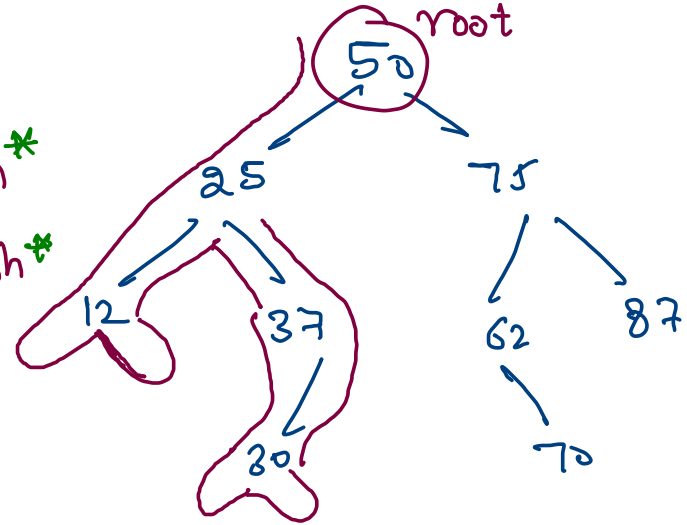
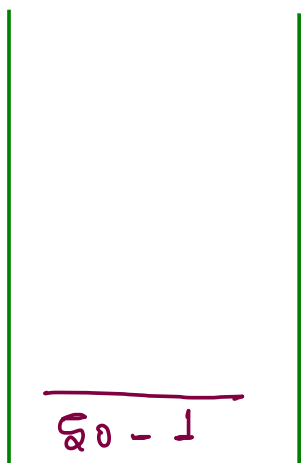
Iterative Pre, Post and Inorder :

State = 0 → preorder + left child push*

State = 1 → Inorder + right child push*

State = 2 → Post order + wipeout





Stack - Node-state

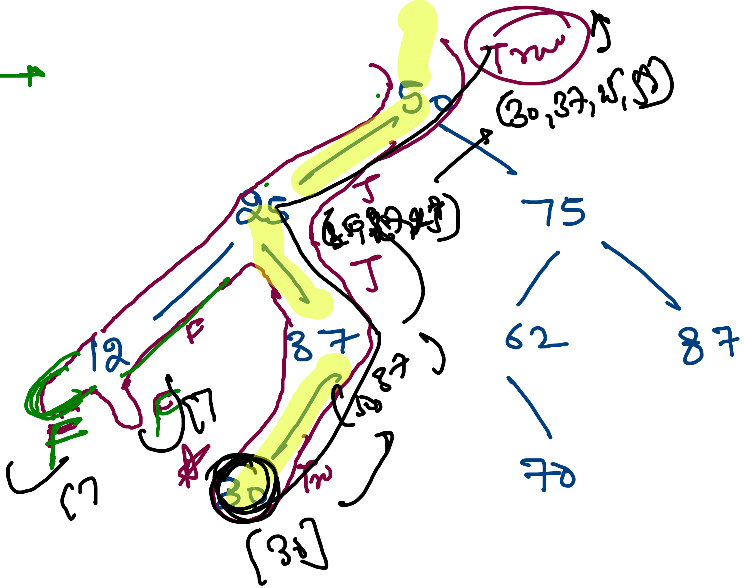
preOrder → 50 25 12 37 30 ---

Inorder → 12 25 30 37 ----

Post order → 12 30 37 25 ---

Find and node to Rootpath: →

Find. 3:
Self check
left Check → True
right check return
return false True.



