TREES
UNIT-IV

Unit: 4

Data Structures

Course Details
(B Tech 3rd Sem)

Sanchi Kaushik
Assistant Professor
AIML

Content

Course Objective

- To understand the working of tree data structure.
- To understand the applications of tree data structure like data compression using Huffman algorithm.
- To understand the limitations of trees and how these limitations are removed.

Course Outcome

CO	CO Description	Bloom's Knowledge Level (KL)
CO1	Describe how arrays, linked lists, stacks, queues, trees, and graphs are represented in memory, used by the algorithms and their common applications.	K1, K2
CO2	Discuss the computational efficiency of the sorting and searching algorithms.	K2
CO3	Implementation of Trees and Graphs and perform various operations on these data structure.	K3
CO4	Understanding the concept of recursion, application of recursion and its implementation and removal of recursion.	K4
CO5	Identify the alternative implementations of data structures with respect to its performance to solve a real world problem.	K5, K6

CO-PO and PSO Mapping

CO-PO correlation matrix of Data Structure (KCS 301)

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
KCS301.1	3	3	3	2	-	1	-	1	2	2	2	2
KCS301.2	3	3	2	2	-	1	-	1	2	2	1	2
KCS301.3	3	3	2	2	-	1	-	1	2	2	2	2
KCS301.4	3	3	2	2	-	1	-	1	2	2	2	2
KCS301.5	3	3	3	3	2	2	2	2	3	3	3	3
Average	3	3	2.4	2.2	0.4	1.2	0.4	1.2	2.2	2.2	2	2.2

Mapping of Program Specific Outcomes and Course Outcomes

	PSO1	PSO2	PSO3	PSO4
KCS301.1	3	3	2	2
KCS301.2	3	3	2	3
KCS301.3	3	3	2	2
KCS301.4	3	3	3	3
KCS301.5	3	3	3	3
Average	3	3	2.4	2.6

Prerequisite and Recap

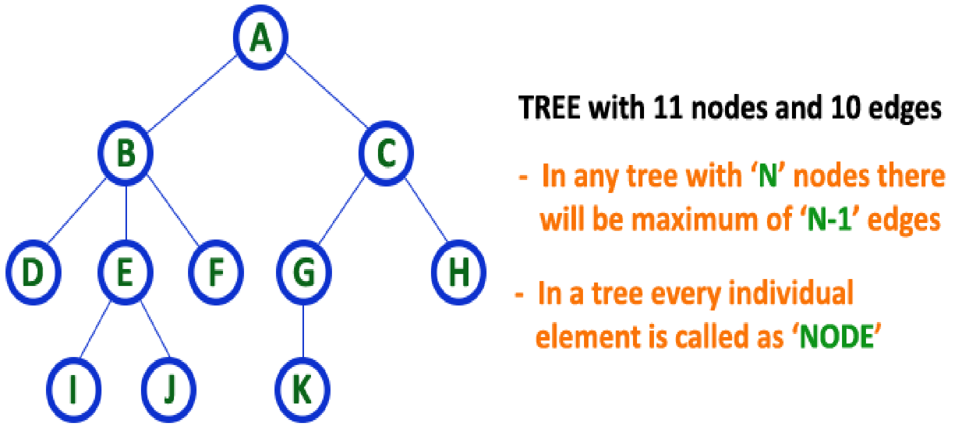
- Knowledge of basics concepts of data structure like searching sorting studied in unit-3.
- Mathematical vision to understand the working and application of trees.
- Programming skills to implement tree data structure in a programming language.

- Recap
  - In last unit we studied
    - About graphs and their application areas.
    - Graph terminology
    - Graph representation and traversal
    - Minimum Spanning Trees
    - Warshall Algorithm

Tree (CO3)

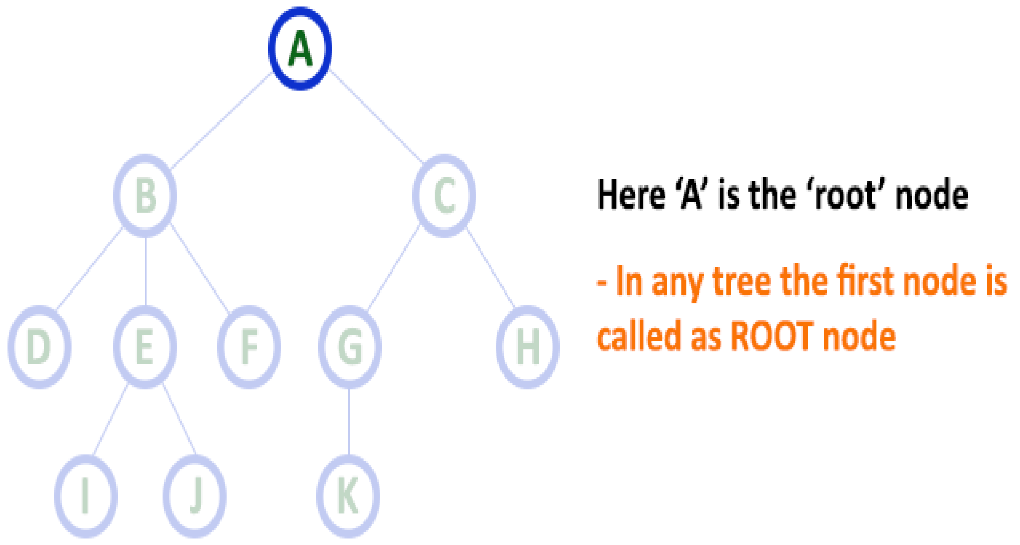
- A tree is a non-linear data structure used in a wide range of applications. A tree data structure can be defined as follows...
  - Tree is a non-linear data structure which organizes data in hierarchical structure.

    OR
  - Tree data structure is a collection of data (Node) which is organized in hierarchical structure.
- In tree data structure, every individual element is called as Node
- 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:-


A
B
C
D
E
F
G
H
I
J
K
TREE with 11 nodes and 10 edges
- In any tree with 'N' nodes there will be maximum of 'N-1' edges
- In a tree every individual element is called as 'NODE'

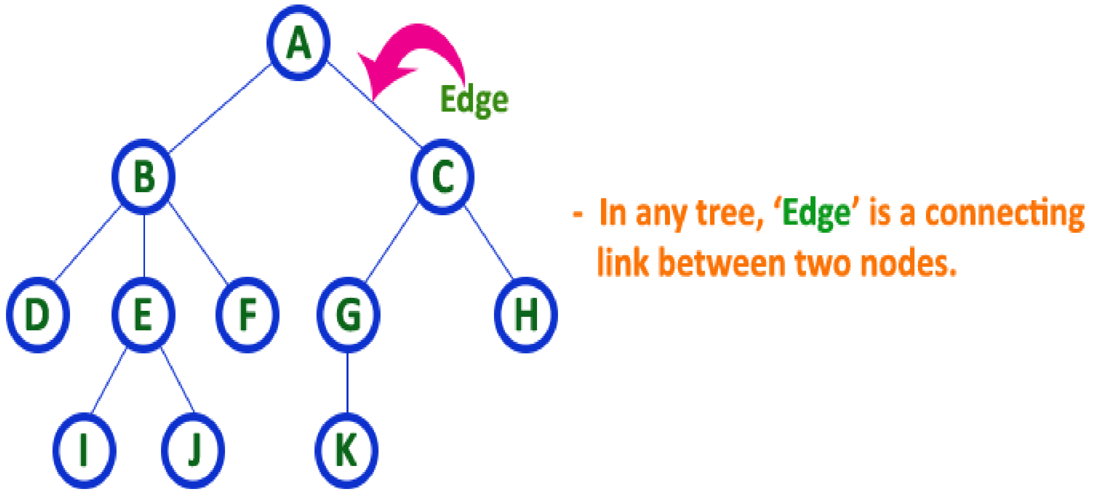
Tree Terminology

- **Root**
  - In a tree data structure, the first node is called as **Root Node**.
  - Every tree must have a root node.
  - Root node is the origin of the tree data structure.
  - In any tree, there must be only one root node.


A
B
C
D
E
F
G
H
I
J
K
Here 'A' is the 'root' node
- In any tree the first node is called as ROOT node

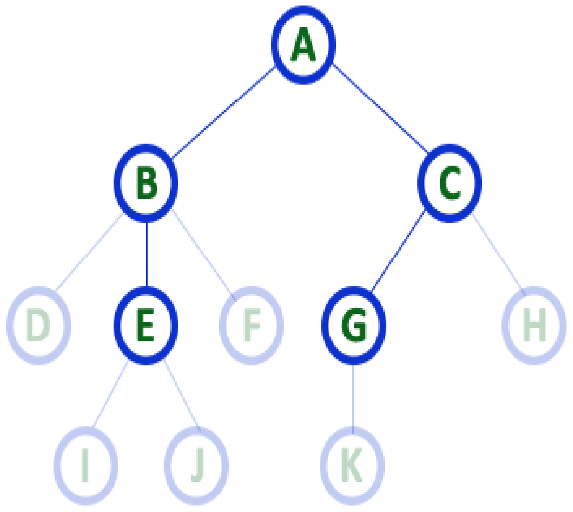
Tree Terminology

- **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.


A
Edge
B
C
D
E
F
G
H
I
J
K
- In any tree, 'Edge' is a connecting link between two nodes.

Tree Terminology

- **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**".

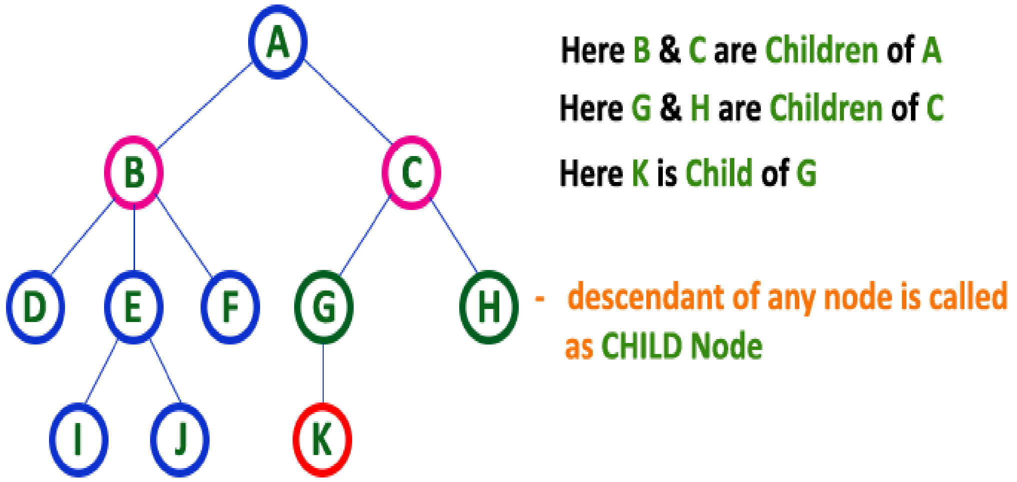


Here A, B, C, E & G are Parent nodes

- In any tree the node which has child / children is called 'Parent'
- A node which is predecessor of any other node is called 'Parent'

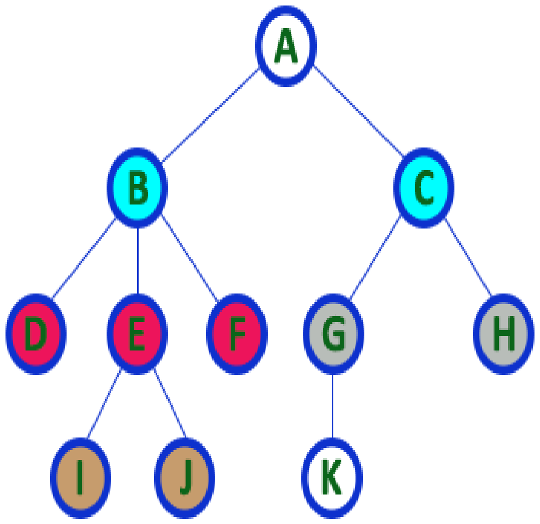
Tree Terminology

- **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.


