NON-LINEAR DATA STRUCTURE

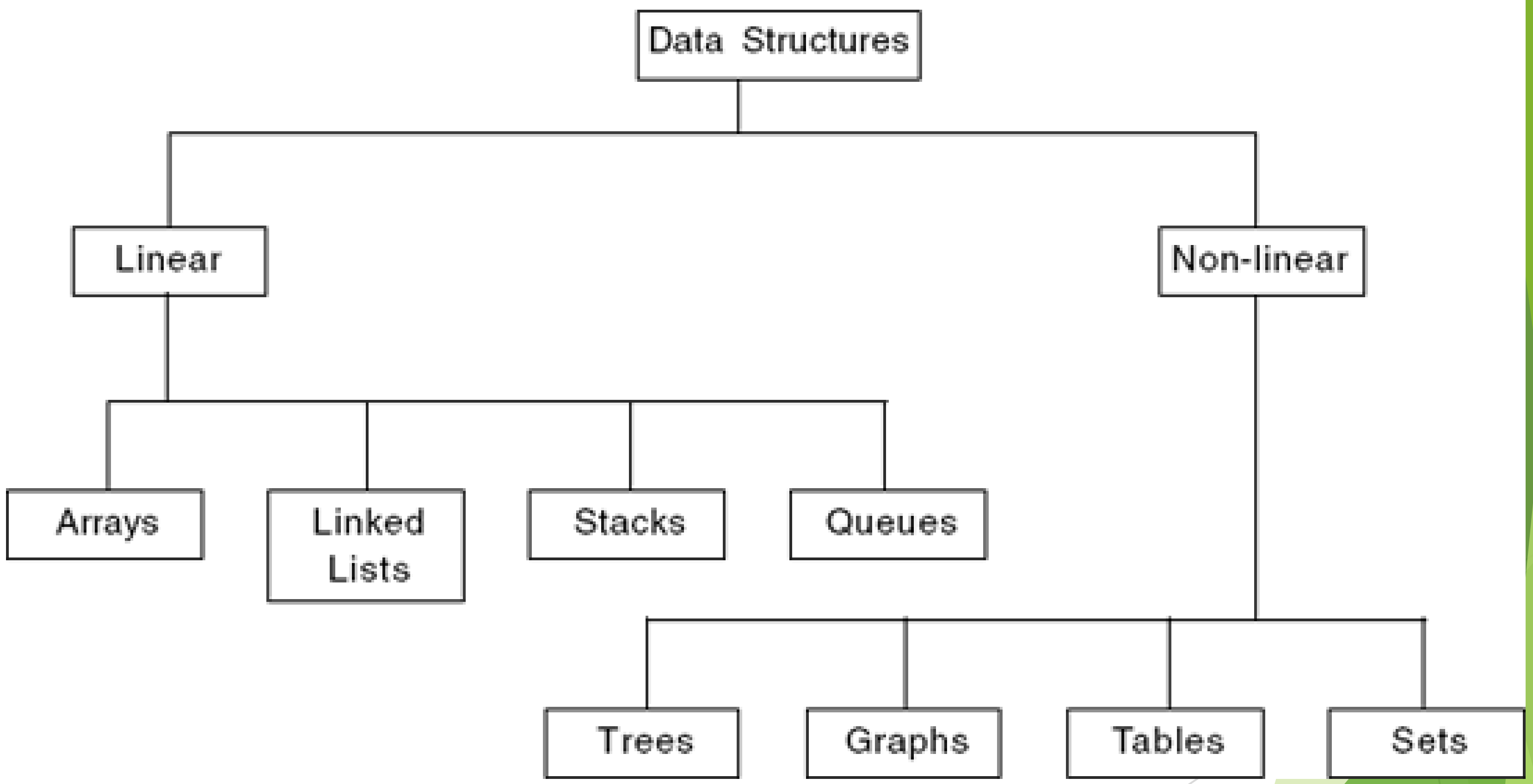
(Prof.) Dr. Aparna Kumari

CSE Dept.,

ICT-Ganpat University

abk03@ganpatuniversity.ac.in

Classification of Data Structure



Non-linear data structure

- Data elements are not arranged in a contiguous manner.
- Arrangement is non-sequential, so the data elements cannot be traversed or accessed in a single run.
- In the case of linear data structure, element is connected to two elements (previous and the next element), whereas, in the non-linear data structure, an element can be connected to more than two elements.
- **Trees** and **Graphs** are the types of non-linear data structure.

Trees

- A tree can be defined as finite set of data items (nodes).
- Tree is non-linear type of data structure in which data items are arranged or stored in a sorted sequence.
- Tree represent the hierarchical relationship between various elements.

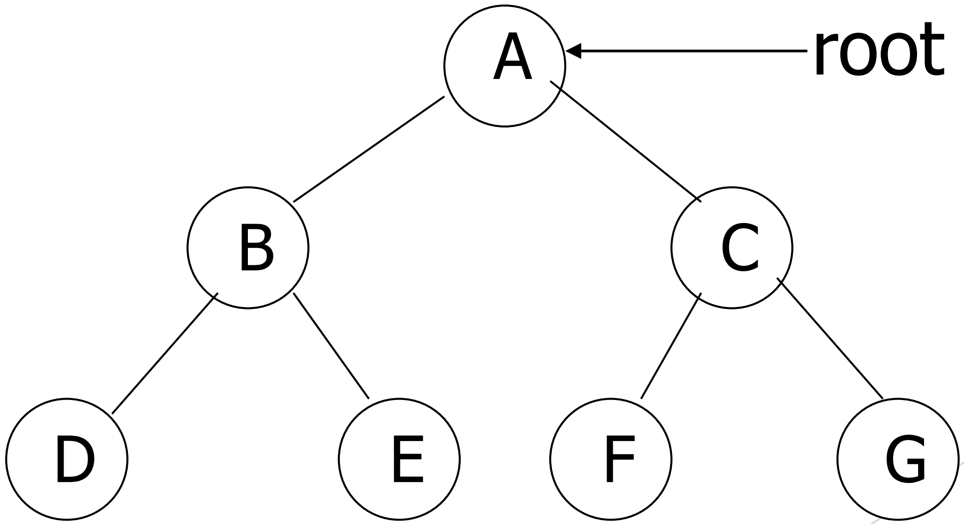
Trees

In trees:

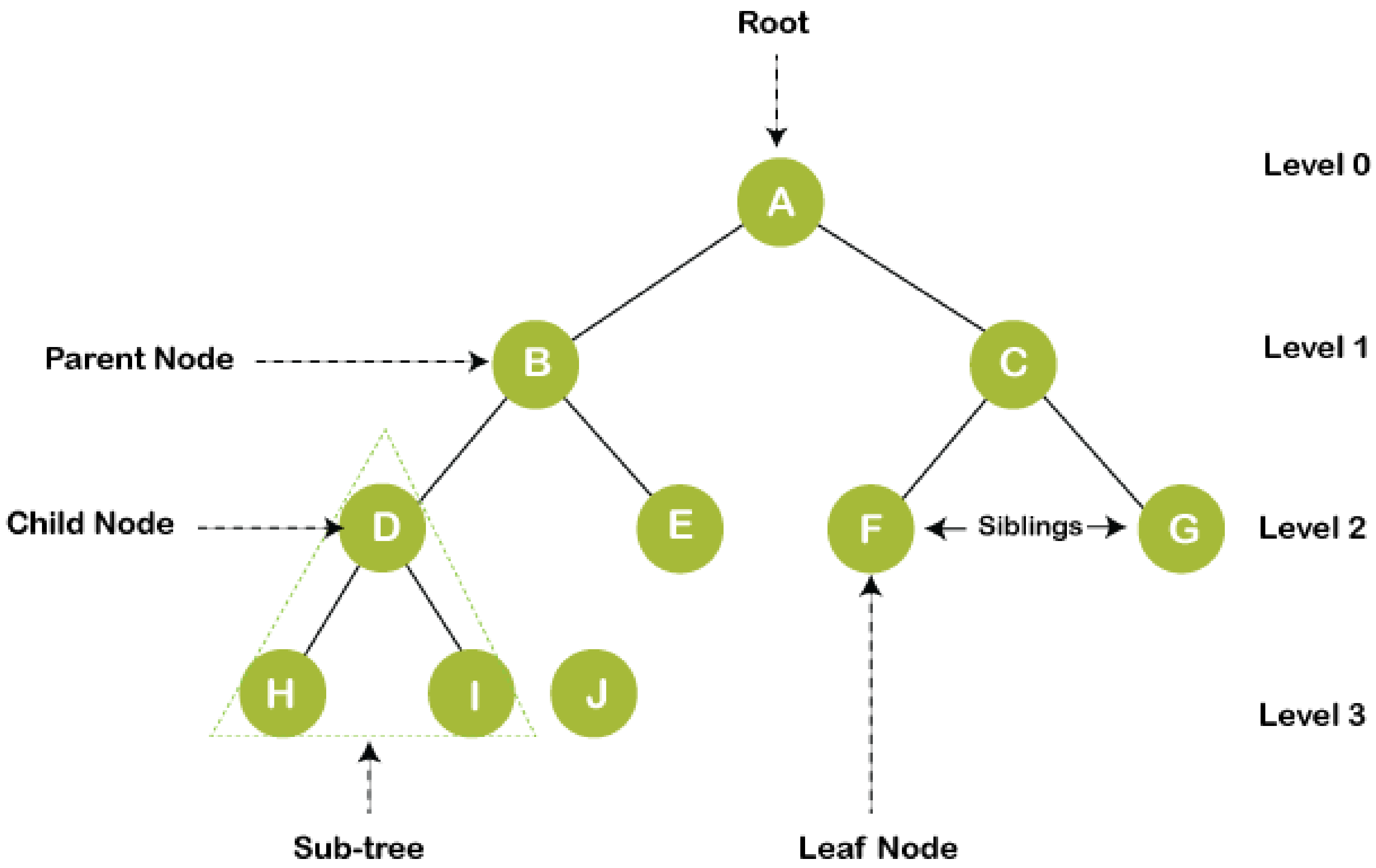
- There is a special data item at the top of hierarchy called the Root of the tree.
- The remaining data items are partitioned into number of mutually exclusive subset, each of which is itself, a tree which is called the sub tree.
- The tree always grows in length towards bottom in data structures, unlike natural trees which grows upwards.

