Non Linear Datastructures

Non-Linear Data structures

- These are a type of data organization in which data elements do not form a simple, linear sequence.
- organize data in a hierarchical manner or in a graph-like structure.
- provide efficient ways to organize and manage complex data.
- Example: Trees, graphs

Importance of Non-Linear Data structures

- Efficient Data Management:
- Modeling real world structures:
- Optimized Performance:
- Algorithm Optimization:

Applications of Non-Linear Data structures

- Hierarchical data representation:
  - File Systems
  - XML/HTML Document Object Model
- Database Management
  - Indexes and Binary Trees
- Networking and communication:
  - Routing Algorithms
  - Social Network
- Web crawling, computer graphics and AI

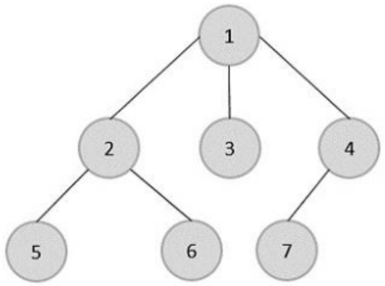
Types of Non-Linear Data structures

- Trees
- Graphs
- Heap

Trees

- A hierarchical structure that is used to represent and organize data in a way that is easy to navigate and search.

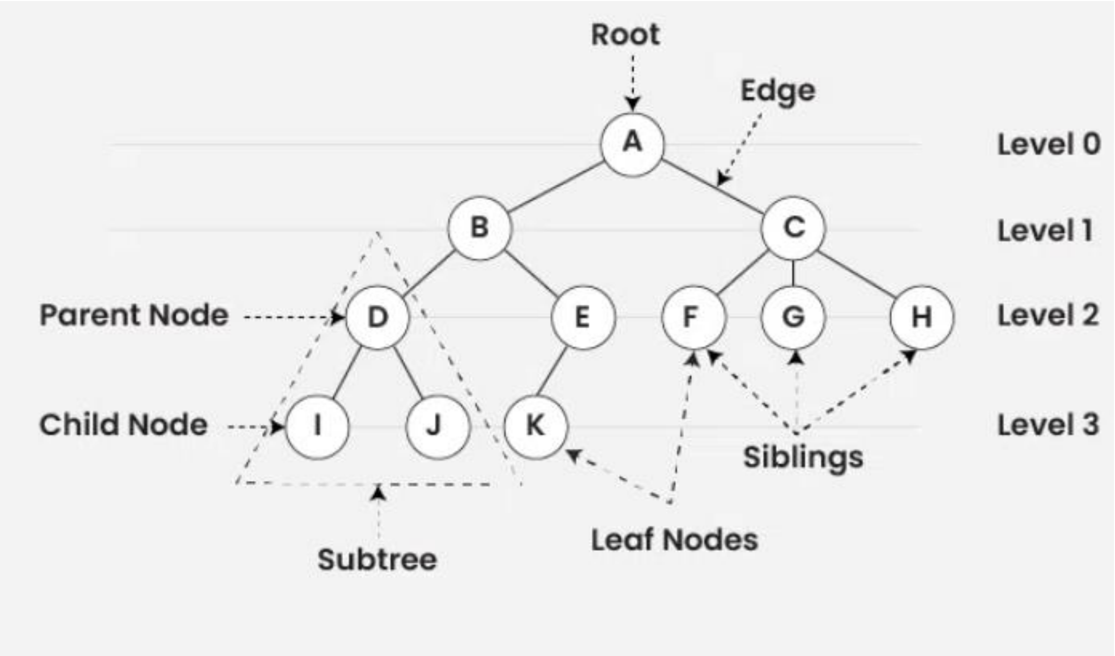
- It is a collection of nodes that are connected by edges and has a hierarchical relationship between the nodes.



General Tree Data Structure

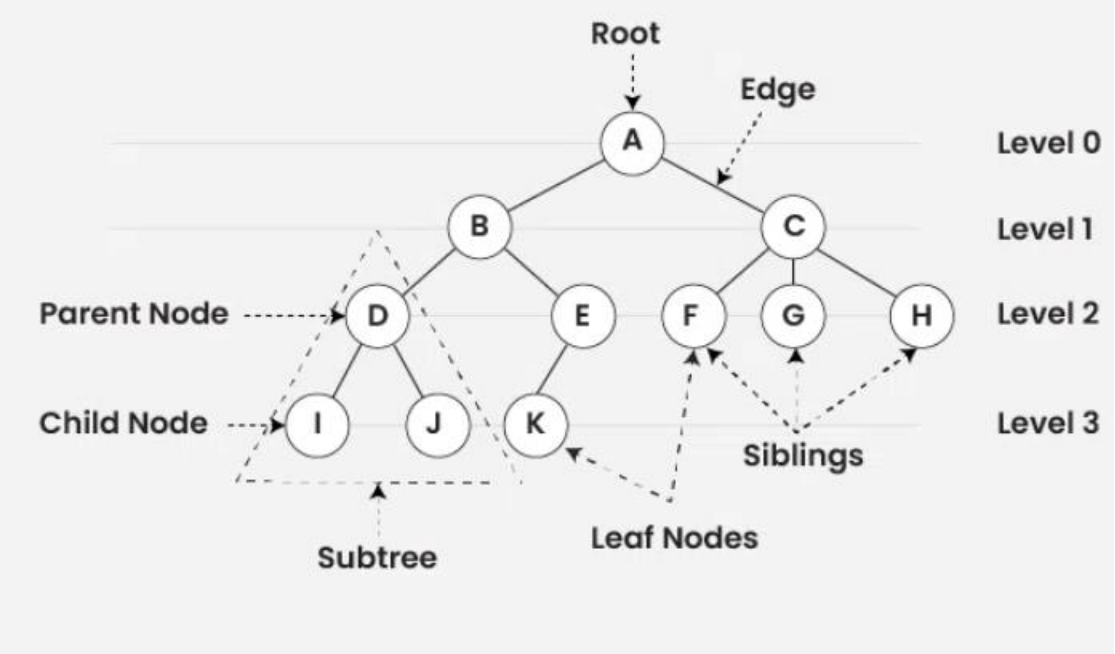
Basic Terminologies

- **Parent Node:**
  - The node which is an immediate predecessor of a node is called the parent node of that node. **{B}** is the parent node of **{D, E}**.



