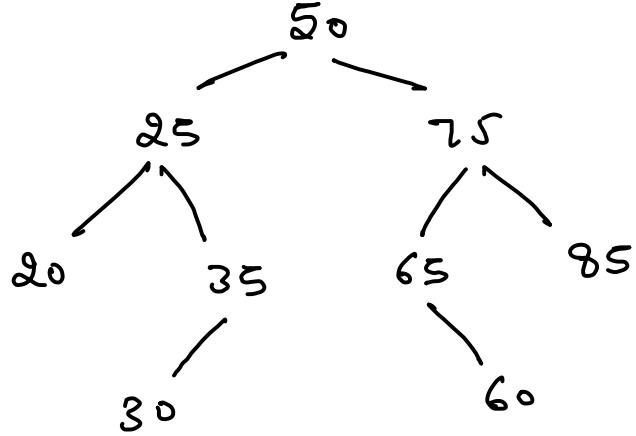
Binary Tree.

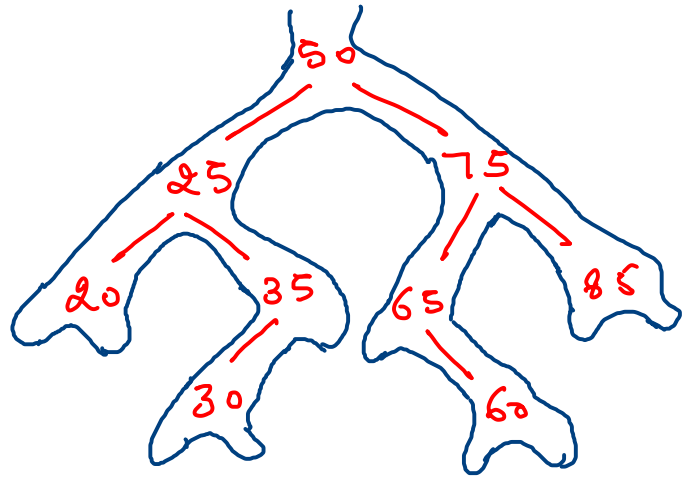
Degree : 2. , Max. no. of child in a Root is (2).



How to Hold the data - →

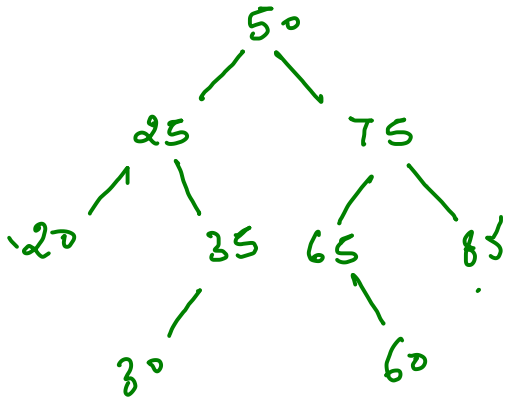
Node → Data of type. int
→ left child → Node data type
→ Right child → Node data type;

How to construct B Tree



PreOrder traversal :

50, 25, 20, null, null, 35, 30, null, null, null
75, 65, null, 60, null, null, 85, null, null



Stack - DS.

Node + State.

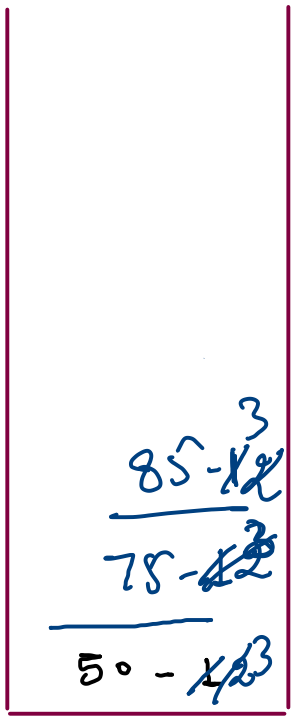
State → 1 → left child
2 → Right child
3 → pop()

