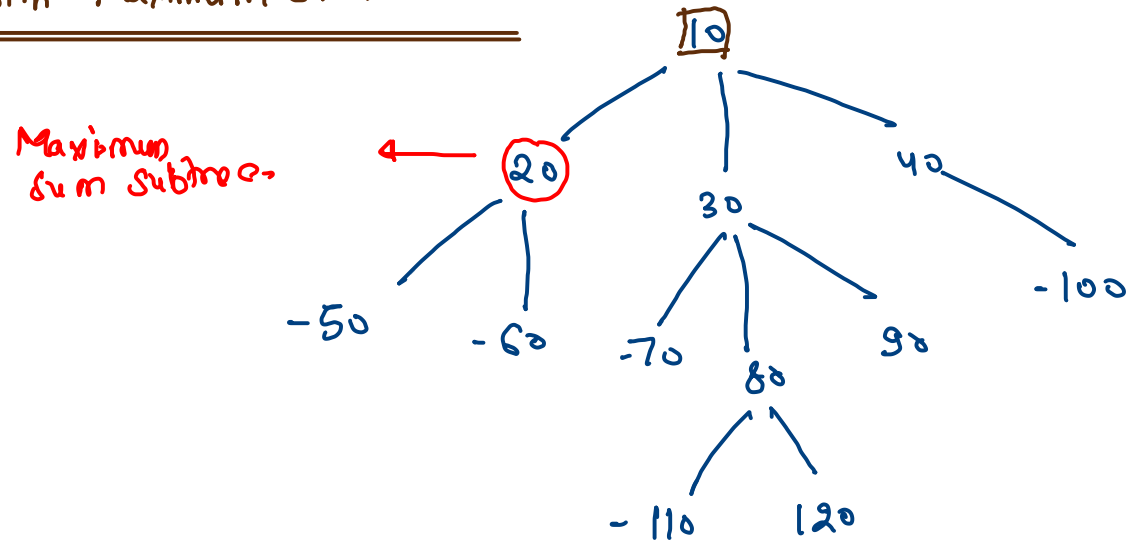
Node with Maximum Sum Subtree

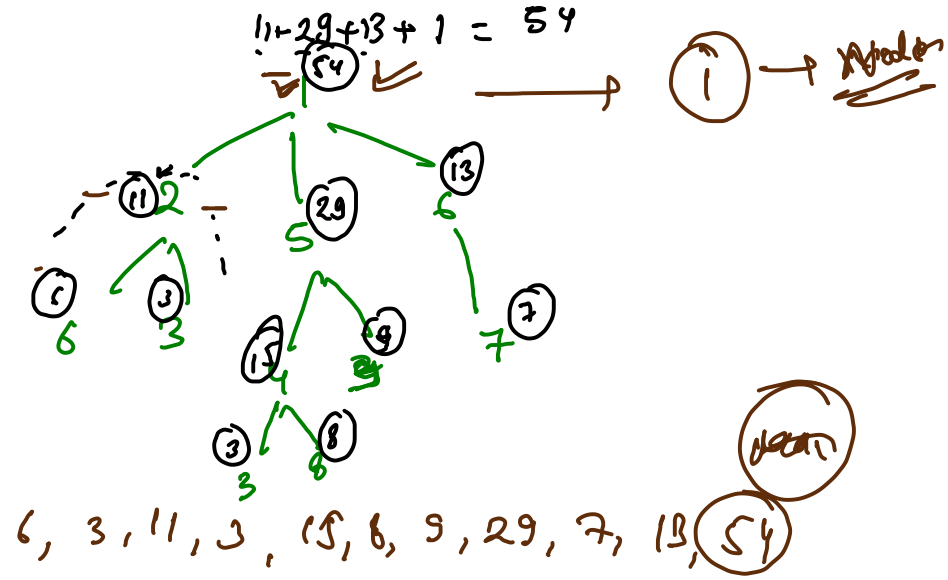


Maximum sum subtree

condition →

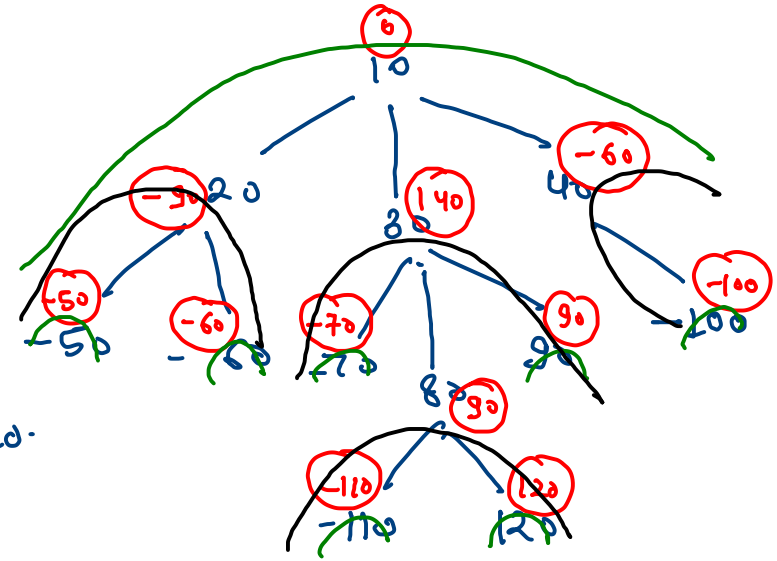
if (node.data >= 0)

INode = root



Maximum sum subt

Sum of Generic Tree.



Recursion.

Expectation → sum(10) → overall sum of all subtree having root is 10.

faith → sum(node.children.get(i)) →

Ex: 10 as root.

sum(20) → 20

sum(30) → 140

sum(40) → -60

Max. sum of subtree. → 140 || Node → 30

30 @ 140

