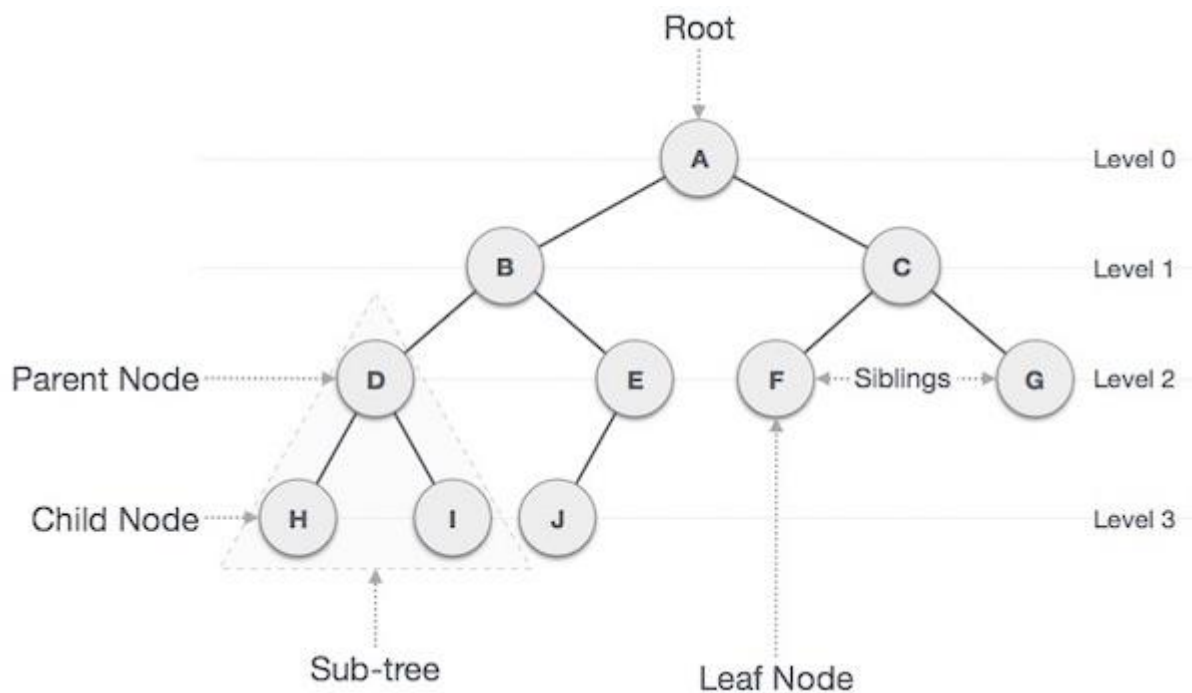


TREE

Tree represents the nodes connected by edges. We will discuss binary tree or binary search tree specifically.

BINARY TREE:

Binary Tree is a special data structure used for data storage purposes. A binary tree has a special condition that each node can have **a maximum of two children**. A binary tree has the benefits of both an ordered array and a linked list as search is as quick as in a sorted array and insertion or deletion operation are as fast as in linked list.



Important Terms

Following are the important terms with respect to tree.

- **Path** – Path refers to the sequence of nodes along the edges of a tree.
- **Root** – The node at the top of the tree is called root. There is only one root per tree and one path from the root node to any node.
- **Parent** – Any node except the root node has one edge upward to a node called parent.
- **Child** – The node below a given node connected by its edge downward is called its child node.
- **Leaf** – The node which does not have any child node is called the leaf node.
- **Subtree** – Subtree represents the descendants of a node.
- **Visiting** – Visiting refers to checking the value of a node when control is on the node.
- **Traversing** – Traversing means passing through nodes in a specific order.

- **Levels** – Level of a node represents the generation of a node. If the root node is at level 0, then its next child node is at level 1, its grandchild is at level 2, and so on.
- **keys** – Key represents a value of a node based on which a search operation is to be carried out for a node.

