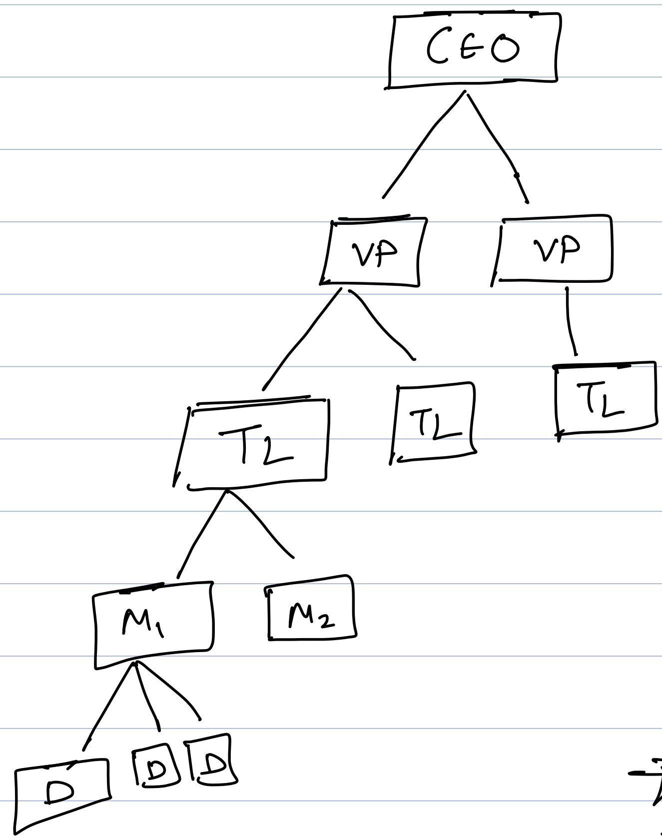
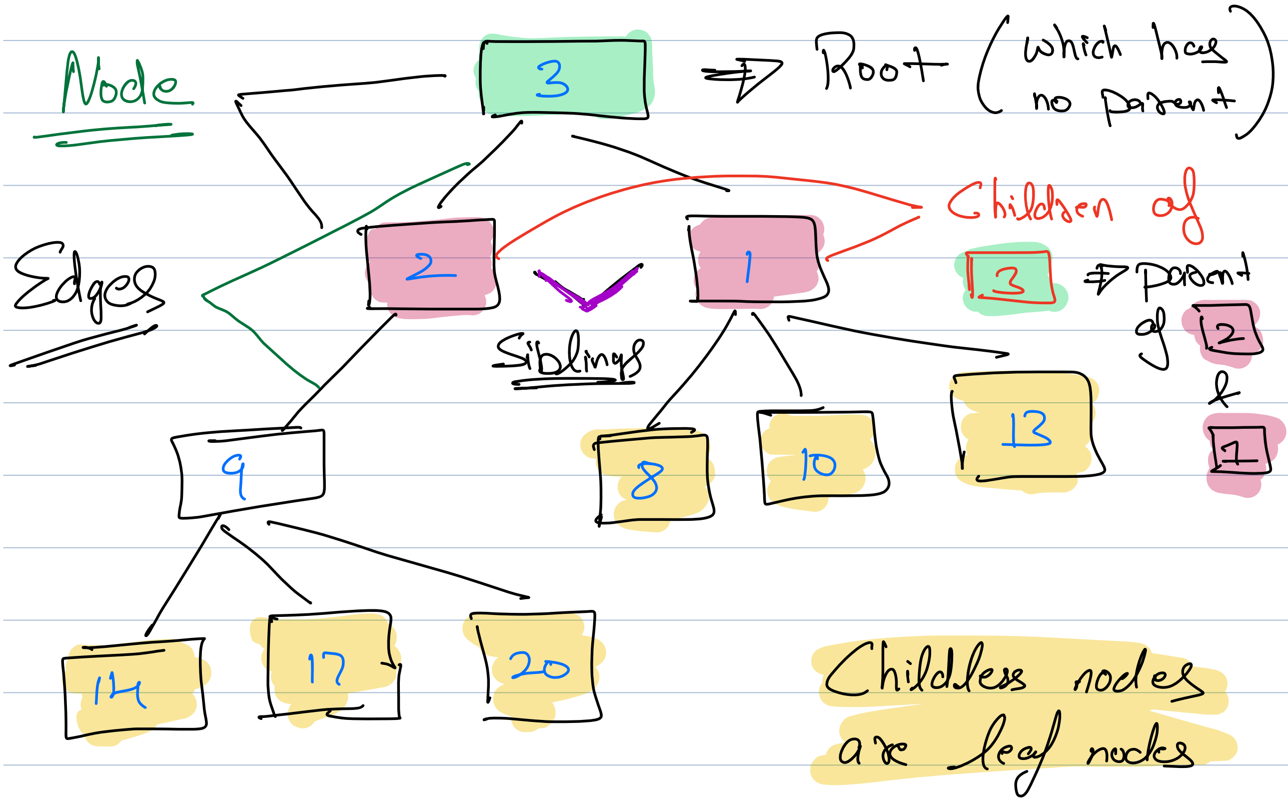


