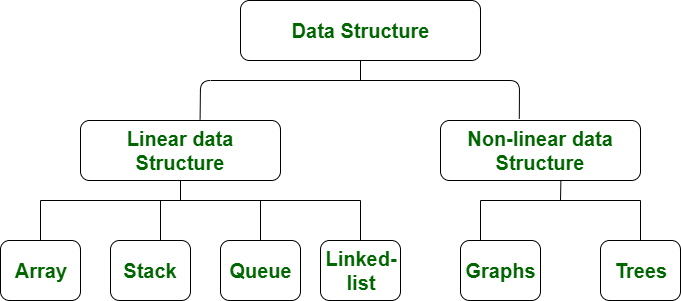
TREE

=======================================================

A tree data structure consists of various nodes linked together. The structure of a tree is hierarchical that forms a relationship like that of the parent and a child.

The structure of the tree is formed in a way that there is one connection for every parent-child node relationship. Only one path should exist between the root to a node in the tree. Various types of trees are present based on their structures like AVL tree, binary tree, binary search tree, etc.

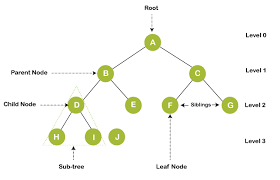
========================================================



=========================================================

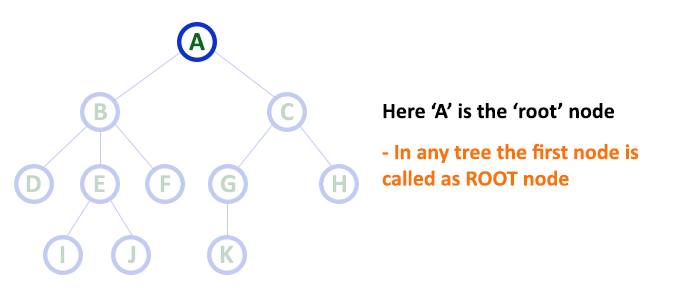
| S.NO | Linear Data Structure | Non-linear DataStructure |
| --- | --- | --- |
| 1. | In a linear data structure, data elements are arranged in a linear order where each and every elements are attached to its previous and next adjacent. | In a non-linear data structure, data elements are attached in hierarchically manner. |
| 2. | In linear data structure, single level is involved. | Whereas in non-linear data structure, multiple levels are involved. |
| 3. | Its implementation is easy in comparison to non-linear data structure. | While its implementation is complex in comparison to linear data structure. |
| 4. | In linear data structure, data elements can be traversed in a single run only. | While in non-linear data structure, data elements can’t be traversed in a single run only. |
| 5. | In a linear data structure, memory is not utilized in an efficient way. | While in a non-linear data structure, memory is utilized in an efficient way. |
| 6. | Its examples are: array, stack, queue, linked list, etc. | While its examples are: trees and graphs. |
| 7. | Applications of linear data structures are mainly in application software development. | Applications of non-linear data structures are in Artificial Intelligence and image processing. |

============================================================



**Tree terminology...**

* **1.Root:**
  + 1)The first node is called as Root Node.
  + 2)Every tree must have root node, there must be only one root node.
  + 3)Root node doesn't have any parent.

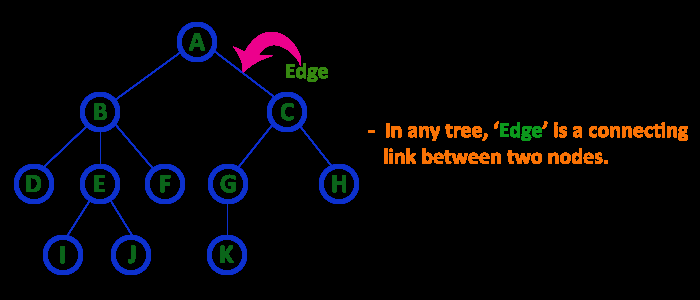


==============================================================================

* **2.Edge:**

1)In a tree data structure, the connecting link between any two nodes is called as EDGE.

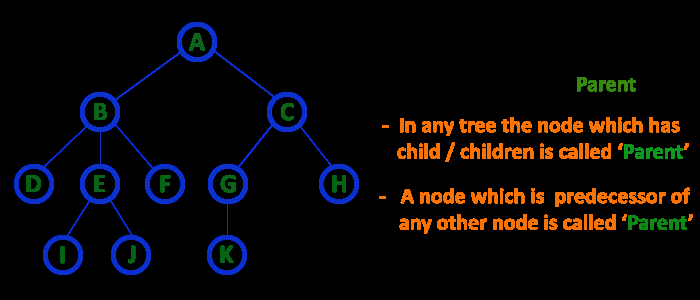
2)In a tree with 'N' number of nodes there will be a maximum of 'N-1' number of edges.



