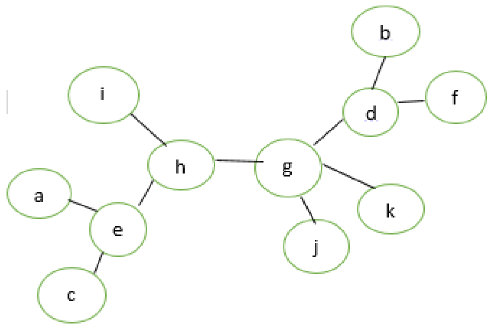
Tree:- A connected graph without any circuit is called a Tree. In other words, a tree is an undirected graph G that satisfies any of the following equivalent conditions:

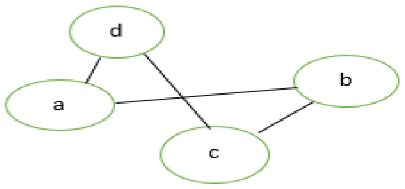
- Any two vertices in G can be connected by a unique simple path.
- G is acyclic, and a simple cycle is formed if any edge is added to G.
- G is connected and has no cycles.
- G is connected but would become disconnected if any single edge is removed from G.

For Example:

- This Graph is a Tree:



- This Graph is not a Tree:



Binary Trees:

If the outdegree of every node is less than or equal to 2, in a directed tree than the tree is called a binary tree. A tree consisting of the nodes (empty tree) is also a binary tree. A binary tree is shown in fig:

Basic Terminology:

Root: A binary tree has a unique node called the root of the tree.

Left Child: The node to the left of the root is called its left child.

Right Child: The node to the right of the root is called its right child

Parent: A node having a left child or right child or both are called the parent of the nodes.

Siblings: Two nodes having the same parent are called siblings.

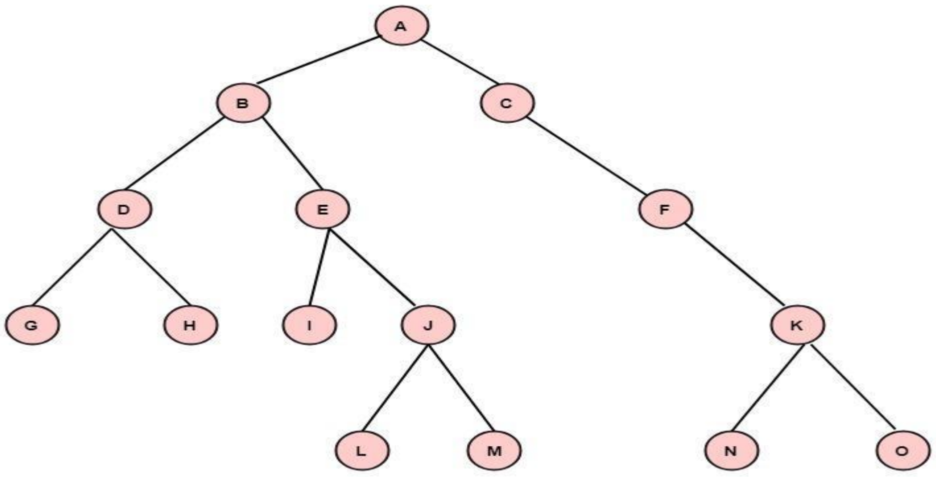
Leaf: A node with no children is called a leaf. The number of leaves in a binary tree can vary from one (minimum) to half the number of vertices (maximum) in a tree.

Descendant: A node is called descendant of another node if it is the child of the node or child of some other descendant of that node. All the nodes in the tree are descendants of the root.

Left Subtree: The subtree whose root is the left child of some node is called the left subtree of that node.

Example: For the tree as shown in fig:

- Which node is the root?
- Which nodes are leaves?
- Name the parent node of each node



Solution: (i) The node A is the root node.
(ii) The nodes G, H, I, L, M, N, O are leaves.
(iii)

Nodes	**Parent**
B, C	A
D, E	B
F	C
G, H	D
I, J	E
K	F
L, M	J
N, O	K

Right Subtree: The subtree whose root is the right child of some node is called the right subtree of that node.

Level of a Node: The level of a node is its distance from the root. The level of root is defined as zero. The level of all other nodes is one more than its parent node. The maximum number of nodes at any level N is 2^N.

Depth or Height of a tree: The depth or height of a tree is defined as the maximum number of nodes in a branch of a tree. This is more than the maximum level of the tree, i.e., the depth of root is one. The maximum number of nodes in a binary tree of depth d is 2^d-1, where $d \geq 1$.

External Nodes: The nodes which have no children are called external nodes or terminal nodes.

Internal Nodes: The nodes which have one or more than one children are called internal nodes or non-terminal nodes.

Binary Expression Trees:

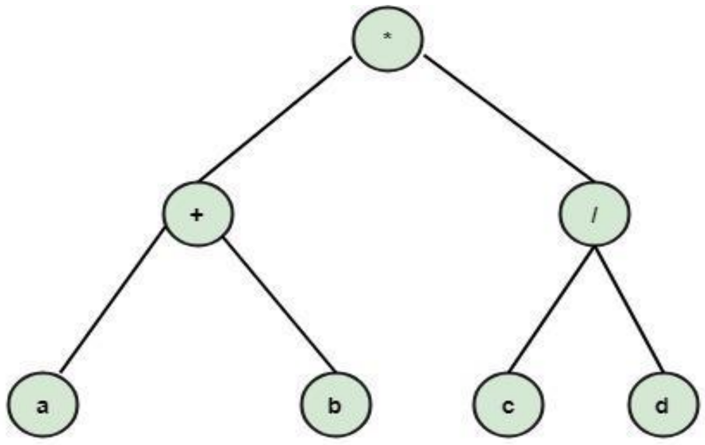
An algebraic expression can be conveniently expressed by its expression tree. An expression having binary operators can be decomposed into
    <left operand or expression> (operator) <right operand or expression>

Depending upon precedence of evaluation.

