Tree

In linear data structure data is organized in sequential order and in non-linear data structure data is organized in random order. A tree is a very popular non-linear data structure used in a wide range of applications. A tree data structure can be defined as follows...

*Tree is a non-linear data structure which organizes data in hierarchical structure and this is a recursive definition.*

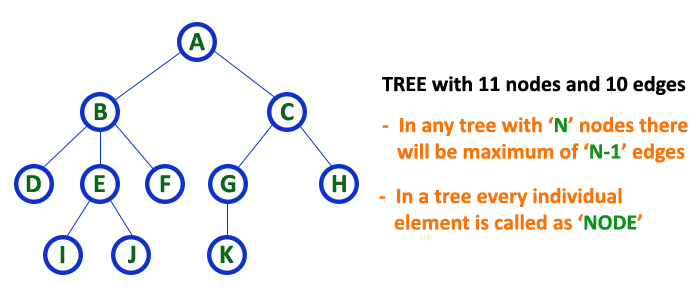
A tree data structure can also be defined as follows...

*Tree data structure is a collection of data (Node) which is organized in hierarchical structure recursively.*

In tree data structure, every individual element is called as Node. Node in a tree data structure stores the actual data of that particular element and link to next element in hierarchical structure.

In a tree, if we have n number of nodes then we can have a maximum of n-1 number of links.

Example

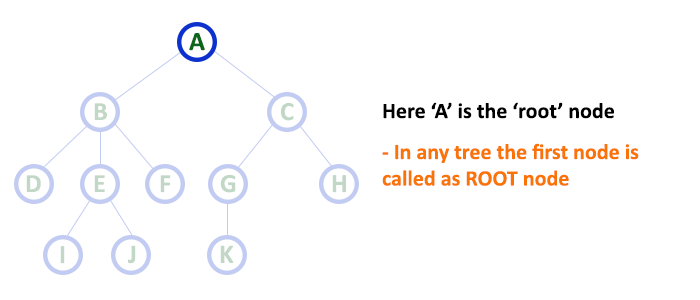


Terminology

In a tree data structure, we use the following terminology...

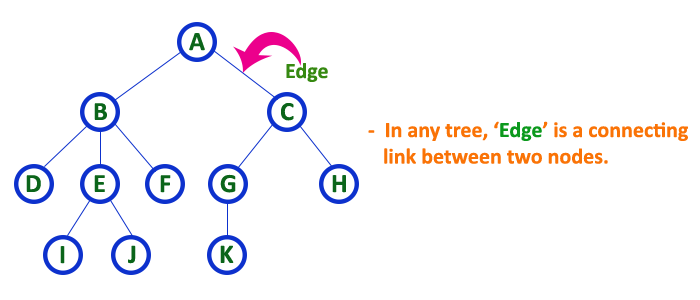
Root

In a tree data structure, the first node is called as Root Node. Every tree must have a root node. We can say that the root node is the origin of the tree data structure. In any tree, there must be only one root node. We never have multiple root nodes in a tree.



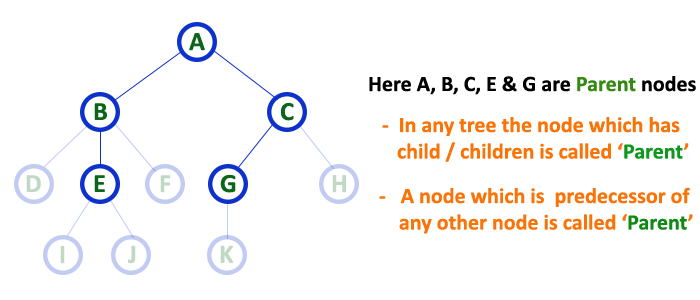
Edge

In a tree data structure, the connecting link between any two nodes is called as EDGE. In a tree with 'N' number of nodes there will be a maximum of 'N-1' number of edges.



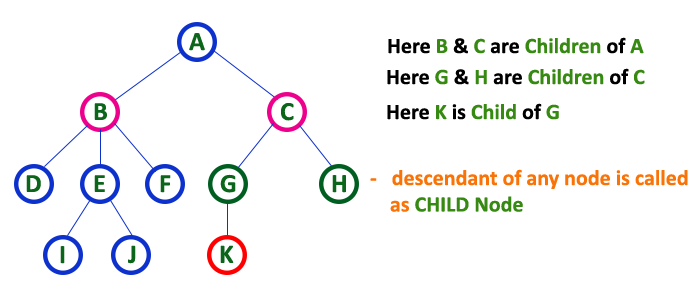
Parent

In a tree data structure, the node which is a predecessor of any node is called as PARENT NODE. In simple words, the node which has a branch from it to any other node is called a parent node. Parent node can also be defined as "The node which has child / children".



