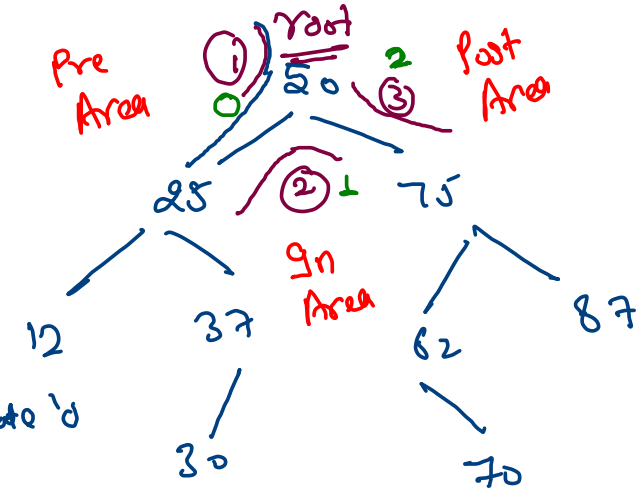
Iterative Pre Post InOrder in B-Tree

Stack → Pair
↓
Node
state.

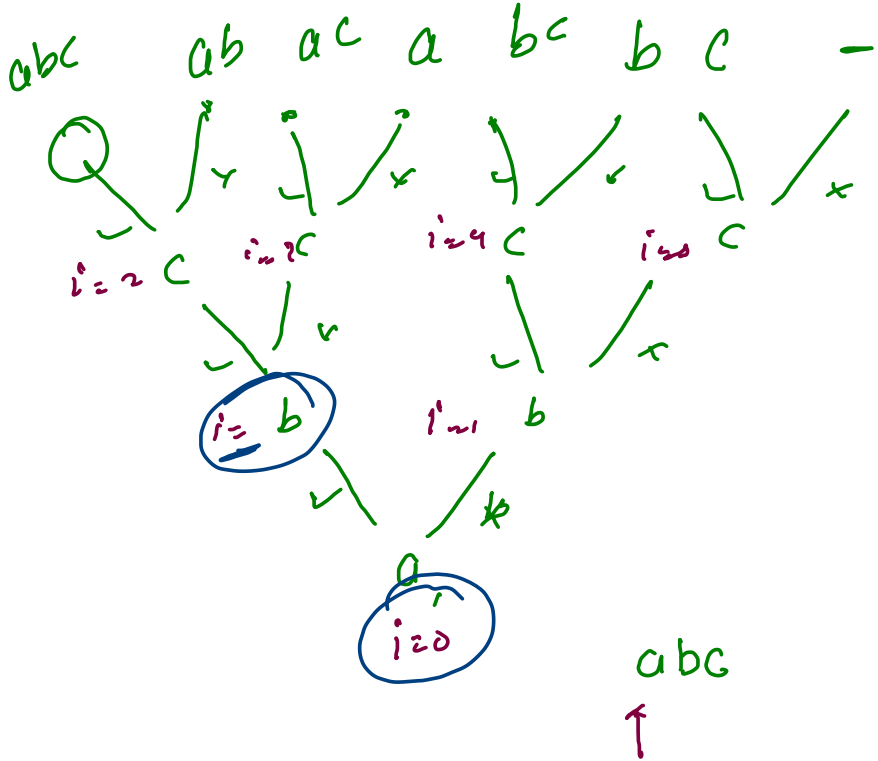



```
State == 0
    pre.add(node.data)
    add left child in stack with state 1
    state ++
State == 1
    in.add(node.data)
    add right child with state 0
    state++
State == 2
    post.add(node.data)
    wipe out
```

Pre - 50 25 12 37 30 75 62 70 87

In → 12 25 30 37 50 62 70 75 87

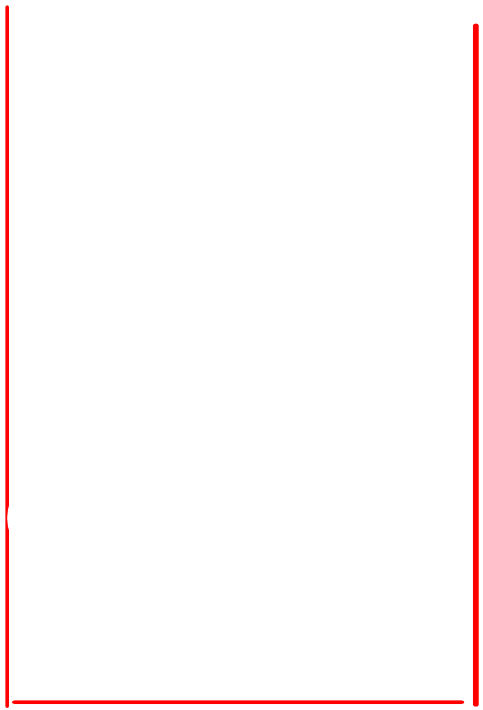
Post → 12 30 37 25 70 62 87 75 50



content on a single level.

```
index, ans
if(index == str.length) { answer.print
                          pop.
    State = 0 }
        st.push(new pair(i+1, ans+str.charAt(i));
        State++
    State = 1
        st.push(new pair(i+1, ans+"(-)");
        State++;
    State = 2
        st.pop()
```

a b c
0 1 2



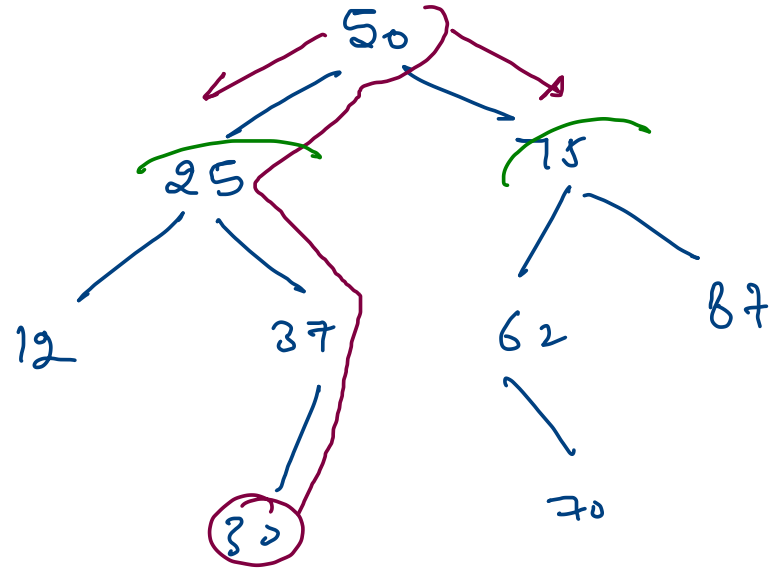
- - -
- - c
- b -
- b c
a - -
a - c
a b -
a b c

