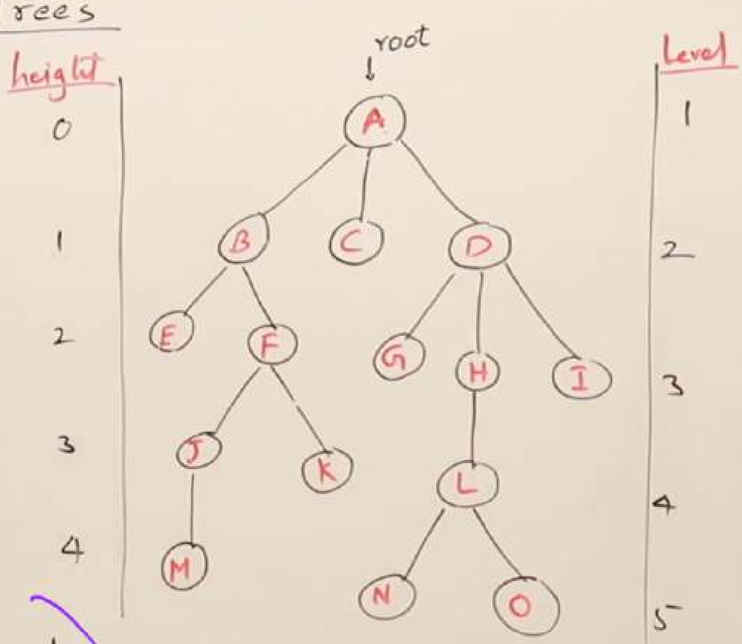
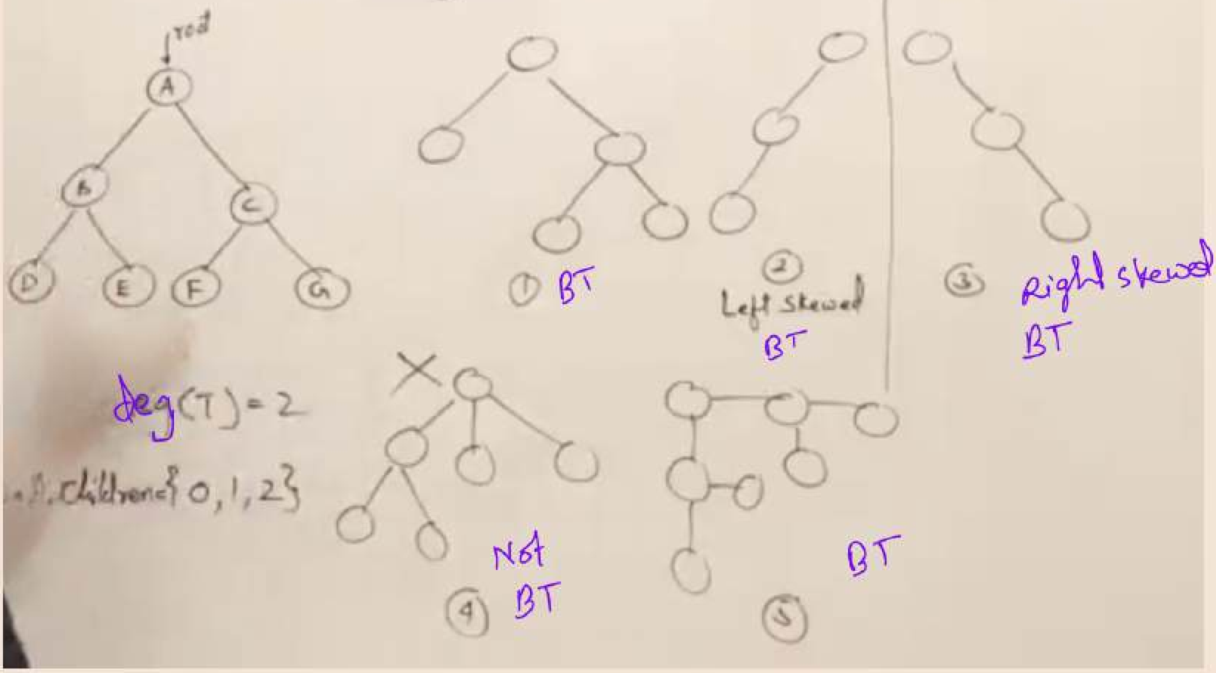
Trees

Terminology

1. Root
2. Parent
3. Child
4. Siblings
5. Descendents
6. Ancestors
7. Degree of a node → degree of node is the number of child it is having
   ↳ max degree of node in a tree called, degree of tree
8. Internal / External nodes → External nodes which does not have any child node (eg. leaf node)
9. Levels → starts from 1
10. Height → starts from 0
11. Forest → collection of trees



