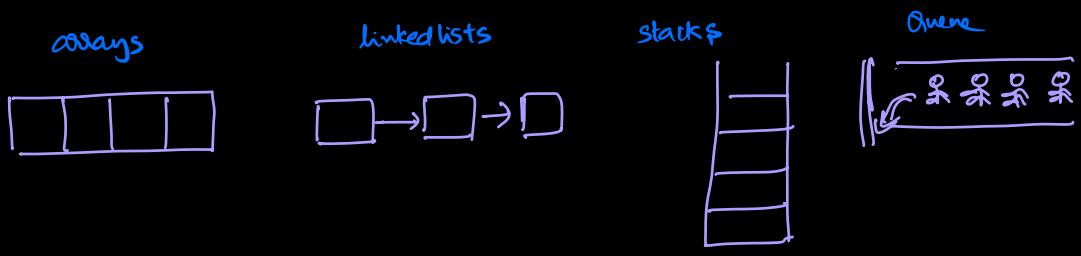
Today's content.

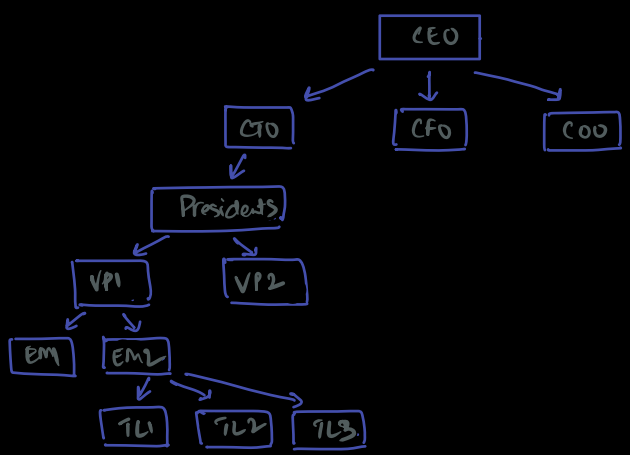
-> Trees introduction
-> Naming convention
-> Tree traversals
-> Basic tree problems.
-> first non-repeating char from queues.

linear:

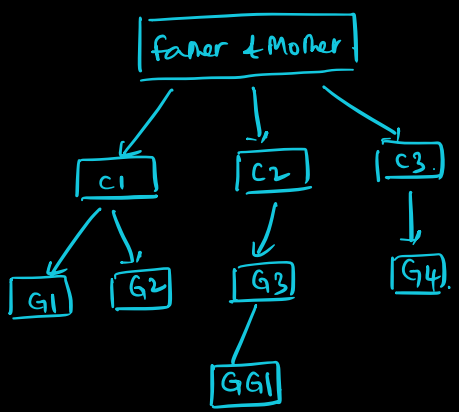



Heirarchical data.

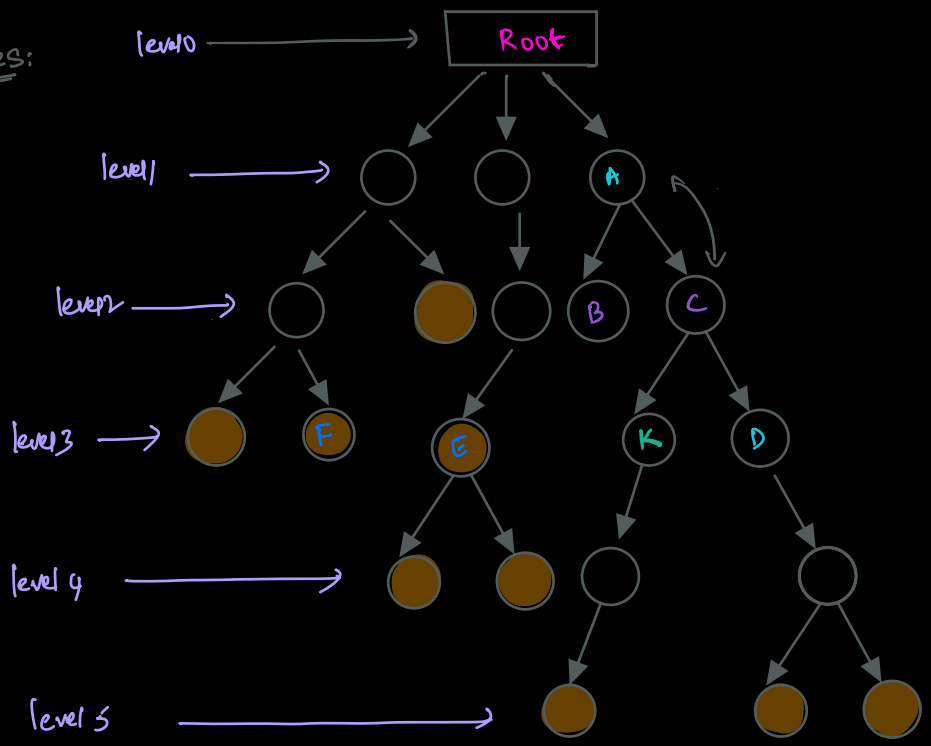
Ex: Company organization.




