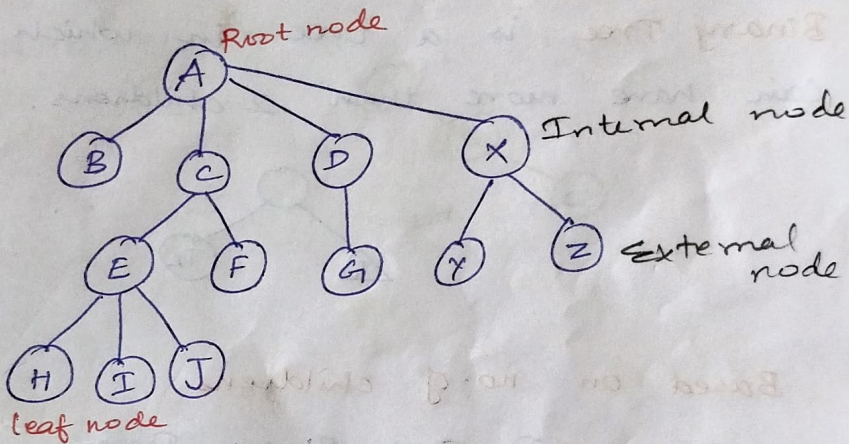
Unit III

Trees

Tree ADT – Tree Traversals – Binary Tree ADT – Expression trees – Binary Search Tree ADT – Balanced Binary Trees – AVL Trees – Priority Queues (Heaps) – Binary heap.

Tree ADT

* Tree – Collection of nodes.
* Non-linear DS in which data are stored in hierarchical manner.



Terminologies:

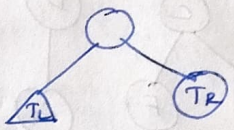
* parent node
* leaf node
* Sibilings
* path
* length : no. of edges on the path.
* Degree : no. of subtree of a node – degree
  A → 4 C → 2 D → 1 X → 2
* level : level of A is 0
  level of B, C, D, X is 1
* Ancestor : path from n_1 to n_2 – n_1 ancestor
  n_2 descendant
* height : height of node n is defined as the longest path from node n to leaf.

General Tree - ~~Node~~ Tree can have any no. of nodes and there is no degree restrictions on nodes. A General tree is a n-ary tree
(any integer Value)

Types of Trees

- Binary Tree (2 childs)
- Ternary Tree (3 childs)
- N-ary Tree (n - any int value)

Binary Tree is a tree in which no node can have more than 2 childrens.

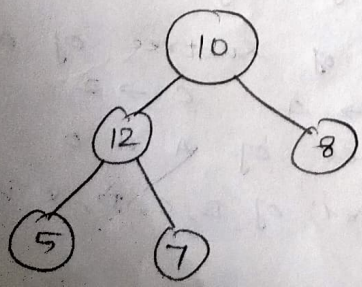


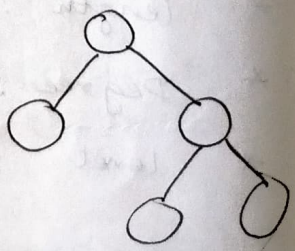
Based on no. of childrens.

1. Full Binary Tree
2. Degenerate Binary Tree
3. Skewed Binary Tree.

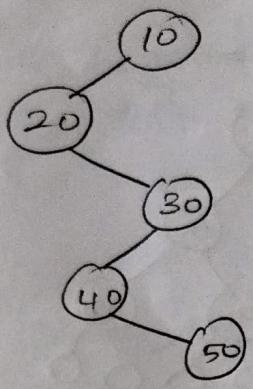
① Full Binary Tree : Node has 0/2 Children.

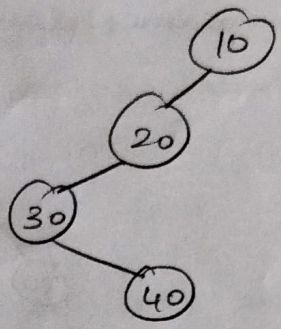
Every parent / internal node has either two / no children. Otherwise called proper Binary Tree.