The expression tree is a binary tree whose root contains the operator and whose left subtree contains the left expression, and right subtree contains the right expression.

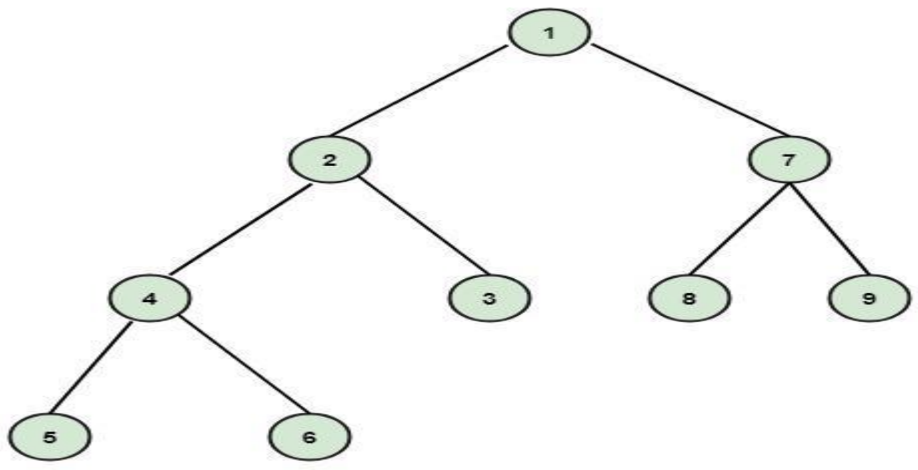
Example: Construct the binary expression tree for the expression (a+b)*(d/c)

Solution: The binary expression tree for the expression (a+b)*(d/c) is shown in fig:

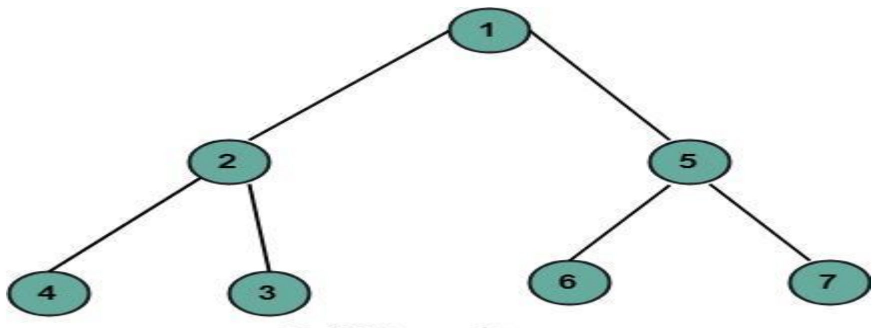


Complete Binary Tree: Complete binary tree is a binary tree if it is all levels, except possibly the last, have the maximum number of possible nodes as for left as possible. The depth of the complete binary tree having n nodes is $\log_2 n+1$.

Example: The tree shown in fig is a complete binary tree.

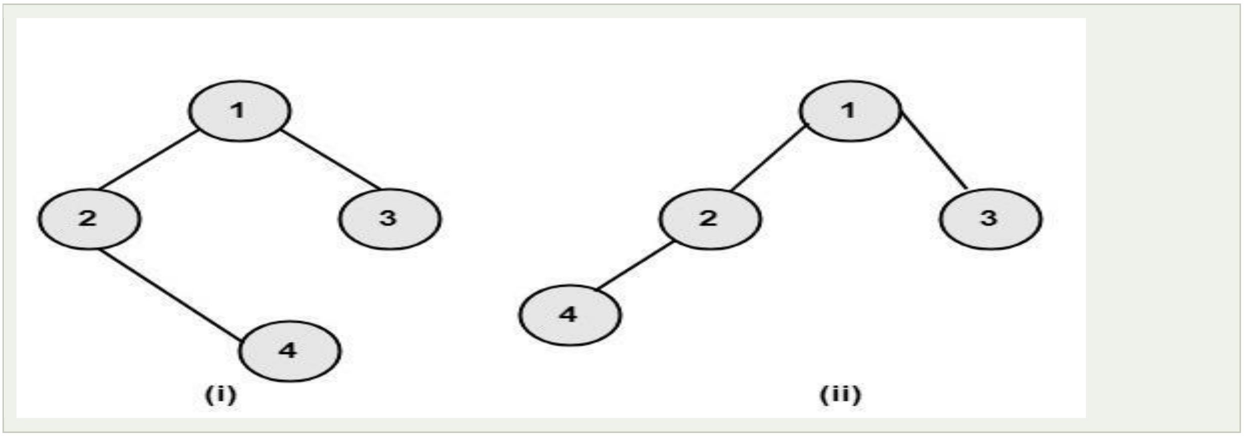


Full Binary Tree: Full binary tree is a binary tree in which all the leaves are on the same level and every non-leaf node has two children.


Full Binary Tree

Differentiate between General Tree and Binary Tree

General Tree	Binary Tree
1. There is no such tree having zero nodes or an empty general tree.	1. There may be an empty binary tree.
2. If some node has a child, then there is no such distinction.	2. If some node has a child, then it is distinguished as a left child or a right child.
3. The trees shown in fig are the same, when we consider them as general trees.	3. The trees shown in fig are distinct, when we consider them as binary trees, because in (i) 4 is the right child of 2 while in (ii) 4 is a left child of 2.



