Tree Basics

Content

- → Trees introduction
- → Naming conventions
- → Tree traversal
- → Basic tree problems

Linear DS

array

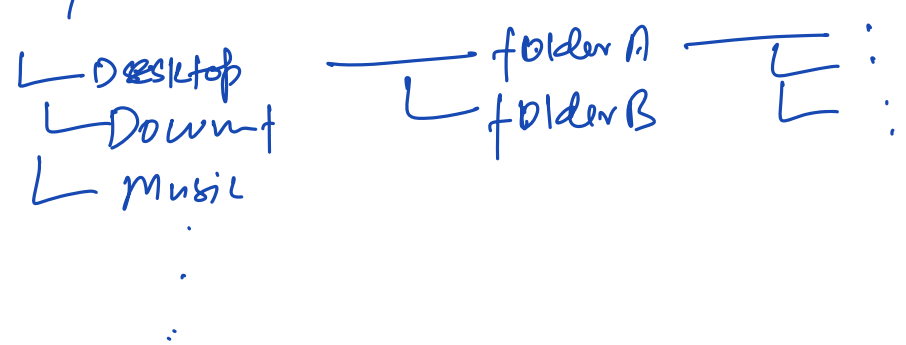
linked list

stack

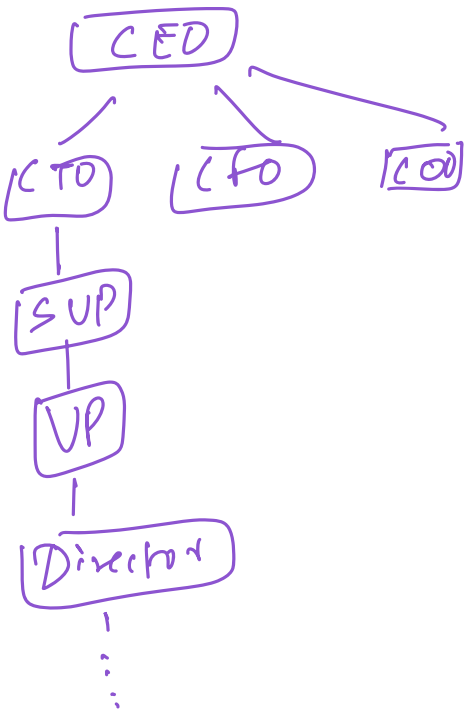
hashmap

Heirarchial Data

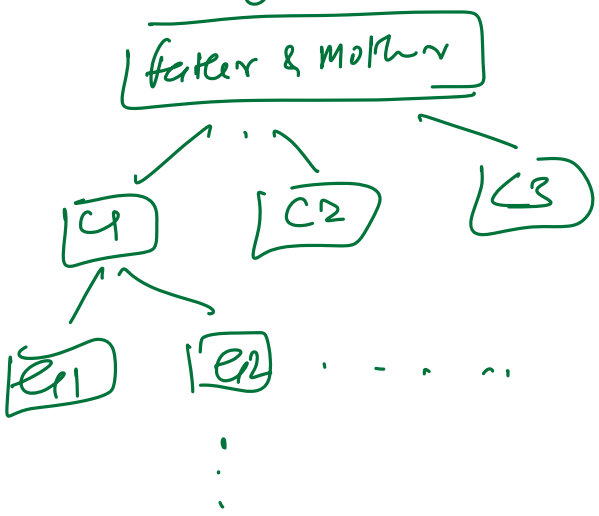
