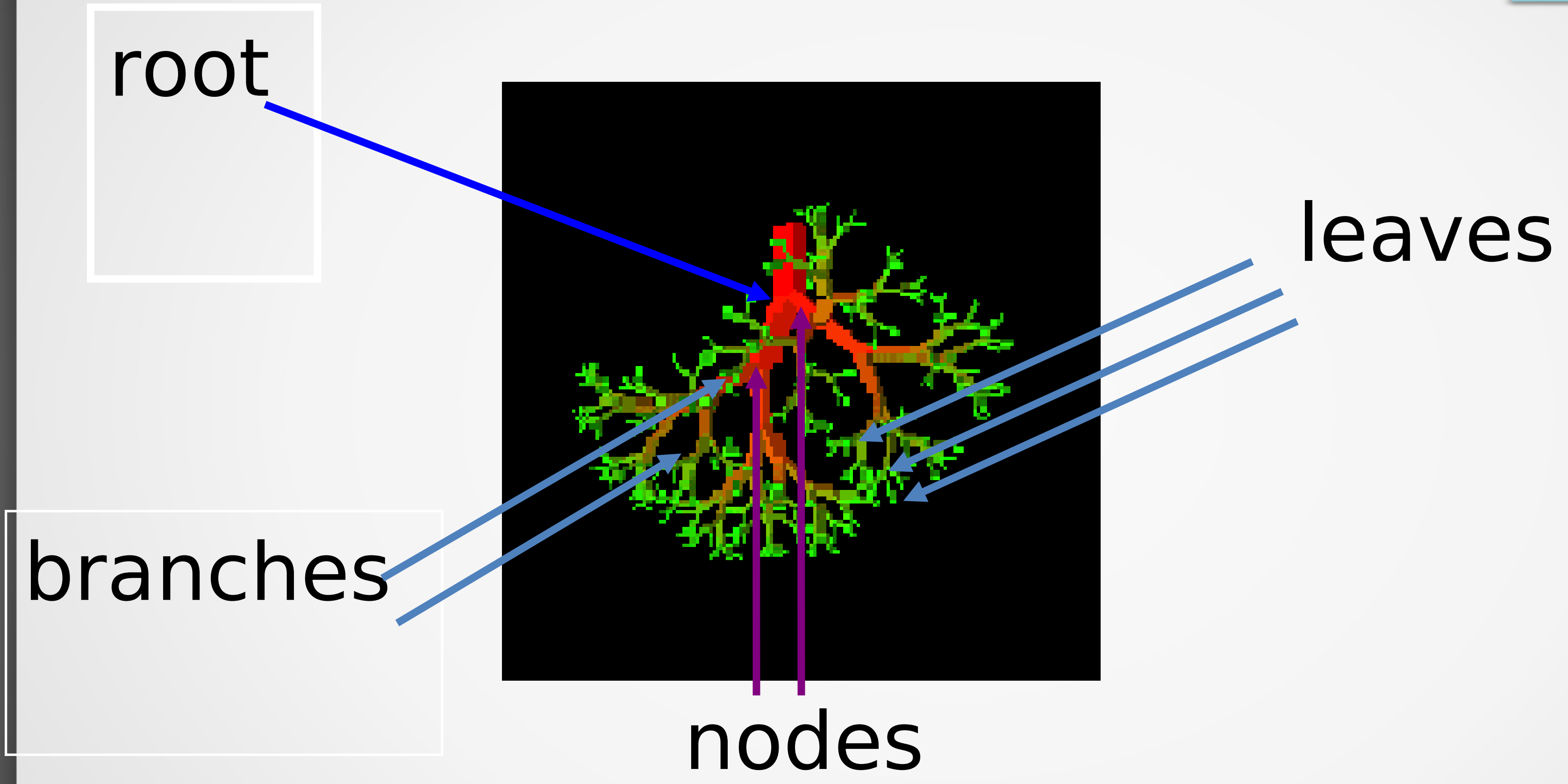
GLS UNIVERSITY

0301302 DATA STRUCTURES.
UNIT– IV

INTRODUCTION TO DATA STRUCTURE

- Tree is a non linear data structure.
- A tree is a collection of nodes connected by directed (or undirected) edges.
- A tree data structure can be defined recursively (locally) as a collection of nodes (starting at a root node), where each node is a data structure consisting of a value, together with a list of references to nodes (the "children"), with the constraints that no reference is duplicated, and none points to the root.
- A tree can be empty with no nodes or a tree is a structure consisting of one node called the **root** and zero or one or more sub trees.
- It is an abstract model of a hierarchical structure.