BINARY TREE TRAVERSAL:

Traversal is a process to visit all the nodes of a tree and may print their values too. Because, all nodes are connected via edges (links) we always start from the root (head) node. That is, we cannot randomly access a node in a tree. There are three ways which we use to traverse a tree –

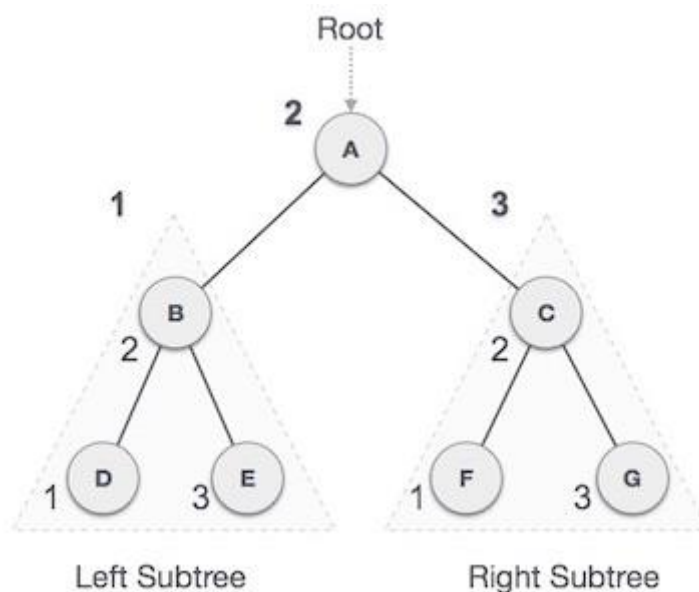
- In-order Traversal
- Pre-order Traversal
- Post-order Traversal

Generally, we traverse a tree to search or locate a given item or key in the tree or to print all the values it contains.

In-order Traversal

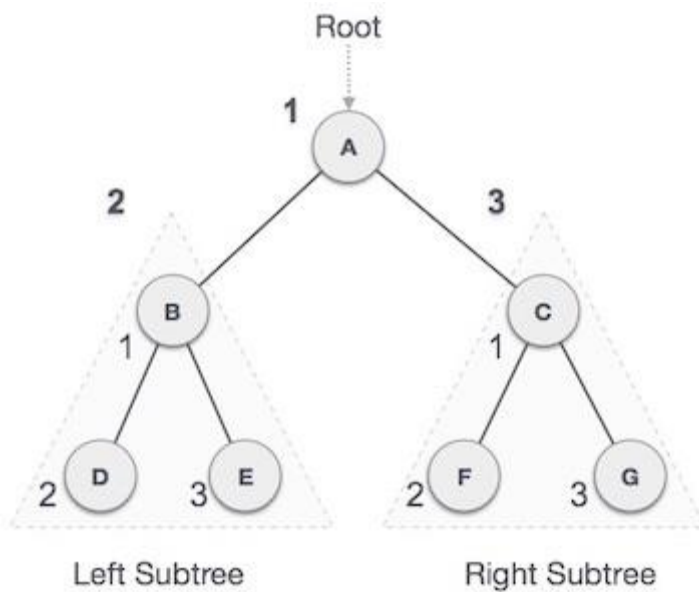
In this traversal method, the left subtree is visited first, then the root and later the right sub-tree. We should always remember that every node may represent a subtree itself.

If a binary tree is traversed **in-order**, the output will produce sorted key values in an ascending order.



Pre-order Traversal

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



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

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

Algorithm

Until all nodes are traversed –

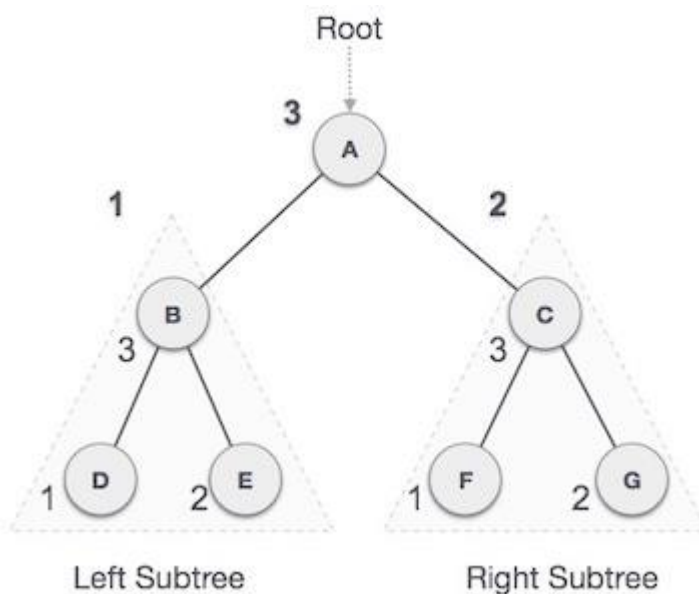
Step 1 – Visit root node.