Expectation

find(root) = True / false

faith →

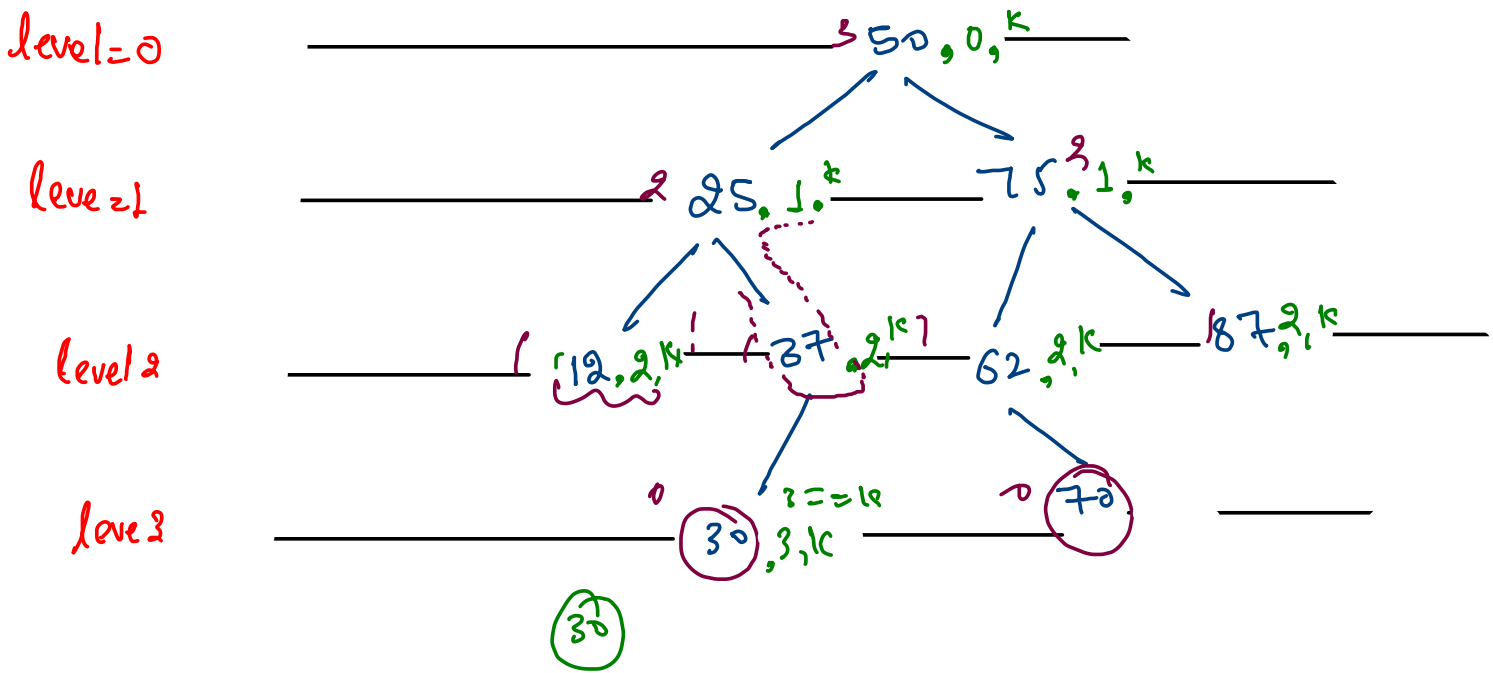
find(root.left) → T / F

find(root.right) → T / F

merging.

Node to root path:

k-level down k=3 → 30, 70 node, level, k



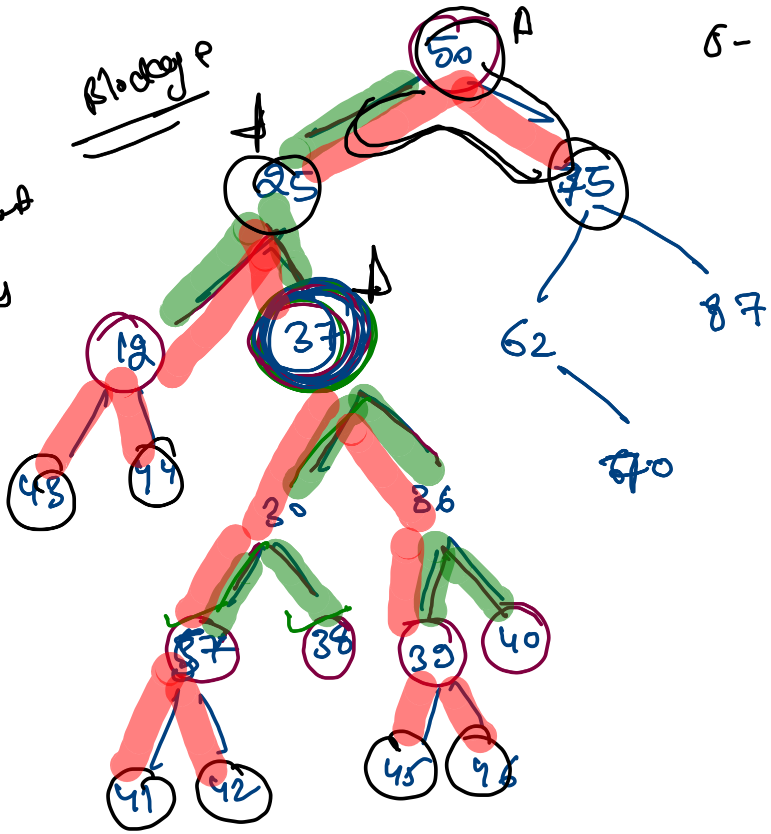
k=2 12 37

k-far (k-distance away)