root
leaves
branches
nodes

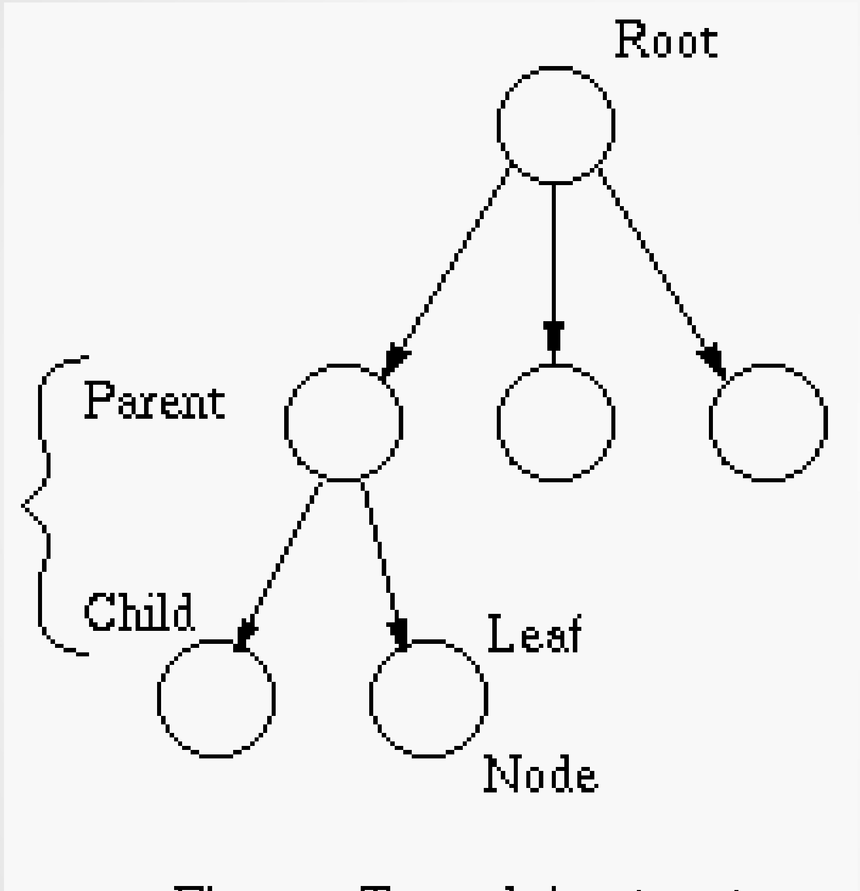
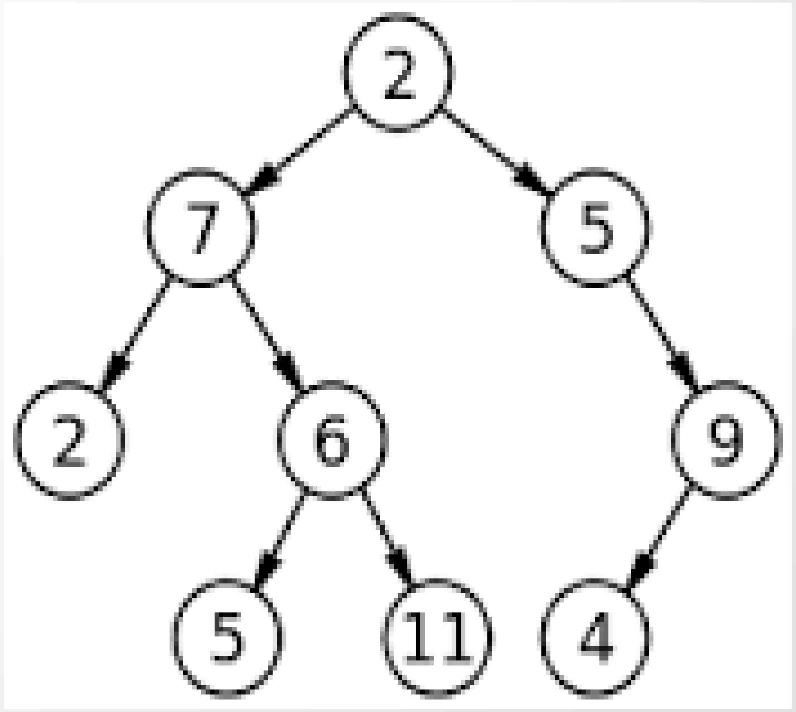


