SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

23ITT101-PROGRAMMING IN C AND DATA STRUCTURES

I YEAR - II SEM

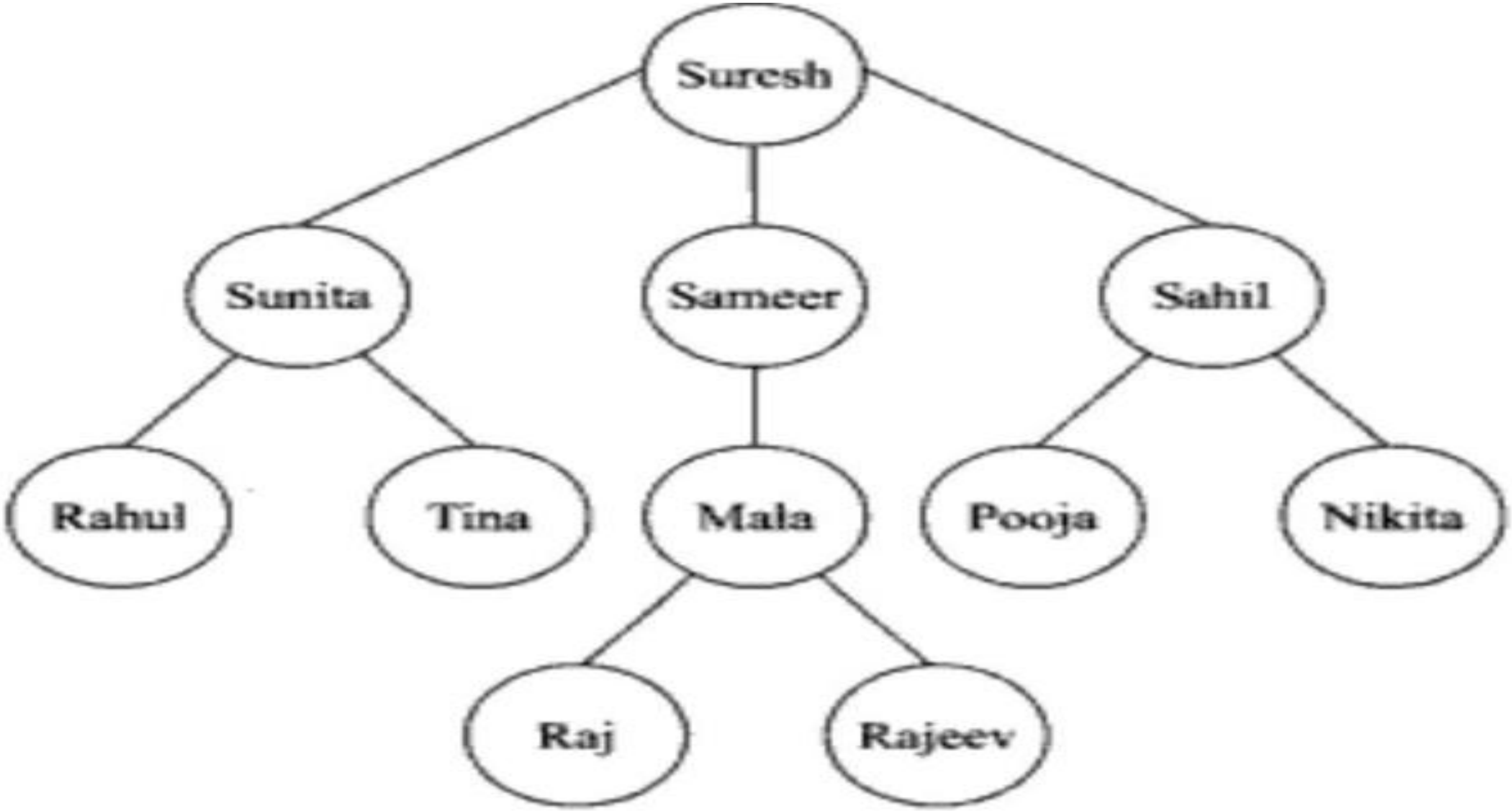
UNIT-V

Trees

Non-Linear Data Structure

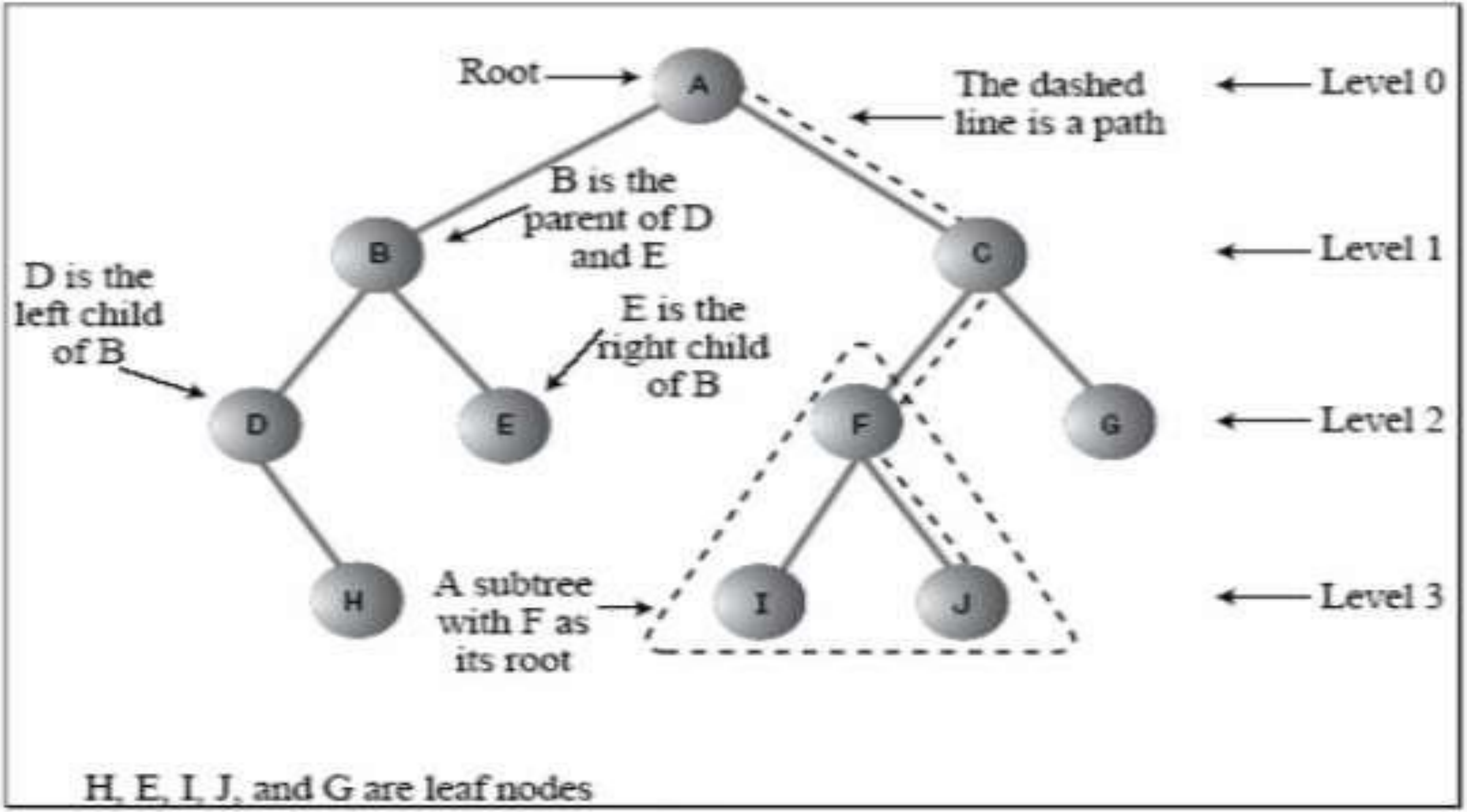
- Data are **not arranged sequentially** or linearly are called **non-linear data structures**
- It Represents data in **hierarchical relationship**
- **Example** : **Graph and Tree**

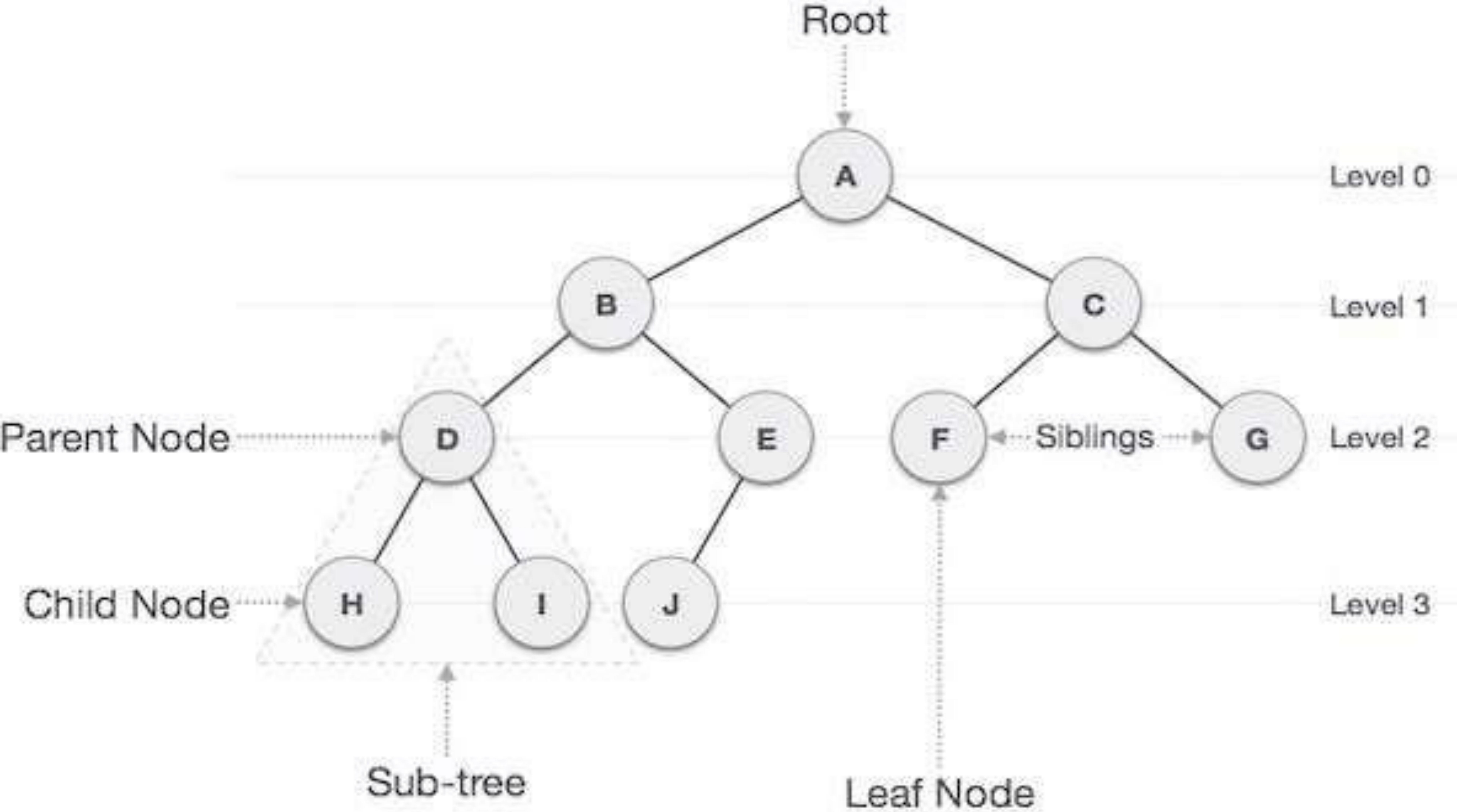
Representation of Tree



Tree

- A tree is an abstract model of a hierarchical structure that consists of nodes with a parent-child relationship
  - There is a starting node known as a **root node**
  - Every node other than the root has a **parent node.**
  - Nodes may have any number of children


Root
A
The dashed line is a path
Level 0
B is the parent of D and E
B
C
Level 1
D is the left child of B
E is the right child of B
D
E
F
G
Level 2
H
A subtree with F as its root
I
J
Level 3
H, E, I, J, and G are leaf nodes



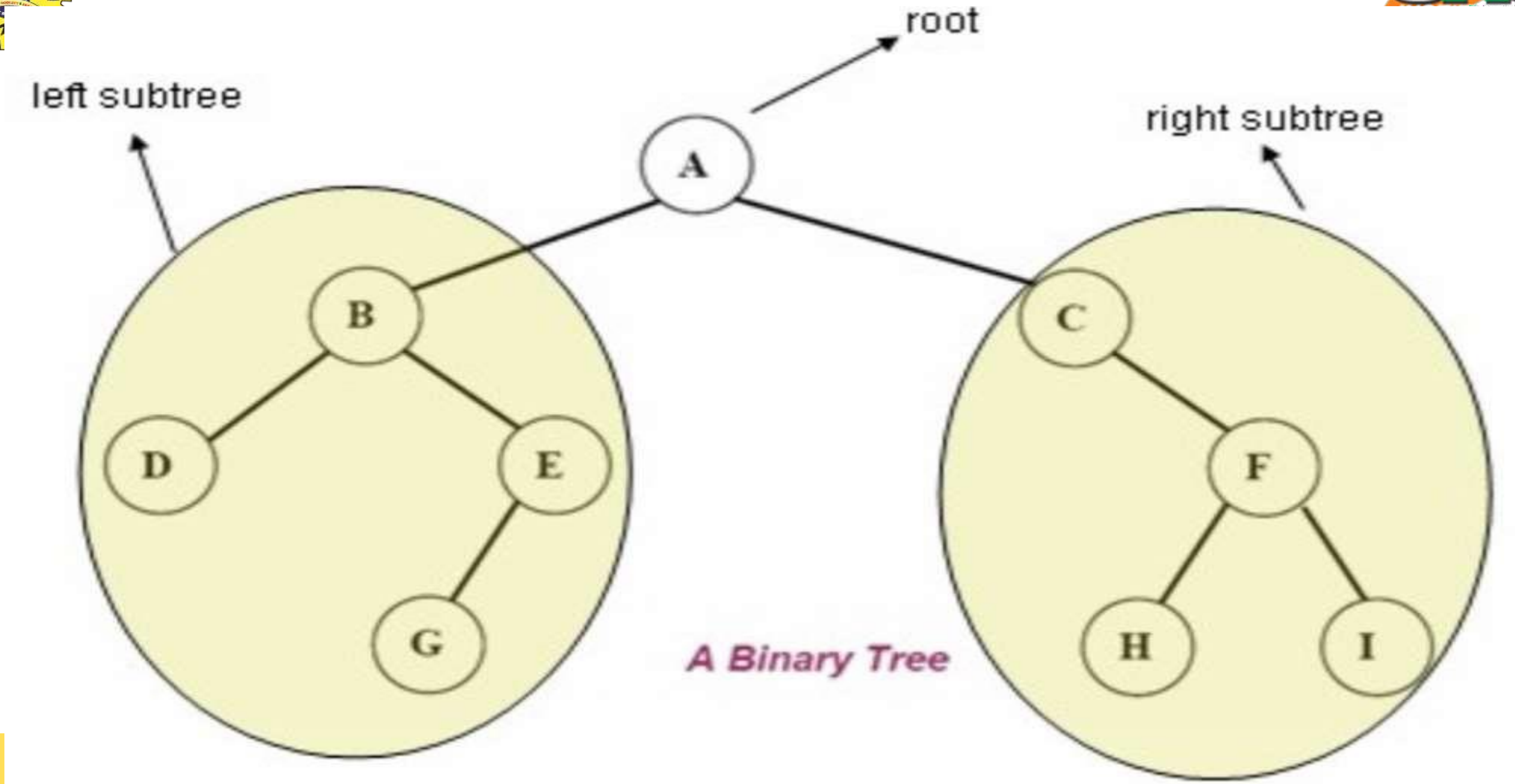
Root
A
Level 0
B
C
Level 1
Parent Node
D
E
F
Siblings
G
Level 2
Child Node
H
I
J
Level 3
Sub-tree
Leaf Node

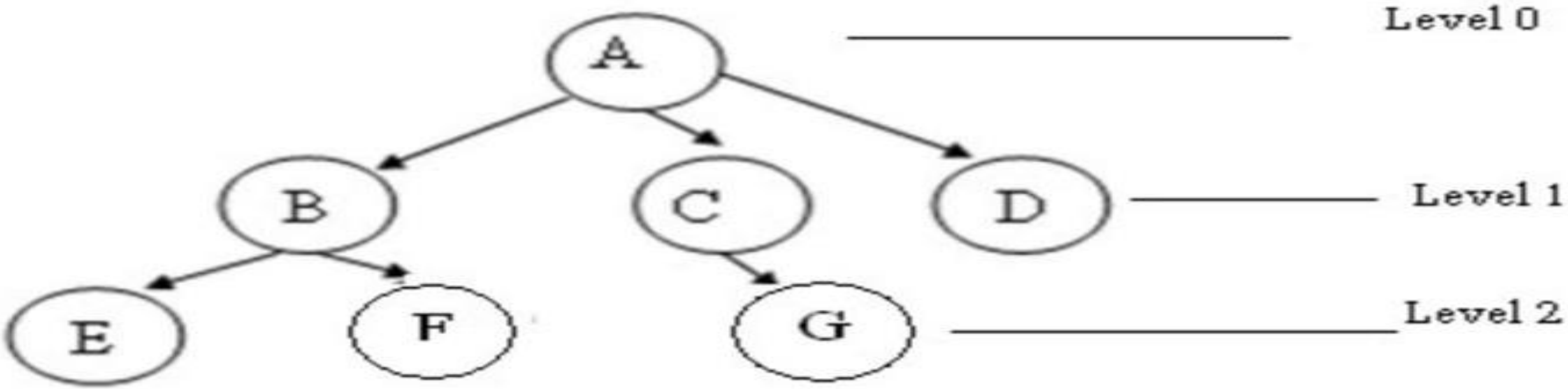
Some Key Terms:

- **Root** – Node at the top of the tree is called root
- **Parent** – Node that has child except root called parent
- **Child** – Node connected to parent is called child node
- **Sibling** – Child of same node are called siblings
- **Leaf** – Node which does not have any child node is called leaf node
- **Sub tree** – Sub tree represents part of a tree

- **Degree of a node:** Number of children of that node
- **Degree of a Tree:** Maximum degree of nodes in a given tree
- **Path:** Sequence of consecutive edges from source node to destination node
- **Height of a node:** The height of a node is the max path length form that node to a leaf node
- **Height of a tree:** The height of a tree is the height of the root. (level+1)
- **Depth/ Level of a tree:** Number of connections between the node and the root

Subtree





- A is the root node
- B is the parent of E and F
- D is the sibling of B and C
- E and F are children of B
- E, F, G, D are external nodes or leaves
- A, B, C are internal nodes
- Depth of F is 2
- the height of tree is 3
- the degree of node A is 3
- The degree of tree is 3

Application

- Directory structure of a file store
- Structure of an arithmetic expressions
- Used in router for storing router-tables.

