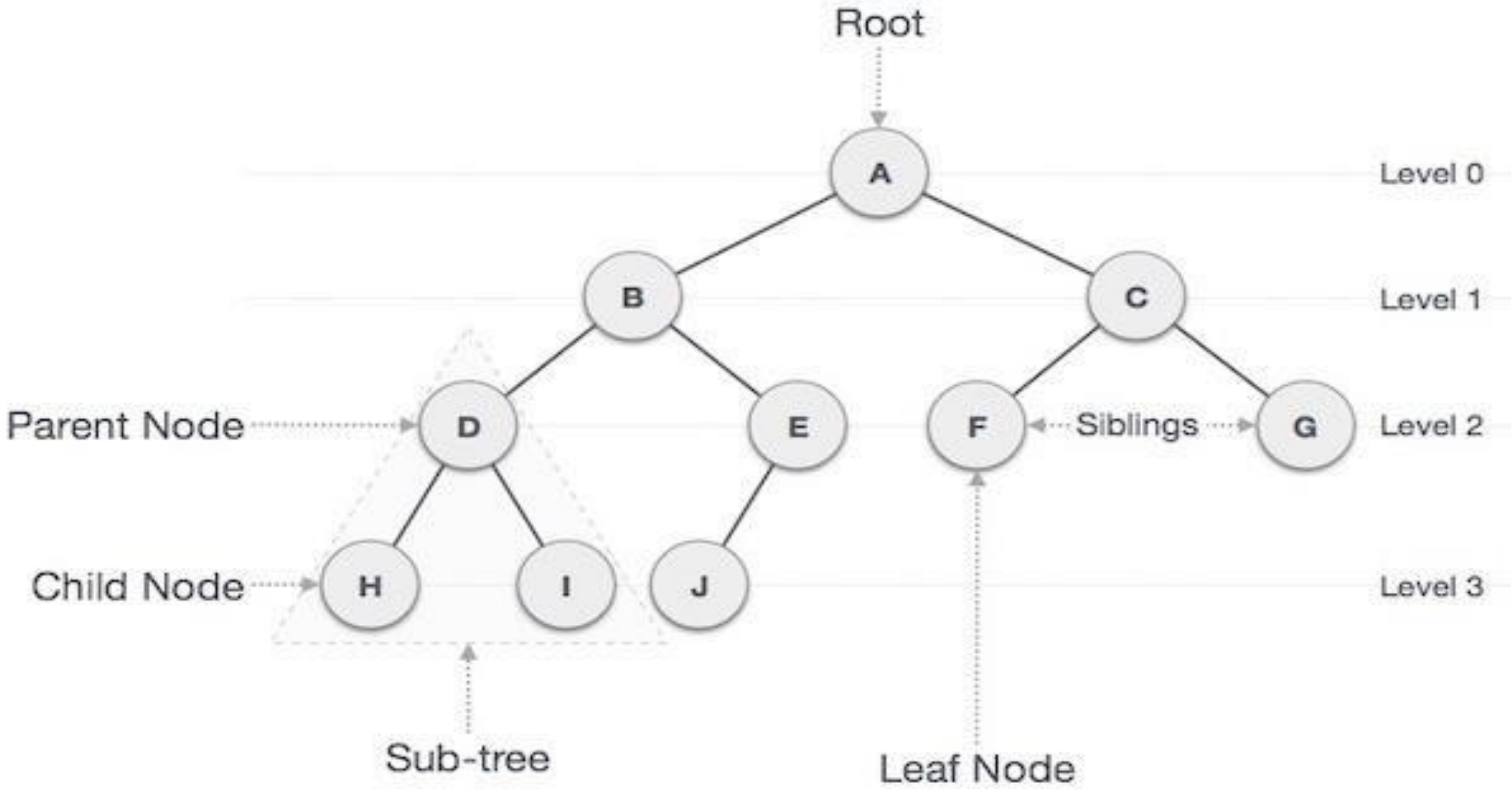
Trees


Root
A
Level 0
B
C
Level 1
Parent Node
D
E
F
Siblings
G
Level 2
Child Node
H
I
J
Level 3
Sub-tree
Leaf Node

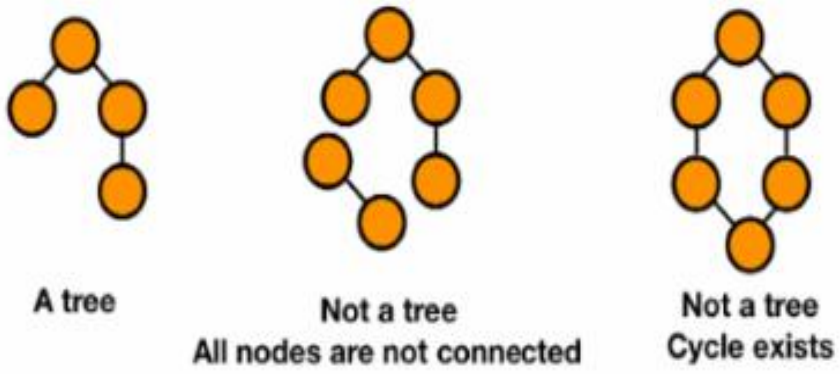
Properties of a Tree

A tree must have some properties so that we can differentiate from other data structures. So, let's look at the properties of a tree.

The numbers of nodes in a tree must be a finite and nonempty set.

There must exist a path to every node of a tree i.e., every node must be connected to some other node.

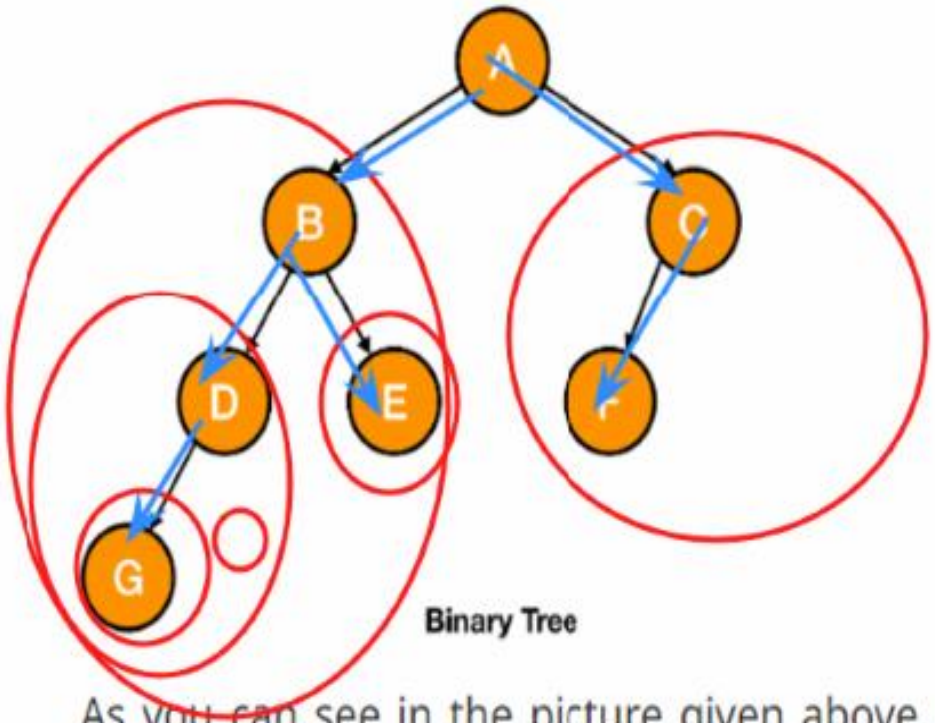
There must not be any cycles in the tree. It means that the number of edges is one less than the number of nodes.



Binary Trees

no. of childrens = 0, 1, 2

A binary tree is a tree in which every node has at most two children.



