UNIT-IV

Tree Terminology

In linear data structure, data is organized in sequential order and in non-linear data structure, data is organized in random order. Tree is a very popular data structure used in 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.

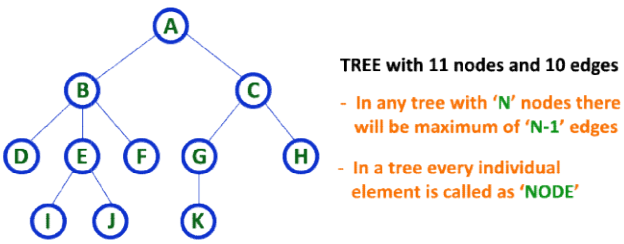
A tree data structure can also be defined as follows...

Tree data structure is a collection of data (Node) which is organized in hierarchical structure and this is a recursive definition

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 data structure, if we have N number of nodes then we can have a maximum of N-1 number of links.

Example

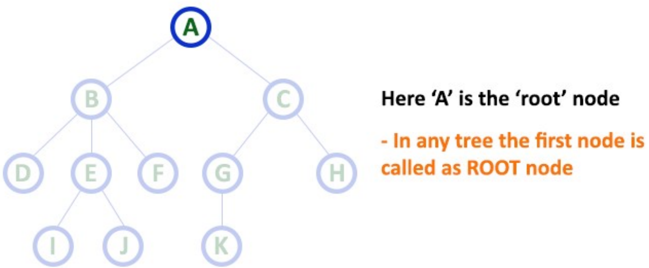


Terminology

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

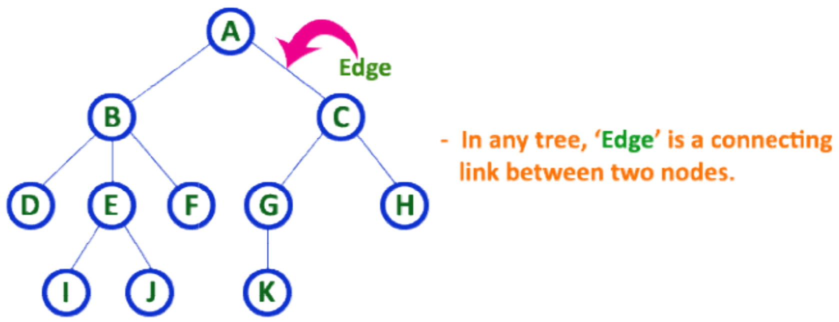
1. Root

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



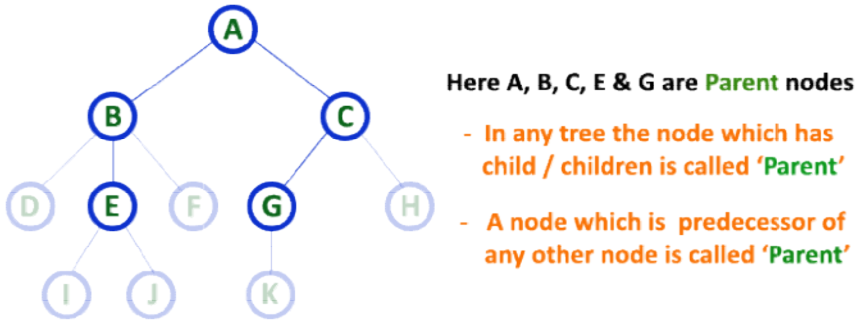
2. Edge

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.



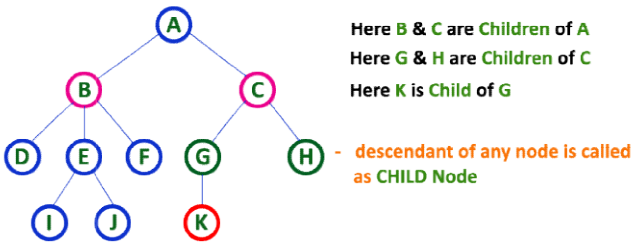
3. Parent

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



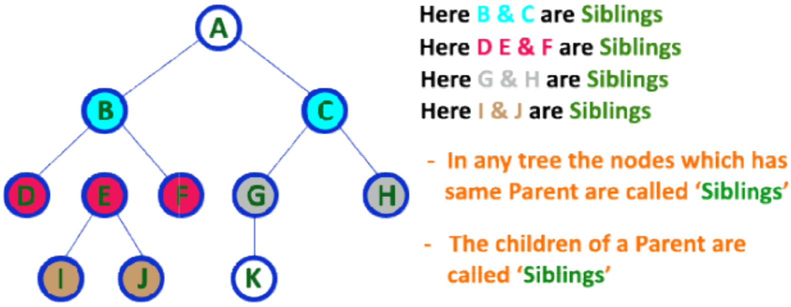
4. Child

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.



