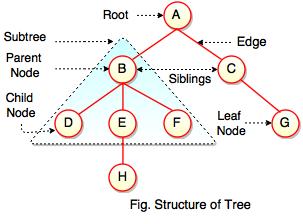
DATA STRUCTURES

UNIT 3: TREES

Definition:

* It is a non-linear data structure compared to arrays, linked lists, stack and queue.
* Tree is a hierarchical data structure which stores the information naturally in the form of hierarchy style.
* Tree is one of the most powerful and advanced data structures.
* It represents the nodes connected by edges.



The above figure represents structure of a tree. Tree has 2 sub trees.  
A is a parent of B and C. B is called a child of A and also parent of D, E, F.

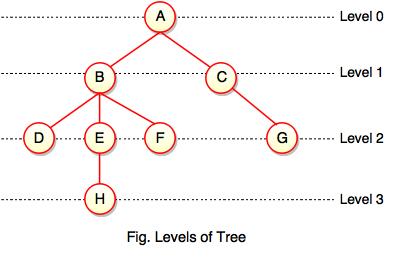
Tree is a collection of elements called Nodes, where each node can have arbitrary number of children.

|  |  |
| --- | --- |
| **Field** | **Description** |
| Root | Root is a special node in a tree. The entire tree is referenced through it. It does not have a parent. |
| Parent node | Parent node is an immediate predecessor of a node. |
| Child node | All immediate successors of a node are its children. |
| Siblings | Nodes with the same parent are called Siblings. |
| Path | Path is a number of successive edges from source node to destination node. |
| Height of a node | Height of a node represents the number of edges on the longest path between that node and a leaf. |
| Height of tree | Height of tree represents the height of its root node. |
| Depth of a node | Depth of a node represents the number of edges from the tree's root node to the node. |
| Degree of a node | Degree of a node represents a number of children of a node. |
| Edge | Edge is a connection between one node to another. It is a line between two nodes or a node and a leaf. |

In the above figure, D, F, H, G are**leaves**. B and C are **siblings**. Each node excluding a root is connected by a direct edge from exactly one other node  
parent →  children.

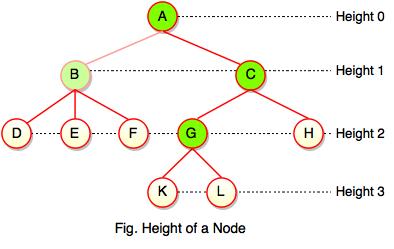
**Levels of a node**

Levels of a node represent the number of connections between the node and the root. It represents generation of a node. If the root node is at level 0, its next node is at level 1, its grand child is at level 2 and so on. Levels of a node can be shown as follows:



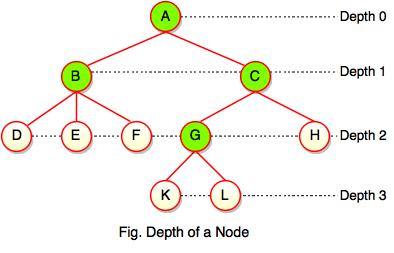
**Note:**  
  
- If node has no children, it is called **Leaves** or **External Nodes.**  
  
- Nodes which are not leaves, are called **Internal Nodes**. Internal nodes have at least one child.  
  
- A tree can be empty with no nodes or a tree consists of one node called the **Root**.

#### Height of a Node



As we studied, height of a node is a number of edges on the longest path between that node and a leaf. Each node has height.  
  
In the above figure, A, B, C, G can have height. Leaf cannot have height as there will be no path starting from a leaf. Node A's height is the number of edges of the path to K not to D. And its height is 3.  
  
**Note:**  
  
- Height of a node defines the longest path from the node to a leaf.  
  
- Path can only be downward.

**Depth of a Node**



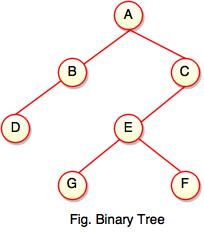
While talking about the height, it locates a node at bottom where for depth, it is located at top which is root level and therefore we call it depth of a node.  
  
In the above figure, Node G's depth is 2. In depth of a node, we just count how many edges between the targeting node & the root and ignoring the directions.  
  
**Note:** Depth of the root is 0.

**Advantages of Tree**

* Tree reflects structural relationships in the data.
* It is used to represent hierarchies.
* It provides an efficient insertion and searching operations.
* Trees are flexible. It allows to move subtrees around with minimum effort.

**Binary Tree:**

Binary tree is a special type of data structure. In binary tree, every node can have a maximum of 2 children, which are known as **Left child** and **Right Child**. It is a method of placing and locating the records in a database, especially when all the data is known to be in random access memory (RAM).  
  