Introduction To Binary Trees

- A binary tree, is a tree in which **no node can have more than two children**
- Consider a binary tree T, here 'A' is the root node of the binary tree T
- 'B' is the left child of 'A' and 'C' is the right child of 'A'
  - i.e A is a father of B and C.
  - The node B and C are called siblings.
- Nodes D,H,I,F,J are leaf node

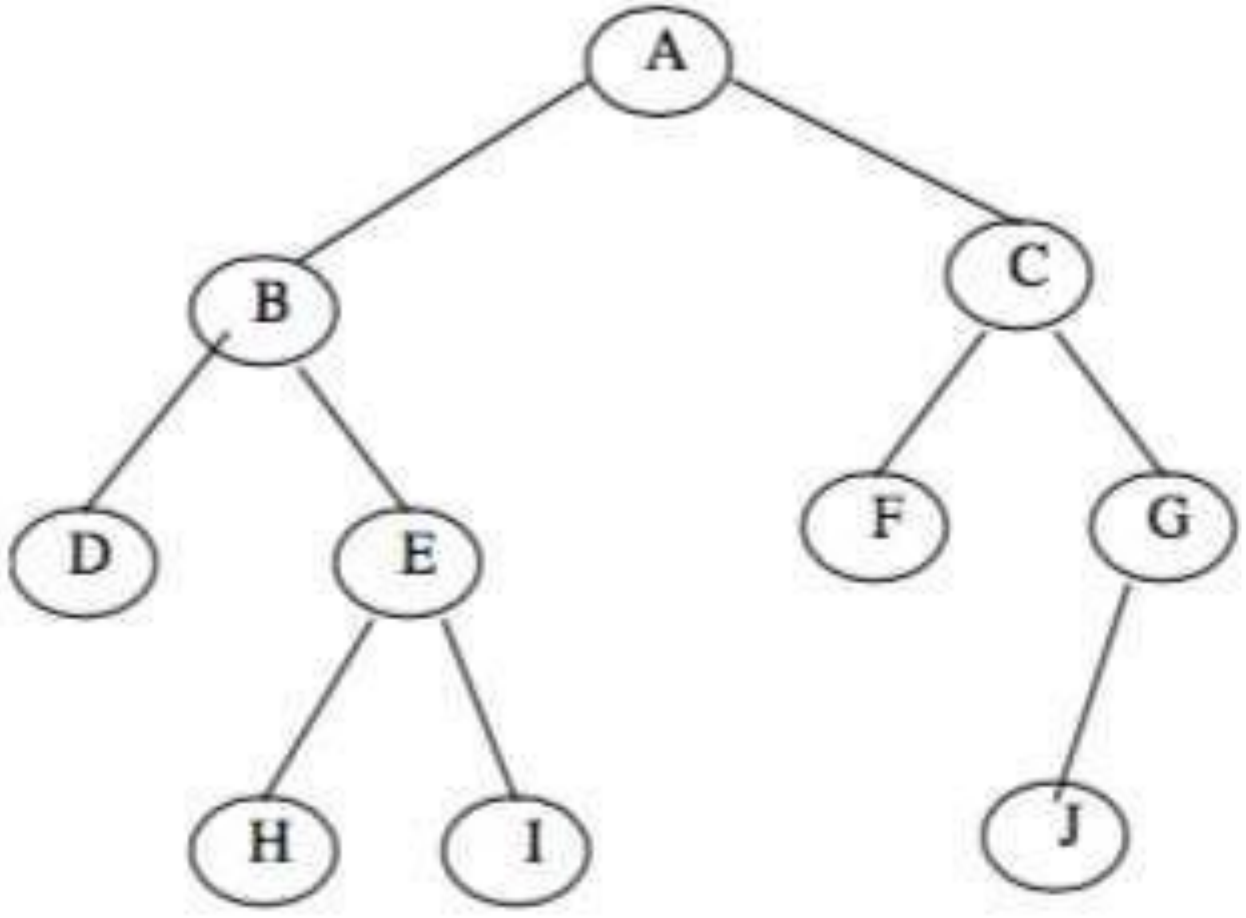
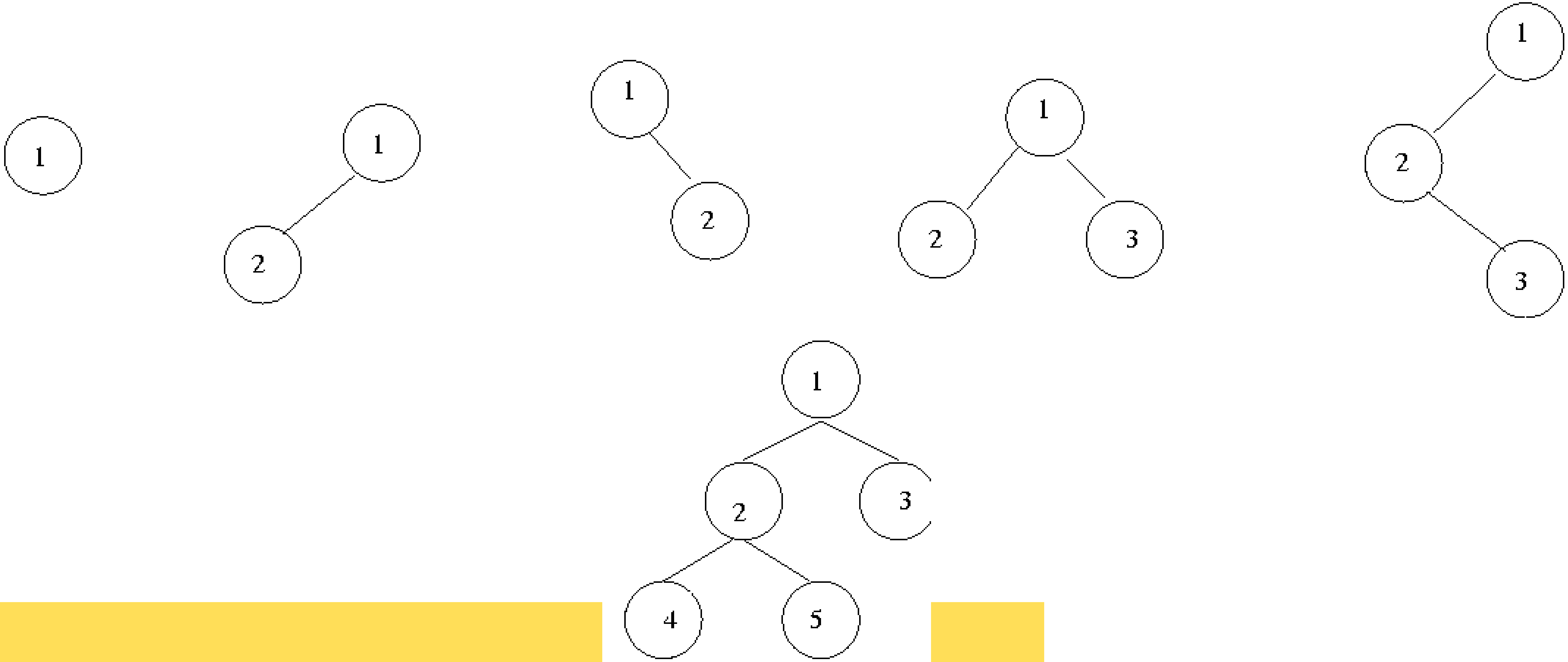


Fig. 8.3. Binary tree

Definition Binary tree

Binary tree is a tree in **which each node contains atmost two children**. In a **binary tree**, nodes are organized as either left or right child



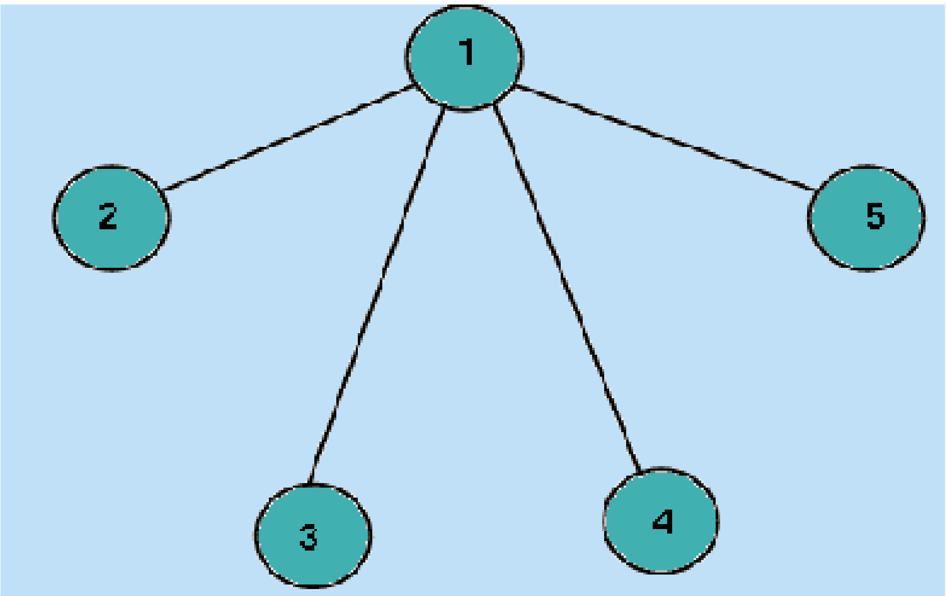
Binary Tree Properties

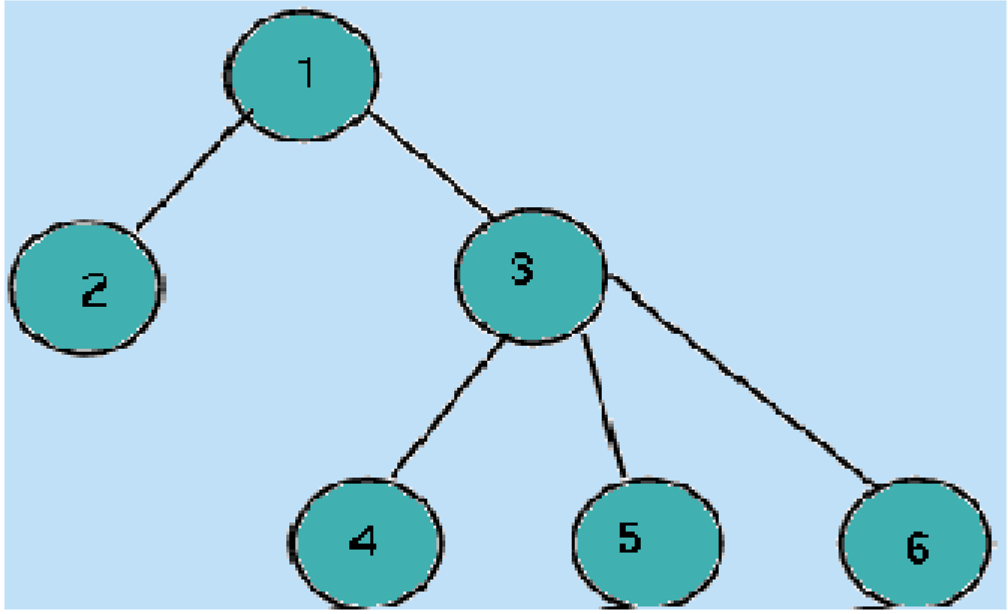
- If a binary tree contains m nodes at level L, it contains at most 2m nodes at level L+1

- Since a binary tree can contain at most 1 node at level 0 (the root), it contains at most 2L nodes at level L

Non Binary tree

Non-binary **tree** is a **tree** in which **at least one node has more than two children**



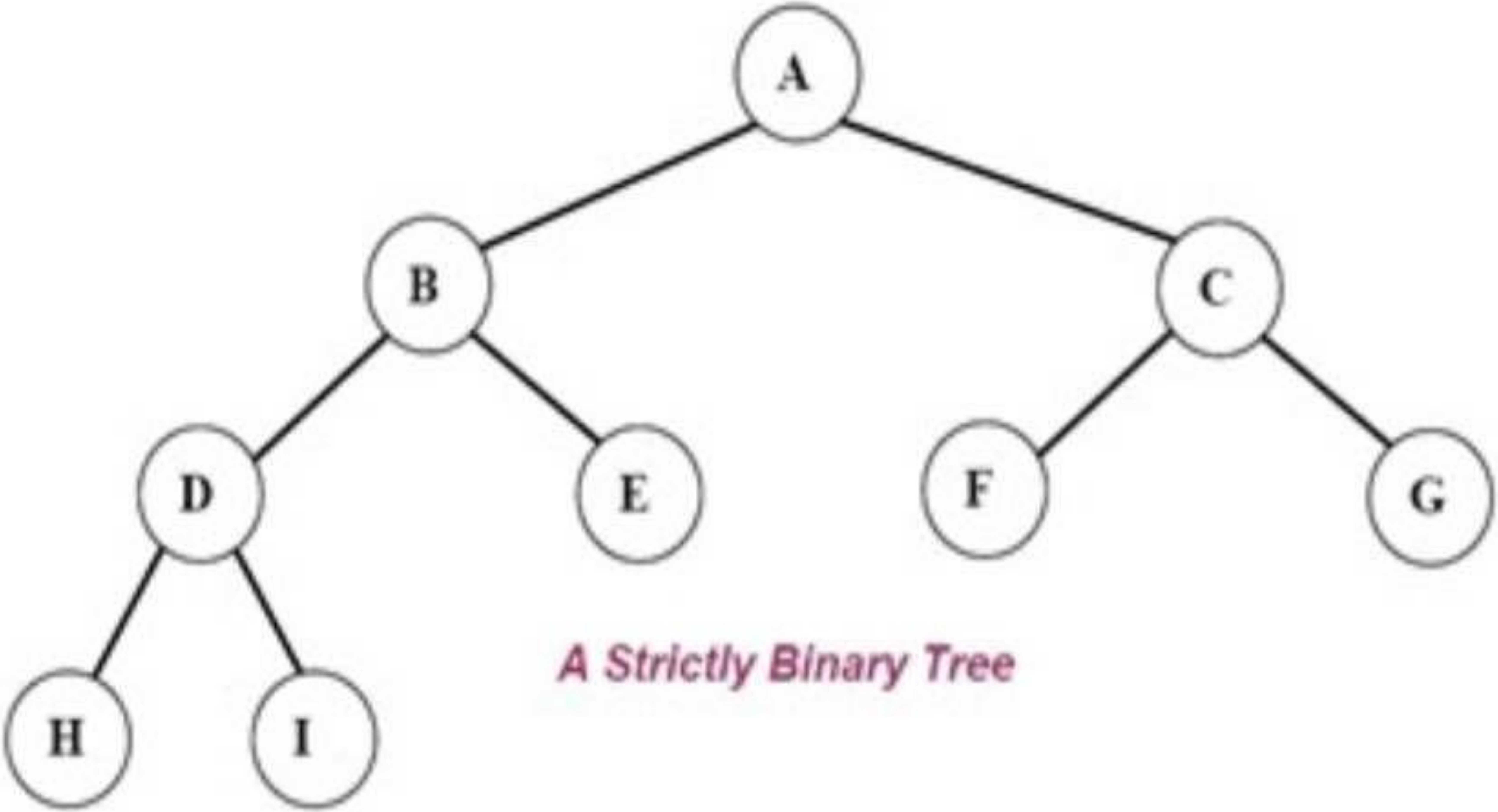


Types of Binary Tree

- Strictly binary tree
- Complete binary tree
- Almost complete binary tree
- Binary Search Tree
- Heap Tree
    - Max Heap Tree
    - Min Heap Tree

Strictly binary tree

- If every non-leaf node in a binary tree has nonempty left and right sub-trees, then such a tree is called a strictly binary tree
- Or, to put it another way, all of the nodes in a strictly **binary tree are of degree zero or two, never degree one**
- A strictly binary tree with N leaves always contains 2N – 1 nodes