**3.Parent**

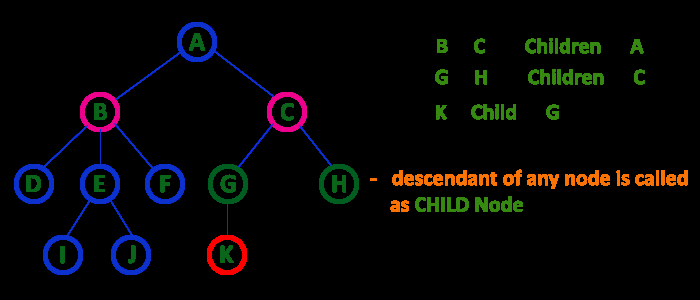
* 1)In a tree data structure, the node which is predecessor of any node is called as PARENT NODE.

2)In simple words, the node which has branch from it to any other node is called as parent node. Parent node can also be defined as "The node which has child / children".



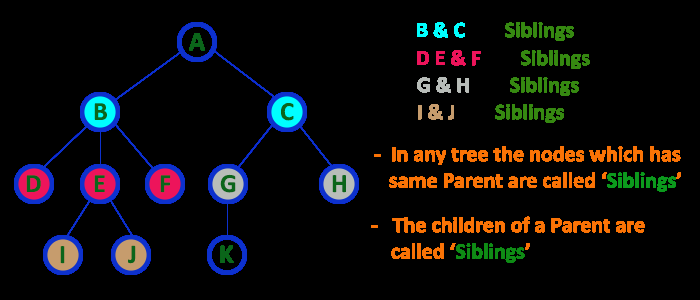
**4.Parent**

* The node which has a link from its parent node is called as child node.
* In a tree, any parent node can have any number of child nodes.
* In a tree, all the nodes except root are child nodes.



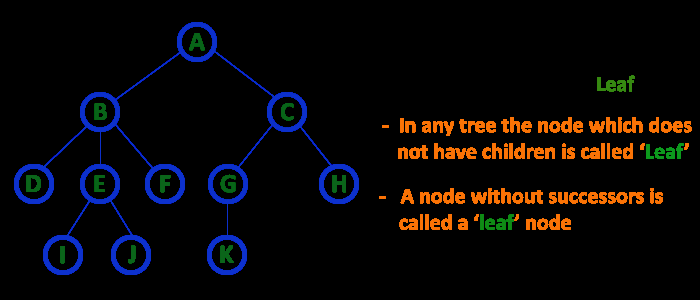
**5.Siblings**

The nodes with same parent are called as Sibling nodes



==============================================================================

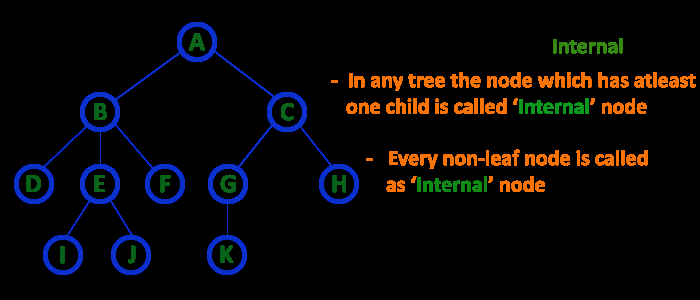
**6.Leaf Node**



==============================================================================

**7.Internal Nodes**

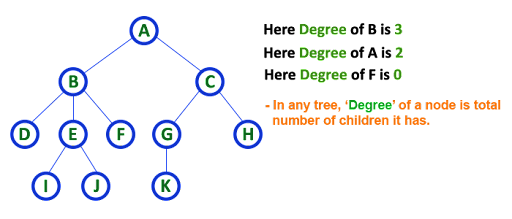
* An internal node is a node with atleast one child.
* Nodes other than leaf nodes are called as Internal Nodes.
* The root node is also said to be Internal Node if the tree has more than one node. Internal nodes are also called as 'Non-Terminal' nodes.