data = 37

k = 2

we to prevent from previous path



Recursion

Node to Root path
+
K-down

k = 2

~~37~~ 38 39 40 12 50

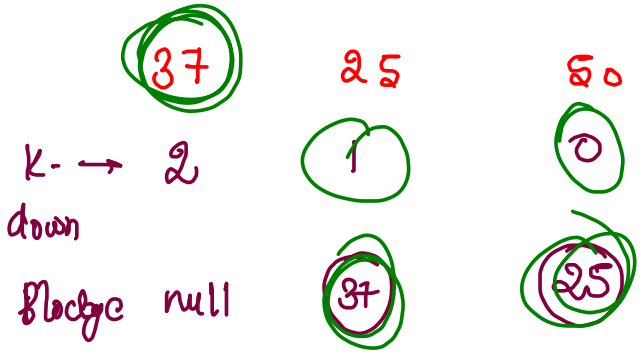
k = 2

41 42 45 46 43 44 75

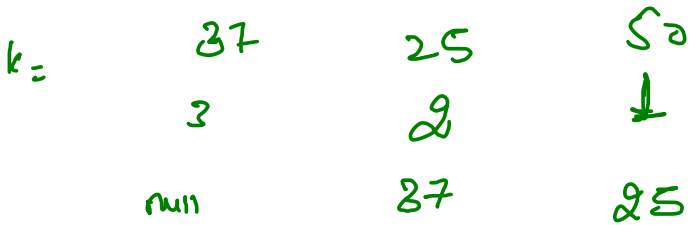
data = 37 k-far

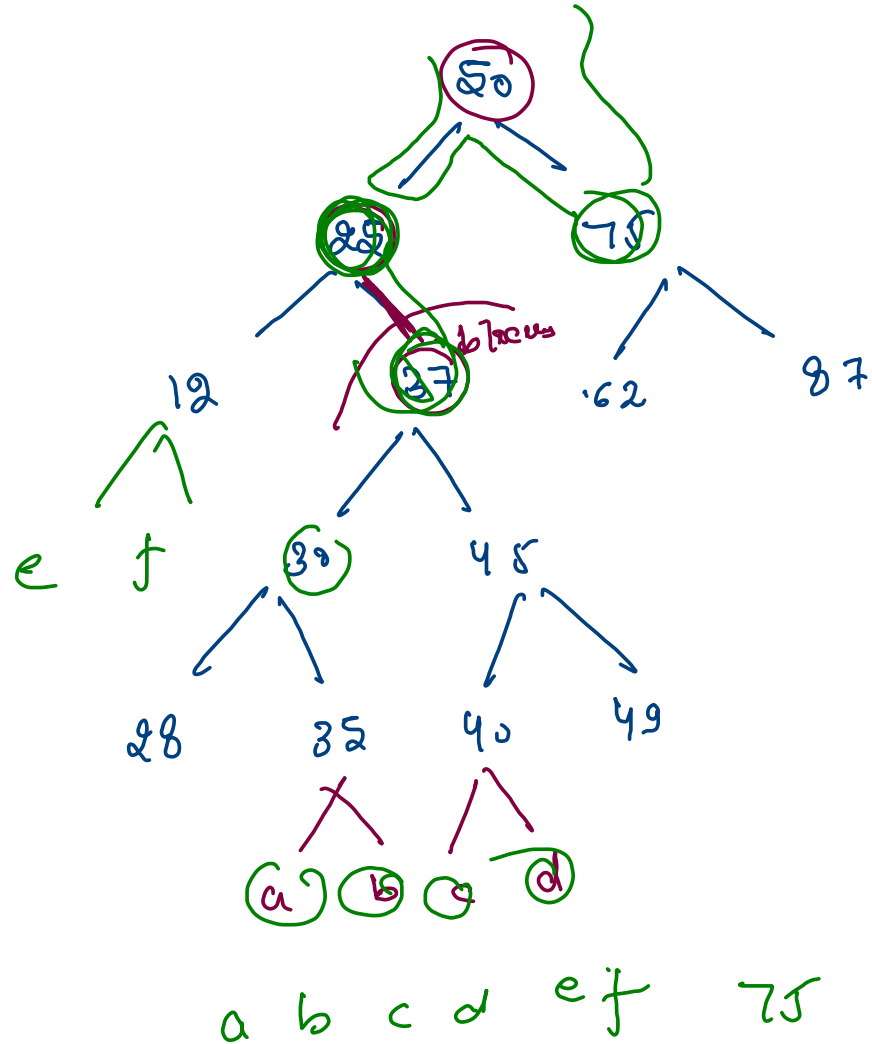
k = 2

Node to rootpath (node)

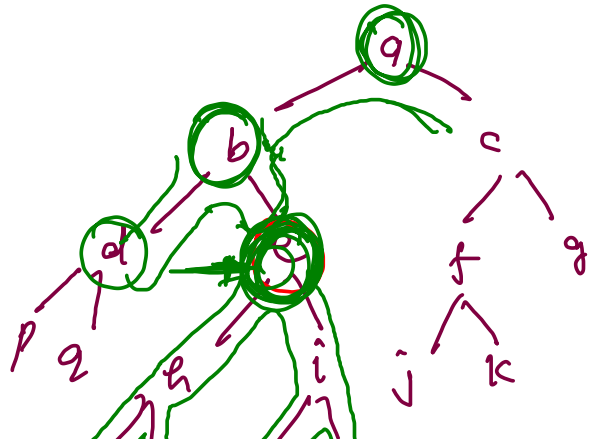


28 35 40 49 12 50





a b c d e f 75



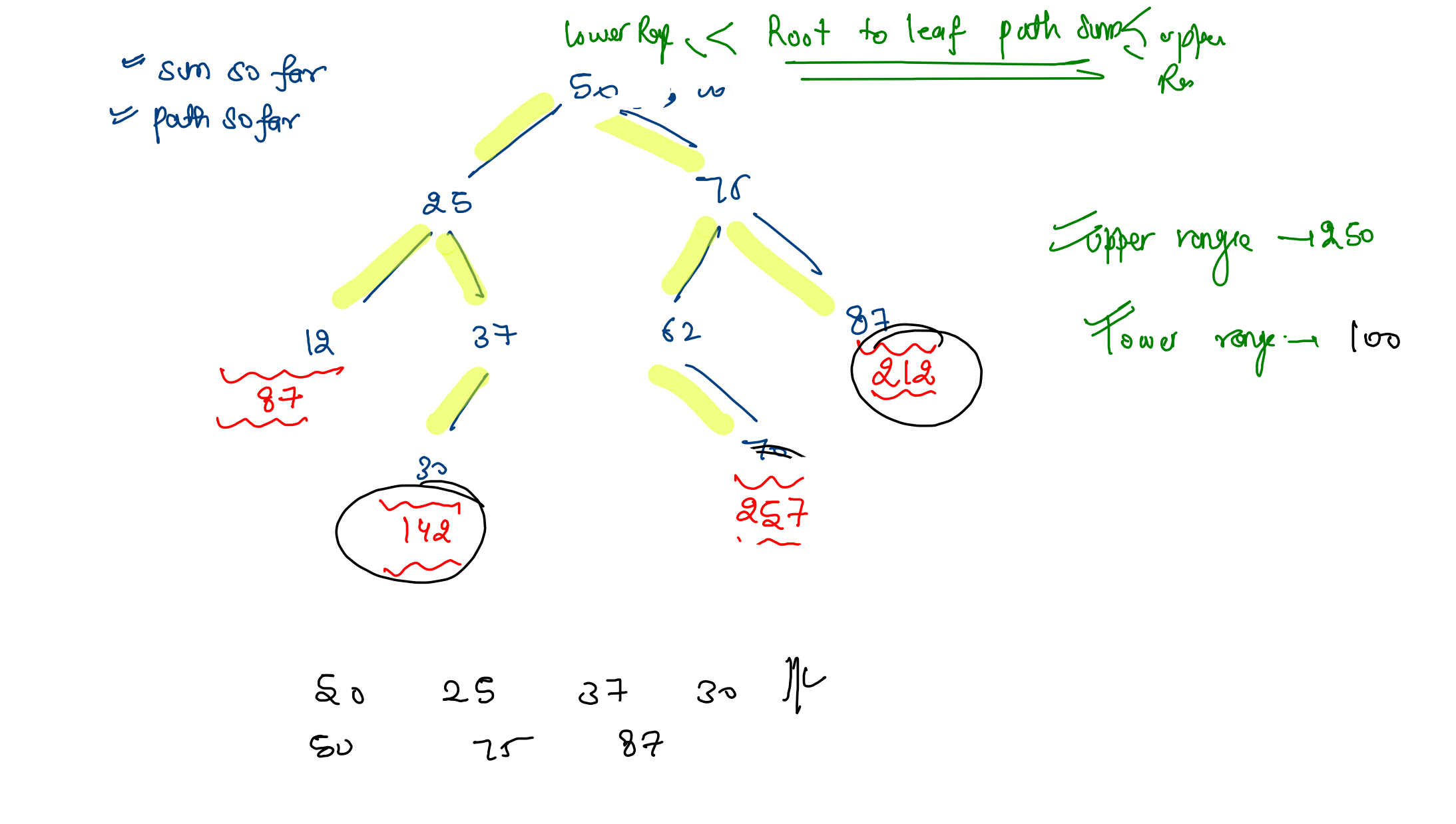

```
public static void printKDown(Node node, Node blockage, int k) {
    if(node == null || node == blockage) return;

    if(k == 0) {
        System.out.println(node.data);
        return;
    }

    printKDown(node.left, blockage, k - 1);
    printKDown(node.right, blockage, k - 1);
}

public static void printKNodesFar(Node root, int data, int k) {
    ArrayList<Node> n2rp = nodeToRoot(root, data);

    Node blockage = null;
    for(int i = 0; i < n2rp.size() && k >= 0; i++) {
        Node node = n2rp.get(i);
        printKDown(node, blockage, k);
        k--;
        blockage = node;
    }
}
```