Ex: Family Tree.



Trees:




Relations: Naming conventions.

A & D -> A is ancestor of D or D is descendent of A.

B & C -> Sibling nodes, share same parent.

F, E, D -> Nodes at same level.

Root -> Node with no parent.

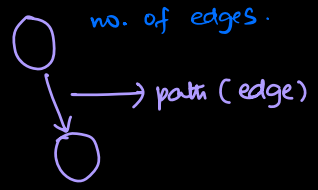
leaf -> Node with no children.

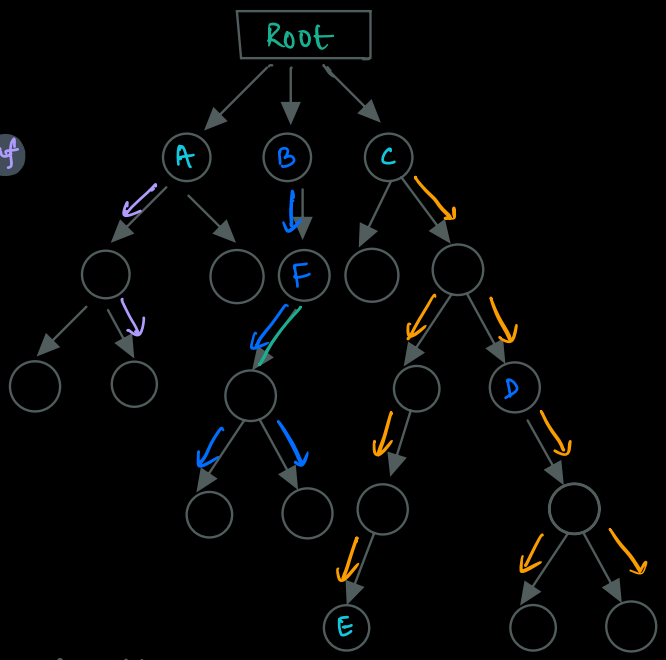
Tree ->
- It should have only 1 root node
- very node must have single parent

## Height (node).

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

Path is calculated based on no. of edges.




Ex:

H(A) = 2

H(B) = 3

H(C) = 4

H(E) = 0

Observation:

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

(ii) H(leaf node) = 0.

H(Root) = Height of tree.

## depth of a node.

length of path from root to the node.

d(A) = 1

d(F) = 2

d(E) = 5

d(D) = 3

Observation:

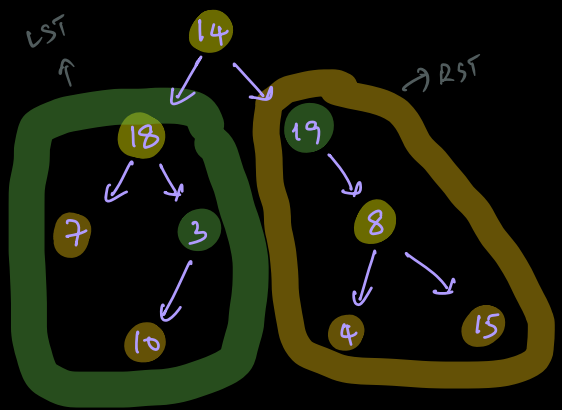
(i) depth (root) = 0.