Index, answer, state.

```java
public static void iterativeSubseq(String str) {
    Stack<Spair> st = new Stack<>();
    st.push(new Spair(0, "", 0));

    while(st.size() > 0) {
        Spair p = st.peek();
        if(p.indx == str.length()) {
            // base case
            System.out.println(p.ans);
            st.pop();
        } else if(p.state == 0) {
            // no call
            st.push(new Spair(p.indx + 1, p.ans + "- ", 0));
            p.state++;
        } else if(p.state == 1) {
            // yes call
            st.push(new Spair(p.indx + 1, p.ans + str.charAt(p.indx) + " ", 0));
            p.state++;
        } else {
            // wipe out
            st.pop();
        }
    }
}
```

Calls mangat

find and Node to Root path.



① find - 30 T/F

② Node to root path.

30 → 37 → 25 → 50

find → Expectation

find(50) → True (Present)

find(55) → False (Absent)

find(55.left) → F

find(50.right) → F

null → return faith, false

Merging →

self check
left check
right check

Arrays.

1. Subarray.
2. Subset

Subarray → {1, 2, 3} → $\frac{3 \times 4}{2} = 6$

$no. = \frac{n(n+1)}{2}$

1. {1}
2. {1, 2}
3. {1, 2, 3}
4. {2}
5. {2, 3}
6. {3}

Subset → {1, 2, 3}

$no = 2^n$

$= 2^3 = 8$

1. { }
2. {3}
3. {2}
4. {2, 3}
5. {1}
6. {1, 3}
7. {1, 2}
8. {1, 2, 3}

Strings.

1. Substring.
2. Subseq

Substring - abc

$no = \frac{n(n+1)}{2}$

1. a
2. ab
3. abc
4. b
5. bc
6. c

Subseq - abc

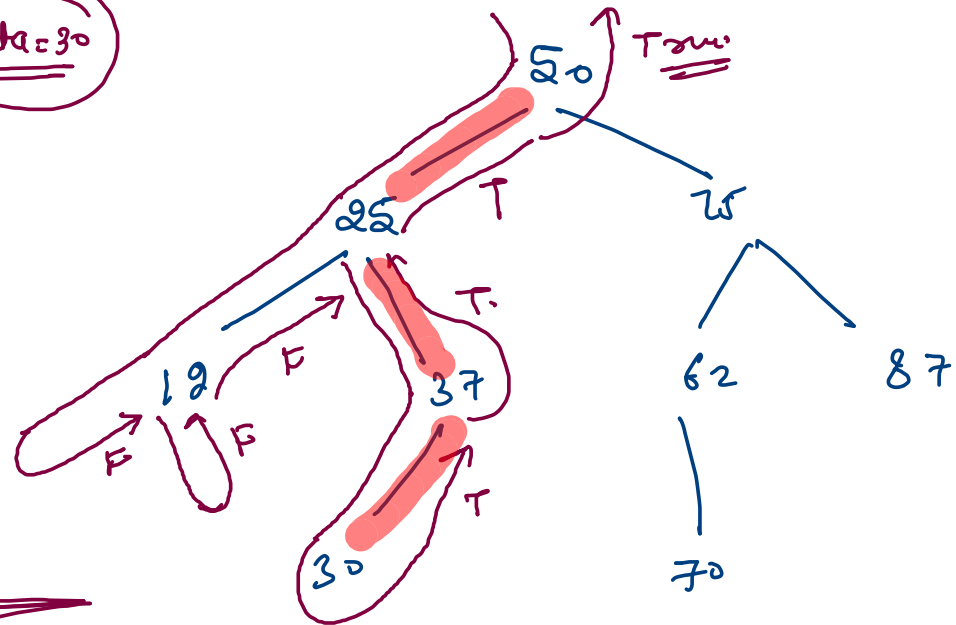
2^n

-	-	-
-	-	c
-	b	-
-	b	c
a	-	-
a	-	c
a	b	-
a	b	c

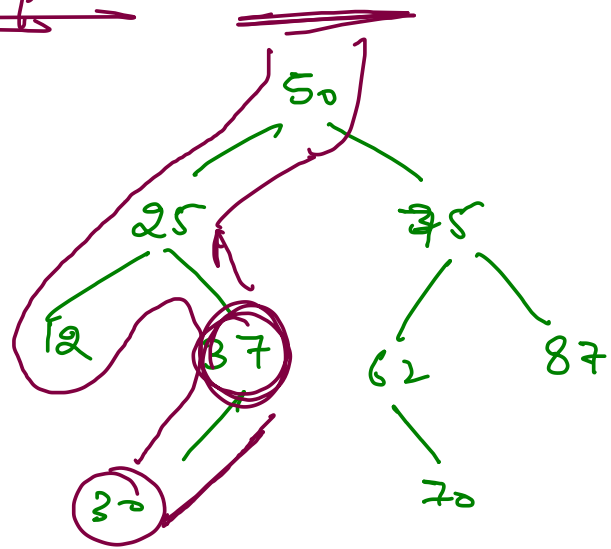
1. -
2. c
3. b
4. bc
5. a
6. ac
7. ab
8. abc

```java
public static boolean find(Node node, int data){
    if(node == null) return false;

    if(node.data == data) return true;

    boolean res = false;
    res = find(node.left, data);
    if(res == true) return true;

    res = find(node.right, data);

    return res;
}
```