A
B
C
D
E
F
G
H
I
J
K
Here B & C are Children of A
Here G & H are Children of C
Here K is Child of G
- descendant of any node is called as CHILD Node

Tree Terminology

- **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



Here B & C are Siblings

Here D E & F are Siblings

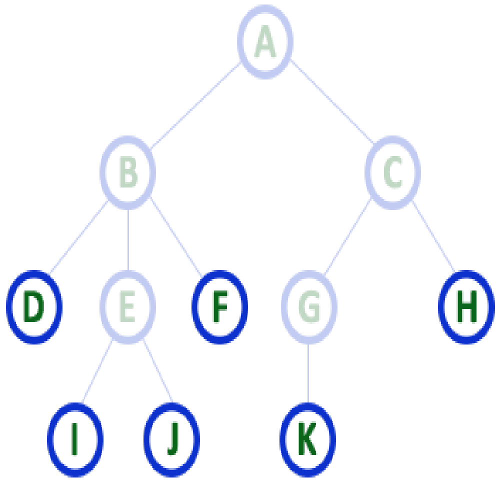
Here G & H are Siblings

Here I & J are Siblings

- In any tree the nodes which has same Parent are called 'Siblings'
- The children of a Parent are called 'Siblings'

Tree Terminology

- **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**.
  - In a tree, leaf node is also called as '**Terminal**' node.

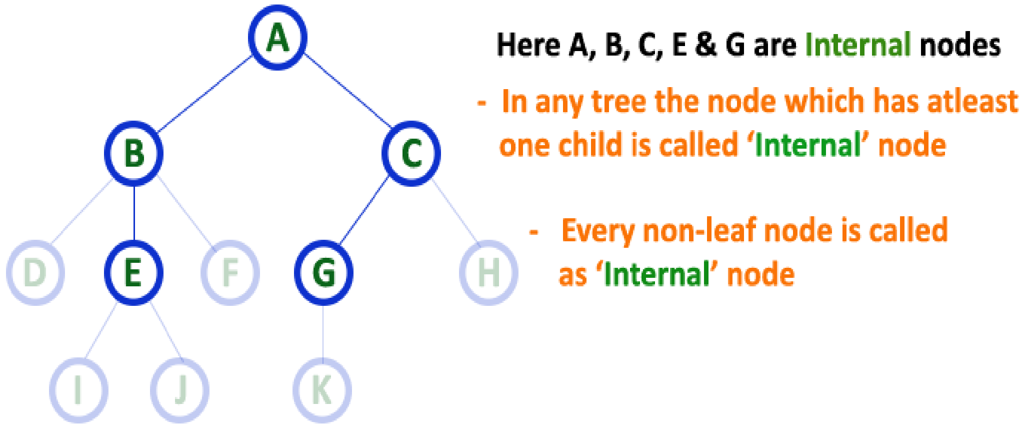


Here D, I, J, F, K & H are Leaf nodes

- In any tree the node which does not have children is called 'Leaf'
- A node without successors is called a 'leaf' node

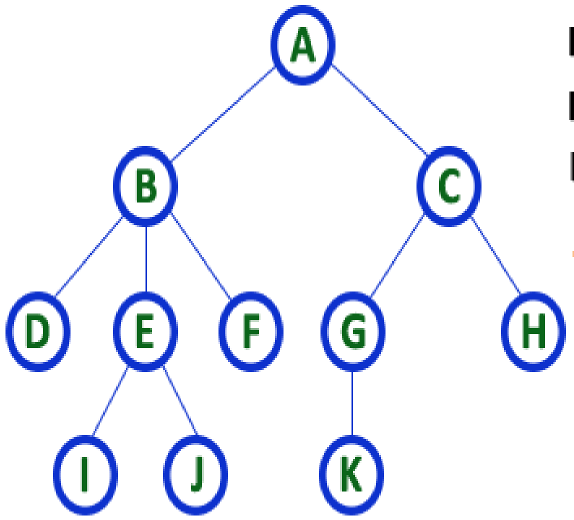
Tree Terminology

- **Internal Nodes**
  - In a tree data structure, the node which has atleast one child is called as **INTERNAL Node**.
  - 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.


A
B
C
D
E
F
G
H
I
J
K
Here A, B, C, E & G are Internal nodes
- In any tree the node which has atleast one child is called 'Internal' node
- Every non-leaf node is called as 'Internal' node

Tree Terminology

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



Here Degree of B is 3

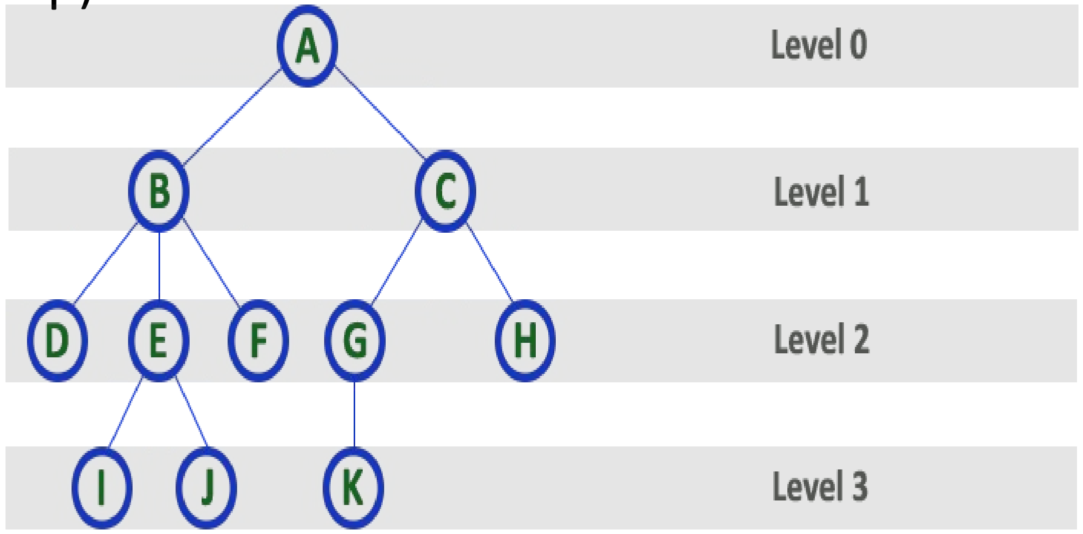
Here Degree of A is 2

Here Degree of F is 0

- In any tree, 'Degree' of a node is total number of children it has.

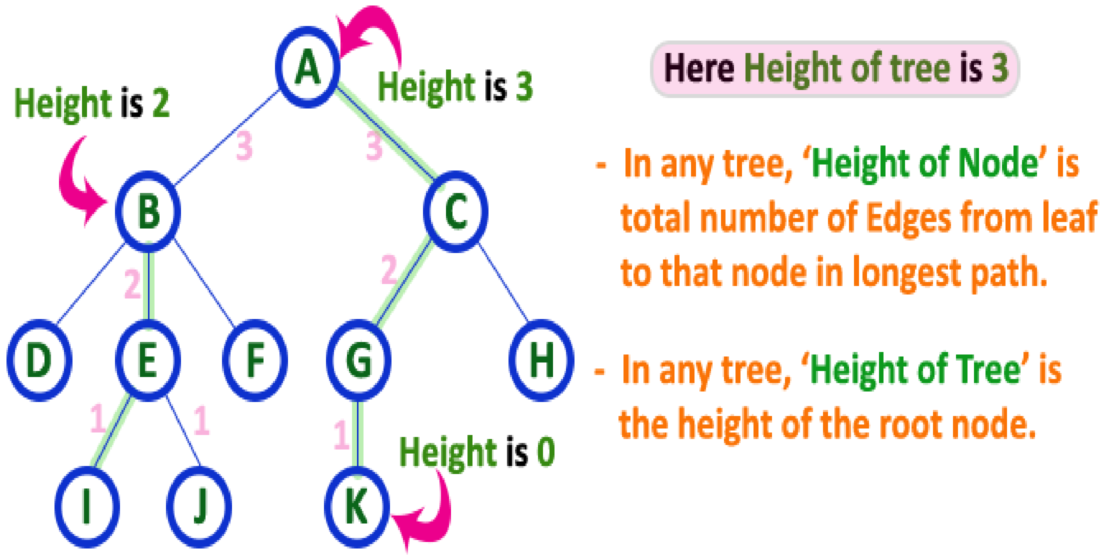
Tree Terminology

- **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).



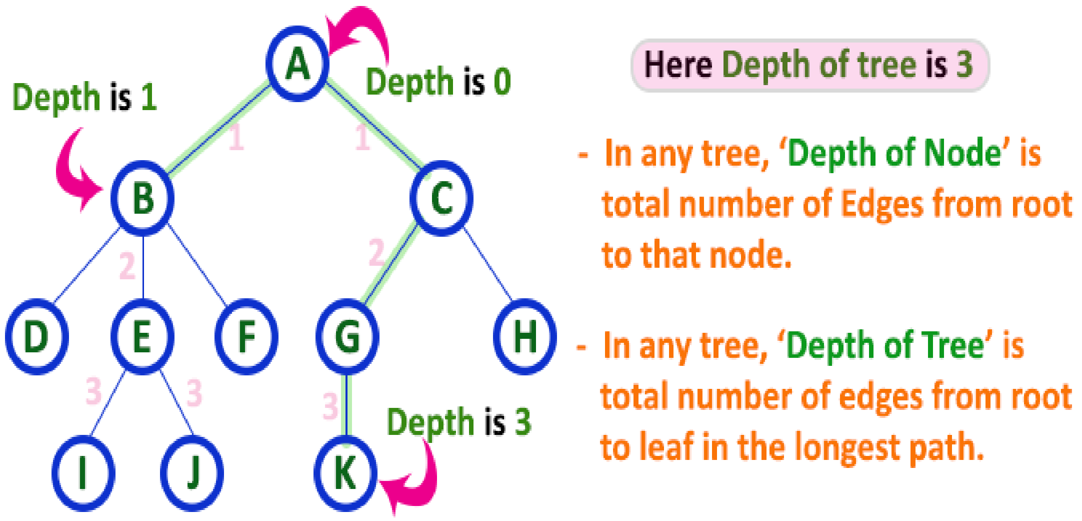
Tree Terminology

- **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'.**


Height is 2
Height is 3
Height is 0
A
B
C
D
E
F
G
H
I
J
K
3
3
2
2
1
1
1
Here Height of tree is 3
- In any tree, 'Height of Node' is total number of Edges from leaf to that node in longest path.
- In any tree, 'Height of Tree' is the height of the root node.