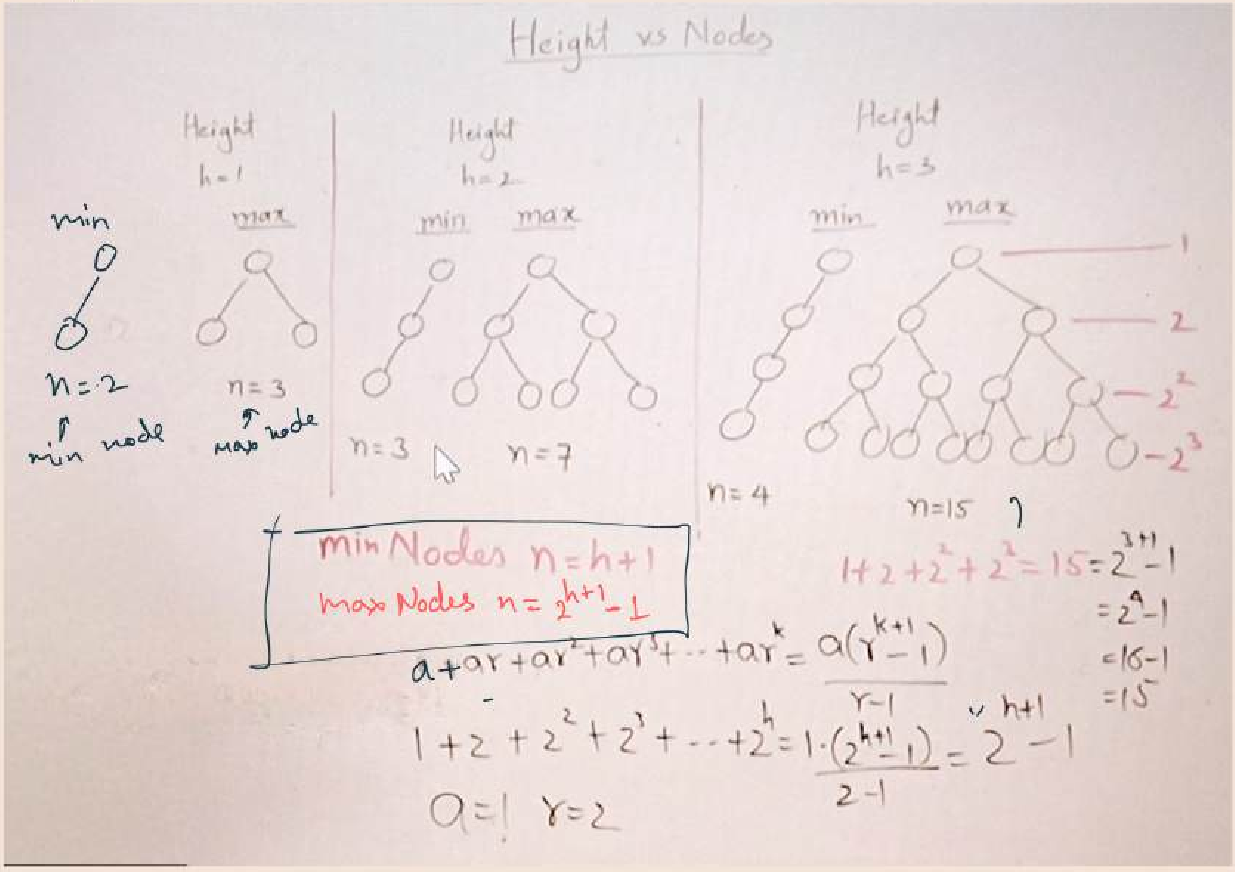
Binary Tree

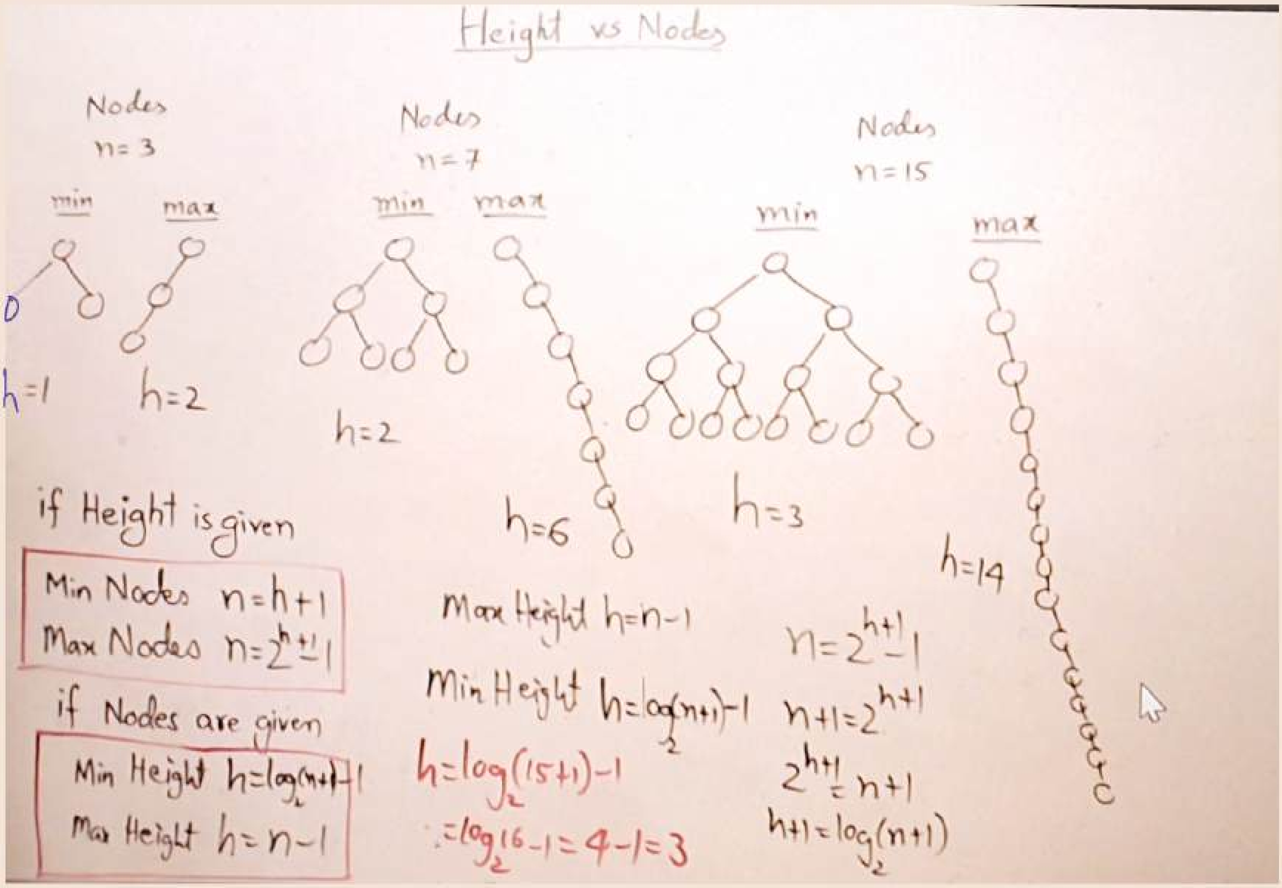


deg(T) = 2

Children {0, 1, 2}

Height vs Nodes :



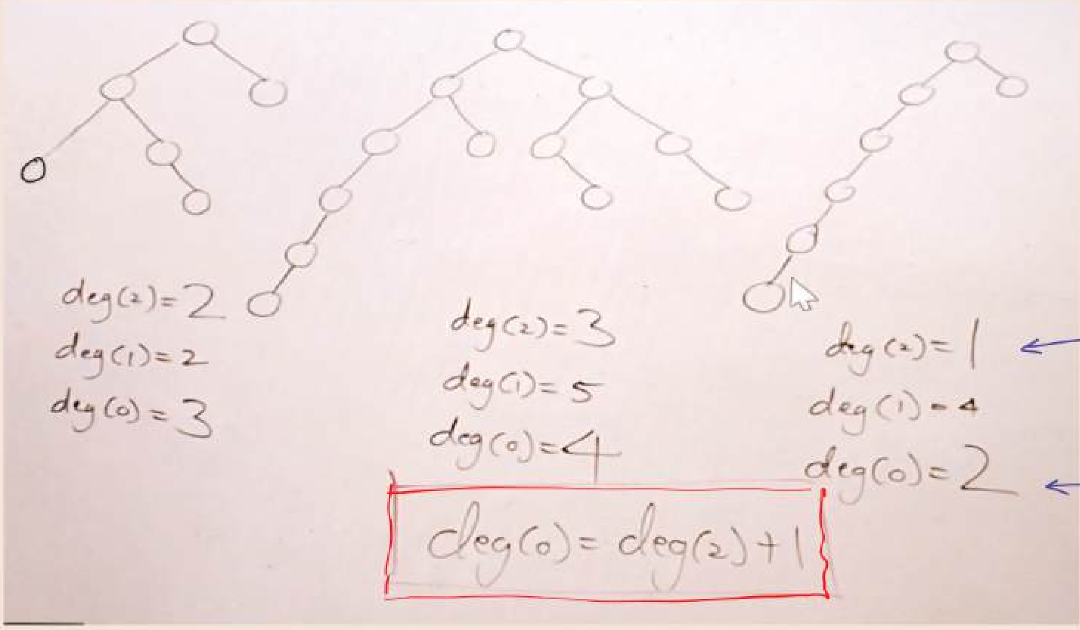


Height of binary tree :

$$\log_2(n+1)-1 \le h \le n-1$$

Number of Nodes in a binary tree :

