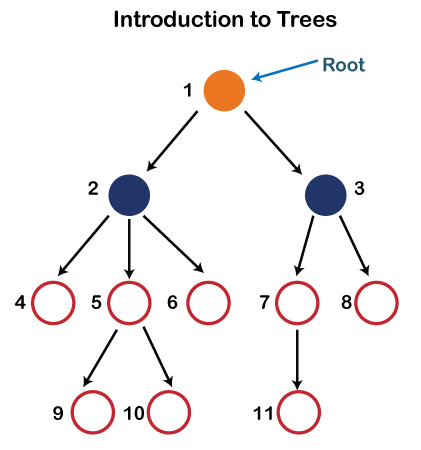
**Tree**

* A tree data structure is defined as a collection of objects or entities known as nodes that are linked together to represent or simulate hierarchy.
* A tree data structure is a non-linear data structure because it does not store in a sequential manner. It is a hierarchical structure as elements in a Tree are arranged in multiple levels.
* In the Tree data structure, the topmost node is known as a root node. Each node contains some data, and data can be of any type.
* Each node contains some data and the link or reference of other nodes that can be called children.

**Some basic terms used in Tree data structure.**

Let's consider the tree structure, which is shown below:

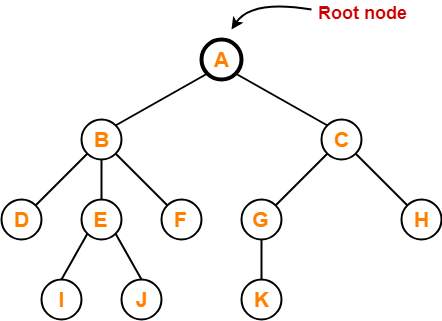


In the above structure, each node is labeled with some number. Each arrow shown in the above figure is known as a ***link*** between the two nodes.

## **1. Root-**

* The first node from where the tree originates is called as a **root node**.
* In any tree, there must be only one root node.
* We can never have multiple root nodes in a tree data structure.

### ****Example-****

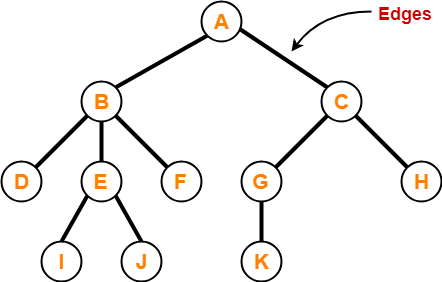


Here, node A is the only root node.

## **2. Edge-**

* The connecting link between any two nodes is called as an **edge**.
* In a tree with n number of nodes, there are exactly (n-1) number of edges.

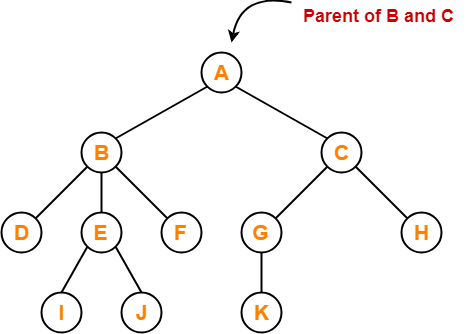
### ****Example-****



## **3. Parent-**

* The node which has a branch from it to any other node is called as a **parent node**.
* In other words, the node which has one or more children is called as a parent node.
* In a tree, a parent node can have any number of child nodes.

### ****Example-****



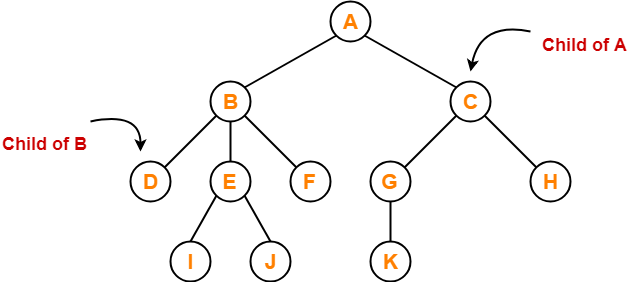
Here,

* Node A is the parent of nodes B and C
* Node B is the parent of nodes D, E and F
* Node C is the parent of nodes G and H
* Node E is the parent of nodes I and J
* Node G is the parent of node K

## **4. Child-**

* The node which is a descendant of some node is called as a **child node**.
* All the nodes except root node are child nodes.