Trees

- The tree structure organizes the data into branches, which related the information.
- It has a hierarchical tree structure that forms a parent-child relationship.



Trees



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

Trees

For example: The posts of employees are arranged in a tree data structure like managers, officers, clerk. In the above figure, A represents a manager, B and C represent the officers, and other nodes represent the clerks.

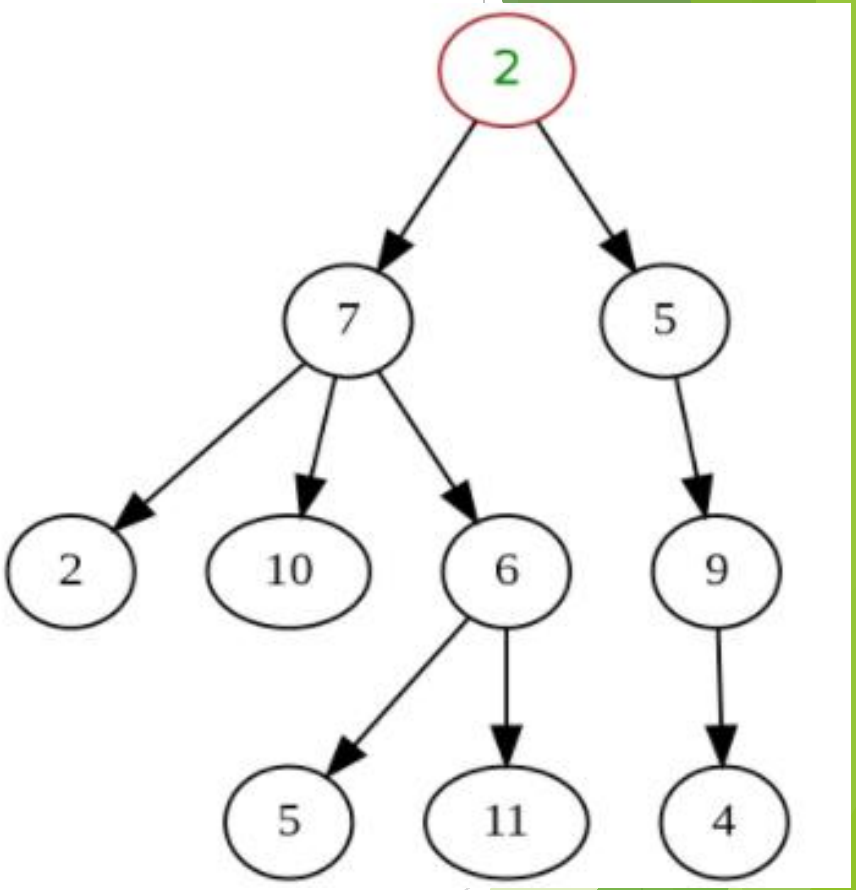
Trees

Types of Trees in Data Structure

1. General Tree
2. Binary Tree
3. Binary Search Tree
4. AVL Tree
5. Red Black Tree
6. Splay Tree
7. Treap
8. B-Tree

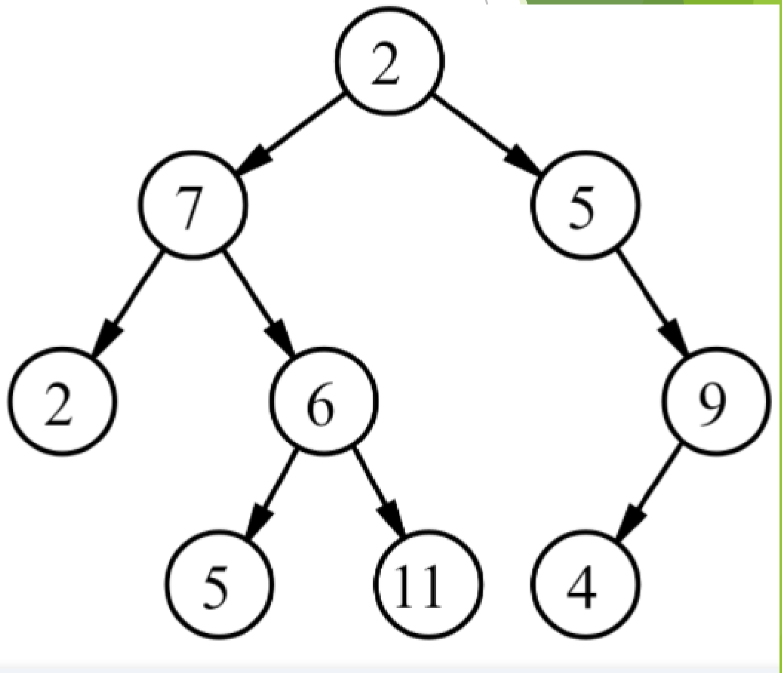
General Tree

- A general tree is characterized by the lack of any configuration or limitations on the number of children a node can have.
- Any tree with a hierarchical structure can be described as a general tree.
- A node can have any number of children.



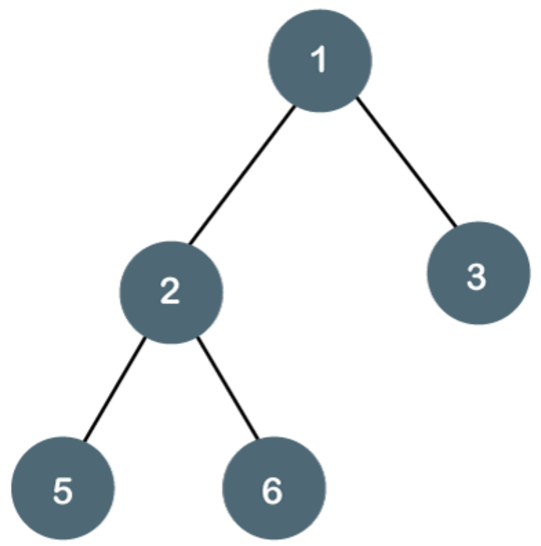
Binary Tree

- A binary tree is made up of nodes that can have two children, as described by the word “binary,” which means “two numbers.”
- In a binary tree, any node can have a maximum of 0, 1, or 2 nodes.
- Data structures’ binary trees are highly functional that can be further subdivided into a variety of types.



Binary Tree

- Each node has at most two children, which are referred to as the left child and the right child.
- If the tree is empty, then the value of root is NULL. A Binary Tree node contains the following parts:
  - Data
  - Pointer to left child
  - Pointer to the right child



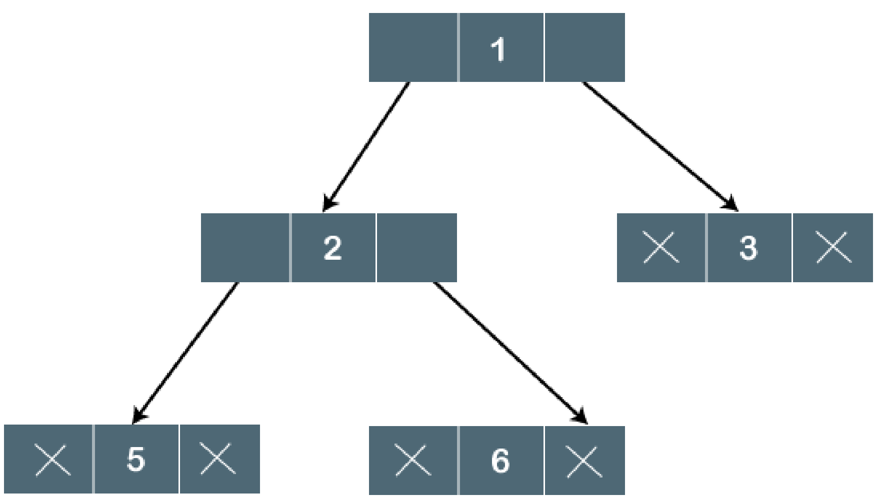
  - **struct node {**

int data;

struct node *leftChild;

struct node *rightChild;

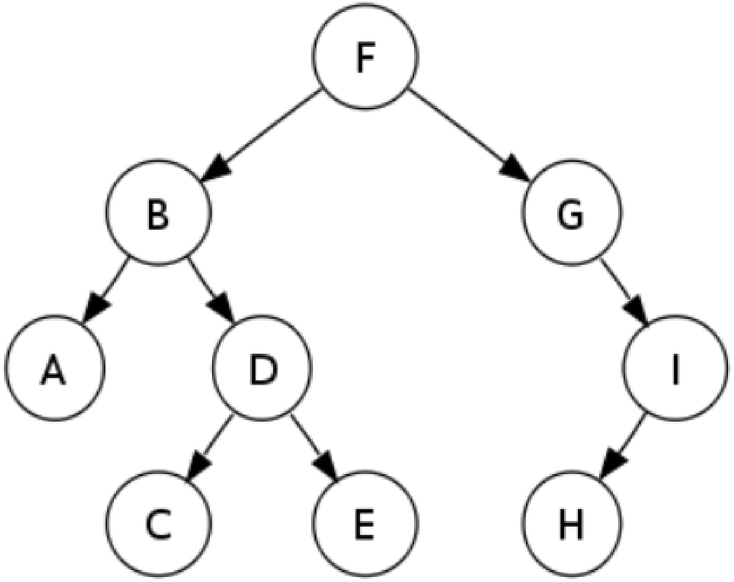
};



Binary Search Tree

- A Binary Search Tree (BST) is a subtype of binary tree that is organised in such a way that it allows for faster searching, lookup, and addition/removal of data.

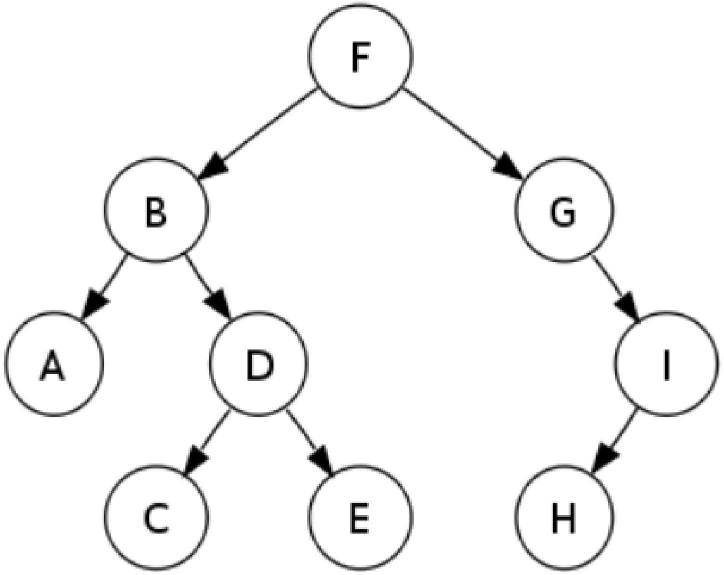
- The representation of nodes in a BST is defined by three fields: **the data, its left child, and its right child.**



Binary Search Tree

- Every node on the left side (left child) must have a value less than the value of its parent node.

