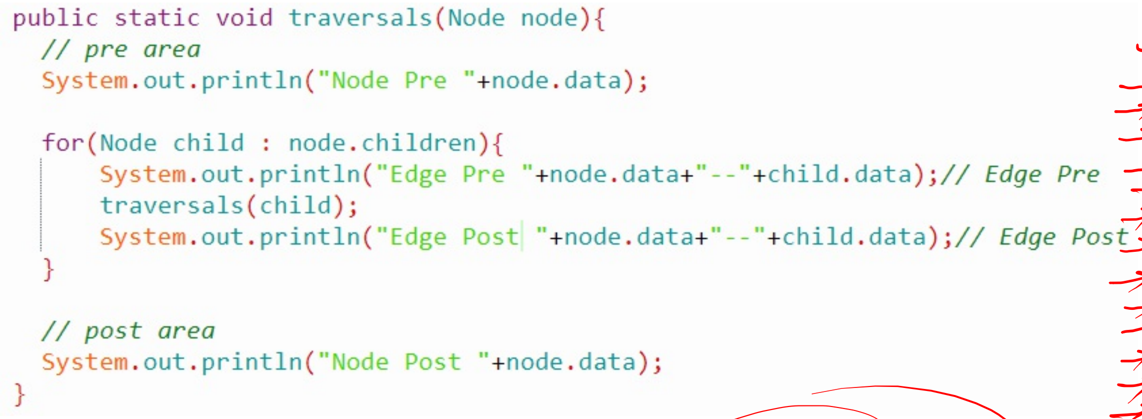
50 → .

60 → .

40 → 100 110 .

Preorder - Postorder





```
public static void traversals(Node node){
  // pre area
  System.out.println("Node Pre "+node.data);

  for(Node child : node.children){
      System.out.println("Edge Pre "+node.data+"--"+child.data);// Edge Pre
      traversals(child);
      System.out.println("Edge Post "+node.data+"--"+child.data);// Edge Post
  }

  // post area
  System.out.println("Node Post "+node.data);
}
```

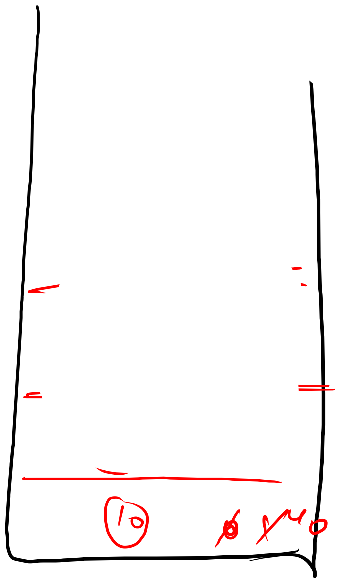
```
Node Pre 10
Edge Pre 10--20
Node Pre 20
Node Post 20
Edge Post 10--20
Edge Pre 10--30
Node Pre 30
Edge Pre 30--50
Node Pre 50
Node Post 50
Edge Post 30--50
Edge Pre 30--60
Node Pre 60
Node Post 60
Edge Post 30--60
Node Post 30
Edge Post 10--30
Edge Pre 10--40
Node Pre 40
Node Post 40
Edge Post 10--40
Node Post 10
```

Time => O(n)

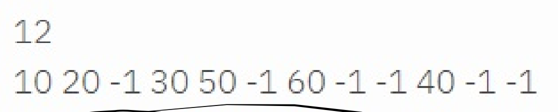
S.C. => O(h)

Recursion stack

Tra

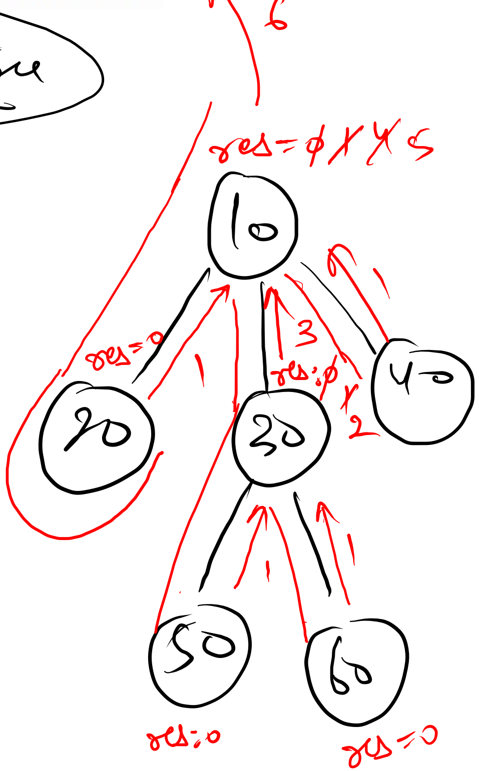


```
Node Pre 10
Node Pre 20
Node Post 20
Node Pre 30
Node Pre 50
Node Post 50
Node Pre 60
Node Post 60
Node Post 30
Node Pre 40
Node Post 40
Node Post 10
```