Merging → faith & Expectation

sum →

+ children. → find sum till you

sum variable.

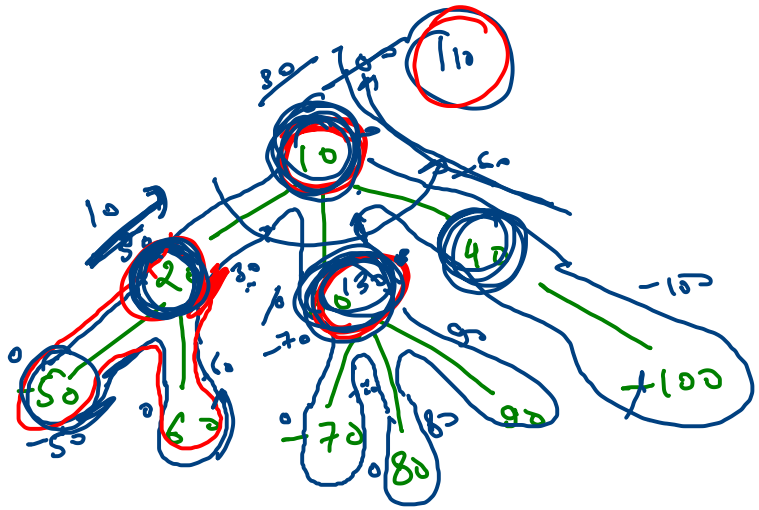
↳ solve sumvariable + node.data;

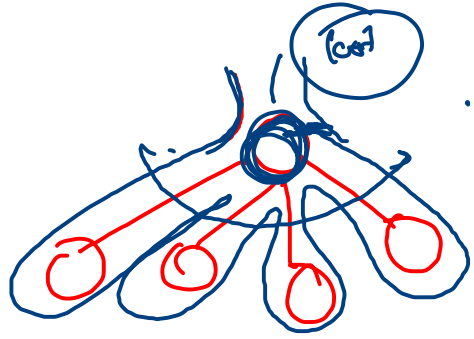
```java
public static int sumOfTree(Node root) {

    int sum = 0; // sum from child
    for(Node child : root.children) {
        sum += sumOfTree(child);
    }
    int osum = sum + root.data; // overall sum

    return osum;
}
```

Post Area → left

MaxSum = 0

MaxSum = Math.max (maxSum vs. osum).



