Trees 1

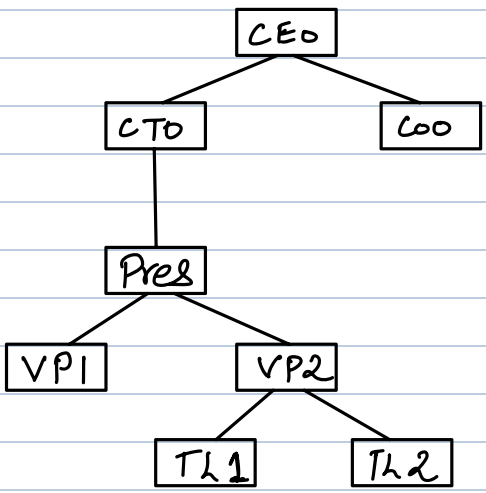
song: Get Down | Storm
- Zeja
↑
Friend

Topics To Cover:
- Trees Intro
- Naming convention
- Trees Traversal
- ★ - Iterative inorder
- ★ - Construct tree with pre [] & in []

Hi Everyone!!!

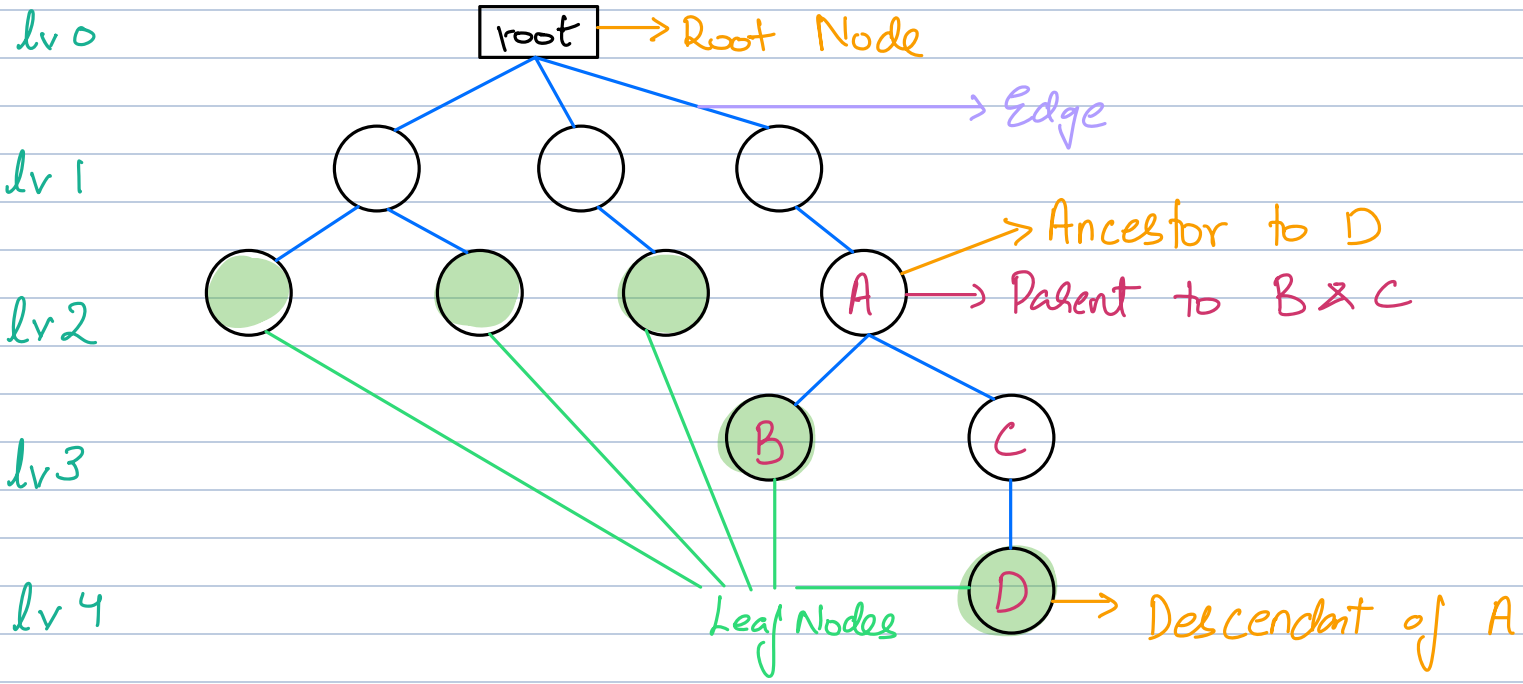
Linear Data Structure:
1. Array
2. LL
3. Stacks
4. Queues
5. String

