# Data Structures
# Lab Manual 8


## Topic: Trees

Session: Spring 2023

Faculty of Information Technology

UCP Lahore Pakistan

## Objectives

The objectives of this lab are to learn basic tree operations. Trees are one of the most important data structures in computing. They appear in various forms and have a wide range of uses as given follows:

1. Data structures such as arrays, linked list, stack, and queue are linear data structures that store data sequentially. In order to perform any operation in a linear data structure, the time complexity increases with the increase in the data size, but it is not acceptable in today's computational world.
2. Trees (with some ordering e.g., BST) provide moderate access/search (quicker than Linked List and slower than arrays).
3. Trees provide moderate insertion/deletion (quicker than Arrays and slower than Unordered Linked Lists).
4. Like Linked Lists and unlike Arrays, Trees don't have an upper limit on the number of nodes as nodes are linked using pointers.
5. One reason to use trees might be because you want to store information that naturally forms a hierarchy. For example, the file system on a computer.

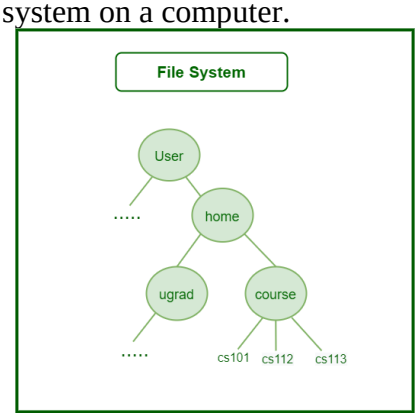



**FIGURE 1: FILE SYSTEM**

## Tree Terminologies

A tree is a nonlinear hierarchical data structure that consists of nodes connected by edges. Following are some of its terminologies.

**Node:** A node is an entity that contains a key or value and pointers to its child nodes.

- **Parent Node:** The node which is a predecessor of a node is called the parent node of that node. {Q} is the parent node of {A, B}.
- **Child Node:** The node which is the immediate successor of a node is called the child node of that node. Examples: {A, B} are the child nodes of {Q}.
- **Leaf Node or External Node:** The nodes which do not have any child nodes are called leaf nodes. {E, F, G, H, I} are the leaf nodes of the tree.
- **Root Node:** The topmost node of a tree or the node which does not have any parent node is called the root node. {P} is the root node of the tree. A non-empty tree must contain exactly one root node and exactly one path from the root to all other nodes of the tree.