Traversing Binary Trees

Traversing means to visit all the nodes of the tree. There are three standard methods to traverse the binary trees. These are as follows:

1. Preorder Traversal
2. Postorder Traversal
3. Inorder Traversal

1. Preorder Traversal: The preorder traversal of a binary tree is a recursive process. The preorder traversal of a tree is

- Visit the root of the tree.
- Traverse the left subtree in preorder. (N L R)
- Traverse the right subtree in preorder.

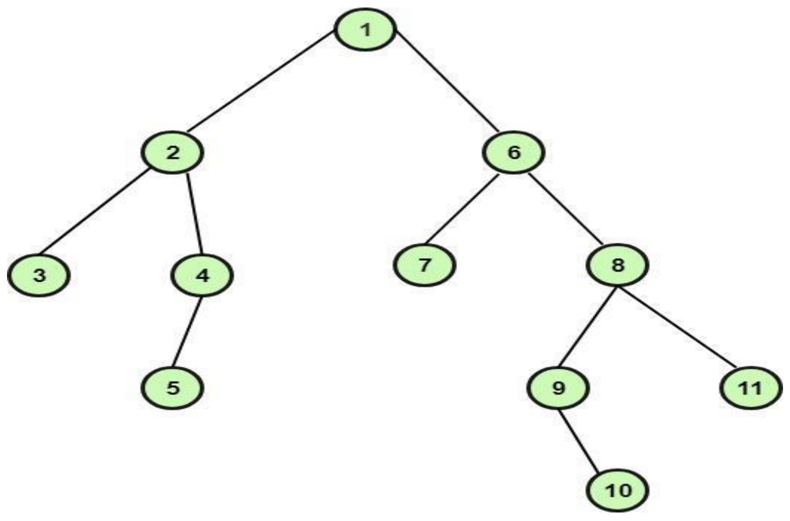
2. Postorder Traversal: The postorder traversal of a binary tree is a recursive process. The postorder traversal of a tree is

- Traverse the left subtree in postorder.
- Traverse the right subtree in postorder. (L R N)
- Visit the root of the tree.

3. Inorder Traversal: The inorder traversal of a binary tree is a recursive process. The inorder traversal of a tree is

- Traverse in inorder the left subtree.
- Visit the root of the tree. (L N R)
- Traverse in inorder the right subtree.

Example: Determine the preorder, postorder and inorder traversal of the binary tree as shown in fig:



Preorder Traversal: (N L R)
Postorder Traversal: (L R N)
Inorder Traversal: (L N R)

Solution: The preorder, postorder and inorder traversal of the tree is as follows:

Preorder	1	2	3	4	5	6	7	8	9	10	11
Postorder	3	5	4	2	7	10	9	11	8	6	1
Inorder	3	2	5	4	1	7	6	9	10	8	11

Algorithms:

(a)Algorithm to draw a Unique Binary Tree when Inorder and Preorder Traversal of the tree is Given:

1. We know that the root of the binary tree is the first node in its preorder. Draw the root of the tree.
2. To find the left child of the root node, first, use the inorder traversal to find the nodes in the left subtree of the binary tree. (All the nodes that are left to the root node in the inorder traversal are the nodes of the left subtree). After that, the left child of the root is obtained by selecting the first node in the preorder traversal of the left subtree. Draw the left child.
3. In the same way, use the inorder traversal to find the nodes in the right subtree of the binary tree. Then the right child is obtained by selecting the first node in the preorder traversal of the right subtree. Draw the right child.
4. Repeat the steps 2 and 3 with each new node until every node is not visited in the preorder. Finally, we obtain a unique tree.

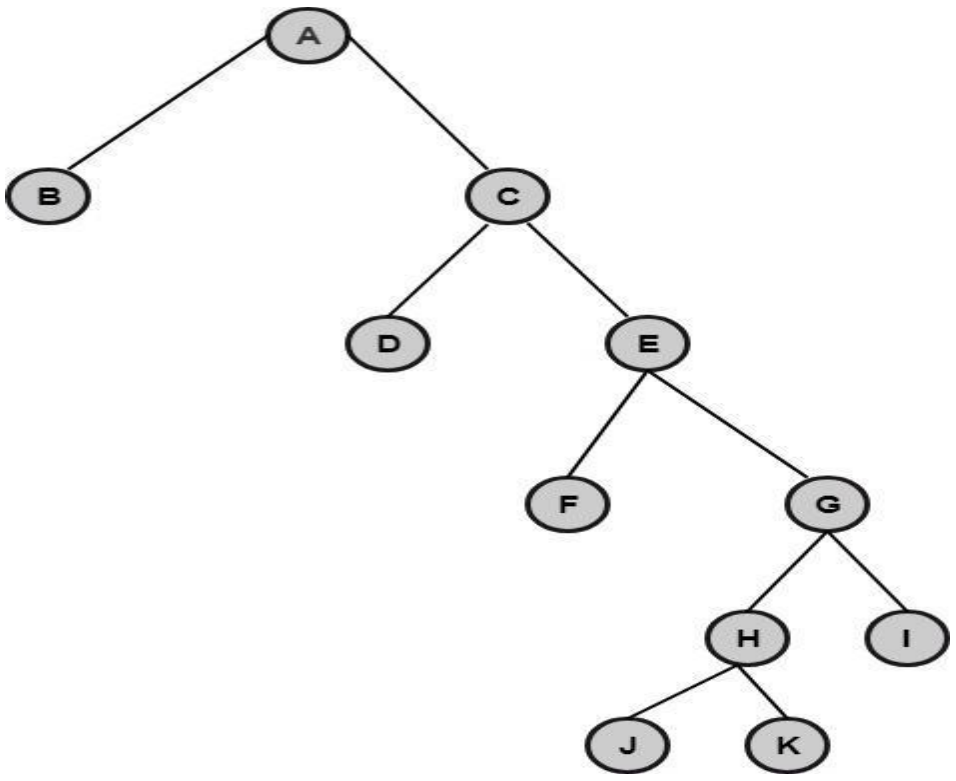
Example: Draw the unique binary tree when the inorder and preorder traversal is given as follows:

Preorder Traversal: (N L R)
Inorder Traversal: (L N R)

Inorder	B	A	D	C	F	E	J	H	K	G	I

Preorder	A	B	C	D	E	F	G	H	J	K	I

Solution: We know that the root of the binary tree is the first node in preorder traversal. Now, check A, in the inorder traversal, all the nodes that are of left A, are nodes of left subtree and all the nodes that are right of A, are nodes of right subtree. Read the next node in the preorder and check its position against the root node, if its left of the root node, then draw it as a left child, otherwise draw it a right child. Repeat the above process for each new node until all the nodes of the preorder traversal are read and finally we obtain the binary tree as shown in fig:



(b) Algorithm to draw a Unique Binary Tree when Inorder and Postorder Traversal of the tree is Given:

1. We know that the root of the binary tree is the last node in its postorder. Draw the root of the tree.
2. To find the right child of the root node, first, use the inorder traversal to find the nodes in the right subtree of the binary tree. (All the nodes that are right to the root node in the inorder traversal are the nodes of the right subtree). After that, the right child of the root is obtained by selecting the last node in the postorder traversal of the right subtree. Draw the right child.
3. In the same way, use the inorder traversal to find the nodes in the left subtree of the binary tree. Then the left child is obtained by selecting the last node in the postorder traversal of the left subtree. Draw the left child.
4. Repeat the steps 2 and 3 with each new node until every node is not visited in the postorder. After visiting the last node, we obtain a unique tree.

Example: Draw the unique binary tree for the given Inorder and Postorder traversal is given as follows:

Inorder	4	6	10	12	8	2	1	5	7	11	13	9	3

Postorder	12	10	8	6	4	2	13	11	9	7	5	3	1

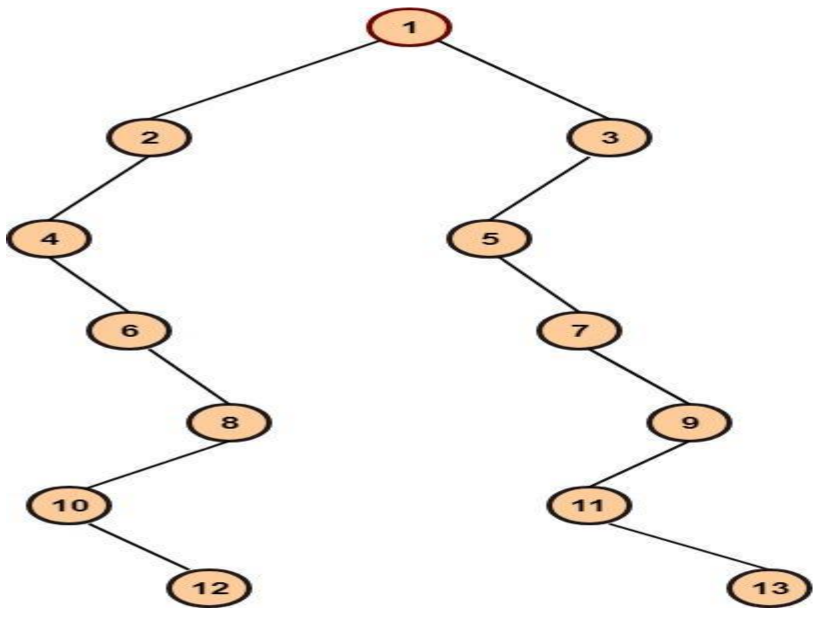
Postorder Traversal: (L R N)
Inorder Traversal: (L N R)

Solution: We know that the root of the binary tree is the last node in the postorder traversal. Hence, one in the root node.

Now, check the inorder traversal, we know that root is at the center, hence all the nodes that are left to the root node in inorder traversal are the nodes of left subtree and, all that are right to the root node are the nodes of the right subtree.

Now, visit the next node from back in postorder traversal and check its position in inorder traversal, if it is on the left of root then draw it as left child and if it is on the right, then draw it as the right child.

Repeat the above process for each new node, and we obtain the binary tree as shown in fig:



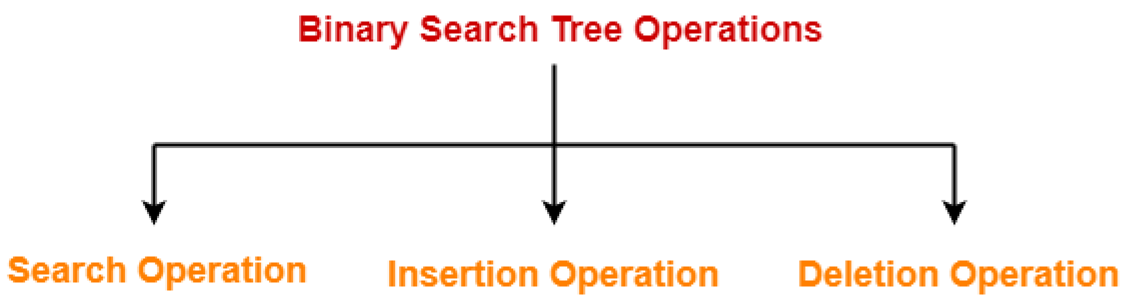
Binary Search Trees

Binary search trees have the property that the node to the left contains a smaller value than the node pointing to it and the node to the right contains a larger value than the node pointing to it.

It is not necessary that a node in a 'Binary Search Tree' point to the nodes whose value immediately precede and follow it.