→ Family Tree
→ File structure

Trees

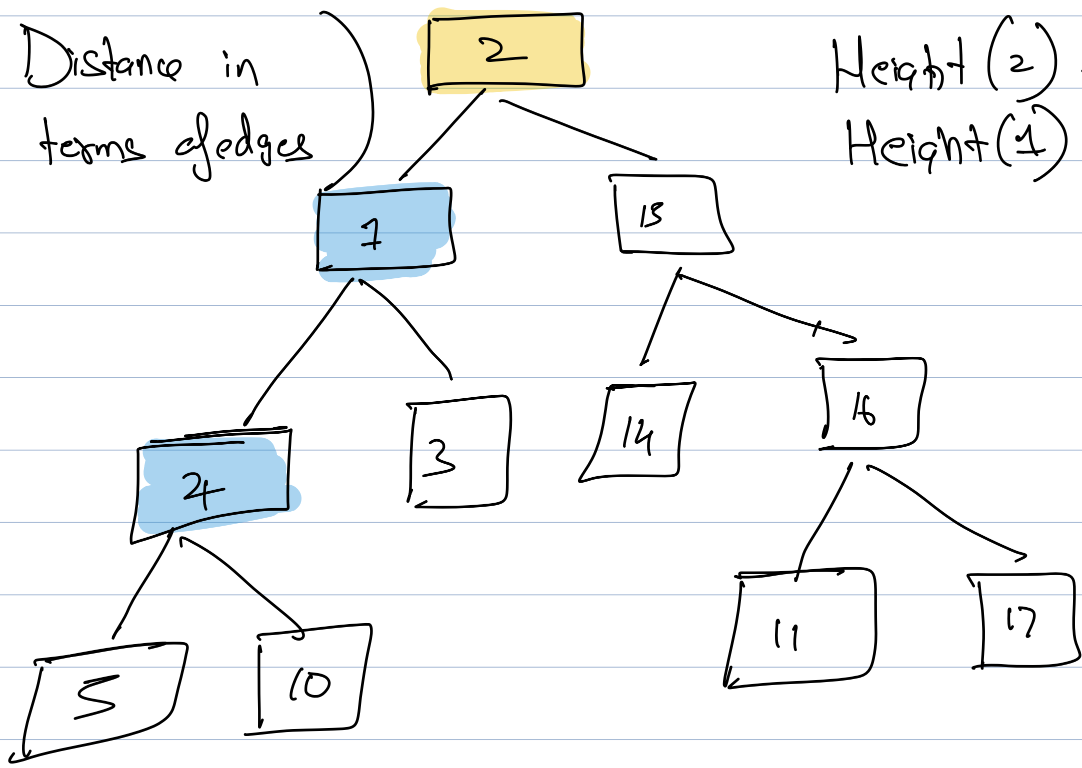
Trees are hierarchical Data Structure.



Height of a node

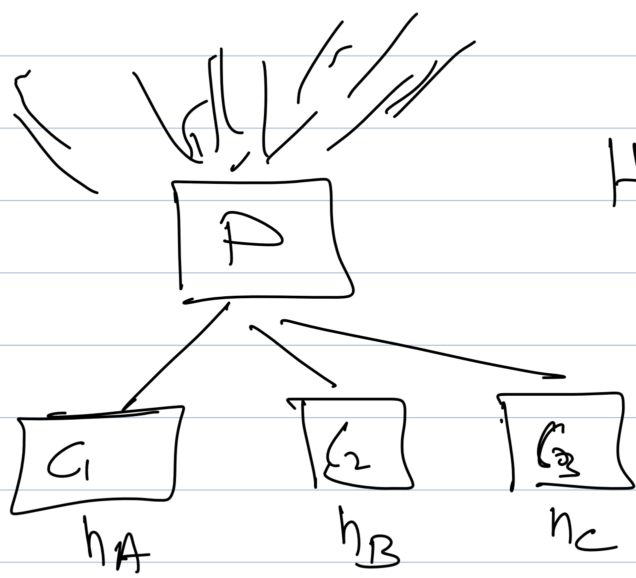
⇒ Distance of the node from the farthest leaf node.