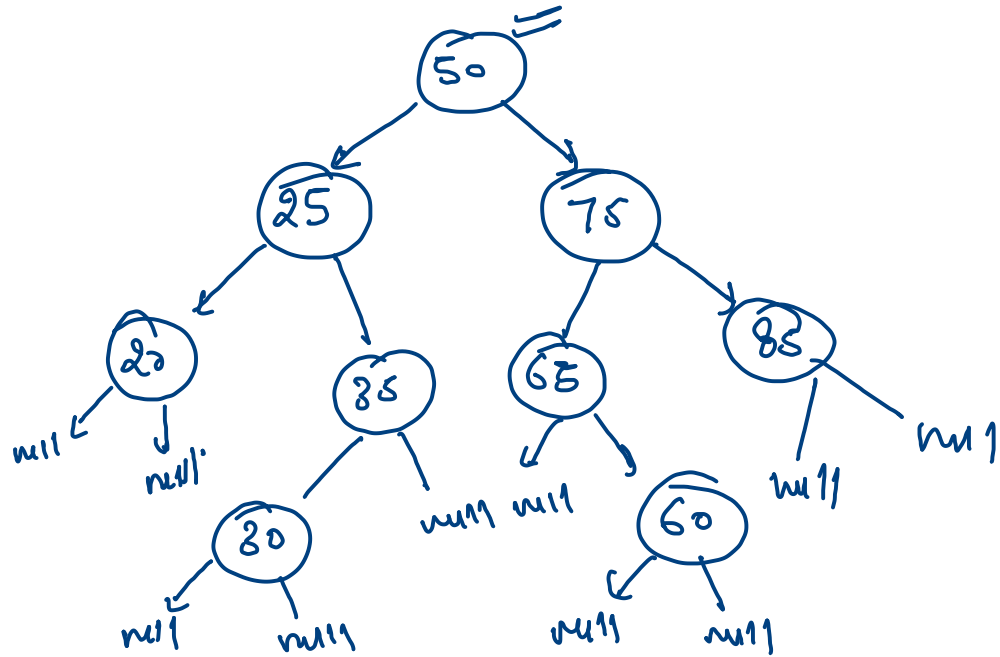
Iterative traversal in Generic Tree.

1 — left child.

2 — Right child.

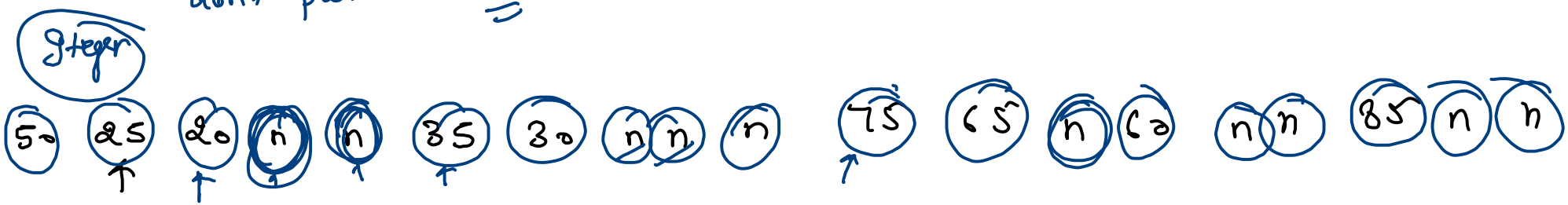
3 → pop from stack

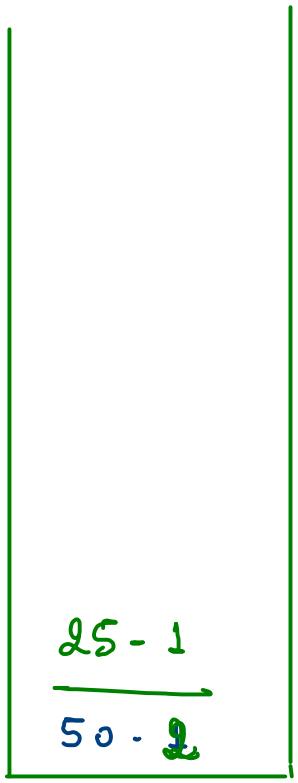




arr[i] == null
don't push in stack.