(ii) If depth of node = d,
Then depth of its child nodes = d+1.

Our learning is limited to binary trees.

Binary trees. : Every node must have at the max 2 children

0, 1, 2 ✓ , 3, 4, 5 ✗




nodes with 1 child : 19, 3

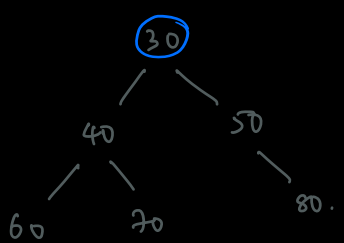
nodes with 0 child : 7, 10, 4, 15

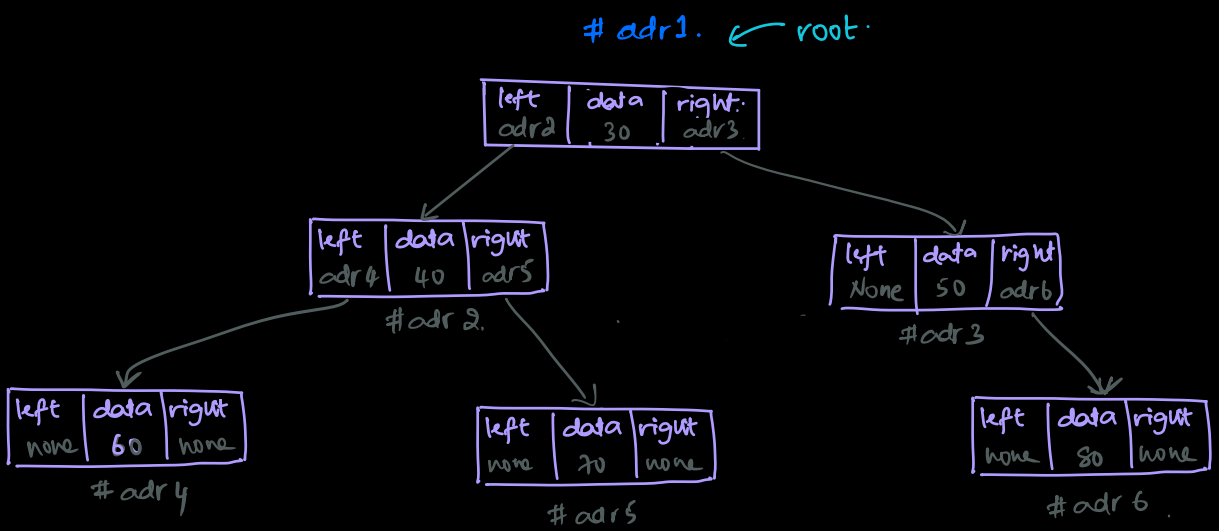
nodes with 2 child : 14, 18, 8

Structure of binary tree nodes.

```
class Node
{
    int data;
    Node left;
    Node right;
}
```

Tree.

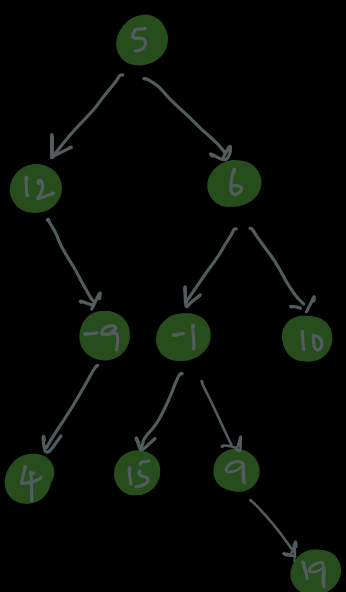





Seriali~~z~~ation & deseriali~~z~~ation → will discuss.

Tree traversals.

today: Pre-order, In-order, Post-order | level order, diagonal, ! ——— Adv.




Preorder traversal : [Data] [LST] [RST].

Step1: print (root.data).

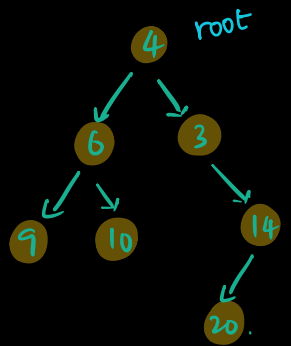
Step2: goto left subtree, & print entire LST in preorder traversal way.