folders & files



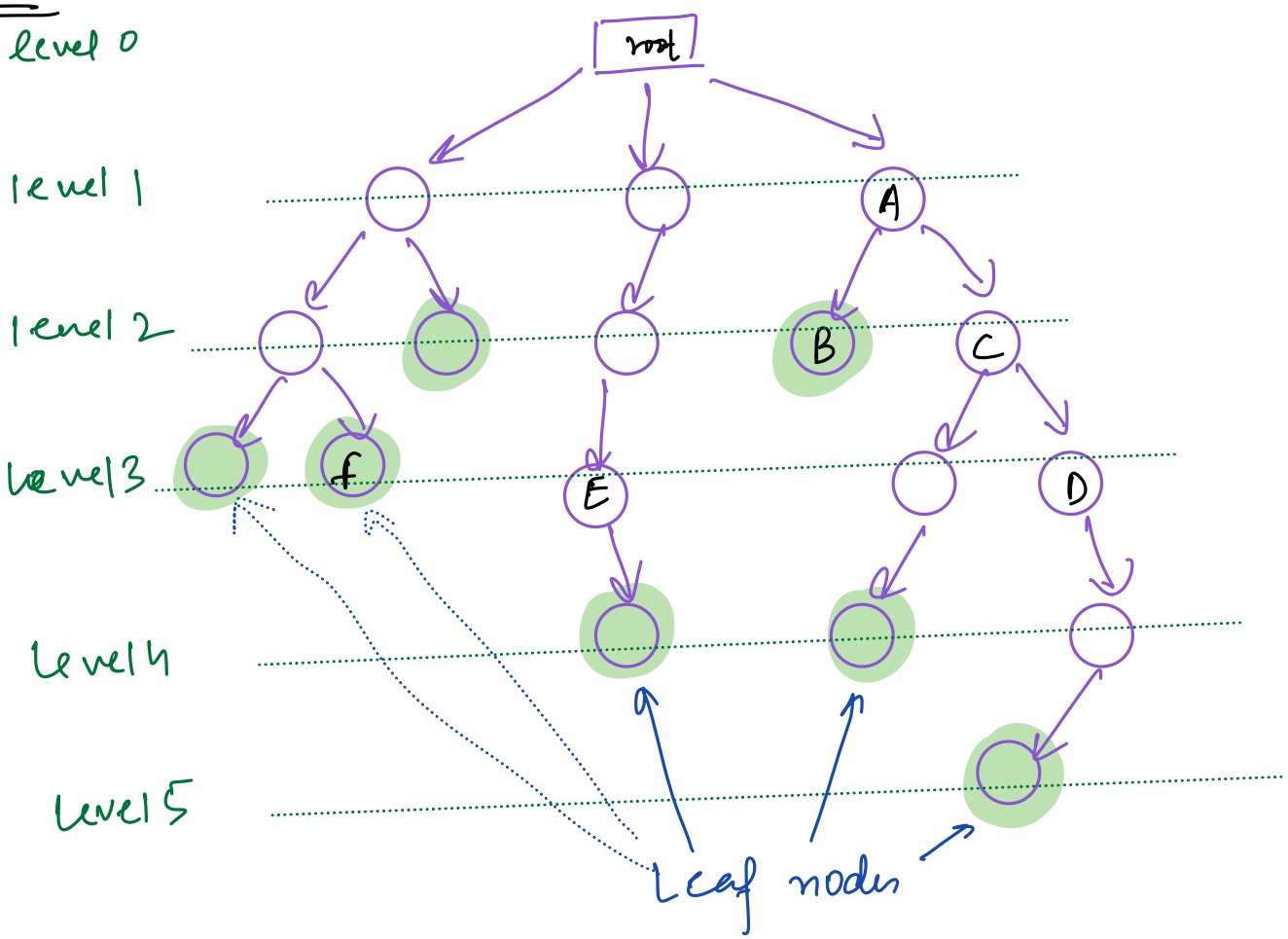
Company Org.



Family Tree



Tree :



Naming

A — B : A is parent of B | B is child of A

A — D : A is ancestor of D | D is descendant of A

B — C : Sibling nodes because they share the same parent.