Binary Search Tree Operations-

Commonly performed operations on binary search tree are-



1. Search Operation
2. Insertion Operation
3. Deletion Operation

1. Search Operation-

Search Operation is performed to search a particular element in the Binary Search Tree.

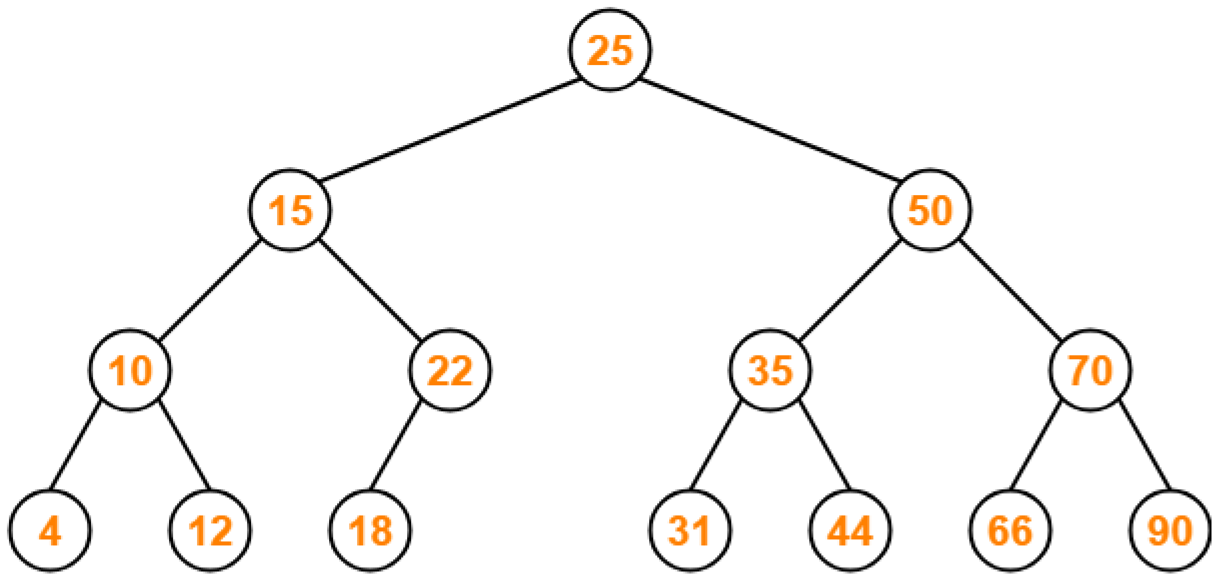
Rules-

For searching a given key in the BST,

- Compare the key with the value of root node.
- If the key is present at the root node, then return the root node.
- If the key is greater than the root node value, then recur for the root node's right subtree.
- If the key is smaller than the root node value, then recur for the root node's left subtree.

Example-

Consider key = 45 has to be searched in the given BST-



Binary Search Tree

- We start our search from the root node 25.
- As 45 > 25, so we search in 25’s right subtree.
- As 45 < 50, so we search in 50’s left subtree.
- As 45 > 35, so we search in 35’s right subtree.
- As 45 > 44, so we search in 44’s right subtree but 44 has no subtrees.
- So, we conclude that 45 is not present in the above BST.

2. Insertion Operation-

Insertion Operation is performed to insert an element in the Binary Search Tree.

Rules-

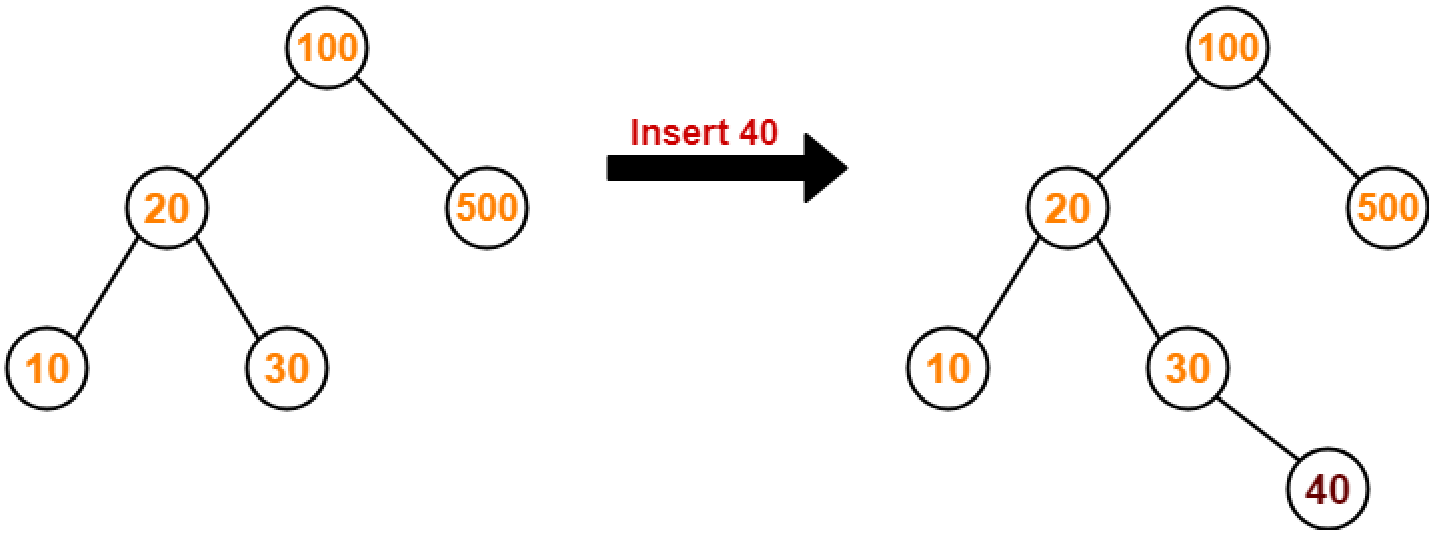
The insertion of a new key always takes place as the child of some leaf node.