data = 30



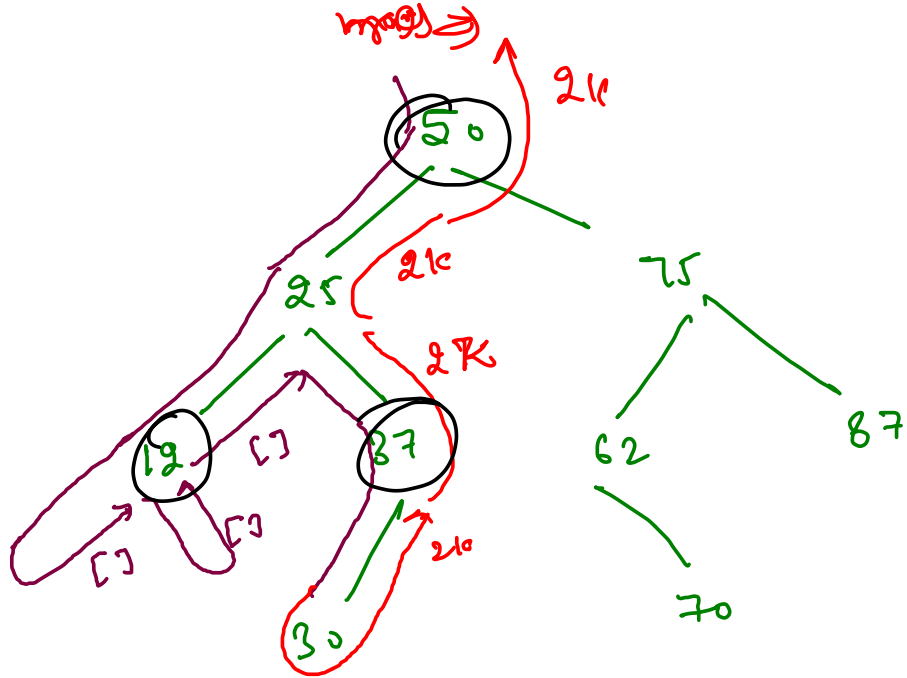
node to Root path.



{30, 37, 25, 50}

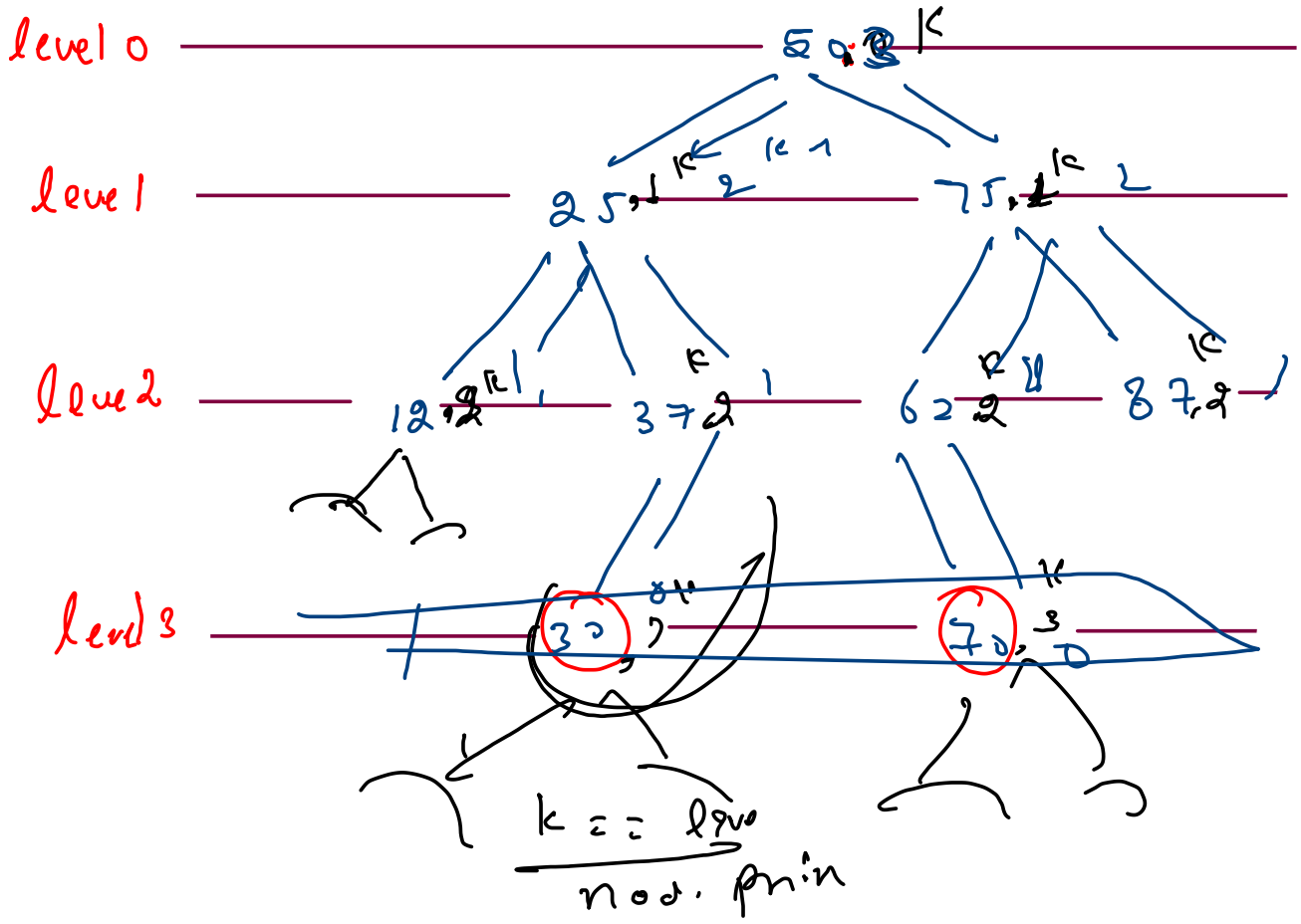
30

```java
public static ArrayList<Integer> nodeToRootPath(Node
    if(node == null) return new ArrayList<>();

    if(node.data == data) {
        ArrayList<Integer> bres = new ArrayList<>();
        bres.add(node.data);
        return bres;
    }

    ArrayList<Integer> res;
    res = nodeToRootPath(node.left, data);
    if(res.size() > 0) {
        res.add(node.data);
        return res;
    }

    res = nodeToRootPath(node.right, data);
    if(res.size() > 0) {
        res.add(node.data);
        return res;
    }

    return new ArrayList<>();
}
```