Hirarichal Data Structures:



1. Family Tree
2. Folder Structure
3. Taxonomy
4. Politics / Polity.

Trees

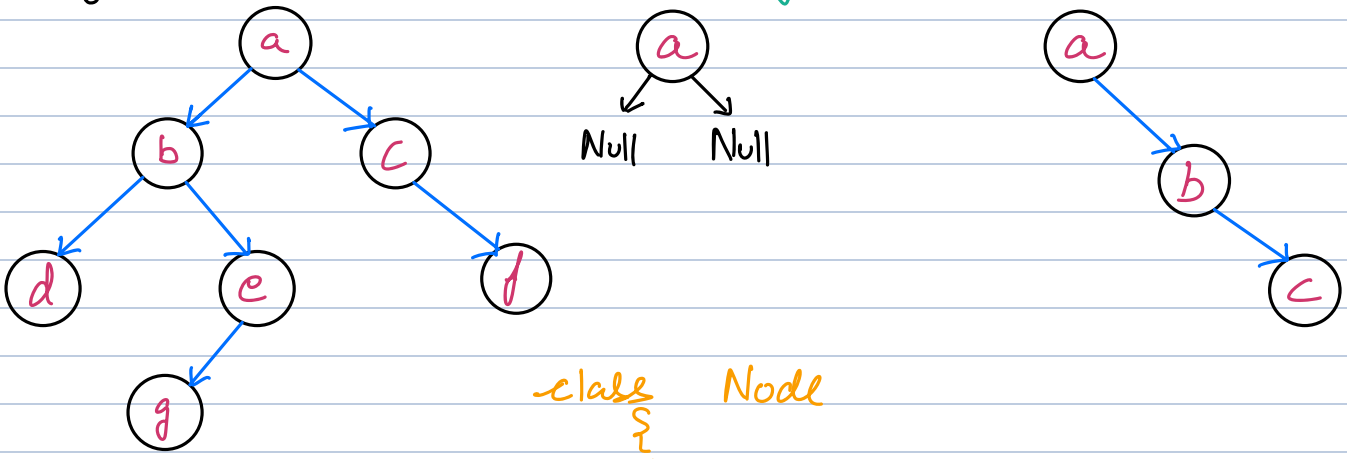


Height : longest path from root to leaf.

★ height based on edges : 4

height based on nodes : 5

Binary Tree : It has a maximum of 2 children




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

    Node (int x)
    {
    }
}
```

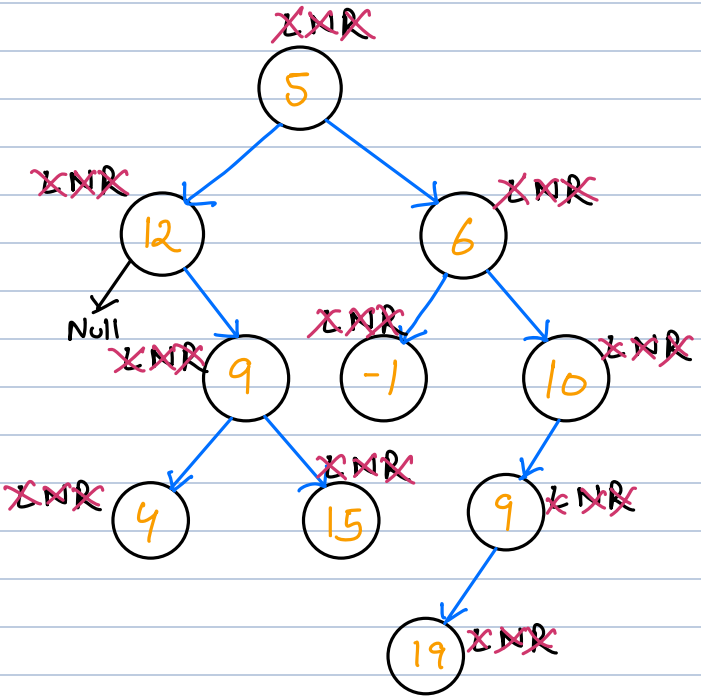
```
    data = x;
    left = Null;
    right = Null;
}
}
```

Traversal in a Tree.

Inorder Traversal

L N R

Left Node Right



Inorder : 12 4 9 15

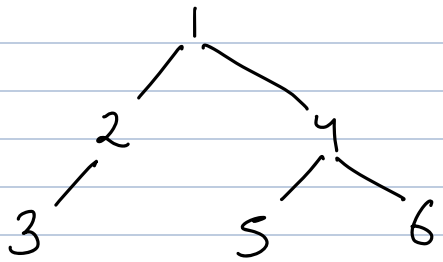
output 5 -1 6 19

9 10

