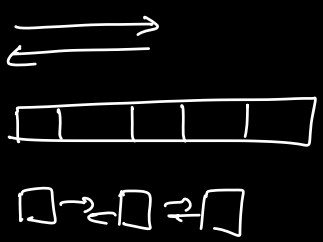
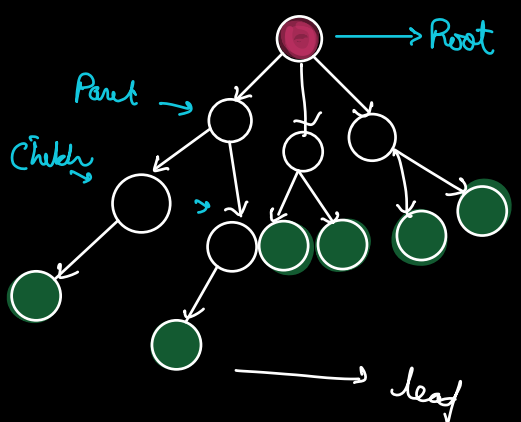
XML
HTML
YAML
JSON
DOM
Directory structure

Map/Set
✓ Heap (Priority Queue)
✓ Trie
Segment Tree
B-Tree (DB indexes)

Amazon, MS, Google, Facebook, Adobe, Expedia, ...





Root → No parent

Leaf → No children

Edges → Connections b/w the nodes

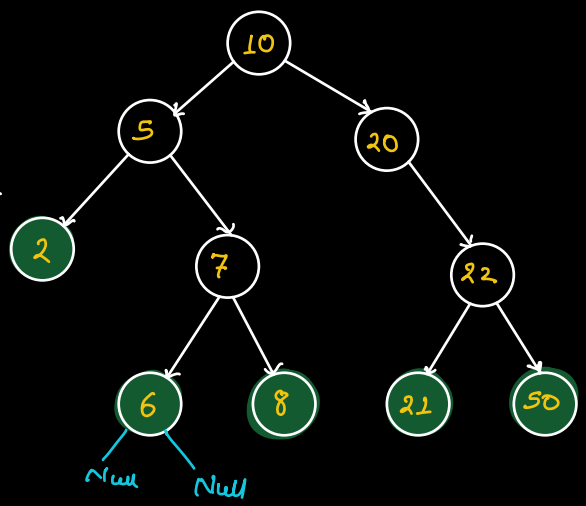
No of edges in a tree with N nodes ?

→ (N-1)

Height of a Node

Distance from that node to the farthest reachable leaf node.
(No of edges)

Ht(5) = 2
Ht(7) = 1
Ht(6) = 0
Ht(Null) = -1

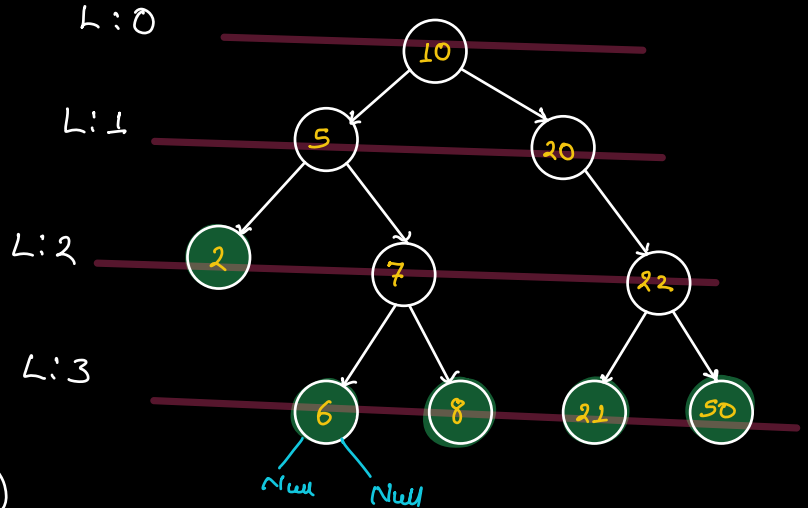


Depth of a Node

Distance from the root node

depth(7) = 2
depth(10) = 0
depth(6) = 3

```
Binary Tree
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
}
```

