Wednesday

Contest 9 - 10:30

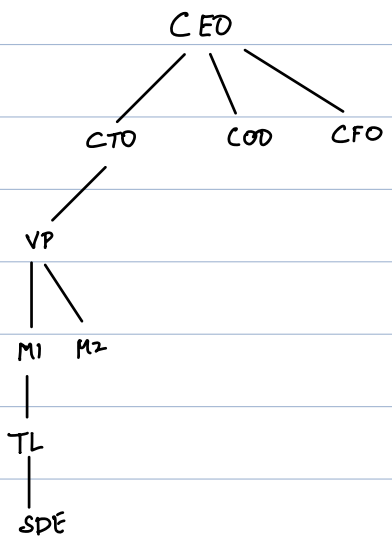
[Strings to LL]

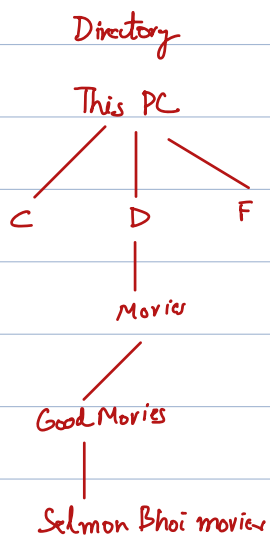
Trees

Strings , Arrays , LL

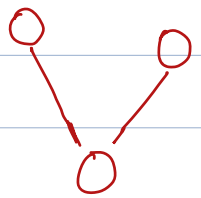
Hierarchical DS

Trees , Graphs





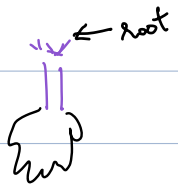
SQL {Relation DB}



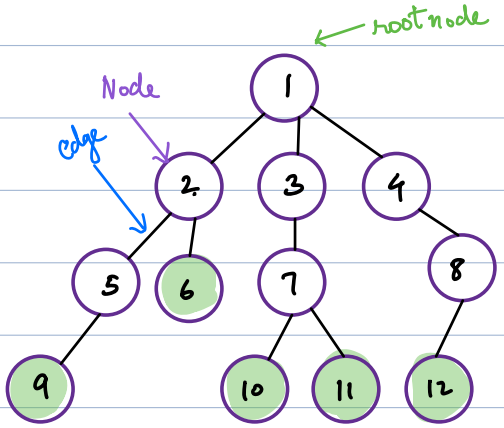
Not a tree

Every node of a tree has only one parent

Naming conventions



1 is ancestor to everyone



4 related to 1

1 ⇄ 4 (parent / child)

8 is a grand child of 1

12 " " descendant of 1

8 " " " " "

4 " " " " "