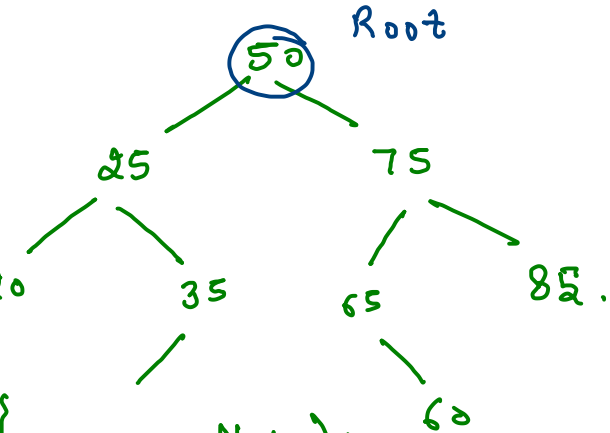
Integer i = new Integer( );





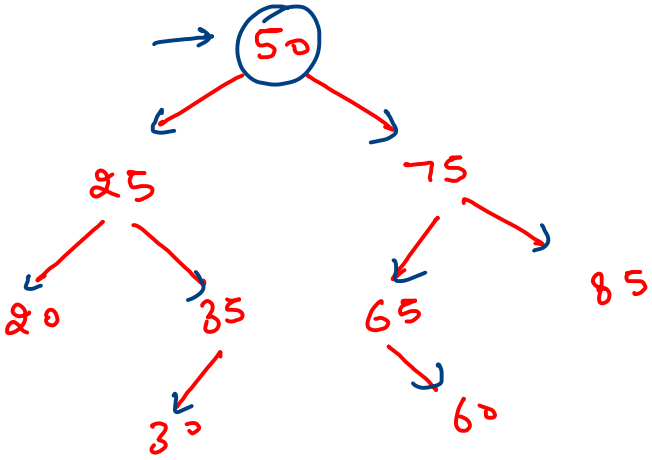
Pair class → Node + State.

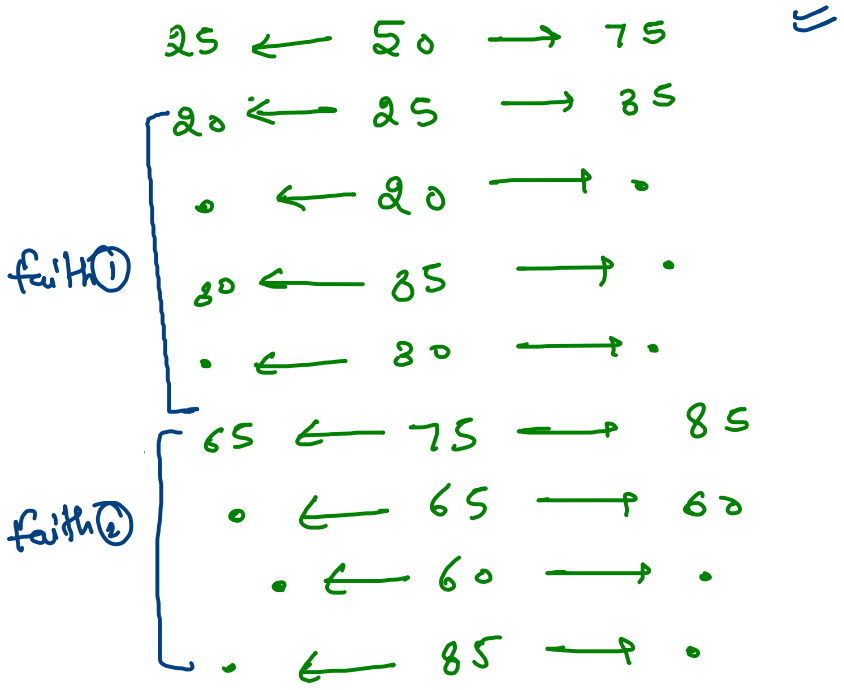
```
indx = 1;

while (st.size() > 0) {
    Pair top = st.peek();
    if (top.state == 1) {
        if (arr[indx] != null) {
            Node nn = new Node(arr[indx]);
            top.node.left = nn;
        } else { top.node.left = null; }
        top.state ++;
    } else if (top.state == 2) {
        left → Right
    } else {
        st.pop();
    }
}
```