**Edge:** It is the link between any two nodes.

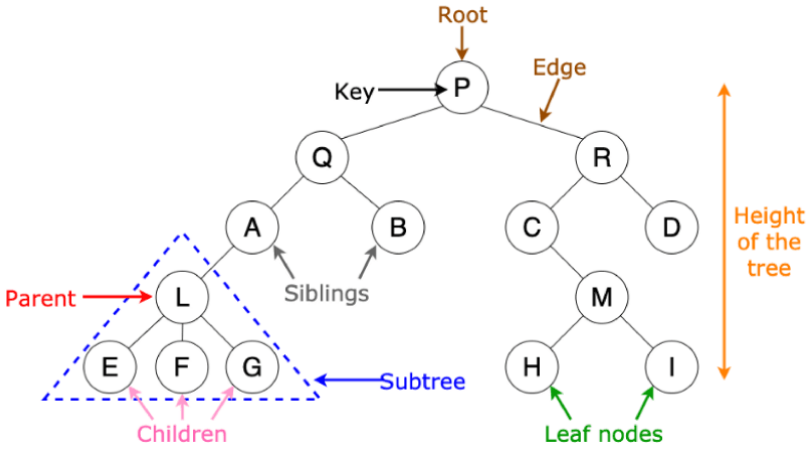



FIGURE 2: TREE- TERMINOLOGIES

## 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.

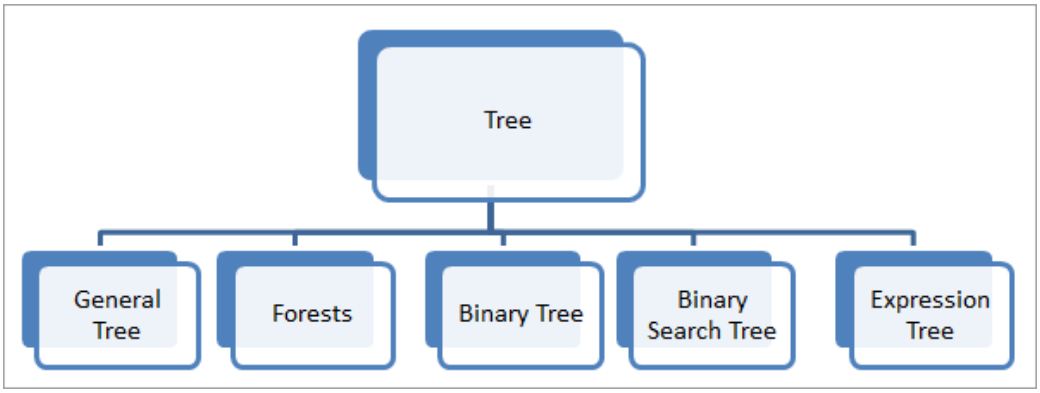
**Height of a node:** The height of a node can be defined as the length of the longest path from the node to a leaf node of the tree.

**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.

**Types of Trees:**

The tree data structure can be classified into the following subtypes as shown in the below diagram.



**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.

**Implementation of Binary Tree:**

```
#include<iostream>
using namespace std;
// Structure of each node of the tree

struct node {
	int data;
	struct node* left;
	struct node* right;
};

struct node * create()
{
	int x;		// Line 1
	struct node * newNode; // Line 2
	newNode = new node;// Line 3
	cout << " Enter value (-1 for no child): "; // Line 4
	cin >> x; // Line 5
	if (x == -1) // Line 6
		return 0; // Line 7
	newNode->data = x; // Line 8
	cout << endl << "Left child of " << x; // Line 9
	newNode->left = create(); // Line 10
	cout << endl << "Right child of " << x; // Line 11
	newNode->right = create(); // Line 12
	return newNode; // Line 13
}

void preOrder(struct node * root)
{
	if (root == NULL)
		return;
	cout << root->data << "\t";
	preOrder(root->left);
	preOrder(root->right);
}

void InOrder(struct node * root)
{
	if (root == NULL)
		return;
	InOrder(root->left);
	cout << root->data << "\t";
	InOrder(root->right);
}

void PostOrder(struct node * root)
{
```

```
	if (root == NULL)
		return;
	PostOrder(root->left);
	PostOrder(root->right);
	cout << root->data << "\t";

}



void main()
{
	struct node * root;
	root = 0;
	root = create();

	cout << endl << "PreOrder Traversal is: ";
	preOrder(root);
	cout << endl;

	cout << endl << "InOrder Traversal is: ";
	InOrder(root);
	cout << endl;

	cout << endl << "PostOrder Traversal is: ";
	PostOrder(root);
	cout << endl;

	system("pause");
}
```

**Task 1**: Consider a ternary tree that stores integer values. Each node in the tree has three children pointers. Implement the following operations:

1. Initialize an empty ternary tree.
2. Insert a new node with a given value into the ternary tree. Prompt the user to input the values for child 1, child 2, and child 3 of the new node.
3. Perform a traversal order root, child 1, child 2, and child 3 of the ternary tree and print the node values.
4. Perform a traversal order child 1, child 2, child 3 and root of the ternary tree and print the node values.

**Task 2**: In the above task after adding multiple values perform search operation in different traversal order like InOrderTraversal, PreOrderTraversal and PostOrderTraversal with different values. And find traversal order where least number of nodes have to visit in order to find the number searched.