```
void inorder (Node node)
{
    if (node == Null)
        return;

    inorder (node.left);

    print (node.data);

    inorde (node.right);

}
```

TC: O(N)

SC: O(ht. of Tree)



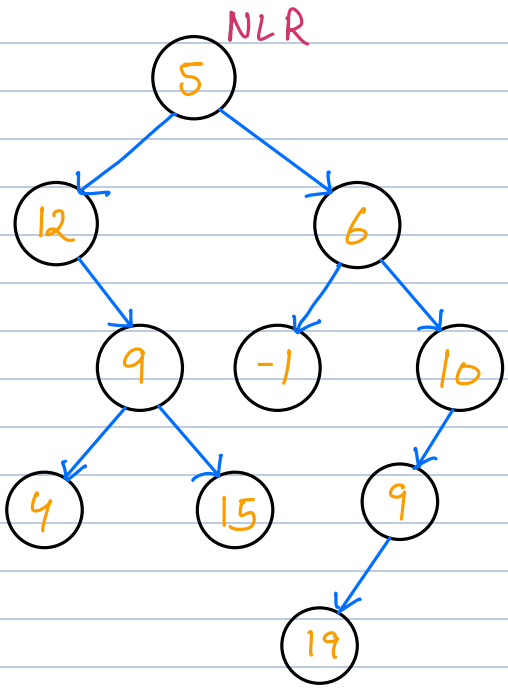
print: 3, 2, 1, 5

4, 6

Preorder Traversal

N L R

Node Left Right



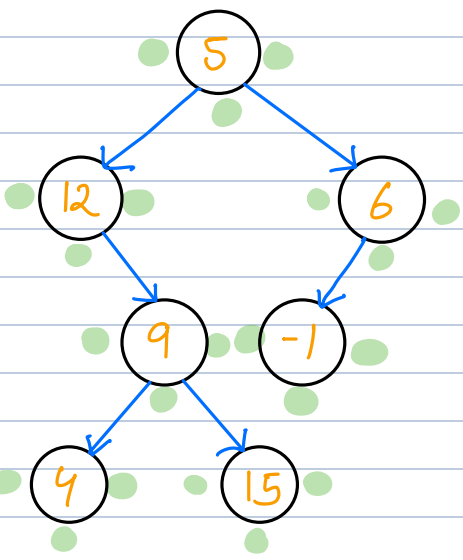
output : 5, 12, 9, 4, 15,
6, -1, 10, 9, 19

```
void preorder (Node node)
{
    if (node == Null)
        return;

    print (node.data);
    preorder (node.left);
    preorde (node.right);
}
```

Post order :

L R N

Left Right Node



Preorder : 5, 12, 9, 4, 15, 6, -1

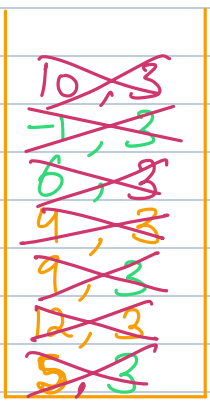
Inorder : 12, 4, 9, 15, 5, -1, 6

Postorder : 4, 15, 9, 12, -1, 6, 5

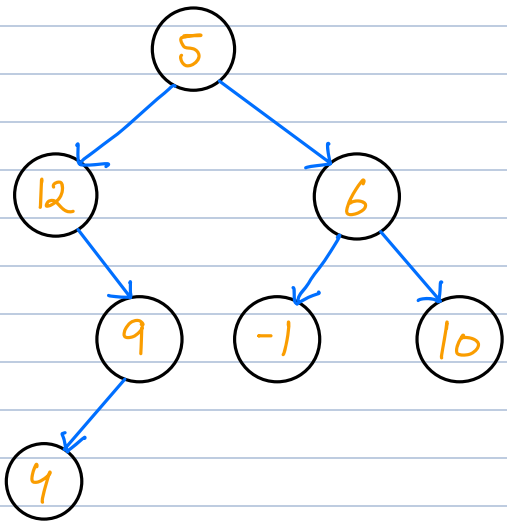
```
void postorder (Node node)
{
    if (node == Null)
        return;
    postorder (node.left);
    postorde (node.right);
    print (node.data);
}
```

Inorder Traversal using iteration.

1. Call left child
2. Print Data
3. Call Right child
4. Go to the parent Node