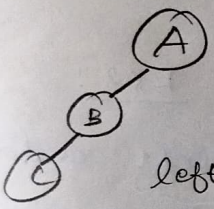


(2) Degenerate / Pathological Tree : A tree having a single child either at left/ right.

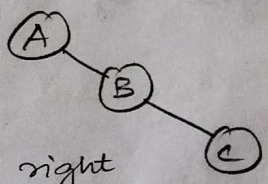




(3) Skewed Binary Tree : dominated by left nodes/ right nodes. 2 types are: left-skewed binary tree & right skewed binary tree.



left



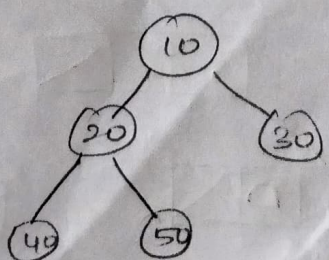
right

Based on Completion of Levels.

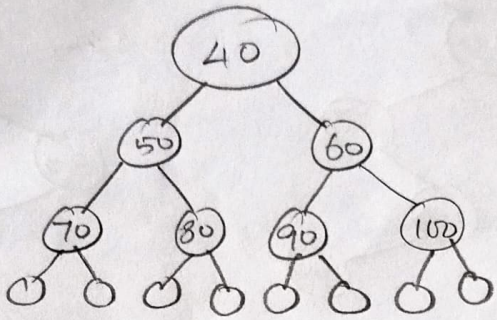
1. Complete Binary Tree
2. Perfect Binary Tree
3. Balanced Binary Tree.

1. Complete Binary Tree.

All the levels are completely filled except the last level & last level has all keys as left as possible. (leaf node must lean towards left)



2. Perfect Binary Tree → all internal nodes have 2 children and all the leaf nodes are at the same level.



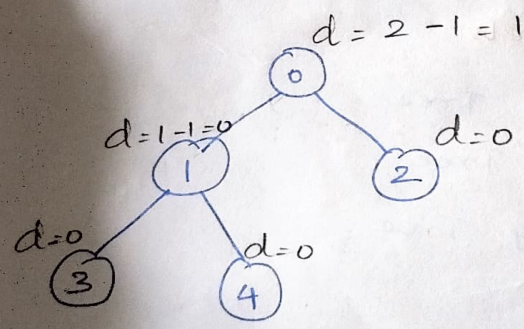
3. Balanced Binary Tree $d \rightarrow 0, 1, -1$

height of the tree - $O(\log n)$

height of h_L - height of $h_R = 0/1$

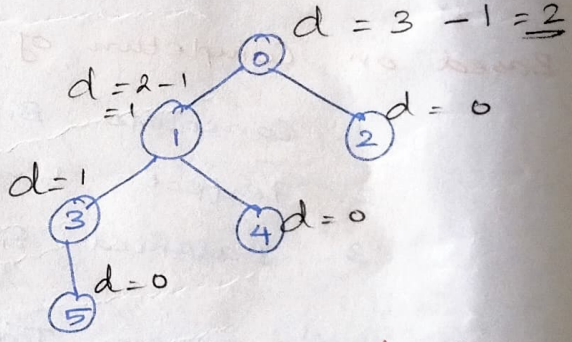
Ex 1:

depth of node = height of left child - height of right child.



Balanced Binary Tree

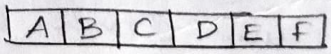
Ex 2:

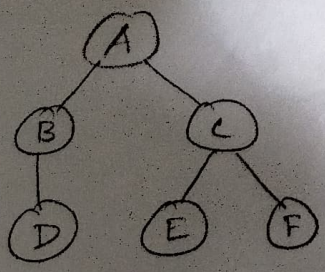


Unbalanced Binary Tree.