==============================================================================

**8.Degree**

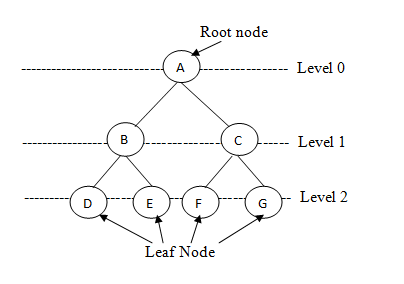
* In a tree data structure, the total number of children of a node is called as DEGREE of that Node.



==============================================================================

**9.Level**

* In a tree data structure, the root node is said to be at Level 0 and the children of root node are at Level 1 and the children of the nodes which are at Level 1 will be at Level 2 and so on...
* In a tree each step from top to bottom is called as a Level and the Level count starts with '0' and incremented by one at each level (Step).

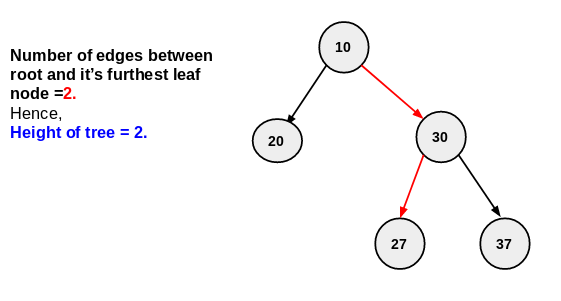


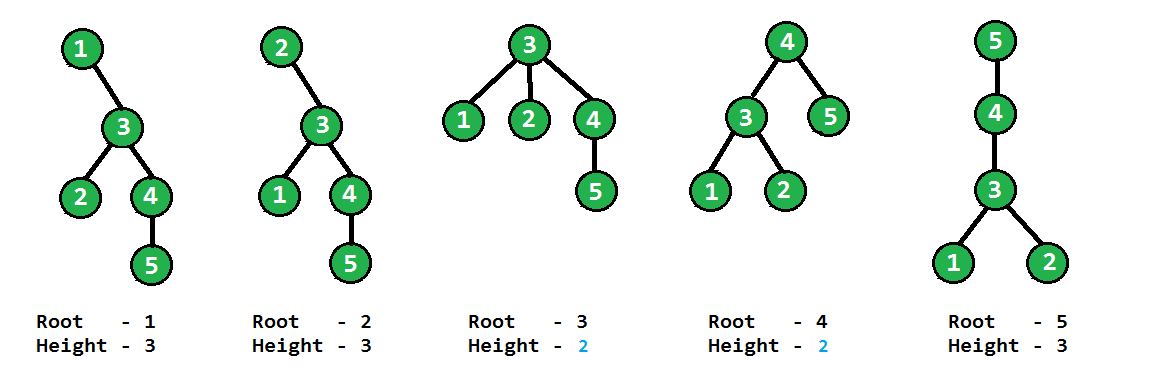
==============================================================================

**10.Height**

 The height of a binary tree is **the maximum distance from the root node to the leaf node**. .

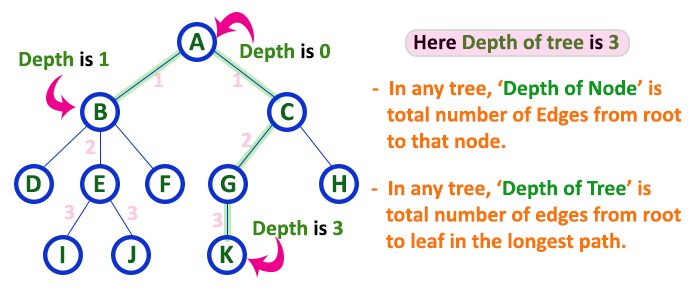
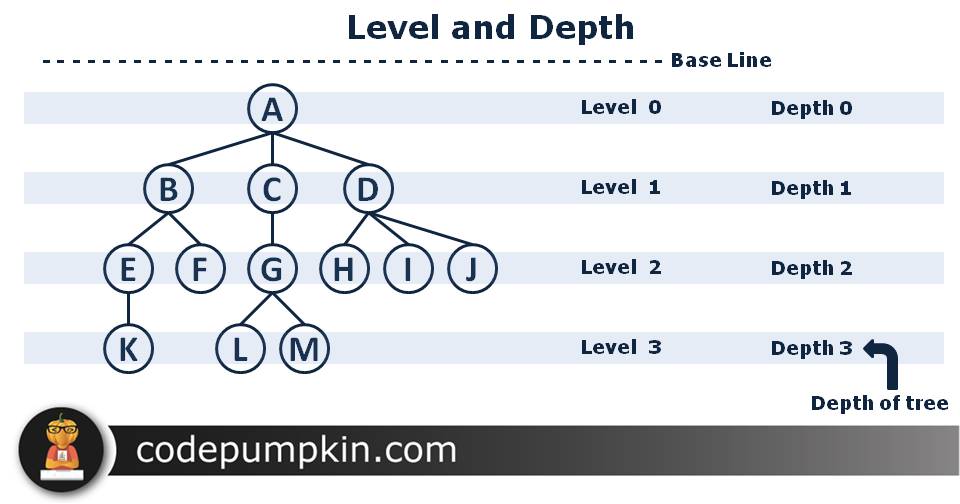
In a tree, height of the root node is said to be height of the tree.