indx ① →

PreOrder → 50 25 20 n n 35 30 n n n 75 65 n 60 n n 85 n n.





Expectation → display(50)

faith → display(25) → ①
display(75) → ②

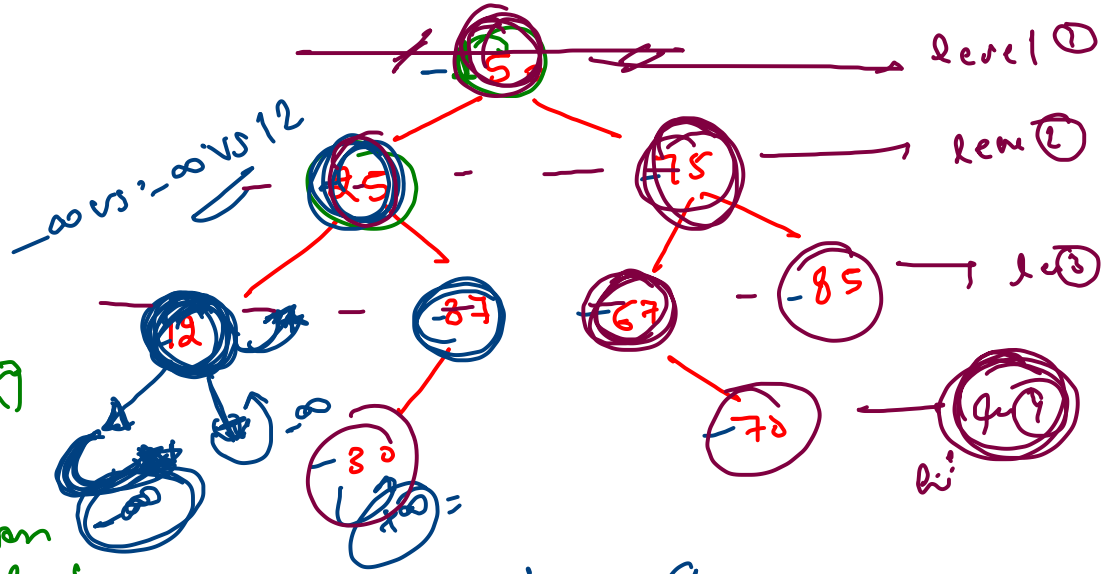
Merging →

print → your left child. node data node.RightChild.

→ call (node.left);

→ call (node.right);

Max from Binary Tree



Expectation — max(50) → overall max

faith:

max(25) → max from subtree having root is 25

37

max(75) → max from subtree having root is 75

85

a * e = a

identity = e

return — lmax vs. Rmax vs. Root.data.

max → operator.