### ****Example-****



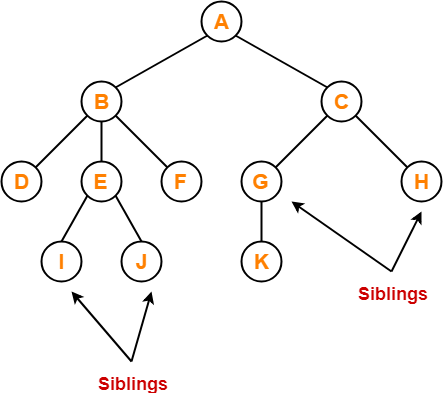
Here,

* Nodes B and C are the children of node A
* Nodes D, E and F are the children of node B
* Nodes G and H are the children of node C
* Nodes I and J are the children of node E
* Node K is the child of node G

## **5. Siblings-**

* Nodes which belong to the same parent are called as **siblings**.
* In other words, nodes with the same parent are sibling nodes.

### ****Example-****



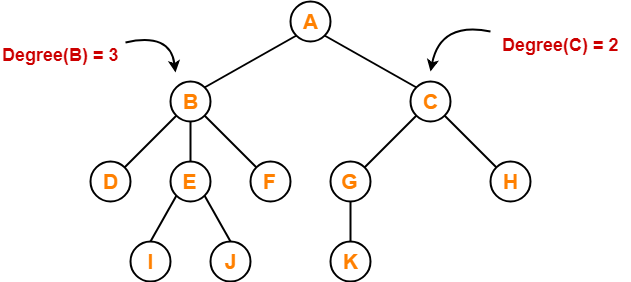
Here,

* Nodes B and C are siblings
* Nodes D, E and F are siblings
* Nodes G and H are siblings
* Nodes I and J are siblings

## **6. Degree-**

* **Degree of a node** is the total number of children of that node.
* **Degree of a tree** is the highest degree of a node among all the nodes in the tree.

### ****Example-****



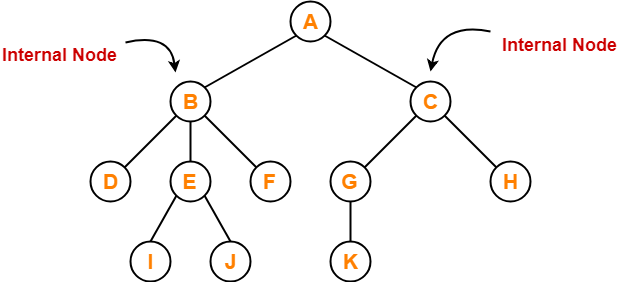
Here,

* Degree of node A = 2
* Degree of node B = 3
* Degree of node C = 2
* Degree of node D = 0
* Degree of node E = 2
* Degree of node F = 0
* Degree of node G = 1
* Degree of node H = 0
* Degree of node I = 0
* Degree of node J = 0
* Degree of node K = 0

## **7. Internal Node-**

* The node which has at least one child is called as an **internal node**.
* Internal nodes are also called as **non-terminal nodes**.
* Every non-leaf node is an internal node.

### ****Example-****

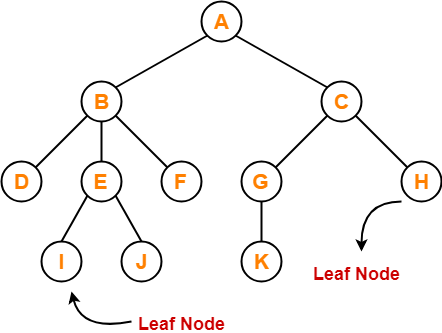


Here, nodes A, B, C, E and G are internal nodes.

## **8. Leaf Node-**

* The node which does not have any child is called as a **leaf node**.
* Leaf nodes are also called as **external nodes** or **terminal nodes**.

### ****Example-****

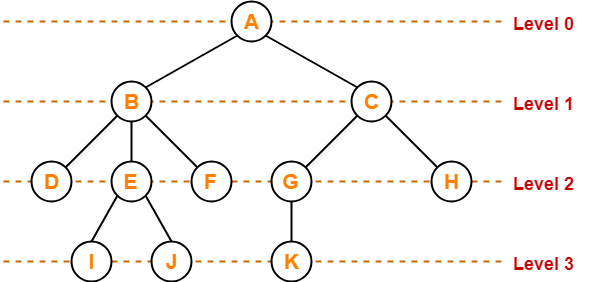


Here, nodes D, I, J, F, K and H are leaf nodes.