**Definition:**  
  
"A tree in which every node can have maximum of two children is called as Binary Tree."

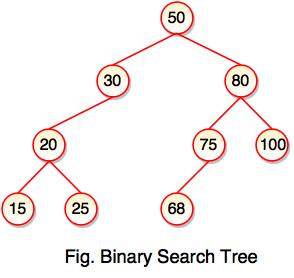


The above tree represents binary tree in which node A has two children B and C. Each children have one child namely D and E respectively.

## Binary Search Tree

* Binary search tree is a binary tree which has special property called BST.
* BST property is given as follows:

**For all nodes A and B,**  
  
I. If B belongs to the left subtree of A, the key at B is less than the key at A.  
  
II. If B belongs to the right subtree of A, the key at B is greater than the key at A.  
  
**Each node has following attributes:**  
  
I. Parent (P), left, right which are pointers to the parent (P), left child and right child respectively.  
  
II. Key defines a key which is stored at the node.  
  
**Definition:**  
  
"Binary Search Tree is a binary tree where each node contains only smaller values in its left subtree and only larger values in its right subtree."



* The above tree represents binary search tree (BST) where left subtree of every node contains smaller values and right subtree of every node contains larger value.
* Binary Search Tree (BST) is used to enhance the performance of binary tree.
* It focuses on the search operation in binary tree.

**Note:** Every binary search tree is a binary tree, but all the binary trees need not to be binary search trees.

**Binary Search Tree Operations**

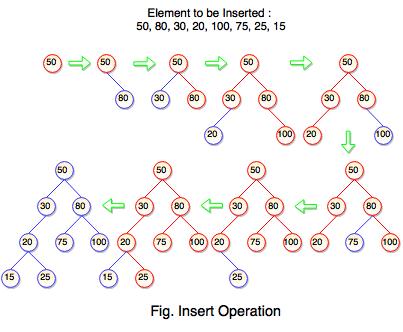
Following are the operations performed on binary search tree:

1. Insert Operation

* Insert operation is performed with O(log n) time complexity in a binary search tree.
* Insert operation starts from the root node. It is used whenever an element is to be inserted.

**The following algorithm shows the insert operation in binary search tree:**  
  
**Step 1:** Create a new node with a value and set its left and right to NULL.  
  
**Step 2:** Check whether the tree is empty or not.  
  
**Step 3:** If the tree is empty, set the root to a new node.  
  
**Step 4:** If the tree is not empty, check whether a value of new node is smaller or larger than the node (here it is a root node).

**Step 5:** If a new node is smaller than or equal to the node, move to its left child.  
  
**Step 6:** If a new node is larger than the node, move to its right child.  
  
**Step 7:** Repeat the process until we reach to a leaf node.



The above tree is constructed a binary search tree by inserting the above elements {50, 80, 30, 20, 100, 75, 25, 15}. The diagram represents how the sequence of numbers or elements are inserted into a binary search tree.

2. Search Operation

* Search operation is performed with O(log n) time complexity in a binary search tree.
* This operation starts from the root node. It is used whenever an element is to be searched.

**The following algorithm shows the search operation in binary search tree:**  
  
**Step 1:** Read the element from the user .  
  
**Step 2:** Compare this element with the value of root node in a tree.  
  
**Step 3:** If element and value are matching, display "Node is Found" and terminate the function.  
  
**Step 4:** If element and value are not matching, check whether an element is smaller or larger than a node value.  
  
**Step 5:** If an element is smaller, continue the search operation in left subtree.  
  
**Step 6:** If an element is larger, continue the search operation in right subtree.

**Step 7:** Repeat the same process until we found the exact element.  
  
**Step 8:** If an element with search value is found, display "Element is found" and terminate the function.  
  
**Step 9:** If we reach to a leaf node and the search value is not match to a leaf node, display "Element is not found" and terminate the function.

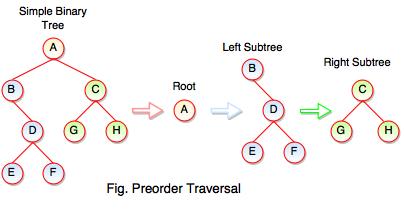
**Binary Tree Traversal**

Binary tree traversing is a process of accessing every node of the tree and exactly once. A tree is defined in a recursive manner. Binary tree traversal also defined recursively.