Figure: Tree data structure

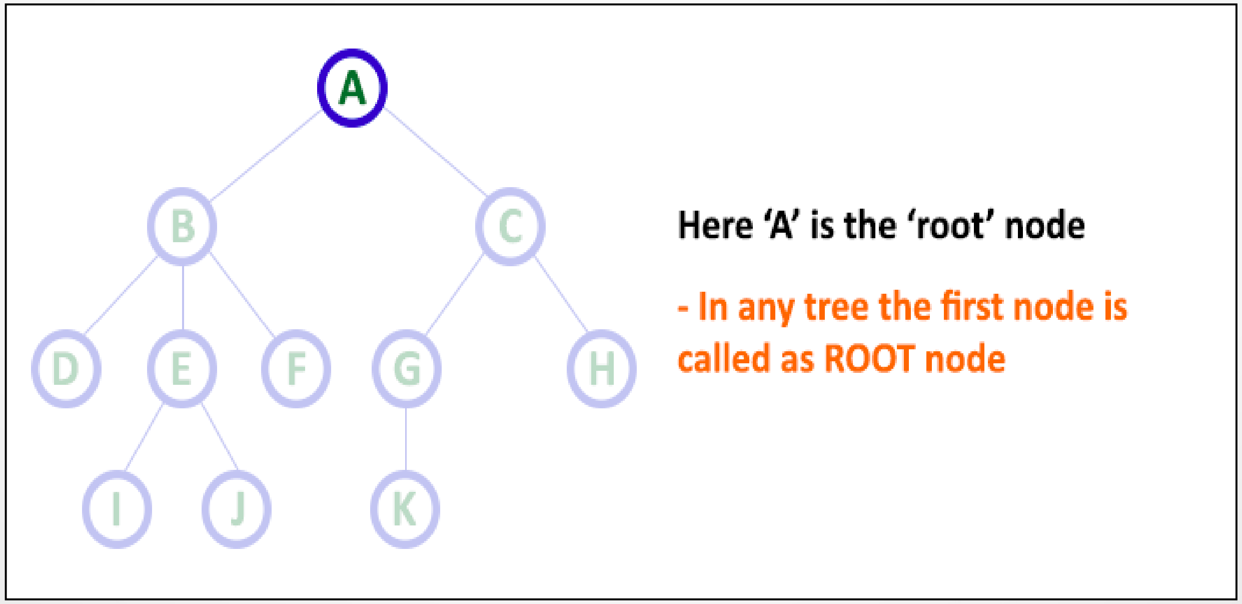


Applications of Tree

- Organizational Charts
- File Systems
- Programming Environments

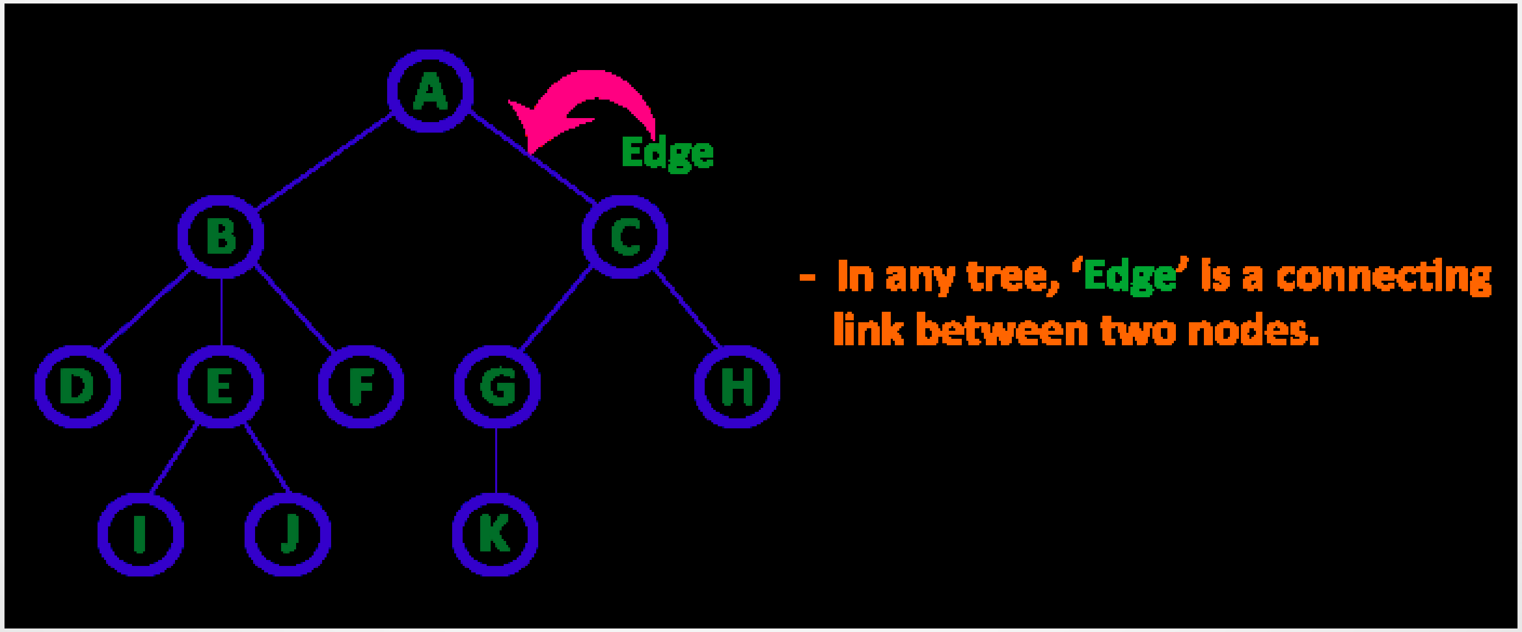
TREE TERMINOLOGY

- **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.