Step 3: goto right subtree & print entire RST in preorder traversal way.

[5 12 -9 4 6 -1 15 9 19 10]

Preorder traversal: D, L, R.

[4, 6, 9, 10, 3, 14, 20]




inorder : L, D, R. : [9, 6, 10, 4, 3, 20, 14]

postorder : L, R, D : [TO-DO].

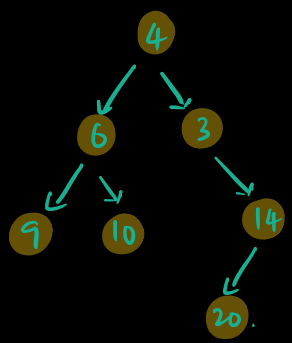
pseudocode: Preorder: D L R.

[4, 6, 9, 10, 3, 14, 20].

```
void preOrder(Node root)
{
    1. if(root is None) return.
    2. print(root.data)
    3. preOrder(root.left)
    4. preOrder(root.right)
}
```

TC: O(N).

SC: O(H).



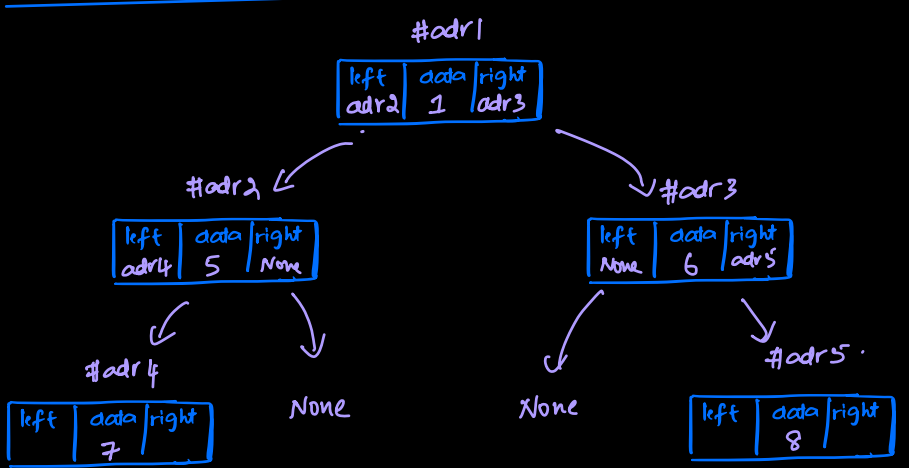

Preorder: 1, 2, 3, 4 (D L R)

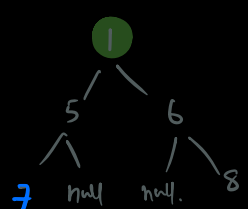
Inorder: 1, 3, 2, 4 (L D R)

Postorder: 1, 3, 4, 2 (L R D)

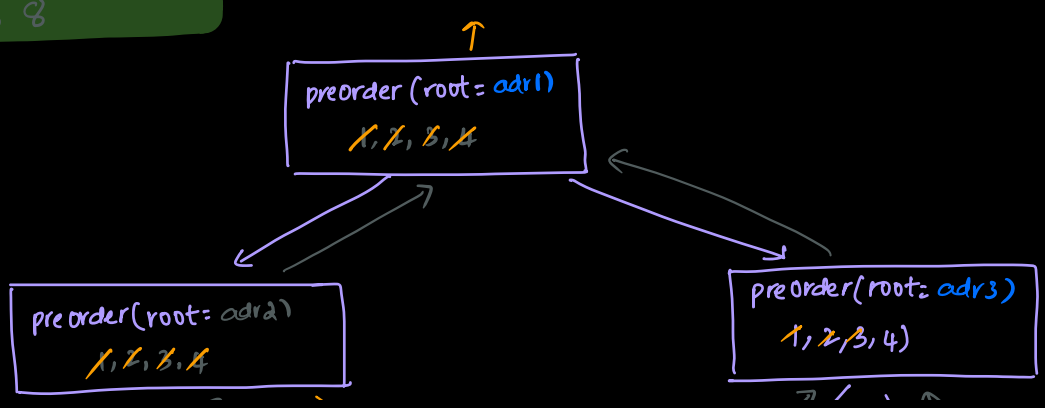
TODO.