Tree Terminology

- **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 a tree, **depth of the root node is '0'.**

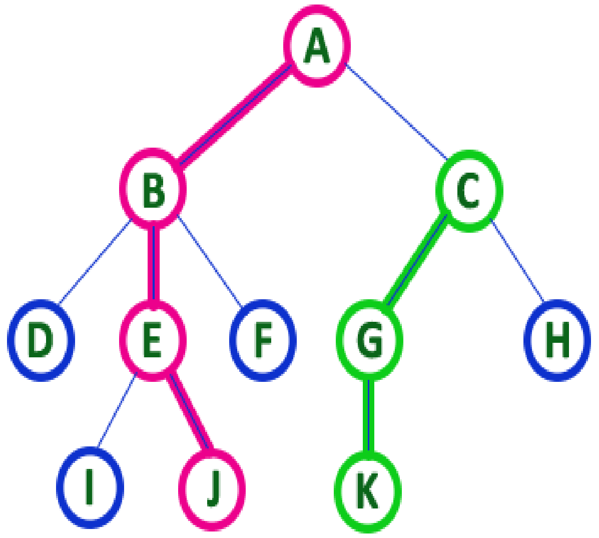


Here Depth of tree is 3

- In any tree, 'Depth of Node' is total number of Edges from root to that node.
- In any tree, 'Depth of Tree' is total number of edges from root to leaf in the longest path.

Tree Terminology

- **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**.



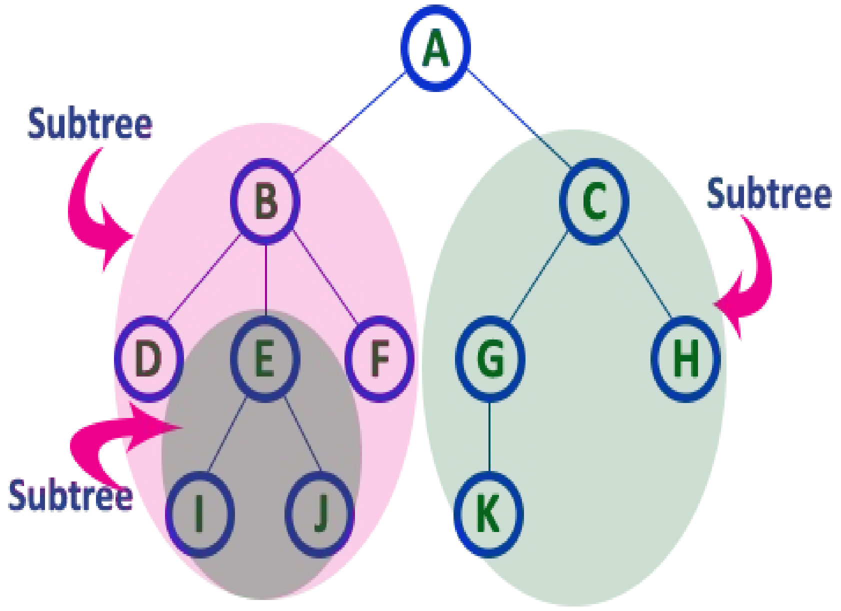
- In any tree, 'Path' is a sequence of nodes and edges between two nodes.

Here, 'Path' between A & J is
A - B - E - J

Here, 'Path' between C & K is
C - G - K

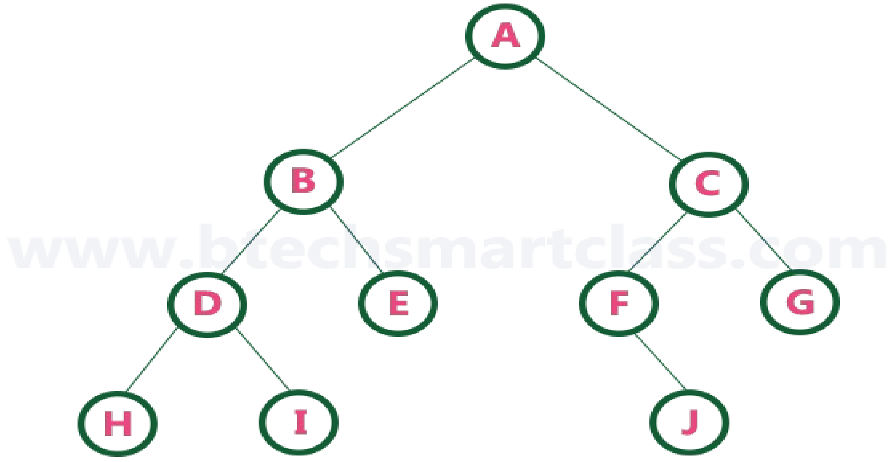
Tree Terminology

- **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.



Binary Tree (CO3)

- In a normal tree, every node can have any number of children.
- A 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 a 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 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**



NIET Greater Noida GET FUTURE READY

Binary Tree (CO3)

Properties of Binary Tree

1. At each level of i, the maximum number of nodes is 2^i.
2. The height of the tree is defined as the longest path from the root node to the leaf node. The tree which is shown above has a height equal to 3. Therefore, the maximum number of nodes at height 3 is equal to (1+2+4+8) = 15. In general, the maximum number of nodes possible at height h is $(2^0 + 2^1 + 2^2 + \dots 2^h) = 2^{h+1} - 1$.
3. The minimum number of nodes possible at height h is equal to **h+1**.
4. If the number of nodes is minimum, then the height of the tree would be maximum. Conversely, if the number of nodes is maximum, then the height of the tree would be minimum.

Binary Tree (CO3)

Types of Binary Tree

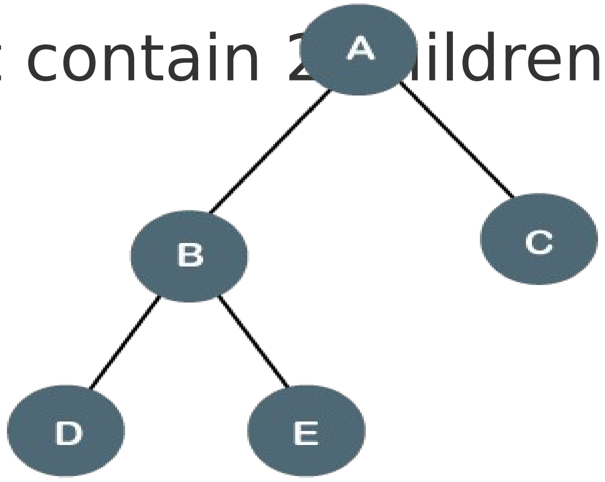
There are four types of Binary tree:

1. Full/ proper/ strict Binary tree
2. Complete Binary tree
3. Perfect Binary tree
4. Degenerate Binary tree

Binary Tree (CO3)

Full/ proper/ strict Binary tree

The full binary tree is also known as a strict binary tree. The tree can only be considered as the full binary tree if each node must contain either 0 or 2 children. The full binary tree can also be defined as the tree in which each node must contain 2 children except the leaf nodes.



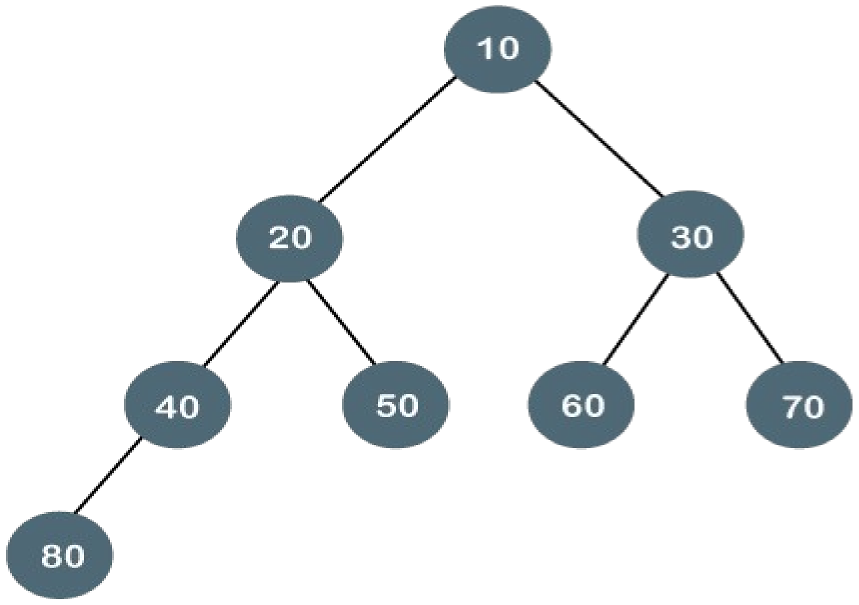
Binary Tree (CO3)

Properties of Full Binary Tree

1. The number of leaf nodes is equal to the number of internal nodes plus 1. In the above example, the number of internal nodes is 5; therefore, the number of leaf nodes is equal to 6.
2. The maximum number of nodes is the same as the number of nodes in the binary tree, i.e., 2^{h+1} -1.
3. The minimum number of nodes in the full binary tree is 2*h+1.
4. The minimum height of the full binary tree is **$\log_2(n+1)$ - 1.**
5. The maximum height of the full binary tree can be

Binary Tree

- **Complete Binary Tree**
  - A complete binary tree is a binary tree in which all the levels are completely filled except possibly the lowest one, which is filled from the left.