(Distance in terms of edges)



Height(2) ⇒ 3

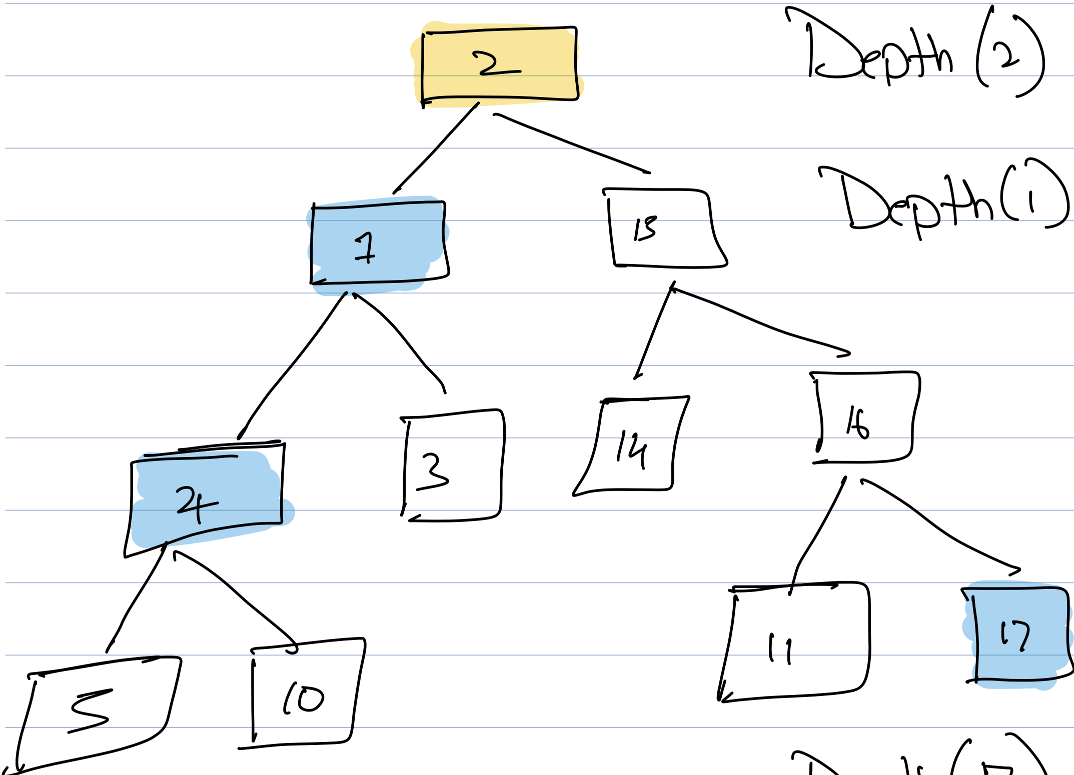
Height(1) ⇒ 2



$$\text{Height}(P) \Rightarrow \max(h_A, h_B, h_c) + 1$$

Depth of a node ⇒ Level

Distance of that node from root.



Depth(2) ⇒ 0

Depth(1) ⇒ 1

Depth(17) ⇒ 3

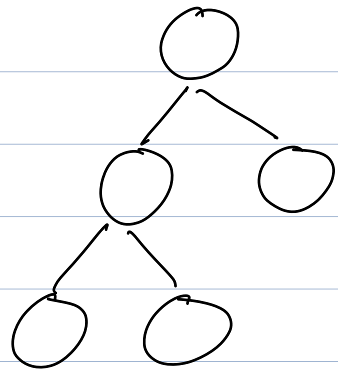
depth(child) ⇒ depth(parent) + 1

Binary tree

[0, 1, 2]