## **9. Level-**

* In a tree, each step from top to bottom is called as **level of a tree**.
* The level count starts with 0 and increments by 1 at each level or step.

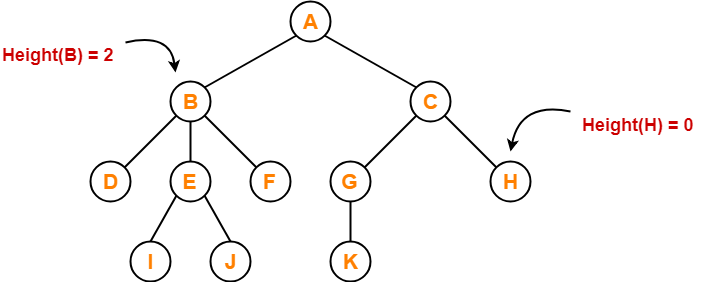
### ****Example-****



## **10. Height-**

* Total number of edges that lies on the longest path from any leaf node to a particular node is called as **height of that node**.
* **Height of a tree** is the height of root node.
* Height of all leaf nodes = 0

### ****Example-****



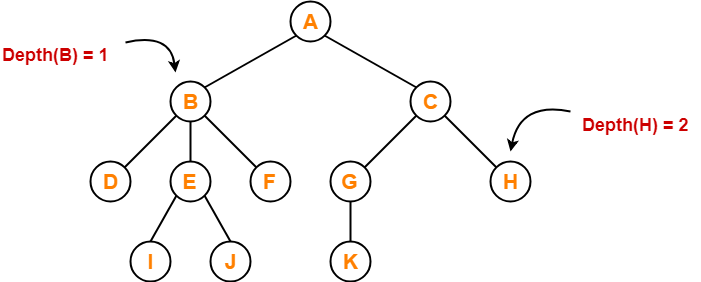
Here,

* Height of node A = 3
* Height of node B = 2
* Height of node C = 2
* Height of node D = 0
* Height of node E = 1
* Height of node F = 0
* Height of node G = 1
* Height of node H = 0
* Height of node I = 0
* Height of node J = 0
* Height of node K = 0

## **11. Depth-**

* Total number of edges from root node to a particular node is called as **depth of that node**.
* **Depth of a tree** is the total number of edges from root node to a leaf node in the longest path.
* Depth of the root node = 0
* The terms “level” and “depth” are used interchangeably.

### ****Example-****



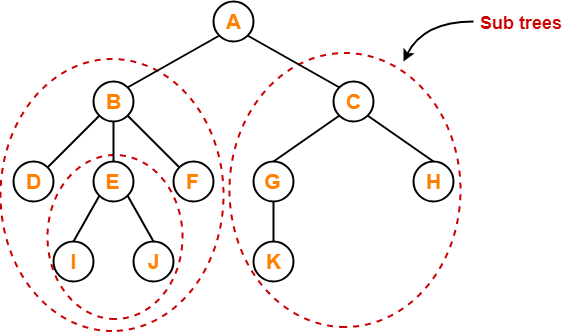
Here,

* Depth of node A = 0
* Depth of node B = 1
* Depth of node C = 1
* Depth of node D = 2
* Depth of node E = 2
* Depth of node F = 2
* Depth of node G = 2
* Depth of node H = 2
* Depth of node I = 3
* Depth of node J = 3
* Depth of node K = 3

## **12. Subtree-**

* In a tree, each child from a node forms a **subtree** recursively.
* Every child node forms a subtree on its parent node.

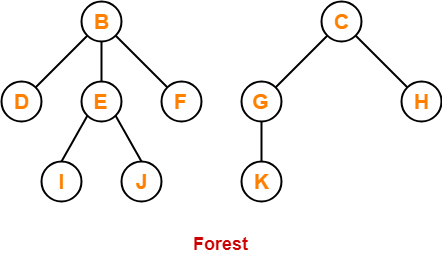
### ****Example-****



## **13. Forest-**

A forest is a set of disjoint trees.