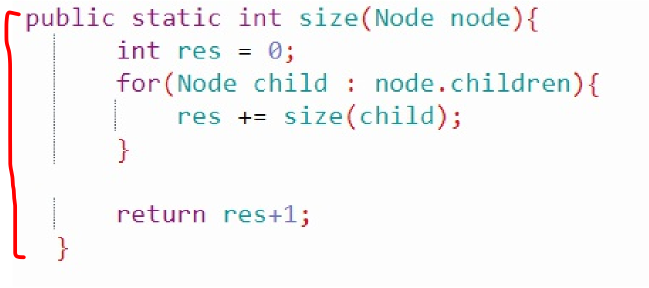


```
12
10 20 -1 30 50 -1 60 -1 -1 40 -1 -1
```

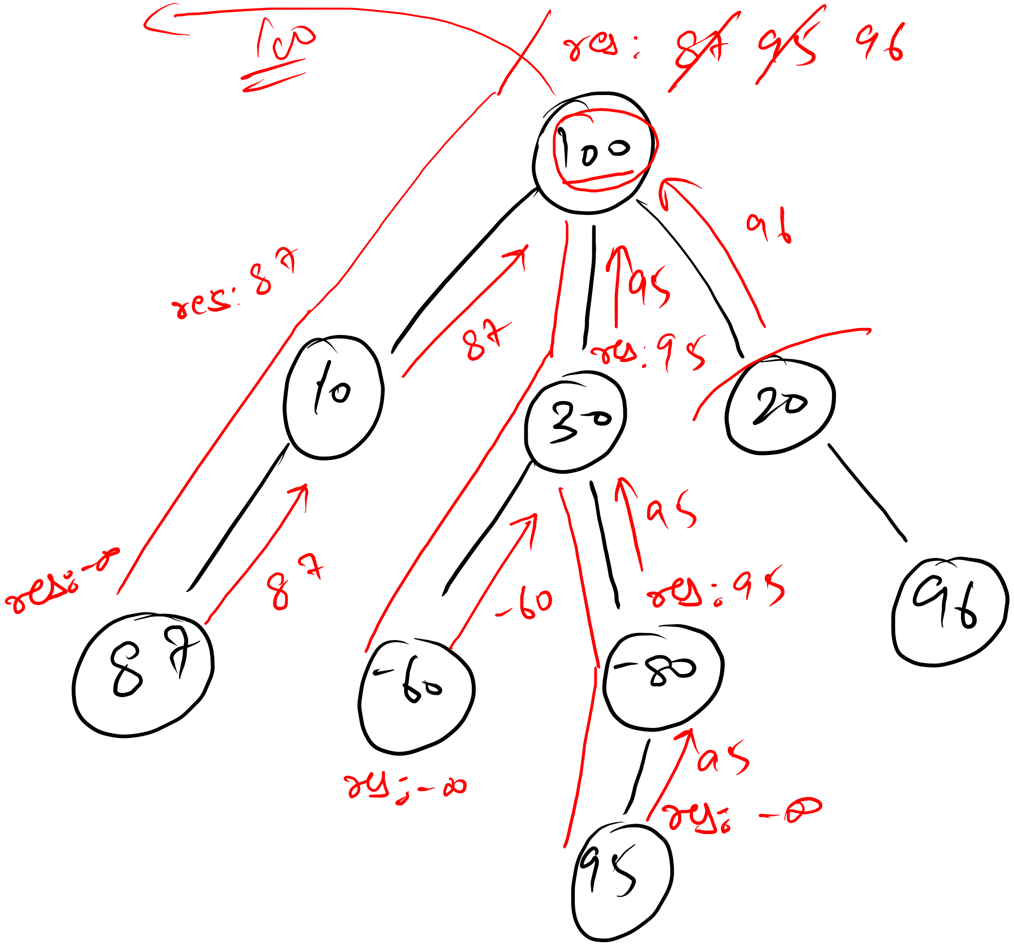
Size of Generic Tree

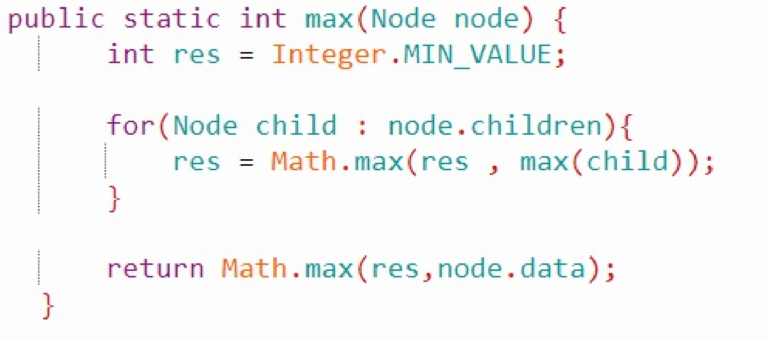


6



```
public static int size(Node node){
    int res = 0;
    for(Node child : node.children){
        res += size(child);
    }

    return res+1;