1 is an ancestor of 7

" " " " " 10

10 is not a descendant of 4, 2

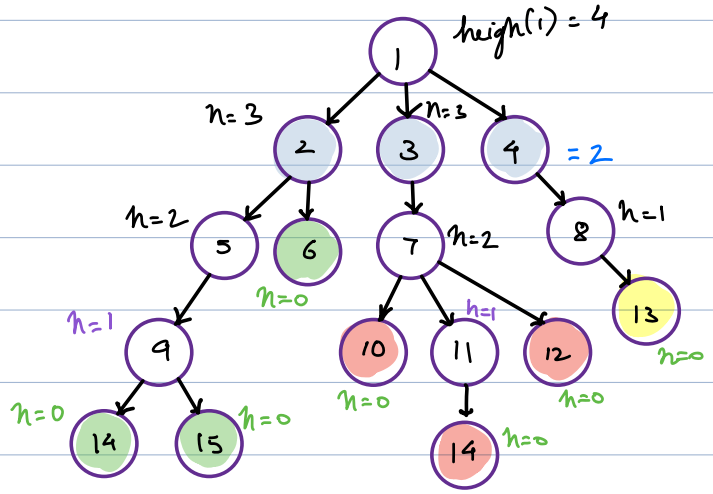
Why? Not in same branch

Leaf nodes: Nodes with 0 children

Height

Height of a node is max path length between a node to its descendant leaf node

Path len: No. of edges b/w nodes



height (node) = max (height of its children) + 1

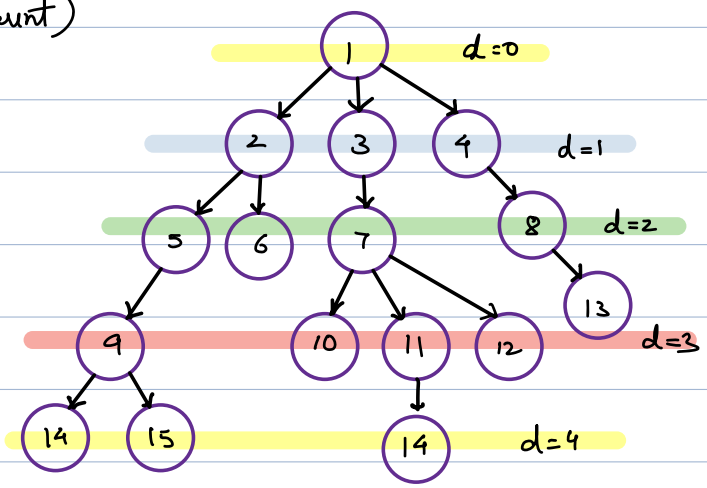
Depth

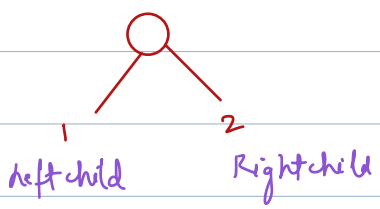
Distance of any node to root node

depth (node) = 1 + depth (parent)

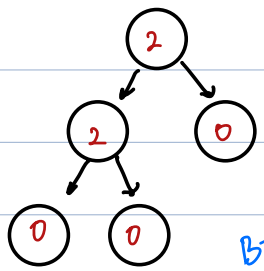
d(root) = 0

by definition

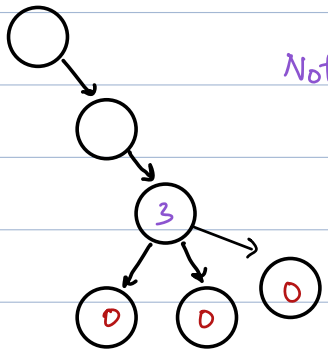



1
2
Leftchild
Rightchild

Binary Tree: A tree having all nodes <= 2 children



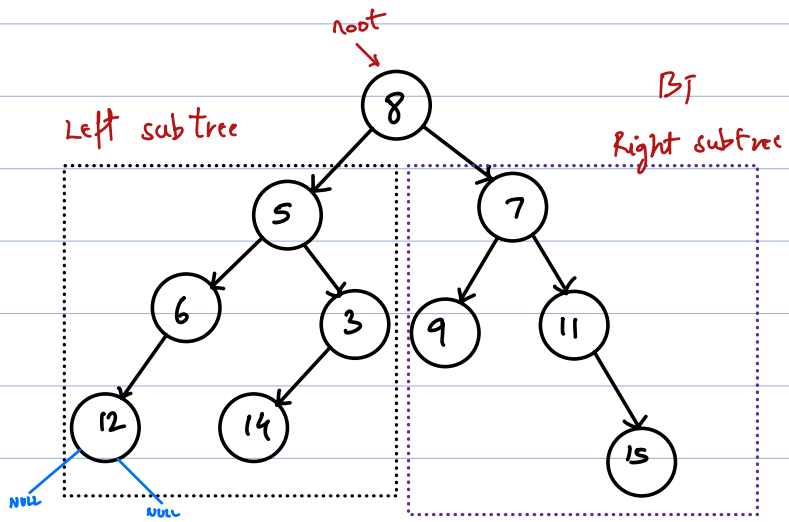
BT



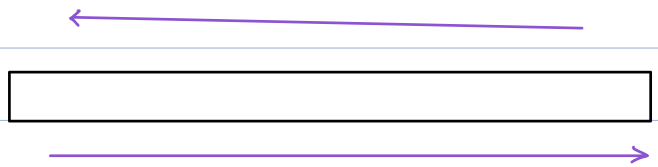
Not a BT

BT

NULL smallest Binary Tree



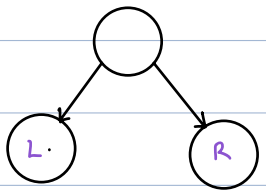
Recursive DS



Traversals in a BT

1) Preorder : Root ⟶ Left subtree ⟶ Right subtree
2) Inorder : LeftSubtree ⟶ Root ⟶ Right subtree
3) Postorder : Left Subtree ⟶ Right Subtree ⟶ Root
4) level order ⟶ Advance [Queue]

Tree Modeling




```
class TreeNode {
    int data;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) {
        data = x;
        left = null;
        right = null;
    }
}
```