- Every node on the right side (right child) must have a higher value than its parent node.



Graph

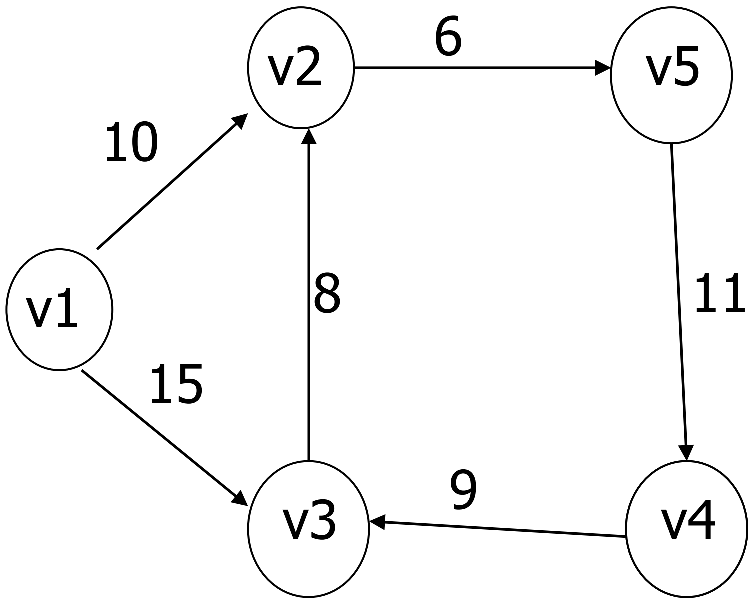
- Graph is a mathematical non-linear data structure capable of representing many kind of physical structures.
- It has found application in Geography, Chemistry and Engineering sciences.
- Definition: A graph G(V,E) is a set of vertices V and a set of edges E.

Graph

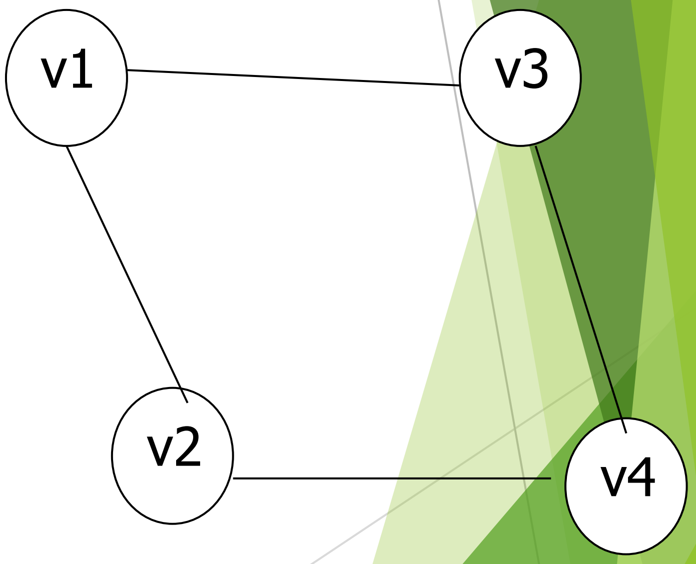
- An edge connects a pair of vertices and many have weight such as length, cost and another measuring instrument for according the graph.
- Vertices on the graph are shown as point or circles and edges are drawn as arcs or line segment.

Graph

- Example of graph:



[a] Directed & Weighted Graph



[b] Undirected Graph

Graph

- Types of Graphs:
  - Directed graph
  - Undirected graph
  - Weighted graph
  - Connected graph
  - Non-connected graph

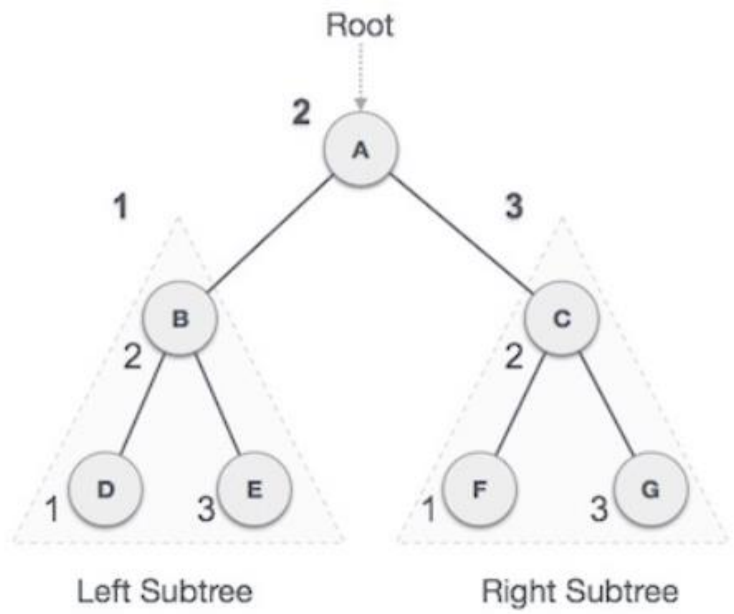
Binary Tree Traversal

- Traversal is a process to visit all the nodes of a tree and may print their values too.
- Cannot randomly access a node in a tree.
- There are three ways which we use to traverse a tree
    - In-order Traversal
    - Pre-order Traversal
    - 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**.
- Every node may represent a subtree itself.
- If a binary tree is traversed **in-order**, the output will produce sorted key values in an ascending order

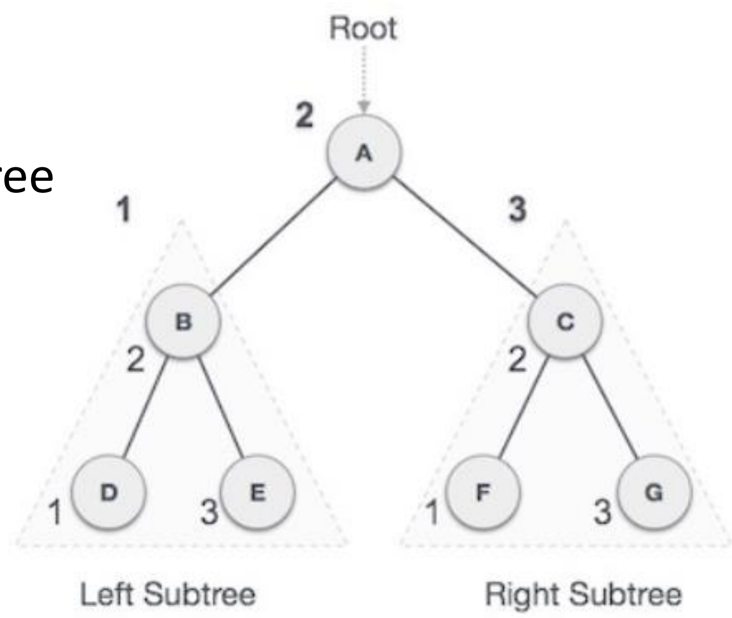
$D \rightarrow B \rightarrow E \rightarrow A \rightarrow F \rightarrow C \rightarrow G$



In-order Traversal

Algorithm:

1. First, visit all the nodes in the left subtree
2. Then the root node
3. Visit all the nodes in the right subtree

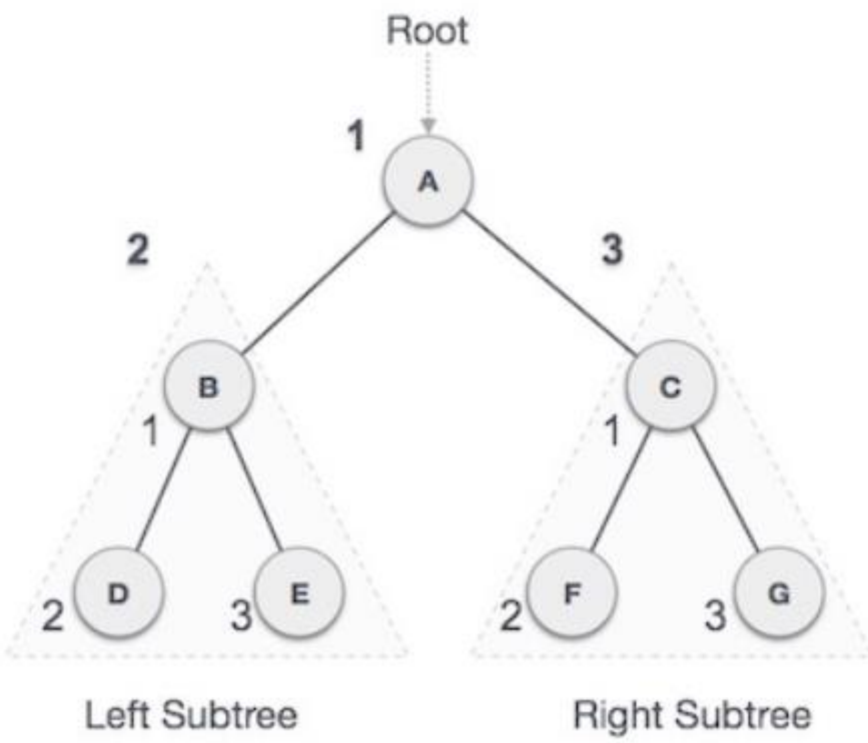


D → B → E → A → F → C → G

Pre-order Traversal

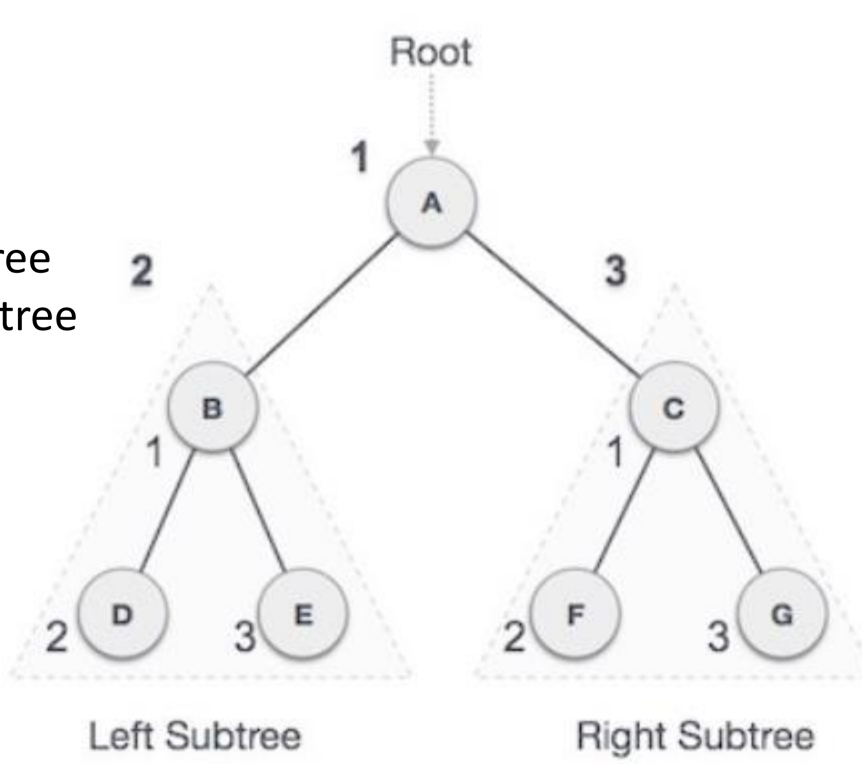
- In this traversal method, the **root node is visited first, then the left subtree and finally the right subtree**.
- Every node may represent a subtree itself.

A $\rightarrow$ ***B*** $\rightarrow$ ***D*** $\rightarrow$ ***E*** $\rightarrow$ ***C*** $\rightarrow$ ***F*** $\rightarrow$ ***G***



Pre-order Traversal

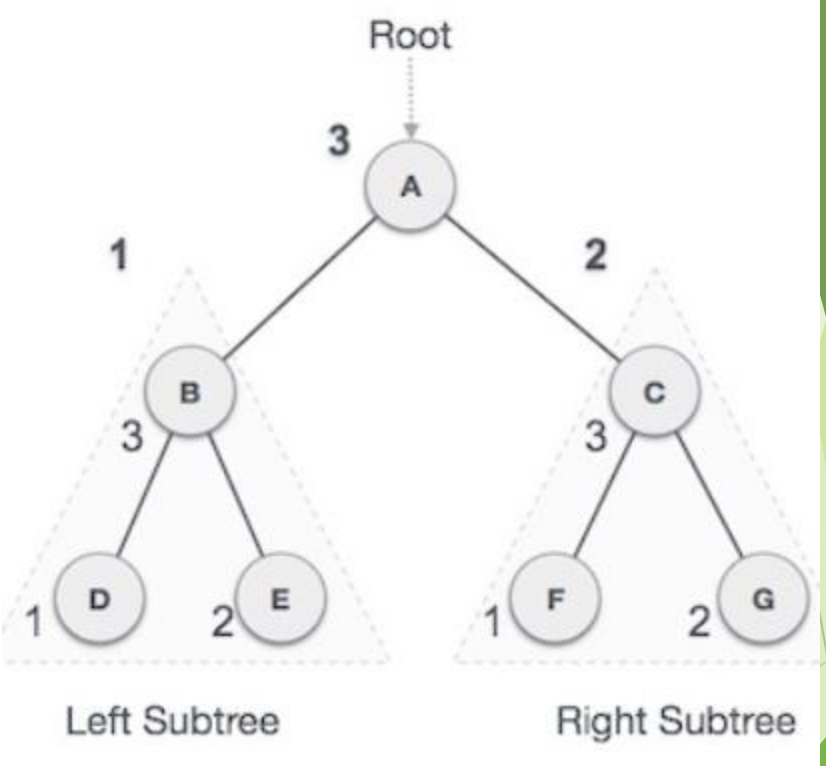
Algorithm:
1.Visit root node
2.Visit all the nodes in the left subtree
3.Visit all the nodes in the right subtree



$A \rightarrow B \rightarrow D \rightarrow E \rightarrow C \rightarrow F \rightarrow G$

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.**
- Every node may represent a subtree itself.

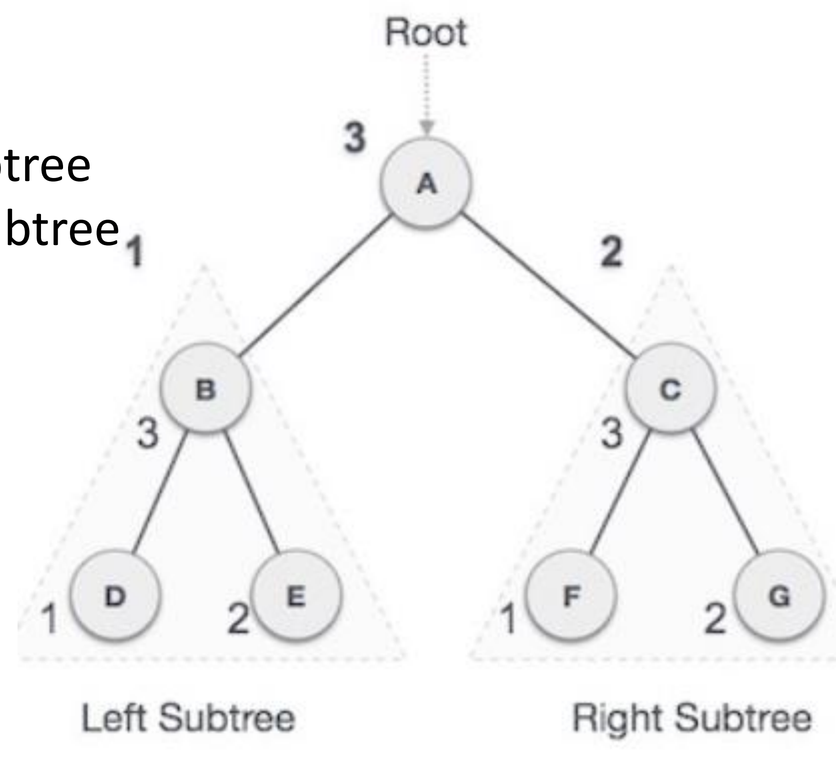


D → E → B → F → G → C → A

Post-order Traversal

Algorithm:

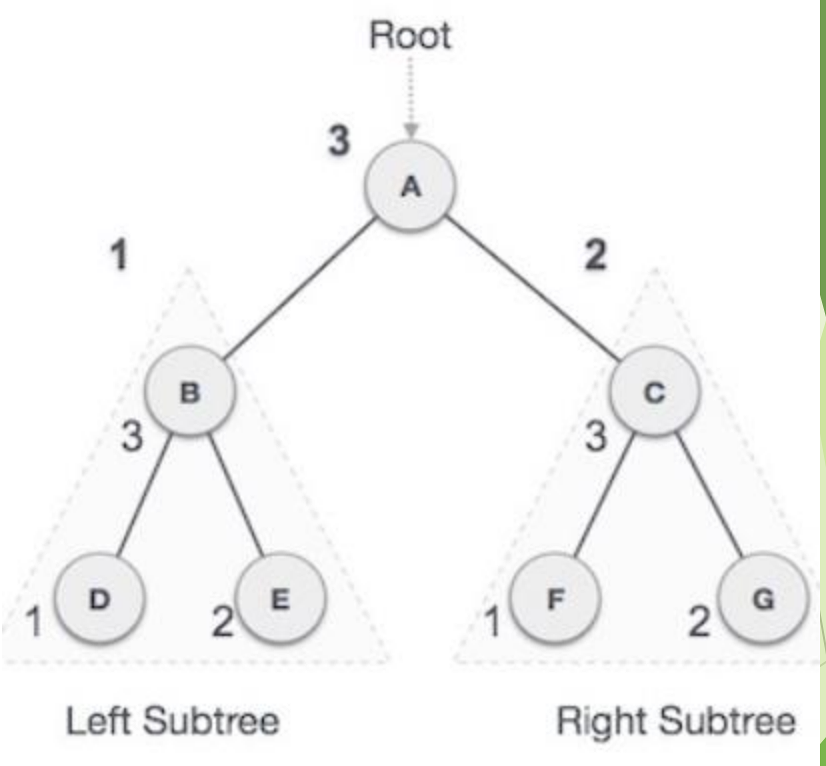
1.Visit all the nodes in the left subtree
2.Visit all the nodes in the right subtree
3.Visit the root node



$D \rightarrow E \rightarrow B \rightarrow F \rightarrow G \rightarrow C \rightarrow A$