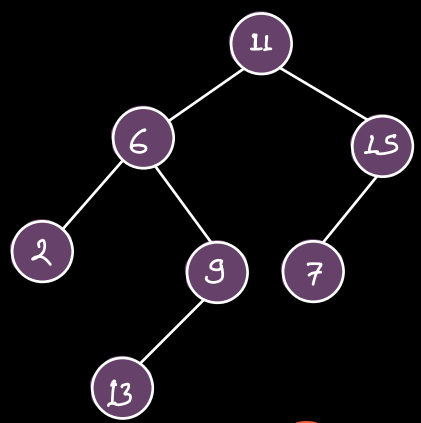


Height of BT → Ht(root)

Tree Traversals

left before right

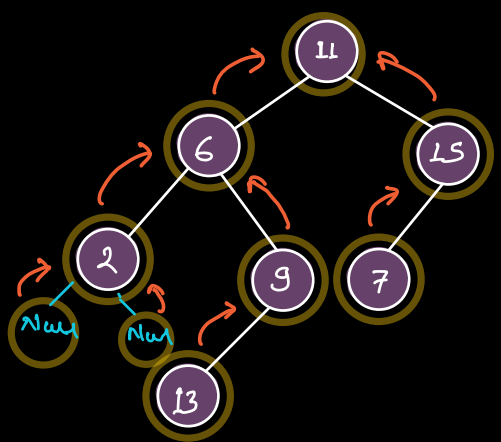
PreOrder	Root	left	right
InOrder	left	Root	right
PostOrder	left	right	Root



PreOrder

```
void preorder(TreeNode root){
    if(root == Null)
        ret;
    print(root.val);
    preOrder(root.left);
    PreOrder(root.right);
}
```

DFS

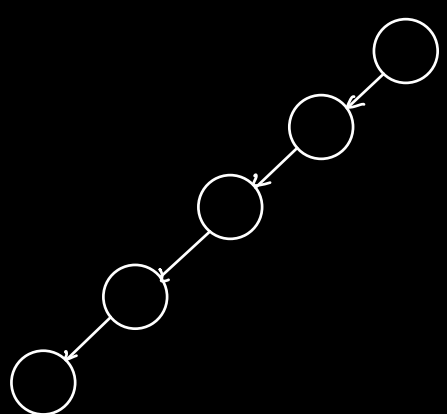


11, 6, 2, 9, 13, 15, 7

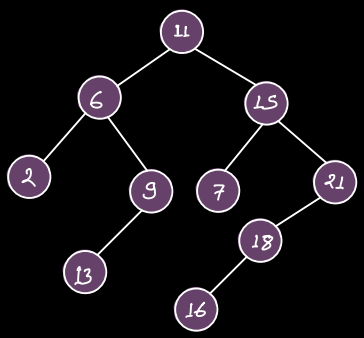
Fail Fast Approach

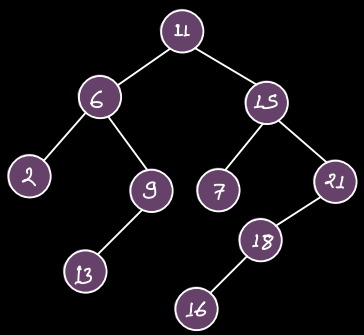
TC : O(N)

SC : O(Ht of tree) ⇒ O(N)



Given 2 trees. Check if they are identical.