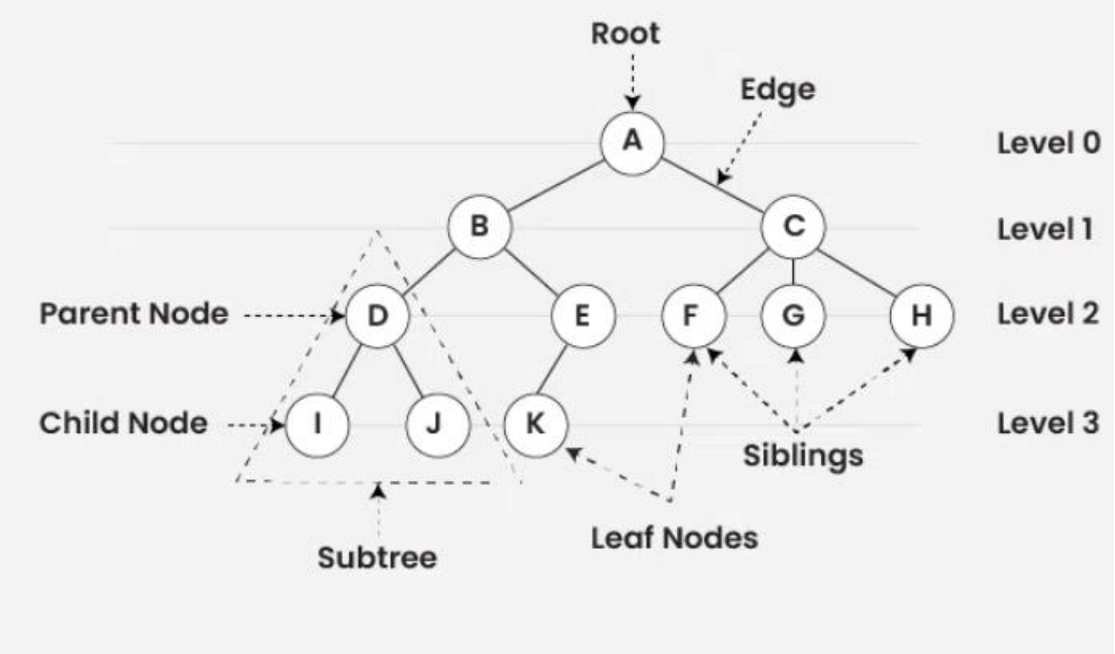
Basic Terminologies

- **Child Node:**
  - The node which is the immediate successor of a node is called the child node of that node. Examples: **{D, E}** are the child nodes of **{B}.**



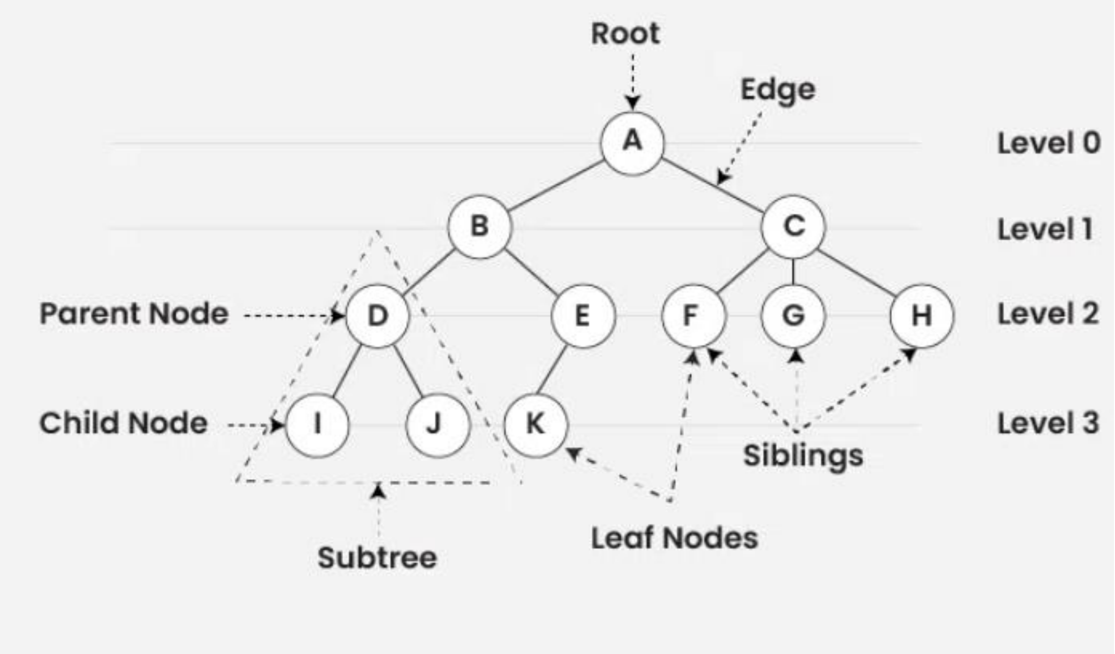
Basic Terminologies

- **Root Node:**
  - The topmost node of a tree or the node which does not have any parent node is called the root node. {A} is the root node of the tree