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

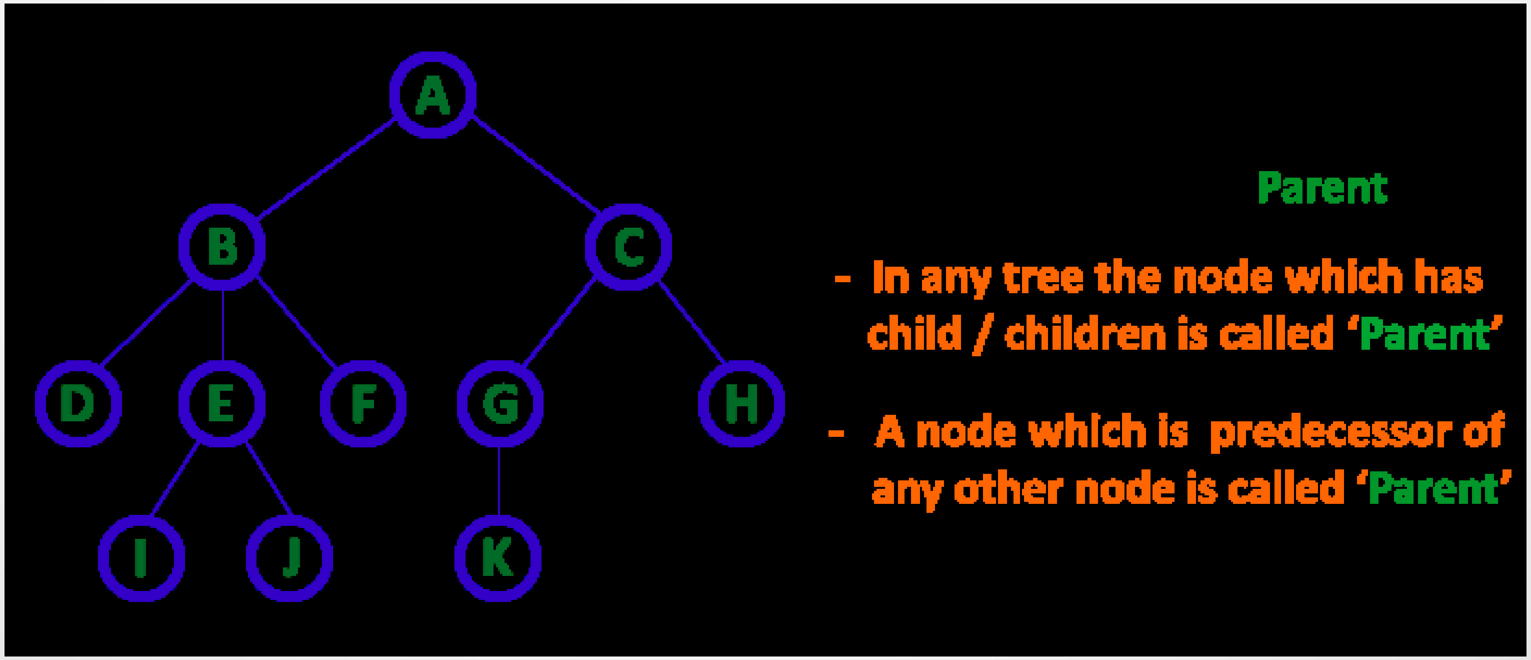
TREE TERMINOLOGY

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

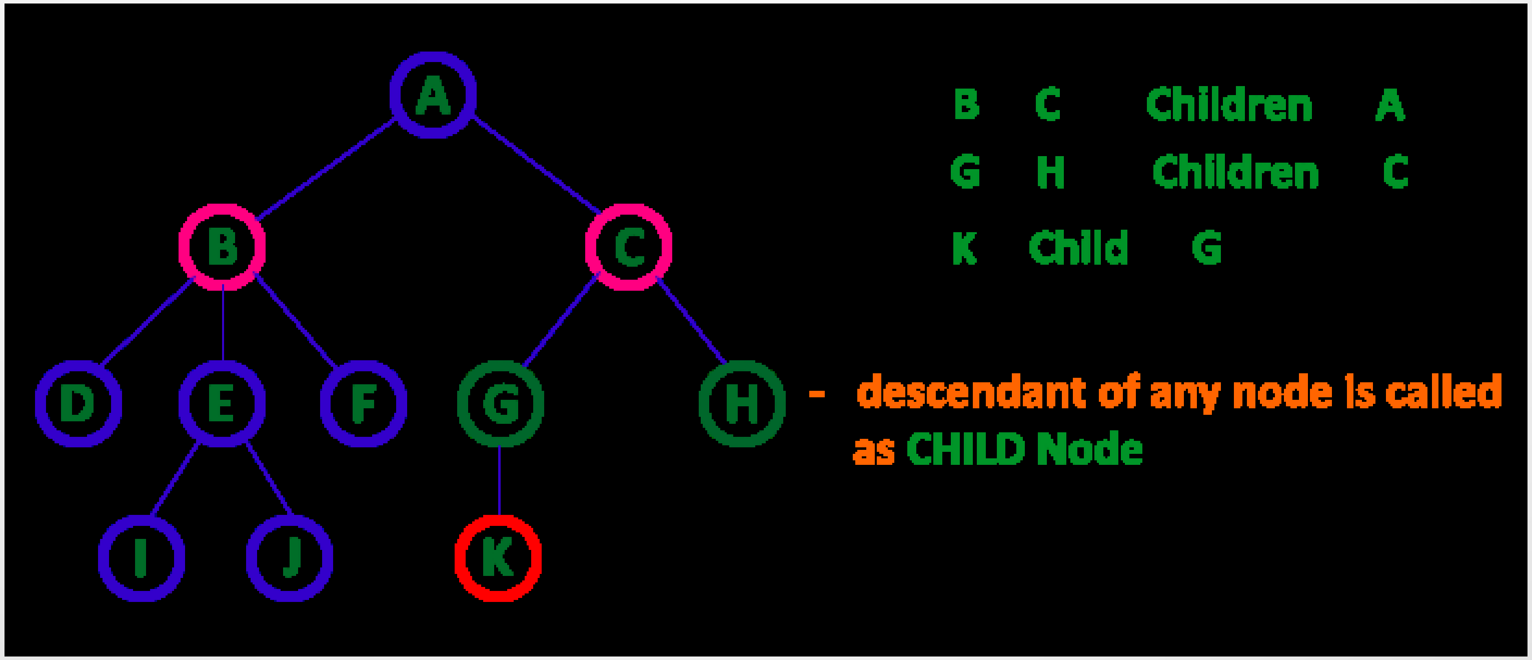
TREE TERMINOLOGY

- **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".