Step 2 – Recursively traverse left subtree.

Step 3 – Recursively traverse right subtree.

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.



We start from **A**, and following Post-order traversal, we first visit the left subtree **B**. **B** is also traversed post-order. The process goes on until all the nodes are visited. The output of post-order traversal of this tree will be –

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

Algorithm

Until all nodes are traversed –

Step 1 – Recursively traverse left subtree.

Step 2 – Recursively traverse right subtree.

Step 3 – Visit root node.

Tree Node

The code to write a tree node would be similar to what is given below. It has a data part and references to its left and right child nodes.

```
struct node {  
    int data;  
    struct node *leftChild;  
    struct node *rightChild;  
};
```

In a tree, all nodes share common construct.

Binary Search Tree(BST) :

A Binary Search Tree (BST) is a tree in which all the nodes follow the below-mentioned properties –

- The left sub-tree of a node has a key less than or equal to its parent node's key.
- The right sub-tree of a node has a key greater than to its parent node's key.

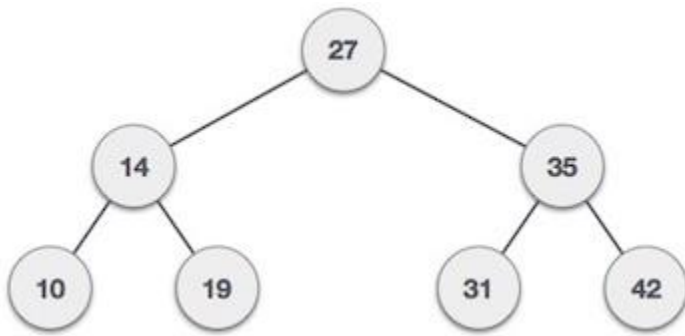
Thus, BST divides all its sub-trees into two segments; the left sub-tree and the right sub-tree and can be defined as –

$\text{left_subtree (keys)} \leq \text{node (key)} < \text{right_subtree (keys)}$

Representation

BST is a collection of nodes arranged in a way where they maintain BST properties. Each node has a key and an associated value. While searching, the desired key is compared to the keys in BST and if found, the associated value is retrieved.

Following is a pictorial representation of BST –



We observe that the root node key (27) has all less-valued keys on the left sub-tree and the higher valued keys on the right sub-tree.

BST Basic Operations

The basic operations that can be performed on a binary search tree data structure, are the following –

- **Insert** – Inserts an element in a tree/create a tree.
- **Search** – Searches an element in a tree.
- **Preorder Traversal** – Traverses a tree in a pre-order manner.
- **Inorder Traversal** – Traverses a tree in an in-order manner.
- **Postorder Traversal** – Traverses a tree in a post-order manner.

Insert Operation

The very first insertion creates the tree. Afterwards, whenever an element is to be inserted, first locate its proper location. Start searching from the root node, then if the data is less than the key value, search for the empty location in the left subtree and insert the data. Otherwise, search for the empty location in the right subtree and insert the data.

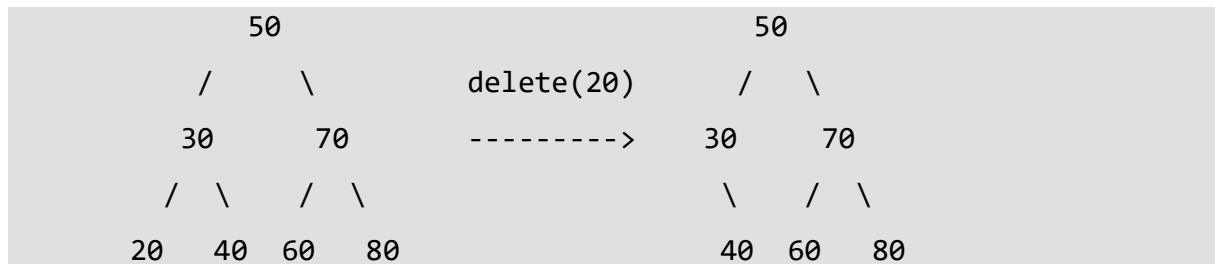
Search Operation

Whenever an element is to be searched, start searching from the root node, then if the data is less than the key value, search for the element in the left subtree. Otherwise, search for the element in the right subtree.