$$h+1 \le n \le 2^{h+1}-1$$

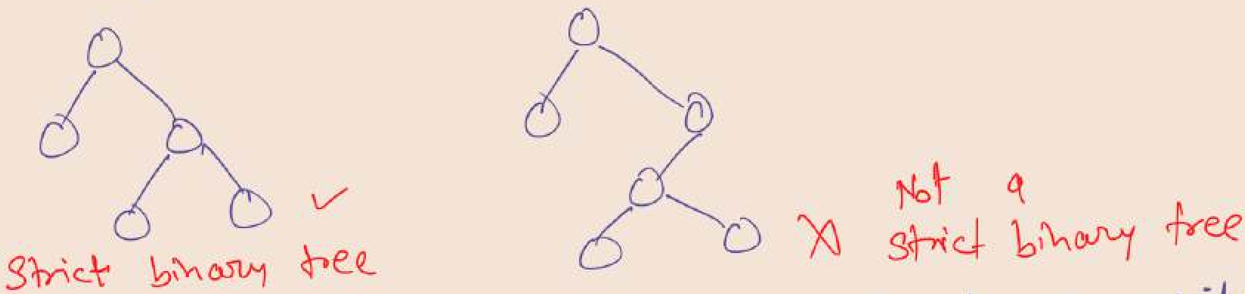


degree is How many Nodes are connected to a node

internal node

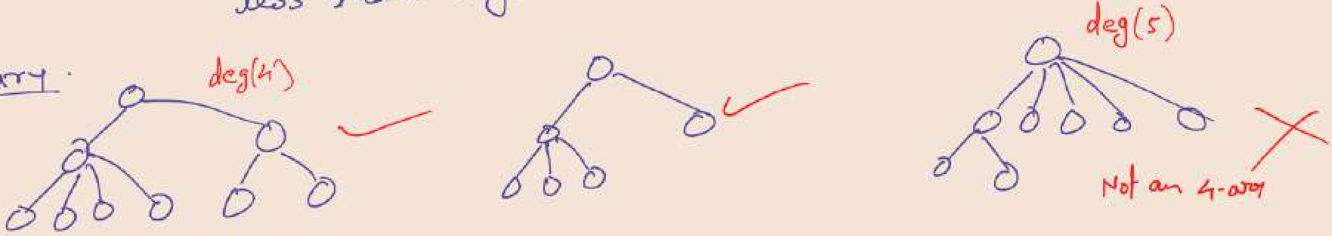
external node

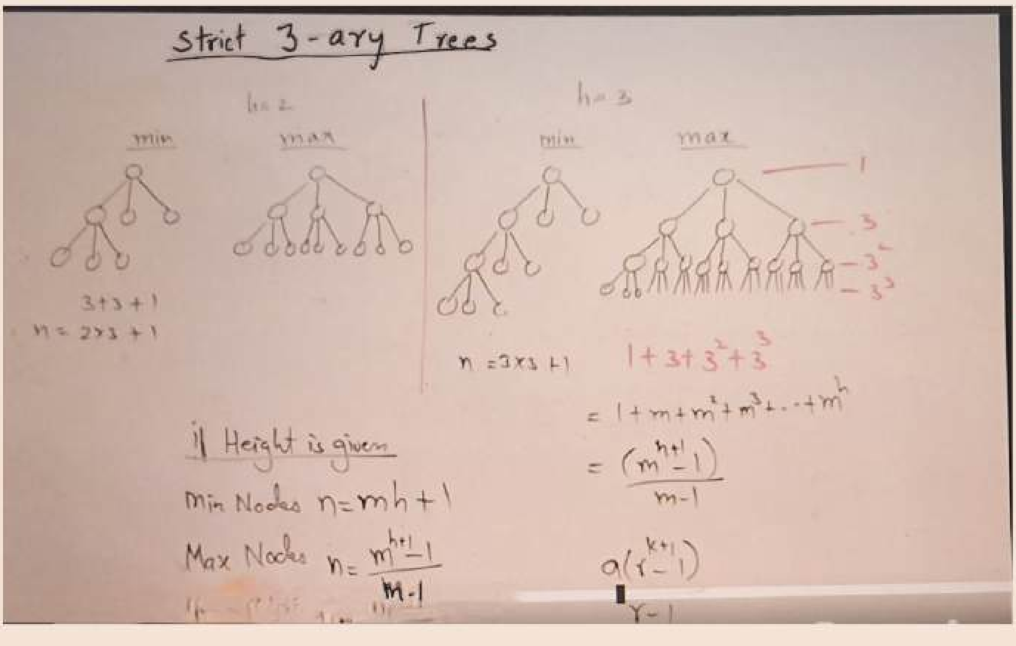
⇒ Strict binary tree : A tree which has degree 2 & deg 0 Nodes only called as strict binary tree.



⇒ m-ary tree / n-ary tree : m-ary tree is a tree which has any of its node less than degree m.

4 - ary.





if 'n' Nodes are given :