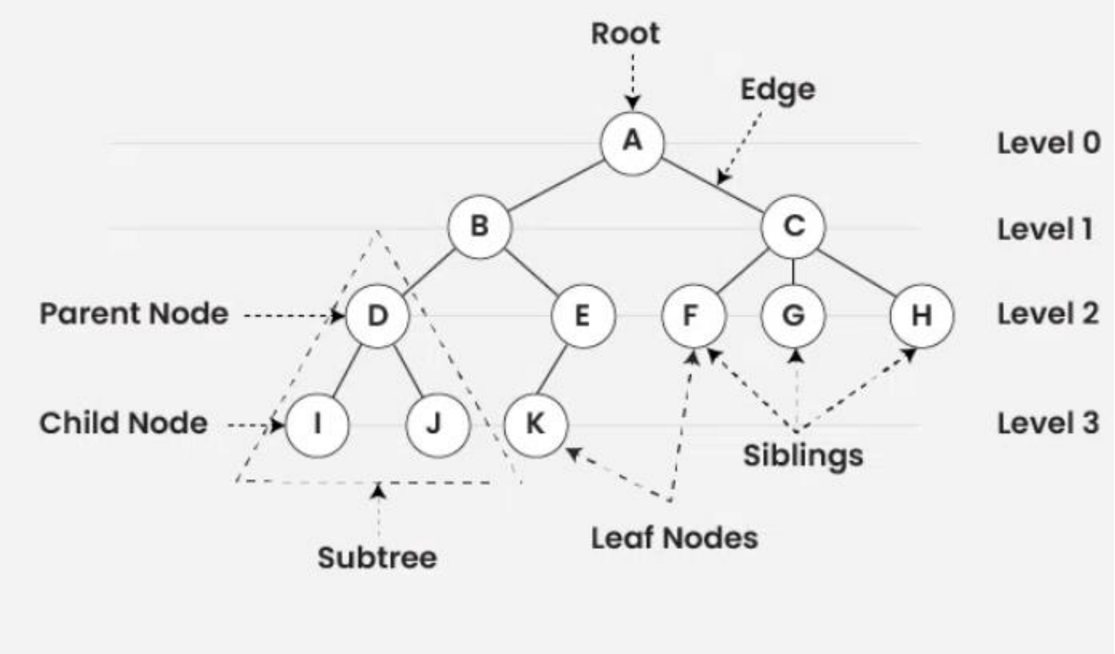
Basic Terminologies

- **Leaf Node or External Node:**
  - The nodes which do not have any child nodes are called leaf nodes. **{I, J, K, F, G, H}** are the leaf nodes of the tree.



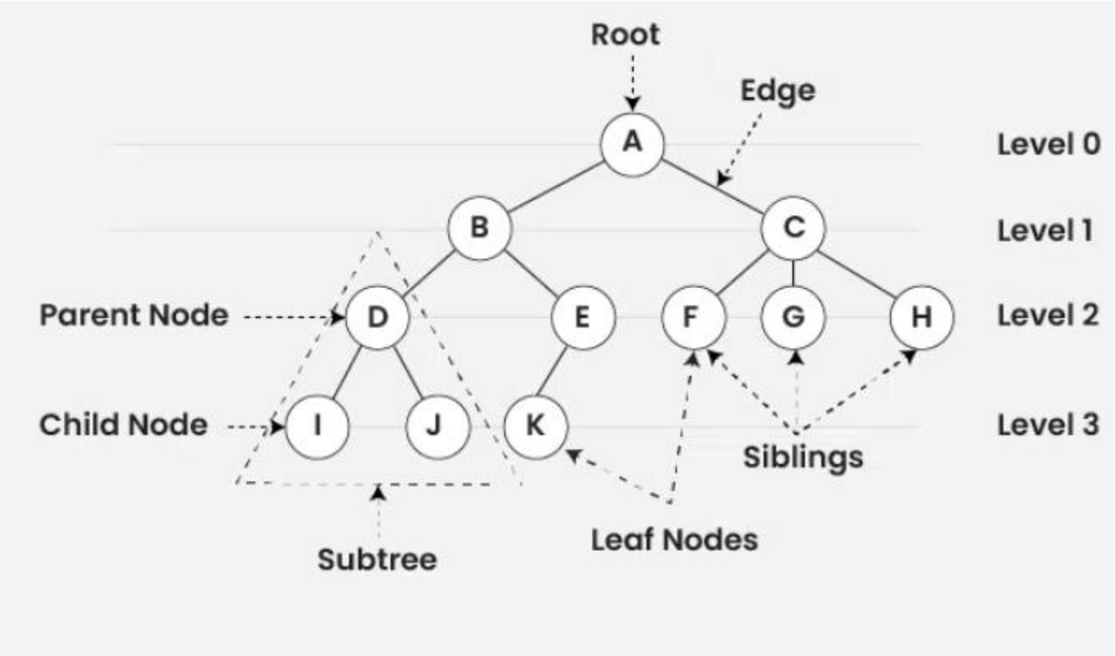
Basic Terminologies

- **Ancestor of a Node:**
  - Any predecessor nodes on the path of the root to that node are called Ancestors of that node. **{A,B}** are the ancestor nodes of the node **{E}**