Binary Tree

Properties of Complete Binary Tree

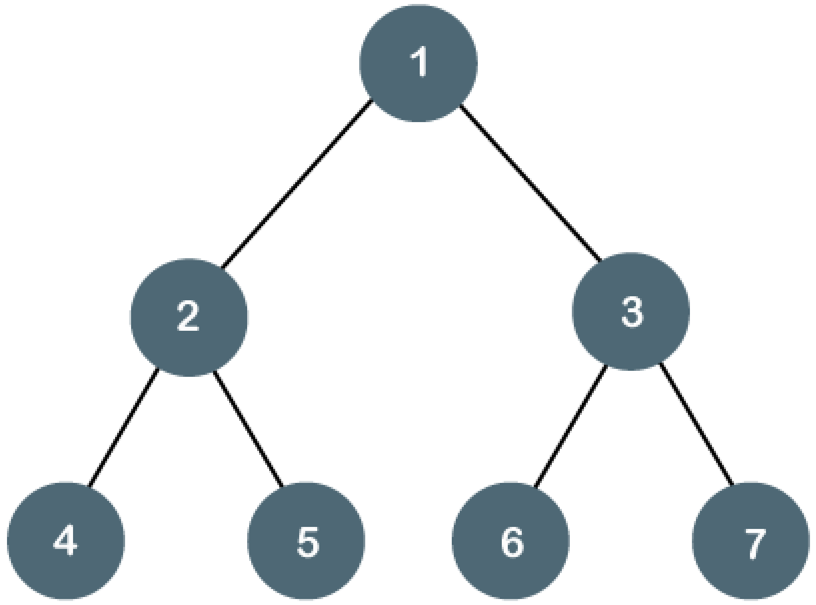
- The maximum number of nodes in complete binary tree is $2^{h+1} - 1$.
- The minimum number of nodes in complete binary tree is 2^h.
- The minimum height of a complete binary tree is $\mathbf{\log_2(n+1) - 1}$**.**
- The maximum height of a complete binary tree is **logn.**

NIET Greater Noida GET FUTURE READY

Binary Tree

Perfect Binary Tree

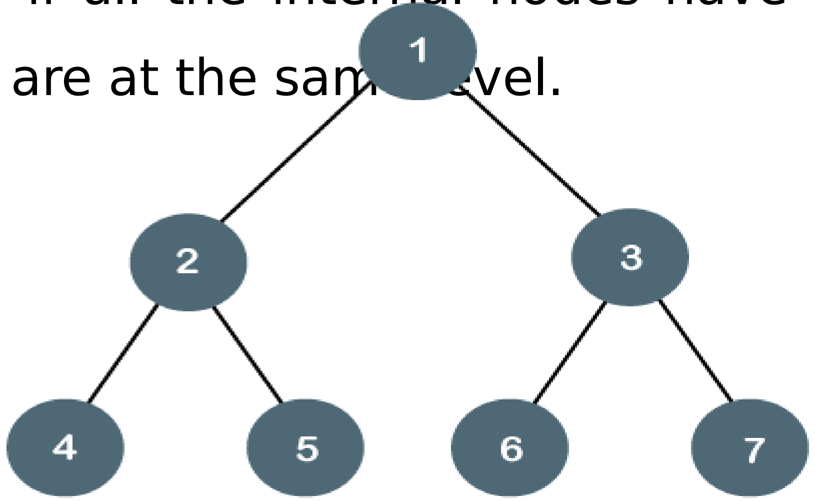
- A tree is a perfect binary tree if all the internal nodes have 2 children, and all the leaf nodes are at the same level.



Binary Tree

Perfect Binary Tree

- A tree is a perfect binary tree if all the internal nodes have 2 children, and all the leaf nodes are at the same level.

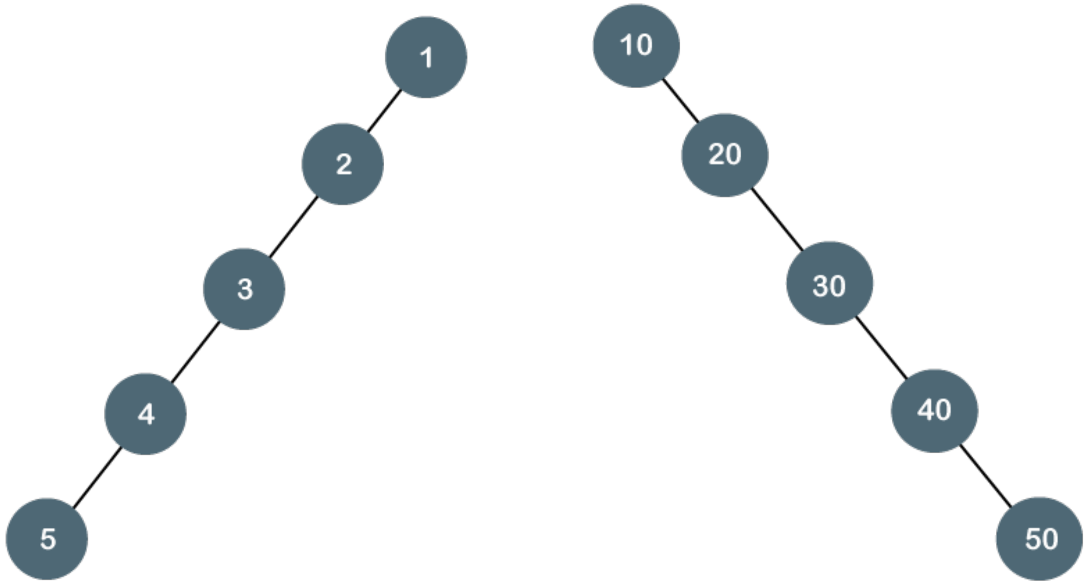


Note: All the perfect binary trees are the complete binary trees as well as the full binary tree, but vice versa is not true, i.e., all complete binary trees and full binary trees are the perfect binary trees.

Binary Tree

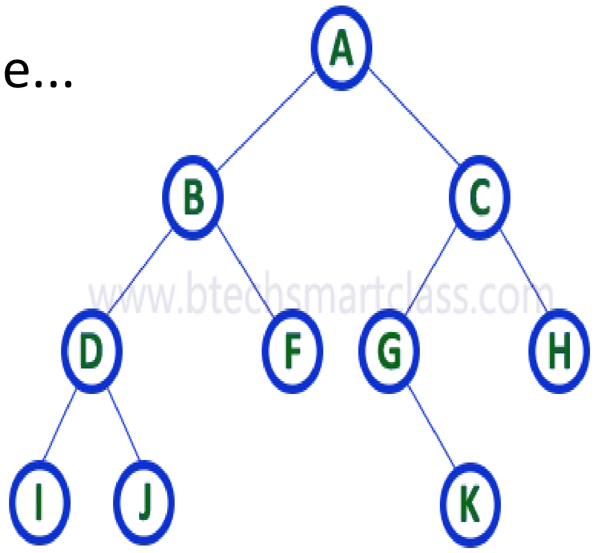
Degenerate Binary Tree

- The degenerate binary tree is a tree in which all the internal nodes have only one children.



Binary Tree Representation (CO3)

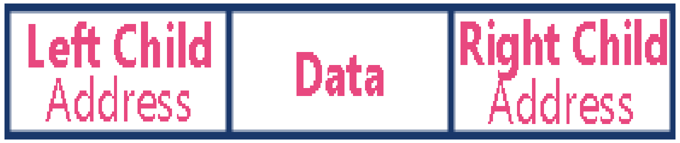
- A binary tree data structure is represented using two methods. Those methods are as follows...
  - **Array Representation**
  - **Linked List Representation**
- Consider the following binary tree...



Binary Tree Representation

Linked List Representation of Binary Tree

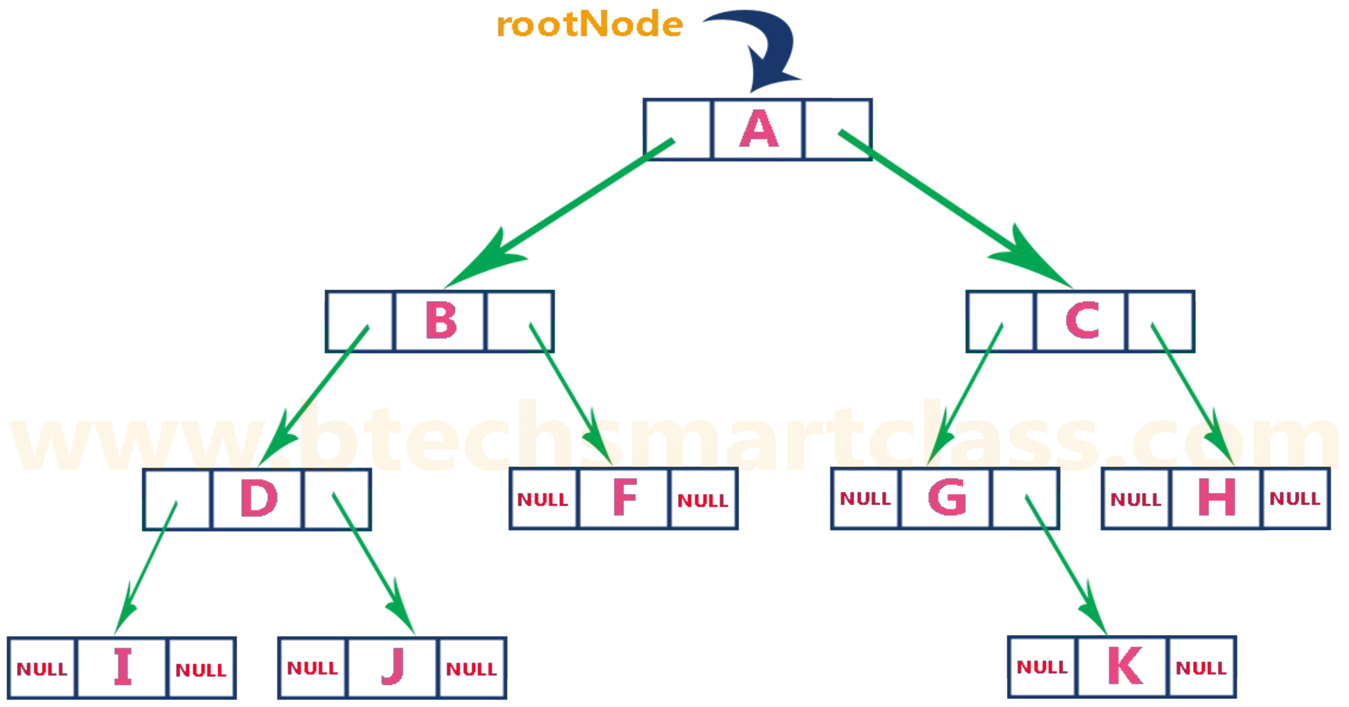
- We use a double linked list to represent a binary tree.
- In a double linked list, every node consists of three fields.
- First field for storing left child address, second for storing actual data and third for storing right child address. In this linked list representation, a node has the following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



Binary Tree Representation

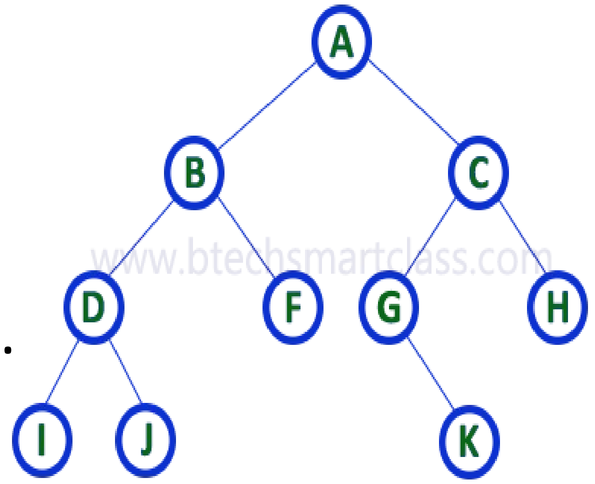
Linked List Representation of Binary Tree

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



Binary Tree Traversal (CO3)

- 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...