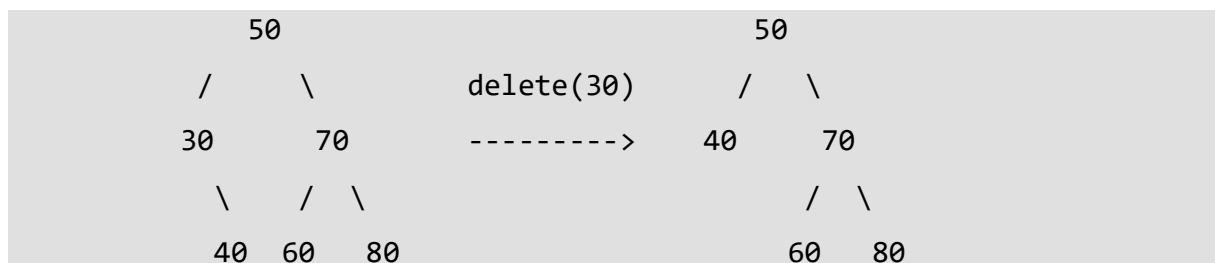
Binary Search Tree (Delete)

We have discussed [BST search and insert operations](#). In this post, delete operation is discussed. When we delete a node, three possibilities arise.

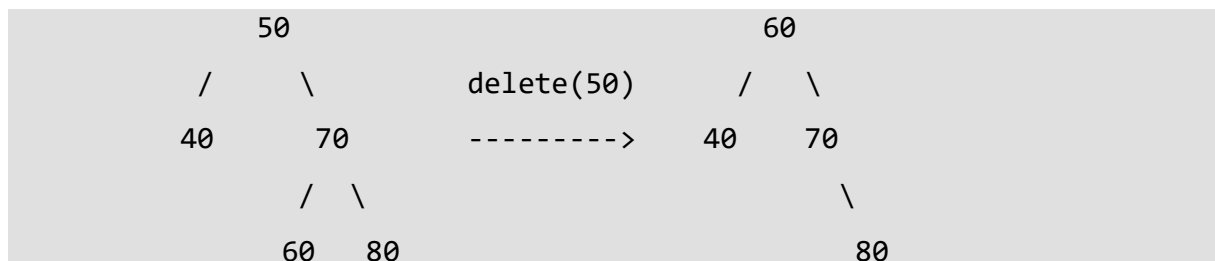
1) Node to be deleted is leaf: Simply remove from the tree.



2) Node to be deleted has only one child: Copy the child to the node and delete the child



3) Node to be deleted has two children: Find **inorder successor** of the node. Copy contents of the inorder successor to the node and delete the inorder successor. Note that inorder predecessor can also be used.

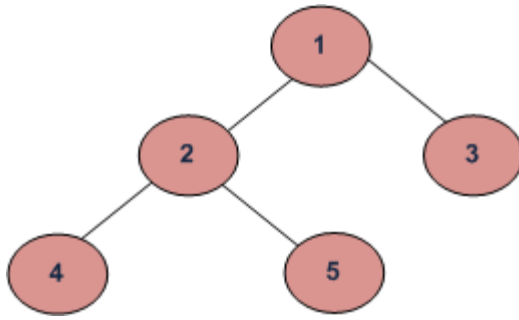


Inoder traversal: 40 **50** 60 70 80

Here inorder successor of 50 is 60. First delete the inorder successor(60) and replace 50 by this inorder successor.

Assignment:

1. Find out Inorder, Preorder and Postorder traversal of the following tree.



2. Construct a BST from given data.
 - a) 60 20 50 80 70 30 10 90
 - b) 7 4 9 1 3 5 2 8
 - c) Delete node 10 after constructing a BST from (a).
 - d) Delete node 50 after constructing a BST from (a).
 - e) Delete node 60 after constructing a BST from (a).