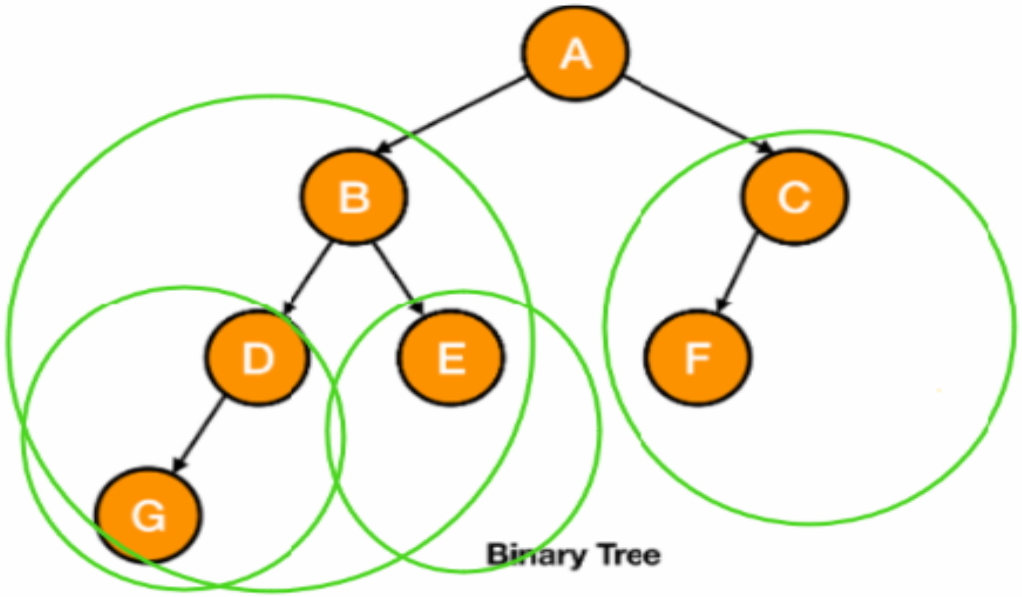
Binary Tree

As you can see in the picture given above, a node can have less than 2 children but not more than that.

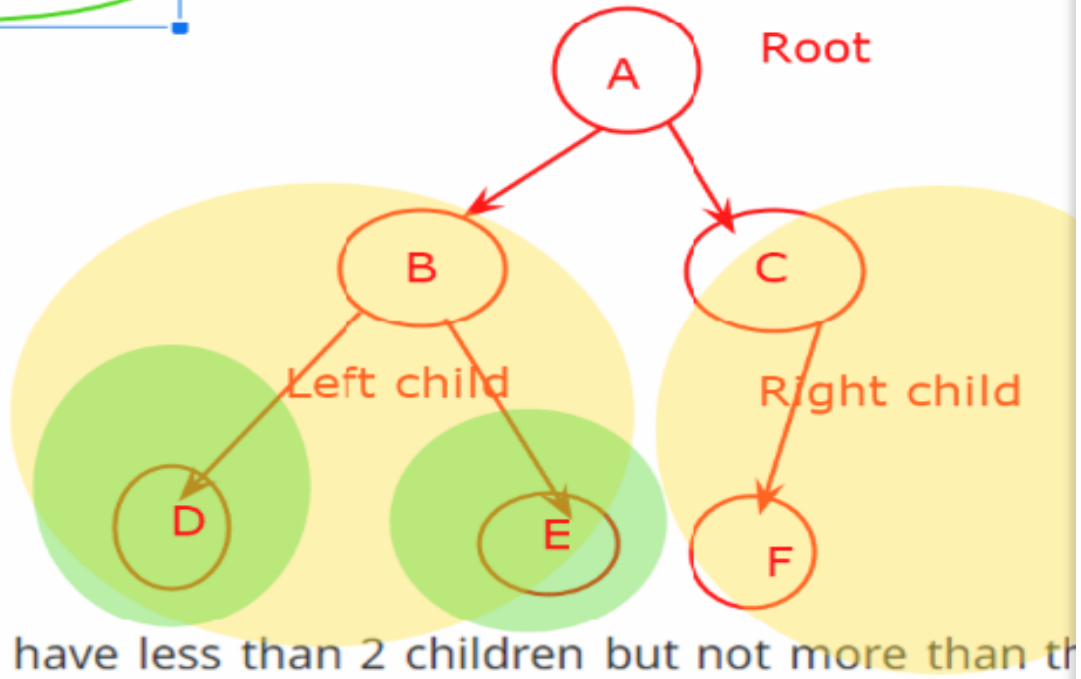
We can also classify a binary tree into different categories. Let's have a look at them:

Binary Trees

A binary tree is a tree in which every node has at most two children.



Binary Tree



As you can see in the picture given above, a node can have less than 2 children but not more than th

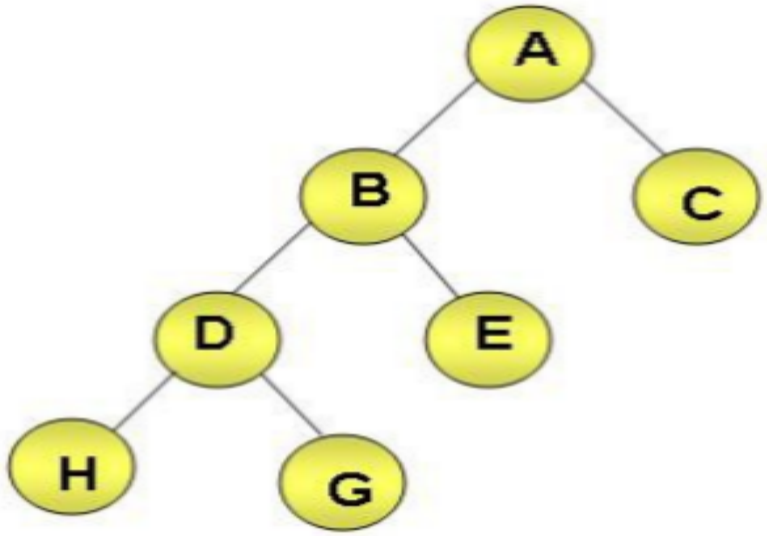
Left subtree Right Su

We can also classify a binary tree into different categories. Let's have a look at them:

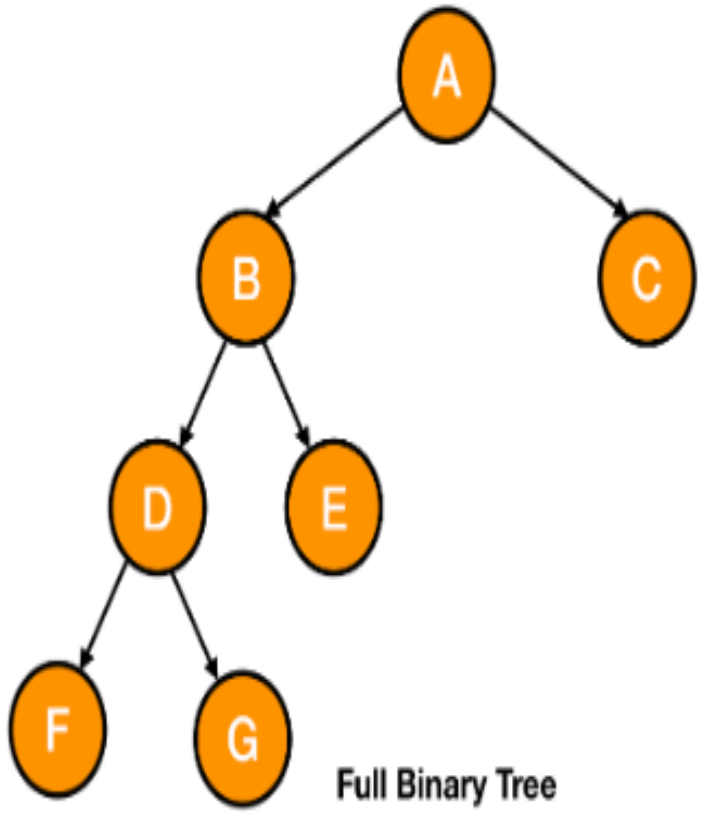
CDAC Mumbai:Kiran Waghmare

Defining Binary Trees

- Binary tree is a specific type of tree in which each node can have at most two children namely left child and right child.
- There are various types of binary trees:
  - Strictly binary tree
  - Full binary tree
  - Complete binary tree
- Strictly binary tree:
  - A binary tree in which every node, except for the leaf nodes, has non-empty left and right children.



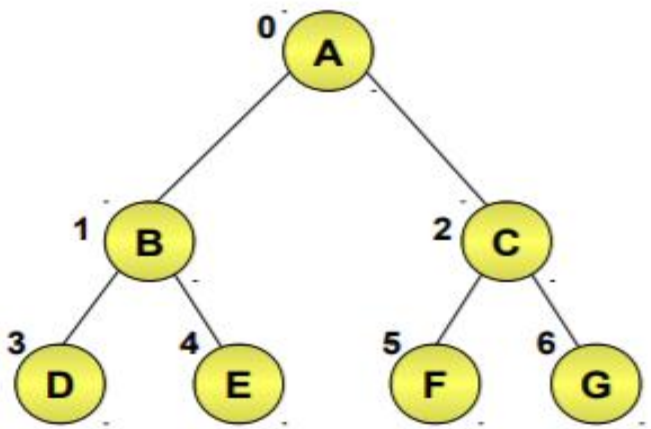
Full Binary Tree → A binary tree in which every node has 2 children except the leaves is known as a full binary tree.