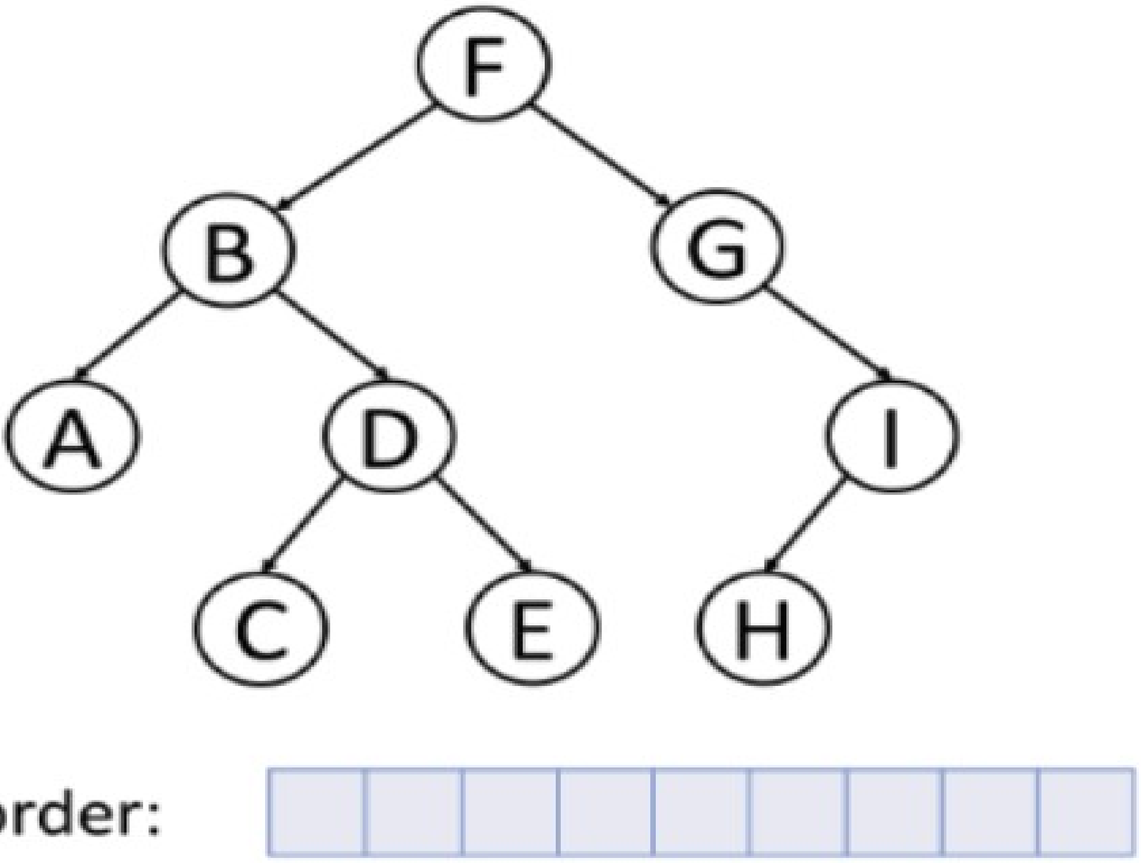
Binary Tree Traversal

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 sub trees in the tree.
- This is performed recursively for all nodes in the tree
- In-Order Traversal for above example of binary tree is

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

Binary Tree Traversal

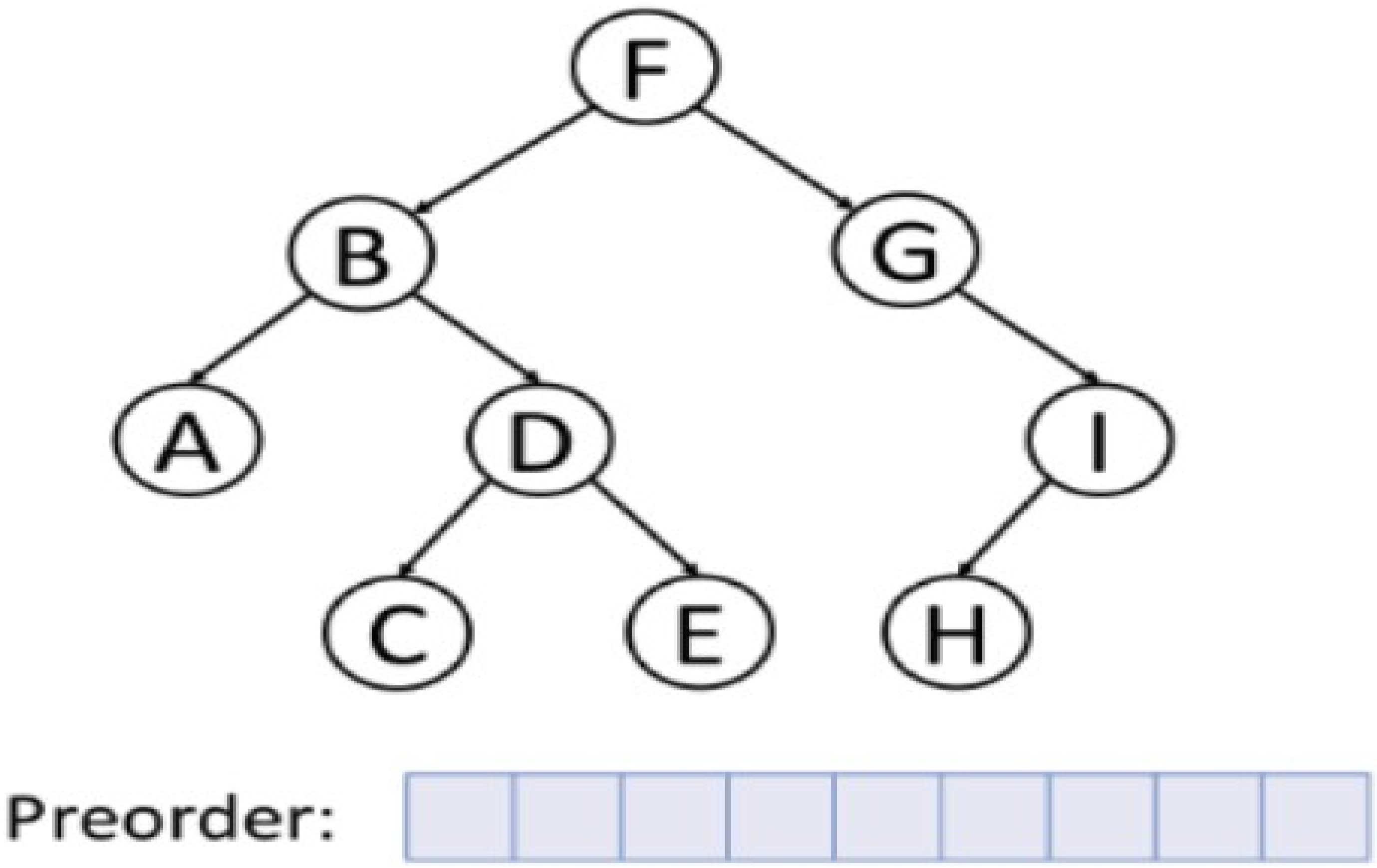


Binary Tree Traversal

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 sub trees in the tree.
- Pre-Order Traversal for above example binary tree is A - B - D - I - J - F - C - G - K - H

Binary Tree Traversal

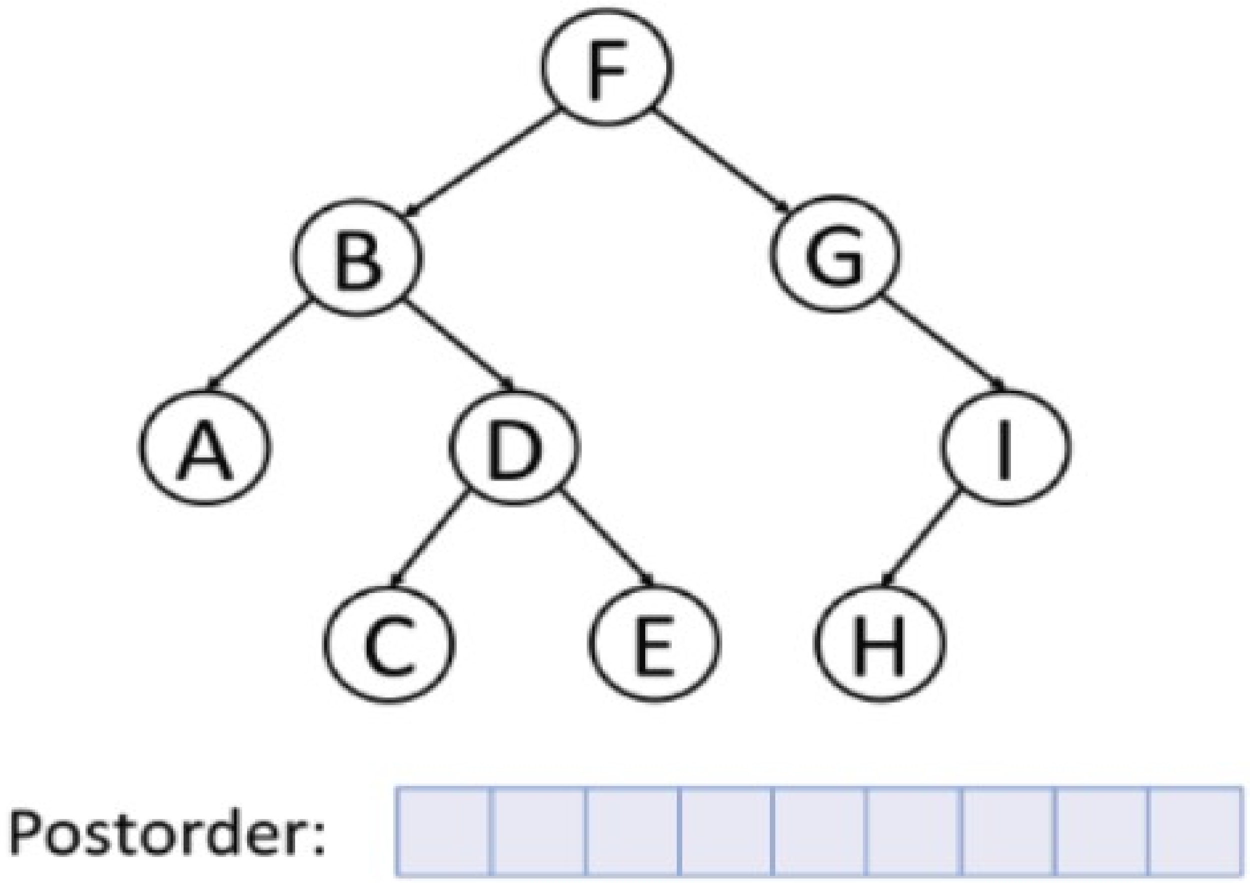


Binary Tree Traversal

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.
- Post-Order Traversal for above example binary tree is I - J - D - F - B - K - G - H - C - A

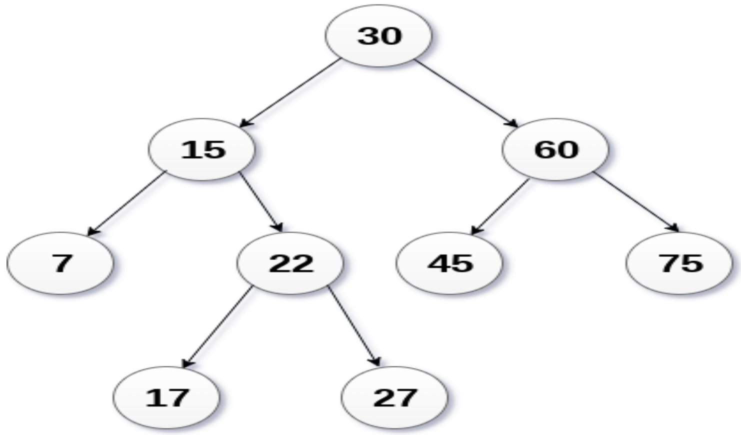
Binary Tree Traversal



Binary Search Tree (CO3)

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.