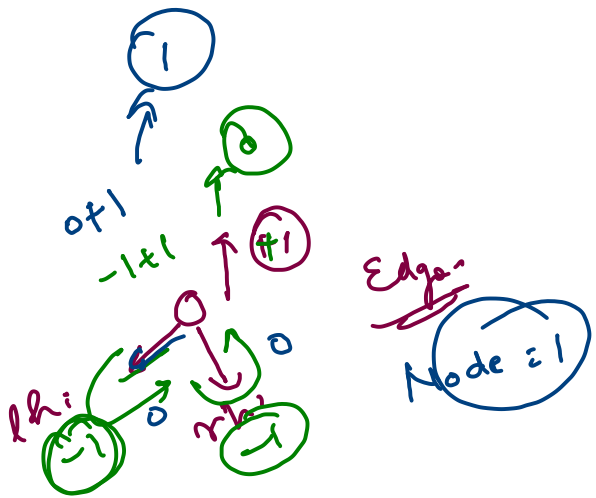
identity = MIN {-∞}

depth + 1 = level

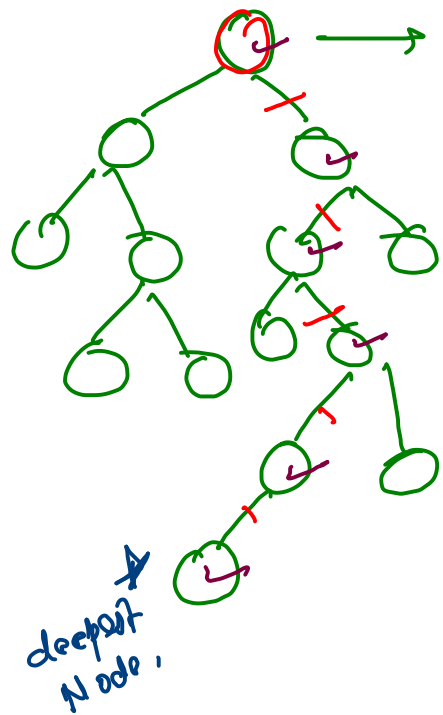
height + 1

depth

Height - distance of deepest Node from the Root.



Edge - Node = 1



Height = on the basis of Edge.

on the basis of Node.

