12-JAN-22

Date :
Page No.

```
public static void preorder(Node node){
    sout(node.data + " ");
    for(Node child : node.children){
        preorder(child);
    }
}
.
.
.
preorder(root);
sout(":");
```

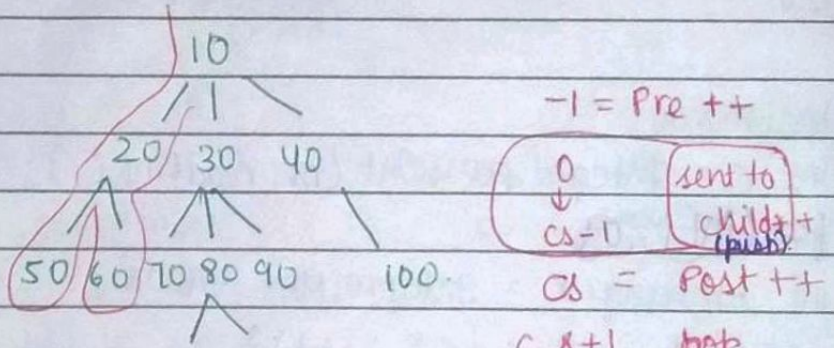
Input

24
-10 20 50 -1 60 -1 -1 30 70 -1 80 110 -1 120 -1 -1 90 -1 -1 40 100
-1 -1 -1

output

10 20 50 60 30 70 80 110 120 90 40 100. [Left side].

Iterative pre & Post GIT



-1 = Pre ++

0 ↓ cs-1 — sent to child++ (push)

cs = Post ++

c.s+1. pop

60 ~~x~~ ~~ø~~ ~~1~~ (Pre)
50 ~~x~~ ~~ø~~ 1 (Pre)
20 ~~x~~ ~~ø~~ ~~1~~ 3
10 x ø 1

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

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

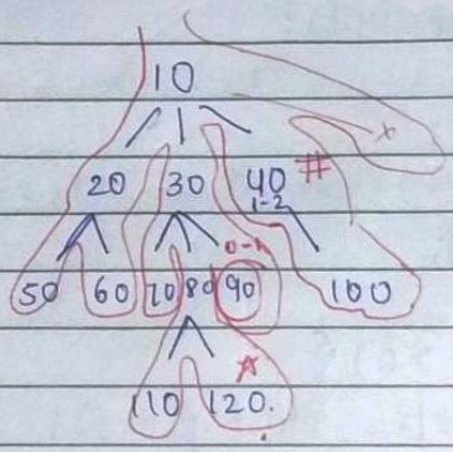
90 -x^0	
120 -x^1	
110 -x^1	
80 -x^2	
70 -x^1	100 -x^0
30 -x^1/3	40 -x^0
10 2	10 3/4

1. Iterative Pre & Post of GT

```
String preorder = " ";
String postorder = " ";
Stack <Pair> stack = new stack<>();
                  ↳ Node store, or node ka state (Euler Tree).

Pair rootp = new Pair();
rootp.state = -1;
rootp.node = root;

stack.push(rootp);

while(stack.size() > 0) {
    Pair peekp = stack.peek();

    if(peekp.state == -1) {
//pre     preorder += peekp.node.data + ".";
        peekp.state++;
    } else if(peekp.state >= 0 && peekp.state < peekp.
                node.children.size()) {
//child   Pair childp = new Pair();
        childp.state = -1;
        childp.node = peekp.node.children.get(peekp.state);
        stack.push(childp);

        peek.state++;
```

```
} else if (peep.state == peekp.node.children.size()) {
//post      postorder += peekp.node.data + ", ";
            peekp.state++;
        } else {
//pop       stack.pop();
        }
    }
    south(preorder + ".");
    south(postorder + ".");
```

2. Predecessor And Successor of An Element.



state = ~~0~~ 1
pre = ~~null~~ ~~10~~ ~~20~~ ~~50~~ ~~60~~ ~~30~~ ~~70~~ ~~80~~ ~~110~~ 120
succ = ~~null~~ 40

```
if (state == 0) {
    if (node.data == data) {
        state++;
    } else {
        predecessor = node;
    }
} else if (state == 1) {
    successor = node;
    state++;
}
```

```
static Node predecessor;
static Node successor;
static int state = 0;
public static void predecessorAndSuccessor(Node node,
                                           int data) {
    if (state == 0) {
        if (node.data == data) {
            state++;
        } else {
            predecessor = node;
        }
    } else if (state == 1) {
        successor = node;
        state++;
    }
    for (Node childrenNode : node.children) {
        predecessorAndSuccessor(childNode, data);
    }
}
```

Date :
Page No.

3. Node with Maximum Subtree Sum

Vo Node btana h jiske subtree ka sum maximum h. (Vo sum or Node dono btane h).

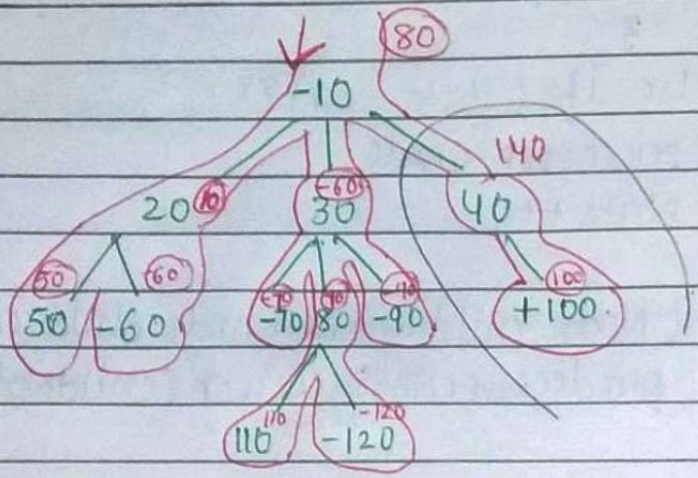
(max sum node).

msn : ~~X 50 110~~ 40 → (40 vo node jo store bda sum contain krta h)

ms : ~~-∞ 50 110~~ 140

(max sum).

→ Hr Node ki responsibility h msn or ms me change kre. (travel & Tree) or apna sum return krde




```
static Node msn = null;
static int msum = Integer.MIN_VALUE;

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

        for(Node child : node.children){
            int csum = question(child);
            sum += csum;
        }
```

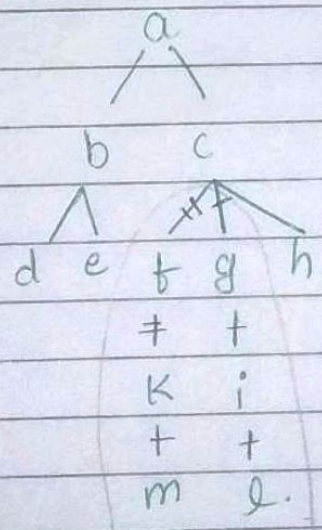
```
        if (sum > msum) {
            msum = sum;
            msn = node;
        }
        return sum;
    }

    public static void main (String[] args) throws Exception{
        .
        .
        .
    Node root = construct (arr);

    question (root);
    System.out.println (msn.data + "@" + msum);
```

4 DIAMETER OF Generic Tree -

Diameter - MAXIMUM NO. OF EDGES B/W ANY 2 NODES.

↓OR

MAX DISTANCE B/W 2 NODES (Edges) is the diameter.



6 edges B/W m & l.
(Diameter is 6. of ml)