==============================================================================

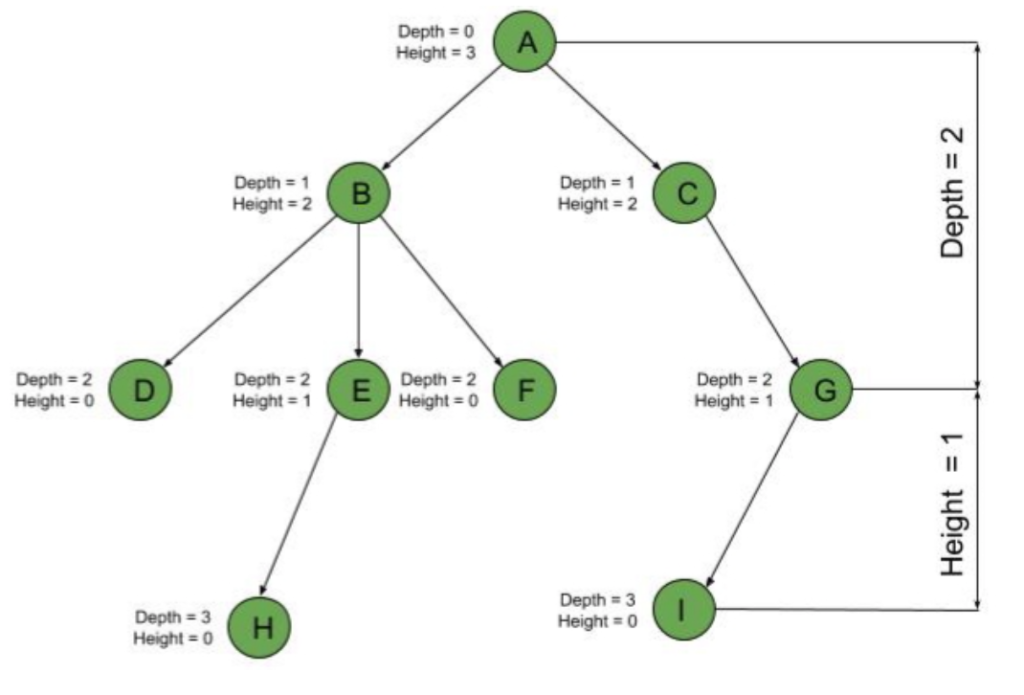
**11.Depth**

In a tree data structure, the total number of egdes from root node to a particular node is called as DEPTH of that Node



**Difference Between Height & Depth**

For each node in a tree, we can define two features: height and depth. A node’s **height is the number of edges to its most distant leaf node**. On the other hand, a **node’s depth is the number of edges back up to the root**. So, the **root always has a depth of** while **leaf nodes always have a height of**. And if we look at the tree as a whole, its depth and height are both the root height.

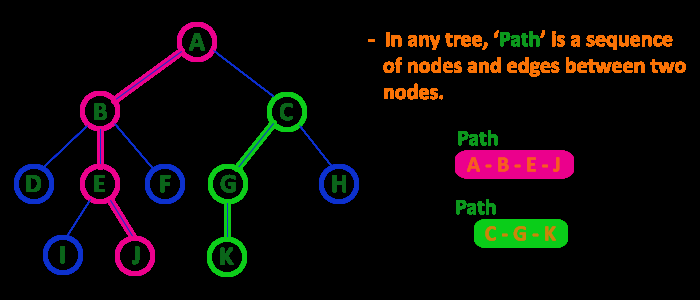


==============================================================================

**12.Path**

In a tree data structure, the sequence of Nodes and Edges from one node to another node is called as PATH between that two Nodes.