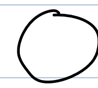
⇒ No of children ≤ 2

Ex1



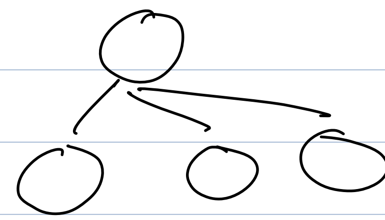
✓

Ex2



✓

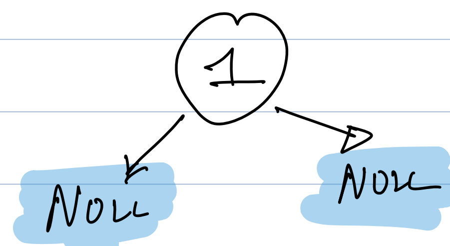
Ex3



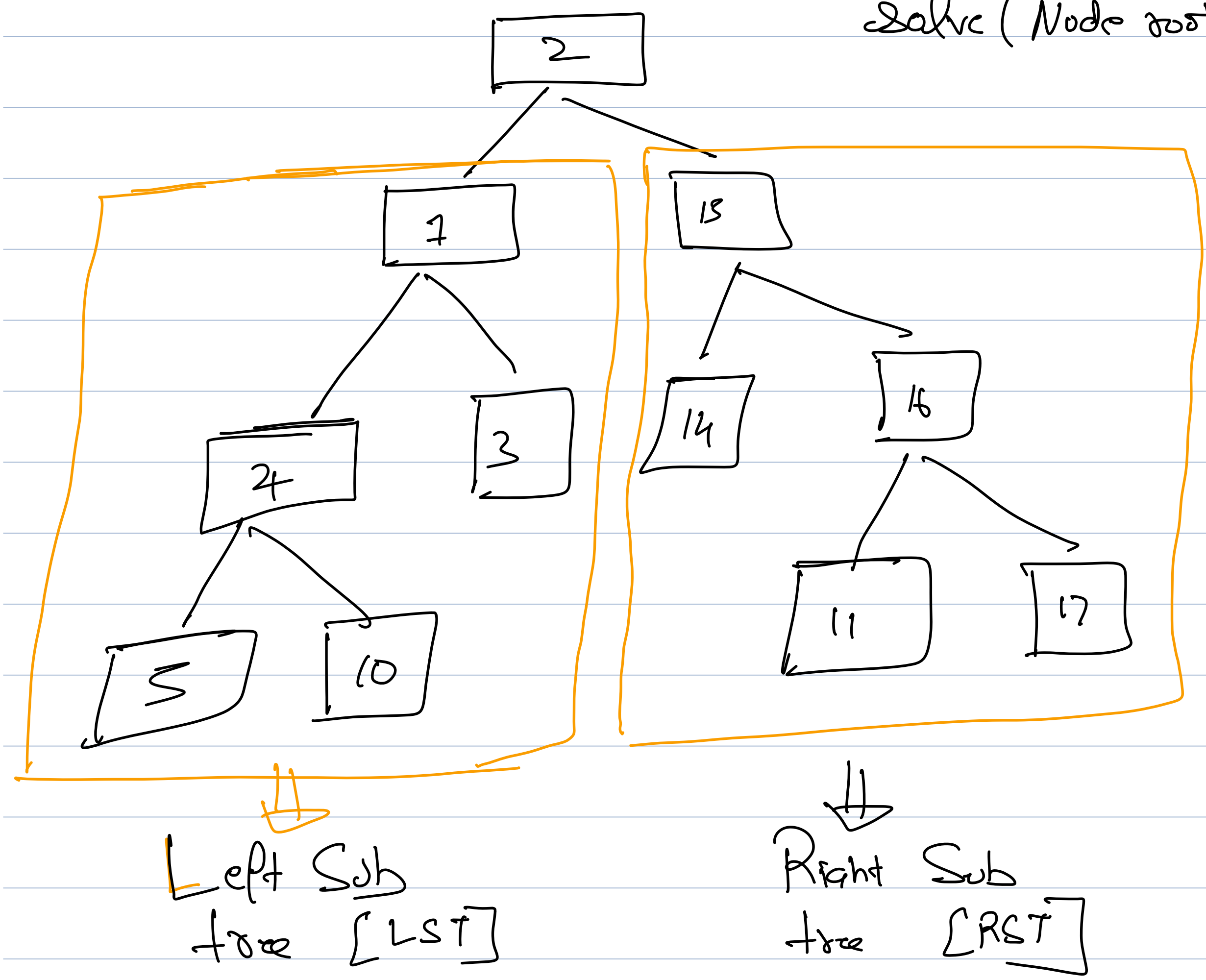
✗

```
Class Node {

    int data;
    Node left;
    Node right;

    Node (int d) {
        data ⇒ d
        left ⇒ null
        right ⇒ null
    }
}
```

Node n_1 ⇒ new Node(1)



solve (Node root);