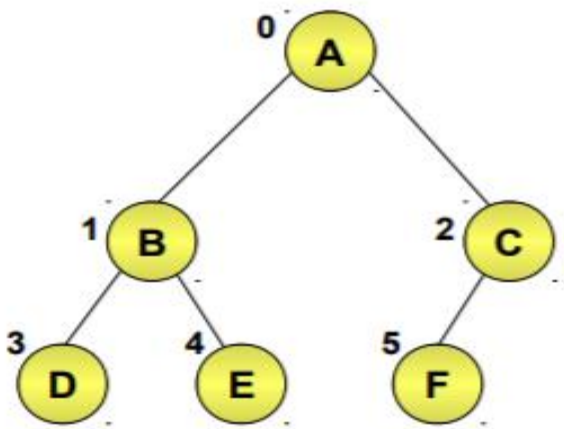
Full Binary Tree

Defining Binary Trees (Contd.)

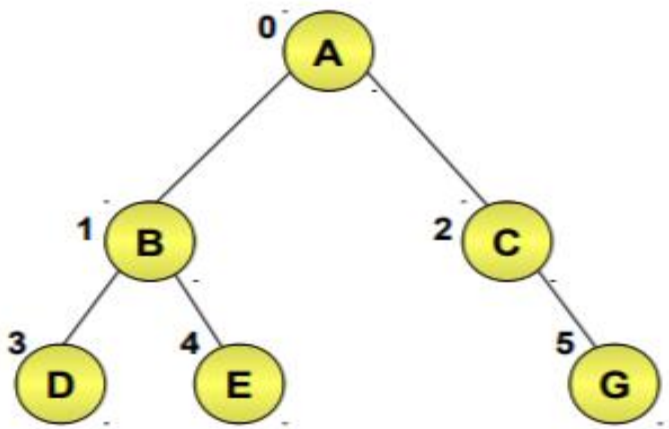
- Complete binary tree:
  - A binary tree with n nodes and depth d whose nodes correspond to the nodes numbered from 0 to n − 1 in the full binary tree of depth k.



Full Binary Tree



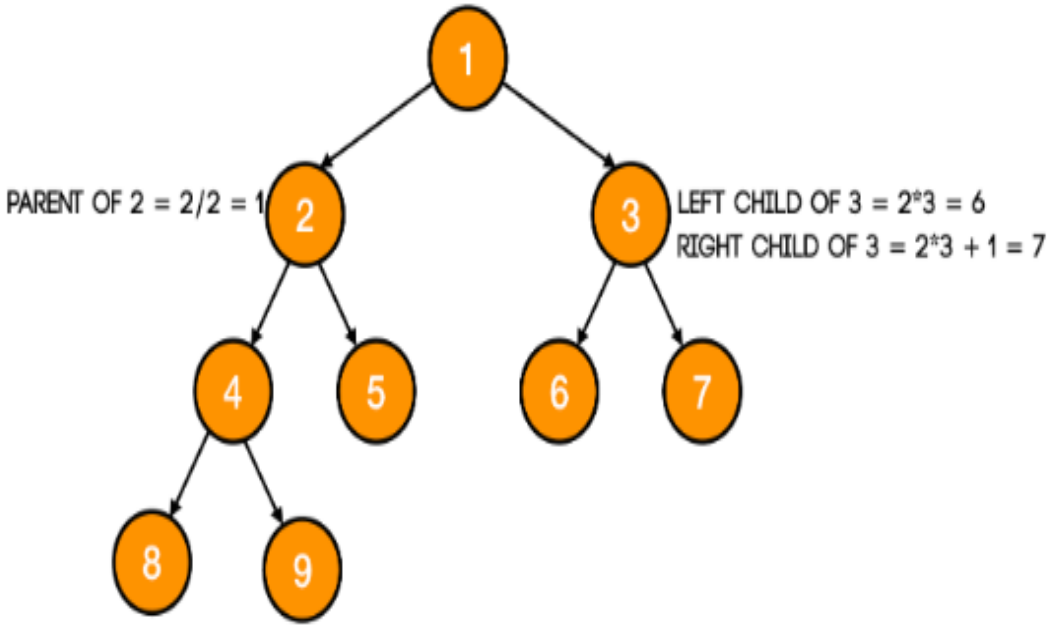
Complete Binary Tree

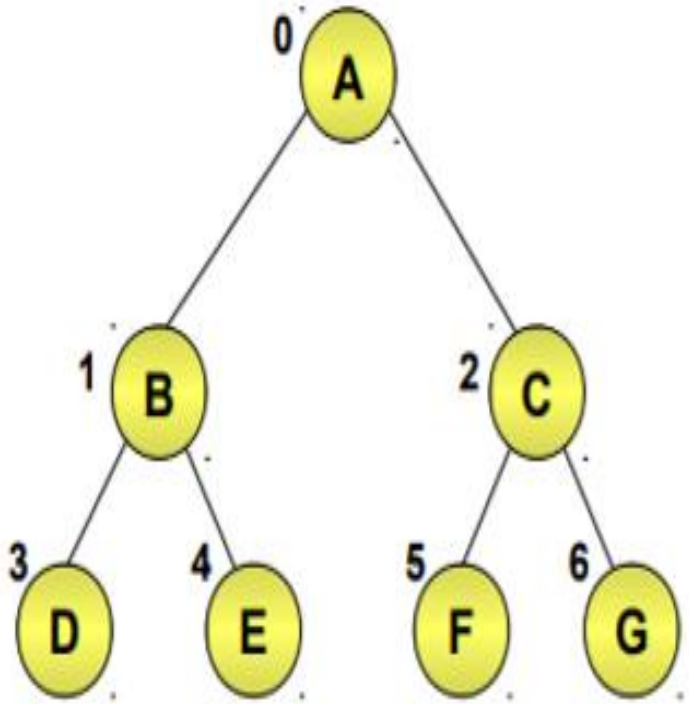


Incomplete Binary Tree

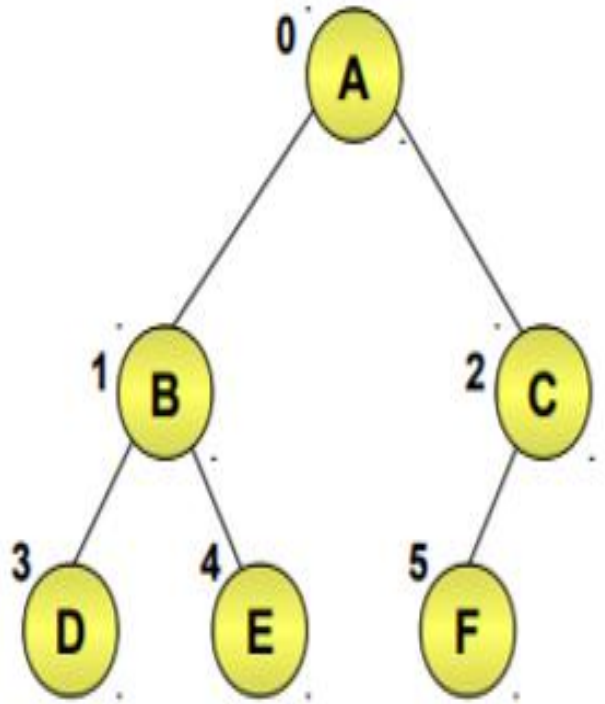
A complete binary tree also holds some important properties. So, let's look at them.