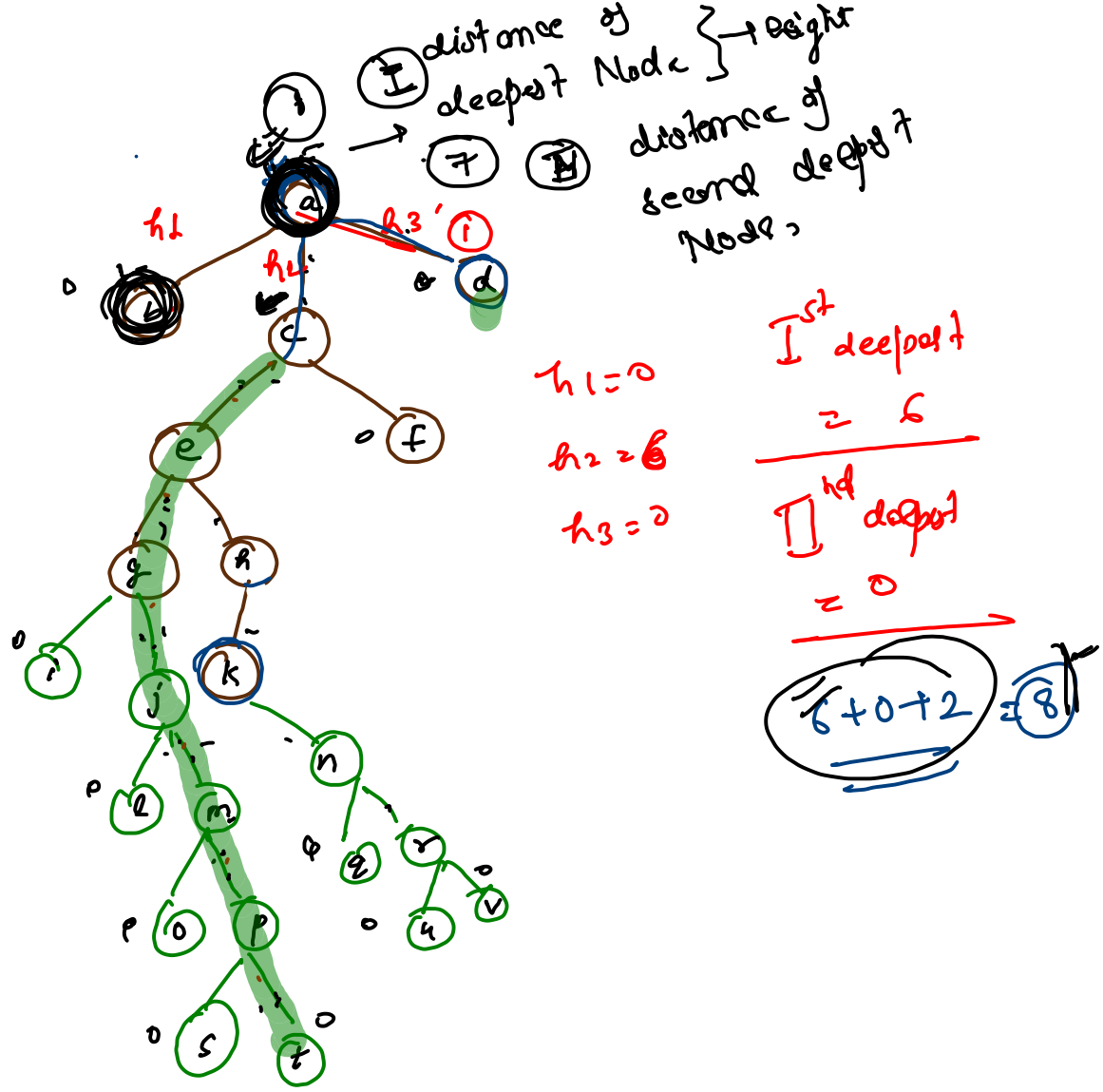


Diameter of Generic Tree

Diameter is Max. distance b/w any two nodes.