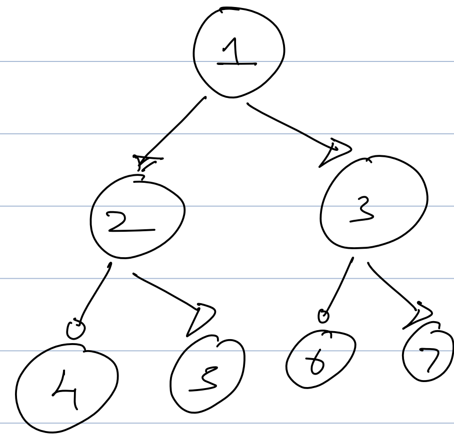


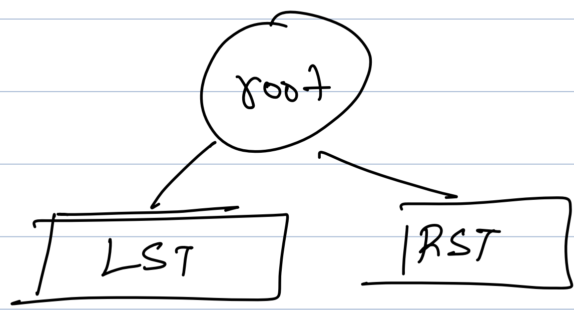
Recursion

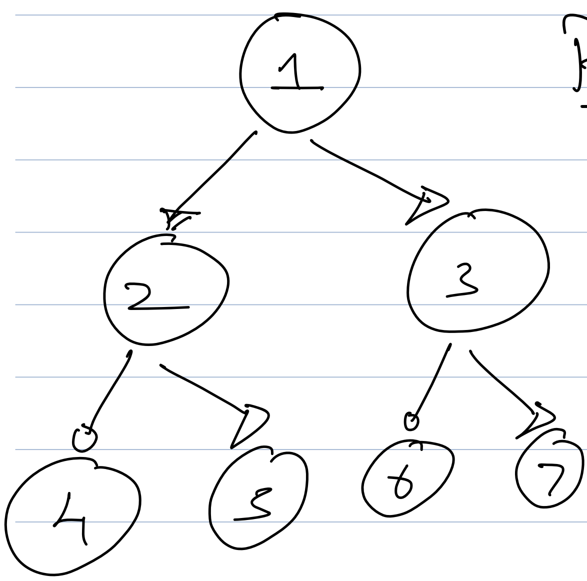
- Manifestation / Assumption.
- Main logic
- Base Cases

Tree traversal

1) Preorder : Root | LST | RST
2) Inorder : LST | Root | RST
3) Postorder : LST | RST | Root







Preorder : 1, 2, 4, 5, 3, 6, 7

Inorder : 4, 2, 5, 1, 6, 3, 7

Postorder : 4, 5, 2, 6, 7, 3, 1

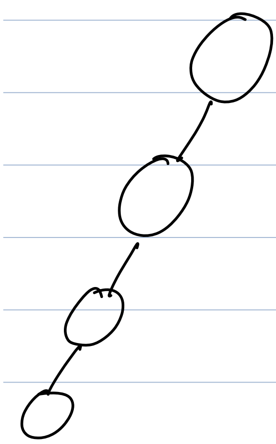
Pseudo Codes

```
Void pre_order (Node root) {
    if (root == null):        => 1
        return;

    print (root.data);        => 2
    pre.order (root.left);    => 3
    pre.order (root.right);   => 4.

}
```

Preorder :	1	2	3	4.
Inorder :	1	3	2	4.
Postorder :	1	3	4	2

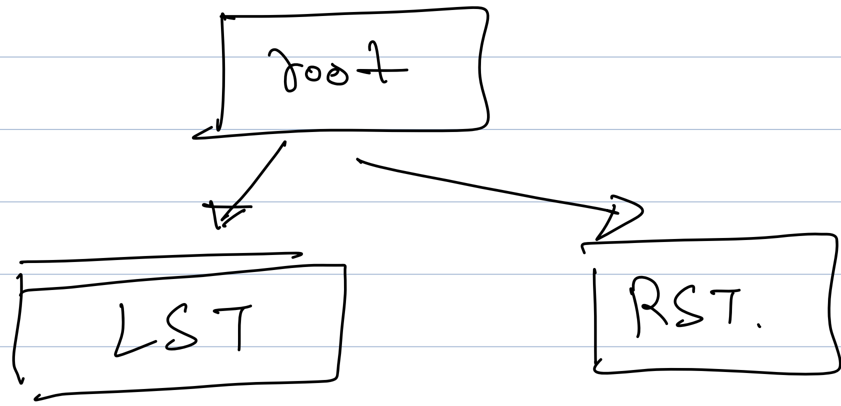
TC : O(n)

SC : O(n)



=> Skewed trees

Q1 Count the number of nodes in a tree.