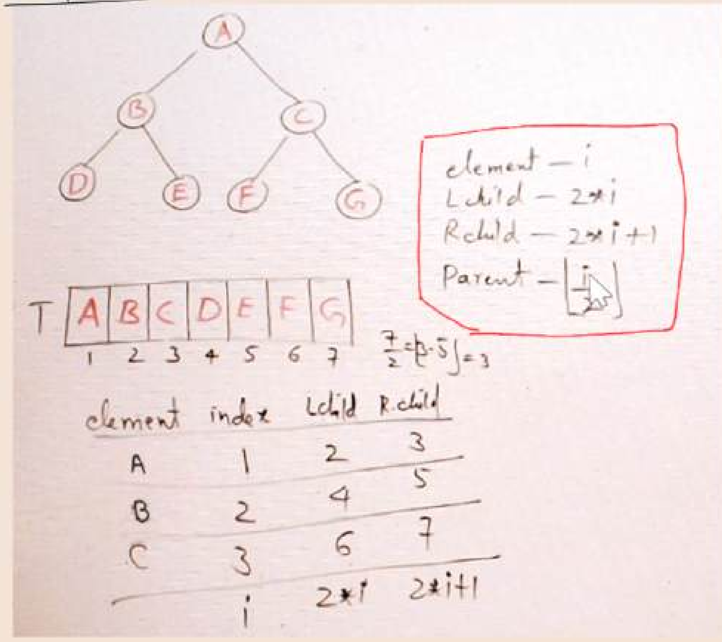
min Height $h = \log_m [n(m-1)+1] - 1$

Max Height $h = \frac{n-1}{m}$

→ lets i are internal nodes in strict m-ary tree then external nodes will be

$$\boxed{e = (m-1)i + 1}$$

Representation of Binary tree :-



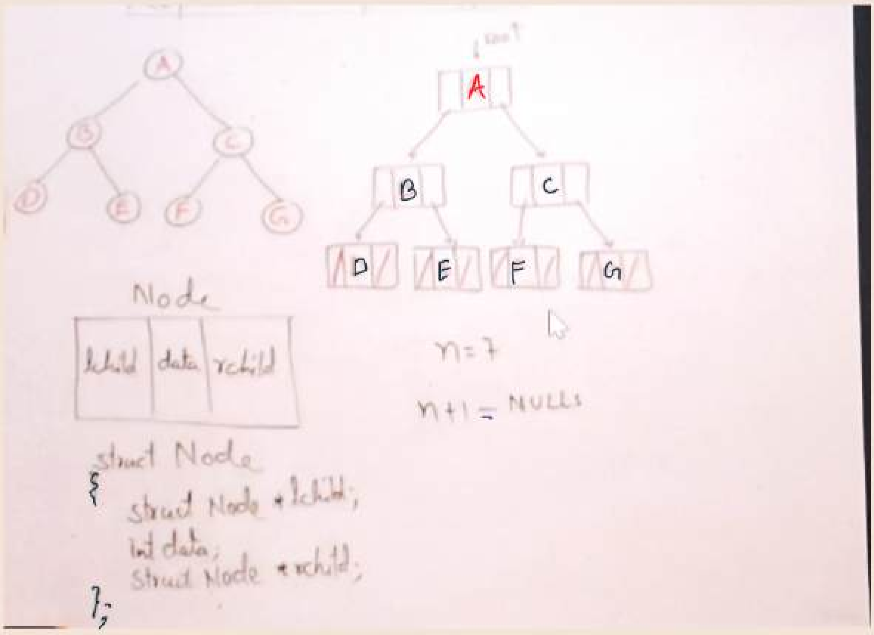
Array Representation of Binary tree

element — i

L child — $2 \times i$

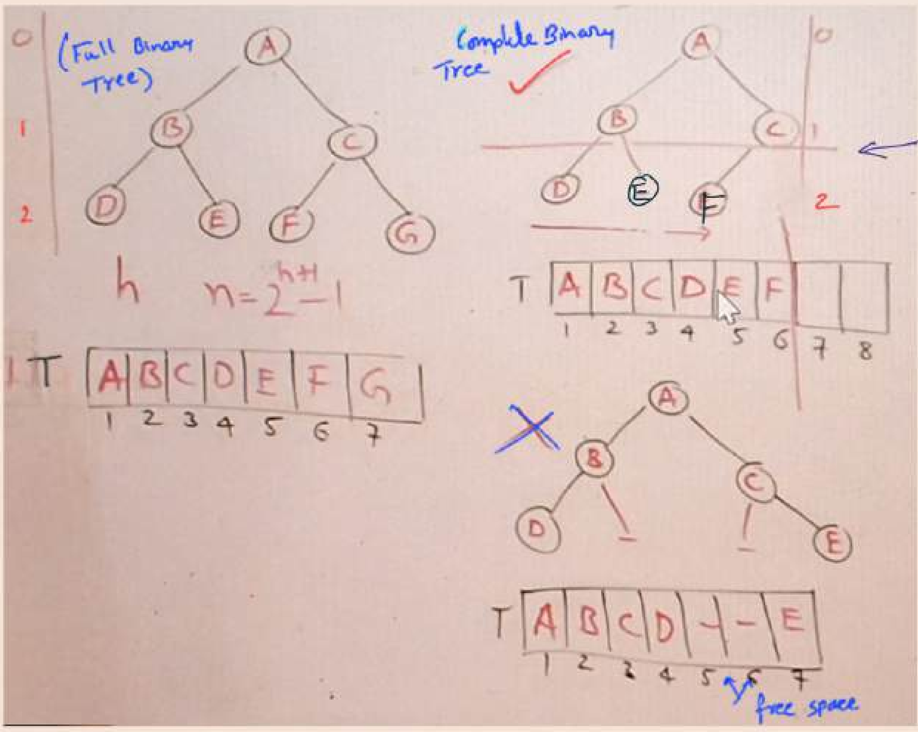
R child — $2 \times i + 1$

Parent — $\lfloor i/2 \rfloor$



Linked Representation of a Tree.

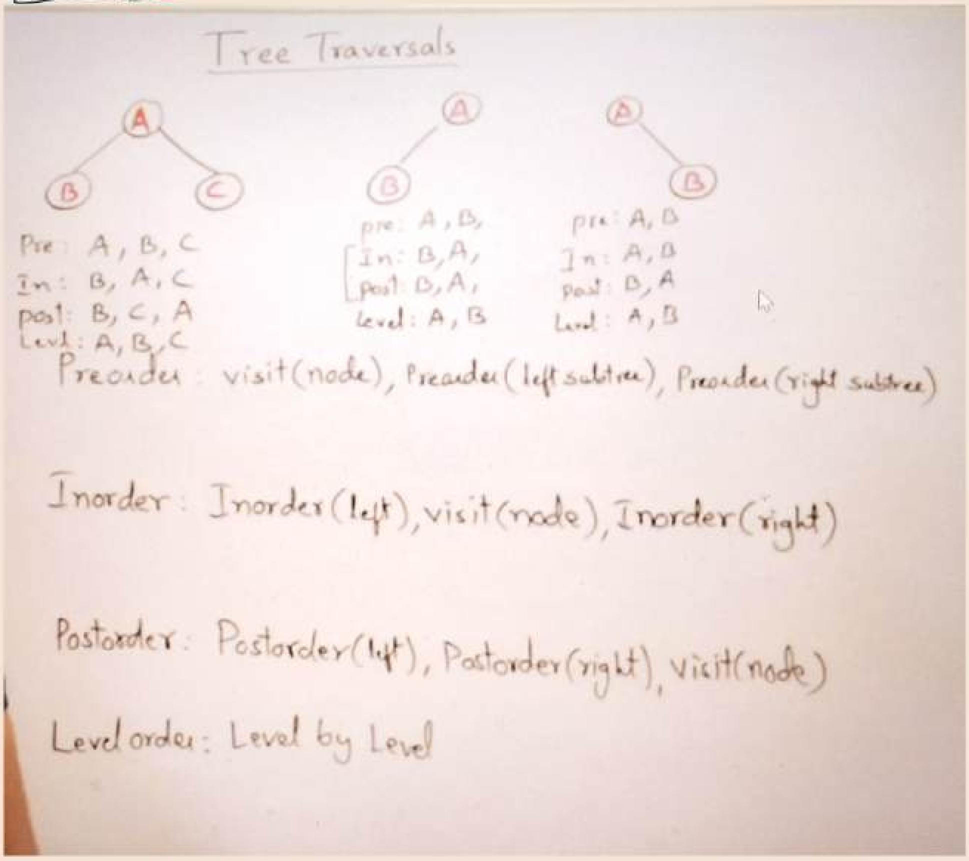
⇒ full vs complete binary tree :-

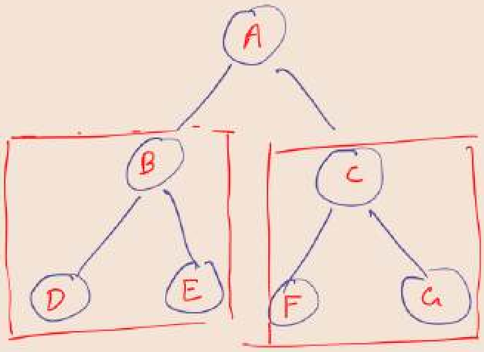


A complete Binary tree is full binary tree till h-1 height and after that filled from left to Right

A full binary tree is a complete tree, but a complete binary tree need not to be a full tree

⇒ Tree Traversal :





Pre : A, (B, D, E), (C, F, G)

A, B, D, E, C, F, G

In : (D, B, E), A, (F, C, G)

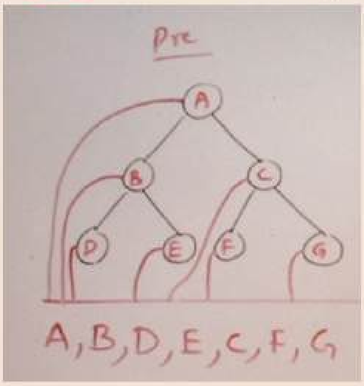
D, B, E, A, F, C, G

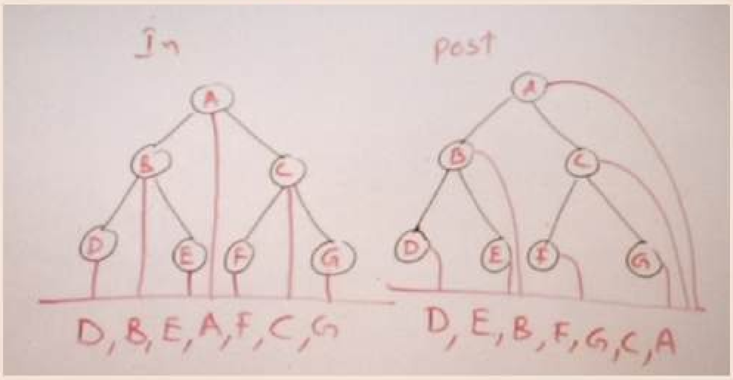
Post : (D, B, E), (F, G, C), A

D, B, E, F, G, C, A

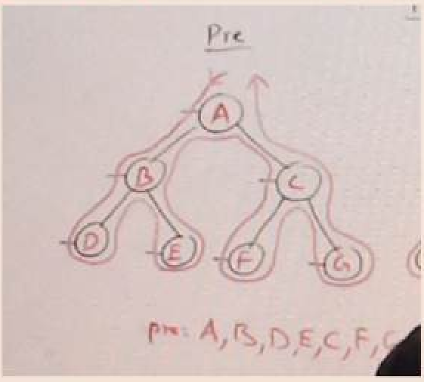
Level : A, B, C, D, E, F, G

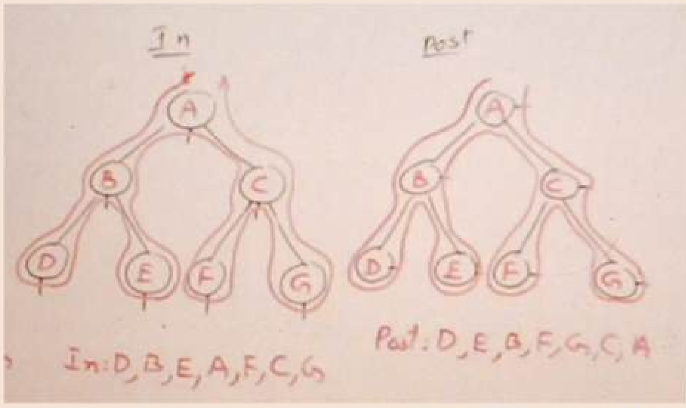
⇒ Easy methods to find tree Traversal :