on the basis of →
- Edge
- Node

diameter = 5



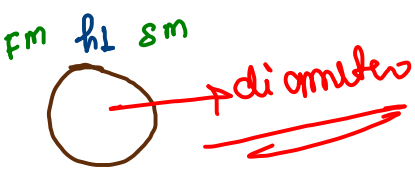
t → u, v
s → u, v
} max dist

- First deepest] - max height
- Second deepest] - 2nd max height

firstmax = -1

secondmax = -1

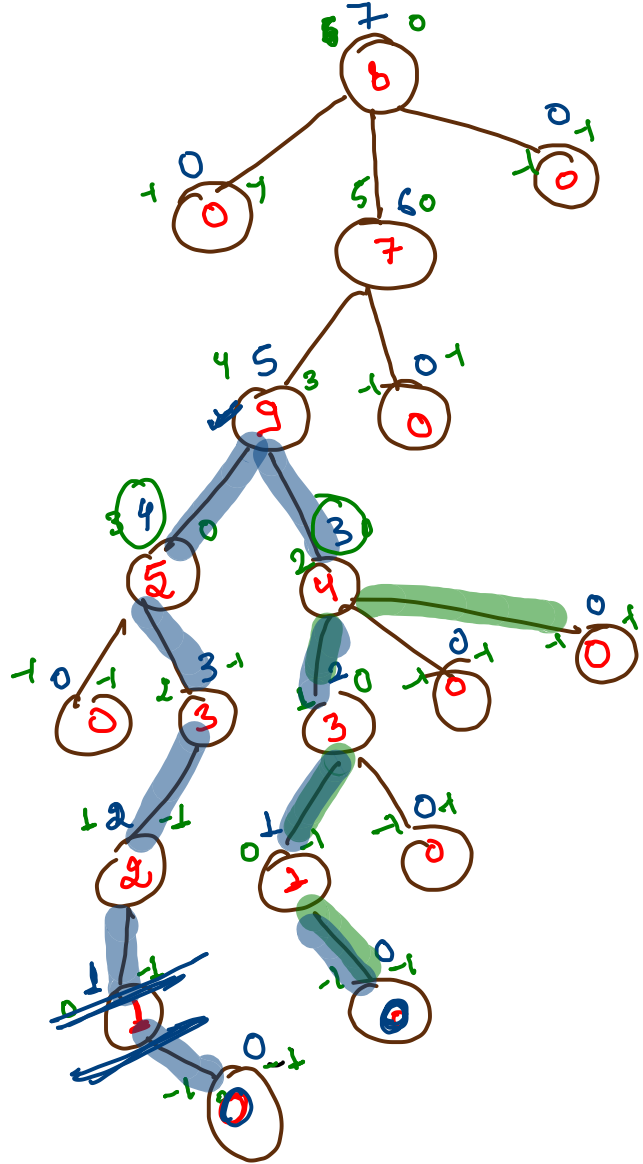
return → FM + SM + 2.