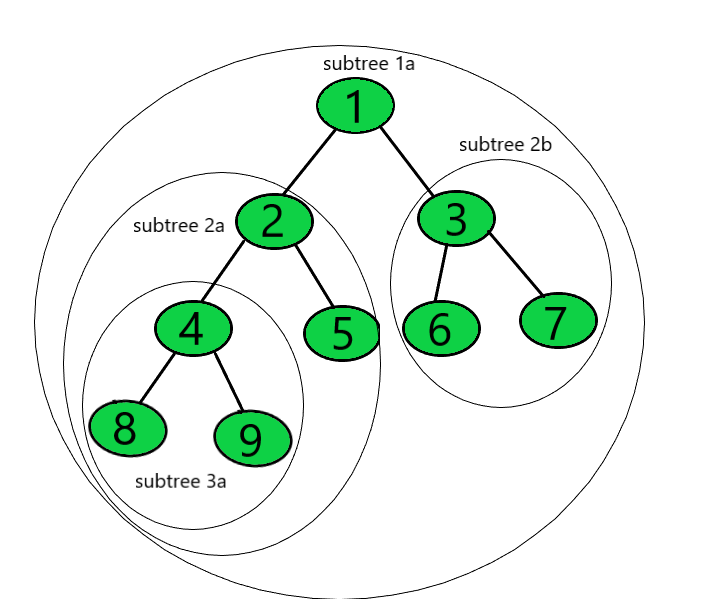
Length of a Path is total number of nodes in that path. In below example the path A - B - E - J has length 4



==============================================================================

**13.Sub Tree**

Every child node will form a subtree on its parent node.



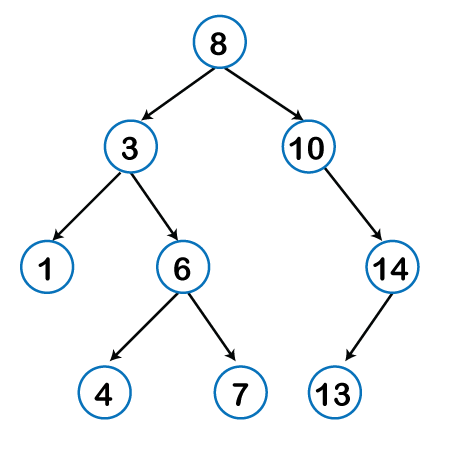
==================================================================================

**Types of Trees in Data Structure**

**A] Bases on number of child nodes**

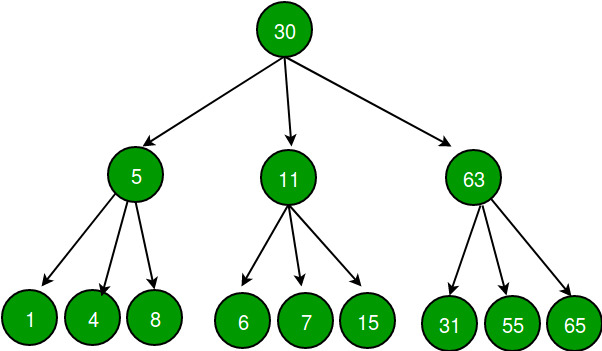
#### 1. Binary Tree

The binary tree is the kind of tree in which most two children can be found for each parent. The kids are known as the left kid and right kid. This is more popular than most other trees