- The **parent of node i** is $\left\lfloor \frac{i}{2} \right\rfloor$. For example, the parent of node 4 is 2 and the parent of node 5 is also 2.
- The **left child of node i** is $2i$.
- The **right child of node i** is $2i + 1$

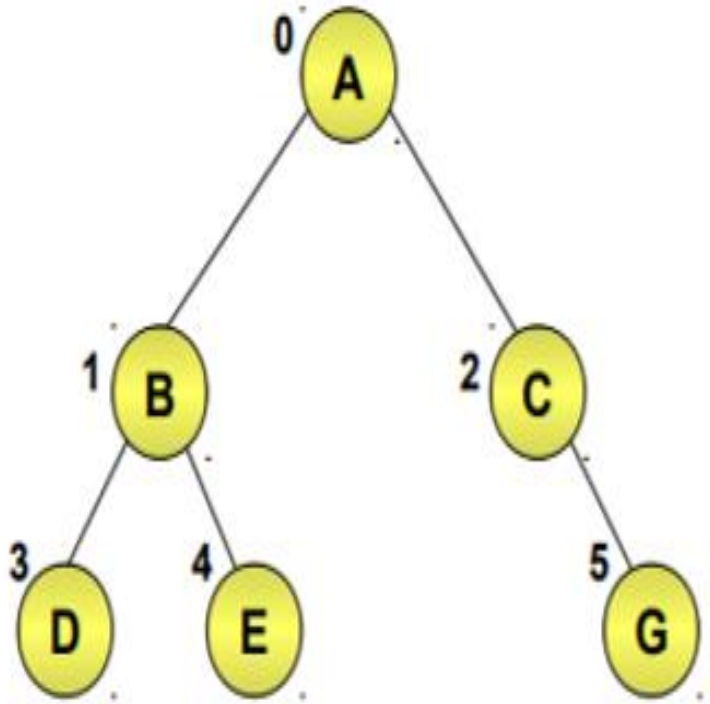




Full Binary Tree