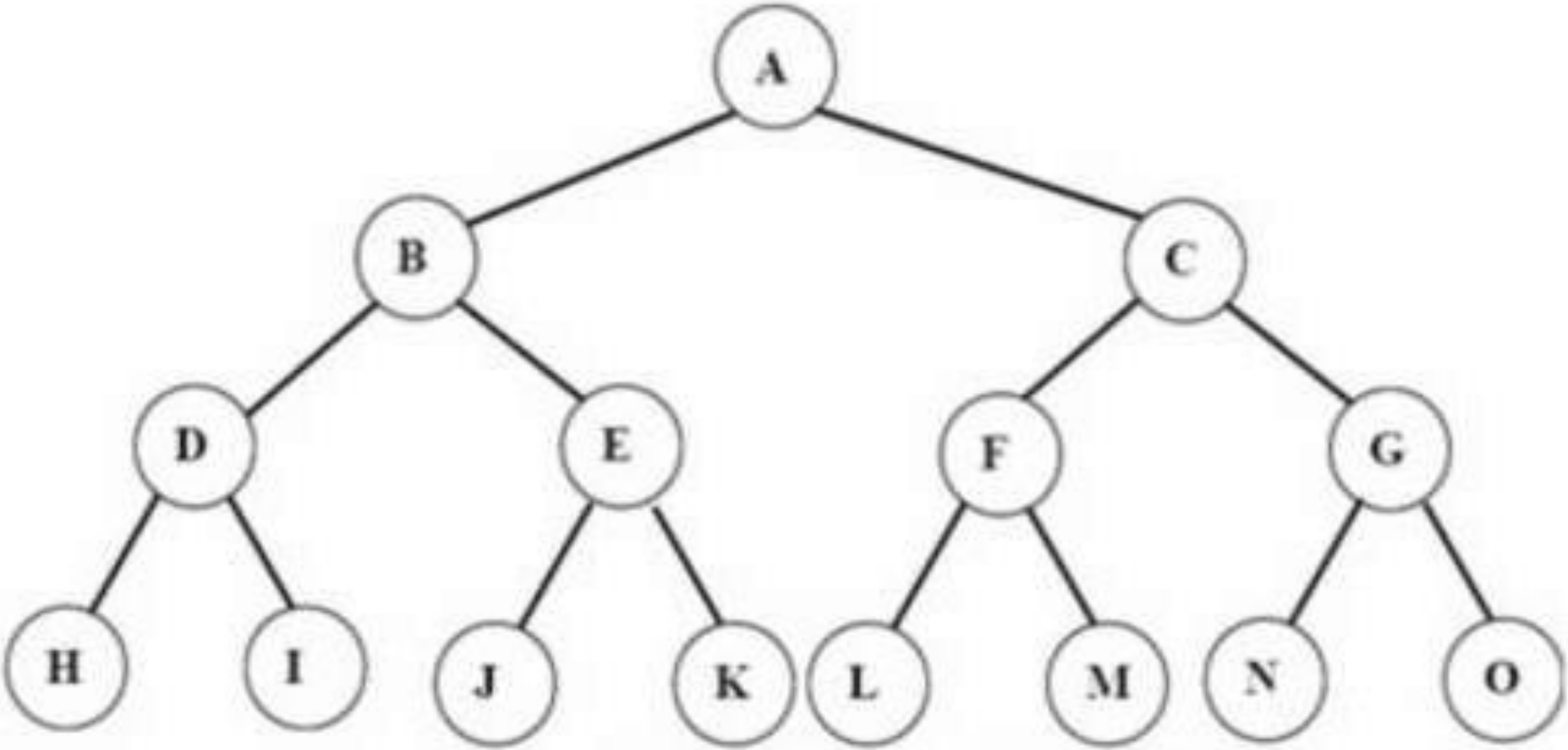
A Strictly Binary Tree

Complete /Full Binary Tree

- A complete binary tree is a binary tree in which every level, except possibly the last, is **completely filled,** and all nodes are as far left as possible

- A complete binary tree of depth d is called strictly binary tree if all of whose leaves are at level d

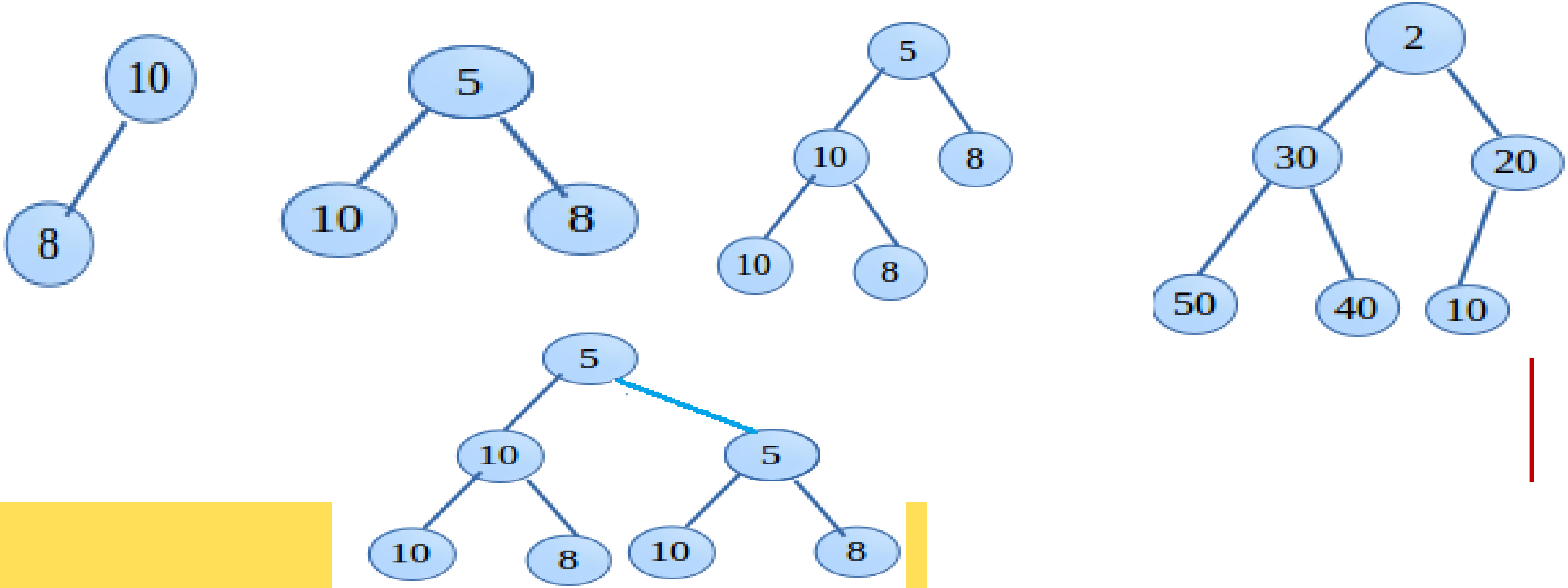
- A complete binary tree has 2d nodes at every depth d and 2d -1 non leaf nodes



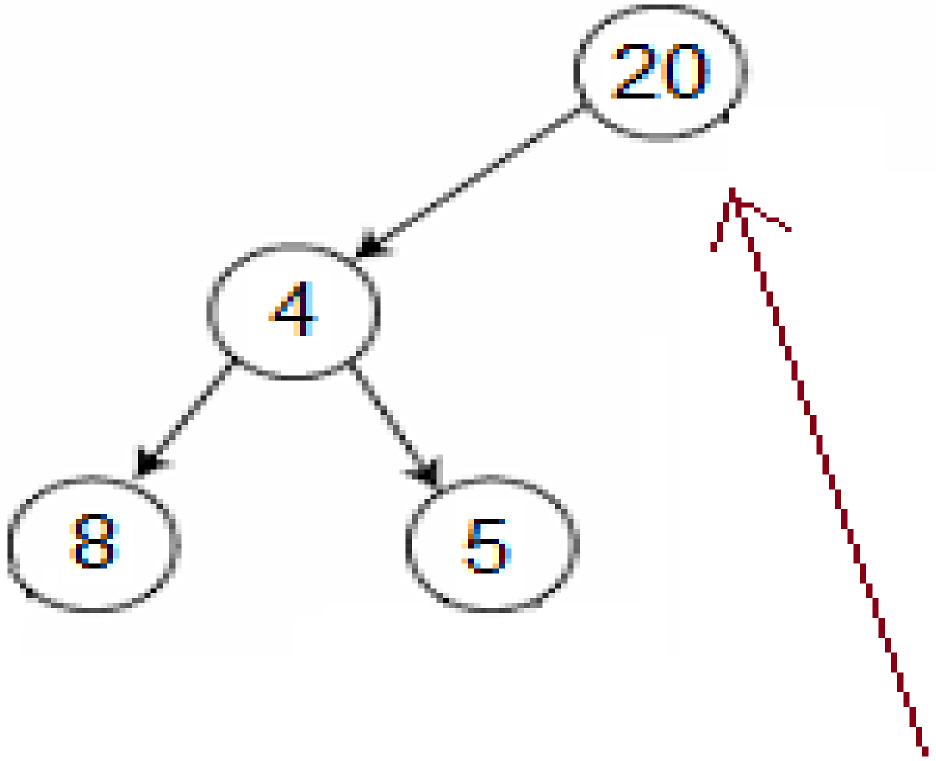
A complete Binary Tree of depth 3

Almost Complete Binary Tree

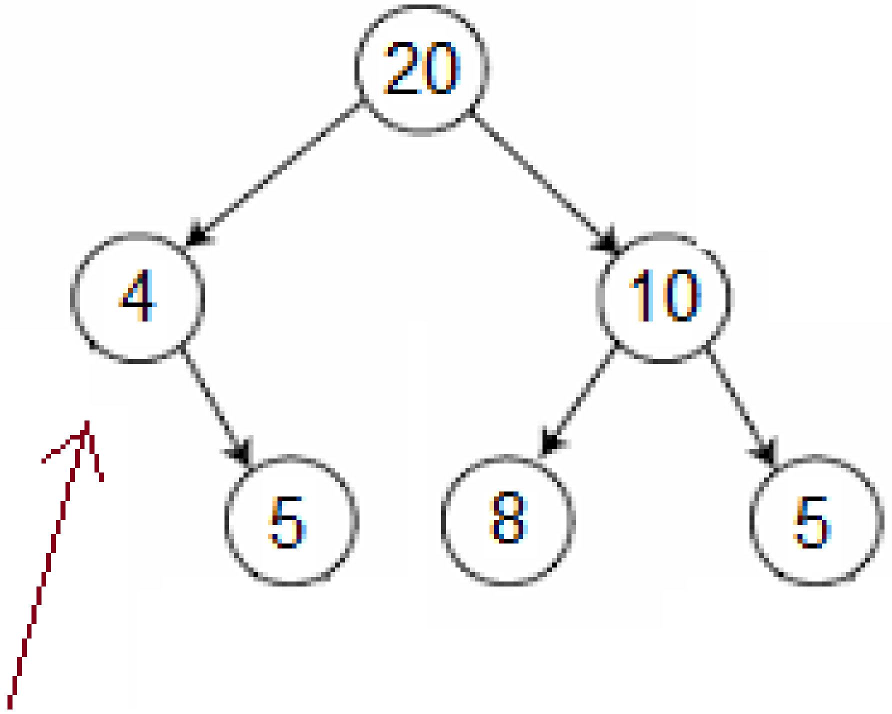
Complete binary tree in **which each node contains atmost of two children** and All **levels are completely filled except possibly the last level**, and all nodes are as far left as possible



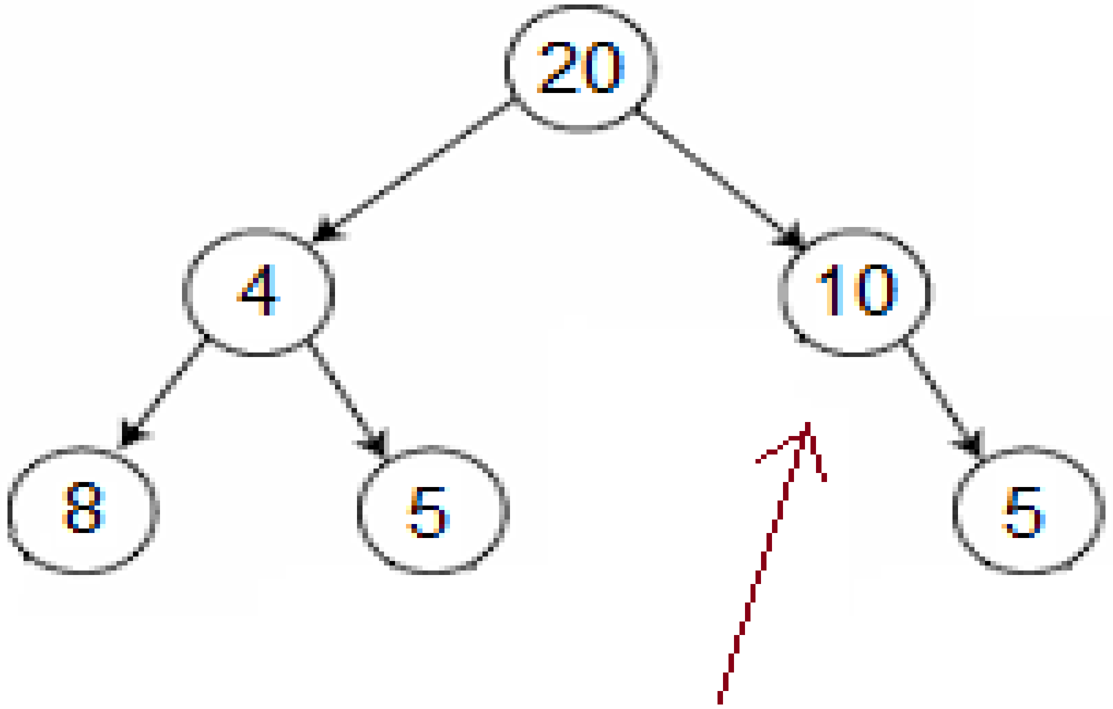
Non- Complete Binary Tree



Childs of Node 4 is
present without
right child of Node 2



Left child of Node 4 is not present
while right child of Node 4 is
present.

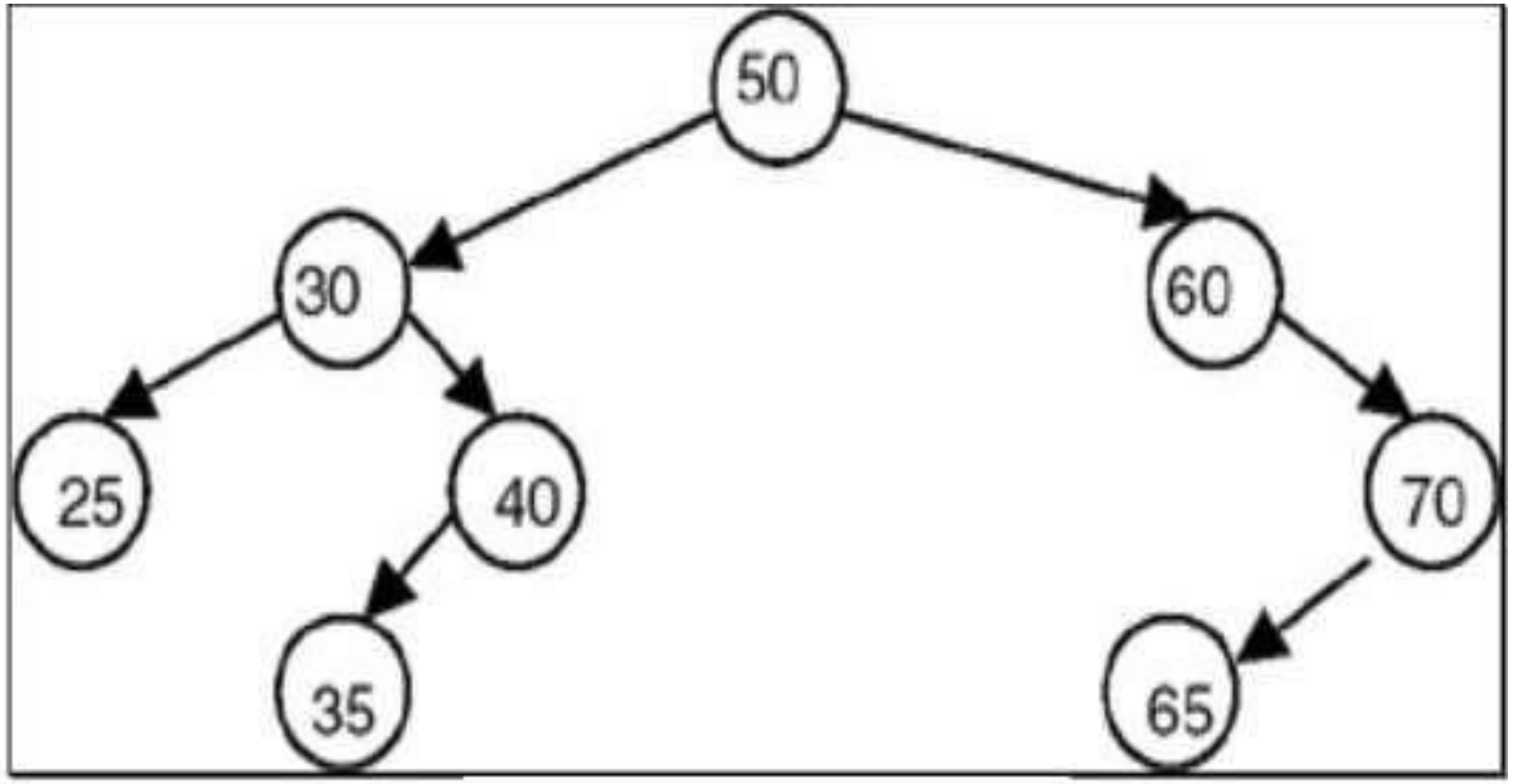


Left child of Node 10 is not present
while right child is present.