{30, 37, 25, 50}

Print k level Down →



k level Down.

k = 3

Recursion.

null → check

leaf check

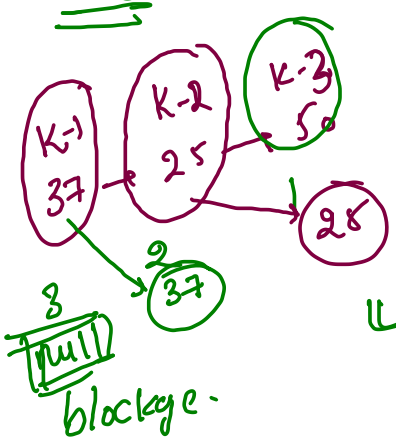
left →

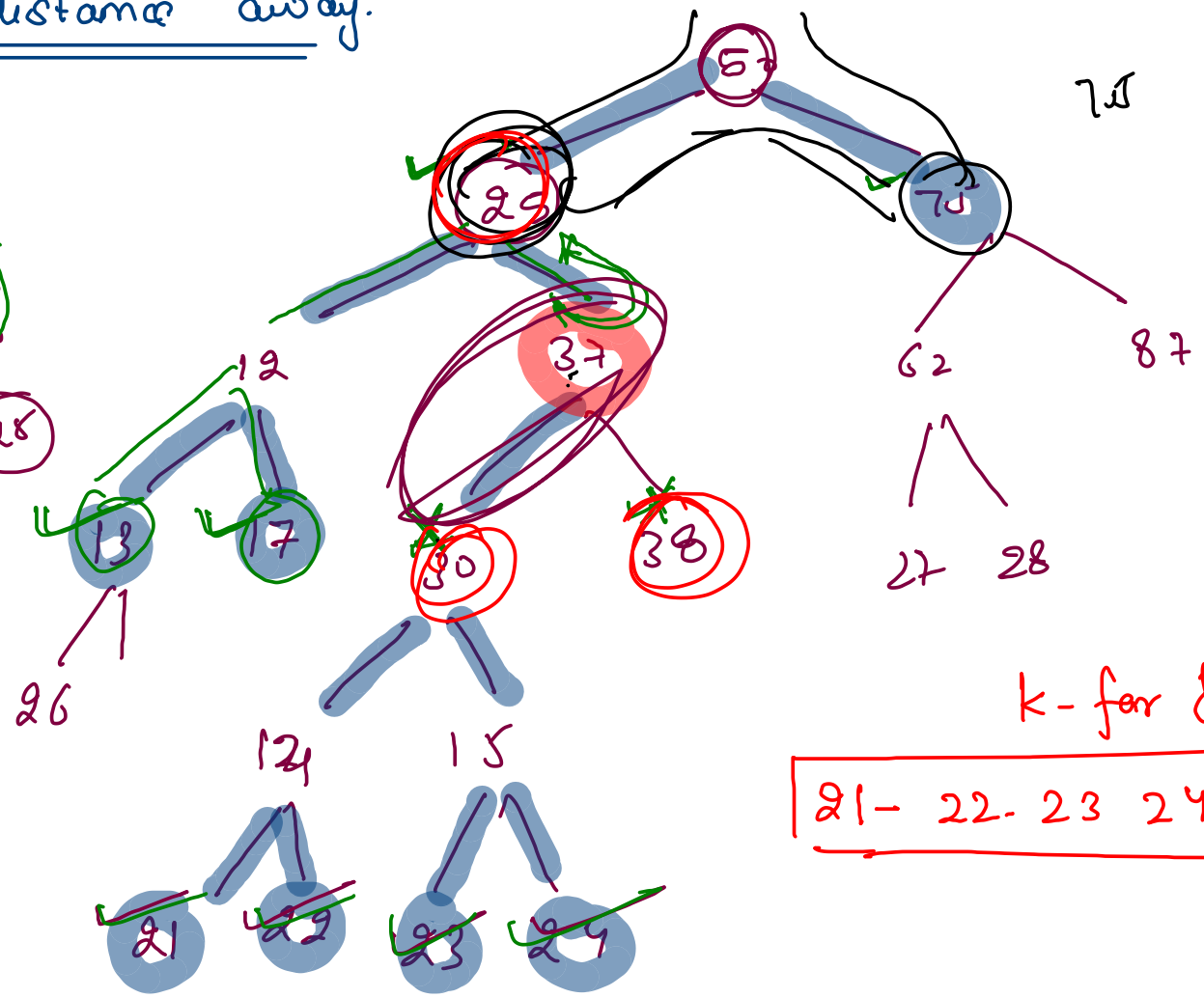
right →

K-far / K. distance away.