5. Siblings

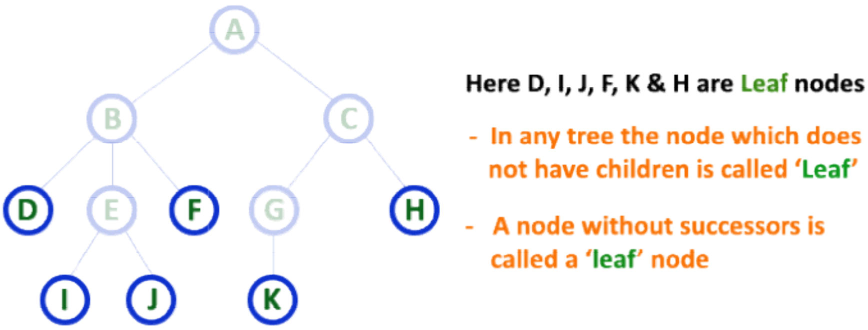
In a tree data structure, nodes which belong to same Parent are called as SIBLINGS. In simple words, the nodes with same parent are called as Sibling nodes.



6. 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.

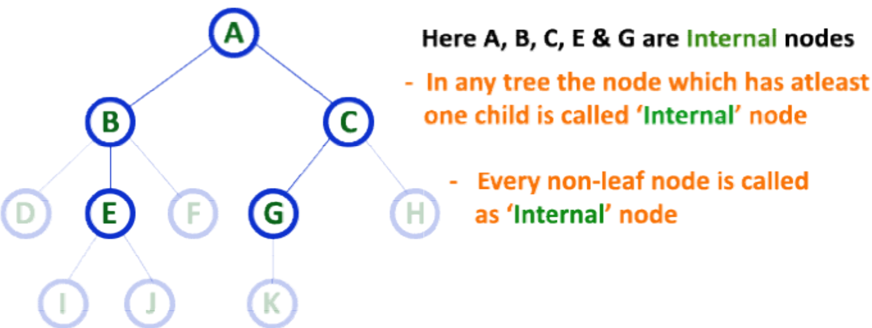
In a tree data structure, the leaf nodes are also called as External Nodes. External node is also a node with no child. In a tree, leaf node is also called as 'Terminal' node.



7. Internal Nodes

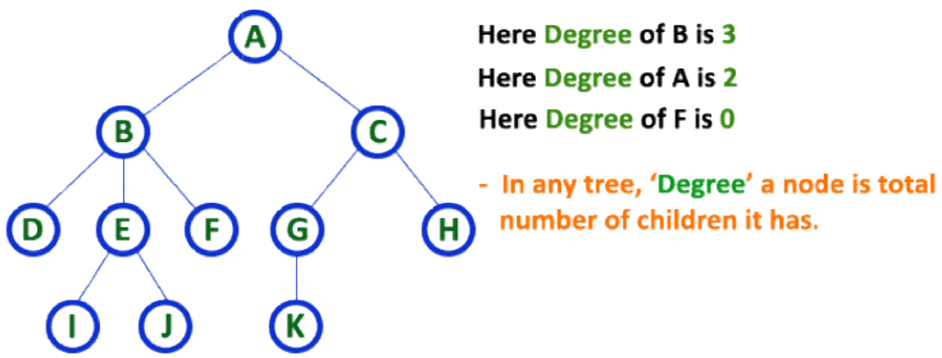
In a tree data structure, the node which has atleast one child is called as INTERNAL Node. In simple words, an internal node is a node with atleast one child.

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.