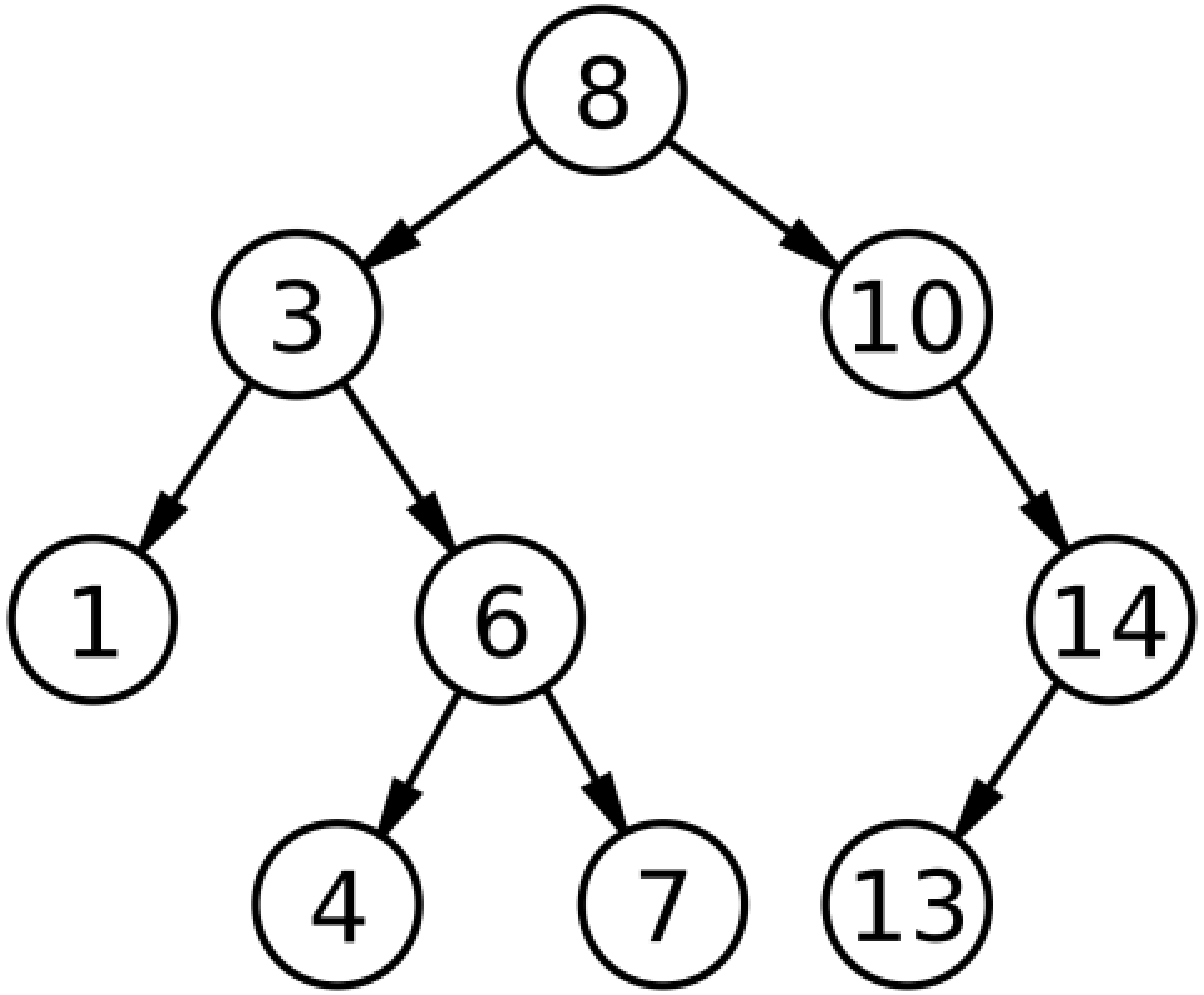
Binary Search Tree(BST)

- A Binary search tree (BST) is a binary tree that is either empty or in which every node contains a key (value) and satisfies the following conditions:
  - All keys in the the **left sub-tree of the root are smaller than the key** in the root node
  - All keys in the **right sub-tree of the root are greater than the key** in the root node
  - The left and right sub-trees of the root are again binary search trees

Binary Search Tree(BST)

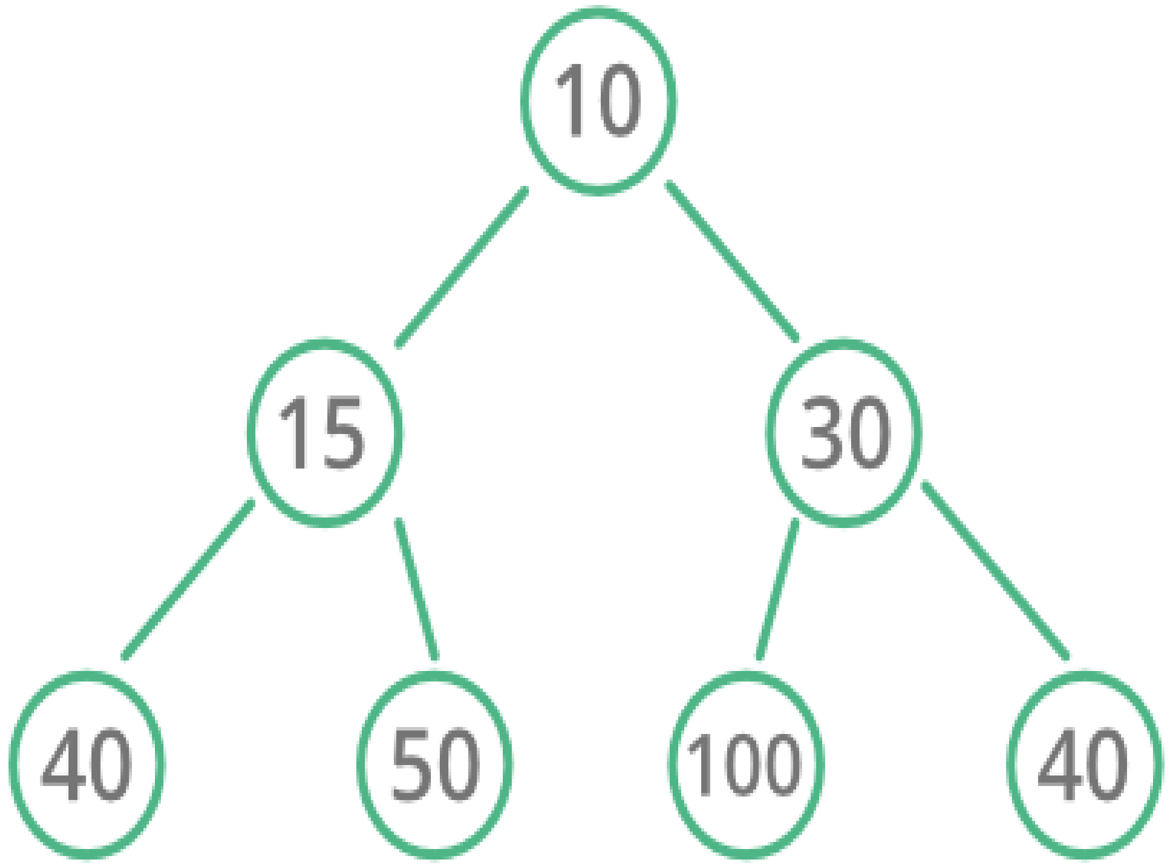


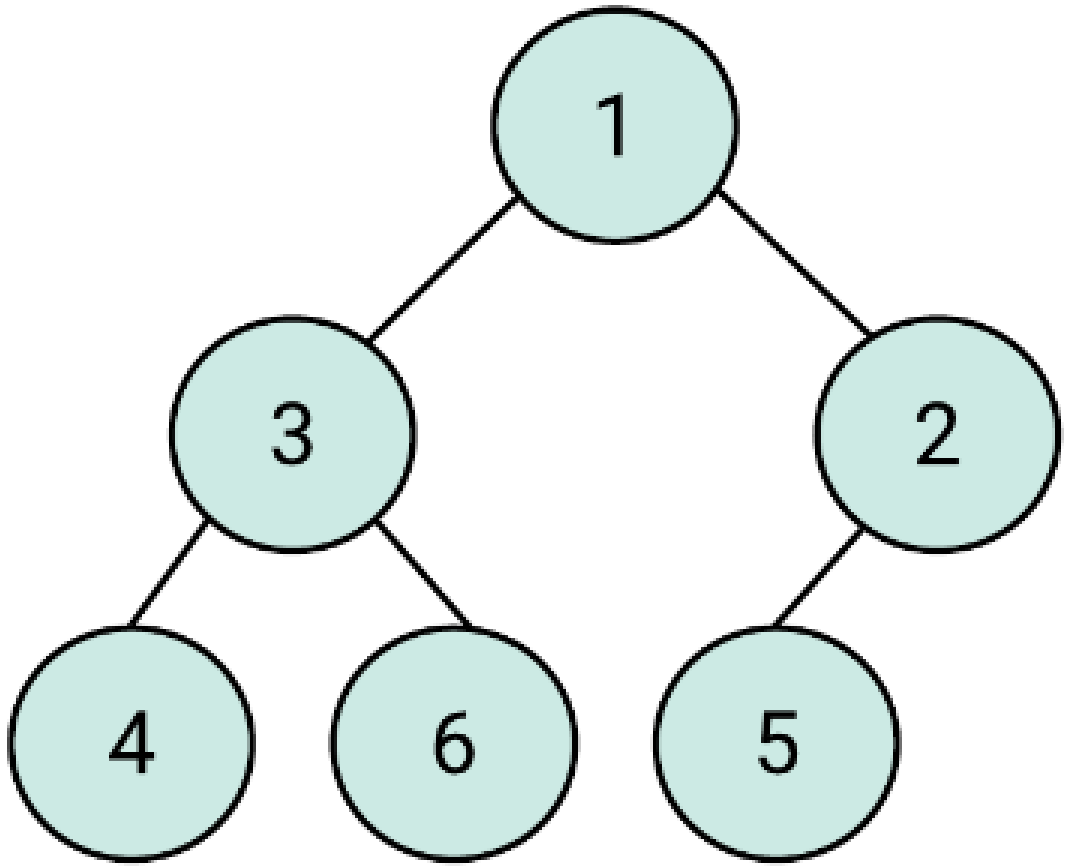
The binary search tree.

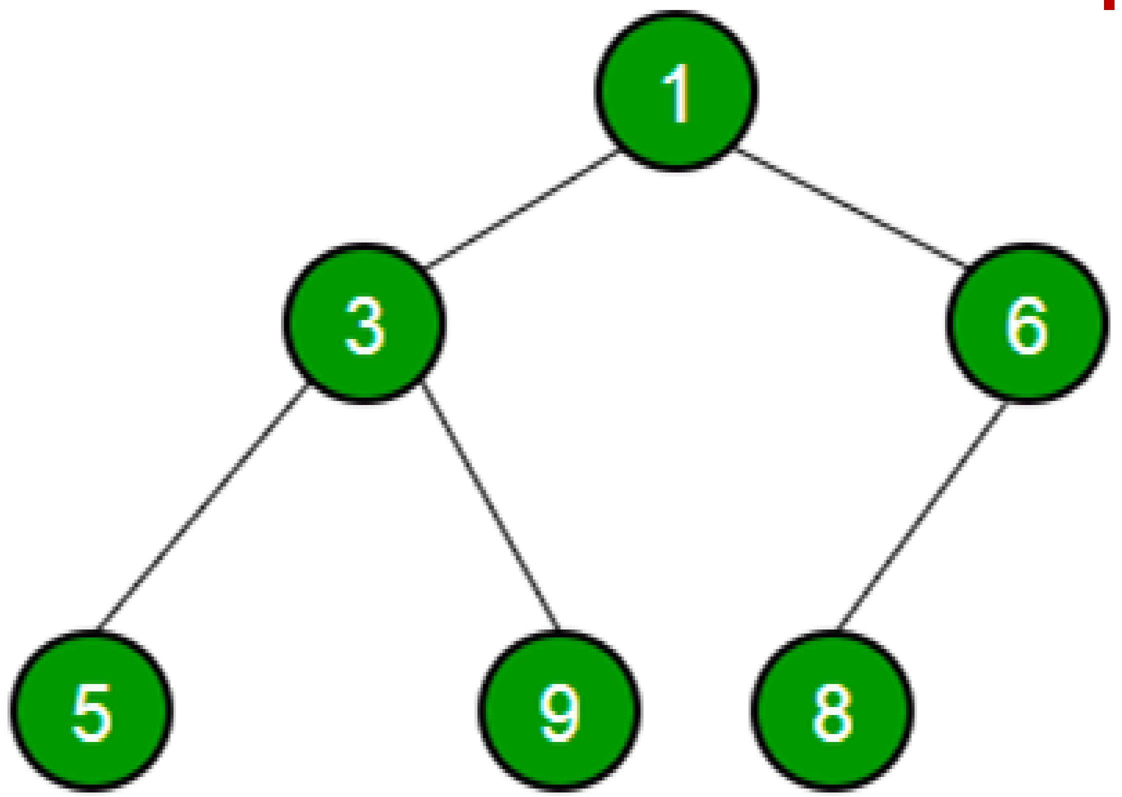


Min Heap

In a Min Binary Heap, the key at **root must be minimum** among all keys present in Binary Heap. The same property must be recursively **true for all nodes** in Binary Tree