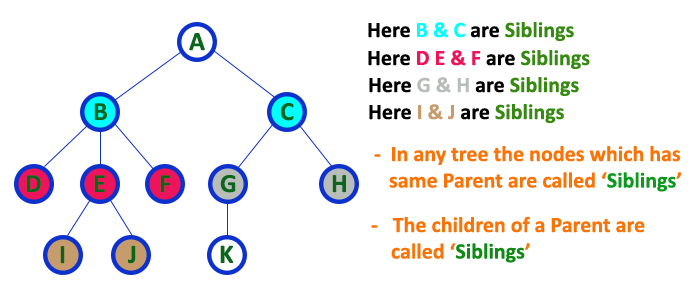
Child

In a tree data structure, the node which is descendant of any node is called as CHILD Node. In simple words, 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.



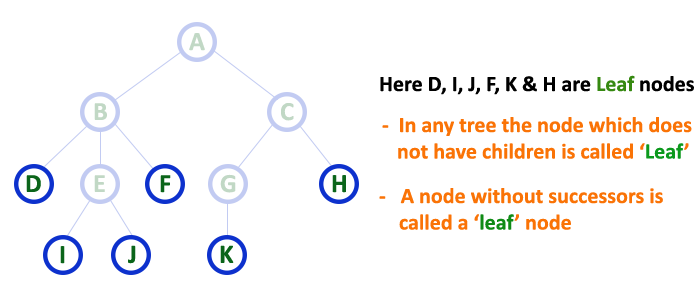
Siblings

In a tree data structure, nodes which belong to same Parent are called as SIBLINGS. In simple words, the nodes with the same parent are called Sibling nodes.



Leaf

In a tree data structure, the node which does not have a child is called as LEAF Node. In simple words, a leaf is a node with no child.

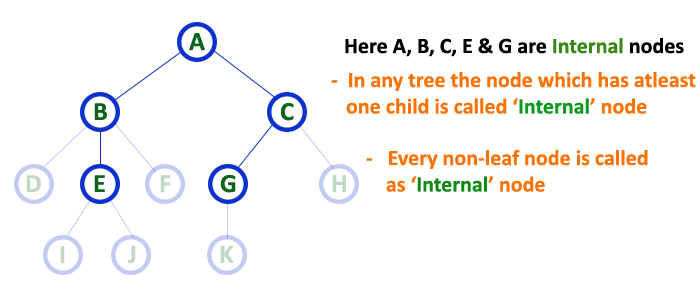


In a tree data structure, the leaf nodes are also called as External Nodes. External node is also a node with no child. In a tree, leaf node is also called as 'Terminal' node.

Internal Nodes

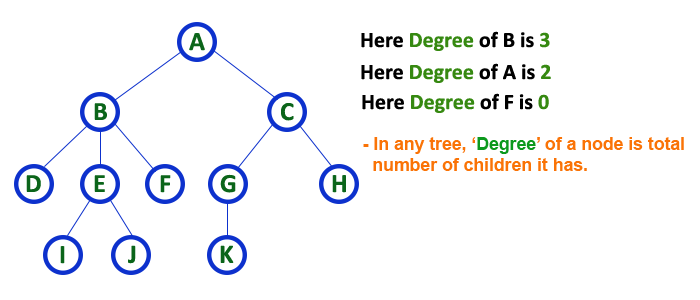
In a tree data structure, the node which has at least one child is called as INTERNAL Node. In simple words, an internal node is a node with at least one child.

In a tree data structure, 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.



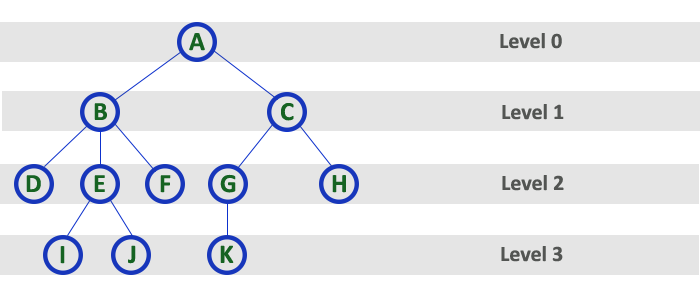
Degree

In a tree data structure, the total number of children of a node is called as DEGREE of that Node. In simple words, the Degree of a node is total number of children it has. The highest degree of a node among all the nodes in a tree is called as 'Degree of Tree'



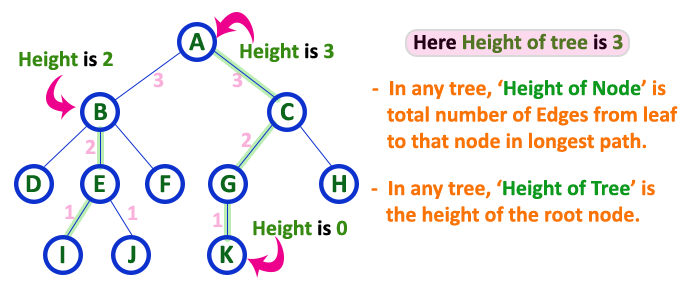
Level

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 simple words, 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).