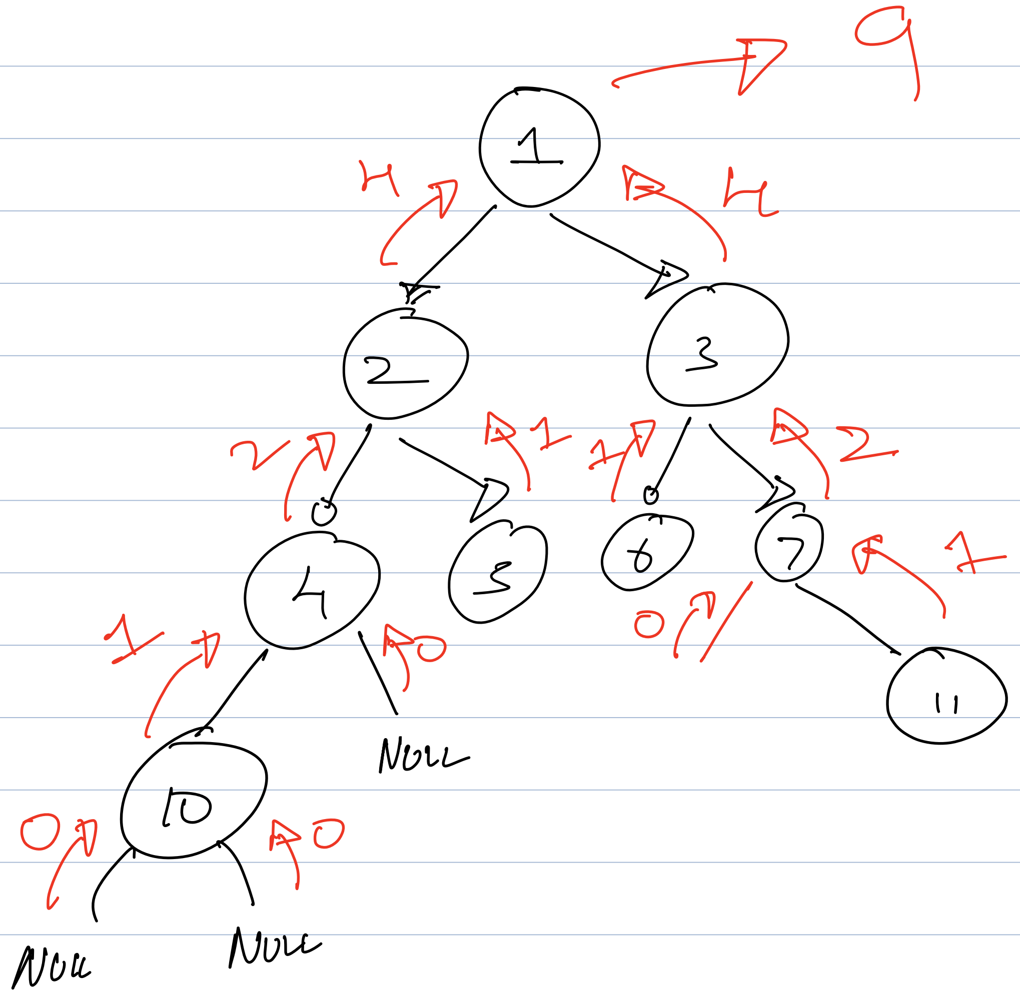


Count (root) ⇒ 1 + Count (LST) + Count (RST)

Pseudo Code

```
int count ( Node root) {
    if (root == null)
        return 0;
    int l ⇒ count (root.left);
    int r ⇒ count (root.right);
    return (l + r + 1);
}
```

Tc : O(n)
Sc : O(n)