For finding out the suitable leaf node,

- Search the key to be inserted from the root node till some leaf node is reached.
- Once a leaf node is reached, insert the key as child of that leaf node.

Example-

Consider the following example where key = 40 is inserted in the given BST-



- We start searching for value 40 from the root node 100.
- As 40 < 100, so we search in 100’s left subtree.
- As 40 > 20, so we search in 20’s right subtree.
- As 40 > 30, so we add 40 to 30’s right subtree.

3. Deletion Operation-

Deletion Operation is performed to delete a particular element from the Binary Search Tree.

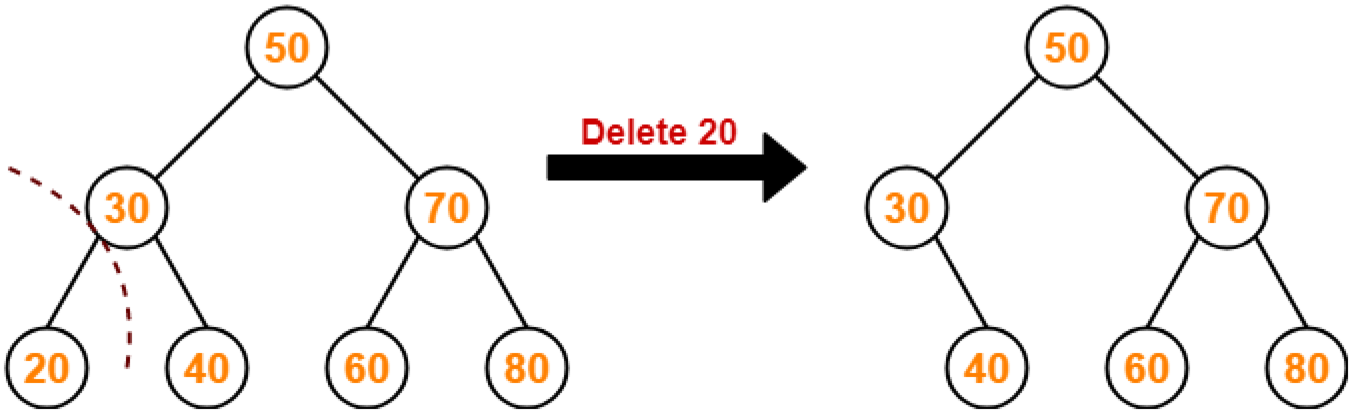
When it comes to deleting a node from the binary search tree, following three cases are possible-

Case-01: Deletion Of A Node Having No Child (Leaf Node)-

Just remove / disconnect the leaf node that is to deleted from the tree.

Example-

Consider the following example where node with value = 20 is deleted from the BST-

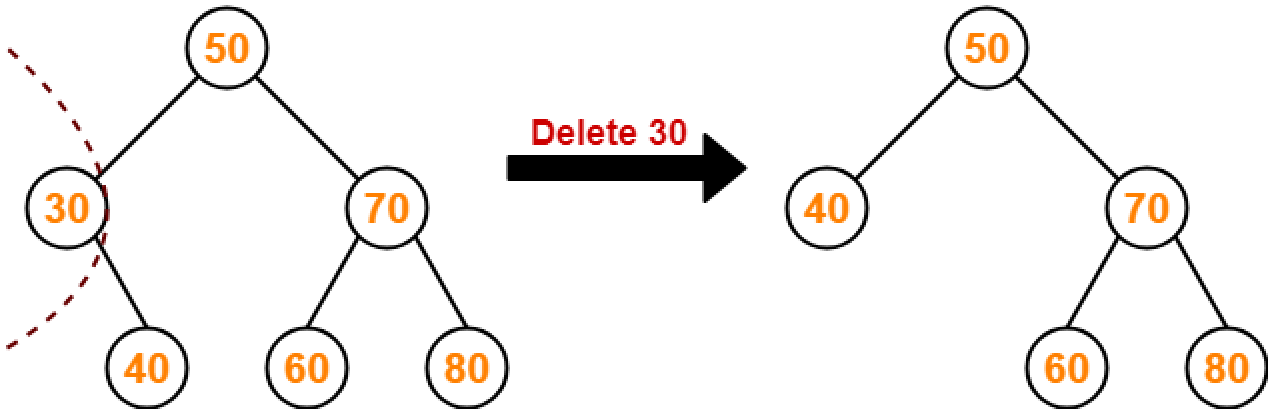


Case-02: Deletion Of A Node Having Only One Child-

Just make the child of the deleting node, the child of its grandparent.