**There are three techniques of traversal:**  
  
**1.** Preorder Traversal  
**2.** Postorder Traversal  
**3.** Inorder Traversal

1. Preorder Traversal

**Algorithm for preorder traversal**  
  
**Step 1 :** Start from the Root.  
  
**Step 2 :** Then, go to the Left Subtree.  
  
**Step 3 :** Then, go to the Right Subtree.

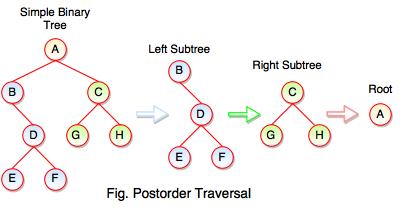


The above figure represents how preorder traversal actually works.

**Preorder Traversal : A B D E F C G H**

#### 2. Post order Traversal

**Algorithm for post order traversal**  
  
**Step 1 :** Start from the Left Subtree (Last Leaf).  
  
**Step 2 :** Then, go to the Right Subtree.  
  
**Step 3 :** Then, go to the Root.

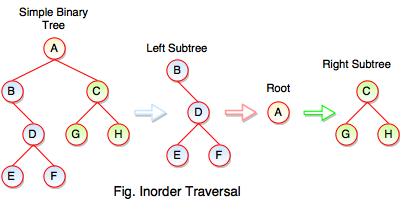


The above figure represents how postorder traversal actually works.

**Postorder Traversal : E F D B G H C A**

3. Inorder Traversal

**Algorithm for inorder traversal**  
  
**Step 1 :** Start from the Left Subtree.  
  
**Step 2 :** Then, visit the Root.  
  
**Step 3 :** Then, go to the Right Subtree.



The above figure represents how inorder traversal actually works.

**Inorder Traversal : B E D F A G C H**

#### Example: Program for Binary Tree

#include<stdio.h>  
#include<stdlib.h>  
  
struct node  
{  
      int data;  
      struct node \*rlink;  
      struct node \*llink;  
}\*tmp=NULL;  
  
typedef struct node NODE;  
NODE \*create();  
void preorder(NODE \*);  
void inorder(NODE \*);  
void postorder(NODE \*);  
void insert(NODE \*);  
  
int main()  
{  
     int n,i,ch;  
     do  
     {  
          printf("\n\n1.Create\n\n2.Insert\n\n3.Preorder\n\n4.Postorder\n\n5.Inorder\n\n6.Exit\n\n");  
          printf("\n\nEnter Your Choice : ");  
          scanf("%d",&ch);  
          switch(ch)  
          {  
               case 1:  
                    tmp=create();  
                    break;  
               case 2:  
                    insert(tmp);  
                    break;  
               case 3:  
                    printf("\n\nDisplay Tree in Preorder Traversal : ");  
                    preorder(tmp);  
                    break;  
               case 4:  
                    printf("\n\nDisplay Tree in Postorder Traversal : ");  
                    postorder(tmp);  
                    break;  
               case 5:  
                    printf("\n\nDisplay Tree in Inorder Traversal : ");  
                    inorder(tmp);  
                    break;  
               case 6:  
                    exit(0);  
                    default:  
                    printf("\n Inavild Choice..");  
          }  
     }  
     while(n!=5);  
}  
void insert(NODE \*root)  
{  
     NODE \*newnode;  
     if(root==NULL)  
     {  
          newnode=create();  
          root=newnode;  
     }  
     else  
     {  
          newnode=create();  
          while(1)  
          {  
               if(newnode->data<root->data)  
               {  
                    if(root->llink==NULL)  
                    {  
                         root->llink=newnode;  
                         break;  
                    }  
                    root=root->llink;  
               }  
               if(newnode->data>root->data)  
               {  
                    if(root->rlink==NULL)  
                    {  
                         root->rlink=newnode;  
                         break;  
                    }  
                    root=root->rlink;  
               }  
          }  
     }  
}  
NODE \*create()  
{  
     NODE \*newnode;  
     int n;  
     newnode=(NODE \*)malloc(sizeof(NODE));  
     printf("\n\nEnter the Data ");  
     scanf("%d",&n);  
     newnode->data=n;  
     newnode->llink=NULL;  
     newnode->rlink=NULL;  
     return(newnode);  
}  
void postorder(NODE \*tmp)  
{  
     if(tmp!=NULL)  
     {  
          postorder(tmp->llink);  
          postorder(tmp->rlink);  
          printf("%d->",tmp->data);  
     }  
}  
void inorder(NODE \*tmp)  
{  
     if(tmp!=NULL)  
     {  
          inorder(tmp->llink);  
          printf("%d->",tmp->data);  
          inorder(tmp->rlink);  
     }  
}  
void preorder(NODE \*tmp)  
{  
     if(tmp!=NULL)  
     {  
          printf("%d->",tmp->data);  
          preorder(tmp->llink);  
          preorder(tmp->rlink);  
     }  
}