h → height

FM → first Max

SM → Second Max



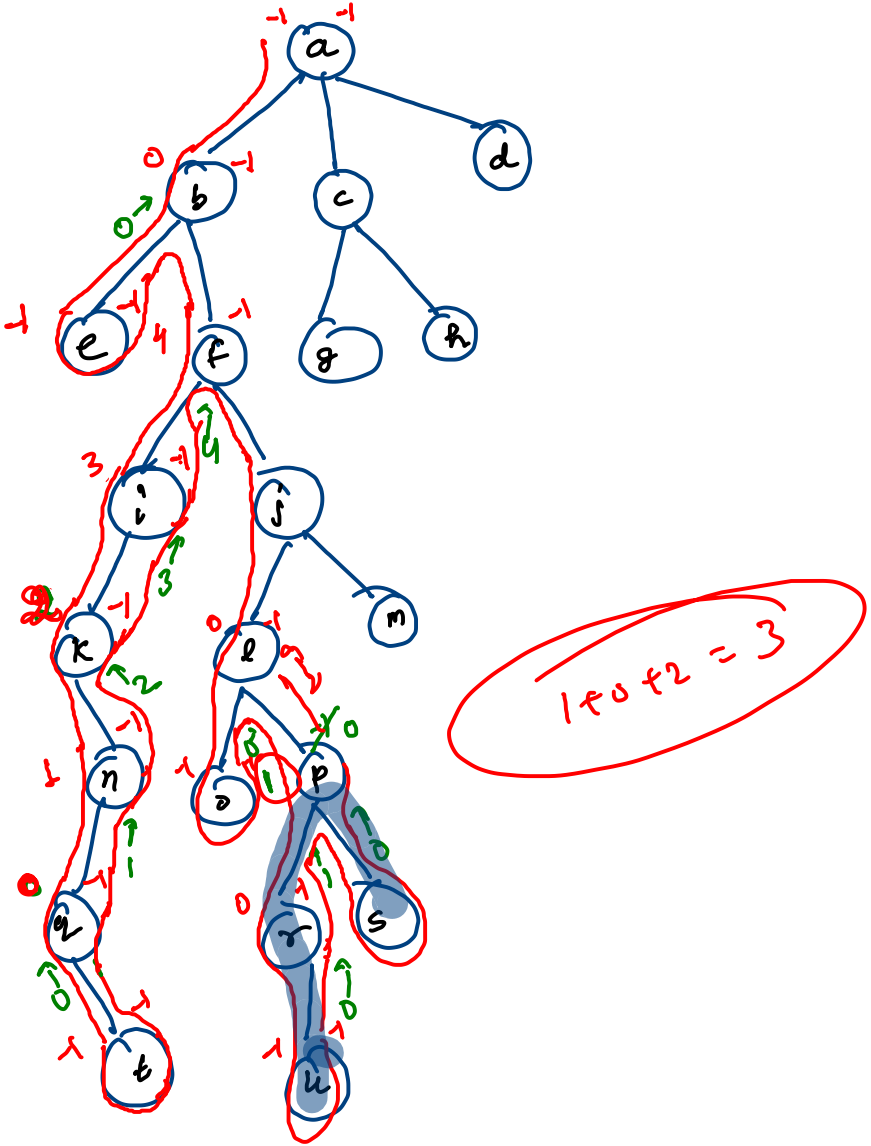
Diameter-

Max (First + Second) + 2

9

```
public static int diameter = 0;

public static int heightForDia(Node root) {
    int firstMax = -1; // actual height
    int secondMax = -1;

    for(Node child : root.children) {
        int rht = heightForDia(child); // rec height
        if(rht >= firstMax) {
            secondMax = firstMax;
            firstMax = rht;
        } else if(rht > secondMax) {
            secondMax = rht;
        }
    }

    diameter = Math.max(diameter, firstMax + secondMax + 2);

    return firstMax + 1;
}
```