```
class pair
{
    Node node;
    int state;
}
```



Step1: put root in stack.

Step2: Perform operations acc to states till stack is empty.

output: 12, 4, 9, 5, -1, 6, 10

States:

0 - Go towards left

1 - Print Node

2 - Go towards right

3 - Pop the Node.

```
fn iterative_inorder ( Node root)
{
    stack <pair> st ;
    p = pair(root, 0);
    st.push (p);
    while ( st.size != 0)
    {
        top_p = st.peek ( );
        if ( top_p.state == 0 )
        {
            if ( top_p.node.left != Null )
            {
                Tp = pair (top_p.node.left, 0);
                st.push(Tp);
            }
        }

        else if ( top_p.state == 1)
        {
            print (top_p.node.data);
        }

        else if ( top_p.state == 2)
        {
            if ( top_p.node.right != Null )
            {
                Tp = pair (top_p.node.right, 0);
                st.push(Tp);
            }
        }

        else          # state == 3
        {
            st.pop ( );
            continue ;
        }
        top_p.state ++ ;
    }
```

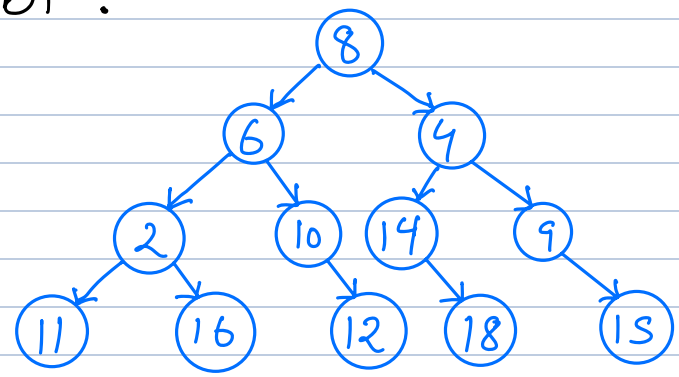
TC: $\Theta(N)$

SC: Θ(height of the tree)

★★★ This is done for states 0, 1, 2

Break: 10:44 – 10:54

Q. Given preorder[] & inorder[] of Binary Tree with distinct values Construct BT.



RST

ps + LST

ps

LST

pe

pre : [8 , 6 , 2 , 11 , 16 , 10 , 12 , 4 , 14 , 18 , 9 , 15]

in : [11 , 2 , 16 , 6 , 10 , 12 , 8 , 14 , 18 , 4 , 9 , 15]

is

LST

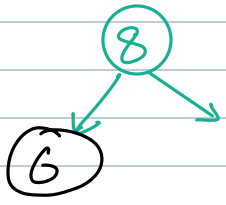
root idx

RST

ie

1. Use pre order to find root node b/w ps & pe
2. Search for root node in inorder b/w is & ie
3. Count elements in LST in inorder
4. Create division based on this Count.

```
fn { create-Tree (pre[], ps, pe, in[], is, ie)
                  int  int        int  int.
     if (ps > pe)
         return Null;

     int root_data = pre[ps];

     Node node = Node (root_data)

     root_idx = In. search (root_data)
```



```
    int LST = root_idx - is;
    node.left = create_tree(pre, ps+1, ps+LST, in, is, root_idx-1);
    node.right = create_tree(pre, ps+LST+1, pe, in, root_idx+1, ie);
    return node;
}
```

priority queue ⟶ <data, int→priority>