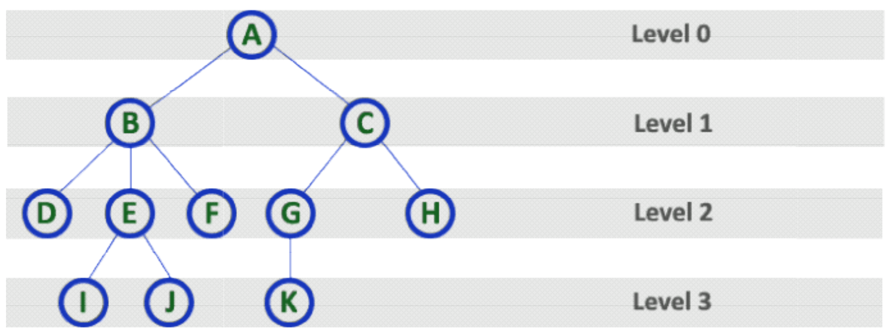
8. Degree

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'



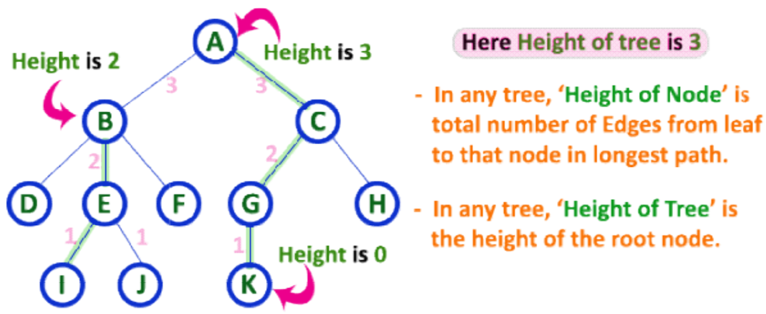
9. Level

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 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).



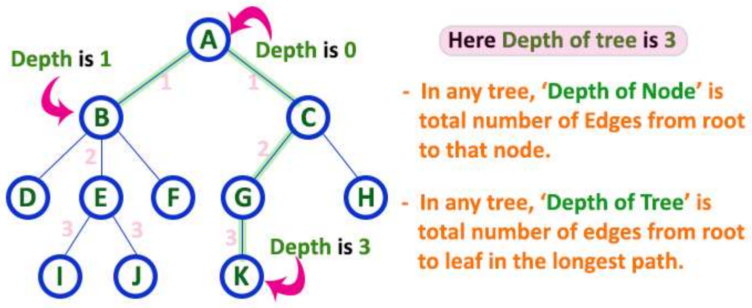
10. Height

In a tree data structure, the total number of egdes 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'.



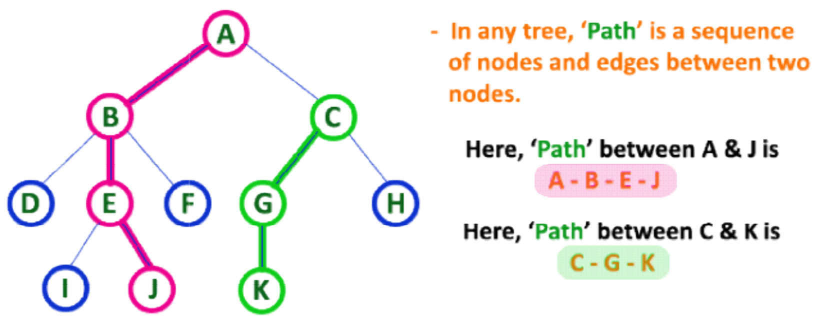
11. Depth

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



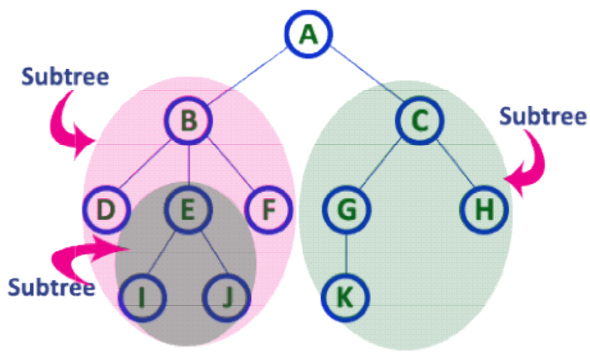
12. Path

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

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.