A
B
C
D
E
F
G
H
I
J
K
Parent
- In any tree the node which has child / children is called 'Parent'
- A node which is predecessor of any other node is called 'Parent'

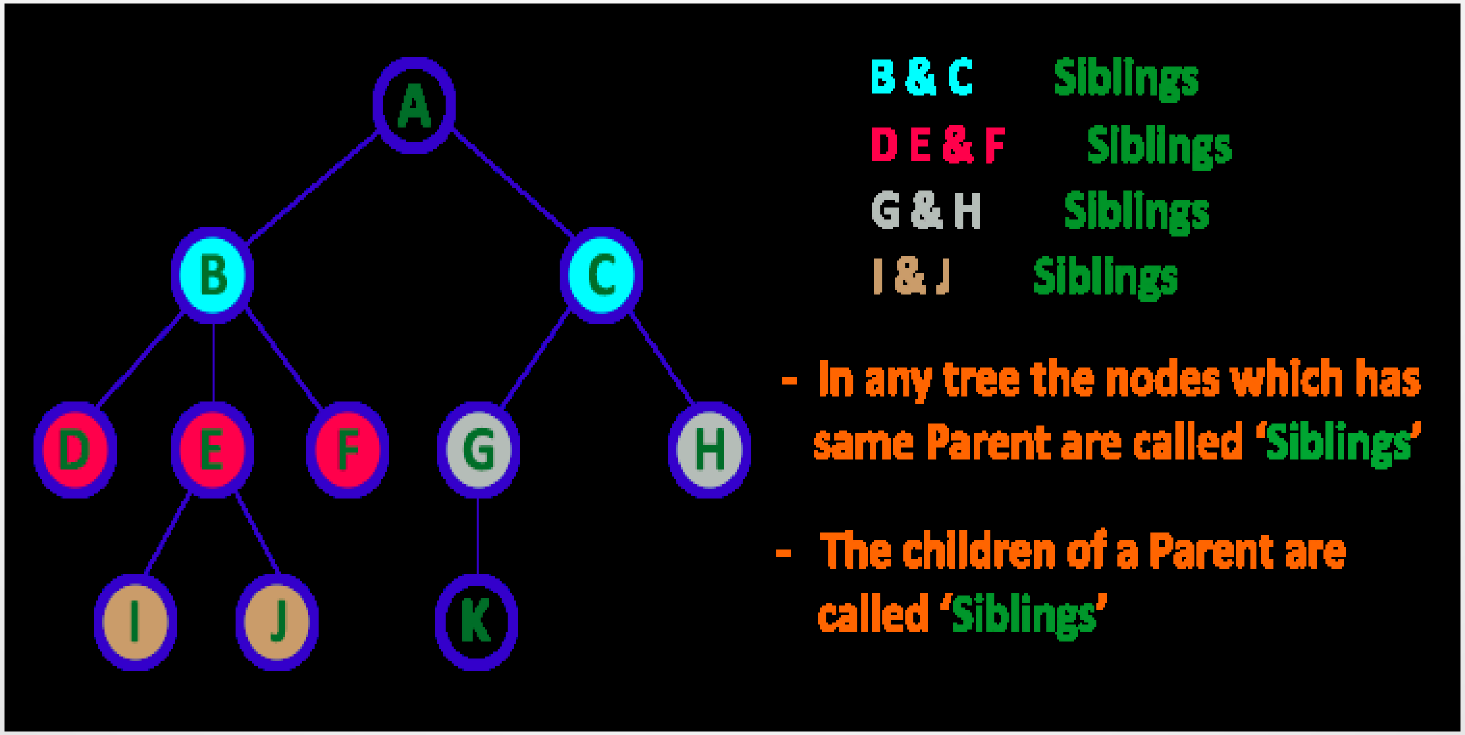
TREE TERMINOLOGY

- **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.