data = 37

K = 3



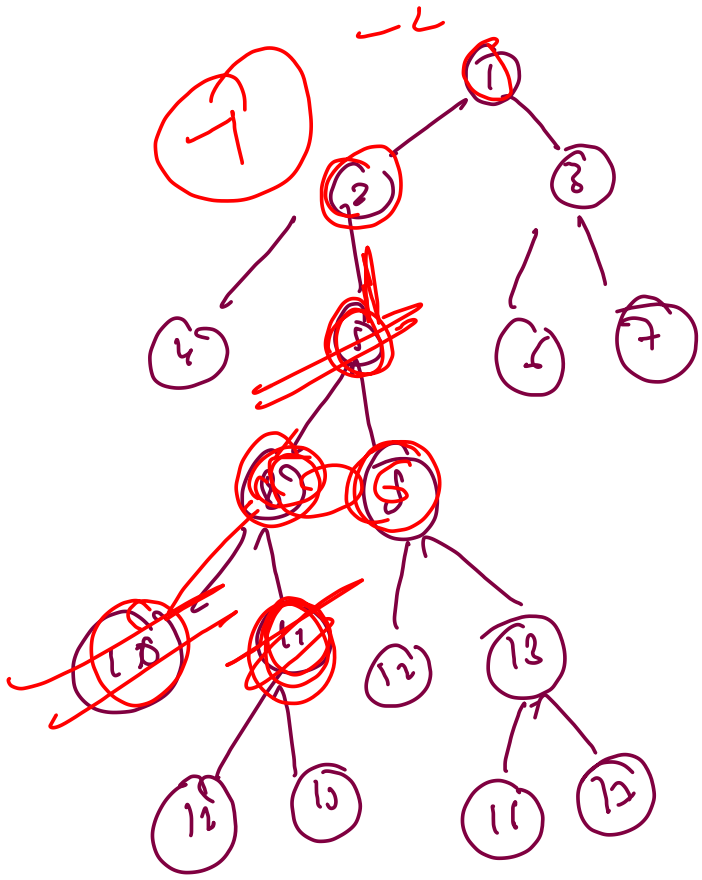


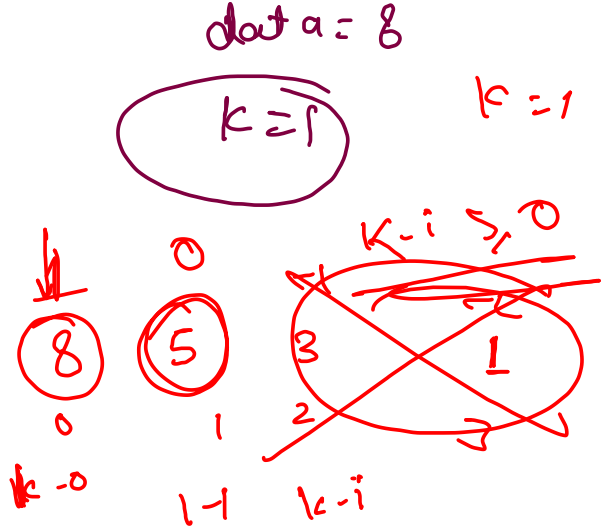
Hint →

Node to root path + # K level down

k-far from 37

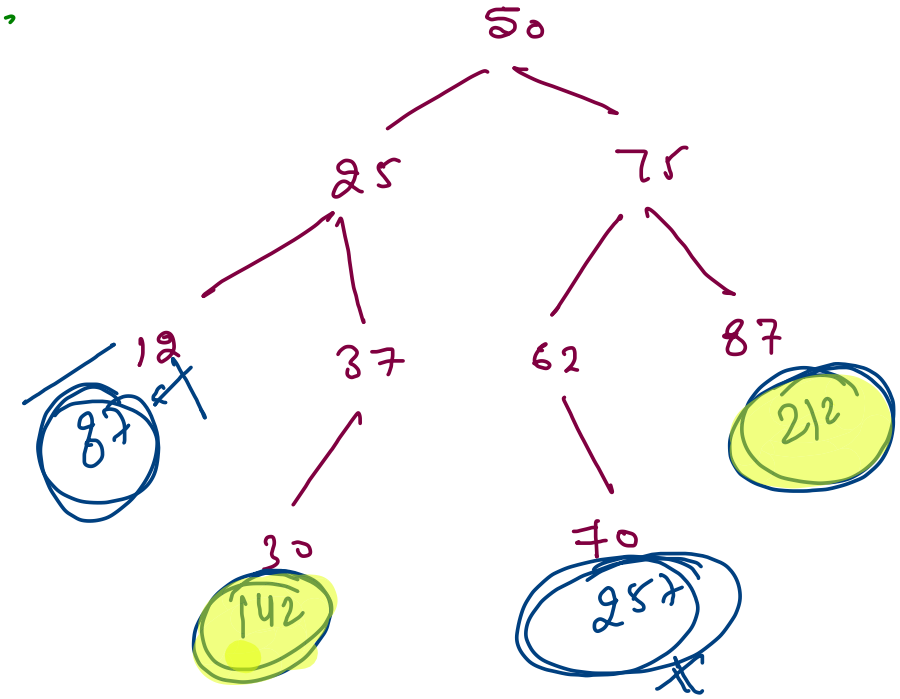
21 - 22 - 23 24 13 17 75

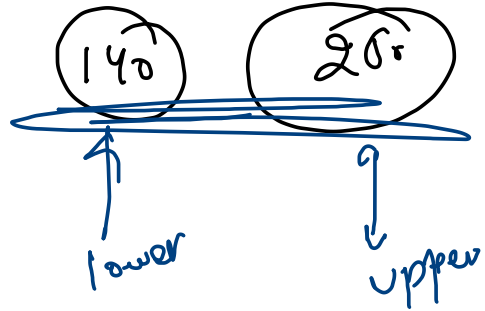



data = 8
k = 1
k = 1
k - i ≥ 0
8
5
3
1
2
0
1
k - 0
i - 1
k - i

Path in Range.