D, E, F : nodes at the same level

root : node without a parent

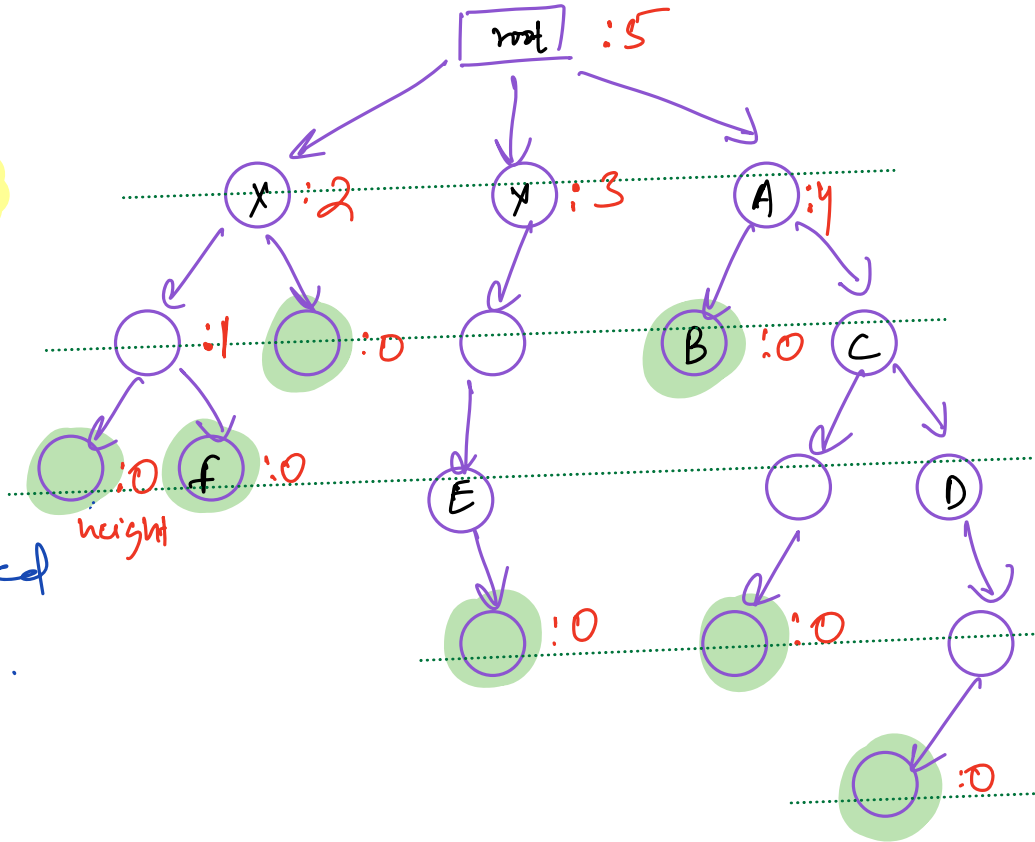
Leaf nodes : nodes without children

what is a tree ?

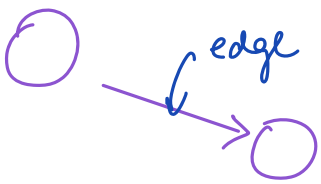
1. tree can have only 1 root node
2. for every node, there is only 1 parent

Height of a node

length of longest path from node to any of its descendant leaf nodes.



Note : Path is calculated based on no. of edges.



Observation 1 :

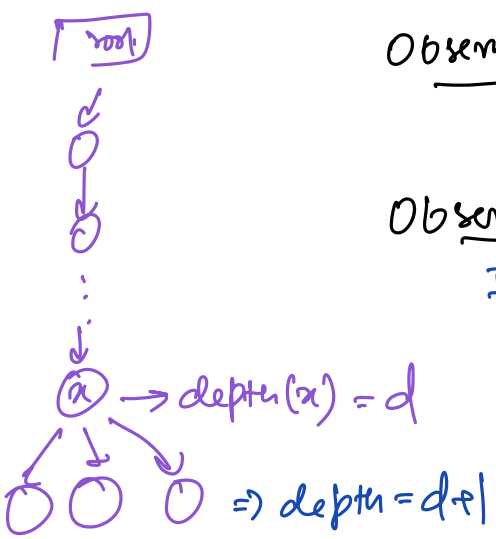
Height(node) = 1 + max(height of its child nodes)

Observation 2 :

Height(leaf node) = 0

Depth of a node

length of path from root to the node.



Observation 1:

Depth(root) = 0

Observation 2:

If depth(node) = d

depth of its child nodes = d+1

Depth of node = level of node

Terminologies

Height (tree) = Height (root node)