### ****Example-****

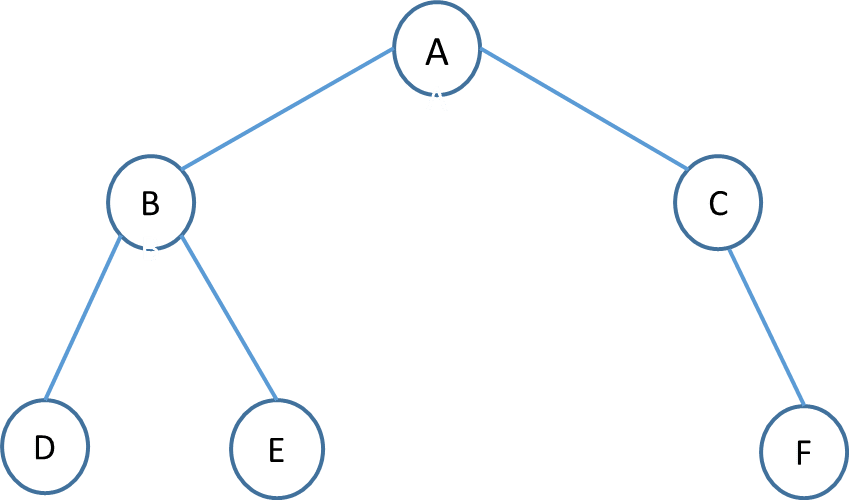


* Implementation of Tree
* The tree data structure can be created by creating the nodes dynamically with the help of the pointers. The tree in the memory can be represented as shown below:
* 
* The above figure shows the representation of the tree data structure in the memory. In the above structure, the node contains three fields. The second field stores the data; the first field stores the address of the left child, and the third field stores the address of the right child.

## **Types of Trees**

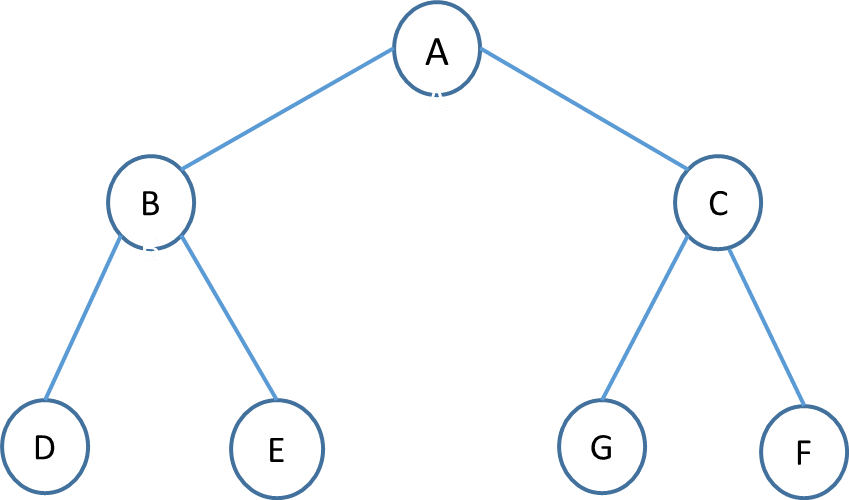
Types of trees depend on the number of children a node has. There are two major tree types:

* **General Tree**: A tree in which there is no restriction on the number of children a node has, is called a General tree. Examples are Family tree, Folder Structure.
* **Binary Tree**: In a Binary tree, every node can have at most 2 children, left and right.  In diagram below, B & D are left children and  C, E & F are right children.



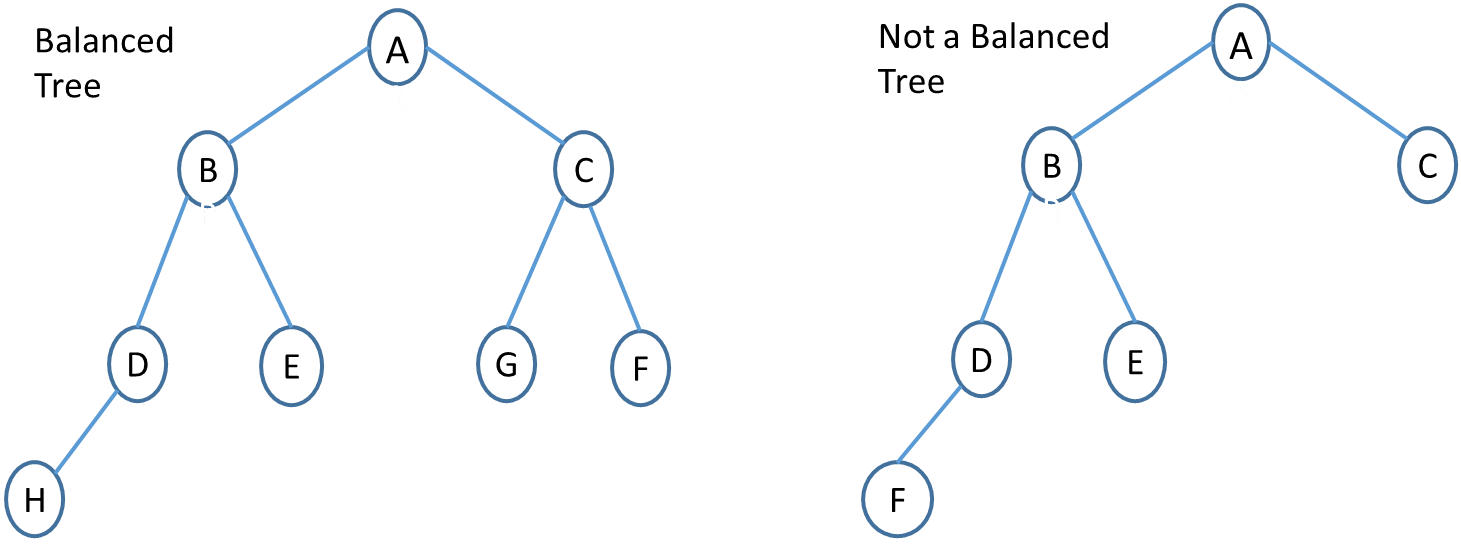
Binary trees are further divided into many types based on its application.