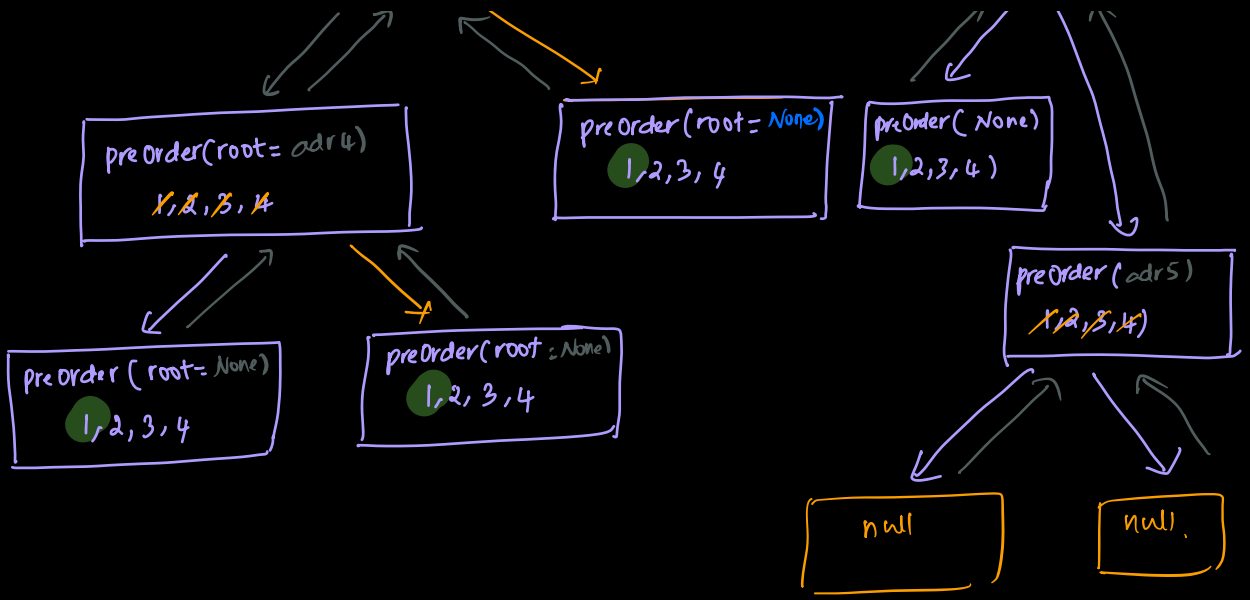
```
def preOrder(Node root)
{
    1. if(root is None) return.
    2. print(root.data)
    3. preOrder(root.left)
    4. preOrder(root.right)
}
```




1 5 7 6 8



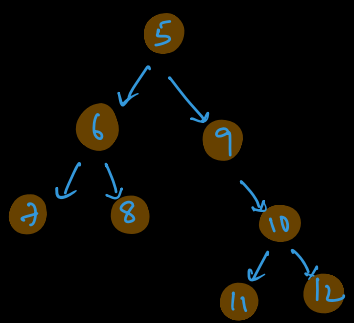



Trees Problems.

// All tree problems, solve them with recursion.

a) Size of the tree: How many elements are present in tree.

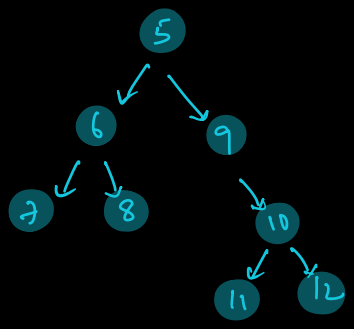
ans: 8.