node to leaf





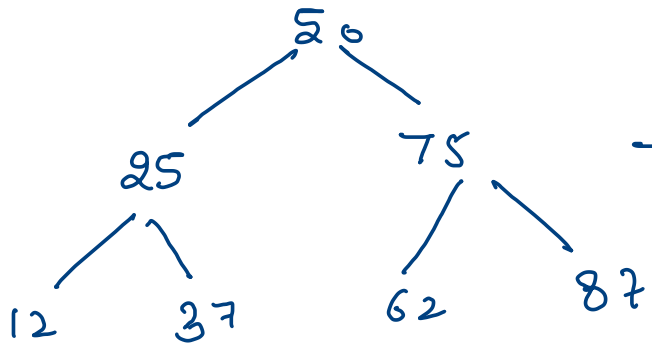
~~left check~~

Right check

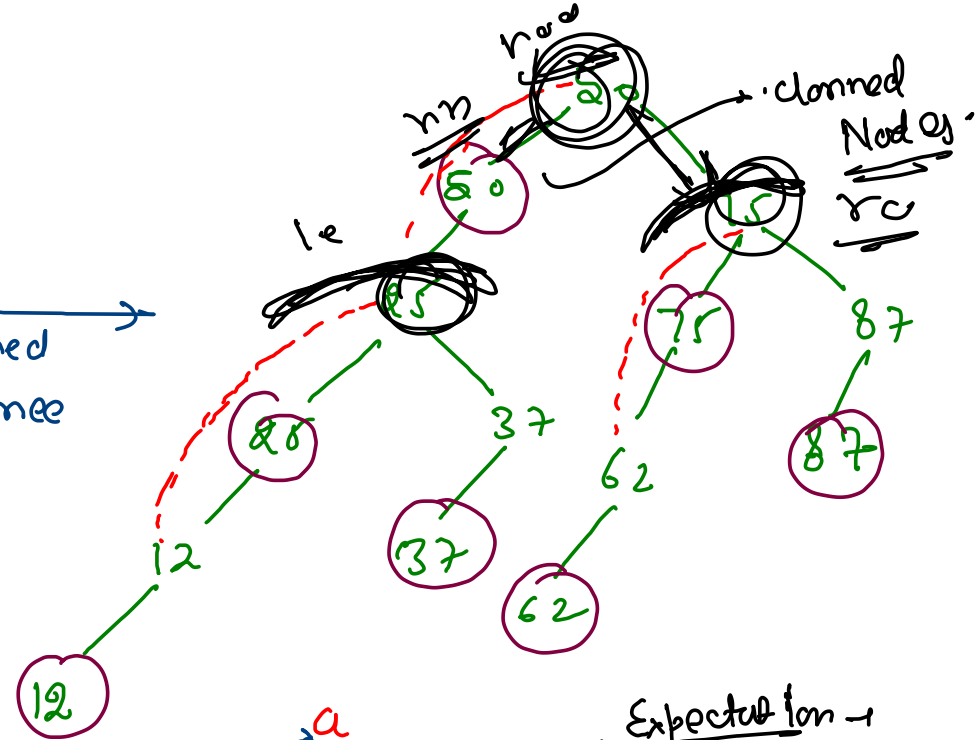
50 25 37 30

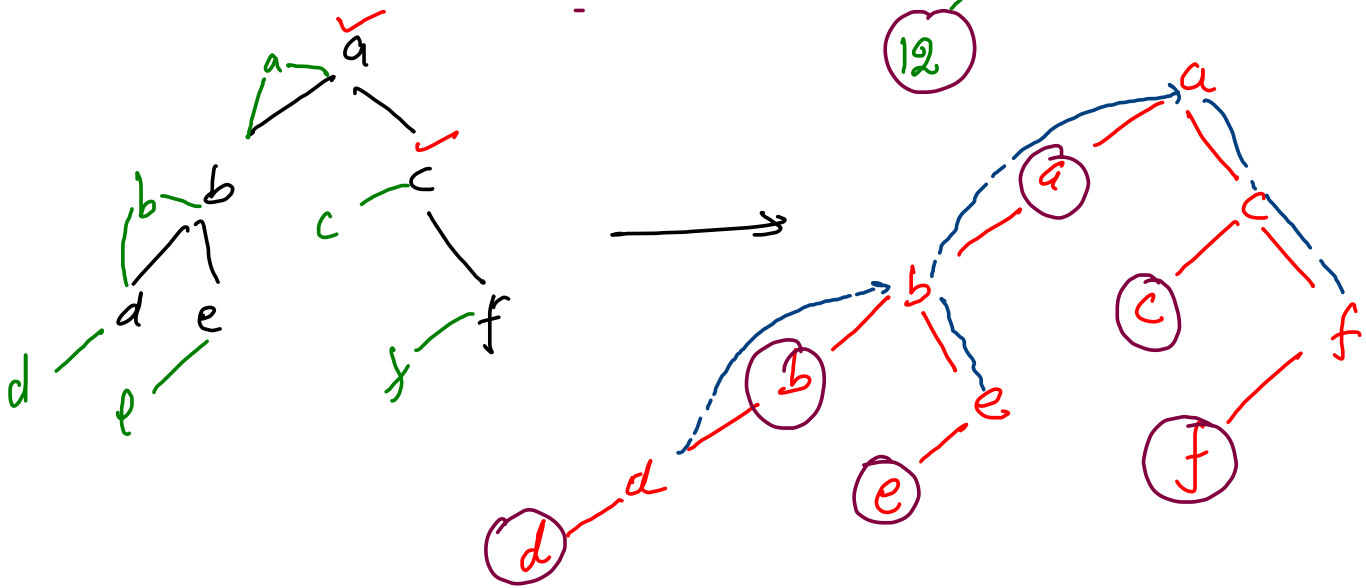
50 75 87

Transform to left cloned tree



left cloned tree





Expectation →

transform(50)

faith: lc = transform(25);