* **Full Binary Tree**: If every node in a tree has either 0 or 2 children, then the tree is called a full tree. The tree in the above diagram is **not a full** binary tree as node C has only the right child.
* **Perfect Binary tree**: It is a binary tree in which all interior nodes have two children and all leaves have the same depth or same level.

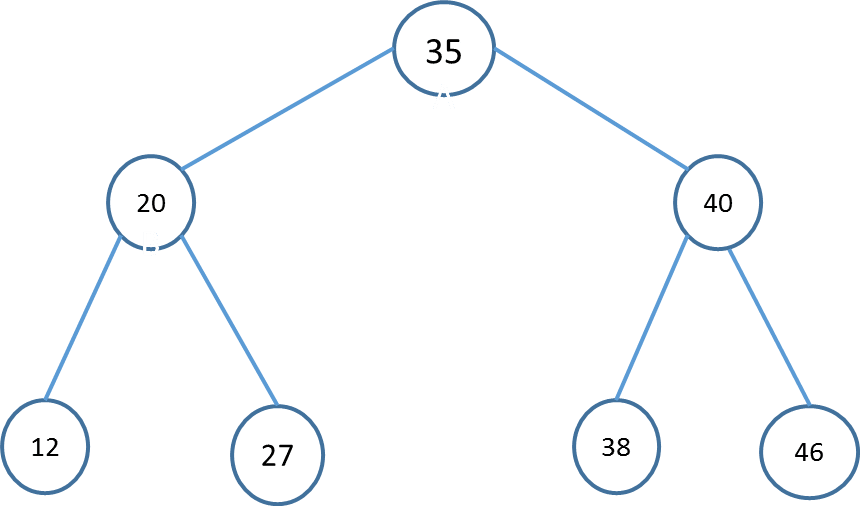


In perfect full binary tree, l = 2h and n = 2h+1 – 1 where, n is number of nodes, h is height of tree and l is number of leaf nodes. In the above diagram, h is 2 so leaves will be 4 and nodes will 23 – 1 which is 7.

* **Balanced Tree**: If the height of the left and right subtree at any node differs at most by -1,0 ,1, then the tree is called a balanced tree.
* **Balance Factor=(left height-right height)**



* **Binary Search Tree**: It is a binary tree with binary search property. Binary search property states that the value or key of the left node is less than its parent and value or key of right node is greater than its parent. And this is true for all nodes.



Binary search trees are used in various searching and sorting algorithms. There are many variants of binary search trees like AVL tree, B-Tree, Red-black tree, etc.

**Tree Traversal**

**Inorder(root)**

* Traverse the left sub-tree, (recursively call inorder(root -> left).
* Visit and print the root node.
* Traverse the right sub-tree, (recursively call inorder(root -> right).

**Preorder(root)**

* Visit and print the root node.
* Traverse the left sub-tree, (recursively call inorder(root -> left).
* Traverse the right sub-tree, (recursively call inorder(root -> right).

**Postorder(root)**

* Traverse the left sub-tree, (recursively call inorder(root -> left).
* Traverse the right sub-tree, (recursively call inorder(root -> right).
* Visit and print the root node.