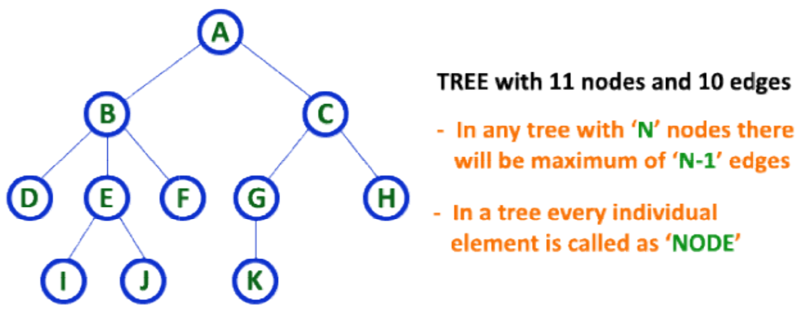
Tree Representations

A tree data structure can be represented in two methods. Those methods are as follows...

List Representation

Left Child - Right Sibling Representation

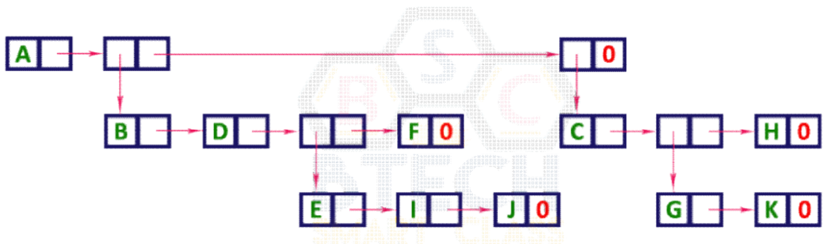
Consider the following tree...



1. List Representation

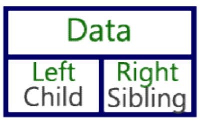
In this representation, we use two types of nodes one for representing the node with data and another for representing only references. We start with a node with data from root node in the tree. Then it is linked to an internal node through a reference node and is linked to any other node directly. This process repeats for all the nodes in the tree.

The above tree example can be represented using List representation as follows...



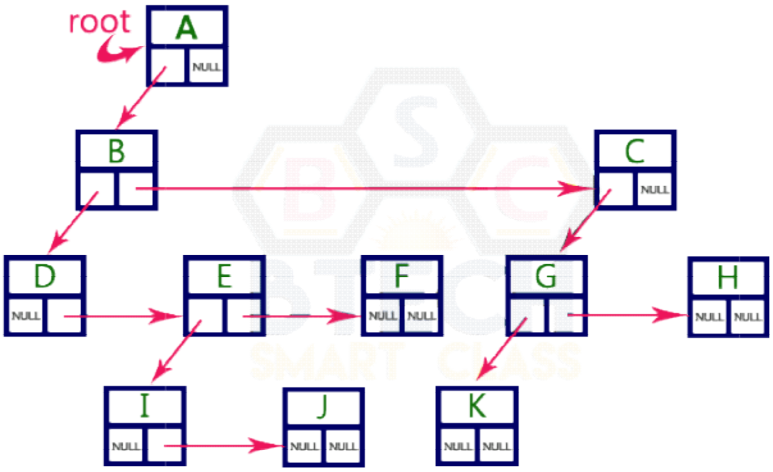
2. Left Child - Right Sibling Representation

In this representation, we use list with one type of node which consists of three fields namely Data field, Left child reference field and Right sibling reference field. Data field stores the actual value of a node, left reference field stores the address of the left child and right reference field stores the address of the right sibling node. Graphical representation of that node is as follows...



In this representation, every node's data field stores the actual value of that node. If that node has left child, then left reference field stores the address of that left child node otherwise that field stores NULL. If that node has right sibling then right reference field stores the address of right sibling node otherwise that field stores NULL.

The above tree example can be represented using Left Child - Right Sibling representation as follows...



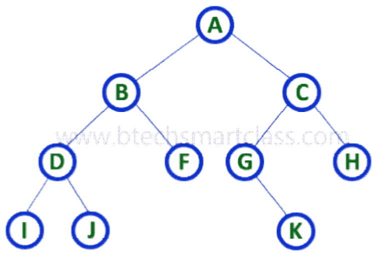
Binary Tree

In a normal tree, every node can have any number of children. Binary tree is a special type of tree data structure in which every node can have a maximum of 2 children. One is known as left child and the other is known as right child.

A tree in which every node can have a maximum of two children is called as Binary Tree.

In a binary tree, every node can have either 0 children or 1 child or 2 children but not more than 2 children.

Example



There are different types of binary trees and they are...