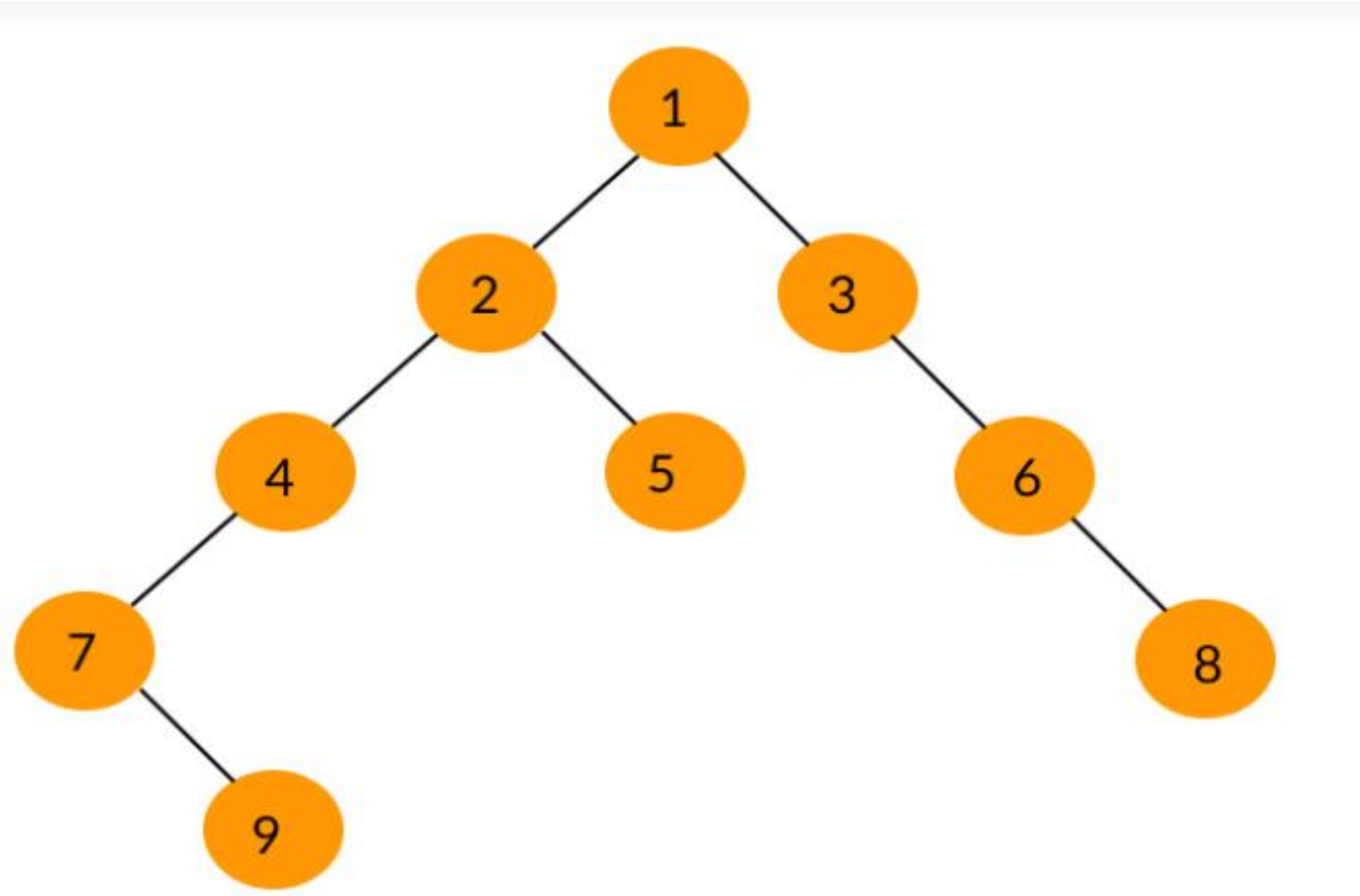
Level Order Binary Tree Traversal

- In this traversal method, we traverse tree level-wise in the increasing order of level, i.e., level-0, level-1, level-2, and so on.



A->B->C->D->E->F->G

Exercise for Binary Tree Traversal



Binary Tree-creation

```
// C program for binary tree creation
//Author: Dr. Aparna Kumari

/* A binary tree node has data, pointer to left child
   and a pointer to right child */
struct node {
    int data;
    struct node* left;
    struct node* right;
};

/* Helper function that allocates a new node */
struct node* newNode(int data)
{
    struct node* node
        = (struct node*)malloc(sizeof(struct node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return (node);}
```

Binary Tree-creation

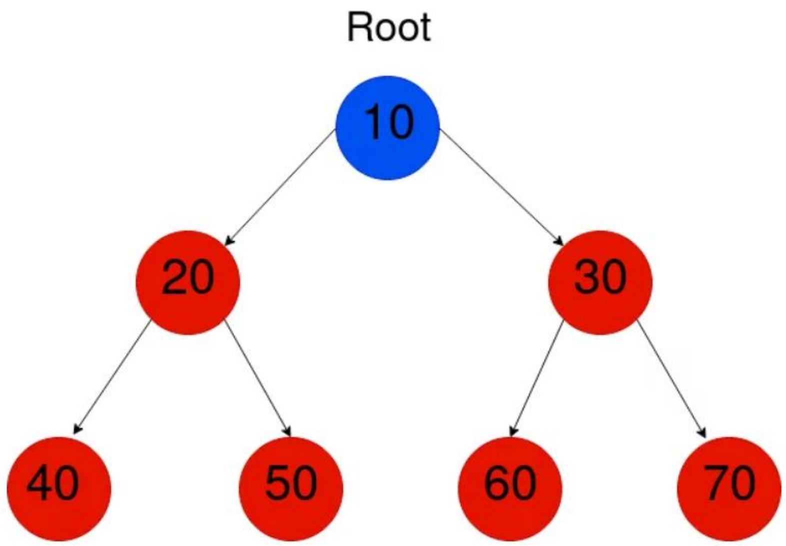
```
int main()
{
    struct node* root = newNode(20);
    root->left = newNode(15);
    root->right = newNode(30);
    root->left->left = newNode(25);
    root->left->right = newNode(19);
    root->right->left = newNode(31);
    root->right->right = newNode(45);

    printf("Elements of binary tree are: ");
    Display(root,2);
    printf("\n");
```

Binary Tree-creation

```
/* Print all elements */
void Display(struct node* root, int level)
{
    if (root == NULL)
        return;
    if (level == 0)
        printf("%d ", root->data);
    else if (level > 0) {
            Display(root->left, level - 1);
            Display(root->right, level - 1);
    }
}
```

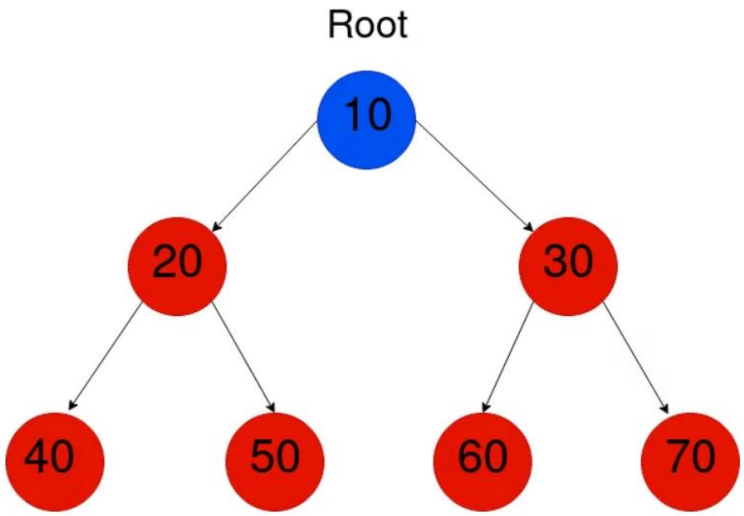
Binary Tree Traversal: In-order

```
void inorder_traversal(struct node* root) {
   if(root != NULL) {
      inorder_traversal(root->leftChild);
      printf("%d ",root->data);
      inorder_traversal(root->rightChild);
   }
}
```



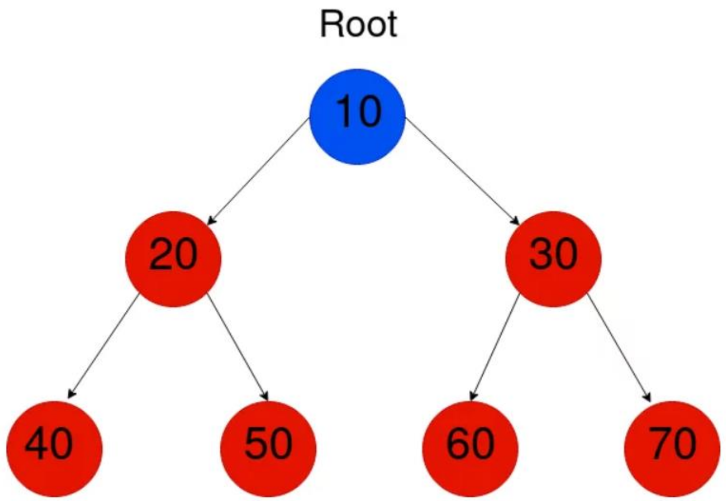
Binary Tree Traversal: Pre-order

```
void pre_order_traversal(struct node* root) {
   if(root != NULL) {
      printf("%d ",root->data);
      pre_order_traversal(root->leftChild);
      pre_order_traversal(root->rightChild);
   }
}
```

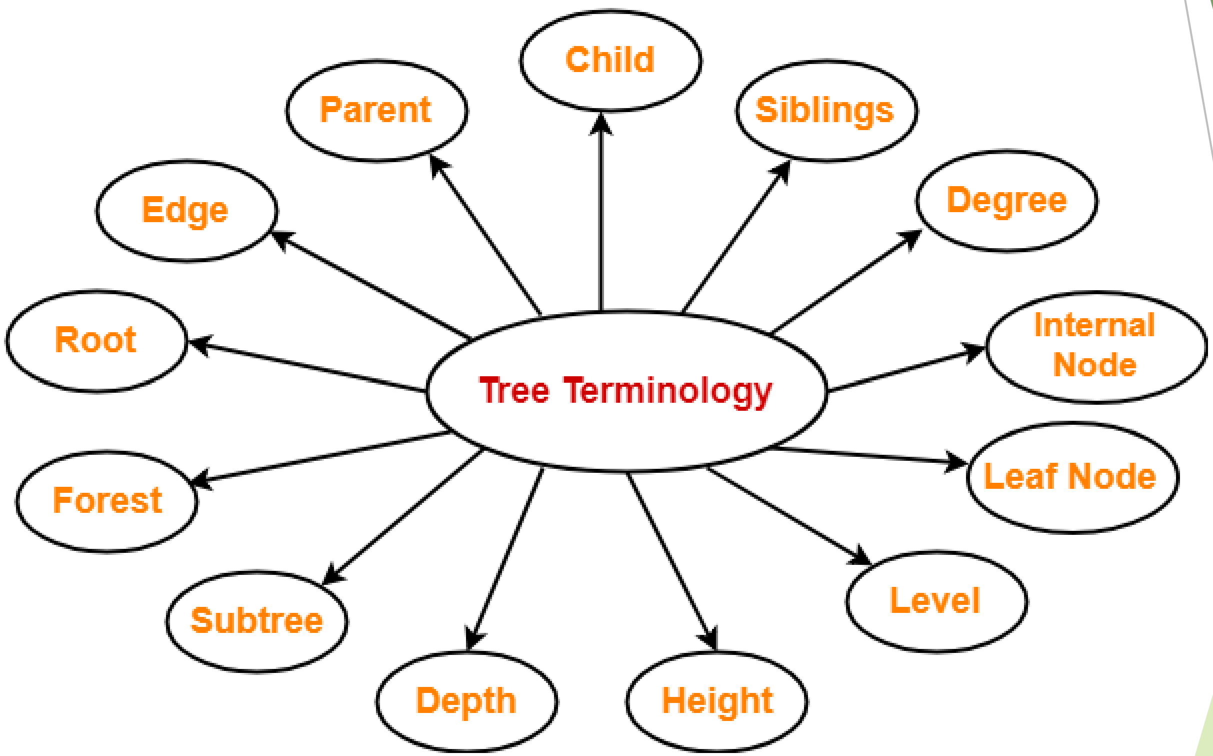


Binary Tree Traversal: Post-order

```
void post_order_traversal(struct node* root) {
   if(root != NULL) {
      post_order_traversal(root->leftChild);
      post_order_traversal(root->rightChild);
      printf("%d ", root->data);
   }
}
```