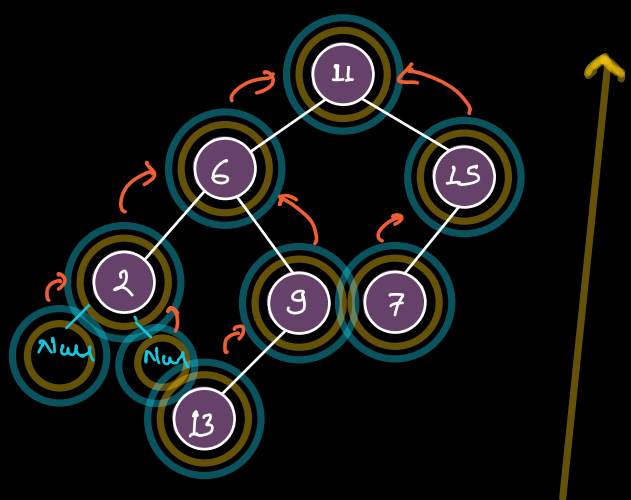




PostOrder

```
void postOrder(root) {
    if (root == null)
        ret;
    postOrder(root.left);
    PostOrder(root.right);
    print(root.val);
}
```

DFS



2, 13, 9, 6, 7, 15, 11

Q Given a tree. Return the ht.

Ht(tree) ⇒ Ht(root)

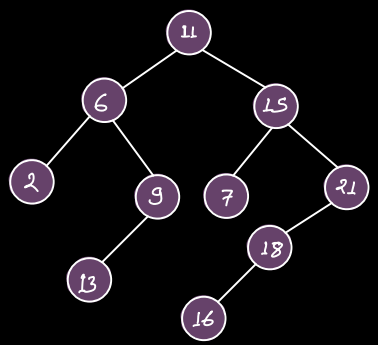
⇒ max(Ht(left), Ht(Right)) + 1

```
int height(root){
    if(root == null)
        ret -1;
    int l = height(root.left);
    int r = height(root.right);
    ret max(l, r) + 1;
}
```

Q Count no of nodes in a tree.

No of nodes = #nodes(left) + #nodes(right) + 1

Q Given a BT. search a val K in it.




```
boolean search (root, K) {
    if (root == null)
        ret false;
    if (root.val == K) {
        ret true;
    }
```

Pre

```
    ret search (root.left, K)
                ||
        Search (root.right, K);
}
```

____ || ____

↑ true — ↑ Not be executed.

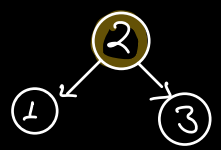
____ && ____

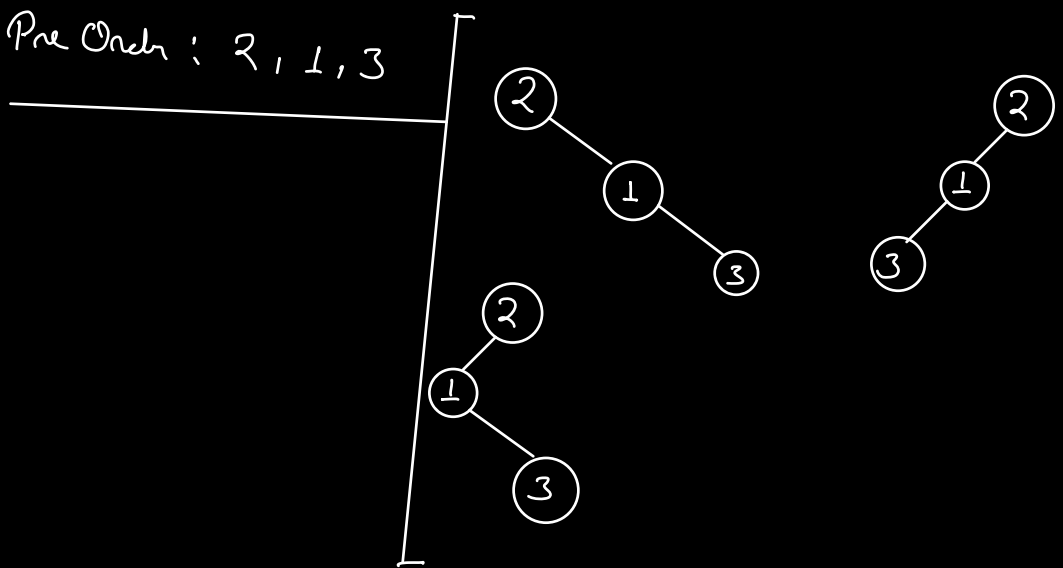
↑ false — ↑ Not be executed.