from null Node,

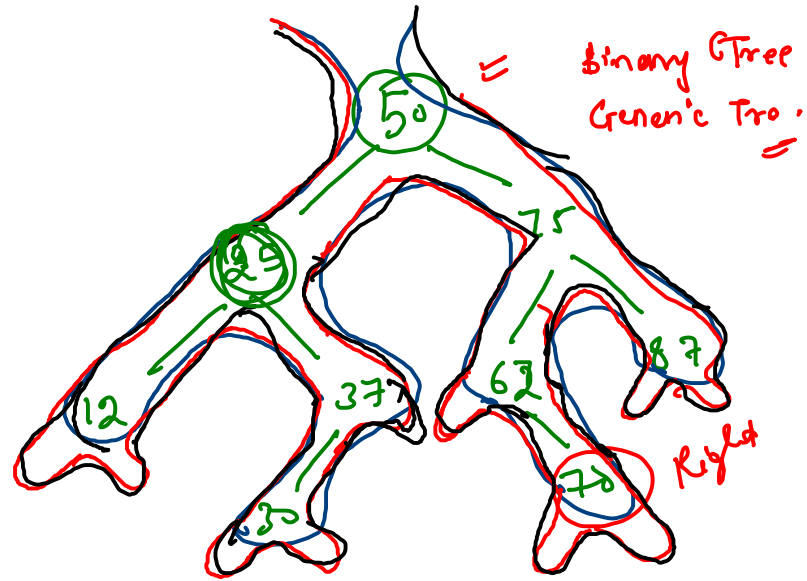
Edge → 5 → -1

Node → 6 → 0

Traversals →

1. PreOrder
2. InOrder
3. Post Order

left · call
In Area
right · call

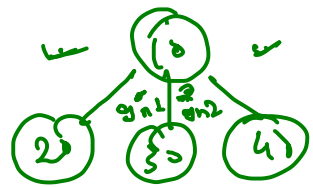


= Binary Tree
Generic Tree

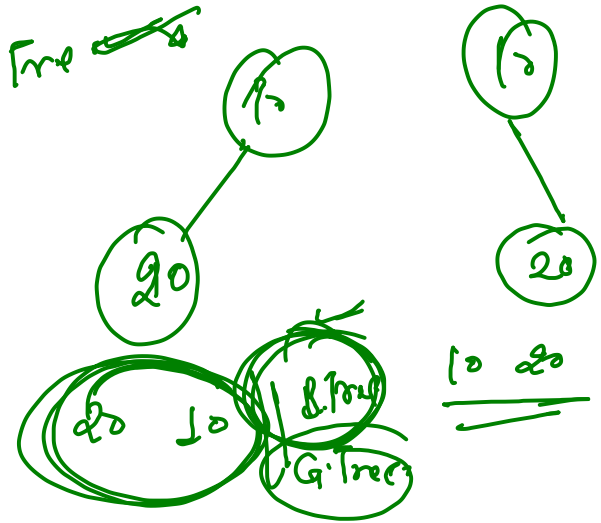
Pre Order → 50 25 12 37 30 75 62 70 87

InOrder → 12 25 30 37 50 62 70 75 87

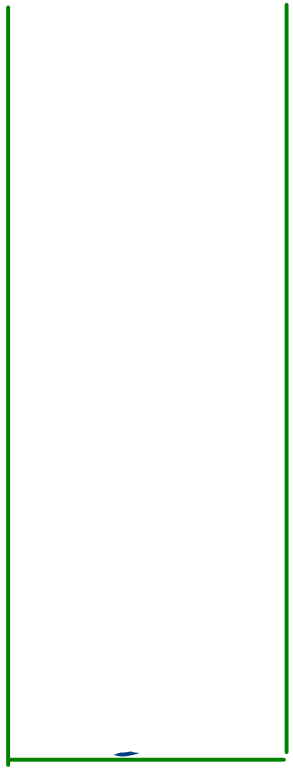
Post order → 12 30 37 25 70 62 87 75 50



BTree

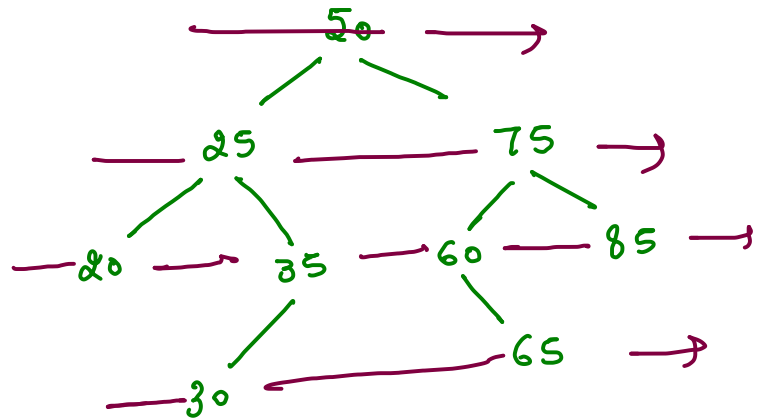


10 20



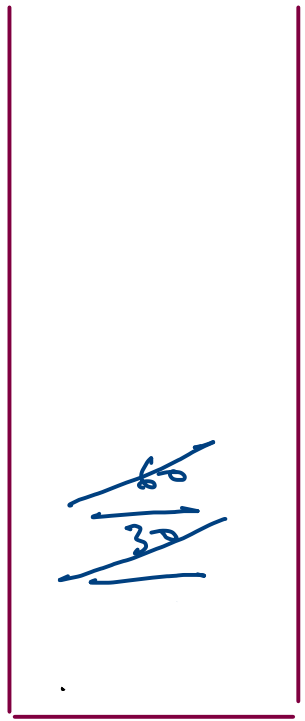
Get
Remove
Work
Add. children.

```
while(q.size()>0) {
    Get
    Remove  } pop.
    work - print
    add children
}
```