⇒ Second method :





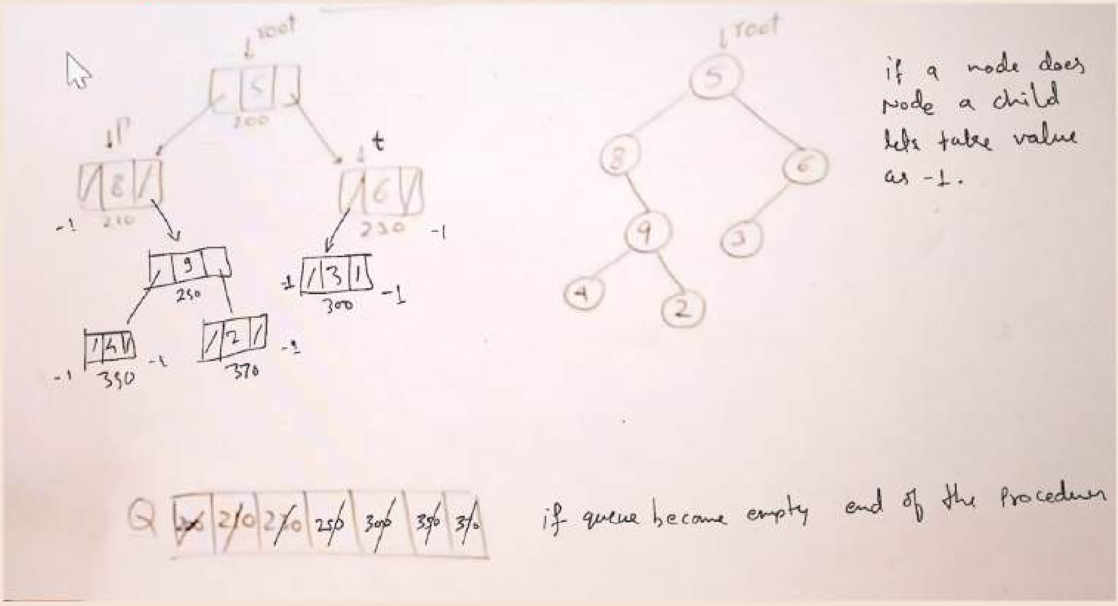
⇒ third method : use one finger and Point at node.

Preorder ⇒ Point from left to Right and go over all the nodes for whichever node its pointing consider that node in sequence

Inorder ⇒ use finger direction from bottom to top

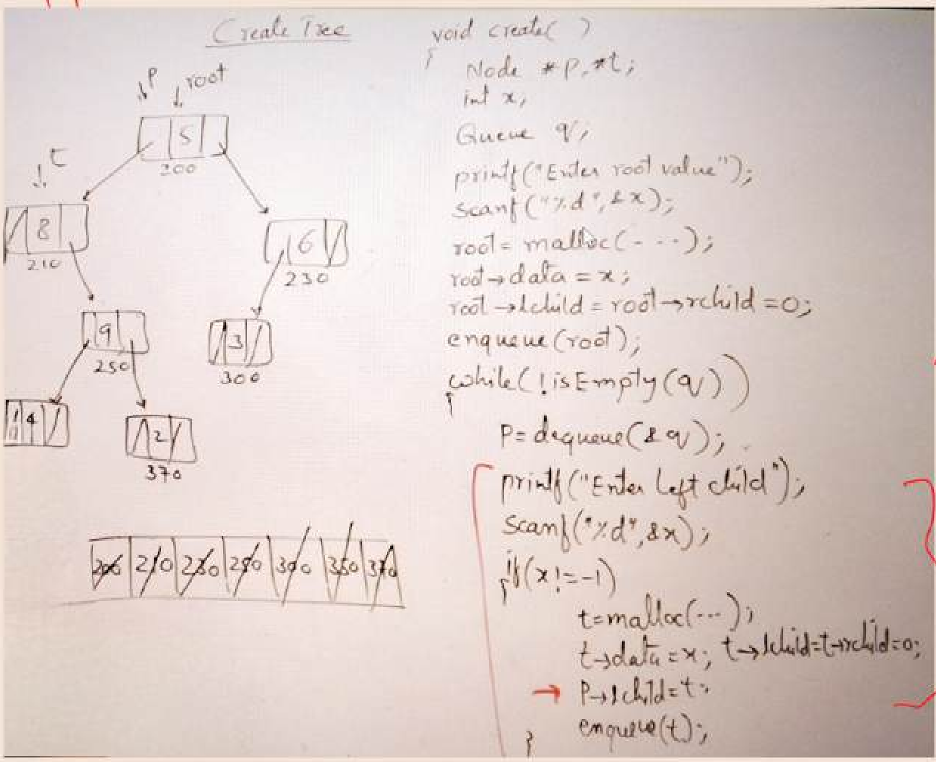
Post order ⇒ use finger direction from right to left.

Creating Binary tree :-



285

Program to create binary tree :-



Same code just some modification will be used for creating Right child.

1

```
class Node{
public:
    Node* lchild;
    int data;
    Node* rchild;
};

class Queue{
private:
    int size;
    int front;
    int rear;
    Node** Q;   // [Node*]*: Pointer to [Pointer to Node]
public:
    Queue(int size);
    ~Queue();
    bool isFull();
    bool isEmpty();
    void enqueue(Node* x);
    Node* dequeue();
};

Queue::Queue(int size) {
    this->size = size;
    front = -1;
    rear = -1;
    Q = new Node* [size];
}

Queue::~Queue() {
    delete [] Q;
}

bool Queue::isEmpty() {
    if (front == rear){
        return true;
    }
    return false;
}

bool Queue::isFull() {
    if (rear == size-1){
        return true;
    }
    return false;
}

void Queue::enqueue(Node* x) {
    if (isFull()){
        cout << "Queue Overflow" << endl;
```

{for malloc function stdlib.h library need to import}

2

(create queue)

```
    } else {
        rear++;
        Q[rear] = x;
    }
}

Node* Queue::dequeue() {
    Node* x = nullptr;
    if (isEmpty()){
        cout << "Queue Underflow" << endl;
    } else {
        front++;
        x = Q[front];
    }
    return x;
}

Node* root = new Node;
```

(create tree by inserting elements one by one.)

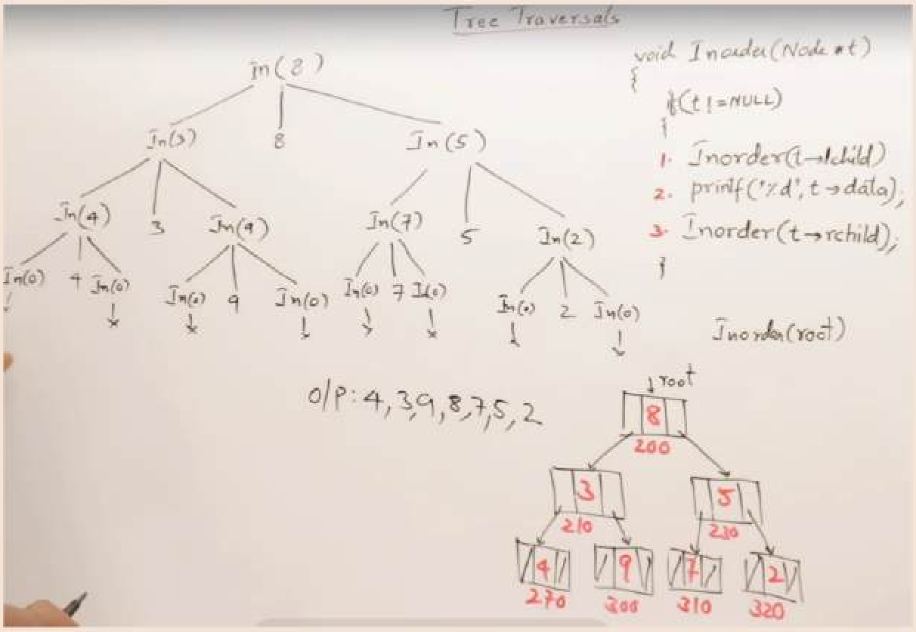
3

```
void createTree(){
    Node* p;
    Node* t;
    int x;
    Queue q(10);

    cout << "Enter root value: " << flush;
    cin >> x;
    root->data = x;
    root->lchild = nullptr;
    root->rchild = nullptr;
    q.enqueue(root);

    while (! q.isEmpty()){
        p = q.dequeue();

        cout << "Enter left child value of " << p->data << ": " <<
flush;
        cin >> x;
        if (x != -1){
            t = new Node;
            t->data = x;
            t->lchild = nullptr;
            t->rchild = nullptr;
            p->lchild = t;
            q.enqueue(t);
        }

        cout << "Enter left child value of " << p->data << ": " <<
flush;
        cin >> x;
        if (x != -1){
            t = new Node;
            t->data = x;
            t->lchild = nullptr;
```