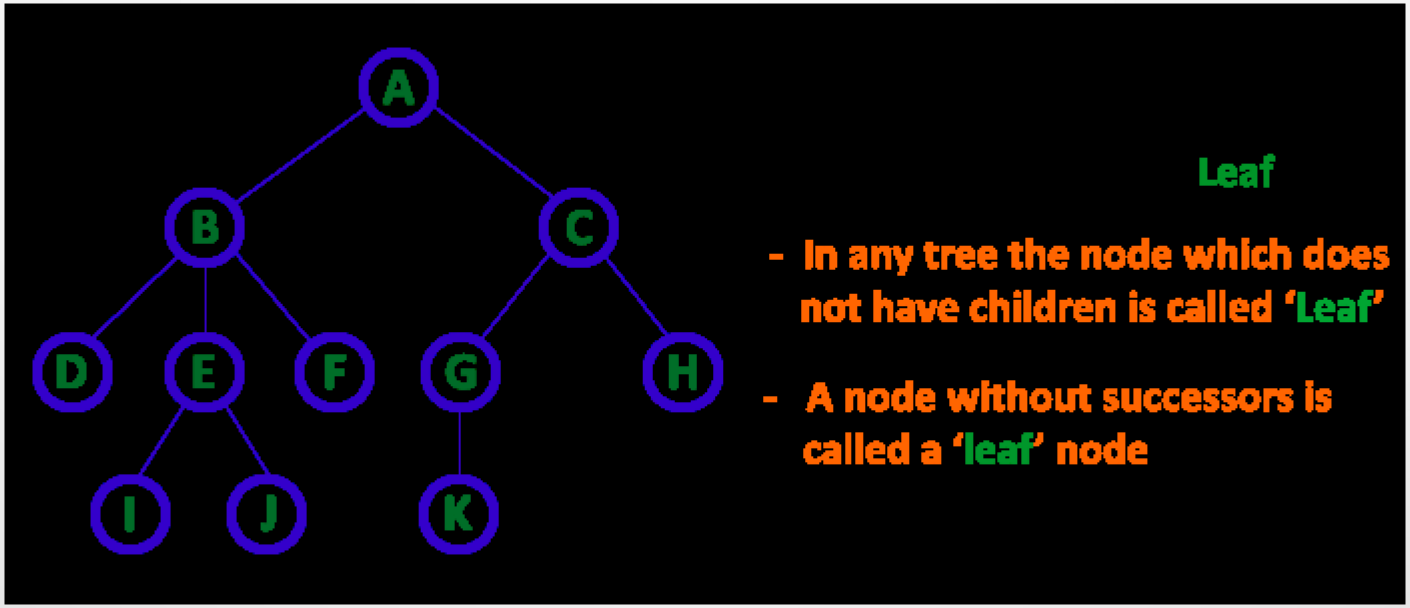
TREE TERMINOLOGY

- **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.


A
B
C
D
E
F
G
H
I
J
K
B & C Siblings
D E & F Siblings
G & H Siblings
I & J 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. External node is also a node with no child. In a tree, leaf node is also called as 'Terminal' node.