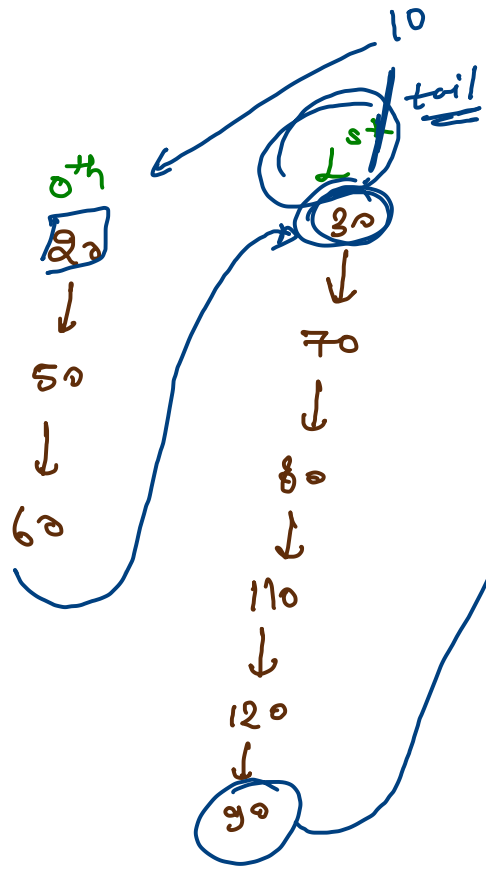
diameter = 0 ~~1~~ ~~2~~ 3 4



linearize → preorder

Expectation → linearize(10)

10 → 20 → 50 → 60 → 80 → 70 → 80 → 110 → 120 → 90 → 40 → 100



Recursive

space → Allowed.

time complexity.

$O(n^2)$ ✓

get i → Remove from Root

for (i-1)

get tail. → 90

90.children.add(40)

faith.

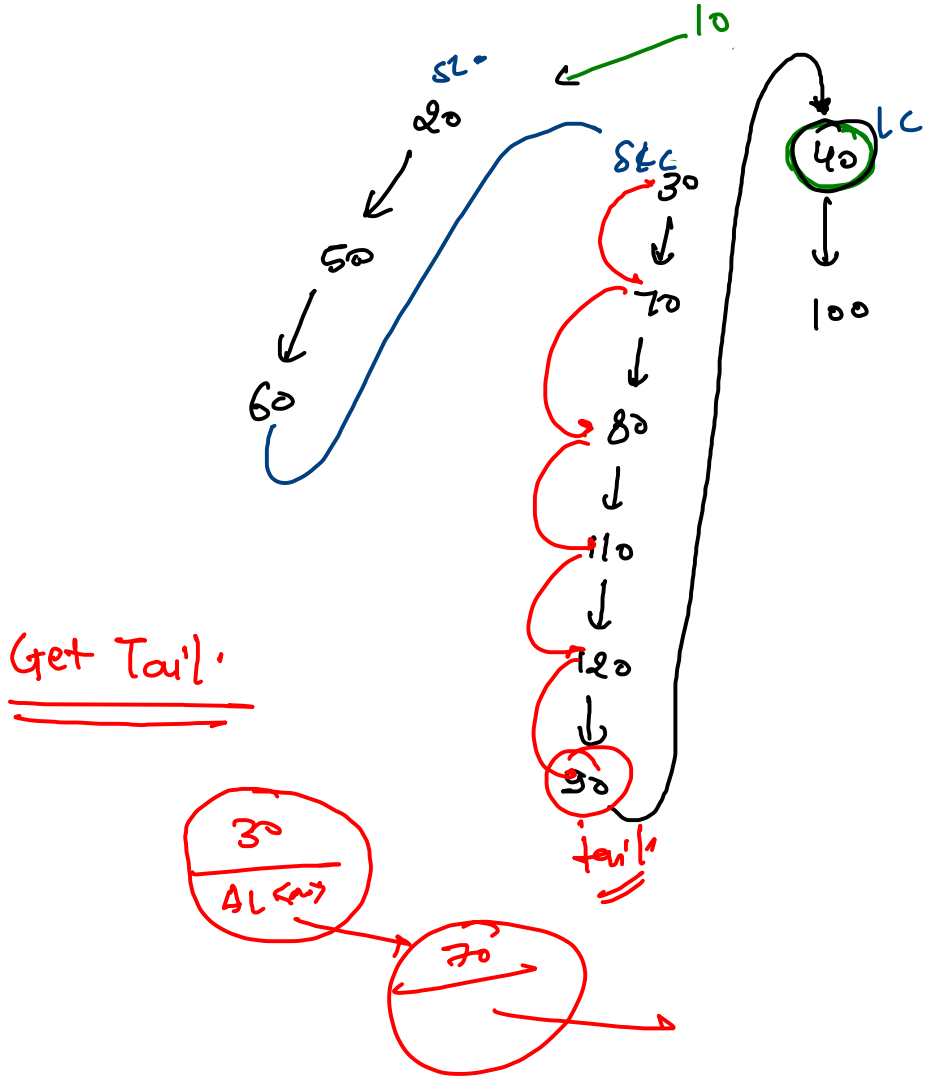
linearize(20)