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

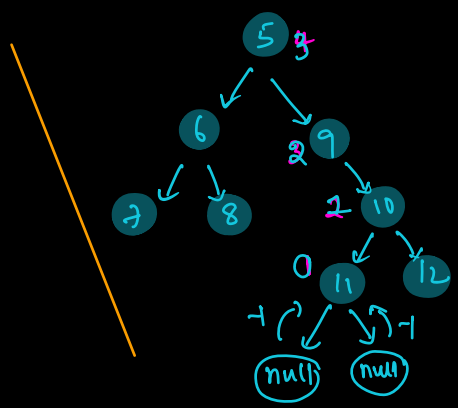
b) Return sum of all nodes.

```
int Sum(root)
{
    if(root=null)
        return 0.
    return root.data + sum(root.left)
                     +sum(root.right)
}
```



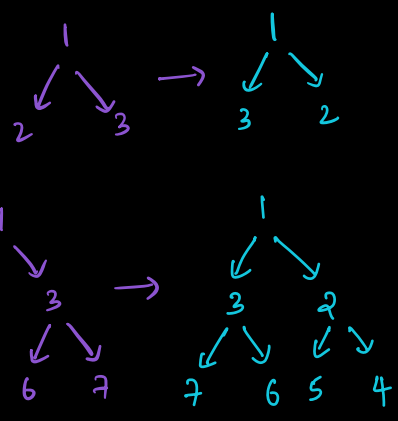

c) Height of tree. Height of node = 1+ max(height of LST, height of RST).

```
int height(root)
{
    if(root==null)
        return -1 // returning 0 doesn't work.
    return 1+ max(height(root.left),
                  height(root.right))
}
```




d) Invert binary tree.

```
Node invert(Node root)
{
    if(root==null)
        return null
    Node temp = root.left
    root.left = invert(root.right)
    root.right = invert(temp)
    return root
}
```



Q1. First non repeating character.

Ex: A = a b a b d c
B = a a b # d d

A = a b a d b c
B = a a b b d d

A = a b c a b c
B = a a a b c #

idea: HashMap<Char, Integer> hm

a
a b
a b b
a b b a
a b b a d
a b b a d c
[ a a a # d d]

| a:2 | c:1 |
|---|---|
| b:2 | |
| d:1 | |

queue.

~~a~~ ~~b~~ ~~b~~ ~~a~~ d c

O/p: [a a a # d d]

Steps:

Iterate over string

(i) add every character to queue & inc freq in hm.

(ii) while (hm[q.front()>1])
    q.pop()

(iii) Check if 'q' is empty !

Yes → append '#'.

No → append q.front().

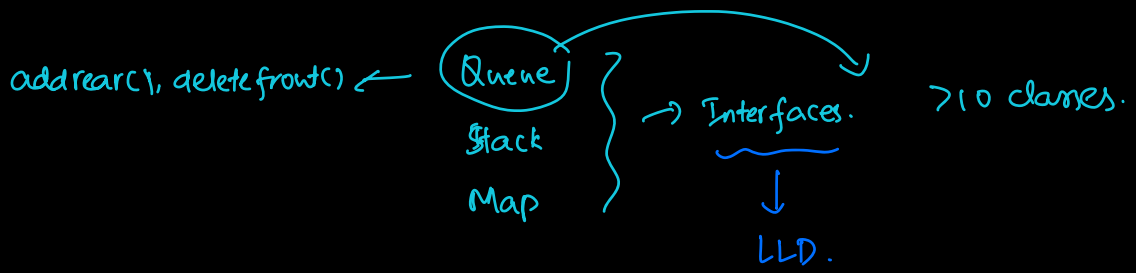
```
String firstNonrepeating(String A)
    Map<character, Integer> hm;
    Queue<character> queue;
    String ans=" "; // Builder.
    for(i=0; i<A.length(); i++)
        queue.add(A.charAt(i))
        if(hm.containsKey(A.charAt(i))) // A[i] in hm.
            hm.put(A.charAt(i), 1+hm.get(A.charAt(i)),
            // hm(A[i]) = hm(A[i]) +1.
        else
            hm.put(A.charAt(i), 1)
        While(q.size>0 && hm.get(q.front())>1)
            q.pop()
        if(q.size>0)
            ans = ans + q.front()
        else
            ans = ans + "#"
```

Step 1.

Step 2

Step 3

TC: O(N)
SC: O(N)




For asking doubts.

* Related to todays content.
  - (1-2) raise it first
  - entire explanation -> last.
* Related to previous classes.
* Not related to content
  - Avoid this in class.