4

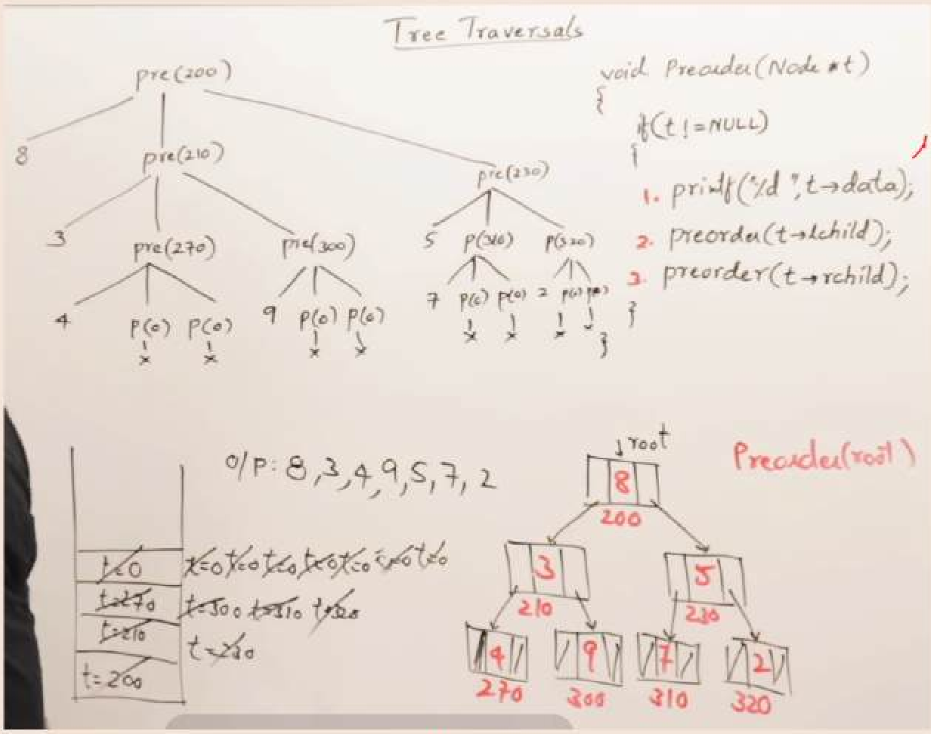
```
            t->rchild = nullptr;
            p->rchild = t;
            q.enqueue(t);
        }
    }
}

void preorder(Node* p){
    if (p){
        cout << p->data << ", " << flush;
        preorder(p->lchild);
        preorder(p->rchild);
    }
}

void inorder(Node* p){
    if (p){
        inorder(p->lchild);
        cout << p->data << ", " << flush;
        inorder(p->rchild);
    }
}

void postorder(Node* p){
    if (p){
        postorder(p->lchild);
        postorder(p->rchild);
        cout << p->data << ", " << flush;
    }
}


int main() {

    createTree();

    preorder(root);
    cout << endl;

    inorder(root);
    cout << endl;

    postorder(root);
    cout << endl;

    return 0;
}
```

Tree Traversal : (In order tree traversal) :

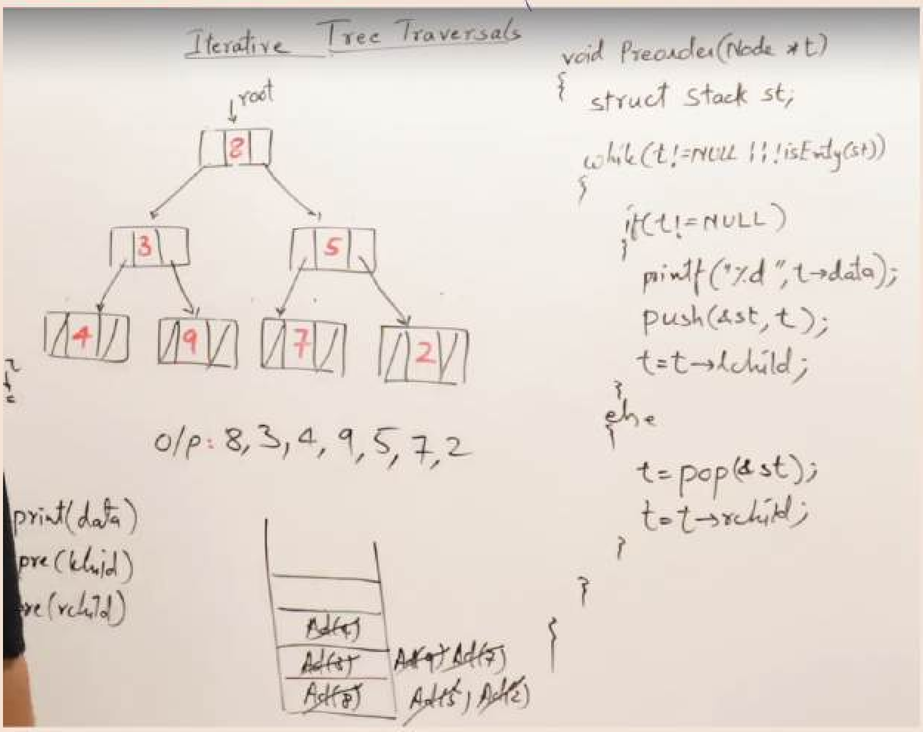


- This is a common tree traversal, it uses the recursion to traverse throgh three nodes.
- where once the left child is traversed then print the nodes value and after that travese through the right child of the current node.

Preorder tree traversal :-

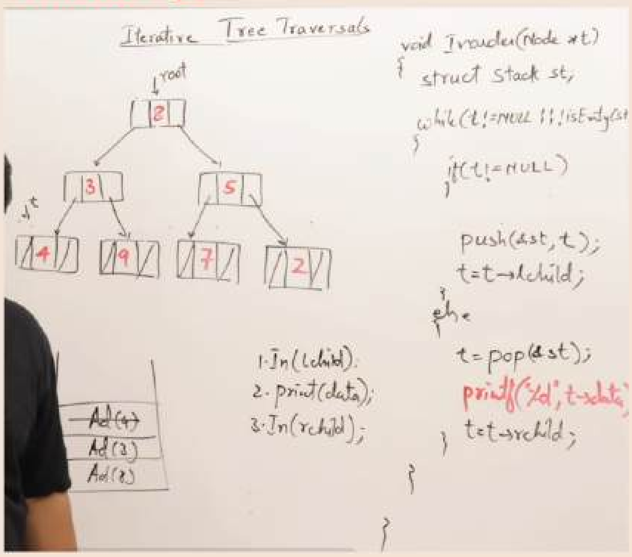


Iterative Tree traversal :- (Preorder)