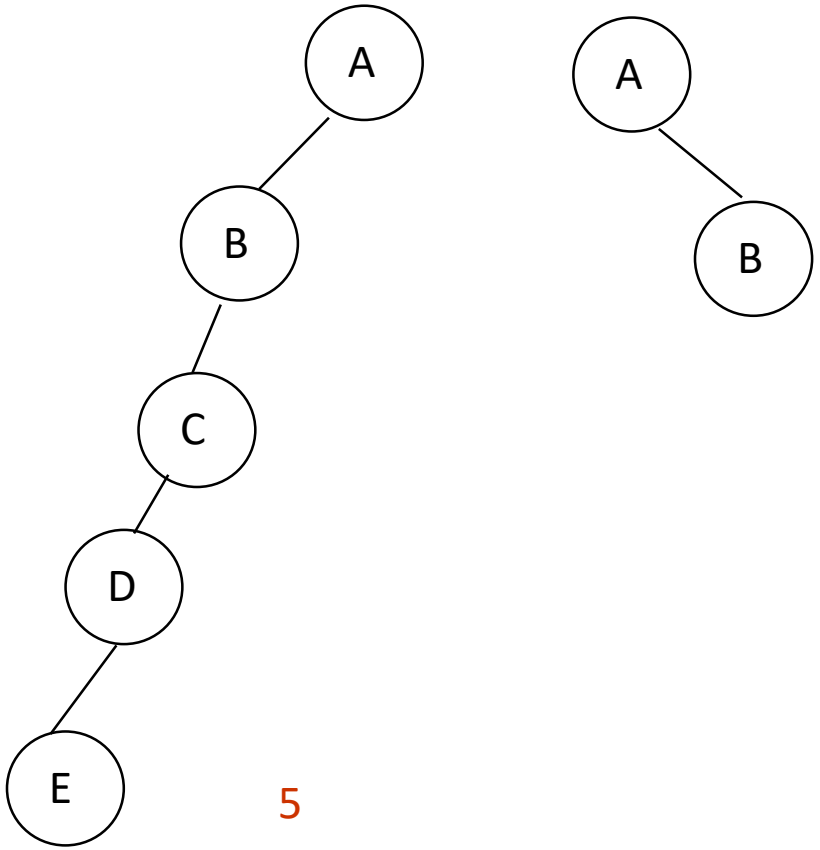
Complete Binary Tree



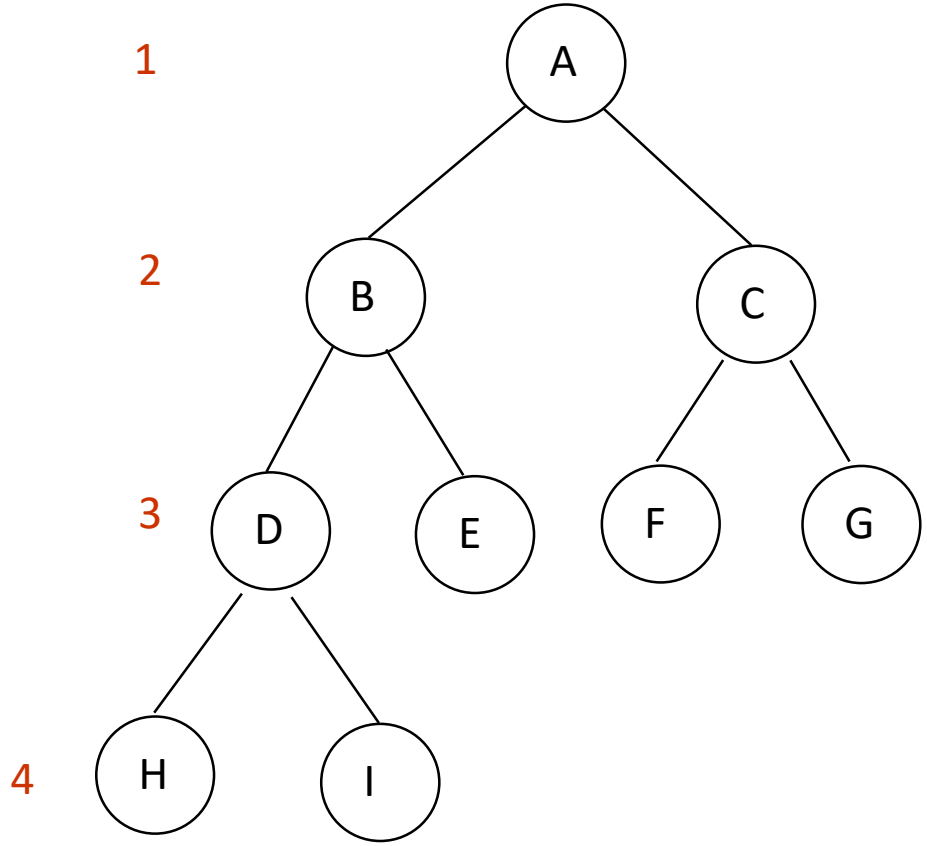
Incomplete Binary Tree

Examples of the Binary Tree

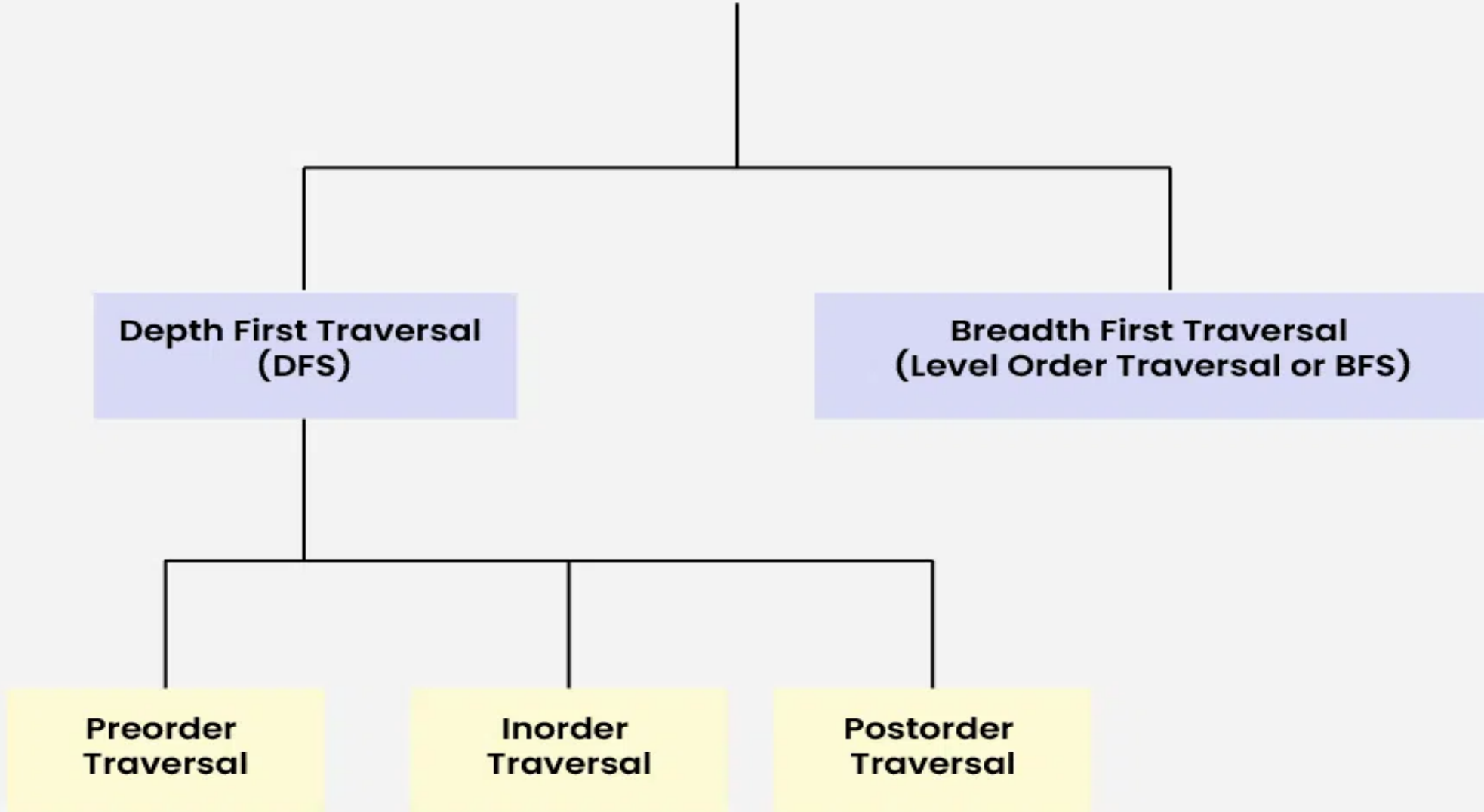
Skewed Binary Tree

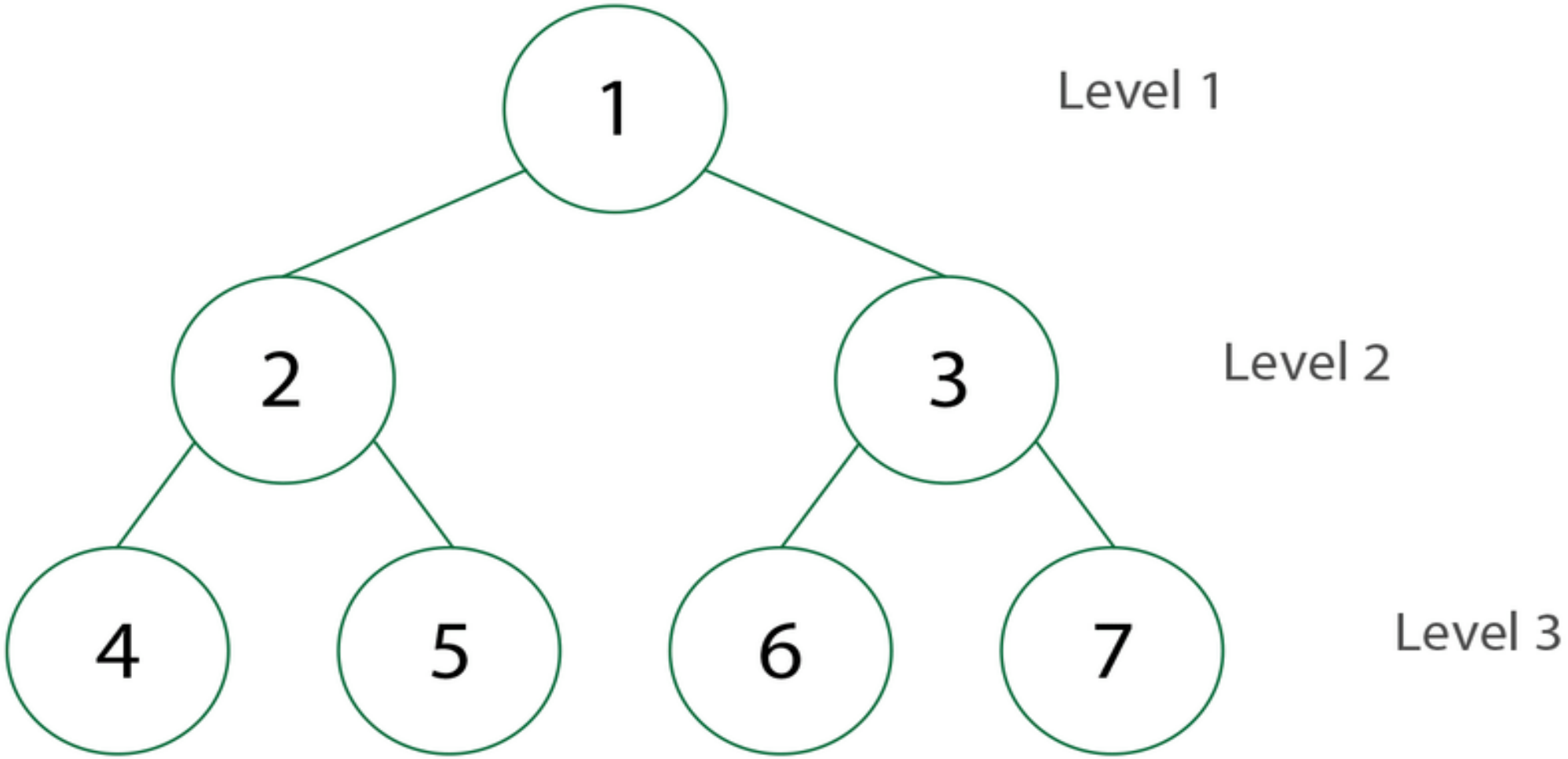


Complete Binary Tree