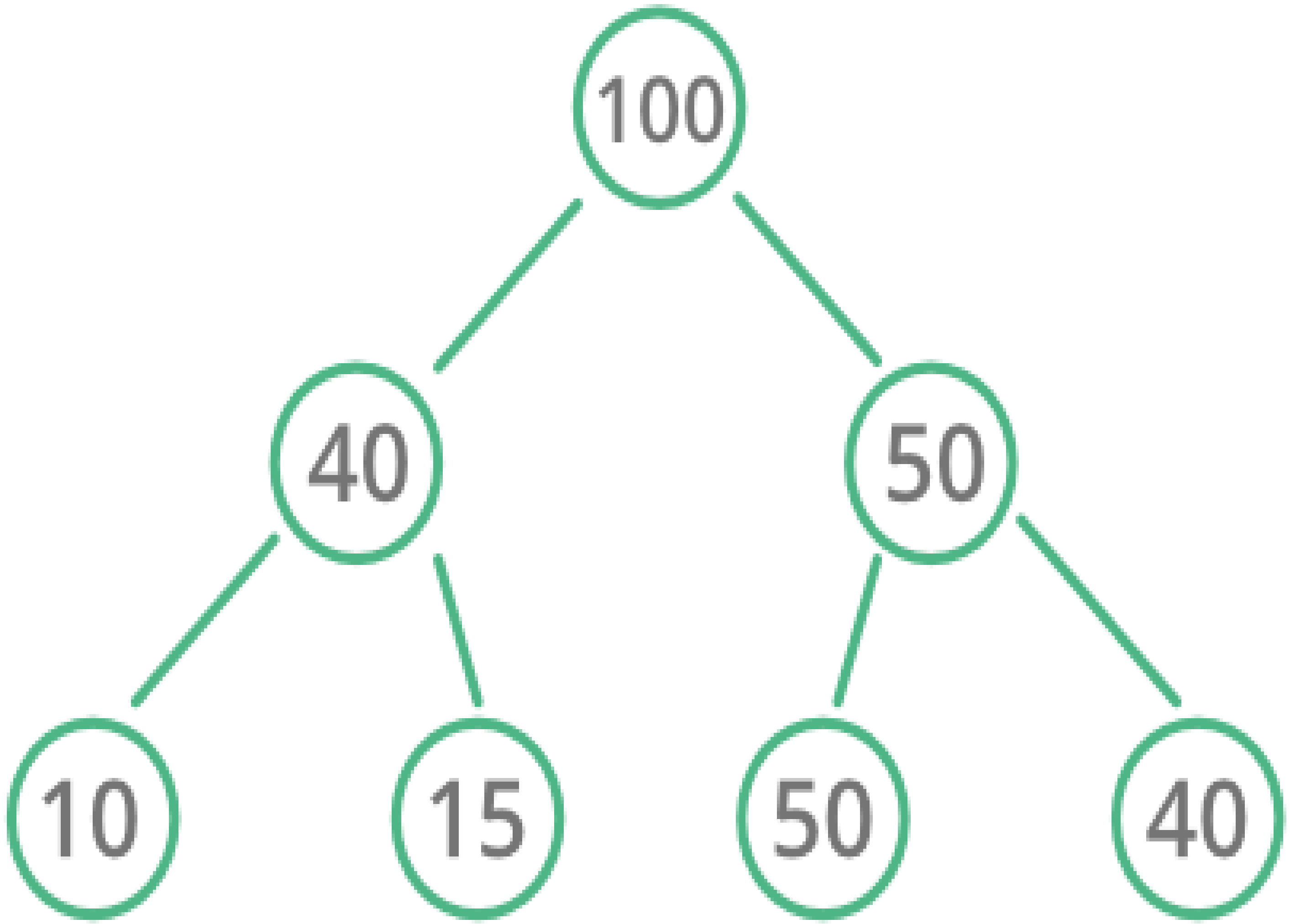


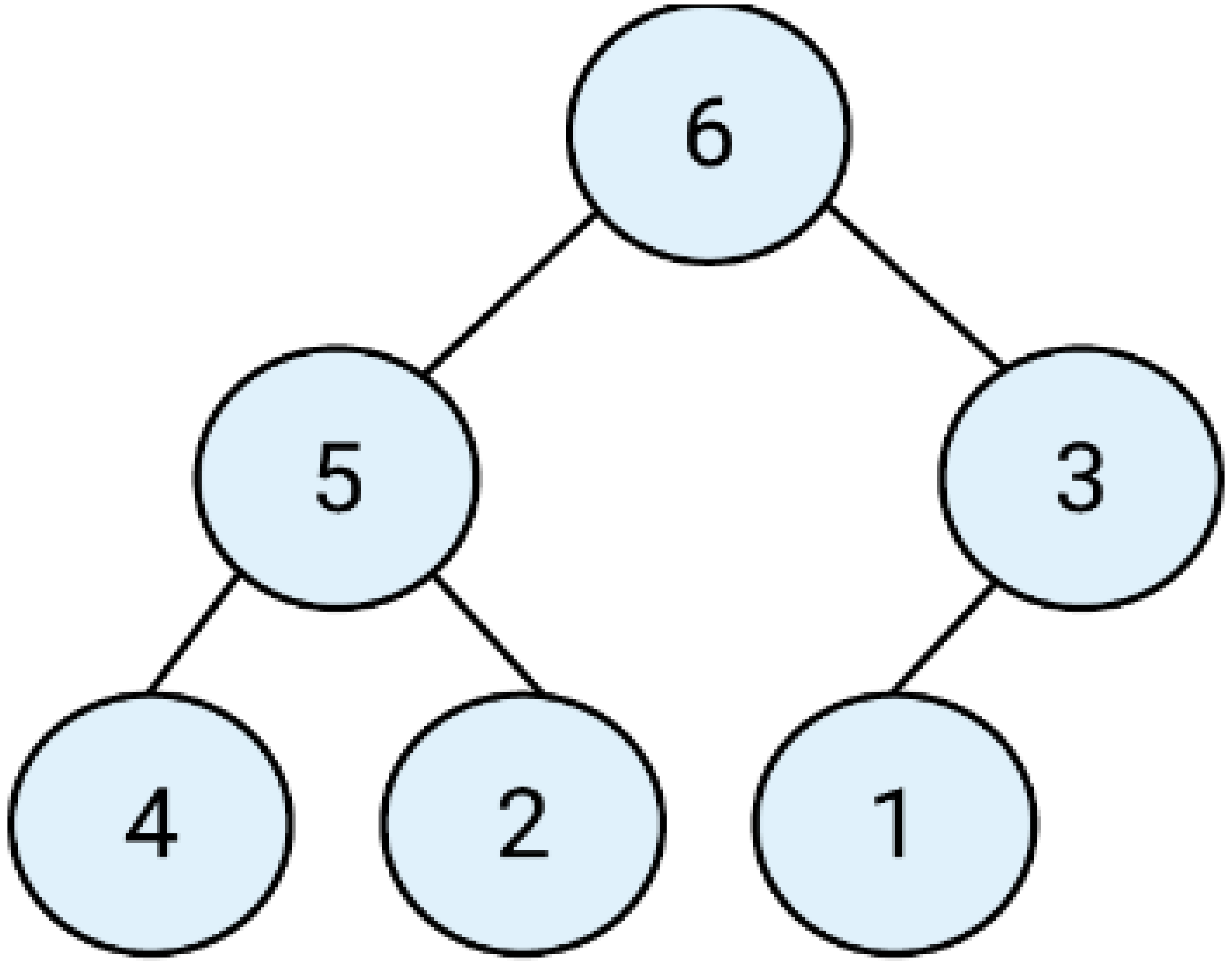




Max Heap

In a Max Binary Heap, the key at **root must be maximum among** all keys present in Binary Heap. The same property must be recursively **true for all nodes in Binary Tree**





Tree Traversals

(Traversing a tree means visiting every node in the tree exactly once. Displaying (or) visiting order of nodes in a binary tree is called as Binary Tree Traversal.)

Types of Tree Traversal

Depth First Traversals:

(a) Inorder (Left, Root, Right)

(b) Preorder (Root, Left, Right)

(c) Postorder (Left, Right, Root)

Breadth First or Level Order Traversal

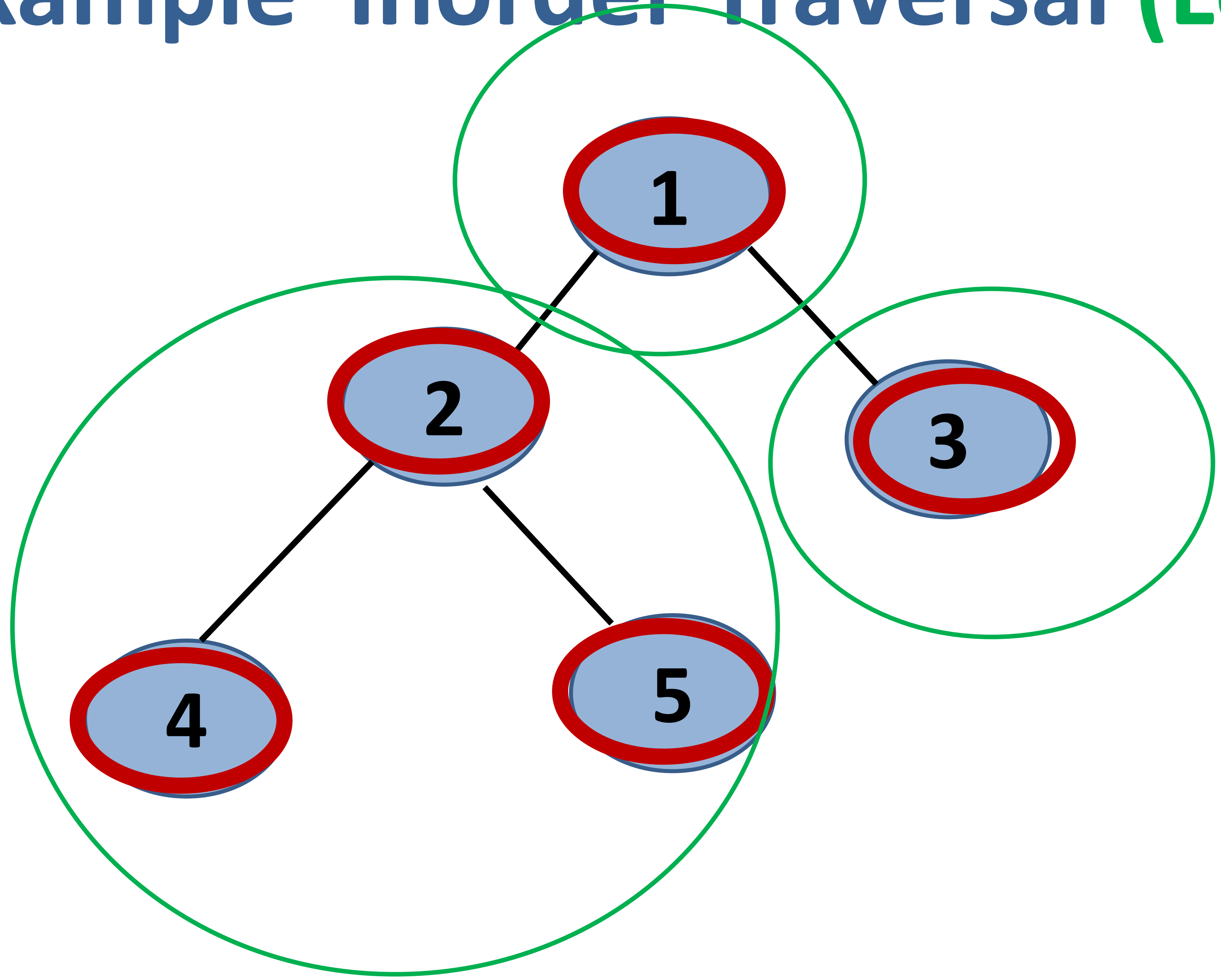
Inorder Traversal (Left, Root, Right)

Inorder Traversal (Left, Root, Right) Algorithm

Inorder traversal of a binary tree is defined as follow

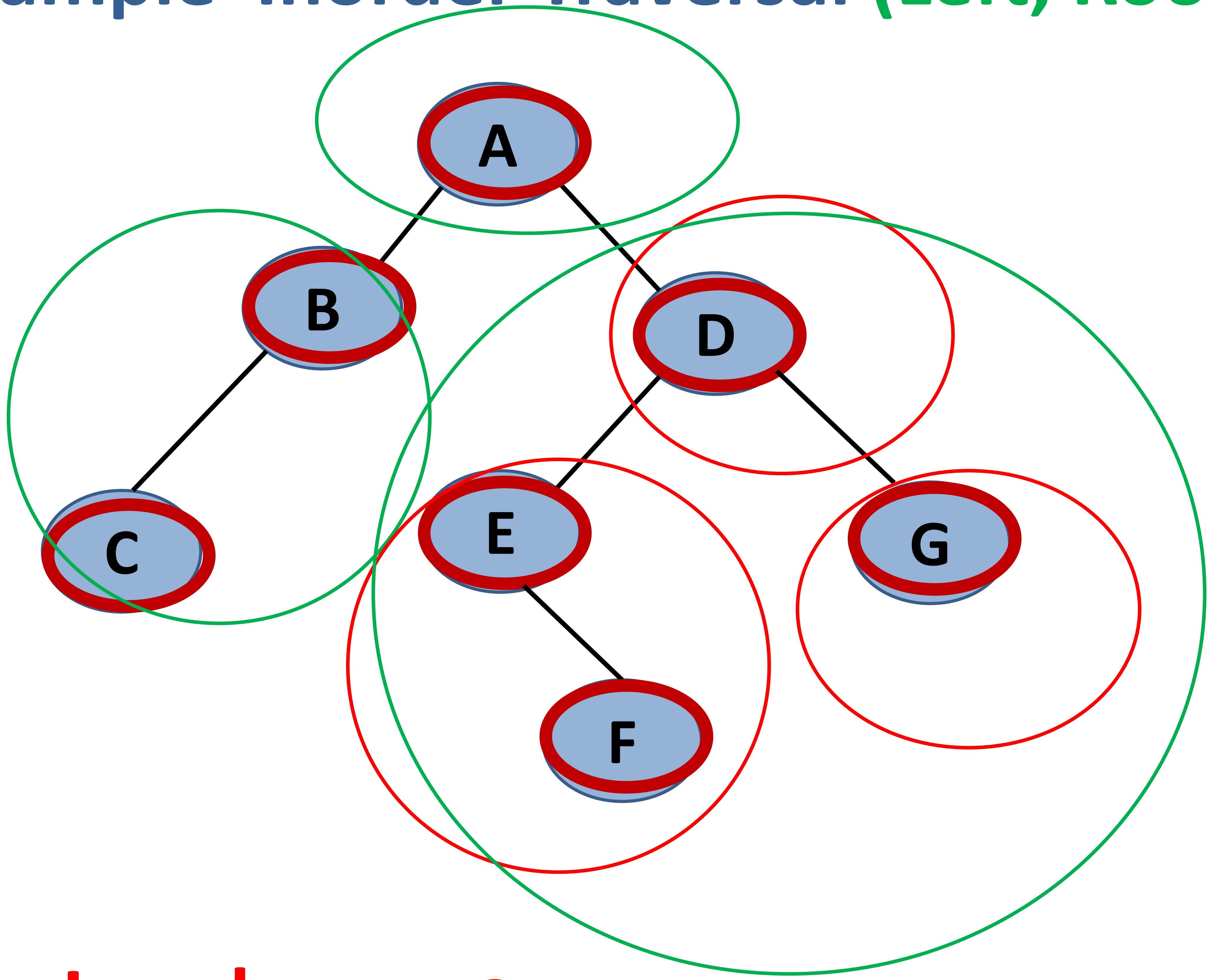
1. Traverse the left subtree, i.e., call Inorder (left-subtree)
2. Visit the root
3. Traverse the right subtree, i.e., call Inorder (right-subtree)

Example Inorder Traversal (Left, Root, Right)



Inorder : 4 2 5 1 3

Example Inorder Traversal (Left, Root, Right)



Inorder : C B A E F D G

Psedocode for Inorder Traversal

```
inOrder(treePointer ptr)
    {
        if (ptr != NULL)
        {
        inOrder(ptr->leftChild);
        visit(ptr);
        inOrder(ptr->rightChild);
        }
    }
```

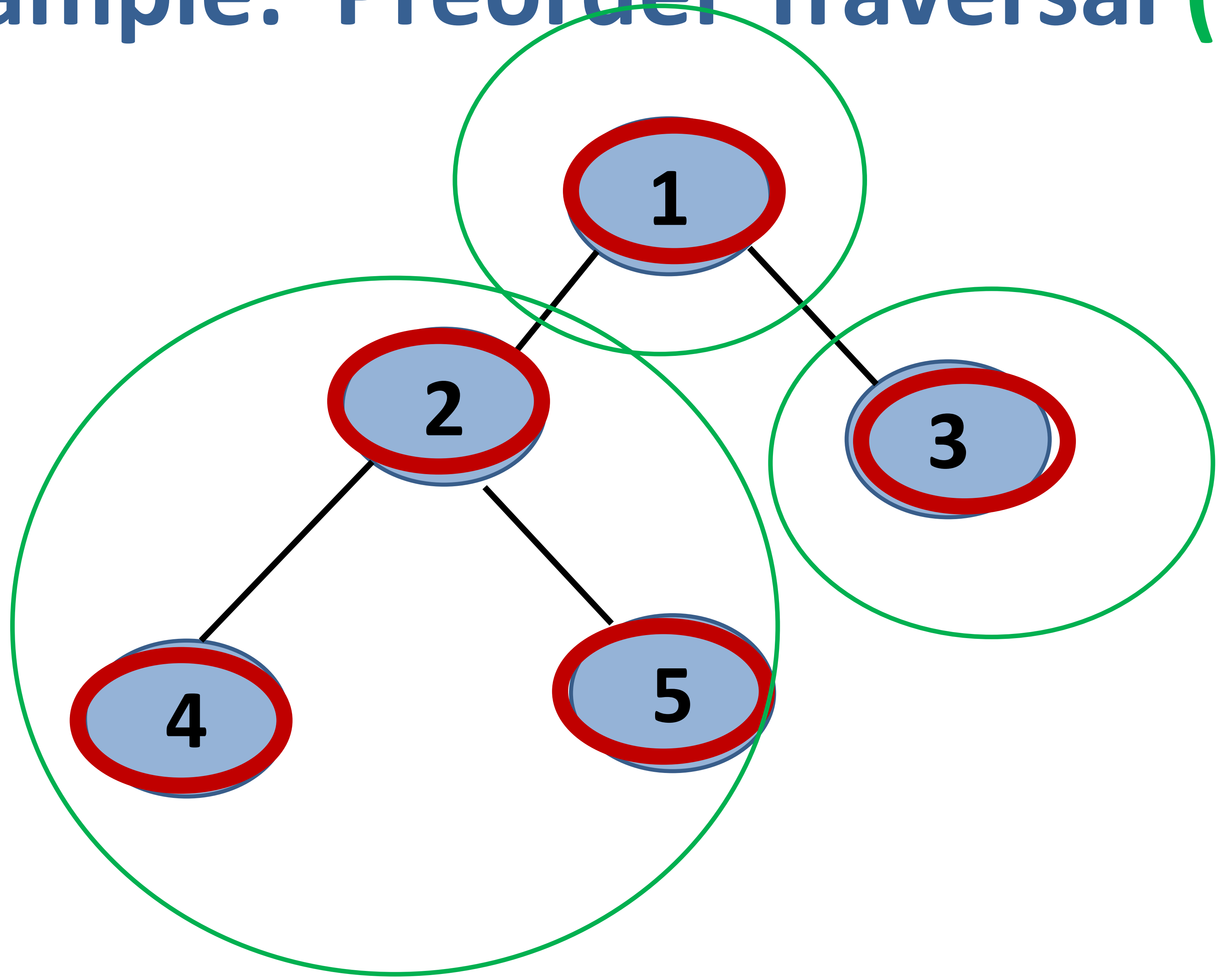
Preorder Traversal (Root, Left, Right)

Preorder Traversal (Root, Left, Right) Algorithm

Preorder traversal of a binary tree is defined as follow

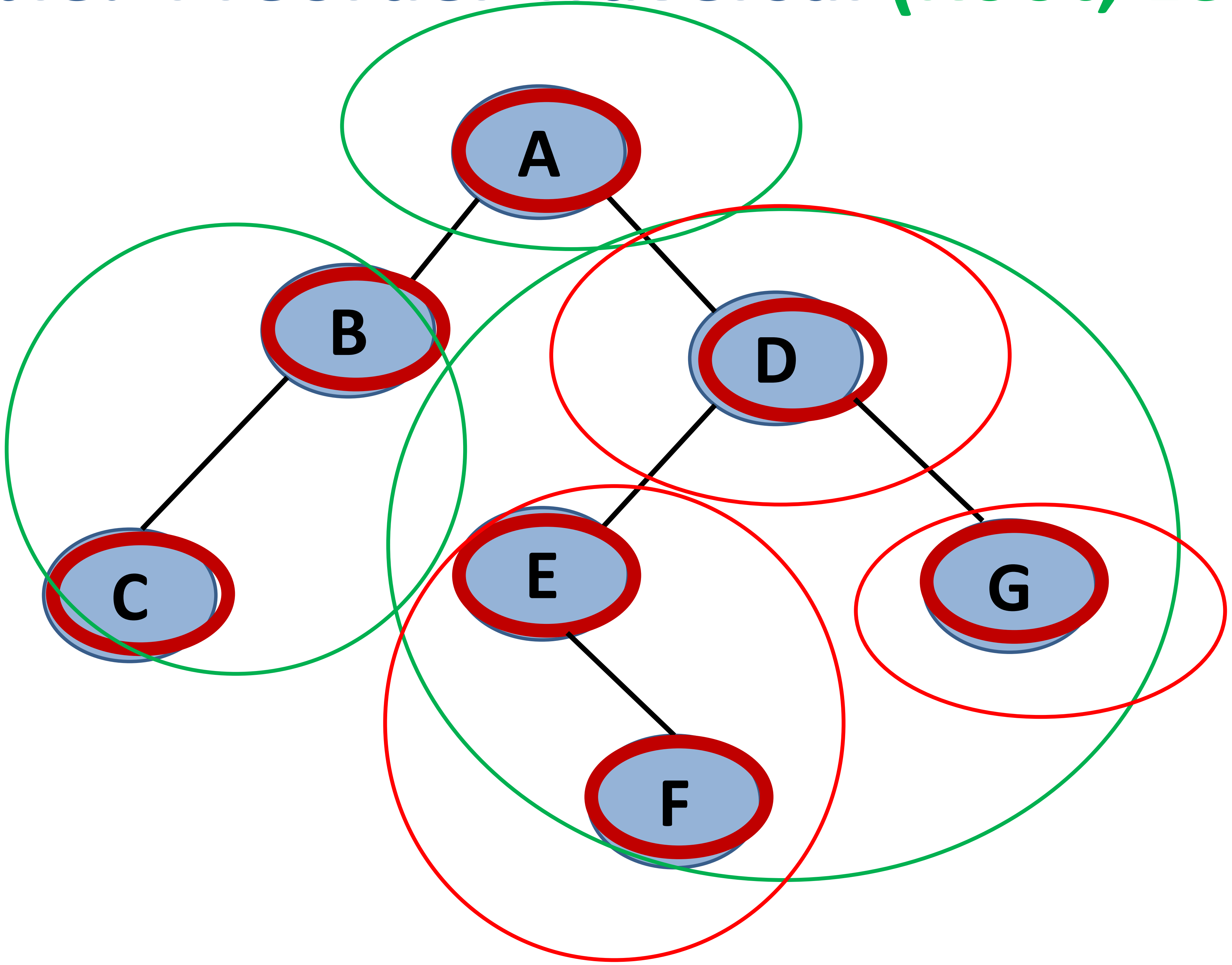
1. Visit the root
2. Traverse the left subtree, i.e., call Preorder (left-subtree)
3. Traverse the right subtree, i.e., call Preorder (right-subtree)