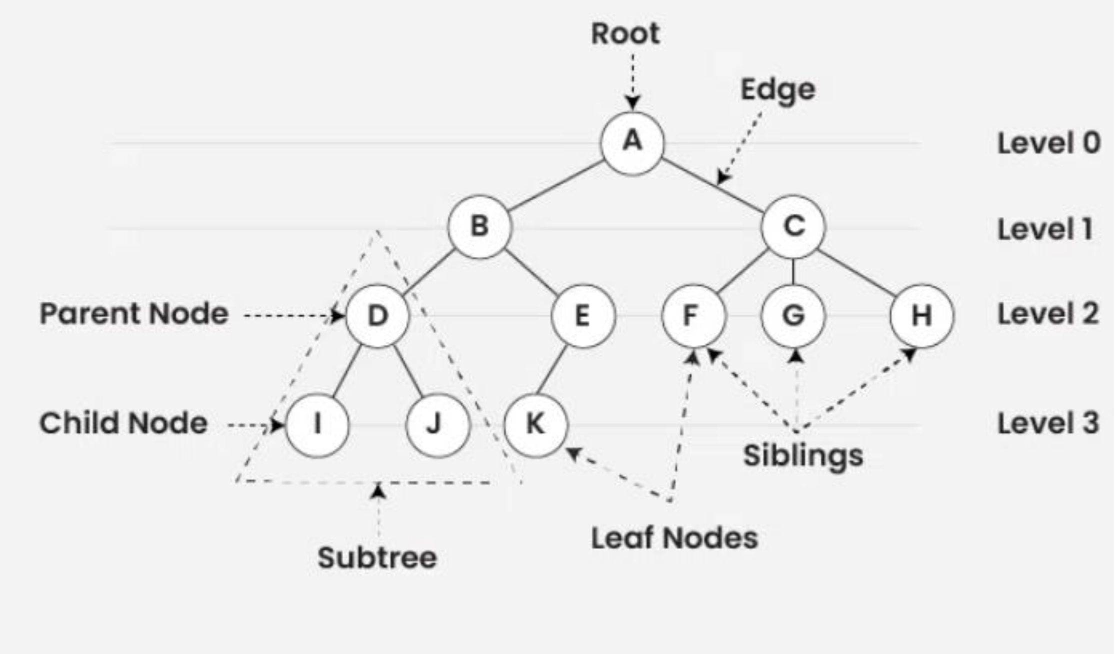
Basic Terminologies

- **Descendant:**
  - A node x is a descendant of another node y if and only if y is an ancestor of x.



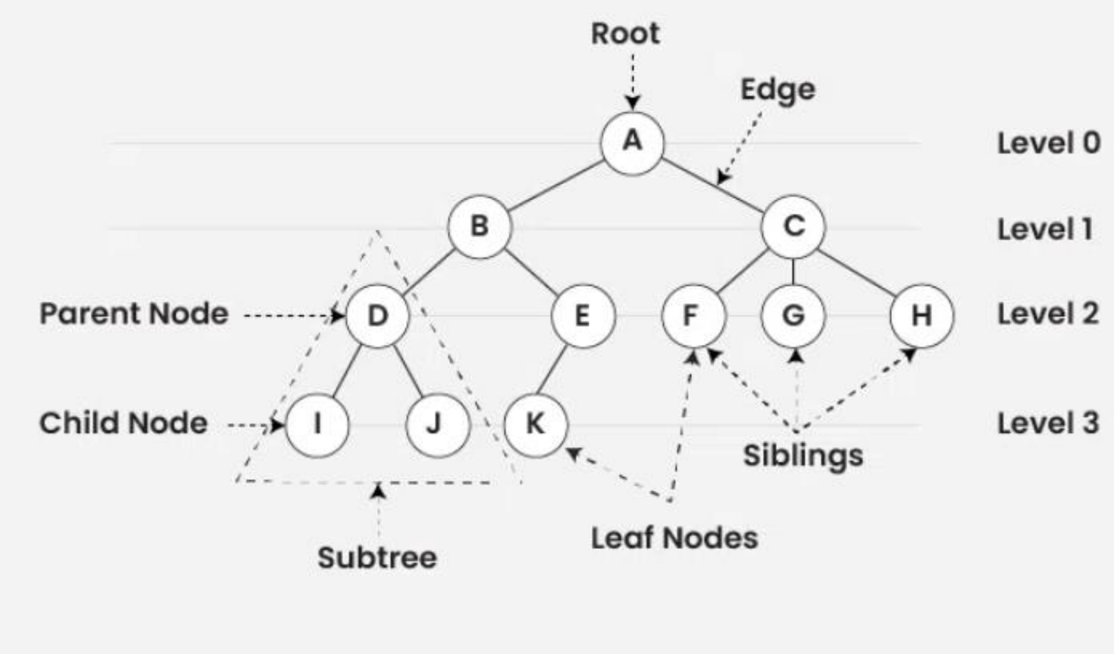
Basic Terminologies

- **Sibling:**
  - Children of the same parent node are called siblings. **{D,E}** are called siblings.