1. Strictly Binary Tree

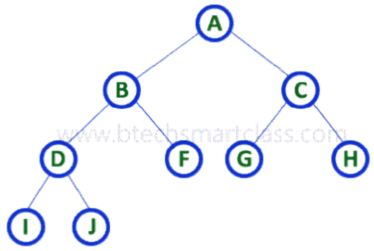
In a binary tree, every node can have a maximum of two children. But in strictly binary tree, every node should have exactly two children or none. That means every internal node must have exactly two children. A strictly Binary Tree can be defined as follows...

A binary tree in which every node has either two or zero number of children is called Strictly Binary Tree

---

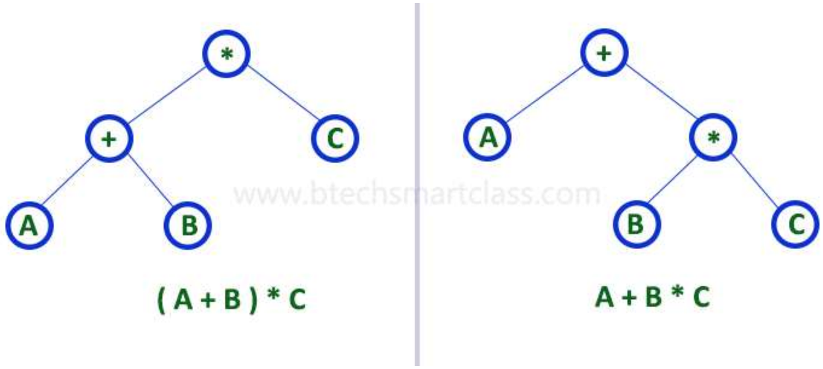
Strictly binary tree is also called as Full Binary Tree or Proper Binary Tree or 2-Tree

---



Strictly binary tree data structure is used to represent mathematical expressions.

Example

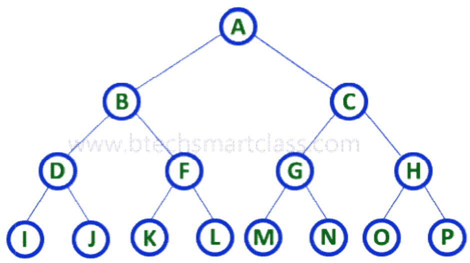


2. Complete Binary Tree

In a binary tree, every node can have a maximum of two children. But in strictly binary tree, every node should have exactly two children or none and in complete binary tree all the nodes must have exactly two children and at every level of complete binary tree there must be 2level number of nodes. For example at level 2 there must be 22 = 4 nodes and at level 3 there must be 23 = 8 nodes.

A binary tree in which every internal node has exactly two children and all leaf nodes are at same level is called Complete Binary Tree.

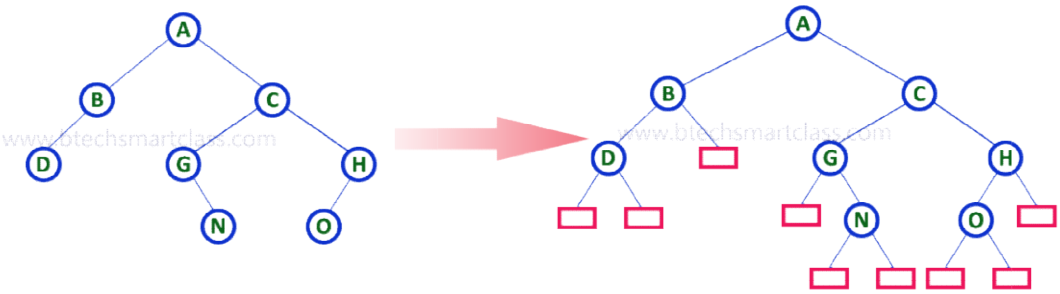
Complete binary tree is also called as Perfect Binary Tree



3. Extended Binary Tree

A binary tree can be converted into Full Binary tree by adding dummy nodes to existing nodes wherever required.

The full binary tree obtained by adding dummy nodes to a binary tree is called as Extended Binary Tree.



In above figure, a normal binary tree is converted into full binary tree by adding dummy nodes (In pink colour).