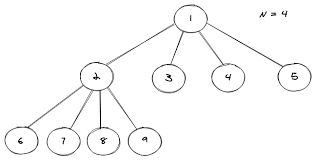
#### 2. Ternary Tree

Each node will have max of 3 Children



#### 3. N-ary Tree

Tree in which each node will have max of n children

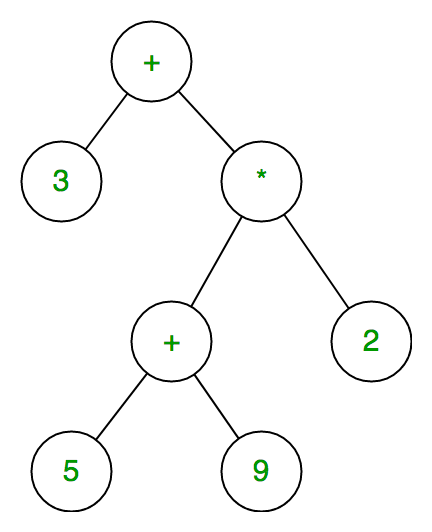


**B] Bases on How data is stored/Organized**

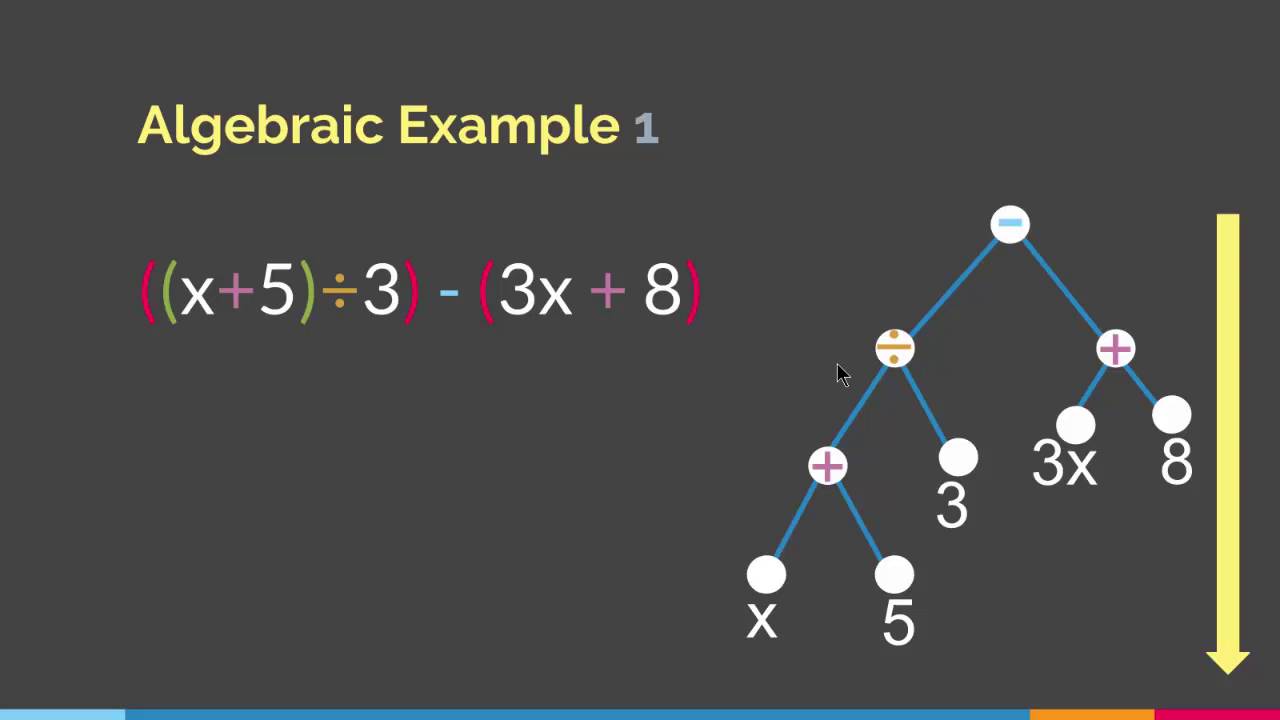
#### 1. Expression Tree

The expression tree is a binary tree in which each internal node corresponds to the operator and each leaf node corresponds to the operand so for example expression tree for 3 + ((5+9)\*2) would be:

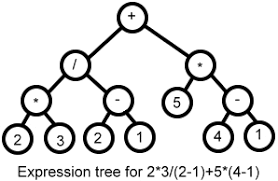
**Example 1)**



**Example 2**



**Example 3**

****

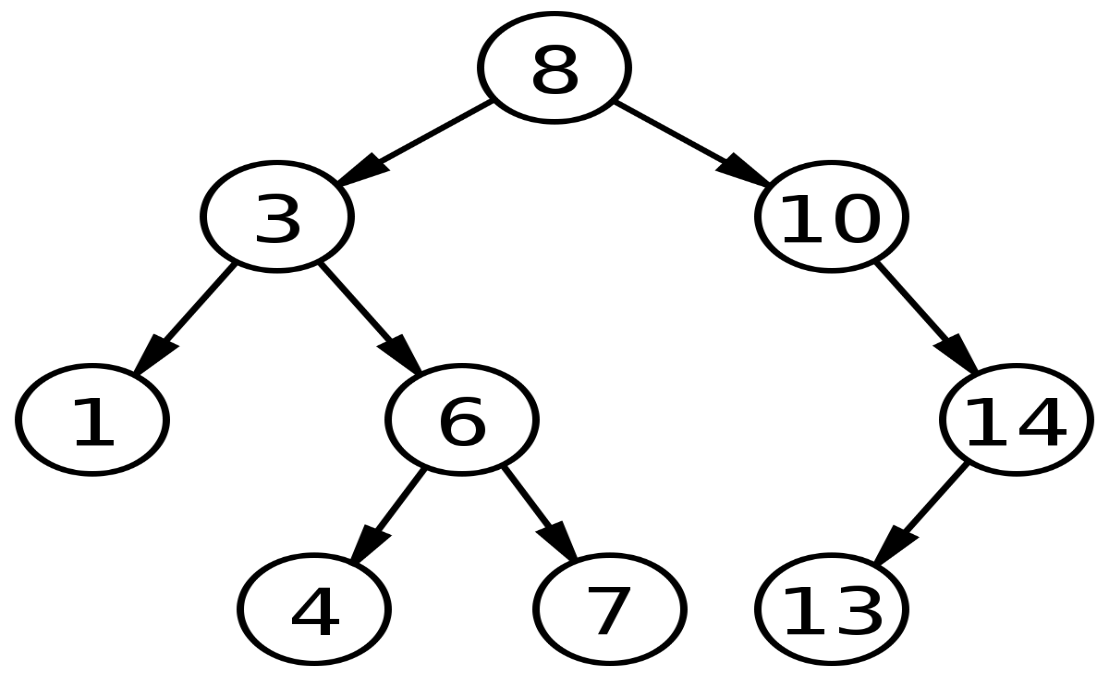
==================================

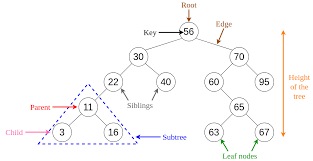
#### 2. Binary Search Tree

\*Each node satisfying following property\*

Data of nodes in left subtree < Node data(root) < Data of nodes in right subtree

Every node in the left subtree must contain a value less than the value of the root node, and the value of each node in the right subtree must be bigger than the value of the root node.





================================================================================

#### 3. Skewed Tree

-Left Skewed (Root have only left side nodes)

-Right Skewed(Root have only right side nodes)

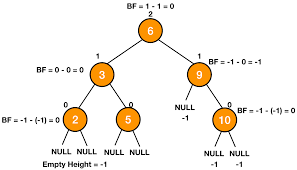


===============================================================================

#### 4. AVL Tree:

Height Balnace Tree

It is one of the types of the binary tree, or we can say that it is a variant of the binary search tree. AVL tree satisfies the property of the **binary tree** as well as of the **binary search tree**. It is a self-balancing binary search tree that was invented by **Adelson Velsky Lindas**



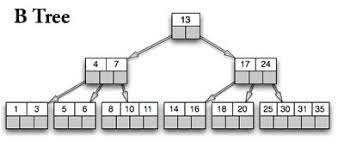
**Applications Of AVL Trees**

AVL trees are mostly used for **in-memory sorts of sets and dictionaries**. AVL trees are also used extensively in database applications in which insertions and deletions are fewer but there are frequent lookups for data required.

=======================================================

#### 5. B-Tree

* B-Tree is the tree data structures used to implement indexing in databases.



B tree is **used to index the data** and provides fast access to the actual data stored on the disks since, the access to value stored in a large database that is stored on a disk is a very time consuming process.

=================================================

**BFS & DFS For Binary Tree**

**What are BFS and DFS for Binary Tree?**

A Tree is typically traversed in two ways:

* [Breadth First Traversal (Or Level Order Traversal)](https://www.geeksforgeeks.org/level-order-tree-traversal/)
* [Depth First Traversals](https://www.geeksforgeeks.org/618/)
  + Inorder Traversal (Left-Root-Right)
  + Preorder Traversal (Root-Left-Right)
  + Postorder Traversal (Left-Right-Root)



BFS and DFSs of above Tree

Breadth First Traversal : 1 2 3 4 5

Depth First Traversals:

Preorder Traversal : 1 2 4 5 3

Inorder Traversal : 4 2 5 1 3

Postorder Traversal : 4 5 2 3 1

========================================================

Algorithm Inorder(tree)

1. Traverse the left subtree, i.e., call Inorder(left-subtree)

2. Visit the root.

3. Traverse the right subtree, i.e., call Inorder(right-subtree)

================================

Algorithm Preorder(tree)

1. Visit the root.

2. Traverse the left subtree, i.e., call Preorder(left-subtree)

3. Traverse the right subtree, i.e., call Preorder(right-subtree)

===============================

Algorithm Postorder(tree)