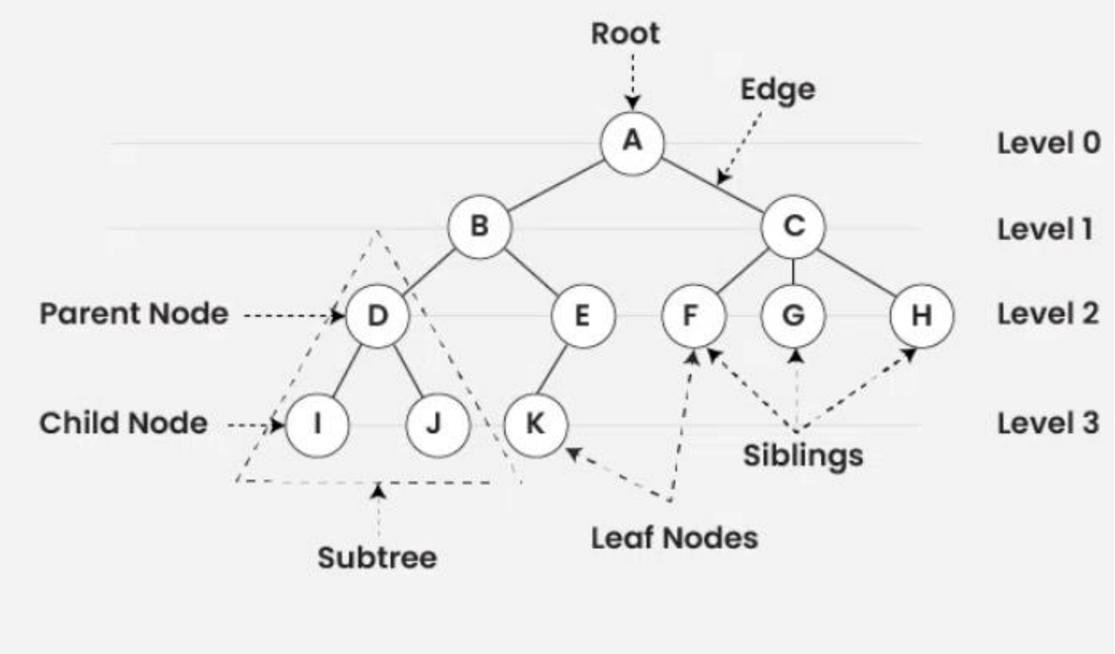
Basic Terminologies

- **Level of a node:**
  - The count of edges on the path from the root node to that node. The root node has level **0**.



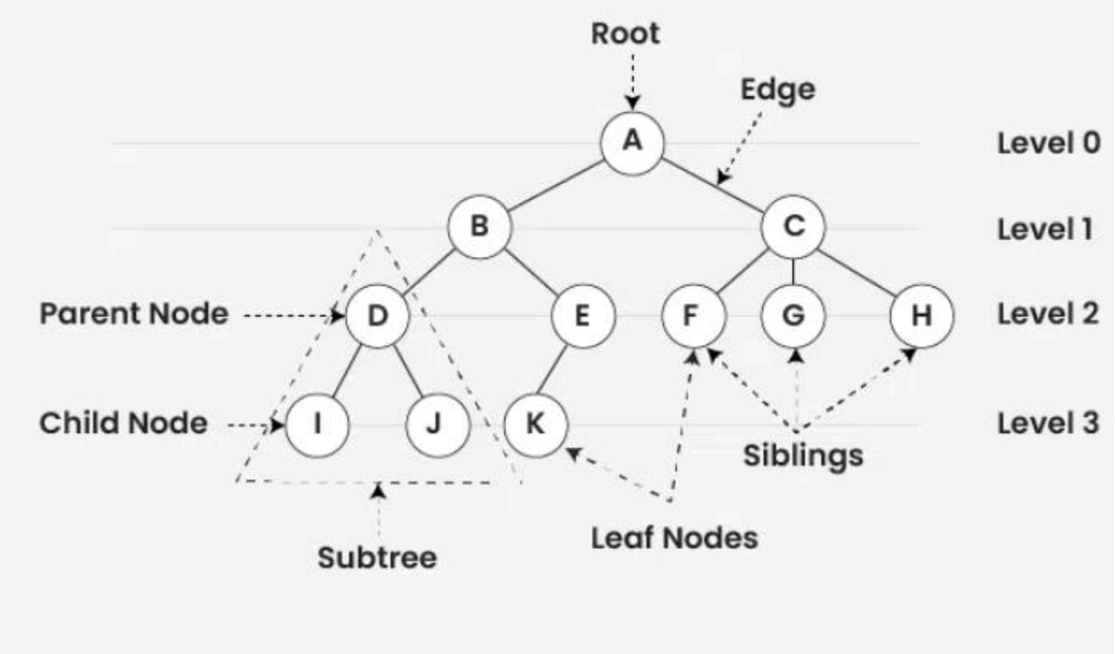
Basic Terminologies

- **Internal node:**
  - A node with at least one child is called Internal Node.