Tree Terminology



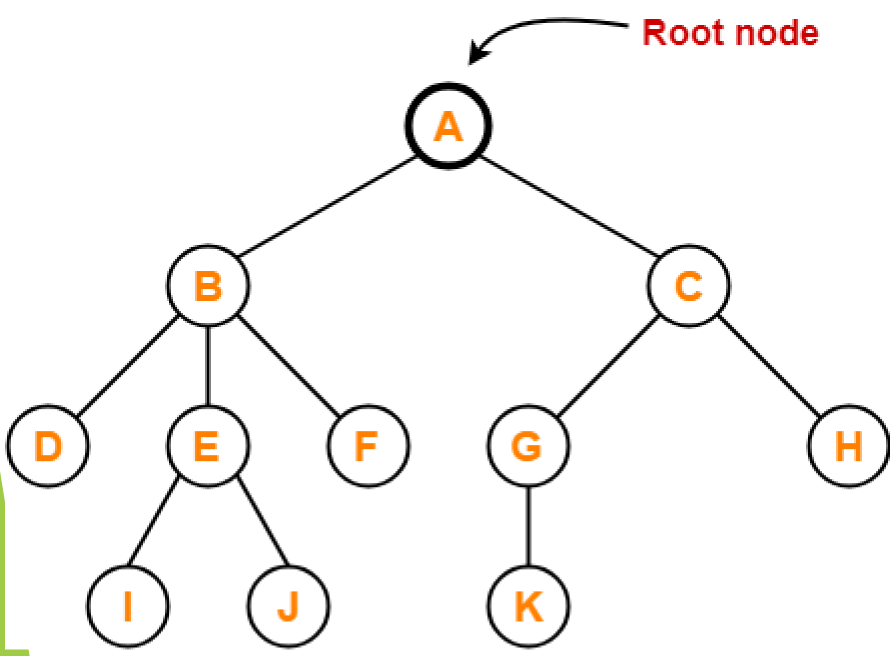
Tree Terminology:

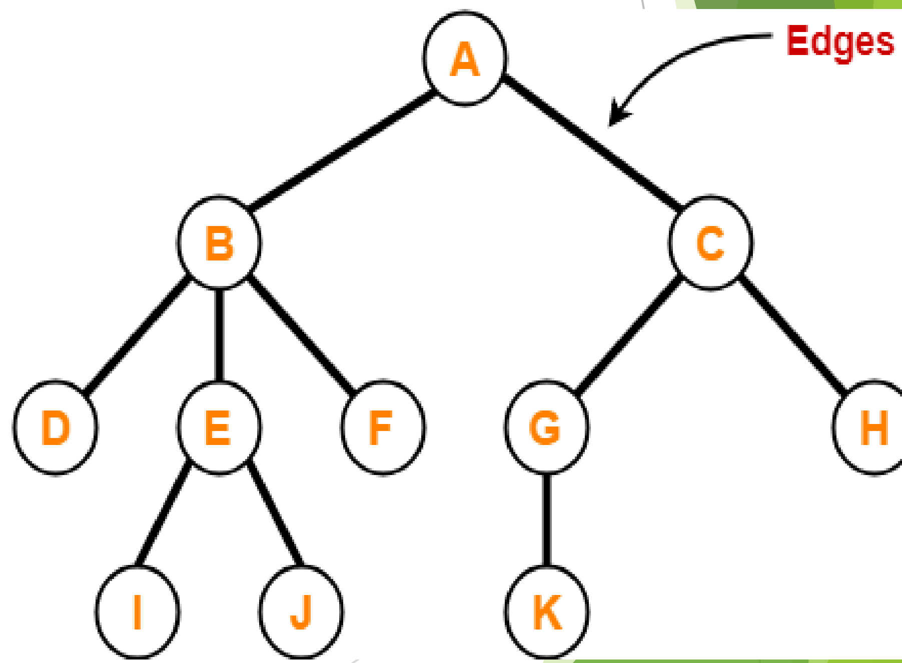
Root:

- The first node from where the tree originates is called as a **root node**.
- In any tree, there must be only one root node.

Edge:

- The connecting link between any two nodes is called as an **edge**.
- In a tree with n number of nodes, there are exactly (n-1) number of edges.

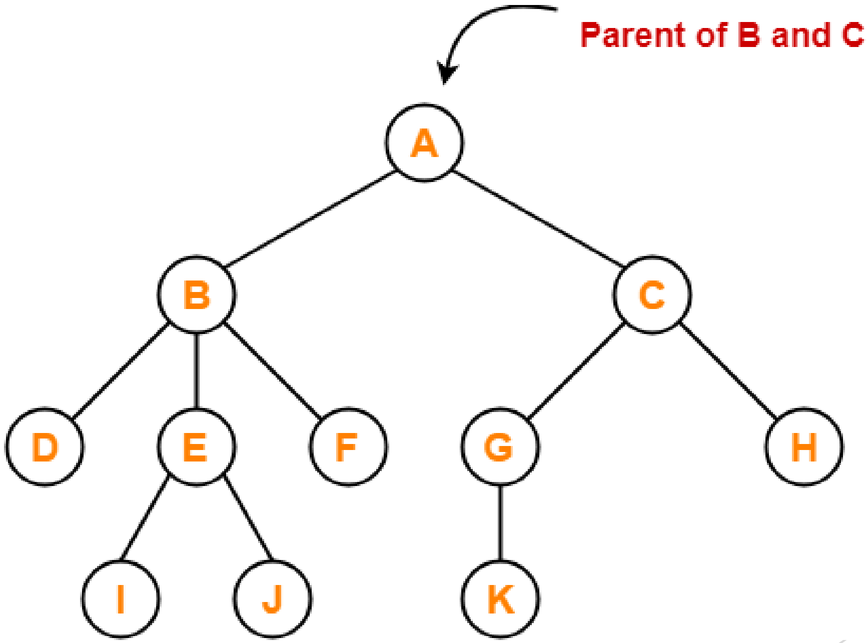




Tree Terminology:

Parent:

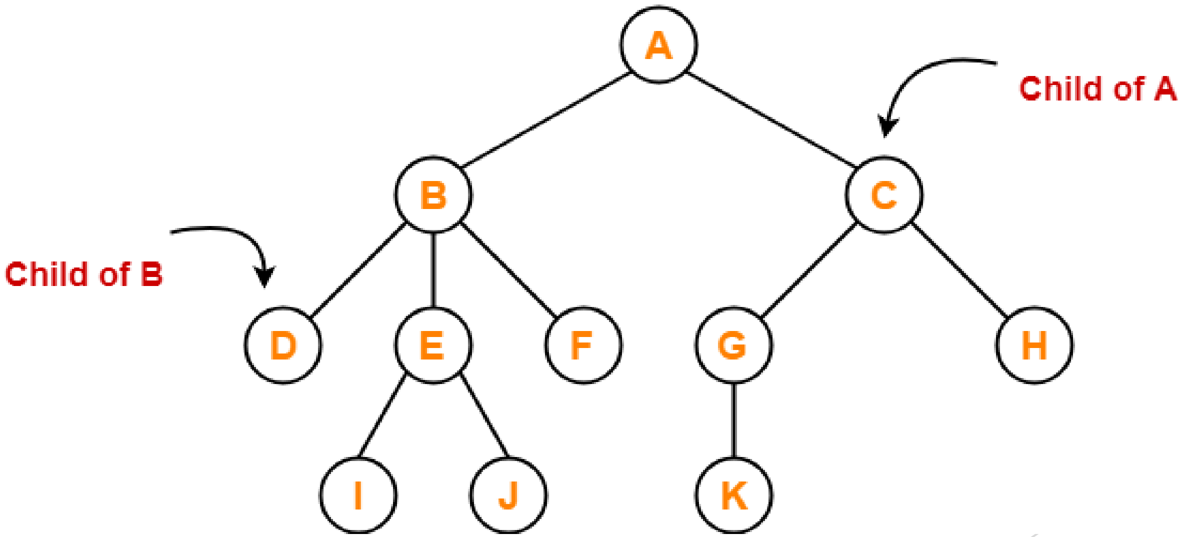
- The node which has a branch from it to any other node is called as a **parent node**.
- In other words, the node which has one or more children is called as a parent node.
- In a tree, a parent node can have any number of child nodes.



Tree Terminology:

Child:

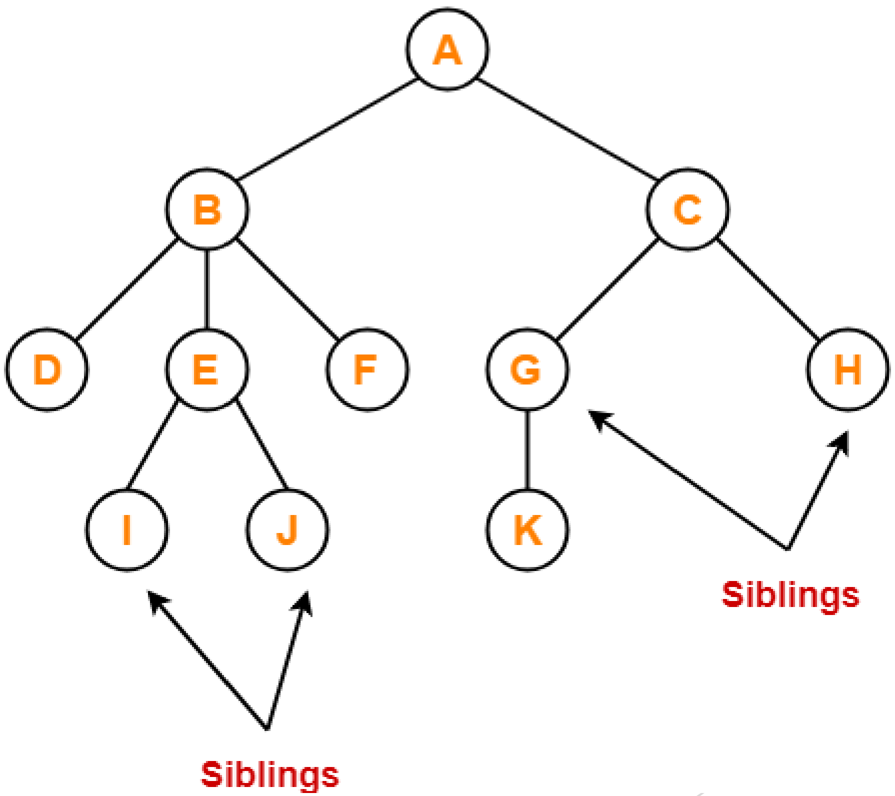
- The node which is a descendant of some node is called as a **child node**.
- All the nodes except root node are child nodes.