1. Traverse the left subtree, i.e., call Postorder(left-subtree)

2. Traverse the right subtree, i.e., call Postorder(right-subtree)

3. Visit the root.

==================================

|  |
| --- |
| // Java program for different tree traversals    /\* Class containing left and right child of current     node and key value\*/  class Node {      int key;      Node left, right;        public Node(int item)      {          key = item;          left = right = null;      }  }    class BinaryTree {      // Root of Binary Tree      Node root;        BinaryTree() { root = null; }        /\* Given a binary tree, print its nodes according to the        "bottom-up" postorder traversal. \*/      void printPostorder(Node node)      {          if (node == null)              return;            // first recur on left subtree          printPostorder(node.left);            // then recur on right subtree          printPostorder(node.right);            // now deal with the node          System.out.print(node.key + " ");      }        /\* Given a binary tree, print its nodes in inorder\*/      void printInorder(Node node)      {          if (node == null)              return;            /\* first recur on left child \*/          printInorder(node.left);            /\* then print the data of node \*/          System.out.print(node.key + " ");            /\* now recur on right child \*/          printInorder(node.right);      }        /\* Given a binary tree, print its nodes in preorder\*/      void printPreorder(Node node)      {          if (node == null)              return;            /\* first print data of node \*/          System.out.print(node.key + " ");            /\* then recur on left sutree \*/          printPreorder(node.left);            /\* now recur on right subtree \*/          printPreorder(node.right);      }        // Wrappers over above recursive functions      void printPostorder() { printPostorder(root); }      void printInorder() { printInorder(root); }      void printPreorder() { printPreorder(root); }        // Driver method      public static void main(String[] args)      {          BinaryTree tree = new BinaryTree();          tree.root = new Node(1);          tree.root.left = new Node(2);          tree.root.right = new Node(3);          tree.root.left.left = new Node(4);          tree.root.left.right = new Node(5);            System.out.println(              "Preorder traversal of binary tree is ");          tree.printPreorder();            System.out.println(              "\nInorder traversal of binary tree is ");          tree.printInorder();            System.out.println(              "\nPostorder traversal of binary tree is ");          tree.printPostorder();      }  } |

**Output:**

Preorder traversal of binary tree is

1 2 4 5 3

Inorder traversal of binary tree is

4 2 5 1 3

Postorder traversal of binary tree is

4 5 2 3 1

==========================================================

**Binary Search Tree :-**

**\*Properties**

1)All nodes of left sub tree are lesser

2)All nodes of right sub btree are greater

3)Left and Right Subtree's are also BST.

4)there are no duplicate nodes

5)InOrder Traversal of BST gives a ascending sorted array.

======================================

**Binary Search Tree** is a node-based binary tree data structure which has the following properties:

* The left subtree of a node contains only nodes with keys lesser than the node’s key.
* The right subtree of a node contains only nodes with keys greater than the node’s key.
* The left and right subtree each must also be a binary search tree.

==========================================

Lightbox

**AVL Tree :-**

**AVL TREE:- Efficient Search**

**It uses balance factor to determine if tree is unbalanced.**

**If unbalanced uses rotation to balance tree.**

For a node

***Balance\_Factor*** = height(left-subtree) − height(right-subtree)

**Which AVL Tree is unbalanced?**

IF BF > 1 then tree is unbalanced.

**Advantage:**

AVL tree requires more space(Extra memory required)

than Red-BLack tree Because it stores int(height)

**Disadvantage:**

AVL tree is faster than Red-black tree in case of searching

====================================

If the difference in the height of left and right sub-trees is more than 1, the tree is balanced using some rotation techniques.

## **AVL Rotations**

To balance itself, an AVL tree may perform the following four kinds of rotations −

* Left rotation
* Right rotation
* Left-Right rotation
* Right-Left rotation

The first two rotations are single rotations and the next two rotations are double rotations. To have an unbalanced tree, we at least need a tree of height 2. With this simple tree, let's understand them one by one.

### Left Rotation

If a tree becomes unbalanced, when a node is inserted into the right subtree of the right subtree, then we perform a single left rotation −



in our example, node **A** has become unbalanced as a node is inserted in the right subtree of A's right subtree. We perform the left rotation by making **A** the left-subtree of B.

### Right Rotation

AVL tree may become unbalanced, if a node is inserted in the left subtree of the left subtree. The tree then needs a right rotation.



As depicted, the unbalanced node becomes the right child of its left child by performing a right rotation.

Link to refer:

<https://www.tutorialspoint.com/data_structures_algorithms/avl_tree_algorithm.htm>

==============================================