Binary Search Tree

Advantages of using binary search tree

- Searching become very efficient in a binary search tree since, we get a hint at each step, about which sub-tree contains the desired element.
- The binary search tree is considered as efficient data structure in compare to arrays and linked lists.
- In searching process, it removes half sub-tree at every step. Searching for an element in a binary search tree takes o(log$_2$n) time. In worst case, the time it takes to search an element is 0(n).
- It also speed up the insertion and deletion operations as compare to that in array and linked list.

Binary Search Tree

Basic operations on a BST

- Create: creates an empty tree.
- Insert: insert a node in the tree.
- Search: Searches for a node in the tree.
- Delete: deletes a node from the tree.
- Inorder: in-order traversal of the tree.
- Preorder: pre-order traversal of the tree.
- Postorder: post-order traversal of the tree.

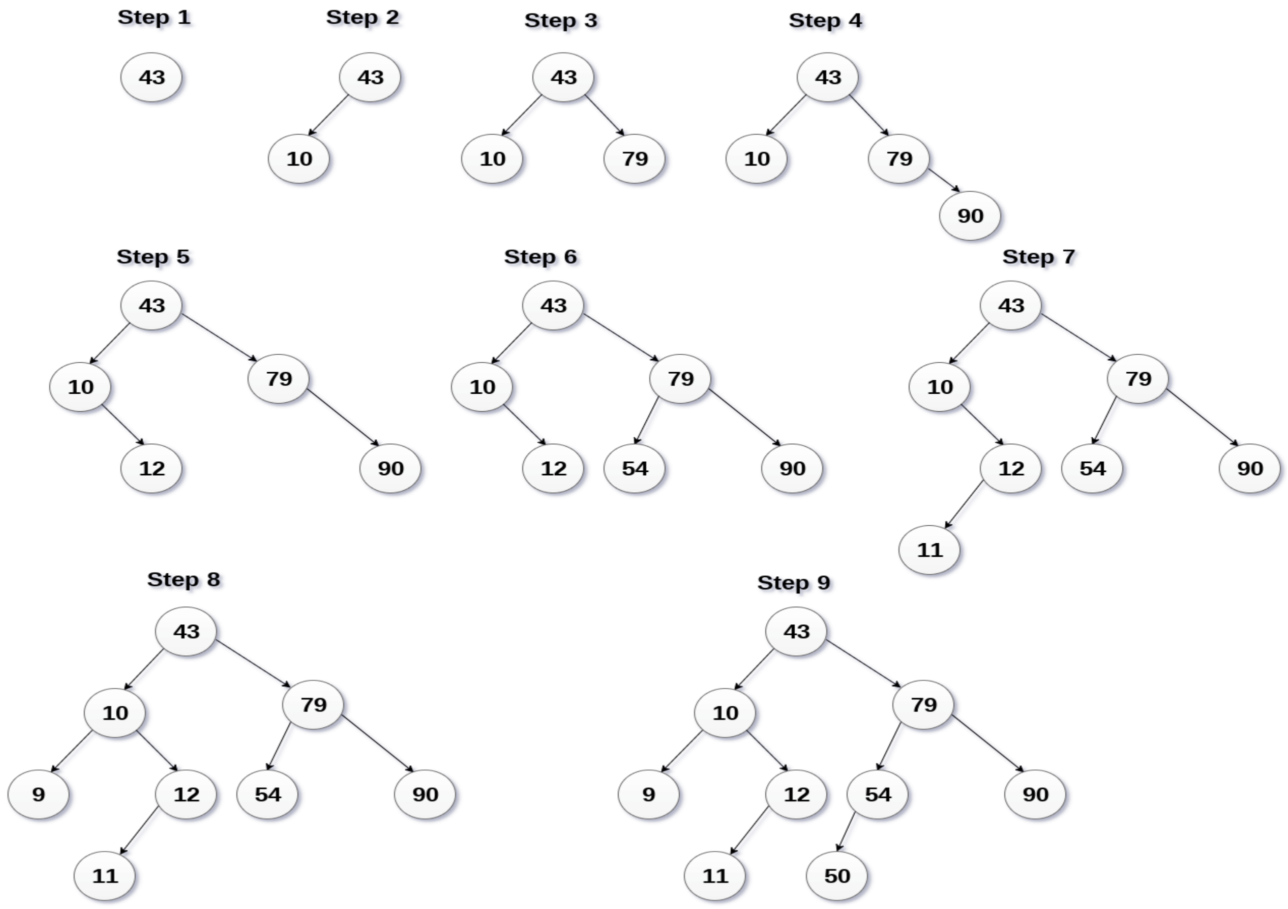
Binary Search Tree

- **Creation or Insertion of binary search tree**
  - Create the binary search tree using the following data elements.

    43, 10, 79, 90, 12, 54, 11, 9, 50
  - Insert 43 into the tree as the root of the tree.
  - Read the next element, if it is lesser than the root node element, insert it as the root of the left sub-tree.
  - Otherwise, insert it as the root of the right of the right sub-tree.

Binary Search Tree

- **Creation or Insertion of binary search tree**



Binary search Tree Creation

Binary Search Tree

Searching in binary search tree

We use the following procedure to search for a node with a given key in a binary search tree. Given a pointer to the root of the tree and a key *k,* TREE-SEARCH returns a pointer to a node with key *k* if one exists; otherwise, it returns NIL.

1. Start from the root.
2. Compare the searching element with root, if less than root, then recurse for left, else recurse for right.
3. If the element to search is found anywhere, return true, else return false.

Binary Search Tree

Deletion Operation

There are three cases for deleting a node from a binary search tree.

Case I : In the first case, the node to be deleted is the leaf node. In such a case, simply delete the node from the tree.

Case II : In the second case, the node to be deleted lies has a single child node

1. Replace that node with its child node.
2. Remove the child node from its original position.

Case III : In the third case, the node to be deleted has two children.

3. Get the inorder successor of that node.
4. Replace the node with the inorder successor.
5. Remove the inorder successor from its original position.

Threaded Binary Tree (CO3)

- 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.

Threaded Binary Tree (CO3)

Why Do We Need Threads?

Adding threads to a binary tree increase complexity, so why do we need threads?

There are few reasons:

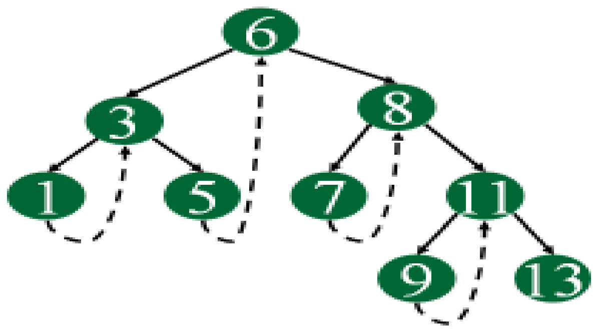
- Fast successor or predecessor access
- No auxiliary stack or recursion approach for certain traversals
- Reduced the memory consumption when performing traversals since no auxiliary stack or recursion are required
- Utilize wasted space. Since the empty left or right attribute of a node does not store anything, we can use them as threads

Threaded Binary Tree (CO3)

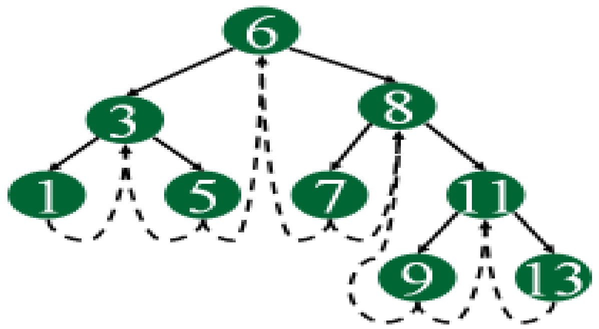
There are two types of threaded binary trees.

Single Threaded: Where a NULL right pointers is made to point to the inorder successor (if successor exists)

Double Threaded: Where both left and right NULL pointers are made to point to inorder predecessor and inorder successor respectively.



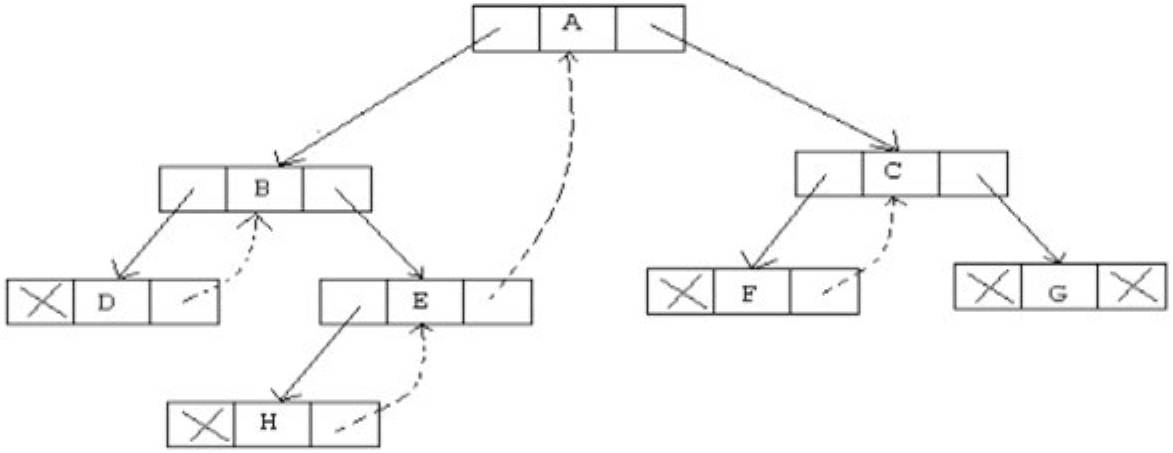
Single Threaded Binary Tree



Double Threaded Binary Tree

Threaded Binary Tree (CO3)

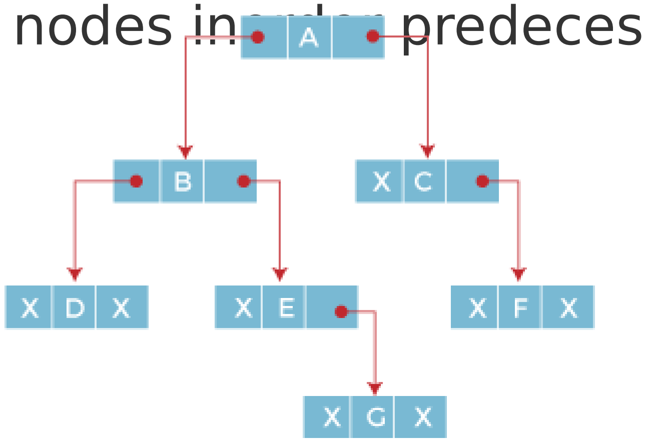
In **one-way threaded** binary trees, a thread will appear either in the right or left link field of a node. If it appears in the right link field of a node then it will point to the next node that will appear on performing in order traversal. Such trees are called **Right threaded binary trees**.