Tree Terminology:

Siblings:

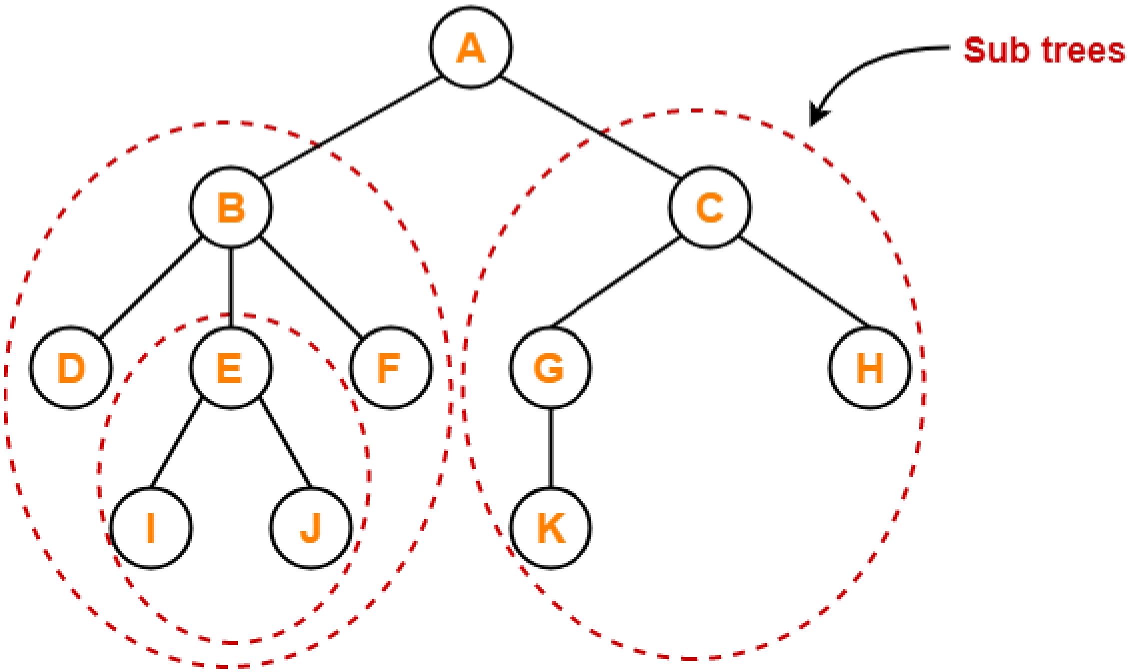
- Nodes which belong to the same parent are called as **siblings**.
- In other words, nodes with the same parent are sibling nodes.



Tree Terminology:

Subtree:

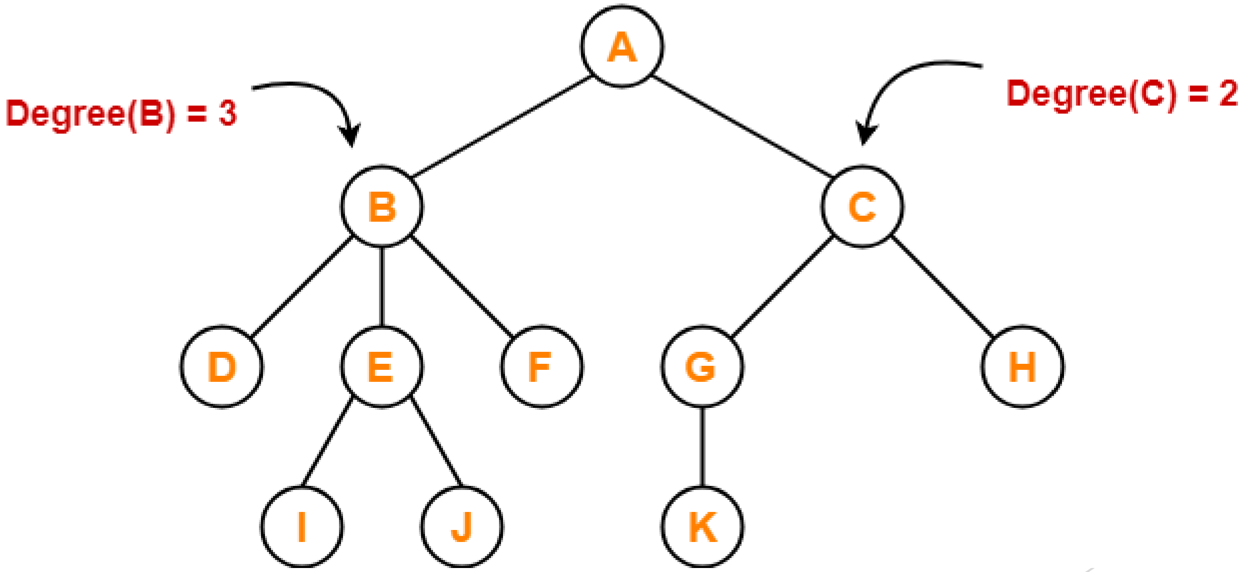
- In a tree, each child from a node forms a **subtree** recursively.
- Every child node forms a subtree on its parent node.



Tree Terminology:

Degree:

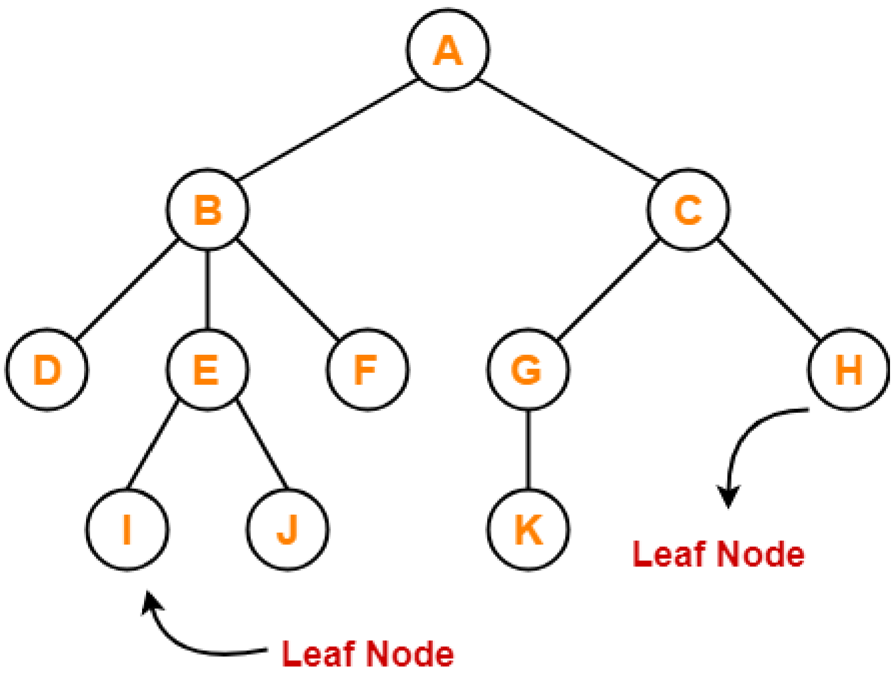
- **Degree of a node** is the total number of children of that node.
- **Degree of a tree** is the highest degree of a node among all the nodes in the tree.



Tree Terminology:

Leaf Node:

- The node which does not have any child is called as a **leaf node**.
- Leaf nodes are also called as **external nodes** or **terminal nodes**.

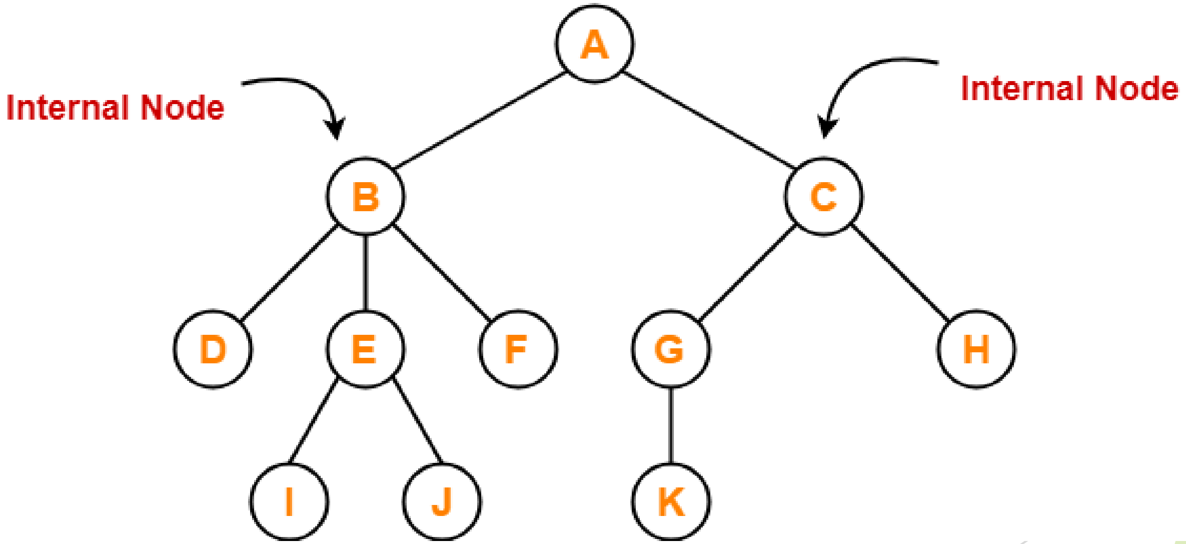


Tree Terminology:

Internal Node:

- The node which has at least one child is called as an **internal node**.
- Internal nodes are also called as **non-terminal nodes**.