linearize(30)

linearize(40)

↓

individually linearize.



Steps -

faith. (1) linearize child from Recursion.

Perform these steps until size of root.children > 1 -

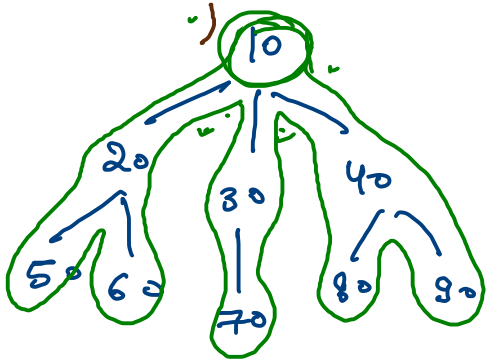
(2) Get last child.

(3) Get second Last child.

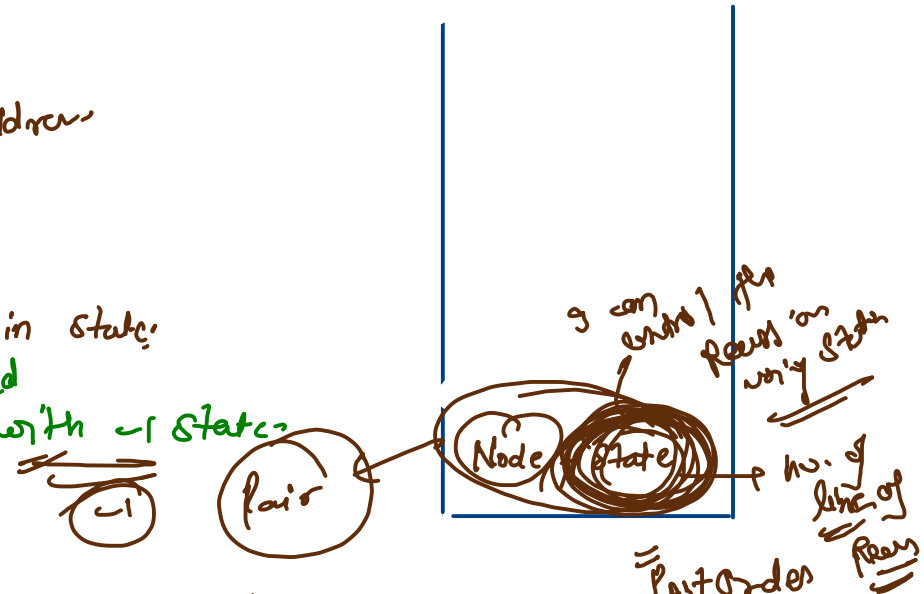
(4) Remove last child from root.children.

(5) Get tail of SLC & add LC as children in SLC

PreOrder and PostOrder from Iterative




```
while (st.size() > 0) {
    if state == -1
        preOrder.add(node);
        state++;
    else if state == no. of children
        postOrder.add(node);
        pop.
    otherwise, increment in state.
        get state under child
        add in stack with -1 state.
```



Node → State.

State → -1 → Node Pre.
0 → edge pre of 0th child
1 → Edge of 1st child loop.
2 → Edge pre of 2nd child loop.
(no. of children) 3 → Node Post