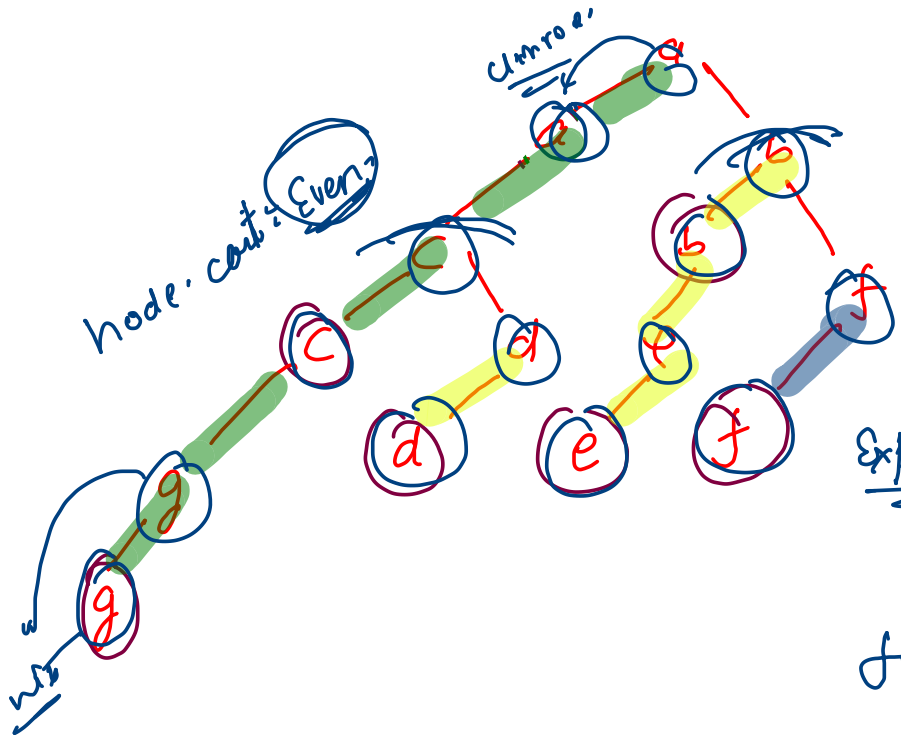
rc = transform(75

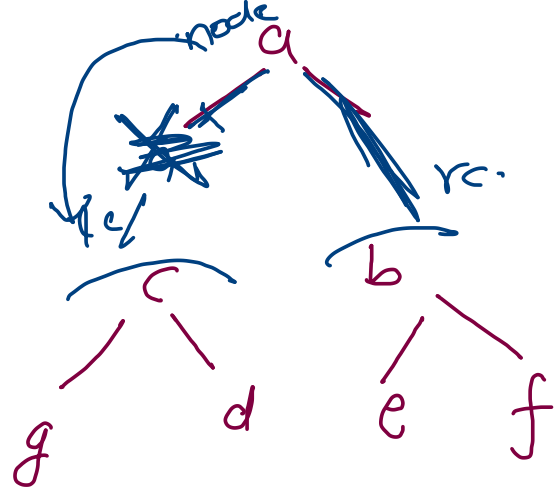
nn

node.left = nn; return nn

nn.left = lc.

left Clonned tree to Normal tree





Expectation:

Retransform (a)

pointer Exception
Null encounter
X

faith →

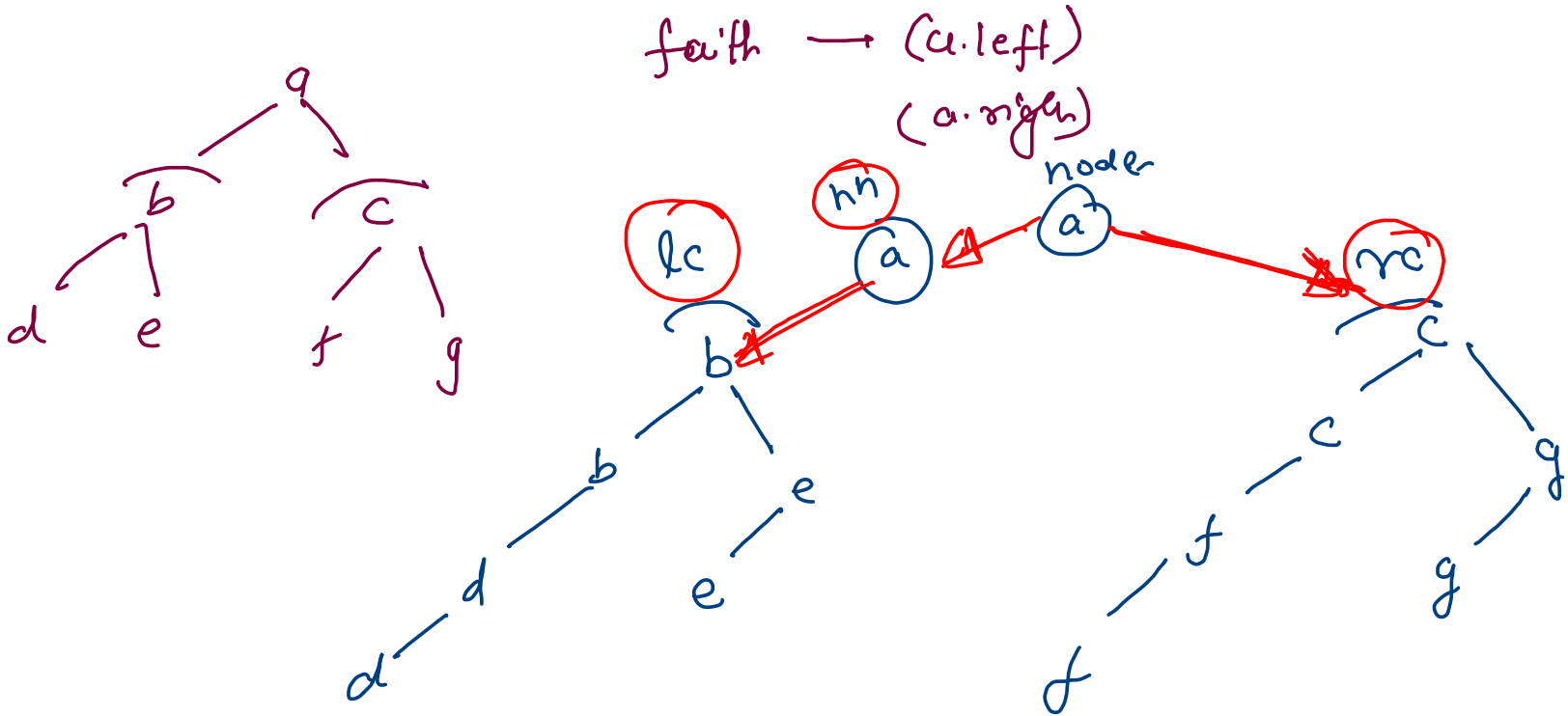
lc = Retransform (a.left.left);

rc → Retransform (a.right);

node.left = lc;

node.right = rc;

return node;