data = e

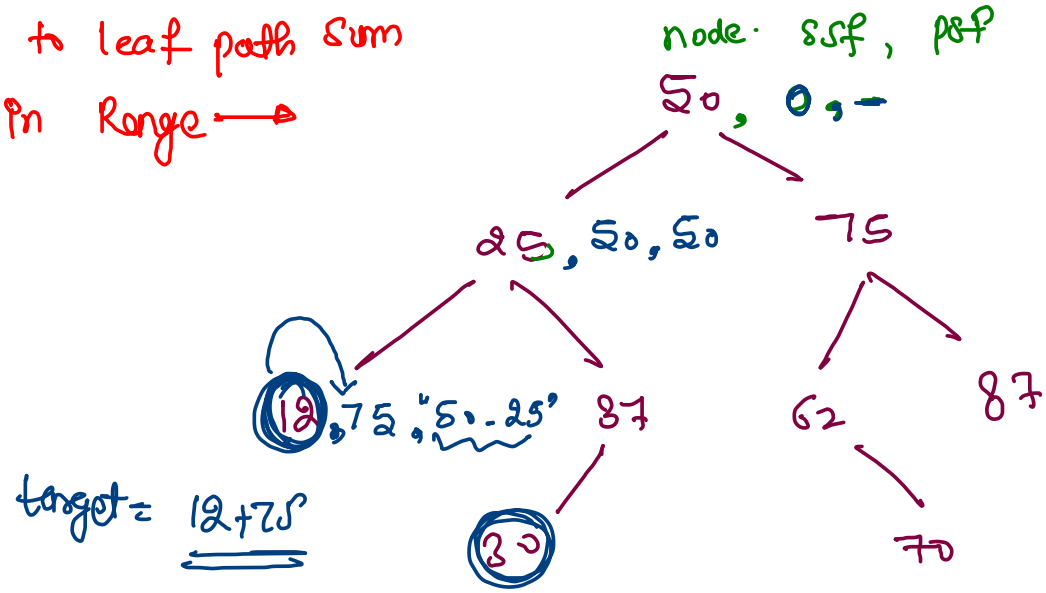
k = 2.

	e	b	a
k =	2	1	0
Blockage	Null	e	b

n o l m d a


lower Rng < Root to leaf path sum < upper Rng
= sum so far
= path so far
50
25
75
12
37
62
87
87
212
30
142
257
Upper range → 250
Lower range → 100
50 25 37 30
50 75 87

Root to leaf path sum in Range ⟶



target = 12+75

50 25 12

path

node.data

Sum so far

path so far

100 → lower Range

I

250 → upper

note → Revise -

Day-1

Day 6

Day Revision Go through