Example: Preorder Traversal (Root, Left, Right)



Preorder : 1 2 4 5 3

Example: Preorder Traversal (Root, Left, Right)



Preorder : A B C D E F G

Psedocode for Preorder Traversal

```
preOrder (treePointer ptr)
	{
				if (ptr != NULL)
				{
			visit(t);
				preOrder(ptr->leftChild);
				preOrder(ptr->rightChild);
				}
		}
```

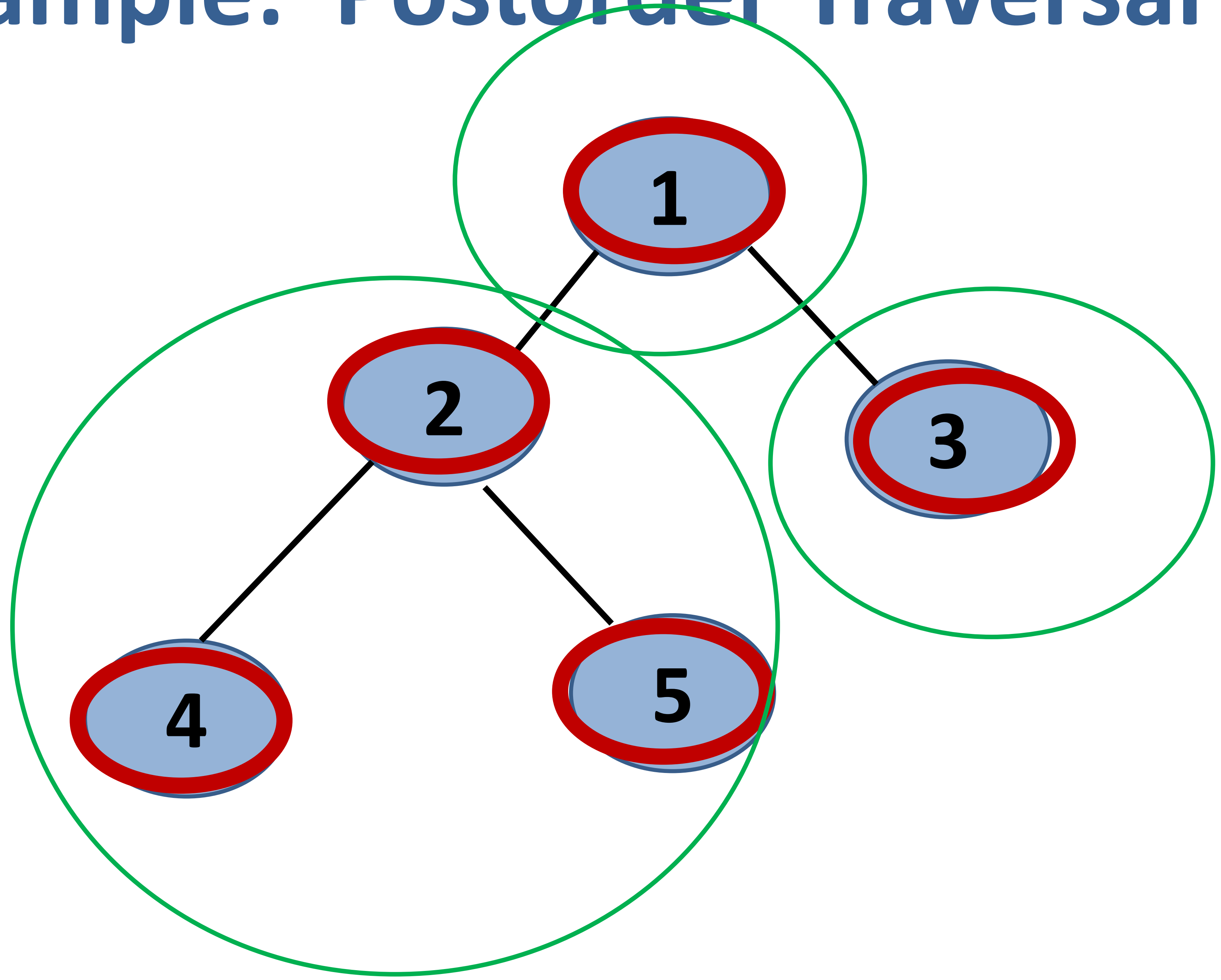
Postorder (Left, Right, Root)

Postorder Traversal (Left, Right, Root) Algorithm

Postorder traversal of a binary tree is defined as follow

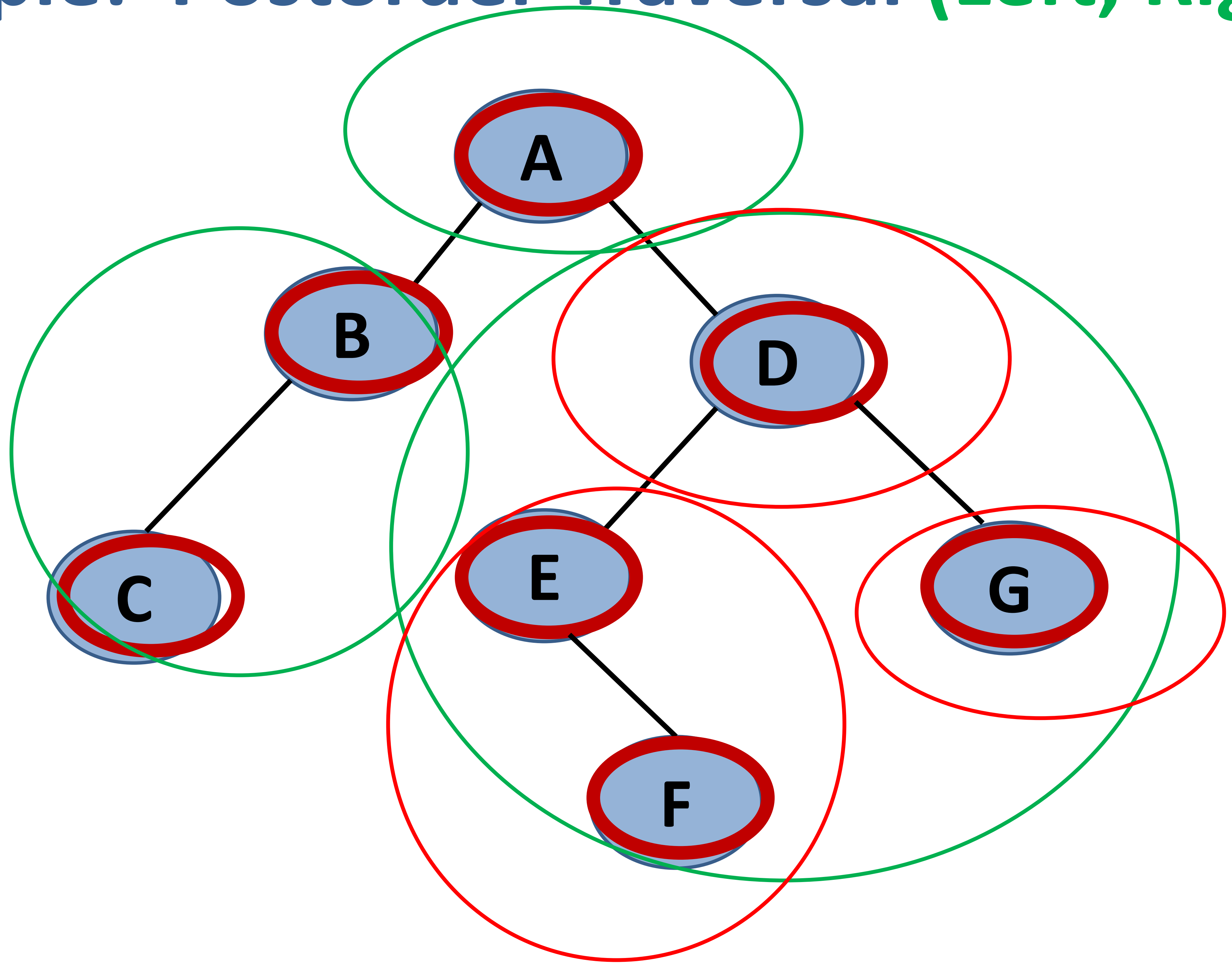
1. Traverse the left subtree, i.e., call Postorder (left-subtree)
2. Traverse the right subtree, i.e., call Postorder (right subtree)
3. Visit the root

Example: Postorder Traversal (Left, Right,Root)



Postorder : 4 5 2 3 1

Example: Postorder Traversal (Left, Right,Root)



Postorder : C B F E G D A

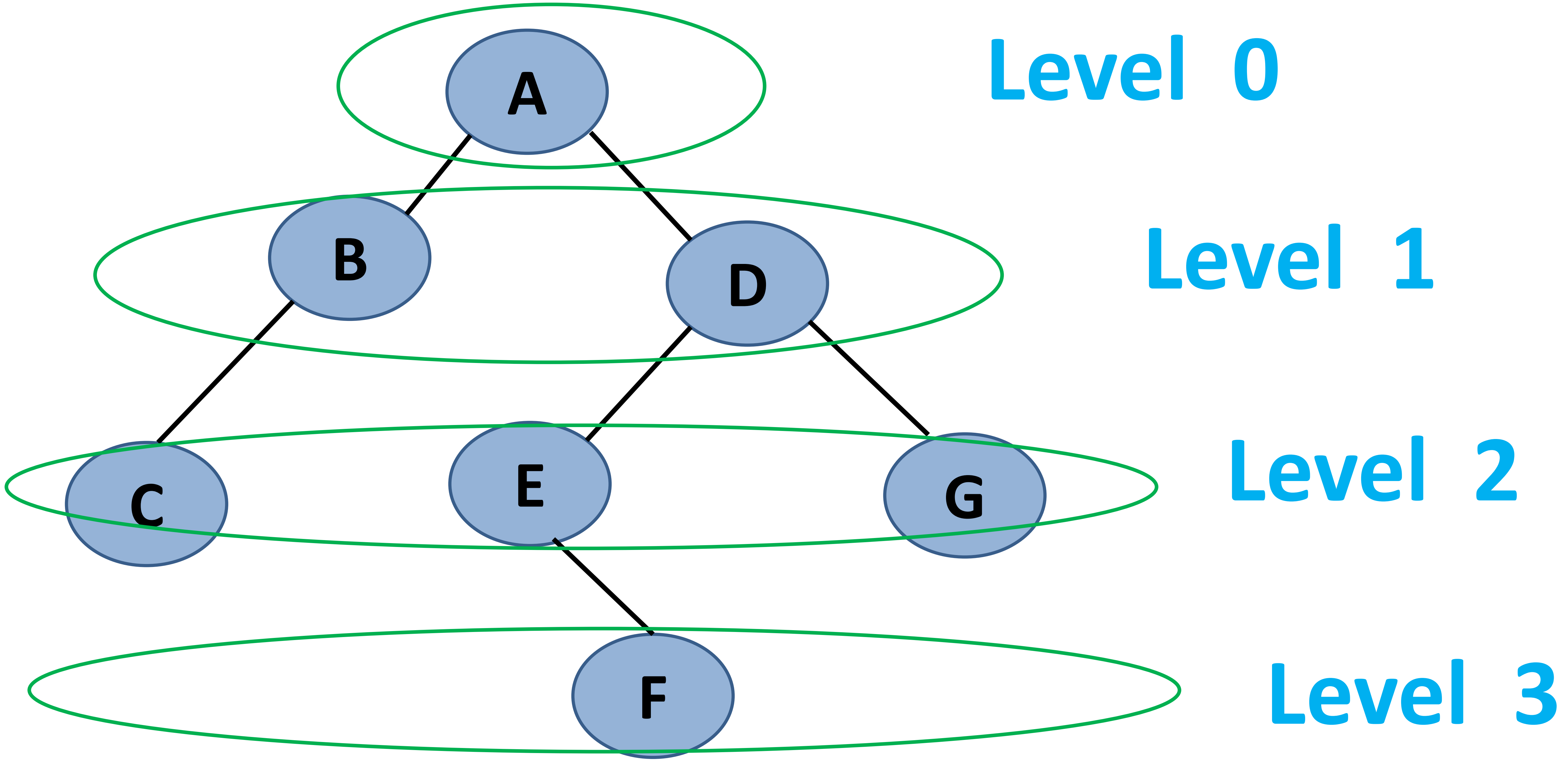
Psedocode for Postorder Traversal

```
postOrder(treePointer ptr)
    {
        if (ptr != NULL)
        {
        postOrder(ptr->leftChild);
        postOrder(ptr->rightChild);
        visit(t);
        }
    }
```

Level Order

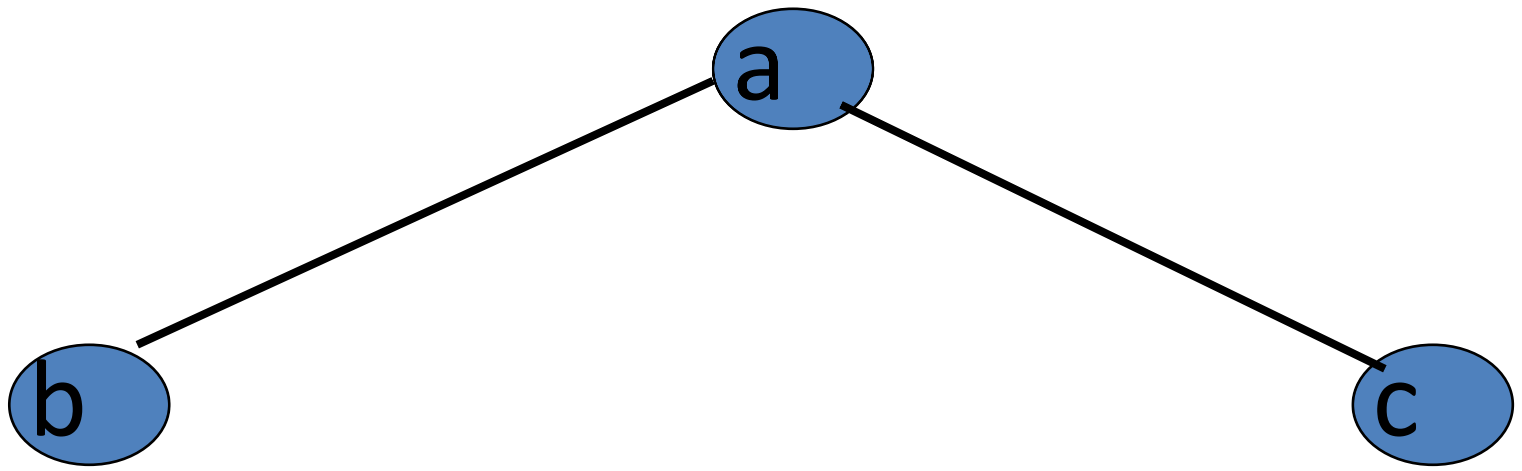
Let ptr be a pointer to the tree root.
while (ptr != NULL)
{
visit node pointed at by ptr and put its children on a FIFO queue;
if FIFO queue is empty, set ptr = NULL;
otherwise, delete a node from the FIFO queue and call it ptr;
}