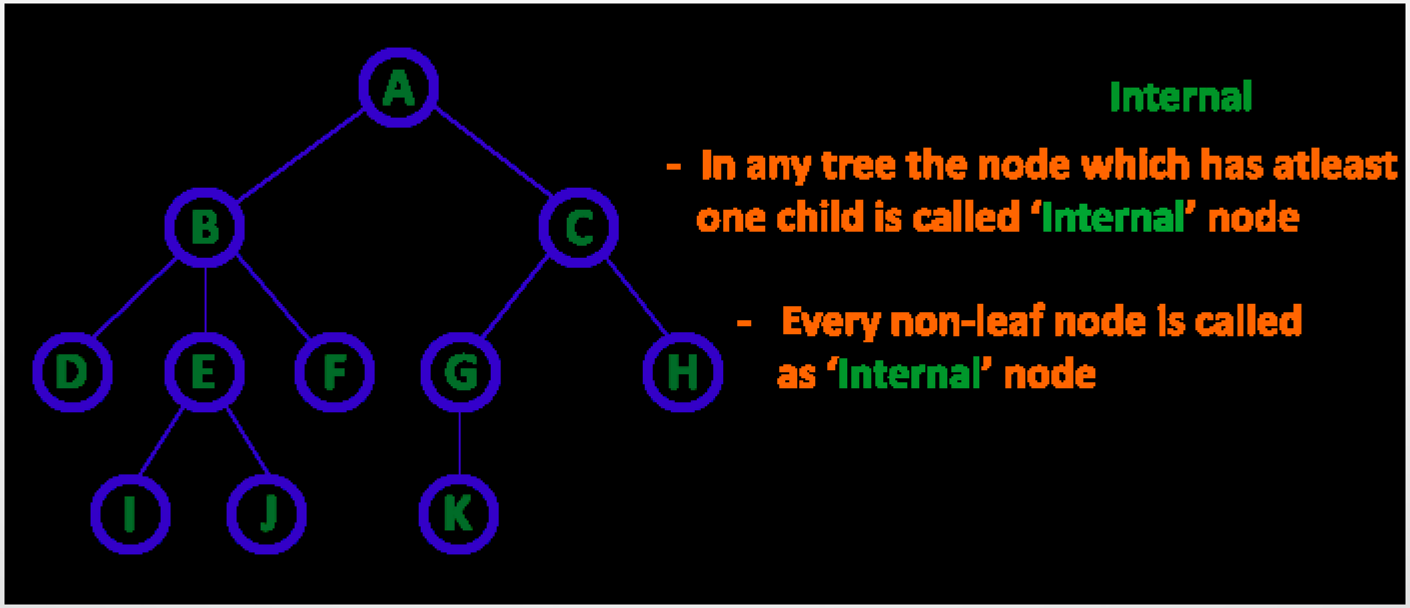


Leaf

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

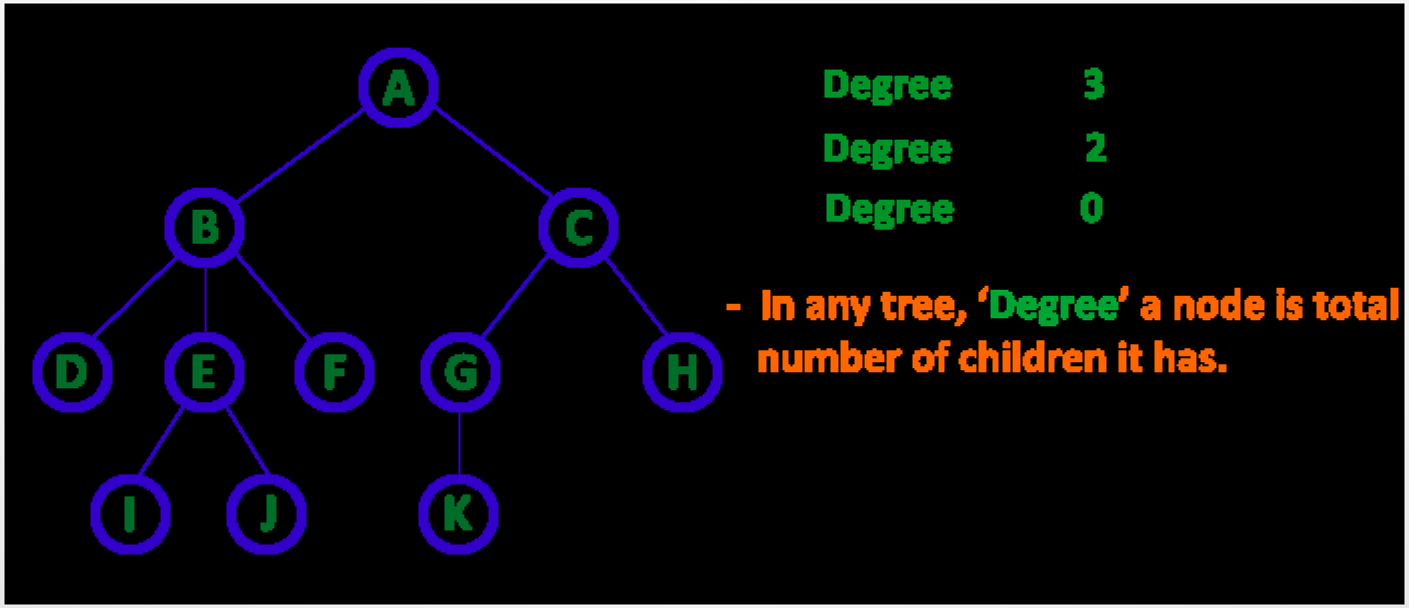
TREE TERMINOLOGY

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


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

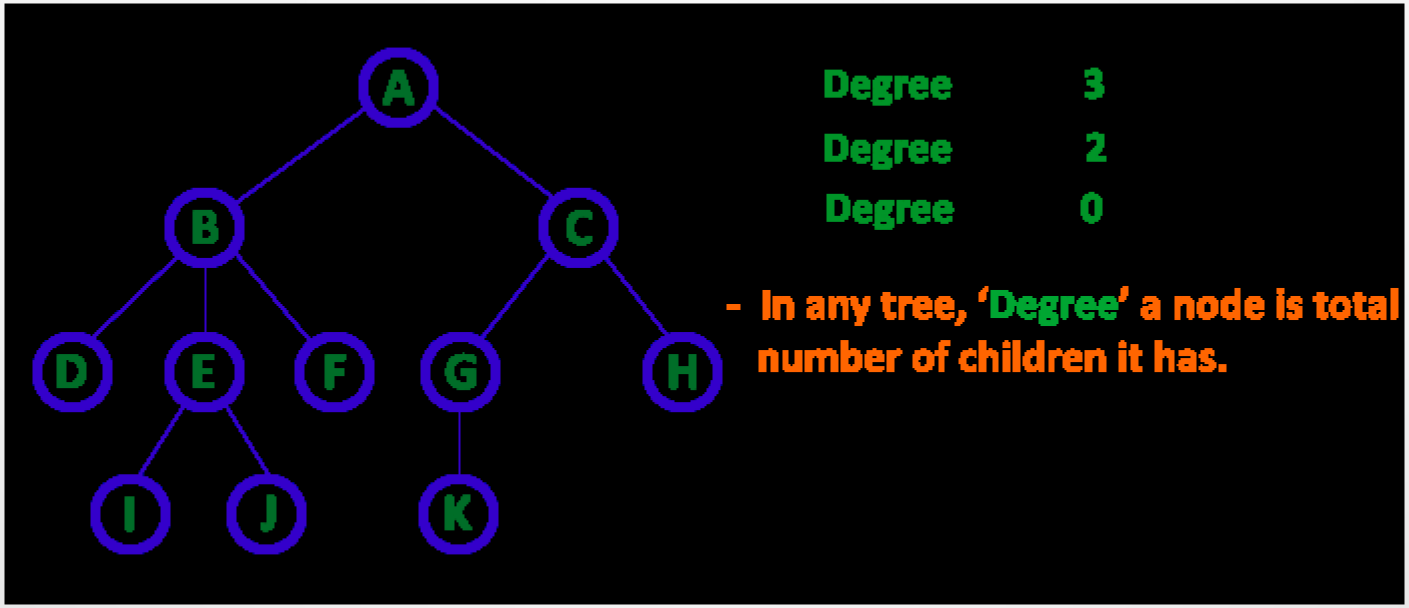
TREE TERMINOLOGY

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


A
B
C
D
E
F
G
H
I
J
K
Degree 3
Degree 2
Degree 0
- In any tree, 'Degree' a node is total number of children it has.