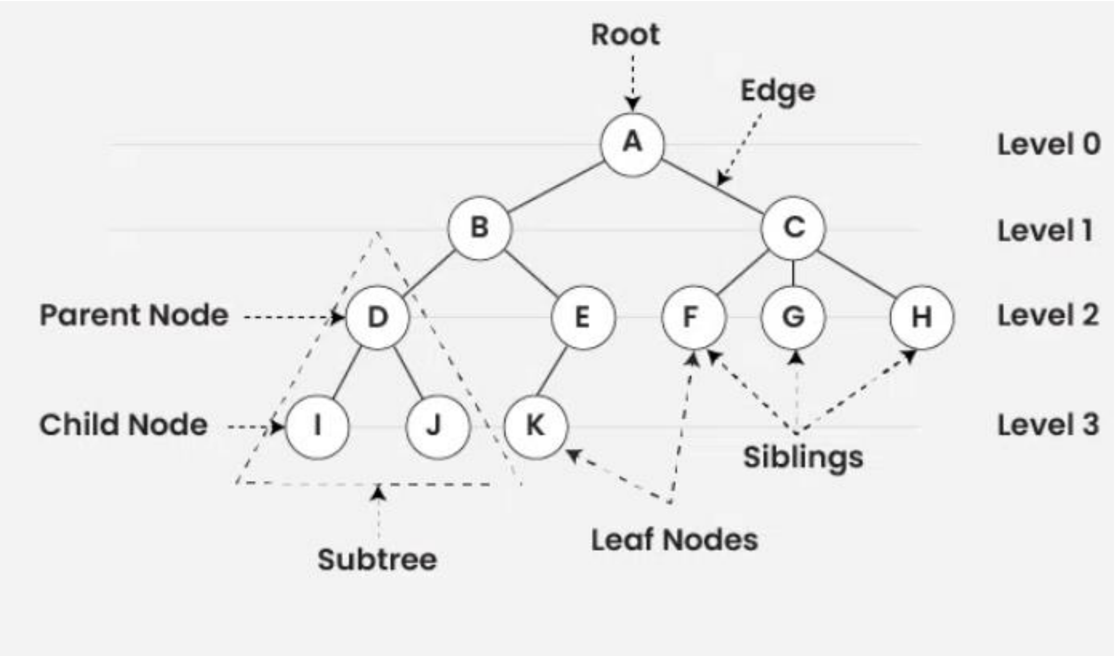
Basic Terminologies

- **Neighbour of a Node:**
  - Parent or child nodes of that node are called neighbors of that node.



Basic Terminologies

- **Subtree**:
  - Any node of the tree along with its descendant.



Representation of a Node

```
class Node {
public:
    int data;
    Node* first_child;
    Node* second_child;
    Node* third_child;
    .
    .
    .
    Node* nth_child;
};
```

Properties of Trees

- **Number of edges:**
  - An edge can be defined as the connection between two nodes. If a tree has N nodes then it will have (N-1) edges.
  - There is only one path from each node to any other node of the tree.

- **Depth of a node:**
  - The depth of a node is defined as the length of the path from the root to that node.
  - Each edge adds 1 unit of length to the path. So, it can also be defined as the number of edges in the path from the root of the tree to the node.

Properties of Trees

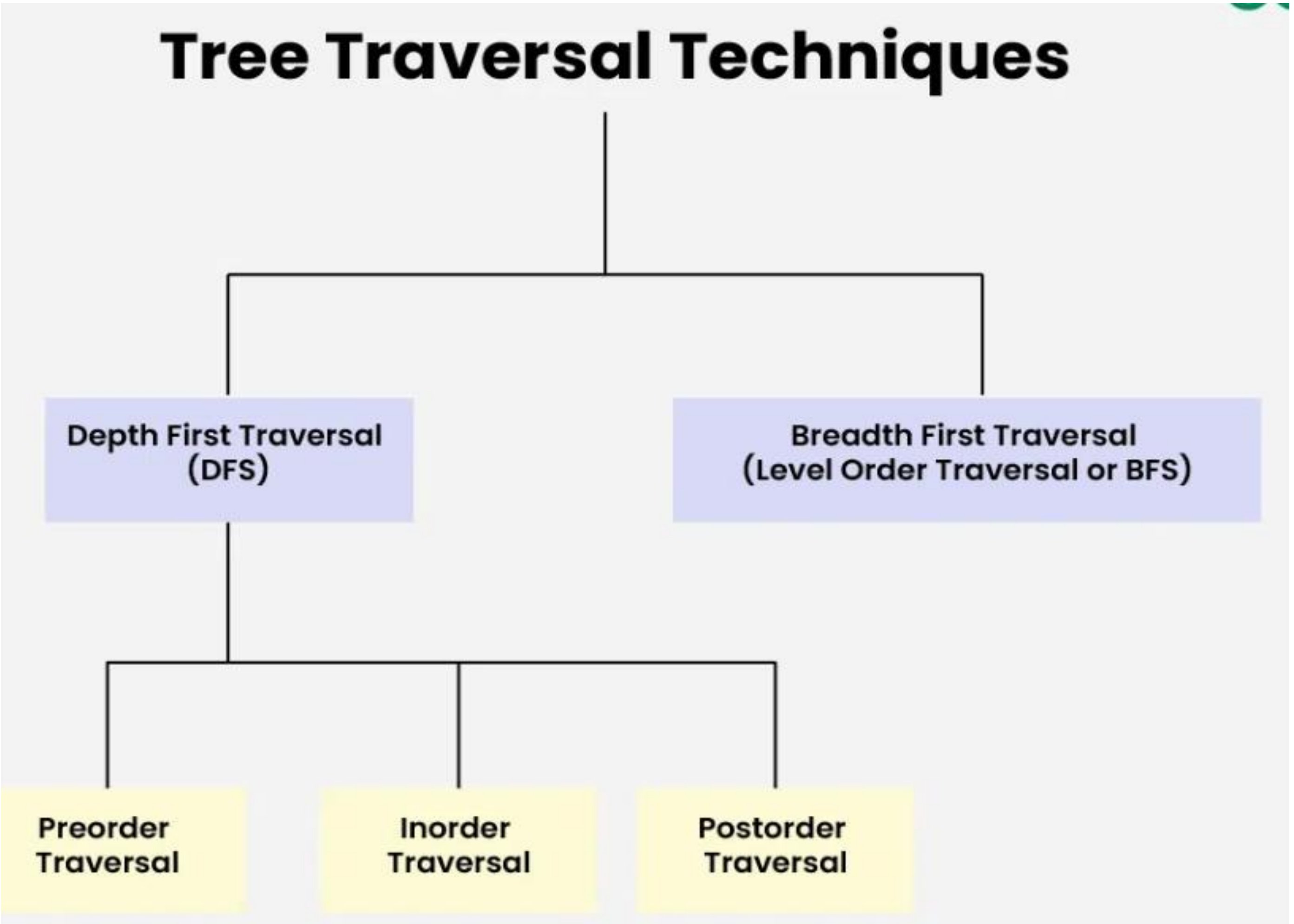
- **Height of a node:**
  - The height of a node can be defined as the length of the longest path from the node to a leaf node of the tree.
- **Height of the Tree:**
  - The height of a tree is the length of the longest path from the root of the tree to a leaf node of the tree.
- **Degree of a Node:**
  - The total count of subtrees attached to that node is called the degree of the node. The degree of a leaf node must be **0**.
  - The degree of a tree is the maximum degree of a node among all the nodes in the tree.