Example-

Consider the following example where node with value = 30 is deleted from the BST-



Case-02: Deletion Of A Node Having Two Children-

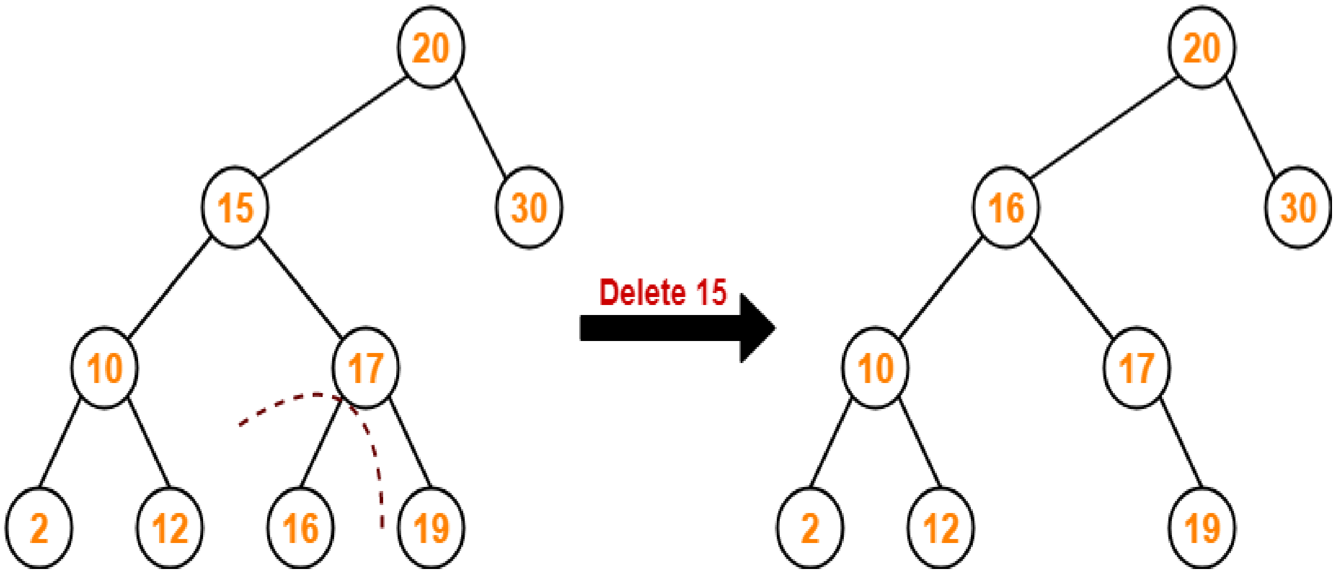
A node with two children may be deleted from the BST in the following two ways-

Method-01:

- Visit to the right subtree of the deleting node.
- Pluck the least value element called as inorder successor.
- Replace the deleting element with its inorder successor.

Example-

Consider the following example where node with value = 15 is deleted from the BST-

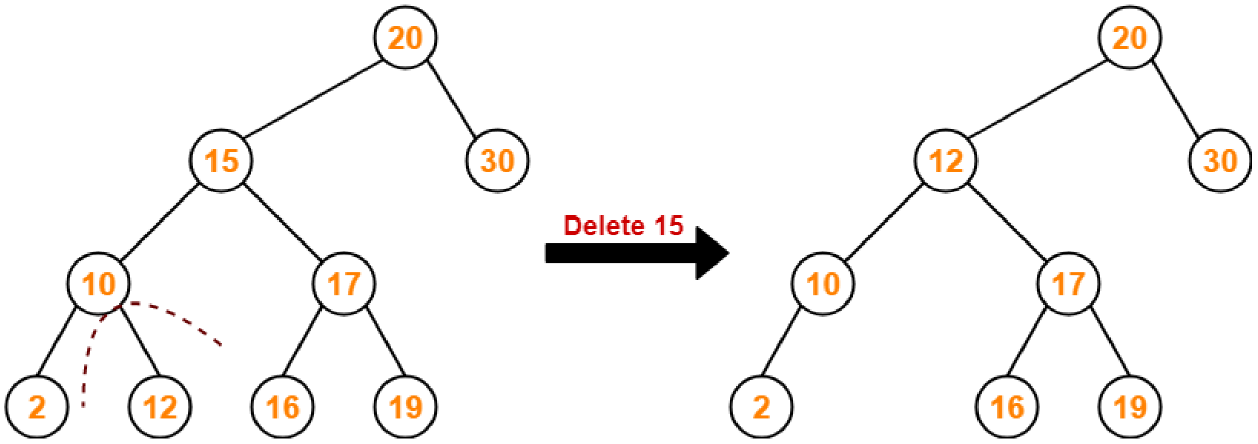


Method-02:

- Visit to the left subtree of the deleting node.
- Pluck the greatest value element called as inorder predecessor.
- Replace the deleting element with its inorder predecessor.

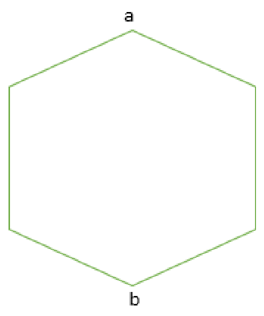
Example-

Consider the following example where node with value = 15 is deleted from the BST-

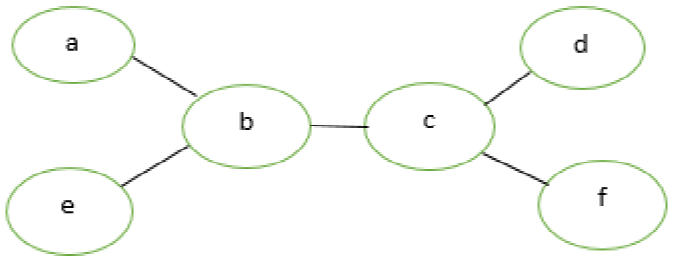


Some theorems related to trees are:

- **Theorem 1**: Prove that for a tree (T), there is one and only one path between every pair of vertices in a tree.
  Proof: Since tree (T) is a connected graph, there exist at least one path between every pair of vertices in a tree (T). Now, suppose between two vertices a and b of the tree (T) there exist two paths. The union of these two paths will contain a circuit and tree (T) cannot be a tree. Hence the above statement is proved.



Theorem 2: If in a graph G there is one and only one path between every pair of vertices than graph G is a tree.
Proof: There is the existence of a path between every pair of vertices so we assume that graph G is connected. A circuit in a graph implies that there is at least one pair of vertices a and b, such that there are two distinct paths between a and b. Since G has one and only one path between every pair of vertices. G cannot have any circuit. Hence graph G is a tree.