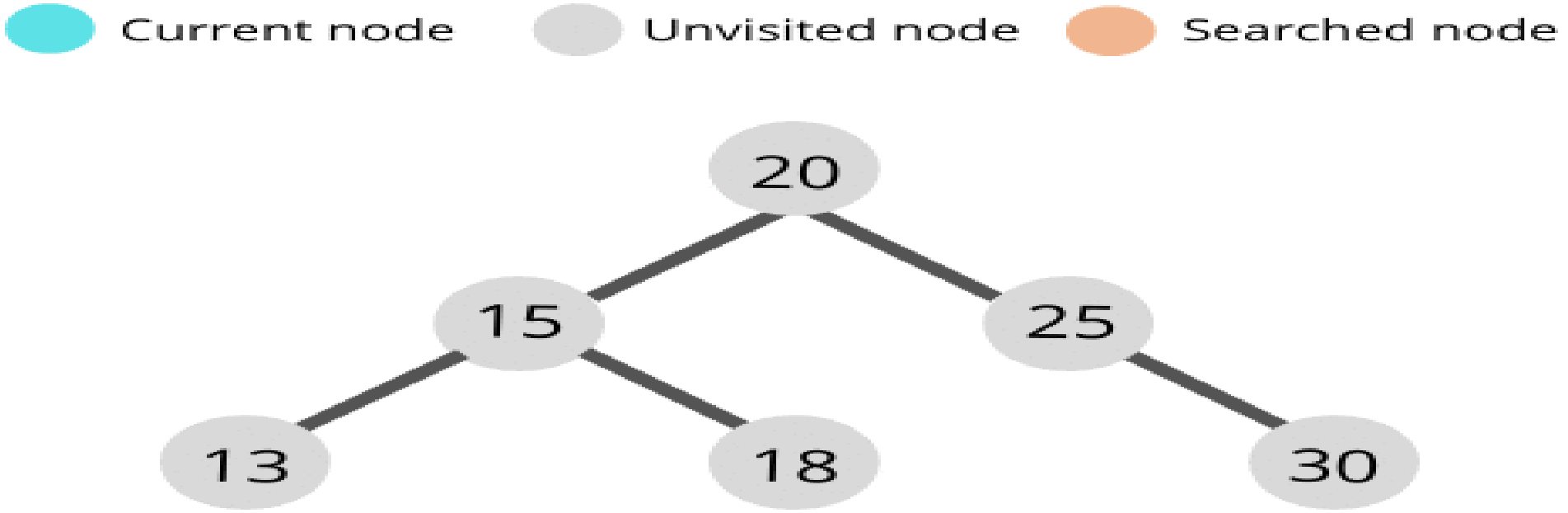
Tree Traversal Techniques



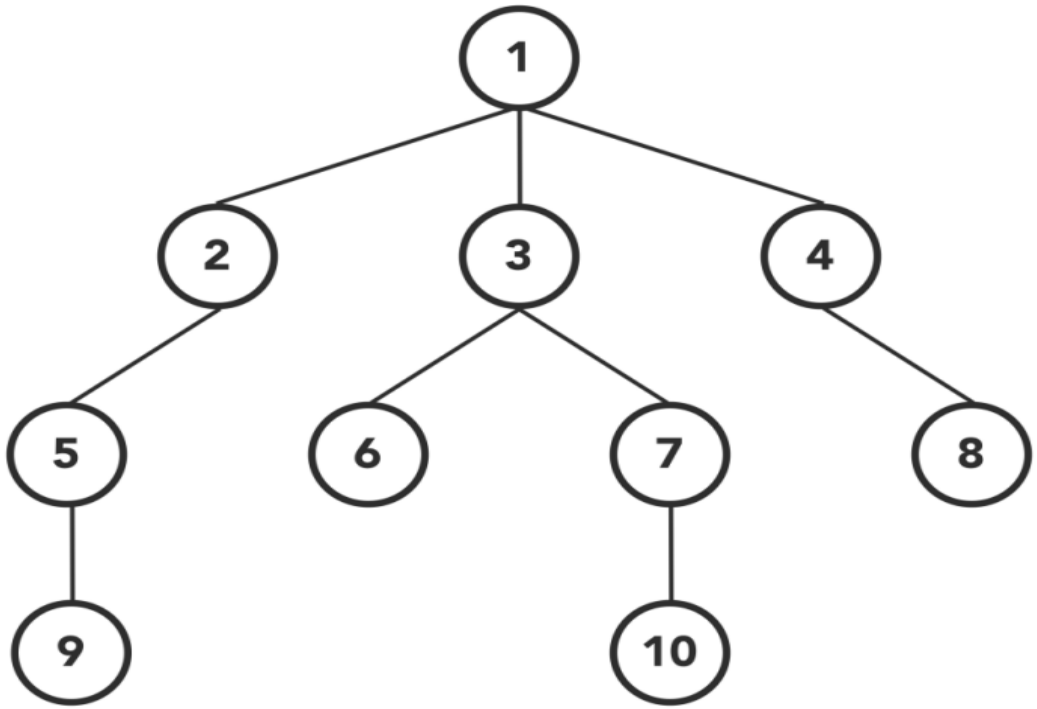

1
Level 1
2
3
Level 2
4
5
6
7
Level 3

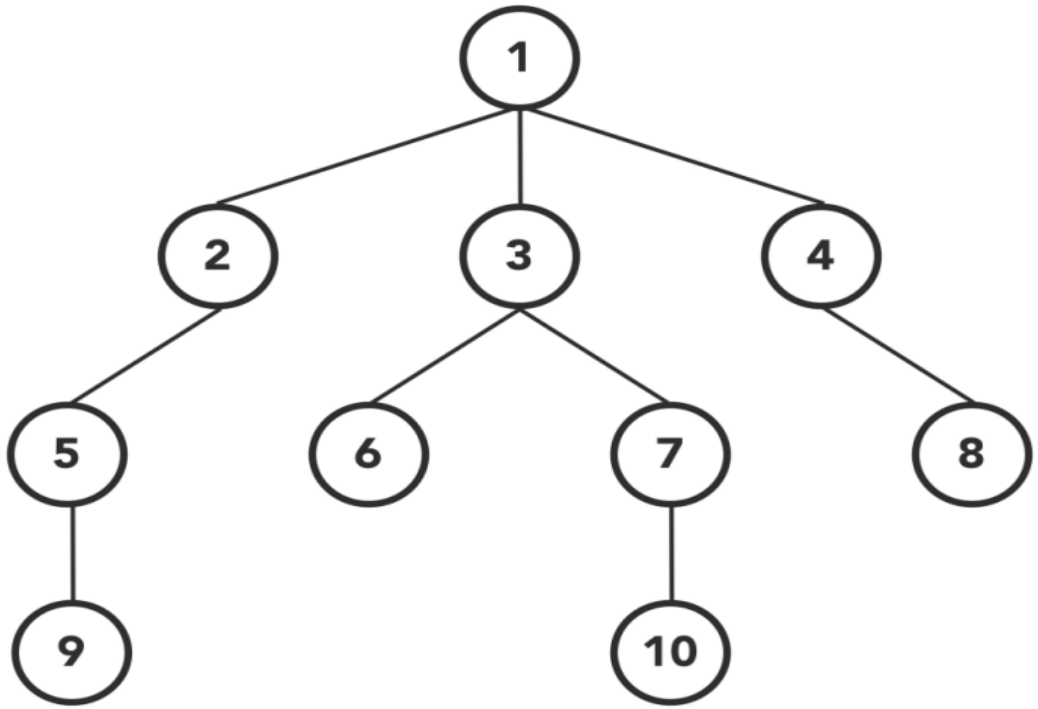
Depth First Search

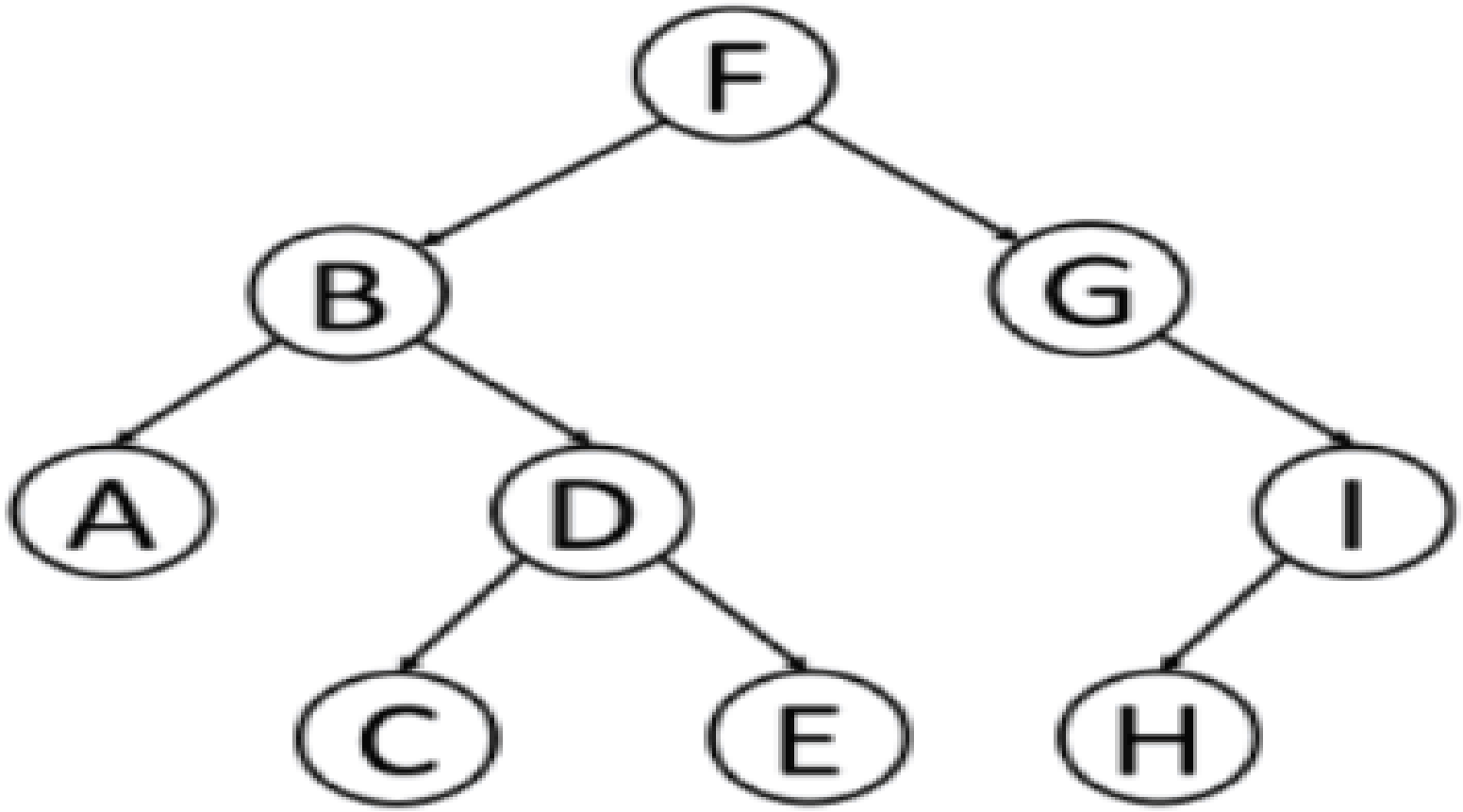
Preorder (Root - Left - Right)