Depth (tree) = ~~Depth (root node) = 0~~

max depth of any leaf node

OR

deepest leaf node

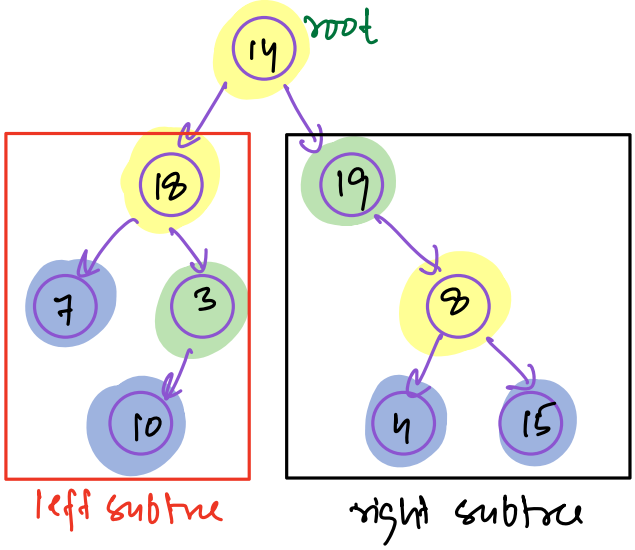
OR

deepest node of the tree

Binary Tree

Tree where every node can have at max 2 children.

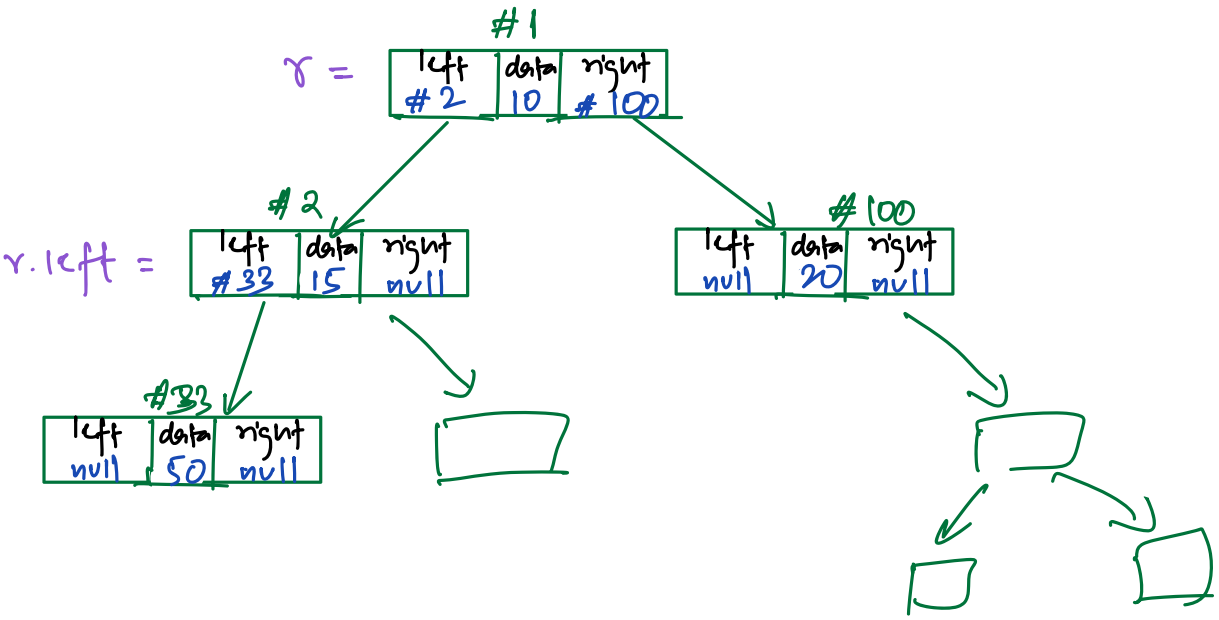
0, 1, 2, ~~3, 4 . . .~~



```
class Node {
    int data;
    Node left; // object reference which can hold address of left child node.
    Node right; // for right child node

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