Theorem 3: Prove that a tree with n vertices has (n-1) edges.
Proof: Let n be the number of vertices in a tree (T).
If n=1, then the number of edges=0.
If n=2 then the number of edges=1.
If n=3 then the number of edges=2.
Hence, the statement (or result) is true for n=1, 2, 3.

Let the statement be true for n=m. Now we want to prove that it is true for n=m+1.

Let be the edge connecting vertices say Vi and Vj. Since G is a tree, then there exists only one path between vertices Vi and Vj. Hence if we delete edge e it will be disconnected graph into two components G1 and G2 say. These components have less than m+1 vertices and there is no circuit and hence each component G1 and G2 have m1 and m2 vertices.

Now, the total no. of edges = (m1-1) + (m2-1) +1

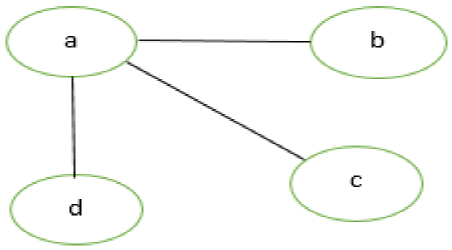
= (m1+m2)-1

= m+1-1

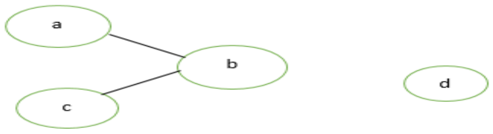
= m.

Hence for **n=m+1** vertices there are m edges in a tree (T). By the mathematical induction the graph exactly has n-1 edges.

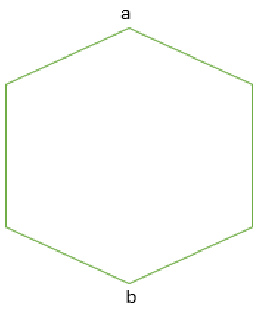
Theorem 4: Prove that any connected graph G with n vertices and (n-1) edges is a tree.
Proof: We know that the minimum number of edges required to make a graph of n vertices connected is (n-1) edges. We can observe that removal of one edge from the graph G will make it disconnected. Thus a connected graph of n vertices and (n-1) edges cannot have a circuit. Hence a graph G is a tree.



Theorem 5: Prove that a graph with n vertices, **(n-1)** edges and no circuit is a connected graph.
Proof: Let the graph G is disconnected then there exist at least two components G1 and G2 say. Each of the component is circuit-less as G is circuit-less. Now to make a graph G connected we need to add one edge e between the vertices Vi and Vj, where Vi is the vertex of G1 and Vj is the vertex of component G2.
Now the number of edges in **G = (n – 1)+1 =n.**

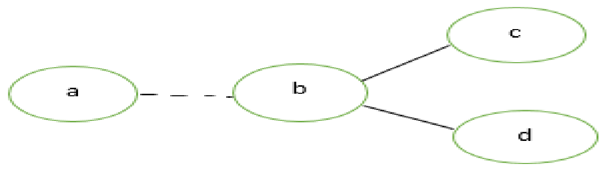


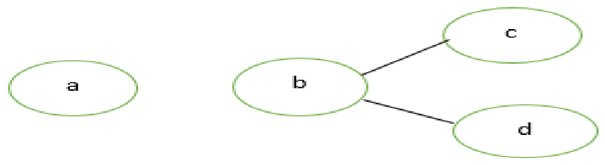
Now, G is connected graph and circuit-less with n vertices and n edges, which is impossible because the connected circuit-less graph is a tree and tree with n vertices has **(n-1)** edges. So the graph G with n vertices, **(n-1)** edges and without circuit is connected. Hence the given statement is proved.



Theorem 6: A graph G is a tree if and only if it is minimally connected.
Proof: Let the graph G is minimally connected, i.e; removal of one edge make it disconnected. Therefore, there is no circuit. Hence graph G is a tree.
Conversely, let the graph G is a tree i.e; there exists one and only one path between every pair of vertices and

we know that removal of one edge from the path makes the graph disconnected. Hence graph G is minimally connected.





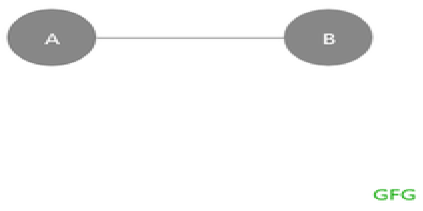
Pendant Vertices

Let **G** be a graph, A vertex **v** of **G** is called a **pendant vertex** if and only if **v** has **degree 1**. In other words, pendant vertices are the vertices that have **degree 1**, also called **pendant vertex**.

Note: *Degree = number of edges connected to a vertex*

In the case of trees, a **pendant vertex** is known as a **terminal node** or **leaf node**, or **leaf** since it has only **1** degree. Remember leaf nodes have only 1 degree so pendant vertex is called a leaf node in the case of trees.

Example: In the given diagram **A** and **B** are pendant vertices because each of them has degree **1**.



Theorem 7: Every tree with at-least two vertices has at-least two pendant vertices.

Proof: Let the number of vertices in a given tree T is n and n>=2. Therefore the number of edges in a tree T=n-1 using above theorems.

```
summation of (deg(Vi)) = 2*e

                       = 2*(n-1)

                       =2n-2
```

The degree sum is to be divided among n vertices. Since a tree T is a connected graph, it cannot have a vertex of degree zero. Each vertex contributes at-least one to the above sum. Thus there must be at least two vertices of degree 1. Hence every tree with at-least two vertices have at-least two pendant vertices.