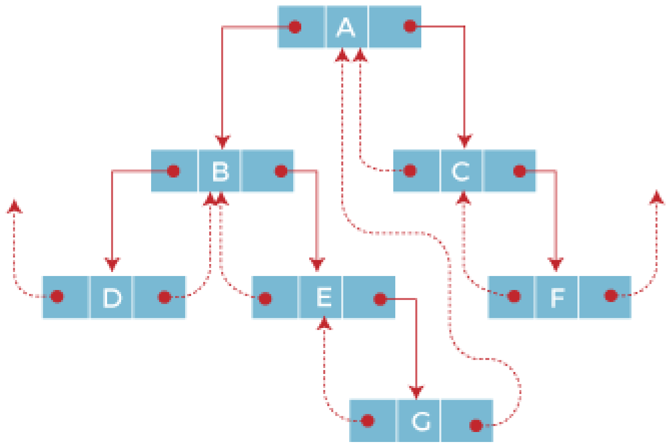
RIGHT THREADED BINARY TREE

Threaded Binary Tree (CO3)

In **two-way threaded** Binary trees, the right link field of a node containing NULL values is replaced by a thread that points to nodes inorder successor and left field of a node containing NULL values is replaced by a thread that points to nodes inorder predecessor.



A binary tree (Inorder traversal - D, B, E, G, A, C, F)

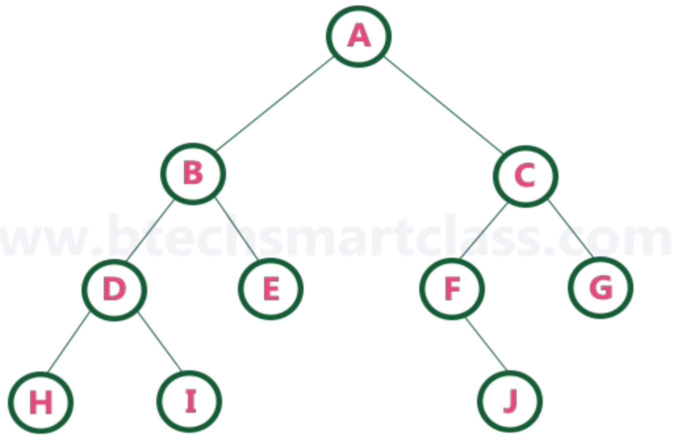


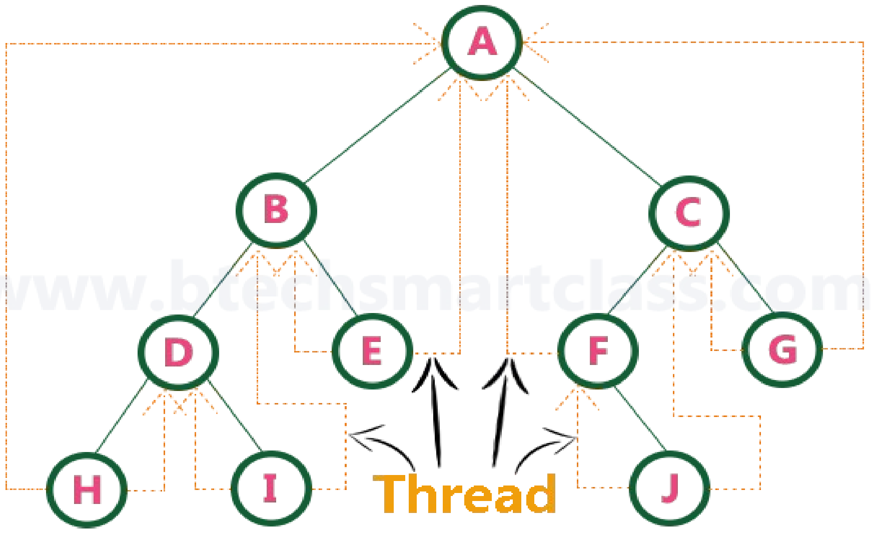
A two - way threaded binary tree

Threaded Binary Tree (CO3)

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**





AVL Tree

- AVL Tree can be defined as height balanced binary search tree in which each node is associated with a balance factor which is calculated by subtracting the height of its right sub-tree from that of its left sub-tree.
- Tree is said to be balanced if balance factor of each node is in between -1 to 1, otherwise, the tree will be unbalanced and need to be balanced.

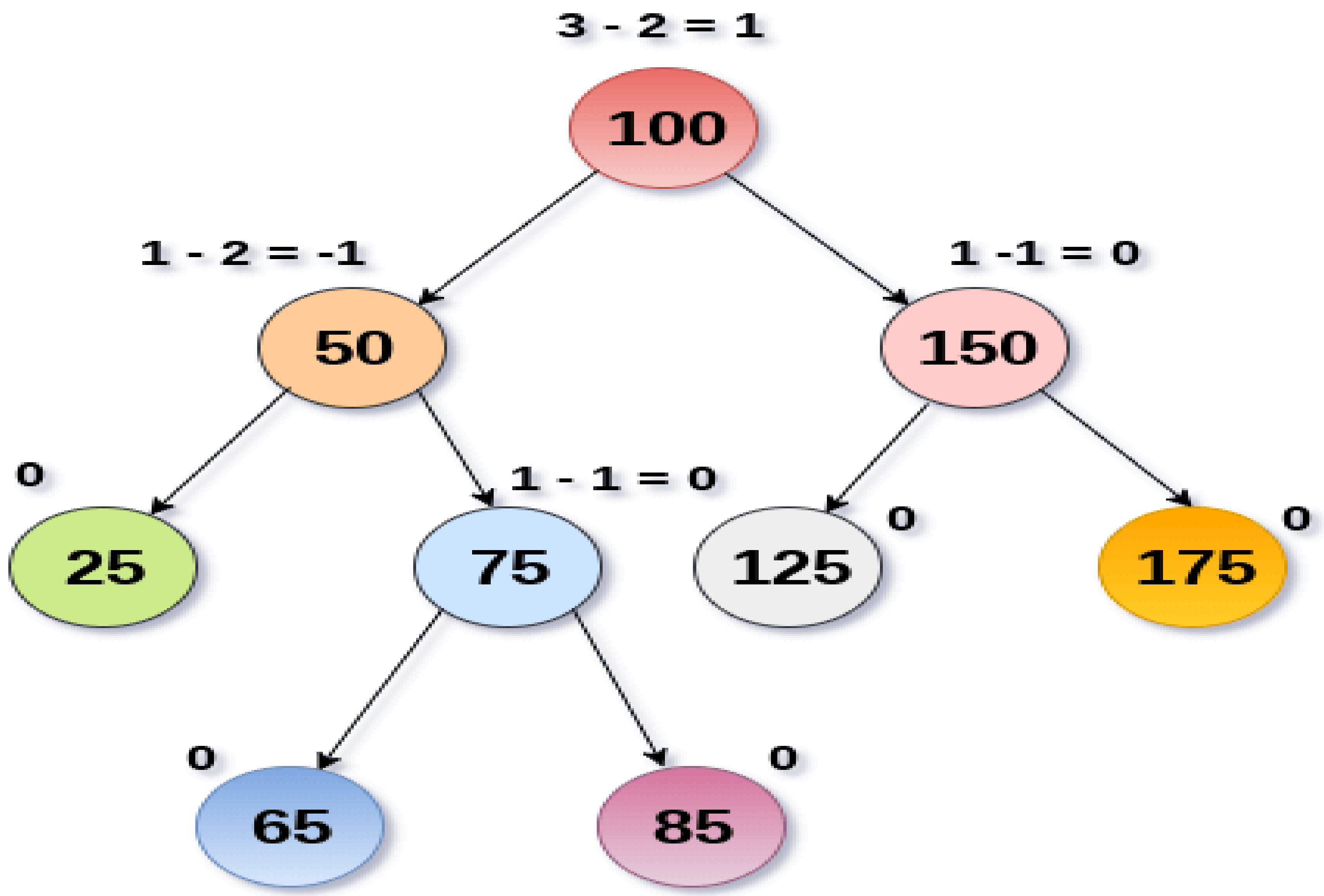
AVL Tree

Balance Factor:-

Balance Factor (k) = height (left(k)) - height (right(k))

- If balance factor of any node is 1, it means that the left sub-tree is one level higher than the right sub-tree.
- If balance factor of any node is 0, it means that the left sub-tree and right sub-tree contain equal height.
- If balance factor of any node is -1, it means that the left sub-tree is one level lower than the right sub-tree

AVL Tree



AVL Tree

AVL Tree(Operations)

SN	Operation	Description
1	Insertion	Insertion in AVL tree is performed in the same way as it is performed in a binary search tree. However, it may lead to violation in the AVL tree property and therefore the tree may need balancing. The tree can be balanced by applying rotations.
2	Deletion	Deletion can also be performed in the same way as it is performed in a binary search tree. Deletion may also disturb the balance of the tree therefore, various types of rotations are used to rebalance the tree.

AVL Tree(Operations)

AVL Rotations

We perform rotation in AVL tree only in case if Balance Factor is other than **-1, 0, and 1**. There are basically four types of rotations which are as follows:

1. L L rotation:

   Inserted node is in the left subtree of left subtree of A.

2. R R rotation :

   Inserted node is in the right subtree of right subtree of A.