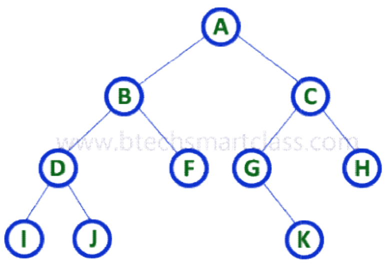
Binary Tree Representations

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

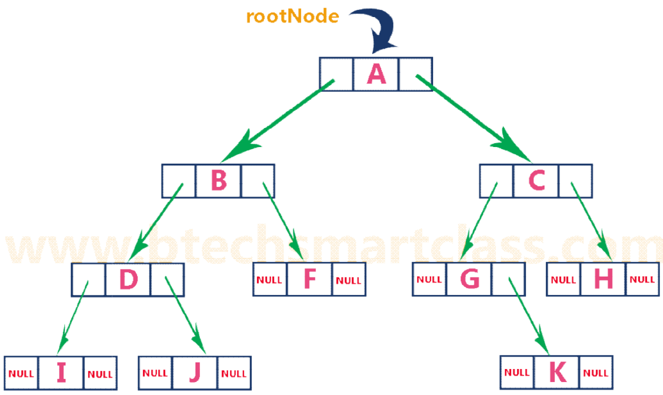
In array representation of binary tree, we use a one dimensional array (1-D Array) to represent a binary tree. Consider the above example of binary tree and it is represented as follows...

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

2. Linked List Representation

We use 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 binary tree represented using Linked list representation is shown as follows...



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.

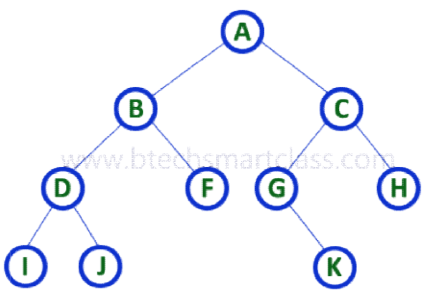
There are three types of binary tree traversals.

In - Order Traversal

Pre - Order Traversal

Post - Order Traversal

Consider the following binary tree...



1. In - Order Traversal (leftChild - root - rightChild)

In In-Order traversal, the root node is visited between 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 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 binary tree, first we try to visit left child of root node 'A', but A's left child 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 left most 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 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 right most 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 - leftChild - rightChild)

In Pre-Order traversal, the root node is visited before 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 left most 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 right most 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

2. Post - Order Traversal (leftChild - rightChild - 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 Tree

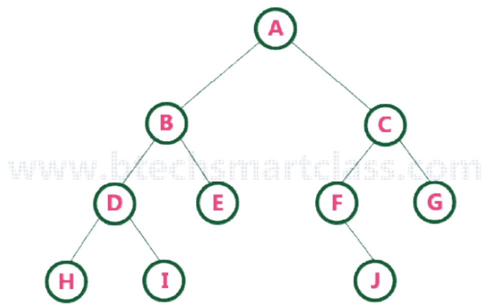
A binary tree is represented using array representation or linked list representation. When a binary tree is represented using linked list representation, if any node is not having a child we use NULL pointer in that position. In any binary tree linked list representation, there are more number of NULL pointer 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 there is no link (no child).

A. J. Perlis and C. Thornton have proposed new binary tree called "Threaded Binary Tree", which make use of NULL pointer to improve its traversal processes. In threaded binary tree, NULL pointers are replaced by references to other nodes in the tree, called 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 point to root node.

Consider the following binary tree...



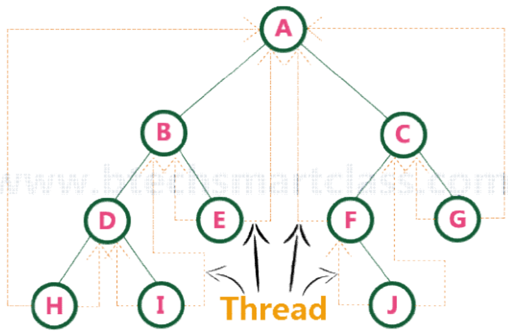
To convert above binary tree into threaded binary tree, first find the in-order traversal of that tree...

In-order traversal of above binary tree...

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

When we represent 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. This NULL ponters 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.

Above example binary tree become as follows after converting into threaded binary tree.



In above figure threadeds are indicated with dotted links.

Binary Search Tree

In a binary tree, every node can have maximum of two children but there is no order of nodes based on their values. In binary tree, the elements are arranged as they arrive to the tree, from top to bottom and left to right.

A binary tree has the following time complexities...