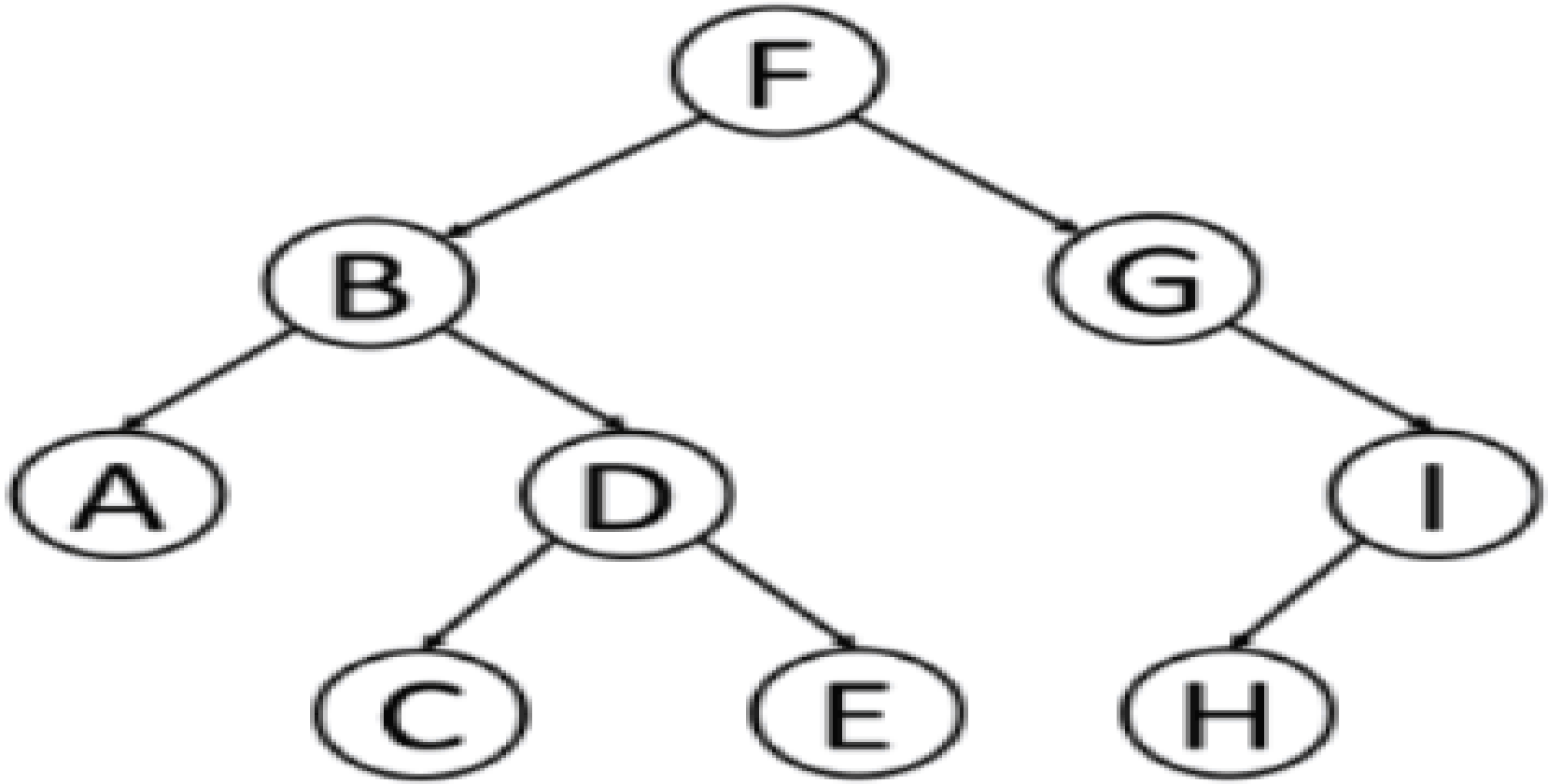
OUTPUT:


1
2
3
4
5
6
7
8
9
10

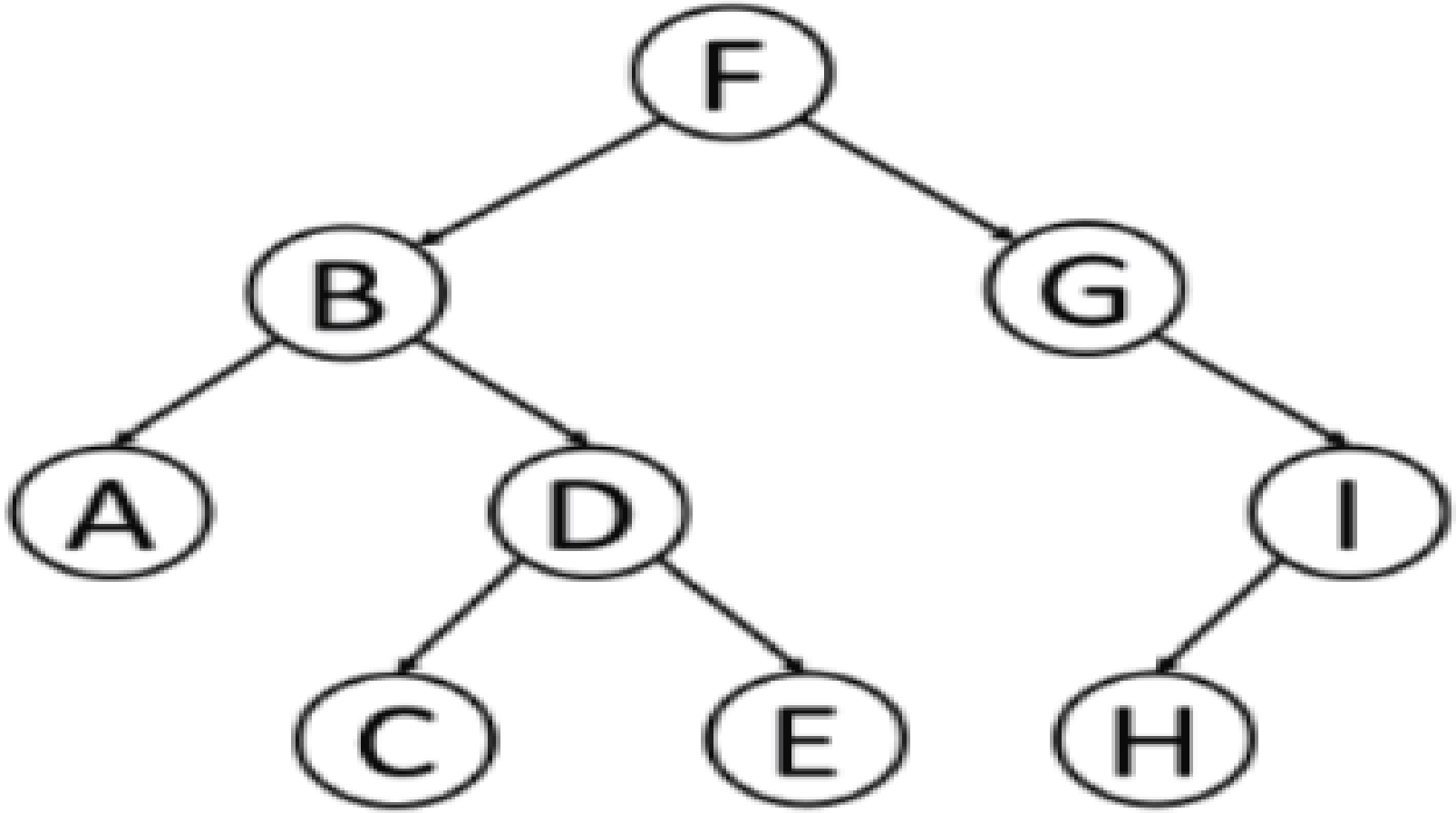

1
2
3
4
5
6
7
8
9
10



Inorder:



Inorder:



Postorder:

OPERATIONS ON TREES

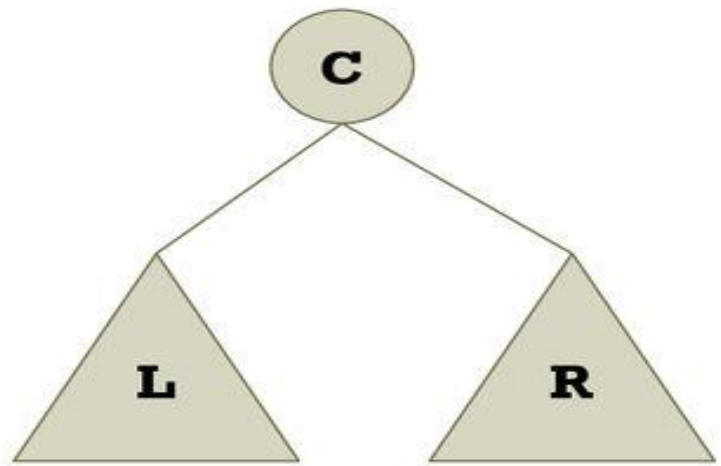
Traversing a Binary Tree

1)TRAVERSING

- You can implement various operations on a binary tree.
- A common operation on a binary tree is traversal.
- Traversal refers to the process of visiting all the nodes of a binary tree once.
- There are three ways for traversing a binary tree:
  - Inorder traversal
  - Preorder traversal
  - Postorder traversal