}
```



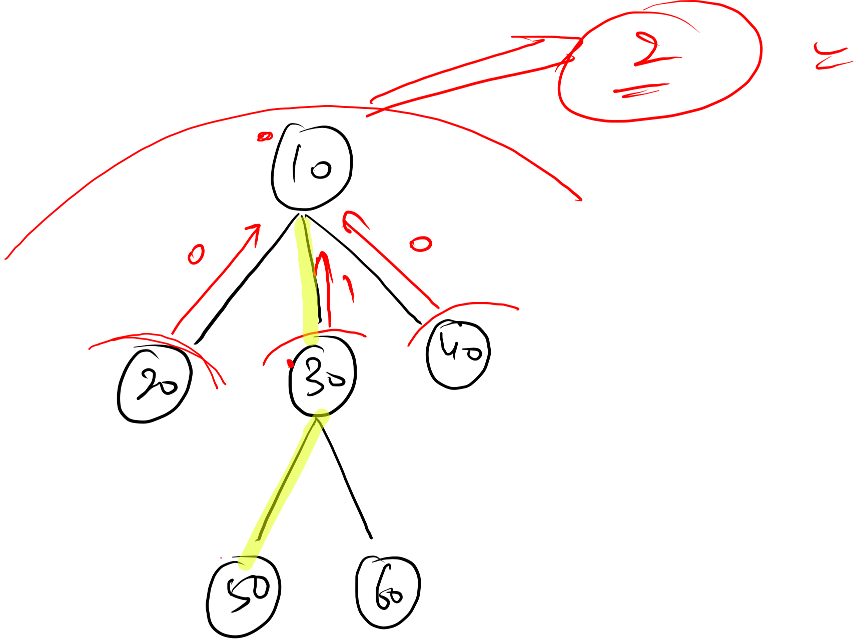


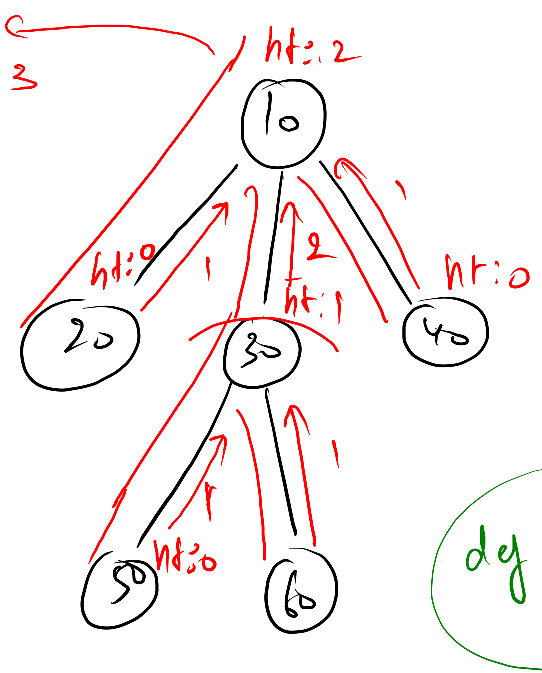
```
public static int max(Node node) {
    int res = Integer.MIN_VALUE;

    for(Node child : node.children){
        res = Math.max(res , max(child));
    }

    return Math.max(res,node.data);
}
```

Height ⇒ Distance b/w root node deepest node.