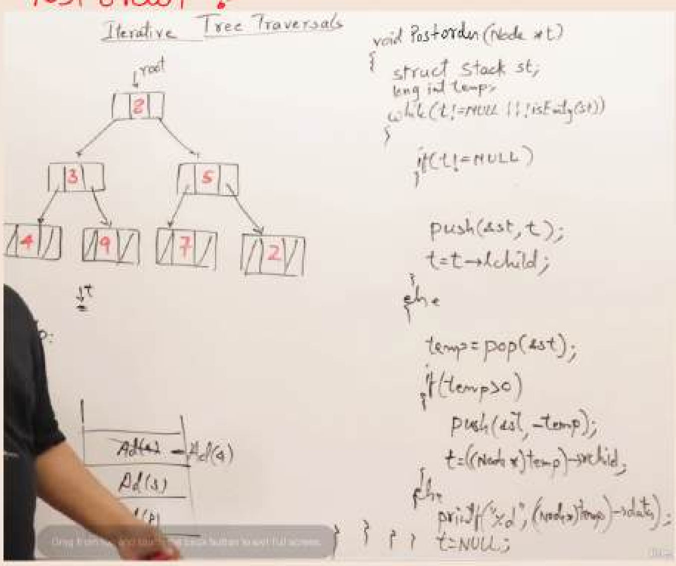


- Instead of the using recussion in iterative tree traversal we use stack to hold the information of the traversed nodes. Here stack should be of address type here.
- for Itereative tree traversal we have to use the stack which holds the address values. by using stack we can save the nodes which traversed.

Inorder :-

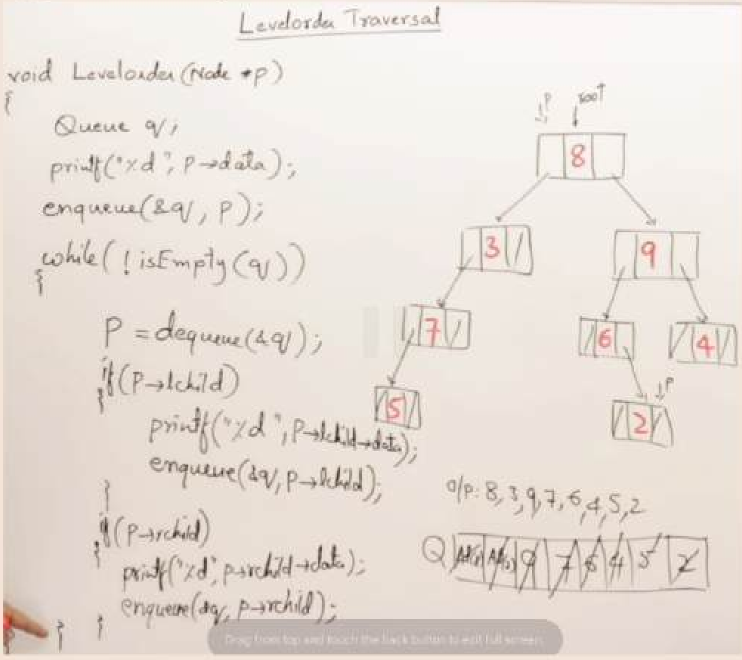


Post order :-



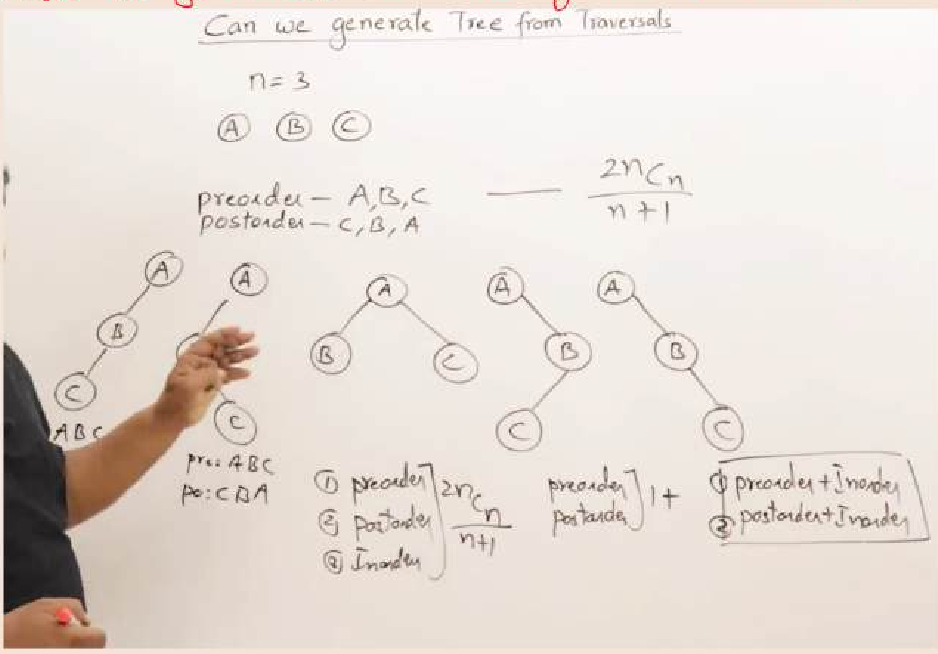
- the post order tree travesal is a little bit tricky, here we have to iterate through its nodes before printing its value.
- therfore when we iteate through left child we save normal address, but in next iteration when we got to the right child, we push the negative address to differentiate between the left and right child iterations. More in algo.

Level order :-



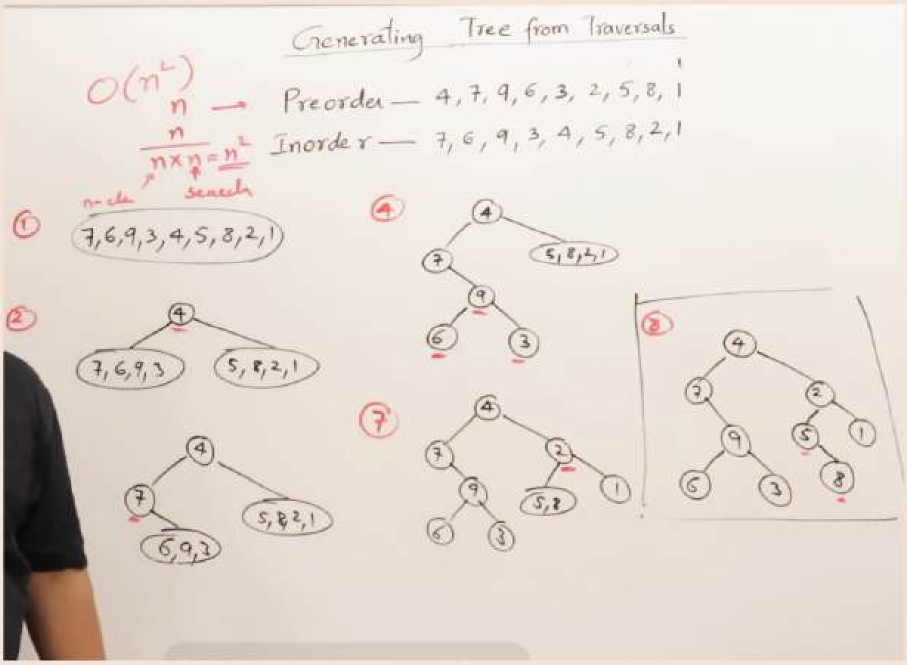
Level order is similar as inserting nodes into tree