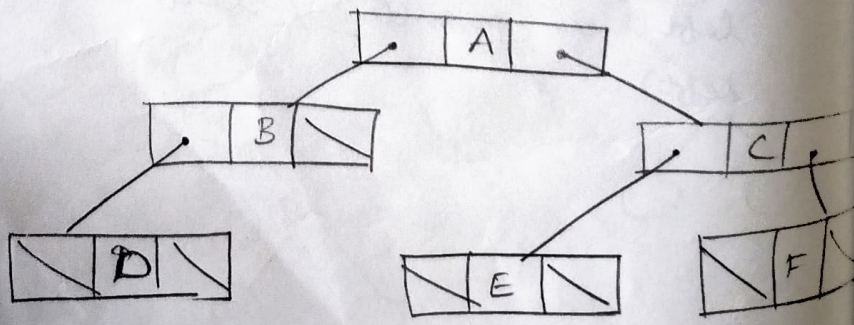
Representation < Array / Linked List

Binary Tree Represation.





Linked list



```
struct node
{
    int data;
    struct node *left;  // left sub-tree
    struct node *right; // right sub-tree
};
```

Tree Traversal

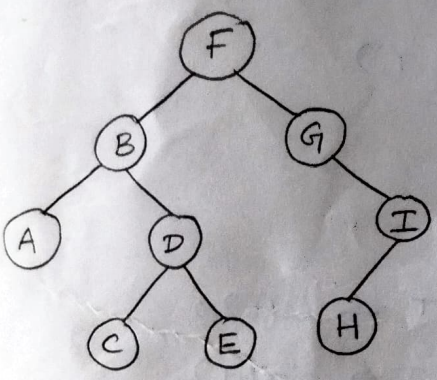
Traversal is visiting each node only once. Visiting all the nodes in the tree exactly once.

3 types of traversal:

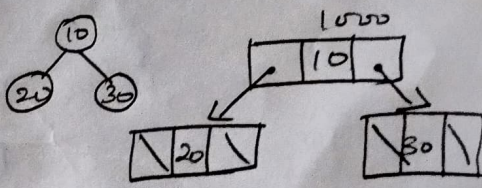
1. Inorder traversal
2. Preorder traversal
3. Postorder traversal.

Inorder Traversal.

* Traverse the left subtree in order
* Visit the root
* Traverse the right subtree in order



Inorder: A B C D E F G H I
preorder: F B A D C E G I H
postorder: A C E D B H I G F




```
Void inorder (Tree T)
{
    if (T != null)
    {
        inorder (T -> left);
        printf ("%d", T -> data);
        inorder (T -> right);
    }
}
```

Preorder Traversal.

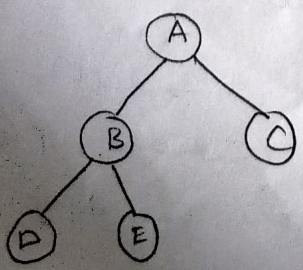
* Visit the root
* Traverse the left subtree in pre-order
* Traverse the right subtree in pre-order

```
Void preorder (Tree T)
{
    if (T != null)
    {
        printf ("%d", T -> data);
        preorder (T -> left);
        preorder (T -> right);
    }
}
```

postorder Traversal

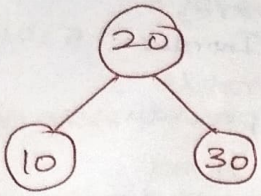
* Traverse the left subtree in postorder
* Traverse the right subtree in postorder
* Visit the root.

```
Void postorder (Tree T)
{
    if (T != Null)
    {
        postorder (T -> left);
        postorder (T -> right);
        Printf (" %d", T -> data);
    }
}
```



DEBCA

Example 1 :