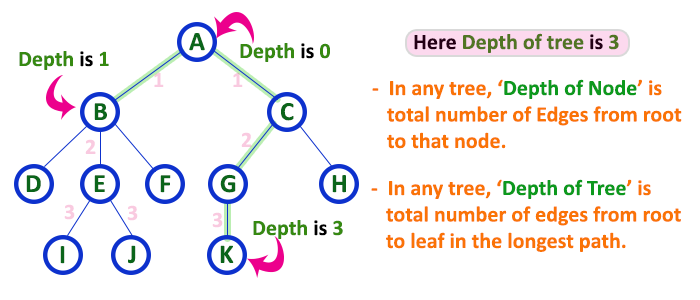
Height

In a tree data structure, the total number of edges from leaf node to a particular node in the longest path is called as HEIGHT of that Node. In a tree, height of the root node is said to be height of the tree. In a tree, height of all leaf nodes is '0'.



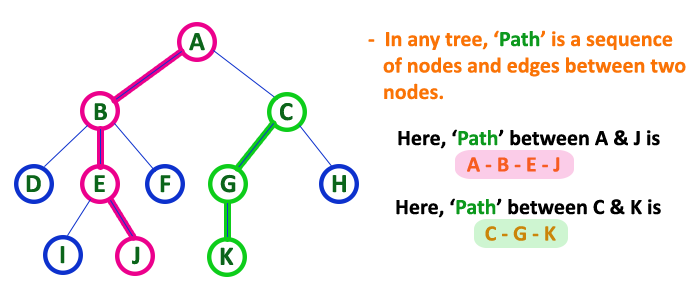
Depth

In a tree data structure, the total number of edges from root node to a particular node is called as DEPTH of that Node. In a tree, the total number of edges from root node to a leaf node in the longest path is said to be Depth of the tree. In simple words, the highest depth of any leaf node in a tree is said to be depth of that tree. In a tree, depth of the root node is '0'.



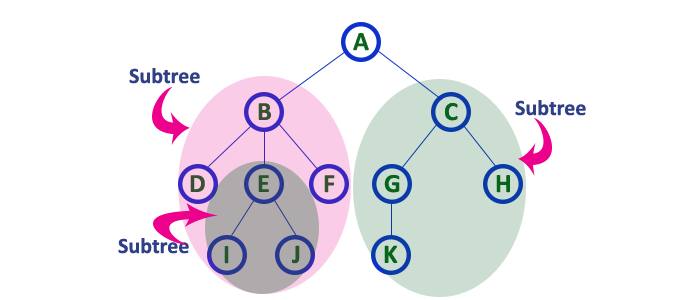
Path

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. **[Ancestor: path from root to leaf node ]**



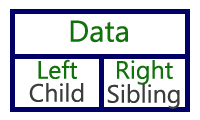
Sub Tree

In a tree data structure, each child from a node forms a subtree recursively. Every child node will form a subtree on its parent node.



# Tree Representation:

Left-Child Right Sibling



# Binary Tree Representations

A binary tree data structure is represented using two methods. Those methods are as follows...

* Array Representation
* Linked List Representation

Consider the following binary tree...

## 1. Array Representation of Binary Tree

In array representation of a binary tree, we use one-dimensional array (1-D Array) to represent a binary tree.

Consider the above example of a binary tree and it is represented as follows...



To represent a binary tree of depth 'n' using array representation, we need one dimensional array with a maximum size of 2n + 1.

2. Linked List Representation of Binary Tree

We use a double linked list to represent a binary tree. In a double linked list, every node consists of three fields. First field for storing left child address, second for storing actual data and third for storing right child address.

In this linked list representation, a node has the following structure...



The above example of the binary tree represented using Linked list representation is shown as follows...

K

J

I

B

C

H

G

F

D

A

# Binary Tree Traversals

When we wanted to display a binary tree, we need to follow some order in which all the nodes of that binary tree must be displayed. In any binary tree, displaying order of nodes depends on the traversal method.

Displaying (or) visiting order of nodes in a binary tree is called as Binary Tree Traversal.

There are three types of binary tree traversals.

* In - Order Traversal
* Pre - Order Traversal
* Post - Order Traversal

Consider the following binary tree...