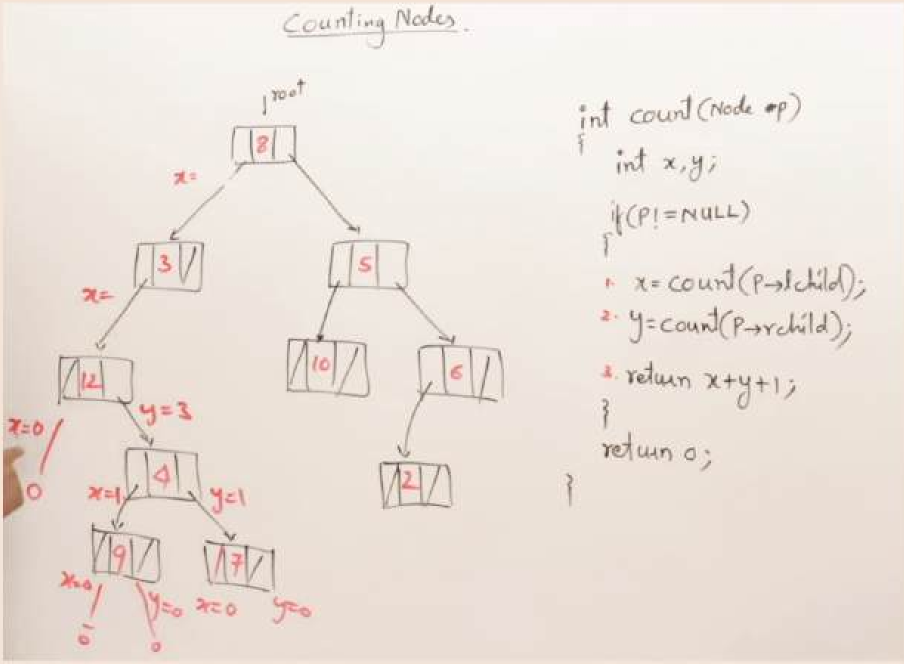
=> How to generate the Tree from Traversals:-



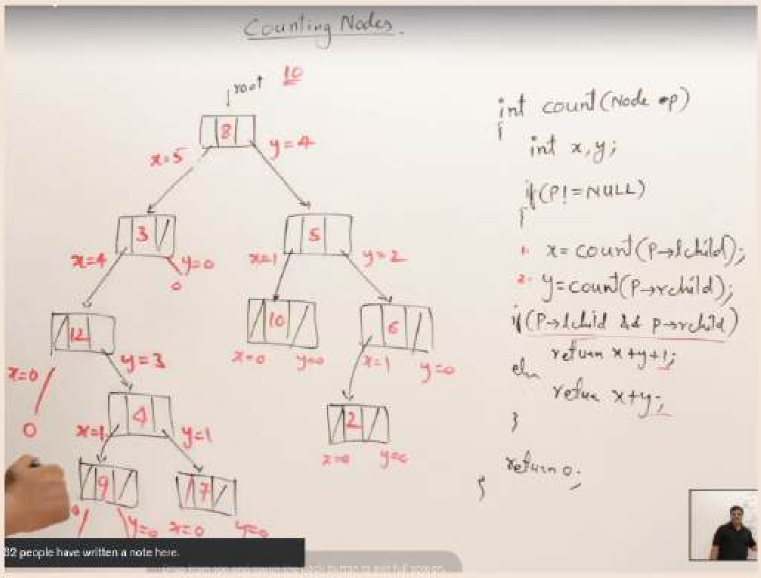
- either using pre order + post order, we can able to generate the tree or post+ in order,
  if we just use the pre +post order we can not construct the tree.

- We start iterating from the preorder elements, we find the same element in order sequence. once we finde that we split the left and right side of nodes for that node. we do this untill all the elemets are iterated. this procedure take the n^2 time complexity.

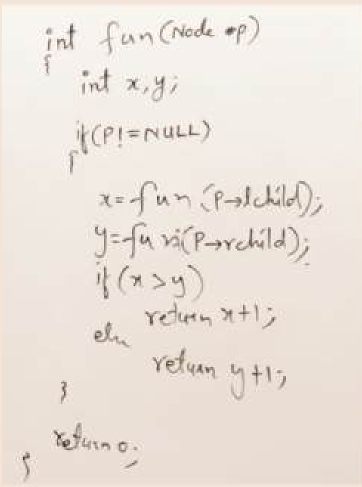




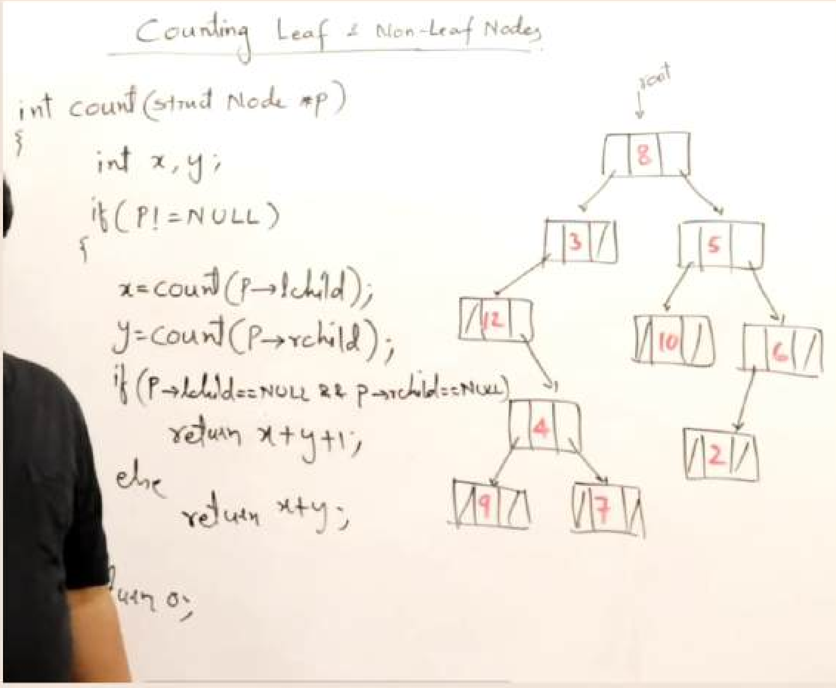
count the nodes which are having degree 2 in a tree.



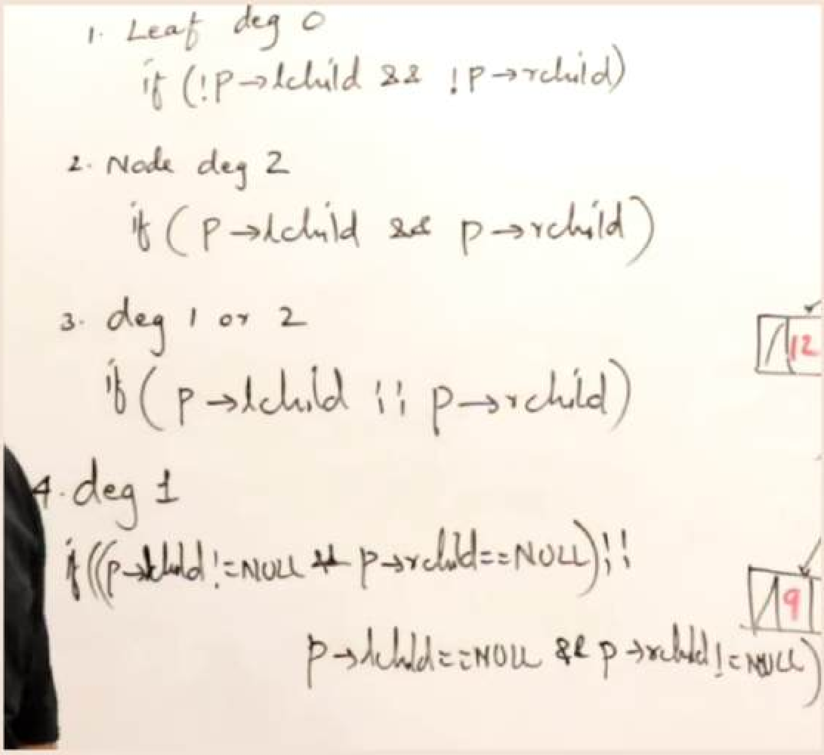
Finding the height of the tree



Counting leaf child.



Counting the leaf nodes in a tree.



Different conditions to find the nodes with different degrees in a tree.

The short condition to find the degree 1 nodes in a tree is below.