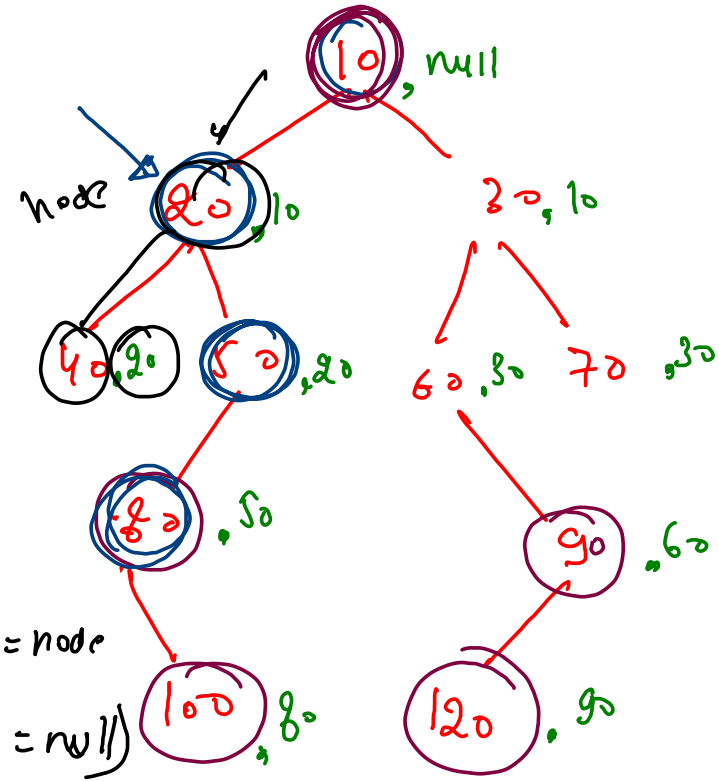



```
Node nn = new Node(node.data);
nn.left = lc;
node.left = nn;
node.right = rc;
return node;
```

Print single child:

left child of my parent

right child of my parent

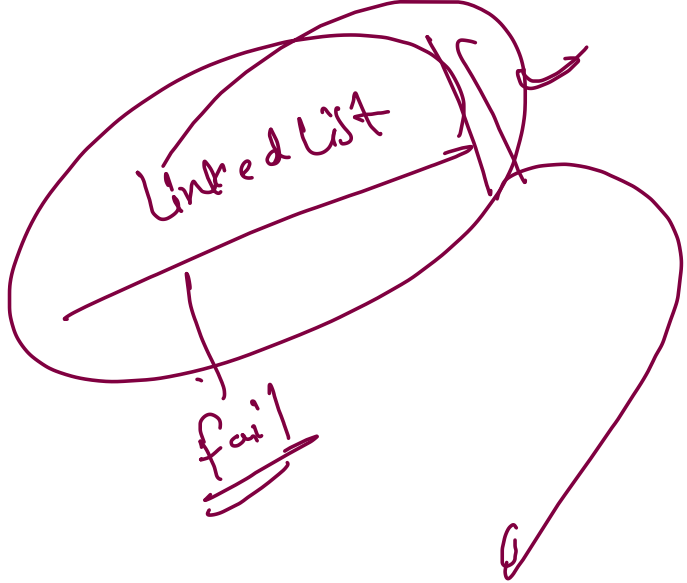


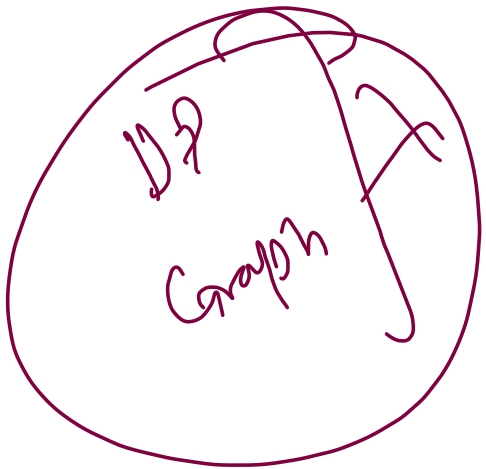
(parent != null && parent.left == node && parent.right == null)

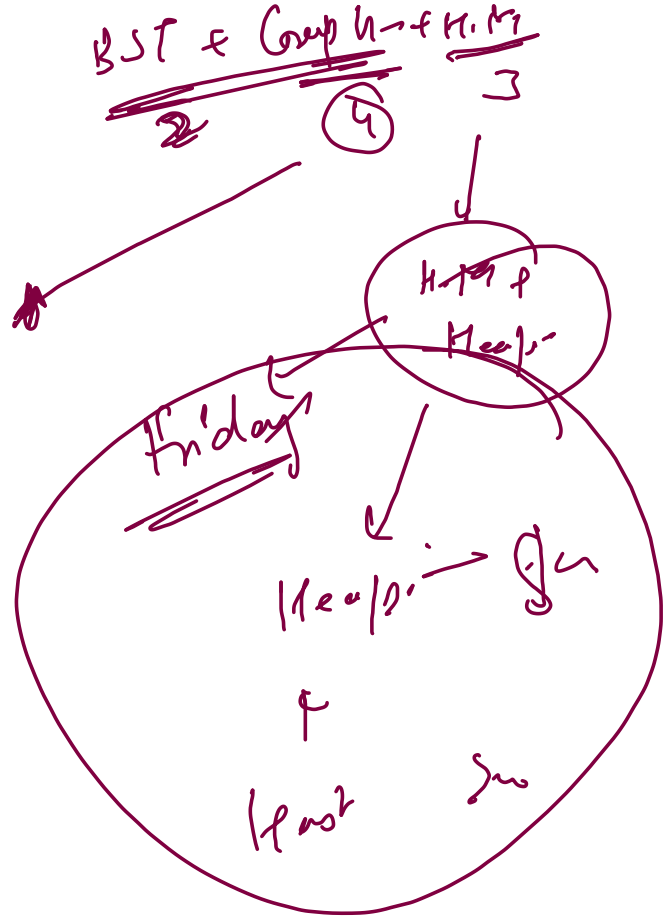
node is single left child of parent

(parent != null && parent.right == node && parent.left == null)

node is single right child


Linked List
fail


DP
Graph


BST & Graph
Heap
Friday