### **1. In - Order Traversal ( left Child - root – right Child )**

In In-Order traversal, the root node is visited between the left child and right child. In this traversal, the left child node is visited first, then the root node is visited and later we go for visiting the right child node. This in-order traversal is applicable for every root node of all subtrees in the tree. This is performed recursively for all nodes in the tree.

In the above example of a binary tree, first we try to visit left child of root node 'A', but A's left child 'B' is a root node for left subtree. so we try to visit its (B's) left child 'D' and again D is a root for subtree with nodes D, I and J. So we try to visit its left child 'I' and it is the leftmost child. So first we visit 'I' then go for its root node 'D' and later we visit D's right child 'J'. With this we have completed the left part of node B. Then visit 'B' and next B's right child 'F' is visited. With this we have completed left part of node A. Then visit root node 'A'. With this we have completed left and root parts of node A. Then we go for the right part of the node A. In right of A again there is a subtree with root C. So go for left child of C and again it is a subtree with root G. But G does not have left part so we visit 'G' and then visit G's right child K. With this we have completed the left part of node C. Then visit root node 'C' and next visit C's right child 'H' which is the rightmost child in the tree. So we stop the process.

That means here we have visited in the order of I - D - J - B - F - A - G - K - C - H using In-Order Traversal.

In-Order Traversal for above example of binary tree is

**I - D - J - B - F - A - G - K - C - H**

### **2. Pre - Order Traversal ( root – left Child – right Child )**

In Pre-Order traversal, the root node is visited before the left child and right child nodes. In this traversal, the root node is visited first, then its left child and later its right child. This pre-order traversal is applicable for every root node of all subtrees in the tree.

In the above example of binary tree, first we visit root node 'A' then visit its left child 'B' which is a root for D and F. So we visit B's left child 'D' and again D is a root for I and J. So we visit D's left child 'I' which is the leftmost child. So next we go for visiting D's right child 'J'. With this we have completed root, left and right parts of node D and root, left parts of node B. Next visit B's right child 'F'. With this we have completed root and left parts of node A. So we go for A's right child 'C' which is a root node for G and H. After visiting C, we go for its left child 'G' which is a root for node K. So next we visit left of G, but it does not have left child so we go for G's right child 'K'. With this, we have completed node C's root and left parts. Next visit C's right child 'H' which is the rightmost child in the tree. So we stop the process.

That means here we have visited in the order of A-B-D-I-J-F-C-G-K-H using Pre-Order Traversal.

Pre-Order Traversal for above example binary tree is

**A - B - D - I - J - F - C - G - K - H**

### **3. Post - Order Traversal ( left Child – right Child - root )**

In Post-Order traversal, the root node is visited after left child and right child. In this traversal, left child node is visited first, then its right child and then its root node. This is recursively performed until the right most node is visited.

Here we have visited in the order of I - J - D - F - B - K - G - H - C - A using Post-Order Traversal.

Post-Order Traversal for above example binary tree is

**I - J - D - F - B - K - G - H - C – A**

# Threaded Binary Trees

A binary tree can be represented using array representation or linked list representation. When a binary tree is represented using linked list representation, the reference part of the node which doesn't have a child is filled with a NULL pointer. In any binary tree linked list representation, there is a number of NULL pointers than actual pointers. Generally, in any binary tree linked list representation, if there are 2N number of reference fields, then N+1 number of reference fields are filled with NULL ( N+1 are NULL out of 2N ). This NULL pointer does not play any role except indicating that there is no link (no child).

A. J. Perlis and C. Thornton have proposed new binary tree called "Threaded Binary Tree", which makes use of NULL pointers to improve its traversal process. In a threaded binary tree, NULL pointers are replaced by references of other nodes in the tree. These extra references are called as threads.

***Threaded Binary Tree is also a binary tree in which all left child pointers that are NULL (in Linked list representation) points to its in-order predecessor, and all right child pointers that are NULL (in Linked list representation) points to its in-order successor.***

If there is no in-order predecessor or in-order successor, then it points to the root node.

Consider the following binary tree...

To convert the above example binary tree into a threaded binary tree, first find the in-order traversal of that tree...

In-order traversal of above binary tree...

**H - D - I - B - E - A - F - J - C - G**

When we represent the above binary tree using linked list representation, nodes H, I, E, F, J and G left child pointers are NULL. This NULL is replaced by address of its in-order predecessor respectively (I to D, E to B, F to A, J to F and G to C), but here the node H does not have its in-order predecessor, so it points to the root node A. And nodes H, I, E, J and G right child pointers are NULL. These NULL pointers are replaced by address of its in-order successor respectively (H to D, I to B, E to A, and J to C), but here the node G does not have its in-order successor, so it points to the root node A.