Example Level Order Traversal



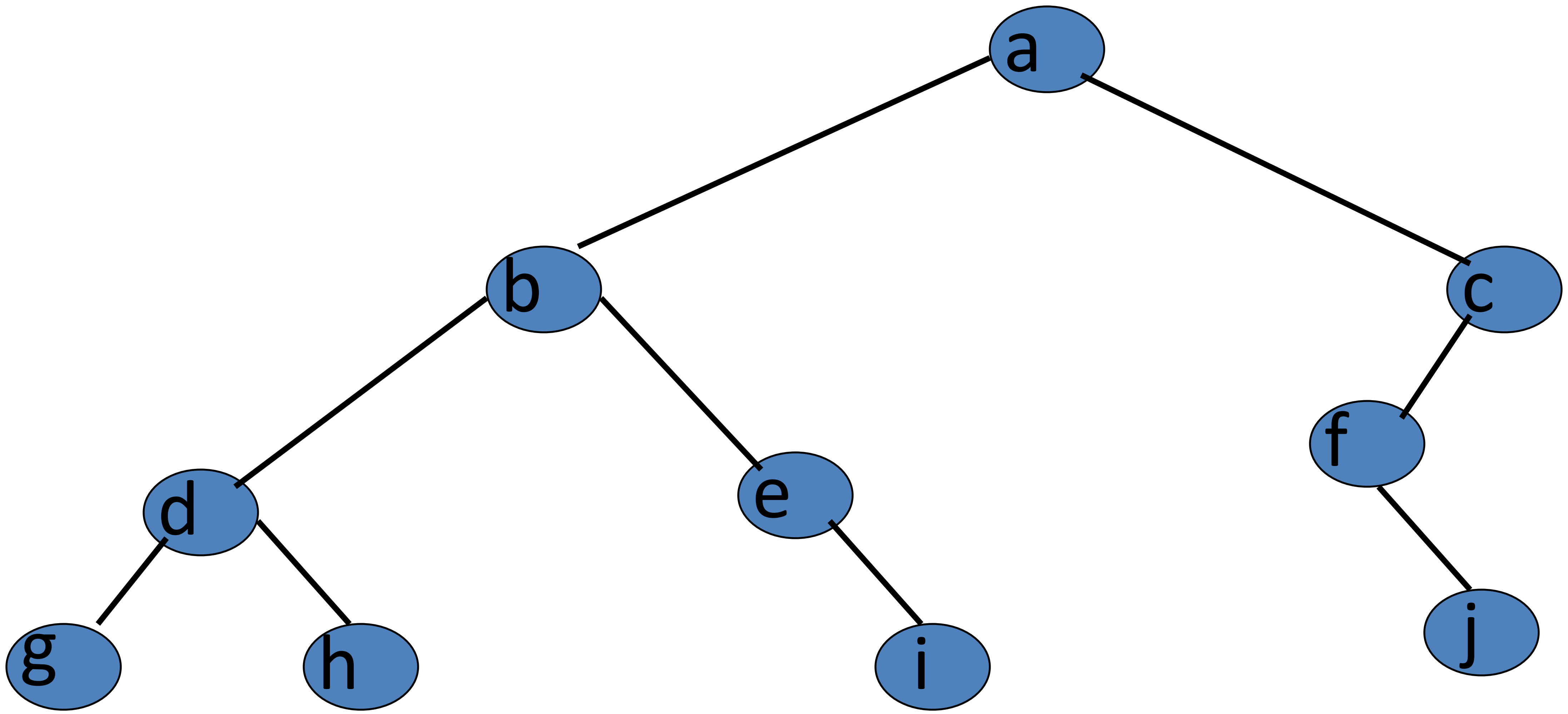
Level order : A B D C E G F

Preorder Example



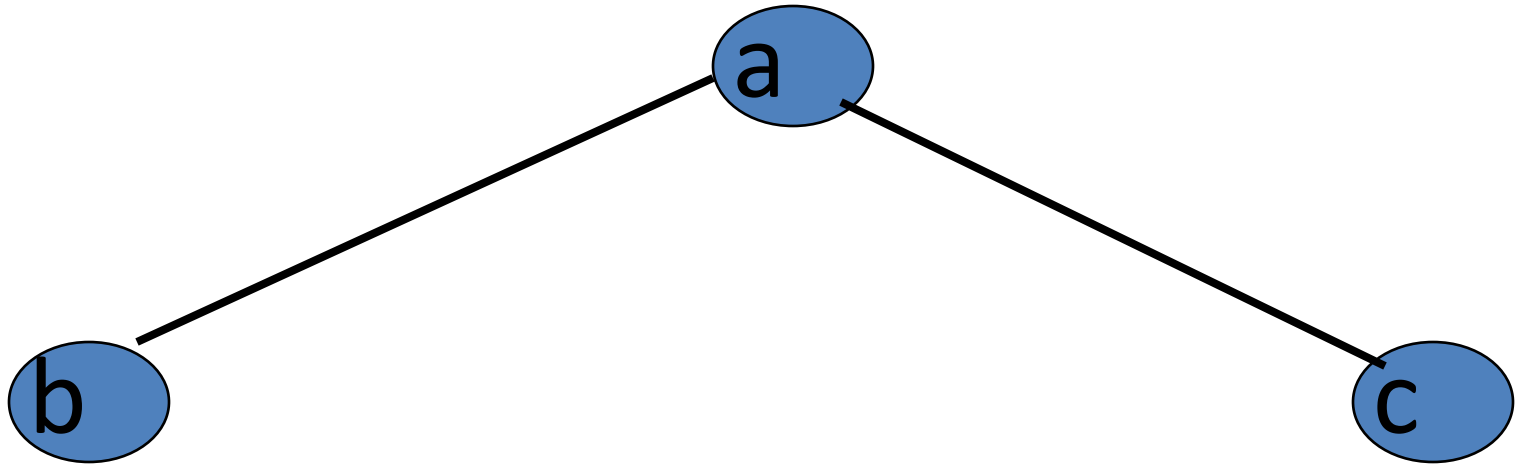
a b c

Preorder Example



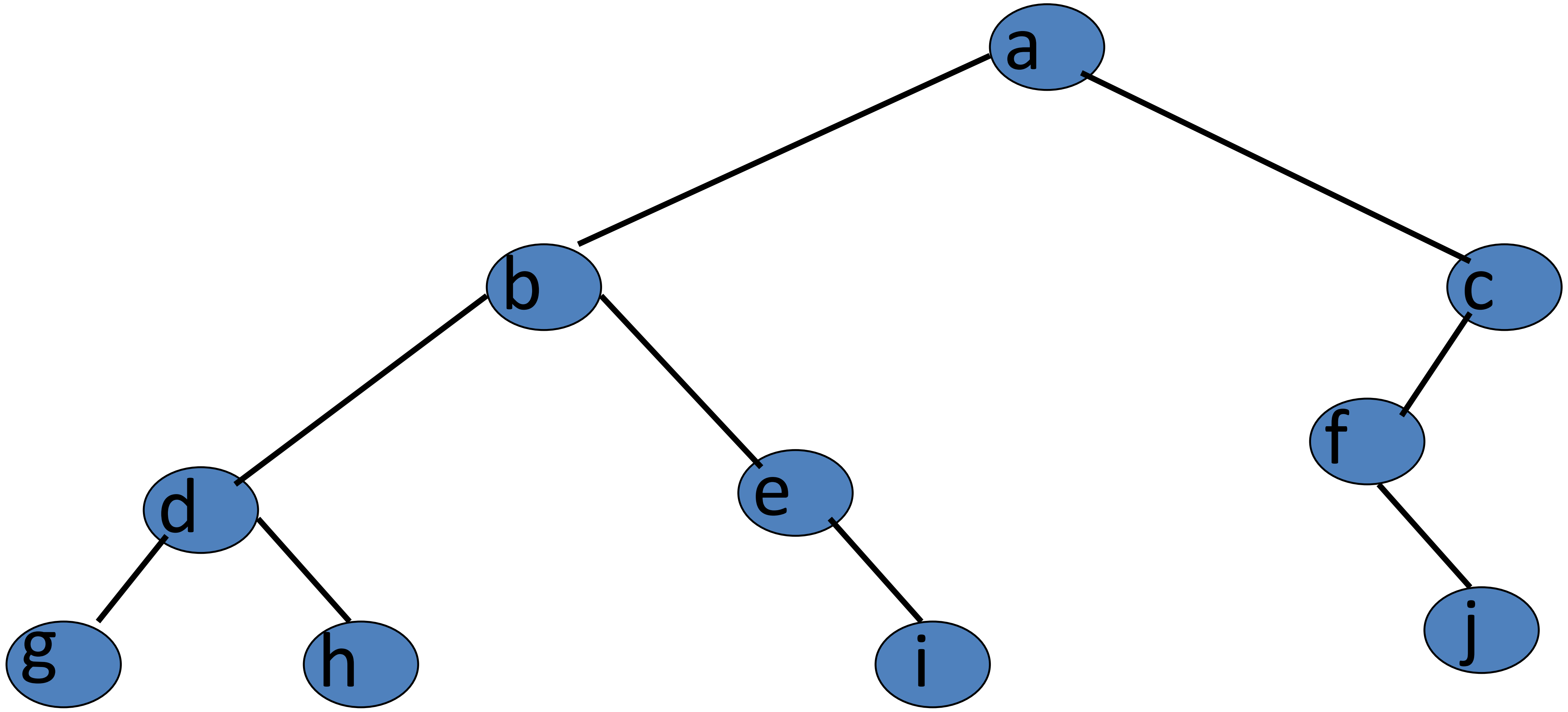
a b d g h e i c f j

Inorder Example



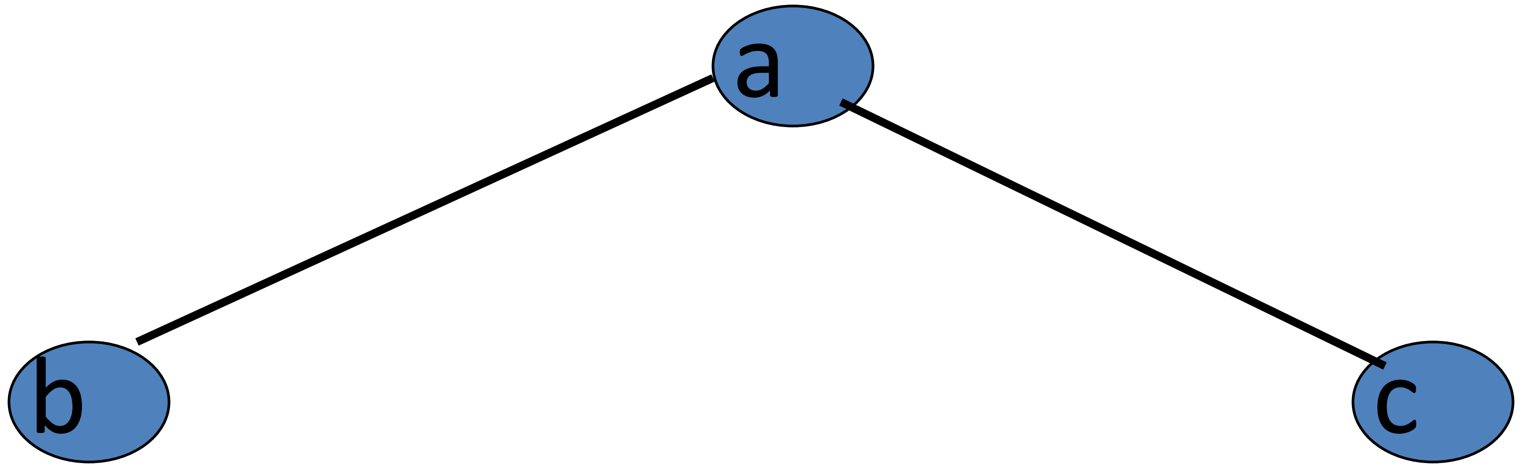
b a c

Inorder Example



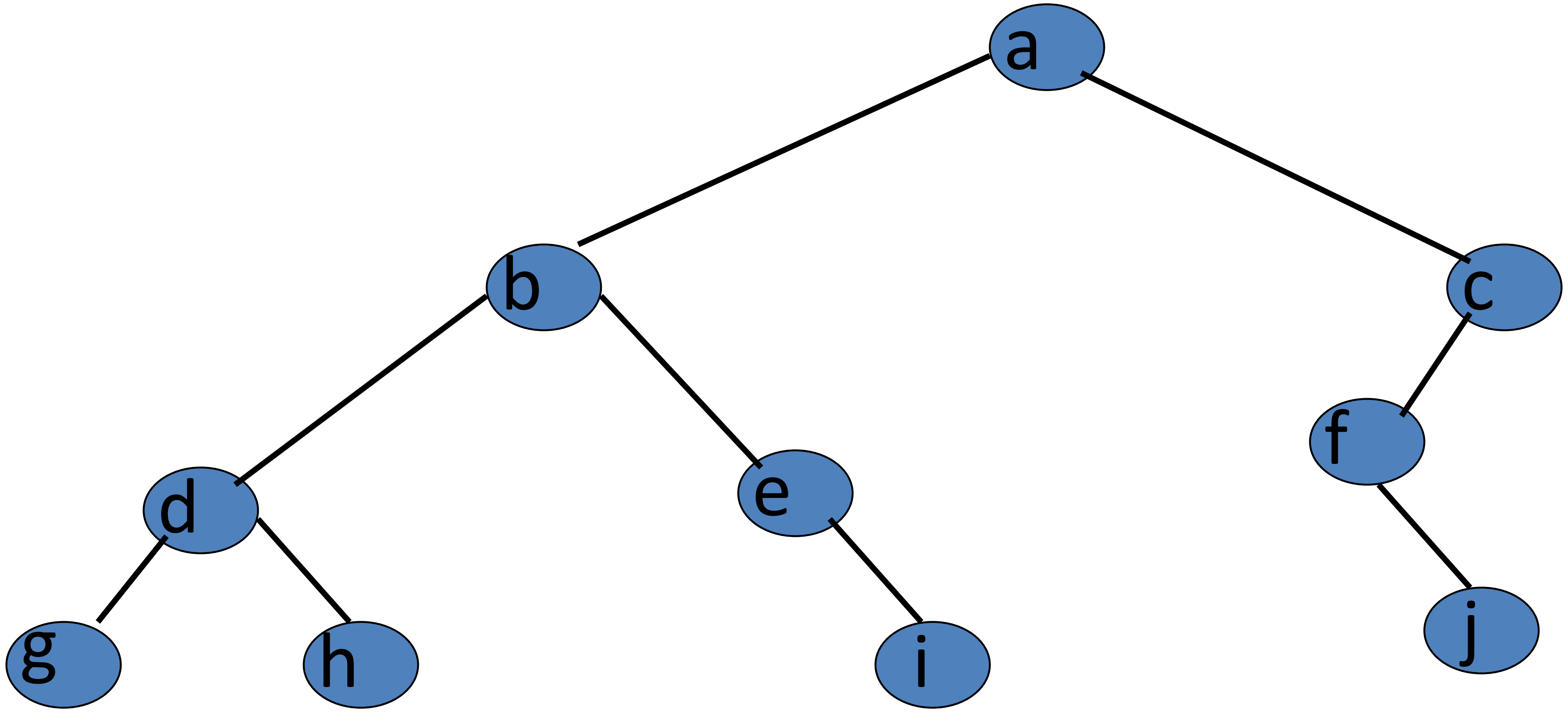
g d h b e i a f j c

Postorder Example



b c a

Postorder Example



g h d i e b j f c a

Another Examples and definition for Tree Traversal

Tree Traversal

- Displaying (or) visiting order of nodes in a binary tree is called as Binary Tree Traversal.
- There are three types of binary tree traversals.

1. In - Order Traversal
2. Pre - Order Traversal
3. Post - Order Traversal

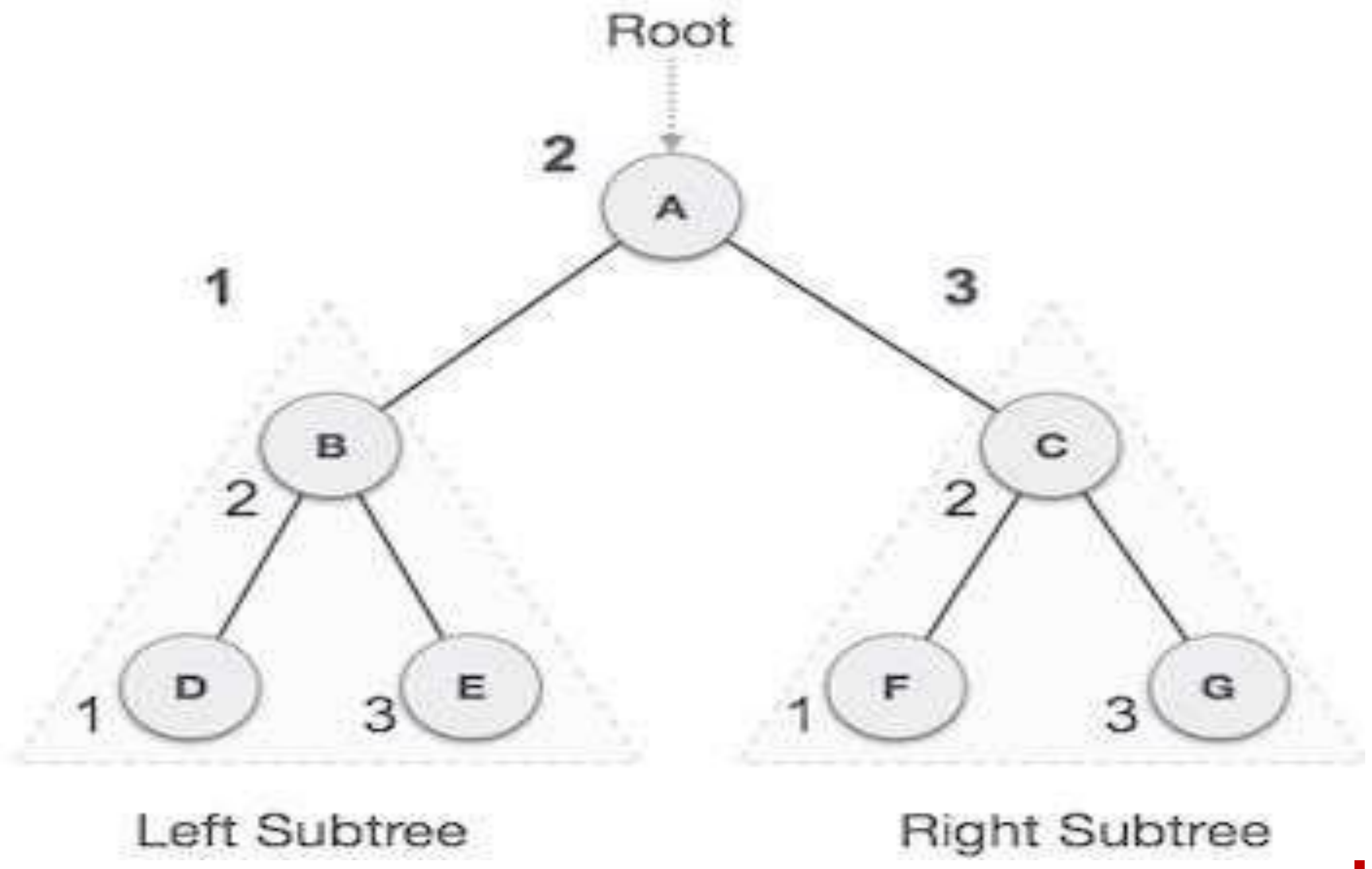
In-order Traversal

- In this traversal method, the left subtree is visited first, then the root and later the right sub-tree.