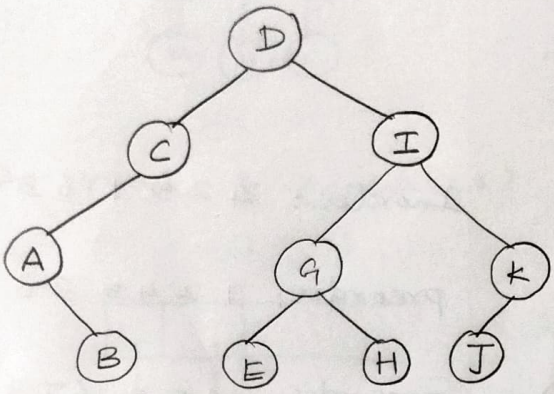


Inorder : 10 20 30

preorder : 20 10 30

postorder : 10 30 20

Example 2 :



l r R

~~Inorder : A B C E H G J K I D~~

root l r

Preorder : D C A B I G E H K J

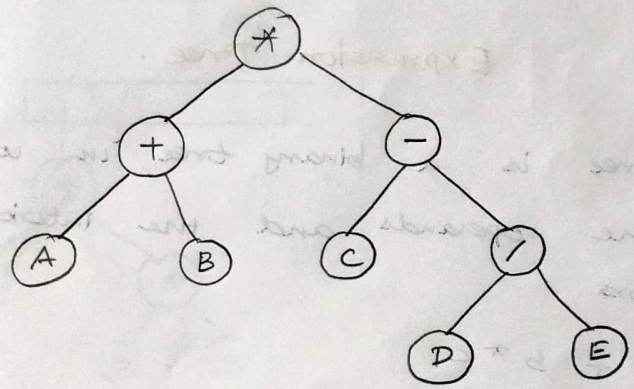
l r root

postorder : B A C E H G J K I D

l root r

Inorder : A B C D E G H I J K

Example 3 :



l root R

Inorder : A + B * C - D / E

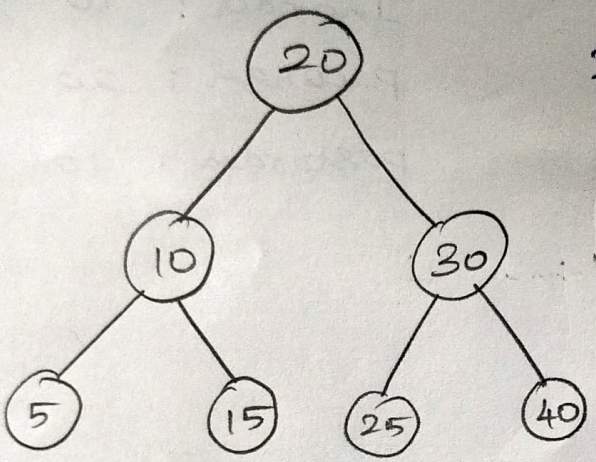
root l R

preorder : * + A B - C / D E

l R root

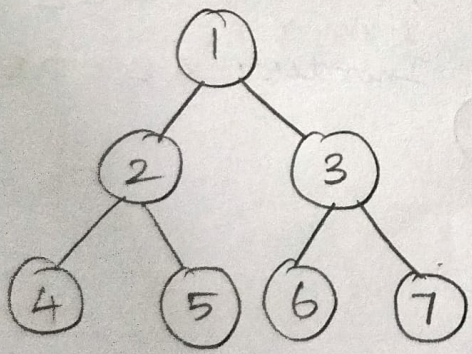
postorder : A B + C D E / - *

Example 4:



l root R
Inorder: 5 10 15 20 25 30 40
root l R
preorder: 20 10 5 15 30 25 40
l R root
postorder: 5 15 10 25 40 30 20

Example 5:



Inorder: 4 2 5 1 6 3 7

preorder: 1 2 4 5 3 6 7

postorder: 4 5 2 6 7 3 1

Expression Tree.

* Expression tree is a binary tree in which the leaf node are operands and the interior nodes are operators

Ex: a + b * c