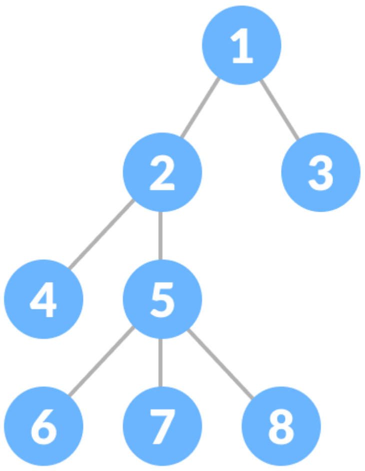
Tree Data Structure

A tree is a nonlinear hierarchical data structure that consists of nodes connected by edges.

This data structure is a specialized method to organize and store data in the computer to be used more effectively. It consists of a central node, structural nodes, and sub-nodes, which are connected via edges. We can also say that tree data structure has roots, branches, and leaves connected with one another.



Why Tree Data Structure?

Arrays, LinkedLists, Stacks and Queues are Linear data structure that store data sequentially.

In Order to perform any operation, the time complexity increases with the increase of data size.

But it is not acceptable in today’s computational world.

That is why the Tree Data Structure came into the Frame.

Tree Data structure is a non-linear data structure which allows quicker and easier access to the data.

Tree Terminologies

The data in a tree are not stored in a sequential manner i.e, they are not stored linearly. Instead, they are arranged on multiple levels or we can say it is a hierarchical structure. For this reason, the tree is considered to be a non-linear data structure.

Parent Node: The node which is a predecessor of a node is called the parent node of that node.

Child Node: The node which is the immediate successor of a node is called the child node of that node.

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

Leaf Node or External Node: The nodes which do not have any child nodes are called leaf nodes

Ancestor of a Node: Any predecessor nodes on the path of the root to that node are called Ancestors of that node.

Descendant: Any successor node on the path from the leaf node to that node.

Sibling: Children of the same parent node are called siblings.

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

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

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

Subtree: Any node of the tree along with its descendant Properties of a Tree:

- **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.

- **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.

Some more properties are:

- Traversing in a tree is done by depth first search and breadth first search algorithm.
- It has no loop and no circuit
- It has no self-loop
- Its hierarchical model.

Basic Operation Of Tree:

1. Create – create a tree in data structure.
2. Insert − Inserts data in a tree.
3. Search − Searches specific data in a tree to check it is present or not.
4. Preorder Traversal – perform Traveling a tree in a pre-order manner in data structure .
5. In order Traversal – perform Traveling a tree in an in-order manner.
6. Post order Traversal –perform Traveling a tree in a post-order manner.

Tree Applications

- Binary Search Trees(BSTs) are used to quickly check whether an element is present in a set or not.
- Heap is a kind of tree that is used for heap sort.
- A modified version of a tree called Tries is used in modern routers to store routing information.
- Most popular databases use B-Trees and T-Trees, which are variants of the tree structure we learned above to store their data
- Compilers use a syntax tree to validate the syntax of every program you write.

Types of Tree

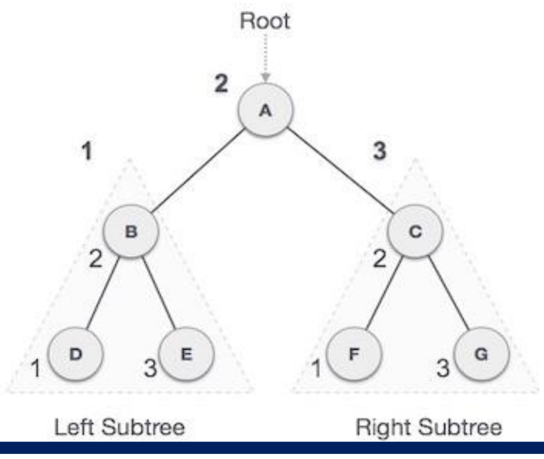
1. Binary Tree
2. Binary Search Tree
3. AVL Tree
4. B-Tree

Tree Traversal - inorder, preorder and postorder

Inorder traversal

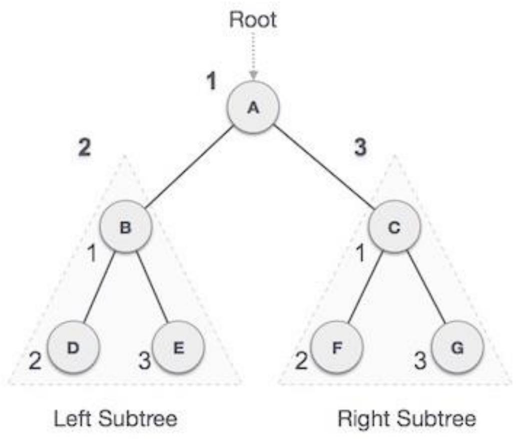
1. First, visit all the nodes in the left subtree
2. Then the root node
3. Visit all the nodes in the right subtree

Left Root Right



Preorder traversal

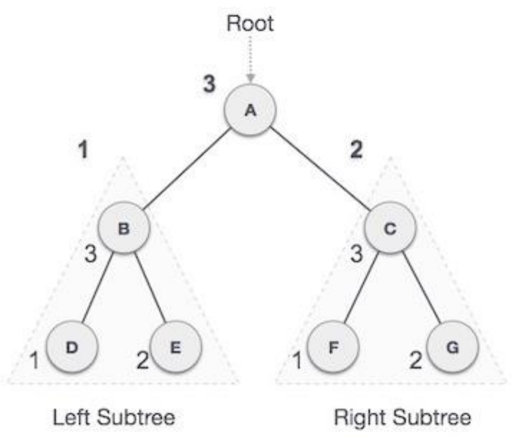
1. Visit root node
2. Visit all the nodes in the left subtree
3. Visit all the nodes in the right subtree



Postorder traversal

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

Left-right-node



Program Implementation

```
#include <iostream>
using namespace std;
struct Node {
 int data;
 struct Node *left, *right;
 Node(int data) {
  this->data = data;
  left = right = NULL;
 }
};
// Preorder traversal
void preorderTraversal(struct Node* node) {
 if (node == NULL)
  return;
 cout << node->data << "->";
 preorderTraversal(node->left);
 preorderTraversal(node->right);
}
int main() {
 struct Node* root = new Node(1);
 root->left = new Node(12);
 root->right = new Node(9);
 root->left->left = new Node(5);
 root->left->right = new Node(6);
 cout << "Inorder traversal ";
 inorderTraversal(root);
 cout << "\nPreorder traversal ";
 preorderTraversal(root);
 cout << "\nPostorder traversal ";
 postorderTraversal(root);
}
```

```
// Postorder traversal
void postorderTraversal(struct Node* node) {
 if (node == NULL)
  return;
 postorderTraversal(node->left);
 postorderTraversal(node->right);
 cout << node->data << "->";
}
```

```
// Inorder traversal
void inorderTraversal(struct Node* node) {
 if (node == NULL)
  return;
 inorderTraversal(node->left);
 cout << node->data << "->";
 inorderTraversal(node->right);
}
```