9
1
4
4
2
3
2
1
1
2
4
5
6
7
1
0
1
0
11
10
Null
0
0
Null
Null

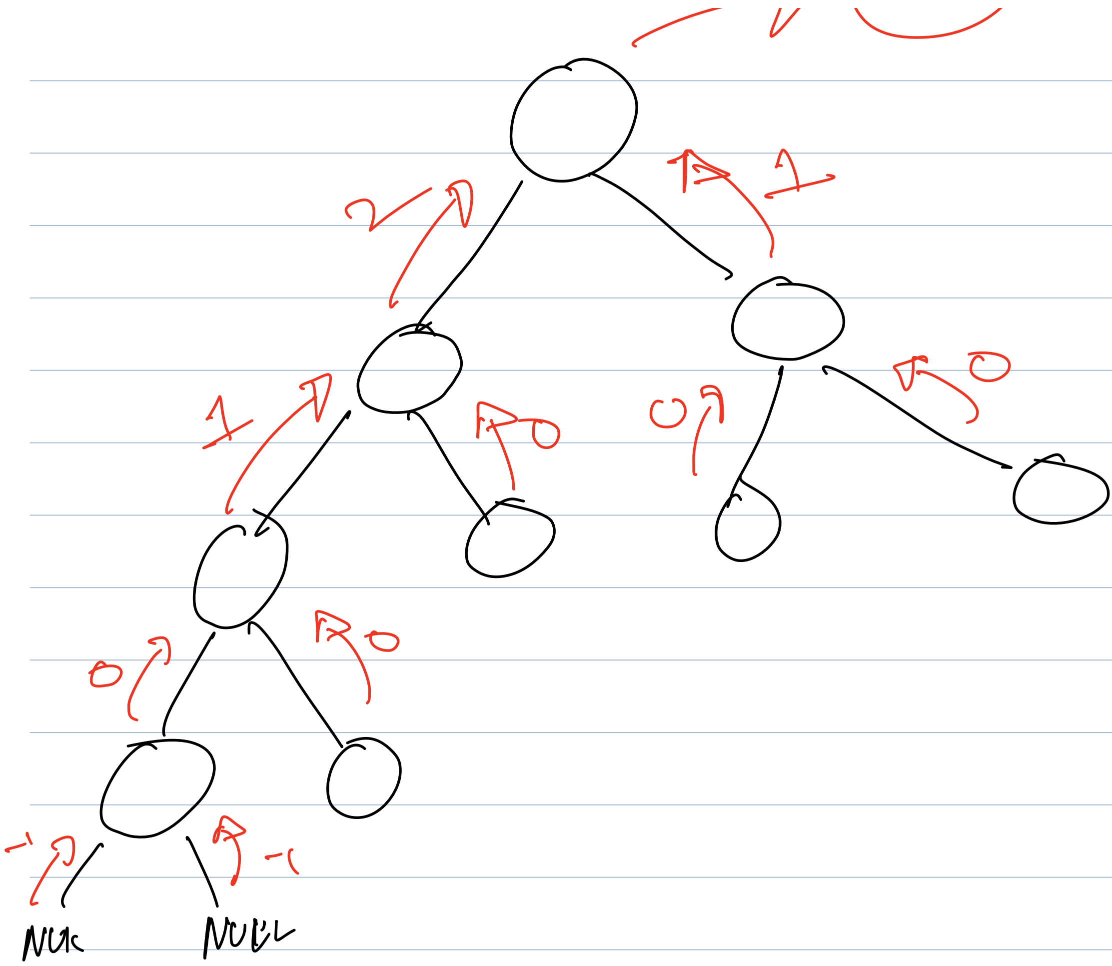
Q Calculate the height of a Binary tree.

⇒ height of the root node

[Edges]

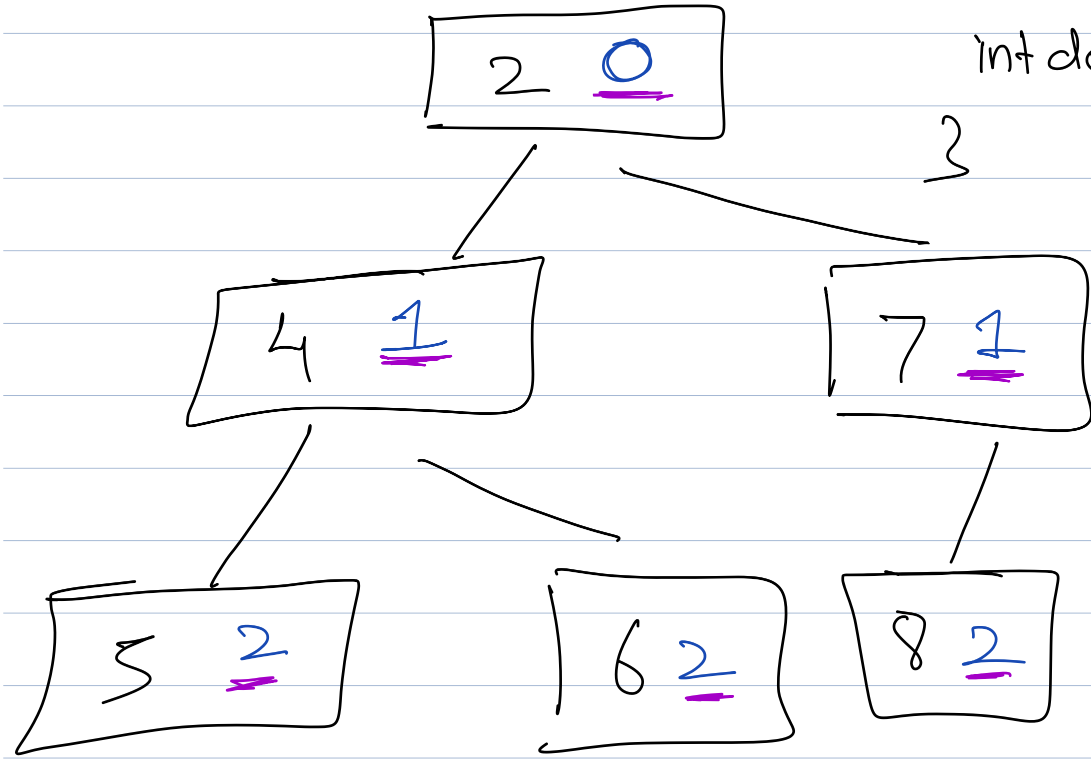
```
int height (Node root) {
    if (root == null)
        return -1;

    int l => height (root.left);

    int r => height (root.right);

    return (1 + max(l, r));
}
```