Tree Traversal

- **Tree Traversal** refers to the process of visiting or accessing each node of the tree exactly once in a certain order.

- Unlike linear data structures, Trees can be traversed in many ways

Tree Traversal



In-order Traversal

Inorder traversal visits the node in the order: **Left -> Root -> Right**

Algorithm: *if(node==null)*
return

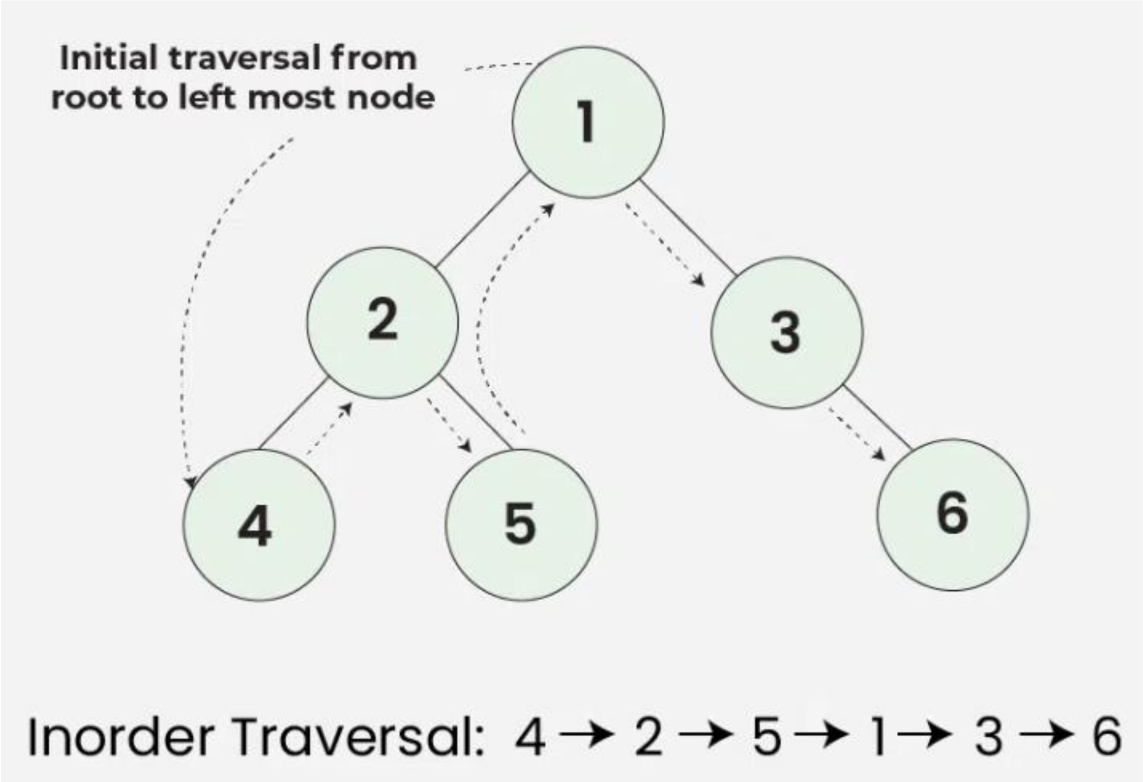
Inorder(tree)

Traverse the left subtree, i.e., call Inorder(left->subtree)

Visit the root.

Traverse the right subtree, i.e., call Inorder(right->subtree)

In-order Traversal



inorder(5)
Inorder(2)
Inorder(1)

Pre-order Traversal

Preorder traversal visits the node in the order: **Root -> Left -> Right**

Algorithm:

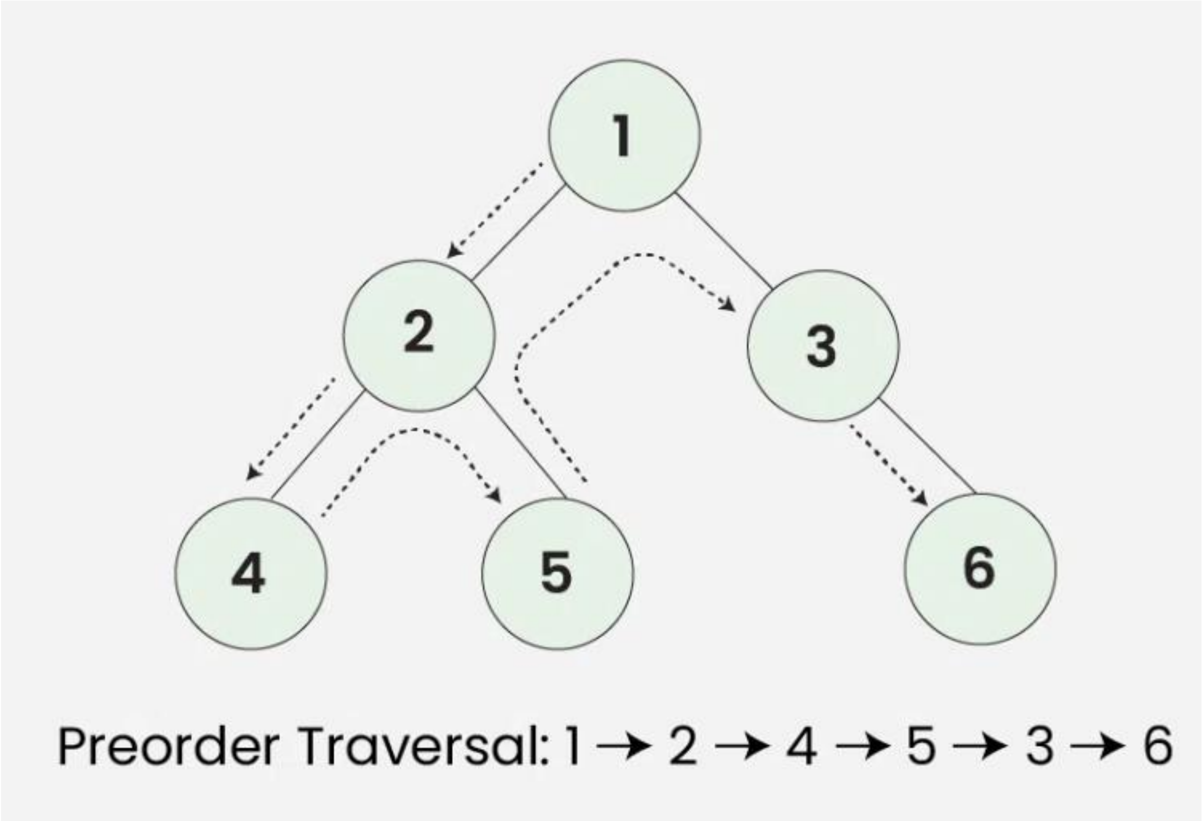
Preorder(tree)