```
Node r = new Node(10)
r.left = new Node(15)
r.right = new Node(20)
r.left.left = new Node(50)
```



Observation : Given root node we can traverse entire tree.

Tree construction/insertion can be explained by Serialization/De-serialization.
(learn in advance batch)

Note : for all tree problems, tree is already constructed. We are just given the root node.

Tree traversals

- → preorder
- → inorder
- → post-order

→ level order

→ vertical level order

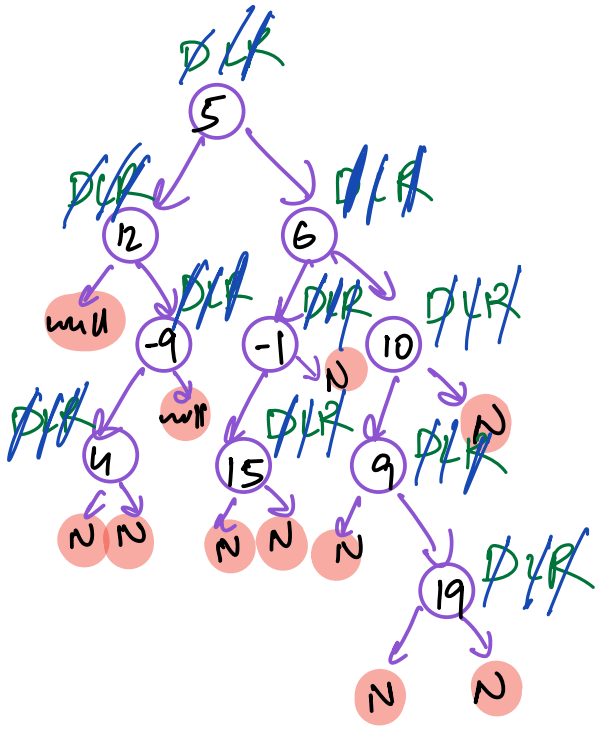
advance batch

Preorder : D (→ data) L (→ left) R (→ right)

Step 1 : print (root.data)

Step 2 : goto left subtree and print entire left subtree in preorder.

Step 3 : goto right subtree and print entire right subtree in preorder.



output :: 5 12 -9 4 6 -1 15 10 9 19

Pseudo Code

```
void preorder(Node r) {
1.  if(r==null) { return; }      → base condition
2.  print(r.data);
3.  preorder(r.left);            → main logic
4.  preorder(r.right);
}
```

TC : O(N) if N = no. of nodes in tree

SC : O(height of tree)
↳ max stack size

height = O(N)

height <= N

height = N-1

Preorder : 1 2 3 4

Inorder : 1 3 2 4

Post order : 1 3 4 2

1 → base
2 → print
3 → goto left
4 → goto right

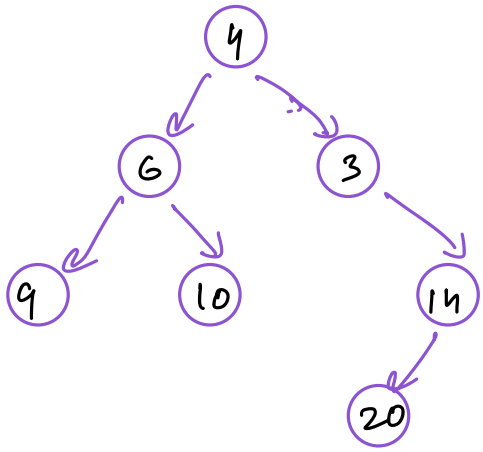
↳ code : TODO

In all anigment questions., use recursion

DLR
preorder : 4 6 9 10 3 14 20

LDR
inorder : 9 6 10 4 3 20 14

LRD
postorder : 9 10 6 20 14 3 4

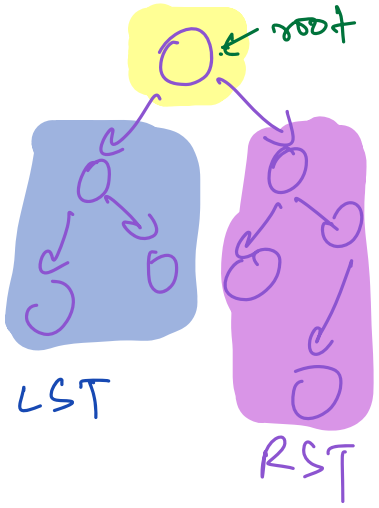


Tree Problems

Solve with recursion & no global variable.

1. Size(Node root) → total nodes
2. Sum(Node root) → total sum of all nodes
3. Height(Node root) → height of node

1. size(root) = size(LST) + size(RST) + 1




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

2. sum(root) = sum of LST + sum of RST + root.data

```
int sum(Node n){
  if(n== null) { return 0; }
  l = sum(n.left)
  r = sum(n.right)
  return l+r+n.data ;
}
```

3. height(root) = max(height of LST, height of RST) + 1

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