3. L R rotation :

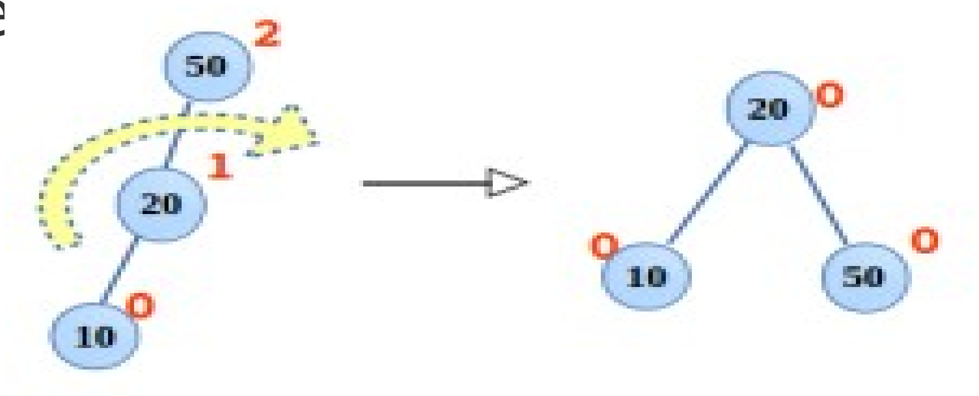
   Inserted node is in the right subtree of left subtree of A.

AVL Tree(Operations)

LL Rotation

When BST becomes unbalanced, due to a node is inserted into the left subtree of the left subtree of C, then we perform LL rotation, LL rotation is clockwise rotation, which is applied on the e

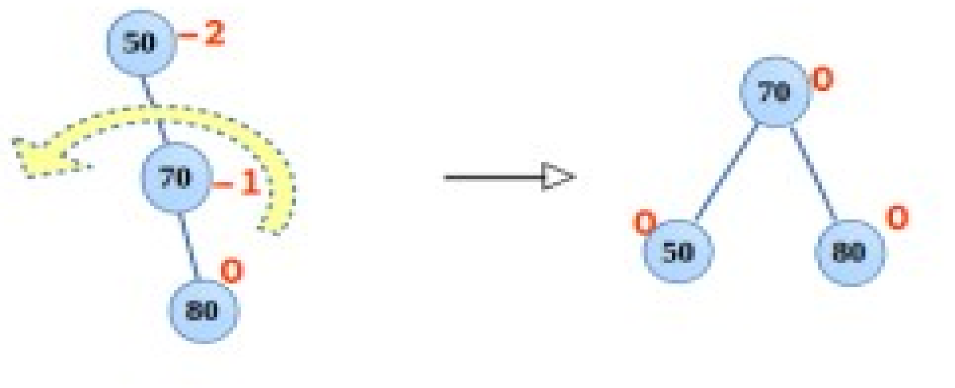
2.



AVL Tree(Operations)

RR Rotation

When BST becomes unbalanced, due to a node is inserted into the right subtree of the right subtree of A, then we perform RR rotation, RR rotation is an anticlockwise rotation, which is applied on the edge below a node having balanc

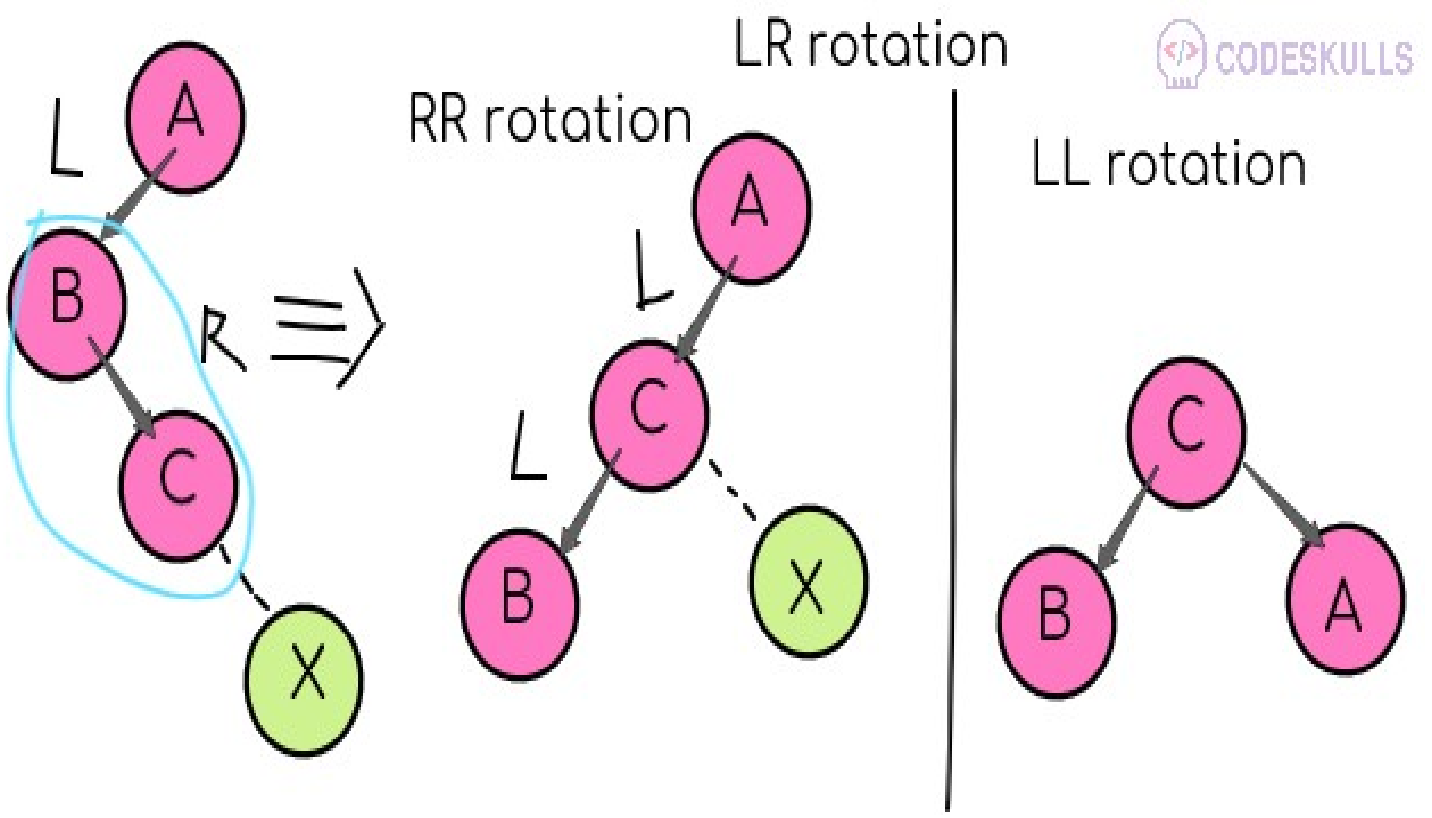


AVL Tree(Operations)

LR Rotation

LR rotation = RR rotation + LL rotation, i.e., first RR rotation is performed on subtree and then LL rotation is performed on full tree, by full tree we mean the first node from the path of inserted node whose balance factor is other than -1, 0, or 1.

AVL Tree(Operations)



AVL Tree(Operations)

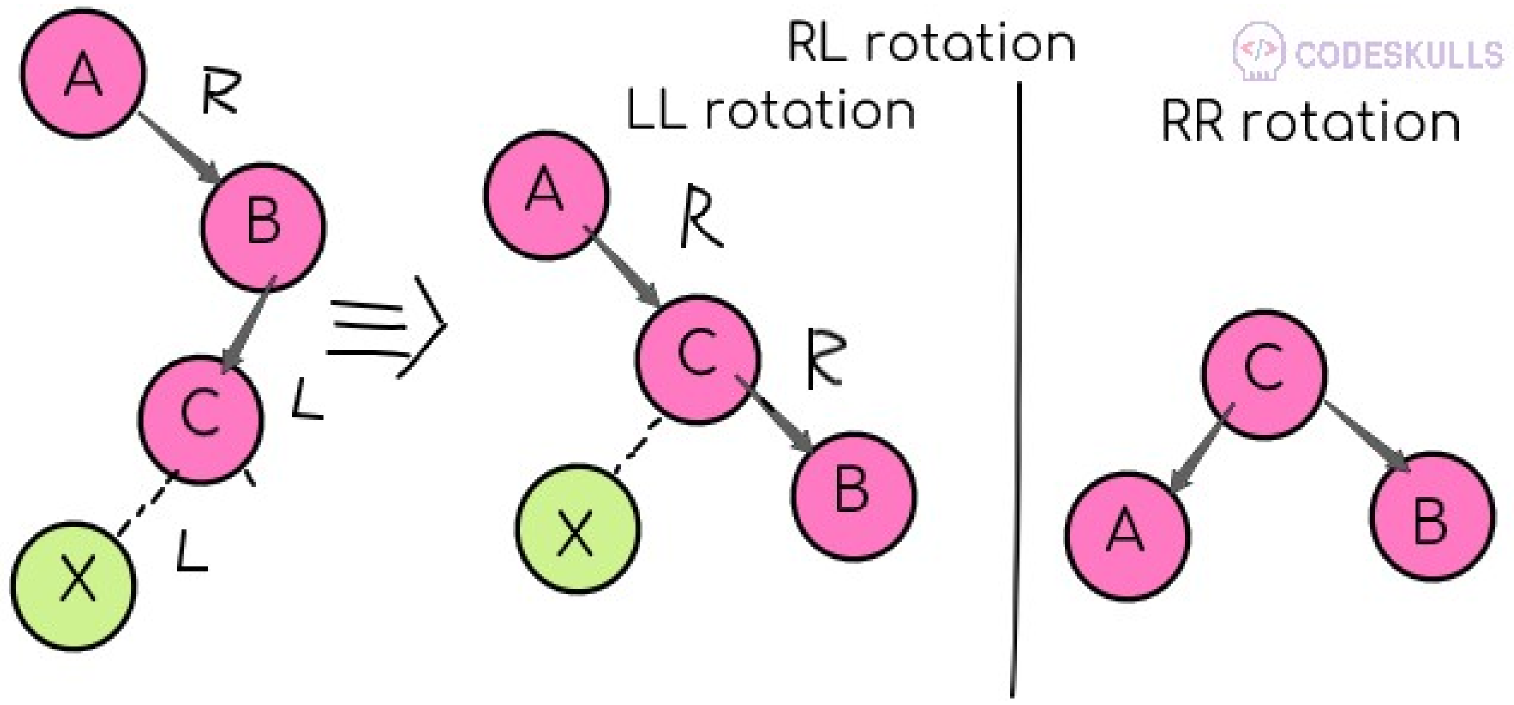
RL Rotation

R L rotation = LL rotation + RR rotation, i.e., first LL rotation is performed on subtree and then RR rotation is performed on full tree, by full tree we mean the first node from the path of inserted node whose balance factor is other than -1, 0, or 1.

B- Trees

B-tree is a special type of self-balancing search tree in which each node can contain more than one key and can have more than two children. It is a generalized form of the binary search tree. It is also known as a height-balanced m-way tree.

AVL Tree(Operations)



RL rotation
CODESKULLS
A
R
B
C
L
X
L
LL rotation
A
R
C
R
X
B
RR rotation
C
A
B