Distance

- ☆ Edges → 2
- Nodes → 3

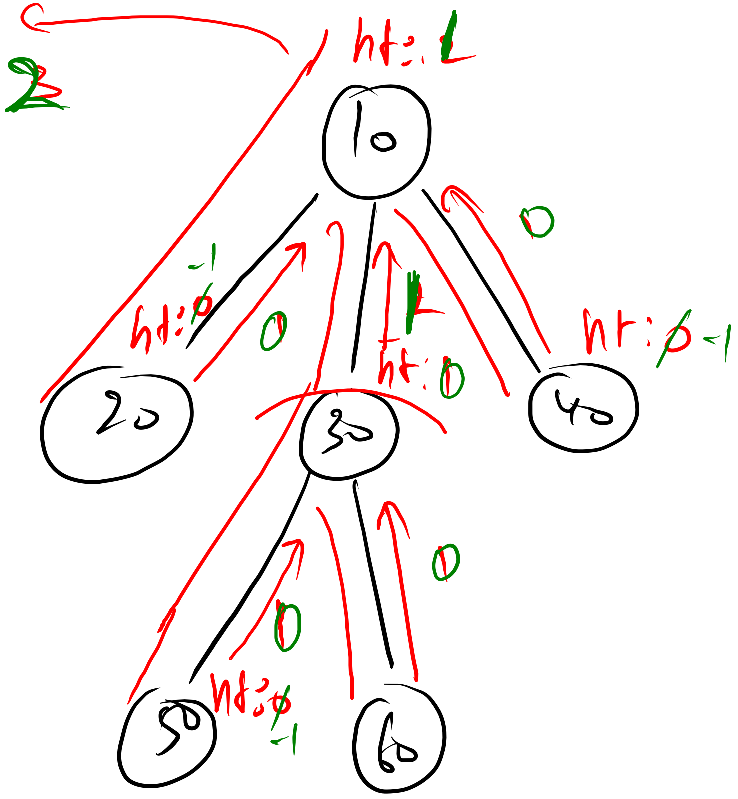


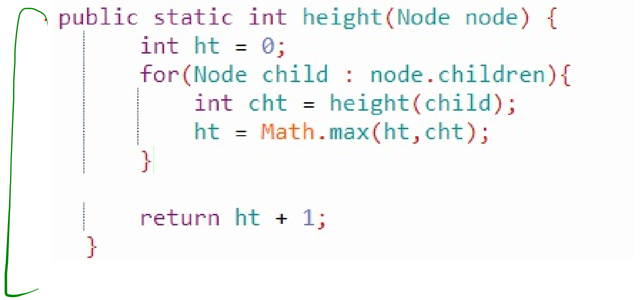


Ht on the basis of nodes

def. ht : 0

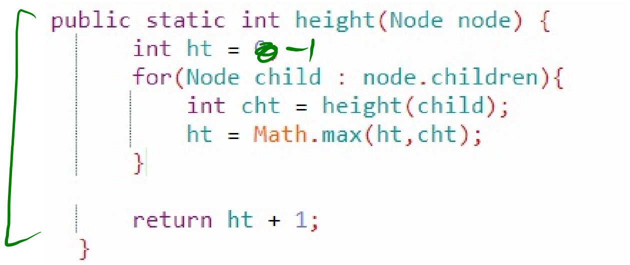
def. Ht = -1





```
public static int height(Node node) {
        int ht = 0;
        for(Node child : node.children){
            int cht = height(child);
            ht = Math.max(ht,cht);
        }

        return ht + 1;
    }
```

Ht on the basis of Edges



```
public static int height(Node node) {
        int ht = ~~0~~ -1;
        for(Node child : node.children){
            int cht = height(child);
            ht = Math.max(ht,cht);
        }

        return ht + 1;
    }
```