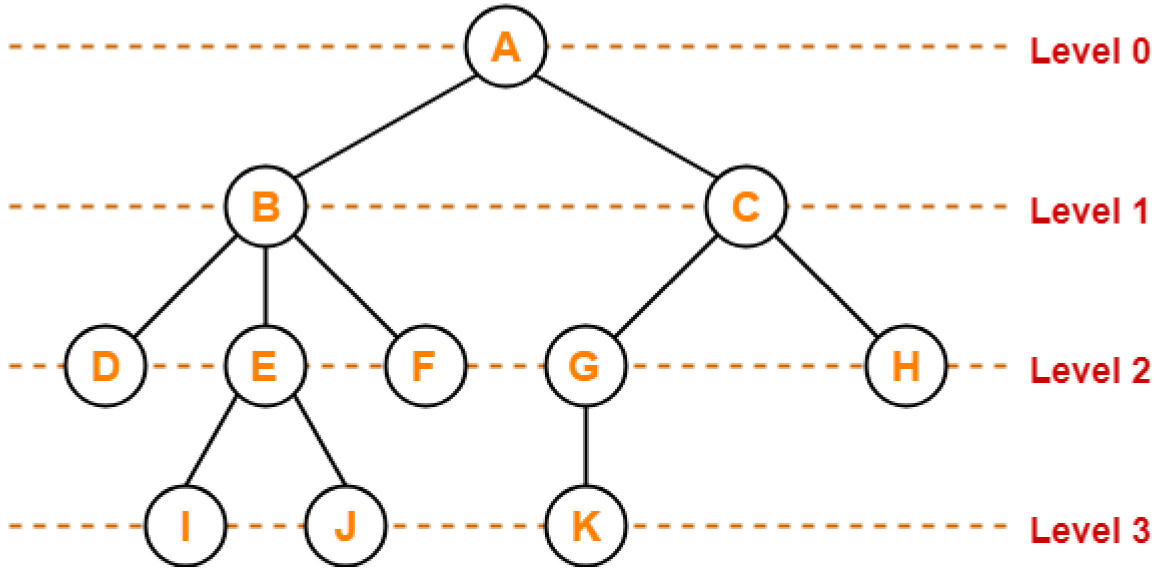
Every non-leaf node is an internal node.



Tree Terminology:

Level:

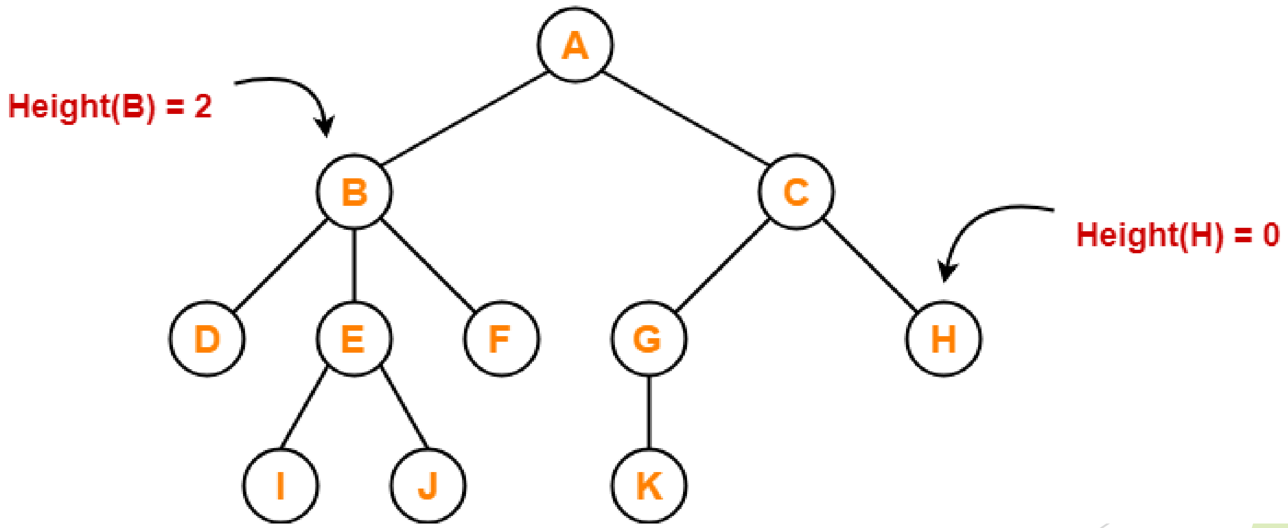
- In a tree, each step from top to bottom is called as **level of a tree**.
- The level count starts with 0 and increments by 1 at each level or step.



Tree Terminology:

Height:

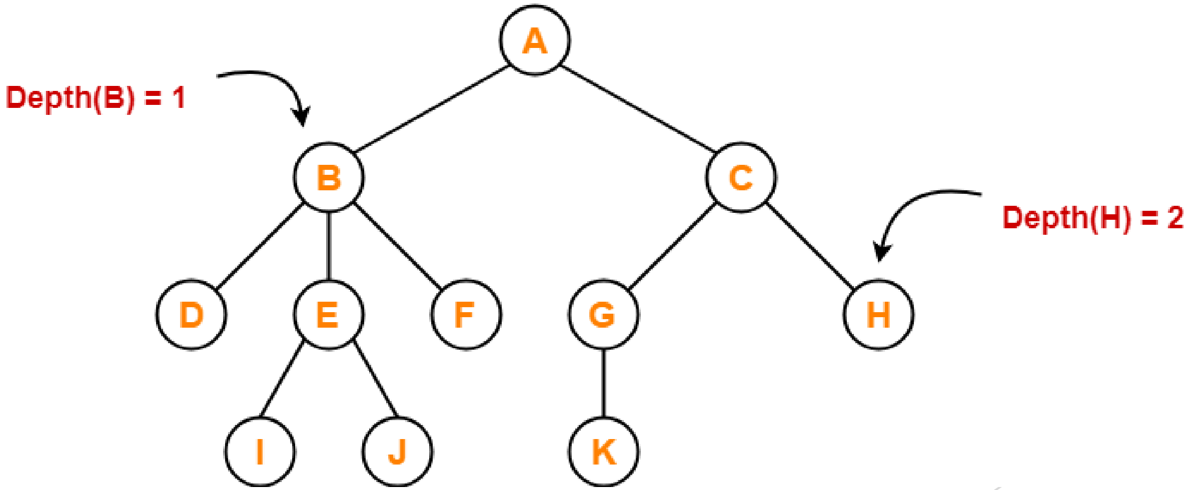
- Total number of edges that lies on the longest path from any leaf node to a particular node is called as **height of that node**.
- **Height of a tree** is the height of root node.
- Height of all leaf nodes = 0



Tree Terminology:

Depth-

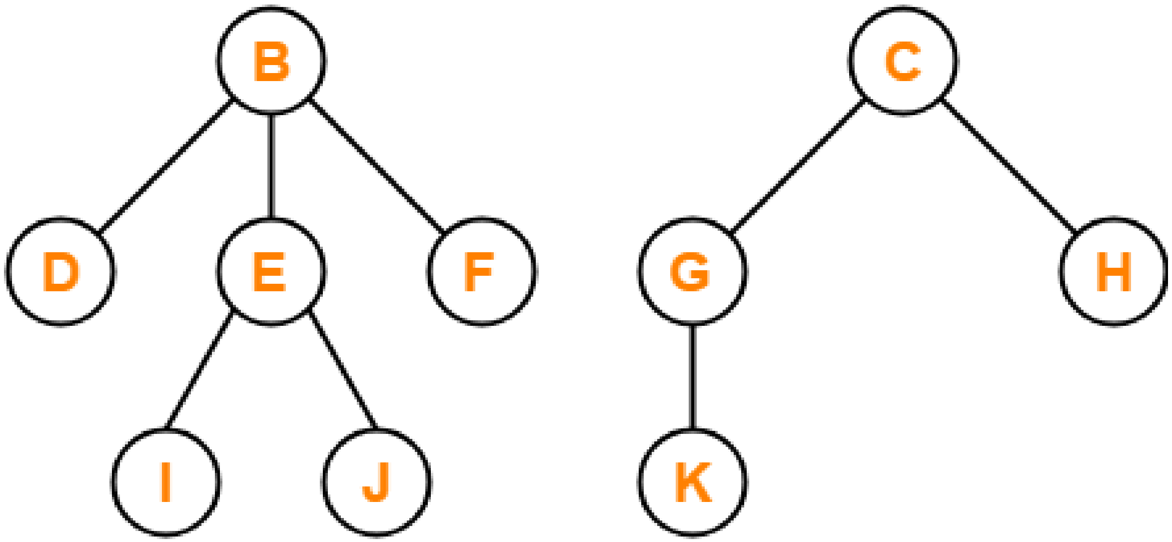
- Total number of edges from root node to a particular node is called as **depth of that node**.
- **Depth of a tree** is the total number of edges from root node to a leaf node in the longest path.
- Depth of the root node = 0
- The terms “level” and “depth” are used interchangeably.



Tree Terminology:

Forest:

A forest is a set of disjoint trees.



Forest

Construct Tree from given Inorder and Preorder traversals

- Inorder sequence: D B E A F C
  Preorder sequence: A B D E C F

```
          A
        /   \
      /       \
    B           C
   / \         /
 /     \     /
D        E  F
```

Algorithm: bTree()
1) Pick an element from Preorder. Increment a Preorder Index Variable (preIndex in below code) to pick the next element in the next recursive call.
2) Create a new tree node tNode with the data as the picked element.
3) Find the picked element's index in Inorder. Let the index be inIndex.
4) Call bTree for elements before inIndex and make the built tree as a left subtree of tNode.
5) Call bTree for elements after inIndex and make the built tree as a right subtree of tNode.
6) return tNode.

Construct a Binary Tree from Postorder and Inorder

```
in[] = {4, 8, 2, 5, 1, 6, 3, 7}
post[] = {8, 4, 5, 2, 6, 7, 3, 1}
```

```
          1

       /     \

     2         3

   /    \   /    \

  4      5  6     7

    \

      8
```

Thankyou!!!