50 25 75 20 35 60 85

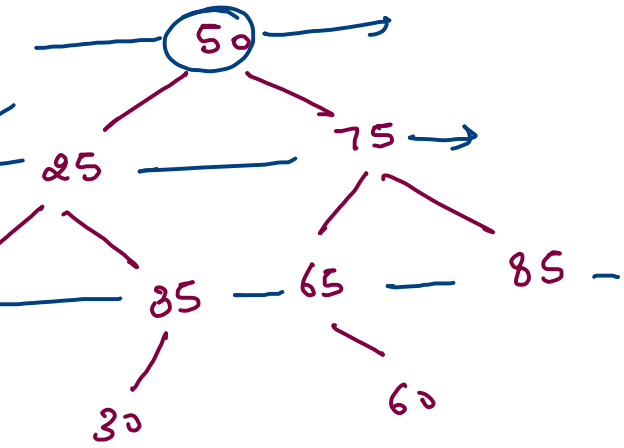
30 65



Linked

Add(x) → O(1) push

Remove first → O(1) pop




```
while(q.size()>0) {
    int sz = q.size();
    while( sz >0) {
        size--;
        Get.
        Remove
        work
        Add Children
    }
} //if Ended
//System.out.println();
}
```

→ 1. 50,

→ 2. 25 75

→ 3. 20 35 65 85

→ 4. 30 60

Iterative pre, in, post Order

Node + State.

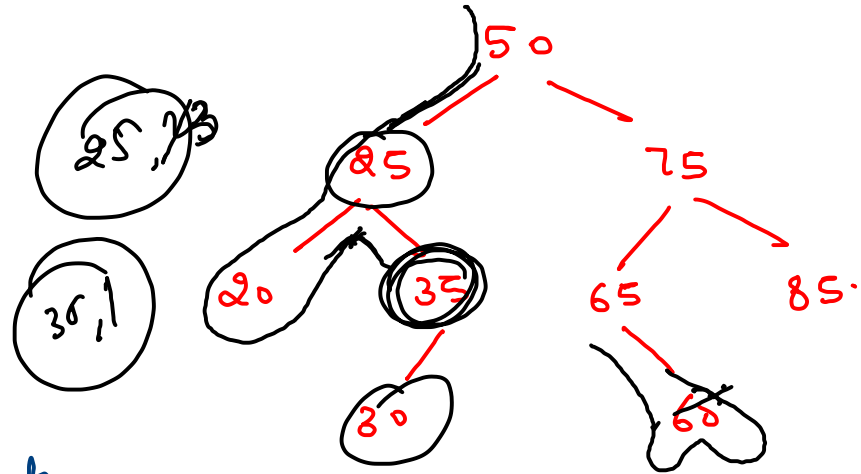
State : 1 → pre Area. → pre-add, state++, left child in stack

State : 2 → In Area. → In Add

State : 3 → post Area. → state++, right child. Add stack.

↳ post add.

& pop():

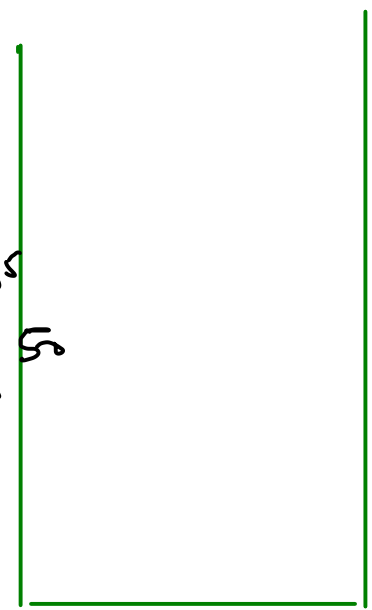


state

1 pre = 50 25 20 35 30 75 65 60 85

3 post = 20 30 35 25 60 65 85 75 50 →

2 In = 20 25 30 35 50 65 60 75 85 →



```
while(st.size()>0){
① get top.
if(top.state==1){
}
else if(top.state==2){
}
else{
}
}
```

20 ← 10 → 30

40 ← 20 → 50

80 ← 40 →