Visit the root.

Traverse the left subtree, i.e., call Preorder(left->subtree)

Traverse the right subtree, i.e., call Preorder(right->subtree)

Pre-order Traversal



Post-order Traversal

Postorder traversal visits the node in the order: **Left -> Right -> Root**

Algorithm:

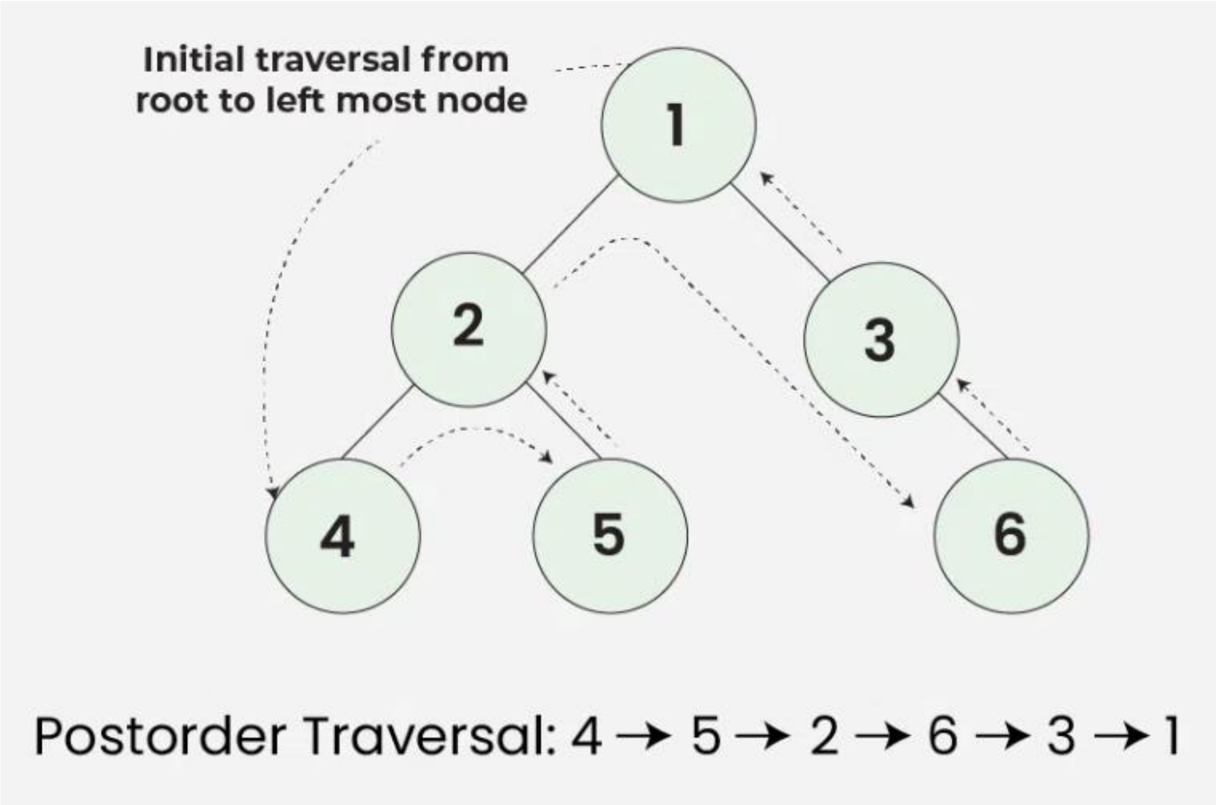
Postorder(tree)

Traverse the left subtree, i.e., call Postorder(left->subtree)

Traverse the right subtree, i.e., call Postorder(right->subtree)

Visit the root

Post-order Traversal



Level-order Traversal

It visits all nodes present in the same level completely before visiting the next level.
Algorithm:

LevelOrder(tree)

Create an empty queue Q
Enqueue the root node of the tree to Q
Loop while Q is not empty
    Dequeue a node from Q and visit it
    Enqueue the left child of the dequeued node if it exists
    Enqueue the right child of the dequeued node if it exists

In-order Traversal