Example:

- We start from **A**, and following in-order traversal, we move to its left subtree **B**.
- **B** is also traversed in-order.
- The process goes on until all the nodes are visited.
- The output of inorder traversal of this tree will be

D → B → E → A → F → C → G



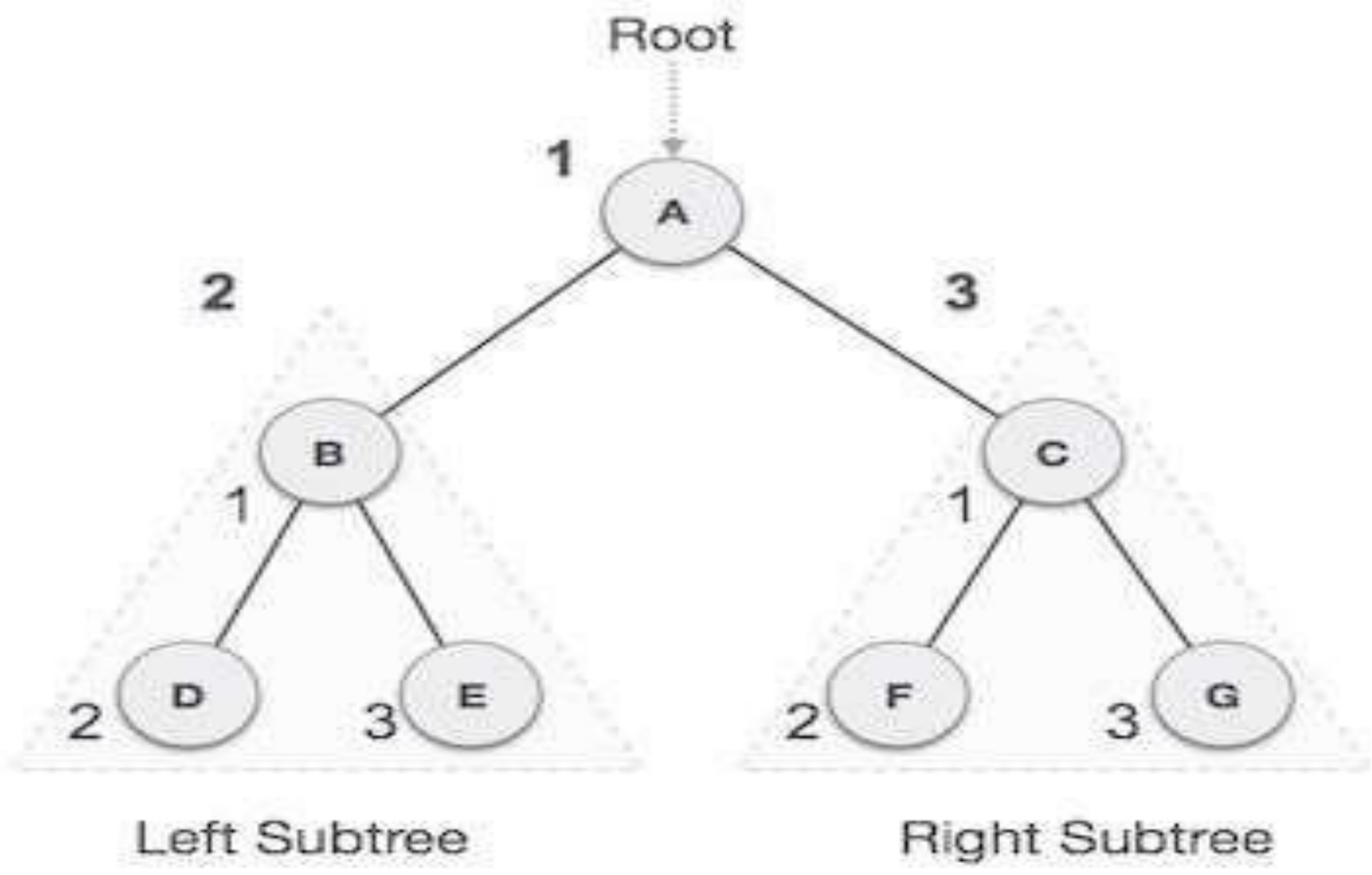
Inorder traversal

Algorithm

- Until all nodes are traversed –
- **Step 1** – Recursively traverse left subtree.
- **Step 2** – Visit root node.
- **Step 3** – Recursively traverse right subtree.

Pre-order Traversal

- In this traversal method, the root node is visited first, then the left subtree and finally the right subtree.



Algorithm:

- Until all nodes are traversed −
- **Step 1** − Visit root node.
- **Step 2** − Recursively traverse left subtree.
- **Step 3** − Recursively traverse right subtree.

We start from **A**, and following pre-order traversal, we first visit **A** itself and then move to its left subtree **B**.

OUTPUT : ***A → B → D → E → C → F → G***

Post-order Traversal

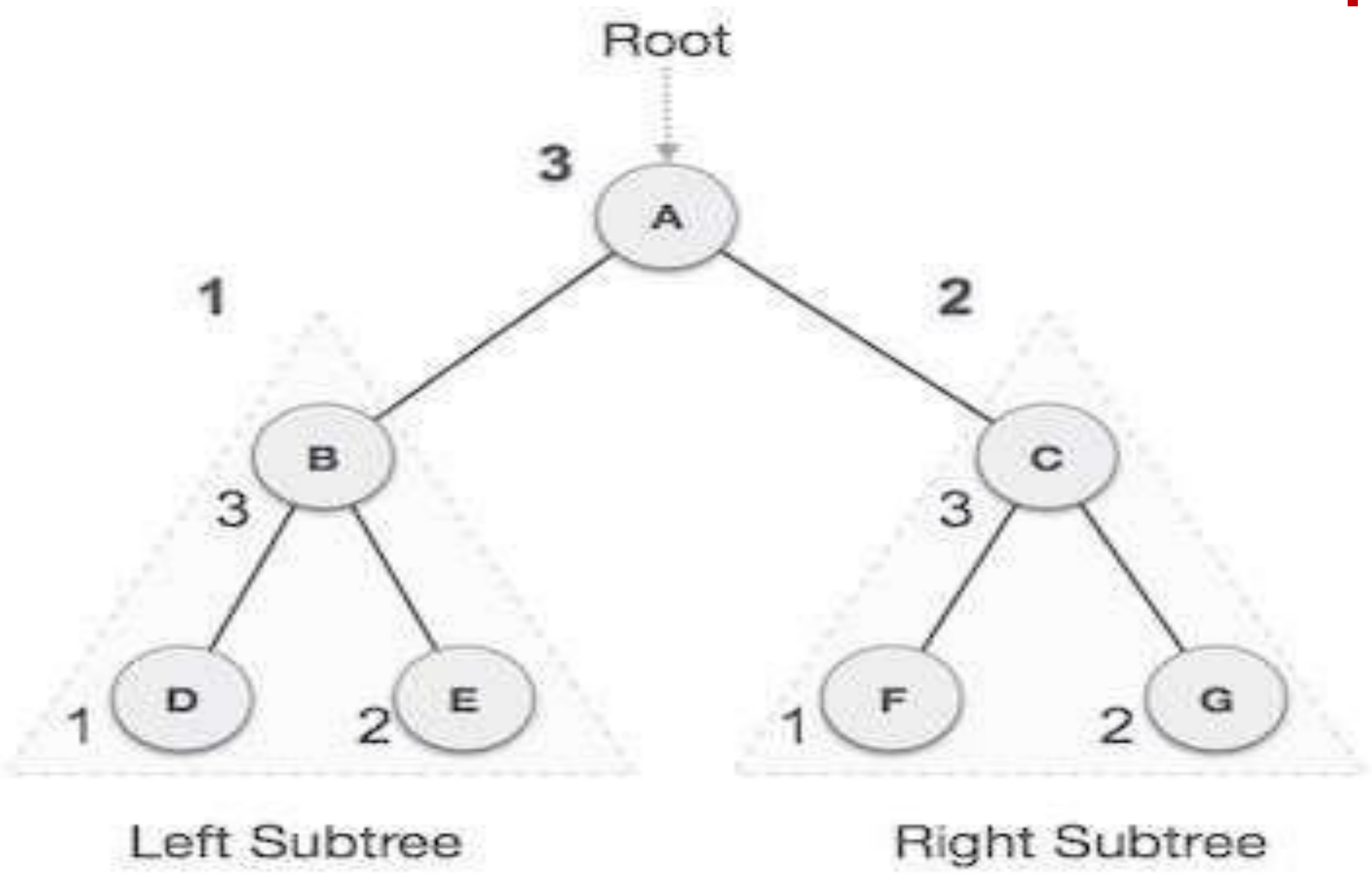
- In this traversal method, the root node is visited last, hence the name. First we traverse the left subtree, then the right subtree and finally the root node.

Algorithm

Until all nodes are traversed −

- **Step 1** − Recursively traverse left subtree.
- **Step 2** − Recursively traverse right subtree.
- **Step 3** − Visit root node.

We start from **A**, and following Post-order traversal, we



OUTPUT:

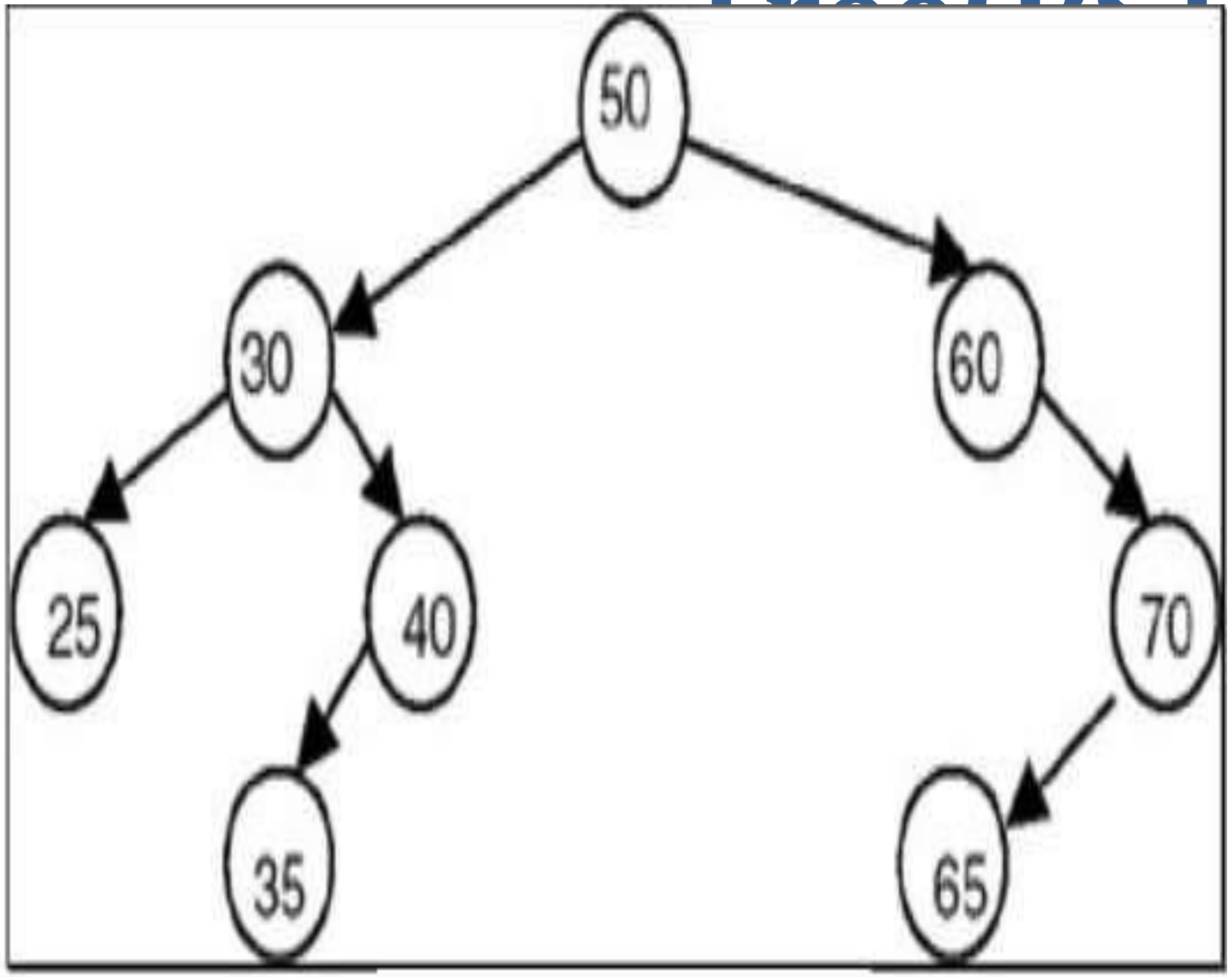
D → E → B → F → G → C → A

Binary Search Tree

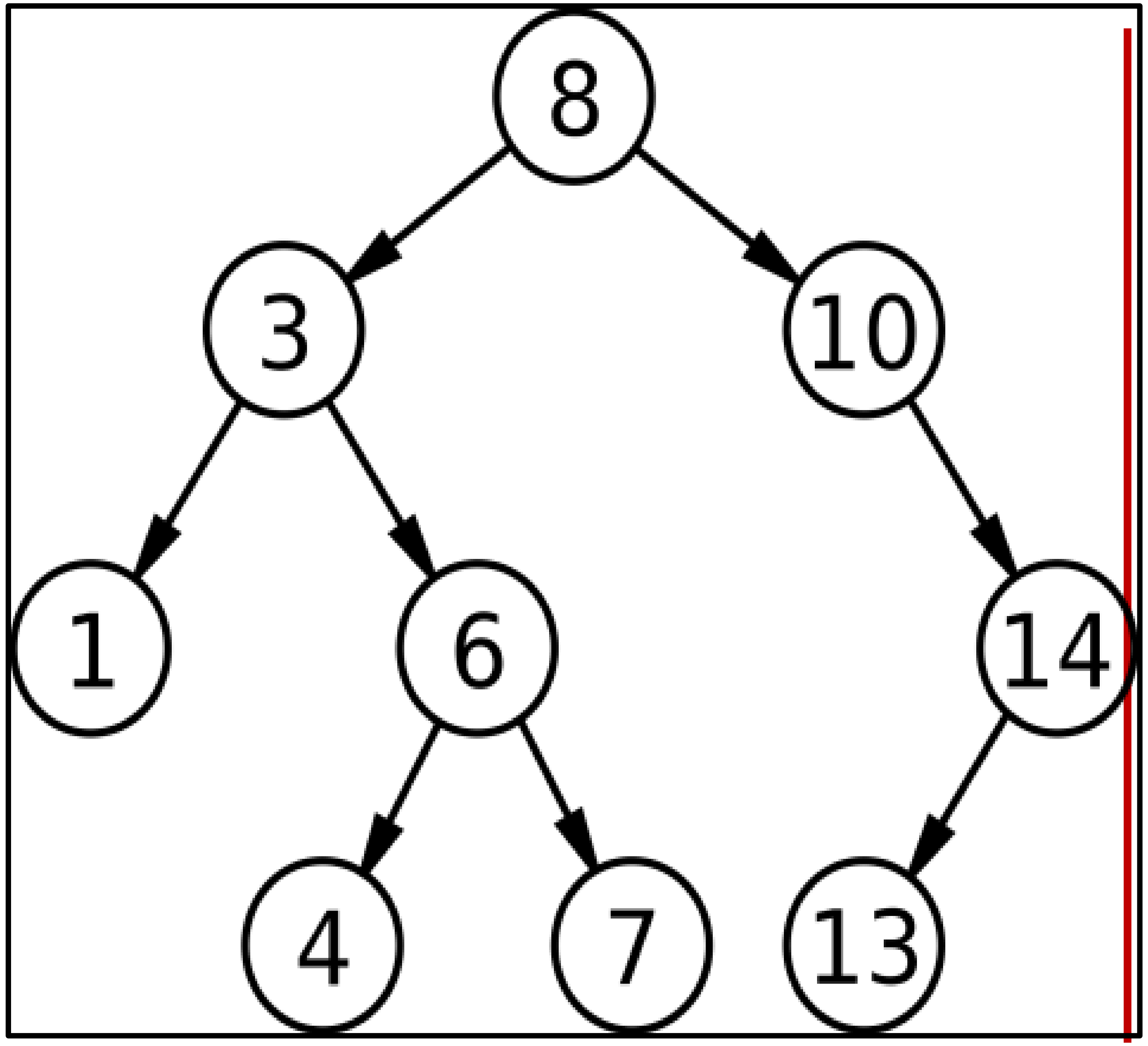
Binary Search Tree(BST)

- A binary search tree (BST) is a **binary tree** because each tree node has a maximum of two children
- The properties of binary search tree are:
  - All nodes of **left sub-tree are less** than the root node
  - All nodes of **right sub-tree are greater** than the root node
  - Both sub-trees of each node are also BSTs i.e. they have the above two properties

Binary Search Tree(BST)



The binary search tree.



Thank you