Java : null

C++ / C / C# : NULL

Python : None

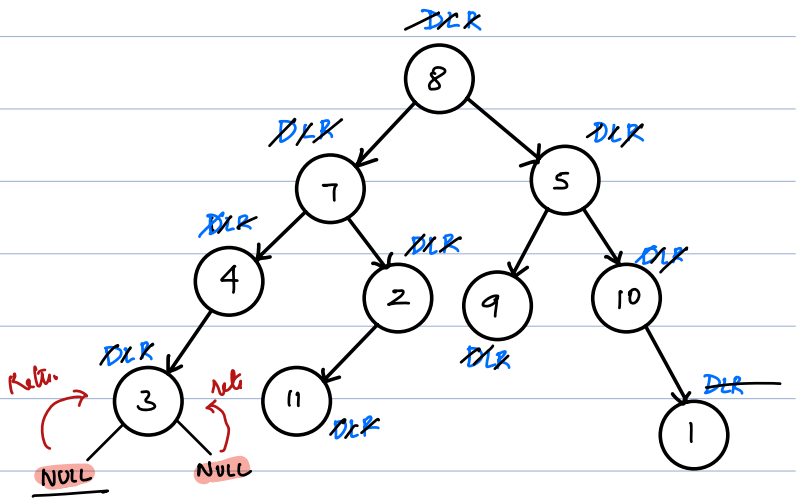
TREE TRAVERSALS

Print root

preorder : DLR

inorder : LDR HW

postorder : LRD

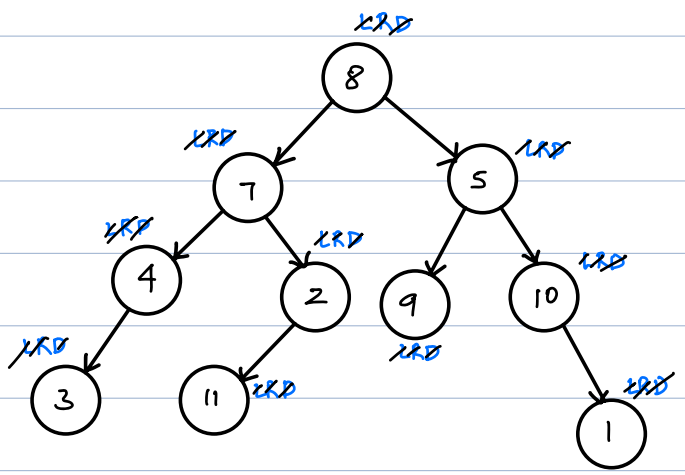


Preorder

Preorder

Print : 8 7 4 3 2 11 5 9 10 1

Post order



Postorder

Print : 3 4 11 2 7 9 1 10 5 8

Break (10:18 - 10:30)

Preorder:

Assumption: preorder(root) will print all the elements of the tree having "root" as its root node in preorder manner

D L R

TC: O(N)

```
void preorder (root) {
    if (root == null) { return }      // Base case
    print (root.data)                 // #D
    preorder (root.left)              // Main logic  #L
    preorder (root.right)             // #R
}
```

```
void inorder (root) {
    if (root == null) { return }
    inorder (root.left)               // #L
    print (root.data)                 // #D
    inorder (root.right)              // #R
}
```

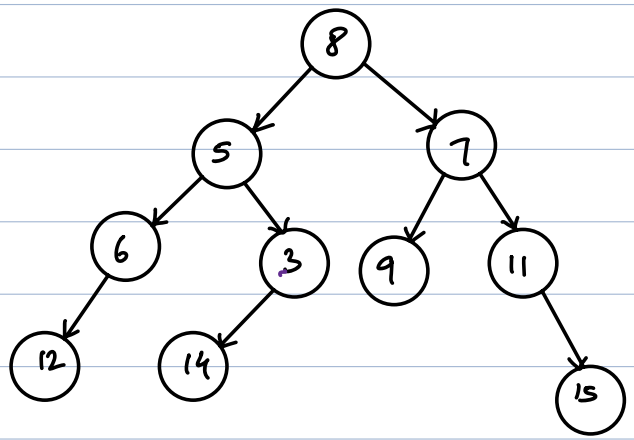
Post order is HW

HW Stack dynamic

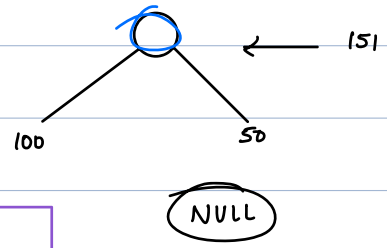
SC:

Calculate size of a tree:

size = 10

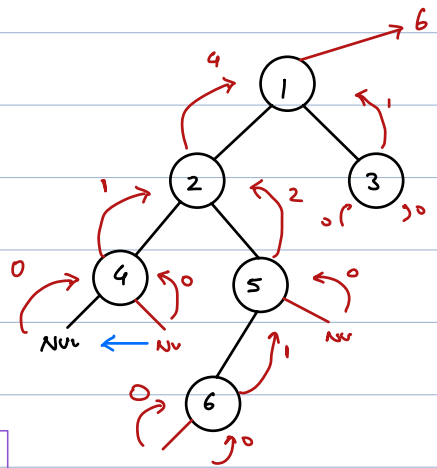


Assumption: size(root) return no. of nodes in tree




```
int size(root) {
    if (root == NULL) { return 0 }
    int lsize = size(root.left)
    int rsize = size(root.right)
    return lsize + rsize + 1
}
```

```
int size(root) {   root = 1
1  if (root == NULL) { return 0 }
2  int lsize = size(root.left)
3  int rsize = size (root.right)
4  return lsize + rsize + 1
}
```




```
int size(root) {   root = 2
1  if (root == NULL) { return 0 }
2  int lsize = size(root.left)
3  int rsize = size (root.right)
4  return lsize + rsize + 1
}
```

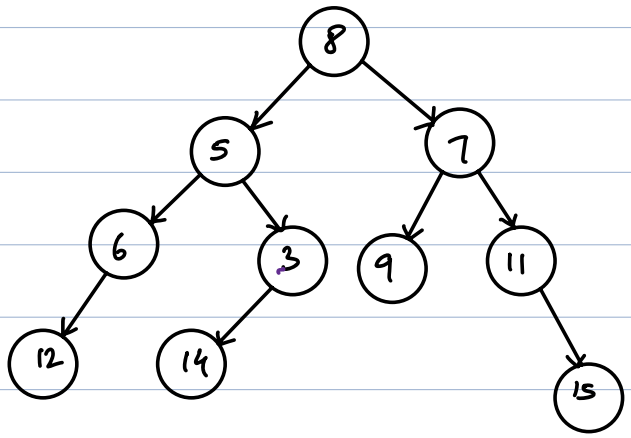
```
size(root) {   root = 5
1  if (root == NULL) { return 0 }
2  int lsize = size(root.left)
3  int rsize = size (root.right)
4  return lsize + rsize + 1
```

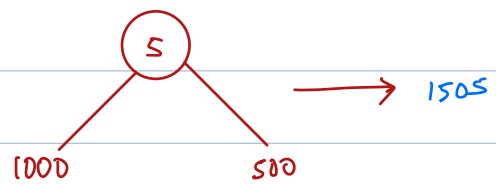
```
nt size(root) {   root = 4
1  if (root == NULL) { return 0 }
2  int lsize = size(root.left)  0
3  int rsize = size (root.right)  0
4  return lsize + rsize + 1
}
```

```
nt size(root) {   root = null
1  if (root == NULL) { return 0 }
2  int lsize = size(root.left)
3  int rsize = size (root.right)
4  return lsize + rsize + 1
}
```

Find sum of all nodes

Ans : 90

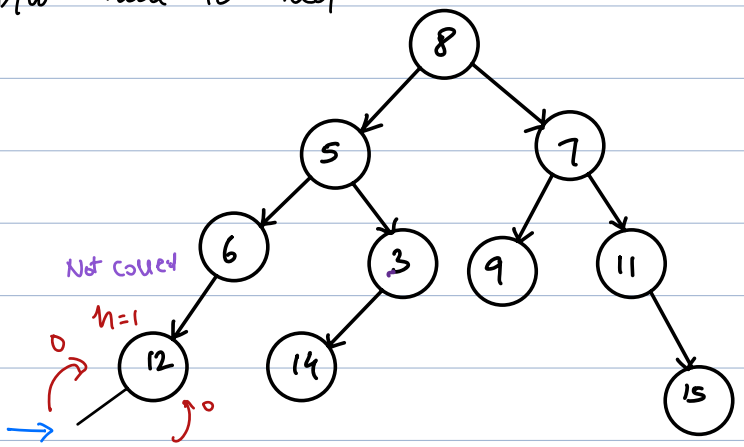





```
int sum(root){
    if (root == NULL) { return 0 }
    int l = sum(root.left)
    int r = sum(root.right)
    return l+r+root.data
}
```

Find height of a tree

Path len b/w node to leaf



height (node) = max (height of its children) + 1

```
int height(root){
    if (root == null){
        return 0
    }
    int lh = height(root.left)
    int rh = height(root.right)
    return max(lh, rh) + 1
}
```

HW

Base case !

Def: No. of nodes b/w root to leaf nd