Search Operation - O(n)

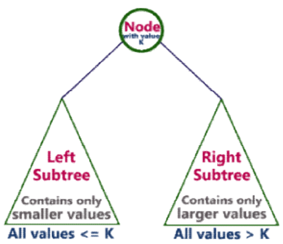
Insertion Operation - O(1)

Deletion Operation - O(n)

To enhance the performance of binary tree, we use special type of binary tree known as Binary Search Tree. Binary search tree mainly focus on the search operation in binary tree. Binary search tree can be defined as follows...

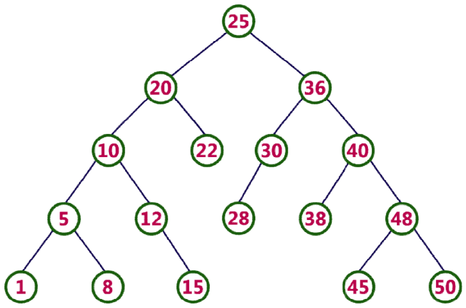
Binary Search Tree is a binary tree in which every node contains only smaller values in its left subtree and only larger values in its right subtree.

In a binary search tree, all the nodes in left subtree of any node contains smaller values and all the nodes in right subtree of that contains larger values as shown in following figure...



Example

The following tree is a Binary Search Tree. In this tree, left subtree of every node contains nodes with smaller values and right subtree of every node contains larger values.



Every Binary Search Tree is a binary tree but all the Binary Trees need not to be binary search trees.

Operations on a Binary Search Tree

The following operations are performed on a binary earch tree...

Search

Insertion

Deletion

Search Operation in BST

In a binary search tree, the search operation is performed with O(log n) time complexity. The search operation is performed as follows...

Step 1: Read the search element from the user

Step 2: Compare, the search element with the value of root node in the tree.

Step 3: If both are matching, then display "Given node found!!!" and terminate the function

Step 4: If both are not matching, then check whether search element is smaller or larger than that node value.

Step 5: If search element is smaller, then continue the search process in left subtree.

Step 6: If search element is larger, then continue the search process in right subtree.

Step 7: Repeat the same until we found exact element or we completed with a leaf node

Step 8: If we reach to the node with search value, then display "Element is found" and terminate the function.

Step 9: If we reach to a leaf node and it is also not matching, then display "Element not found" and terminate the function.

Insertion Operation in BST

In a binary search tree, the insertion operation is performed with O(log n) time complexity. In binary search tree, new node is always inserted as a leaf node. The insertion operation is performed as follows...

Step 1: Create a newNode with given value and set its left and right to NULL.

Step 2: Check whether tree is Empty.

Step 3: If the tree is Empty, then set set root to newNode.

Step 4: If the tree is Not Empty, then check whether value of newNode is smaller or larger than the node (here it is root node).

Step 5: If newNode is smaller than or equal to the node, then move to its left child. If newNode is larger than the node, then move to its right child.

Step 6: Repeat the above step until we reach to a leaf node (e.i., reach to NULL).

Step 7: After reaching a leaf node, then isert the newNode as left child if newNode is smaller or equal to that leaf else insert it as right child.

Deletion Operation in BST

In a binary search tree, the deletion operation is performed with O(log n) time complexity. Deleting a node from Binary search tree has follwing three cases...

Case 1: Deleting a Leaf node (A node with no children)

Case 2: Deleting a node with one child

Case 3: Deleting a node with two children

Case 1: Deleting a leaf node

We use the following steps to delete a leaf node from BST...

Step 1: Find the node to be deleted using search operation

Step 2: Delete the node using free function (If it is a leaf) and terminate the function.

Case 2: Deleting a node with one child

We use the following steps to delete a node with one child from BST...

Step 1: Find the node to be deleted using search operation

Step 2: If it has only one child, then create a link between its parent and child nodes.

Step 3: Delete the node using free function and terminate the function.

Case 3: Deleting a node with two children

We use the following steps to delete a node with two children from BST...

Step 1: Find the node to be deleted using search operation

Step 2: If it has two children, then find the largest node in its left subtree (OR) the smallest node in its right subtree.

Step 3: Swap both deleting node and node which found in above step.

Step 4: Then, check whether deleting node came to case 1 or case 2 else goto steps 2

Step 5: If it comes to case 1, then delete using case 1 logic.

Step 6: If it comes to case 2, then delete using case 2 logic.

Step 7: Repeat the same process until node is deleted from the tree.

Example

Construct a Binary Search Tree by inserting the following sequence of numbers...

10,12,5,4,20,8,7,15 and 13

Above elements are inserted into a Binary Search Tree as follows...