⇒ (3)


NULL
NULL
2
1
1
0
0
0
0
0
-1
-1

Q3 Given a tree, fill the the depth variable for each node.

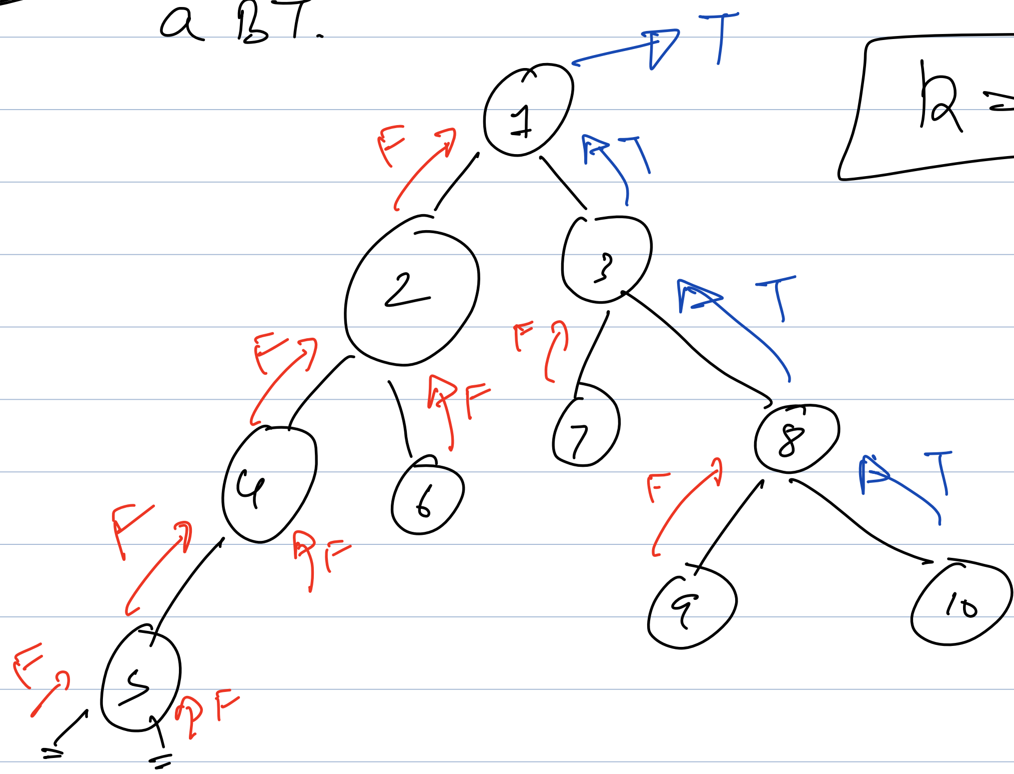
```
Class Node {
   int depth;
}
```




```
void fillDepth(Node root, int d) {
    if (root == null)
        return;

    root.depth = d;
    fillDepth(root.left, d+1);
    fillDepth(root.right, d+1);
}
```

$\boxed{10:32}$

Q Given value k, find if it exists in a BT.



k = 10

```
bool ifExist (Node root, int k) {
    if (root == null)
        return false;

    if (root.data = k)
        return true;

    return ( ifExist (root.left, k)
             ||
             ifExist (root.right, k) );
}
```

TC: O(n)
SC: O(n)

Q Given a BT with unique values, get the path of Node k from root.

Assume that k exists in the tree.

```
list <int> l;

bool ifExist (Node root, int k) {
    if (root == null)
        return false;
    if (root.data == k) {
        l.add (root.data);
        return true;
    }

    bool ans = ifExist (root.left, k) || ifExist (root.right, k);

    if (ans)
        l.add (root.data);

    return ans;
}
```

TC: O(n)
SC: O(n)

TC: O(n)
SC: O(n)