Amazon
MS

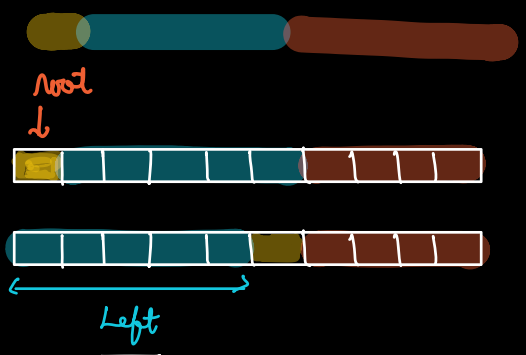
Q Given the pre-order & in order traversal of a tree. Construct the tree. (No duplicates)

Pre-Order : 2, 1, 3
In-Order : 1, 2, 3





The first element of pre-order pre[0] ⟶ root



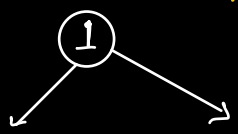
Pre : 1, 2, 4, 8, 5, 3, 6, 7

In : 4, 8, 2, 5, 1, 6, 3, 7

left | right

Root left right

left Root right



Pre : 2, 4, 8, 5

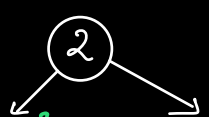
In : 4, 8, 2, 5

left

Pre : 3, 6, 7

In : 6, 3, 7

right

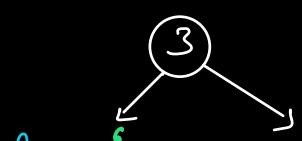


Pre : 4, 8

In : 4, 8

Pre : 5

In : 5

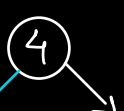


Pre : 6

In : 6

Pre : 7

In : 7



Null

Pre : 8

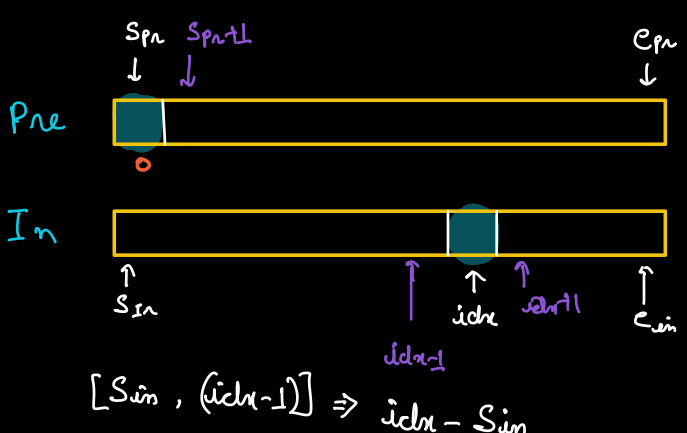
In : 8

5

6

7

8



[Sin , (iclr-1)] ⇒ iclr - Sin

HashMap < Key , value >

↓

in[i] i

In : 4, 8, 2, 5, 1, 6, 3, 7

Key	Val
4	0
8	1
2	2
5	3
1	4
6	5
3	6
7	7

```
TreeNode buildTree(pre[], in[], sin, ein, spr, epr){
// Assumption : buildTree(pre, in, sin, ein, spr, epr)
                returns the root of subtree which can be created
                using in[] -> sin to ein
                &  pre[] -> spr to epr

    if (spr > epr)
        ret null;
    TreeNode root = new TreeNode(pre[spr]);
    // Find index of Pre[spr] in in[];
    int idx = map.get(pre[spr]);
    int x = idx - sin;

    root.left = buildTree(Pre, in, sin, idx-1,
                                    spr+1, spr+x);
    root.right = buildTree(pre, in, idx+1, se,
                                    spr+x+1, epr);
    return root;
}
```

TC : O(N)

SC : O(N) ⟶ HashMap + Recursion

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

c = 1 2 3 4 5 ⑥ — — — —

↑

false