Traversal of Binary Tree

- Traversal methods
  - **Inorder traversal: LCR**
    - Visiting a left subtree, a root node, and a right subtree
  - **Preorder traversal: CLR**
    - Visiting the root node node before subtrees
  - **Postorder traversal: LRC**
    - Visiting subtrees before visiting the root node
  - **Level order traversal**



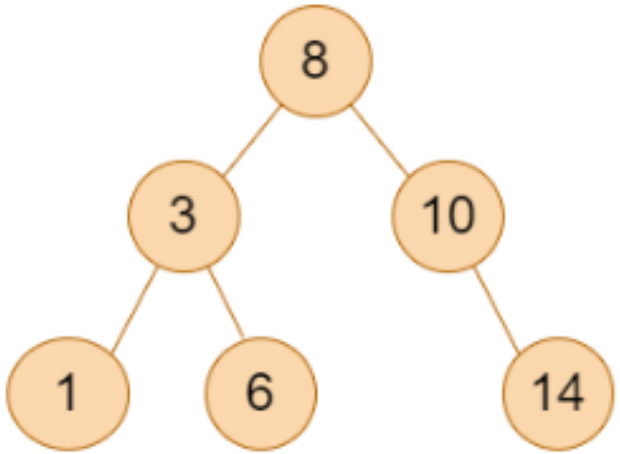
Some Special Types of Trees:

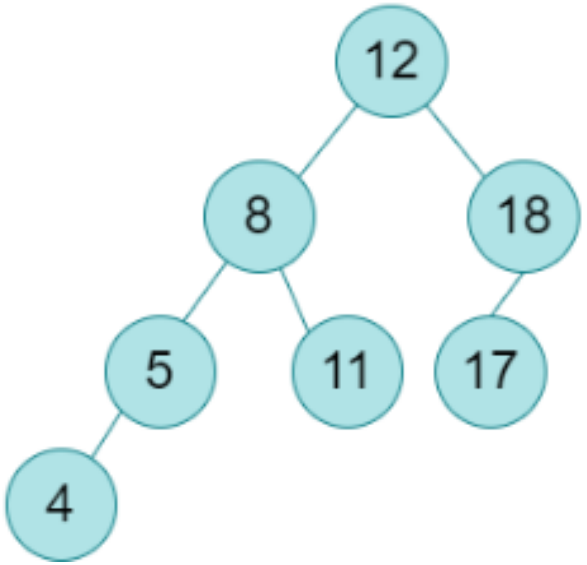
On the basis of node values, the Binary Tree can be classified into the following special types:

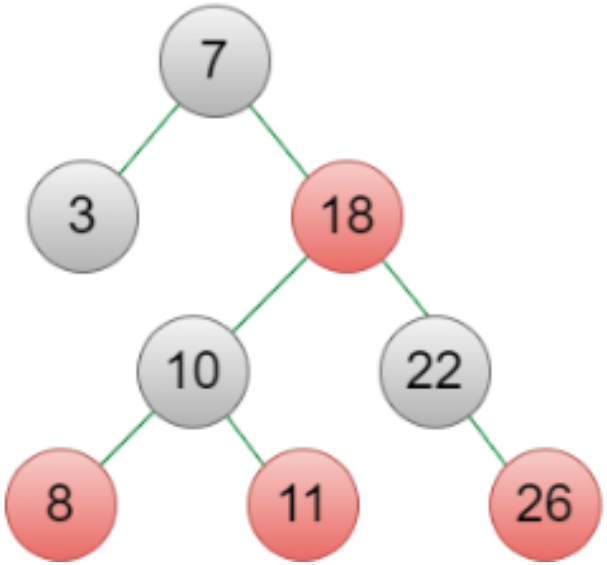
1. Binary Search Tree
2. AVL Tree
3. Red Black Tree
4. B Tree
5. B+ Tree
6. Segment Tree

Binary Tree Special Cases

- Binary Search Tree
- AVL Tree
- Red- Black Tree

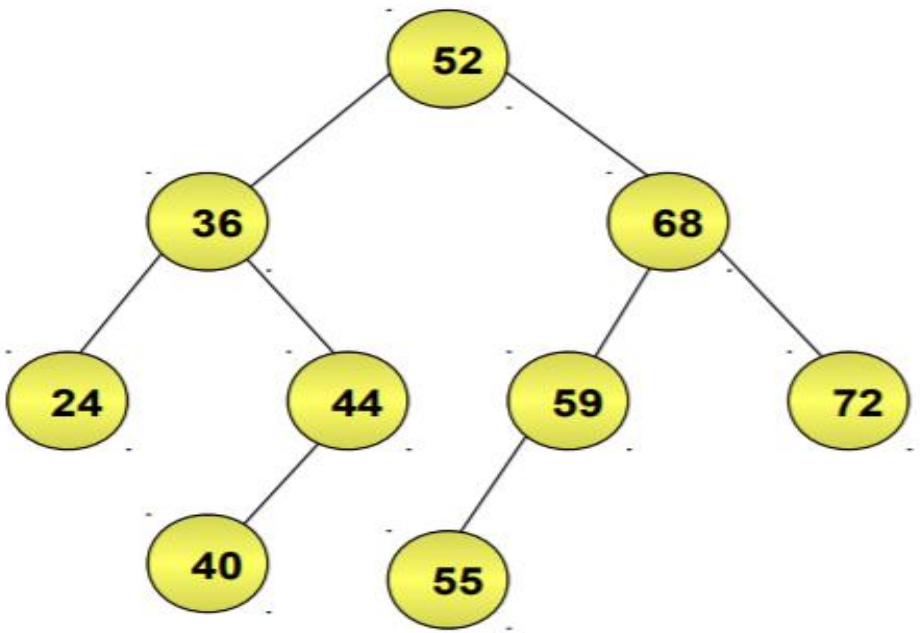






Binary Search Tree

- Binary search tree is a binary tree in which every node satisfies the following conditions:
  - All values in the left subtree of a node are less than the value of the node.
  - All values in the right subtree of a node are greater than the value of the node.
- The following is an example of a binary search tree.



Operations on a Binary Search Tree

- The following operations are performed on a binary earch tree...
  - Search
  - Insertion
  - Deletion
  - Traversal

Insertion of a key in a BST

```
Algorithm:- InsertBST (info, left, right, root, key, LOC)
{
   key is the value to be inserted.
   1. call SearchBST ( info, left, right, root, key, LOC , PAR )    // Find the parent of the new node
   2. If ( LOC != NULL)
   2.1 Print " Node alredy exist"
   2.2 Exit
   3. create a node [ new1 =  ( struct node*) malloc ( sizeof( struct node) ) ]
   4. new1 -> info = key
   5. new1 -> left = NULL , new1 -> right = NULL
   6. If ( PAR = NULL ) Then
   6.1 root = new1
   6.2 exit
     elseif ( new1 -> info < PAR -> info)
   6.1 PAR -> left = new1
   6.2 exit
     else
   6.1 PAR -> right = new1
   6.2 exit
 }
```

Deleting Nodes from a Binary Search Tree

- Write an algorithm to locate the position of the node to deleted from a binary search tree.

- Delete operation in a binary search tree refers to the process of deleting the specified node from the tree.
- Before implementing a delete operation, you first need to locate the position of the node to be deleted and its parent.
- To locate the position of the node to be deleted and its parent, you need to implement a search operation.

Deleting Nodes from a Binary Search Tree (Contd.)

- Once the nodes are located, there can be three cases:
  - **Case I**: Node to be deleted is the leaf node
  - **Case II**: Node to be deleted has one child (left or right)
  - **Case III**: Node to be deleted has two children

Deletion of a key from a BST

```
Algorithm:- Delete1BST (info, left, right, root, LOC, PAR)
            // When leaf node has no child or only one child
{
   1.  if ( ( LOC -> left = NULL)  and  ( LOC -> right = NULL))
          1.1 Child = NULL
       elseIf ( LOC -> left != NULL)
          1.1 Child = LOC -> left
       else
          1.1 Child = LOC -> right
   2. if ( PAR != NULL)
          2.1   if ( LOC = PAR -> left )
                     2.1.1 PAR -> left = Child
          2.1   else
                     2.1.1 PAR -> right = Child
       else
          2.1 root = Child
 }
```