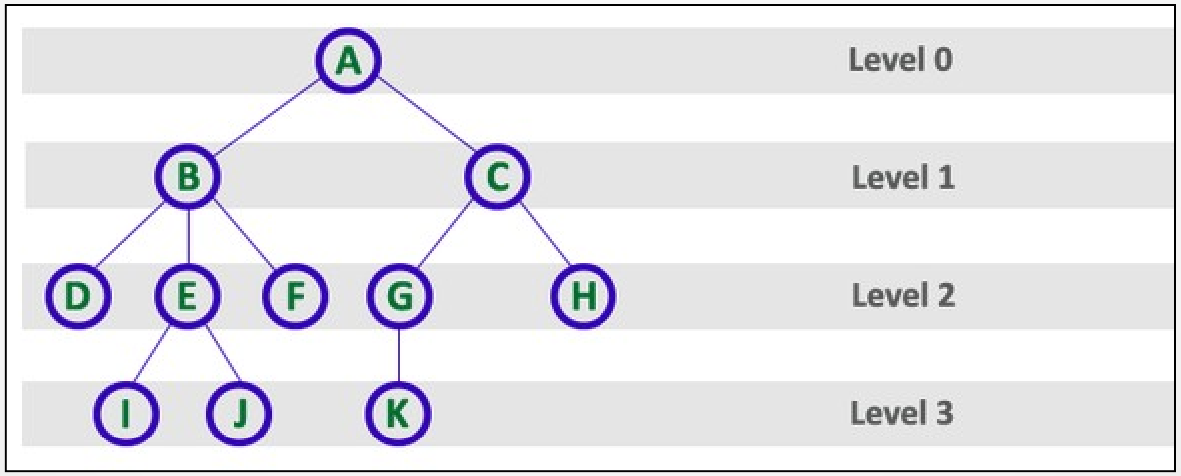
TREE TERMINOLOGY

- **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'


A
B
C
D
E
F
G
H
I
J
K
Degree 3
Degree 2
Degree 0
- In any tree, 'Degree' a node is total number of children it has.

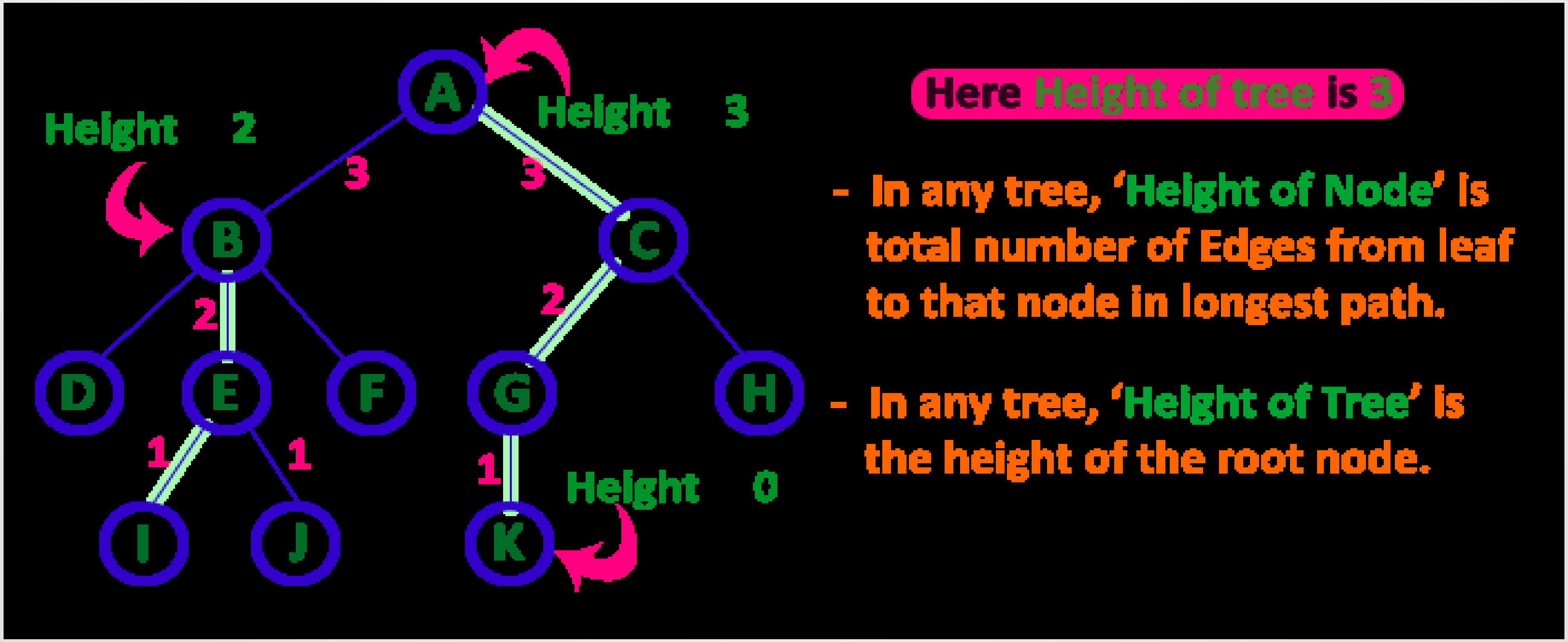
TREE TERMINOLOGY

- **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 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'.

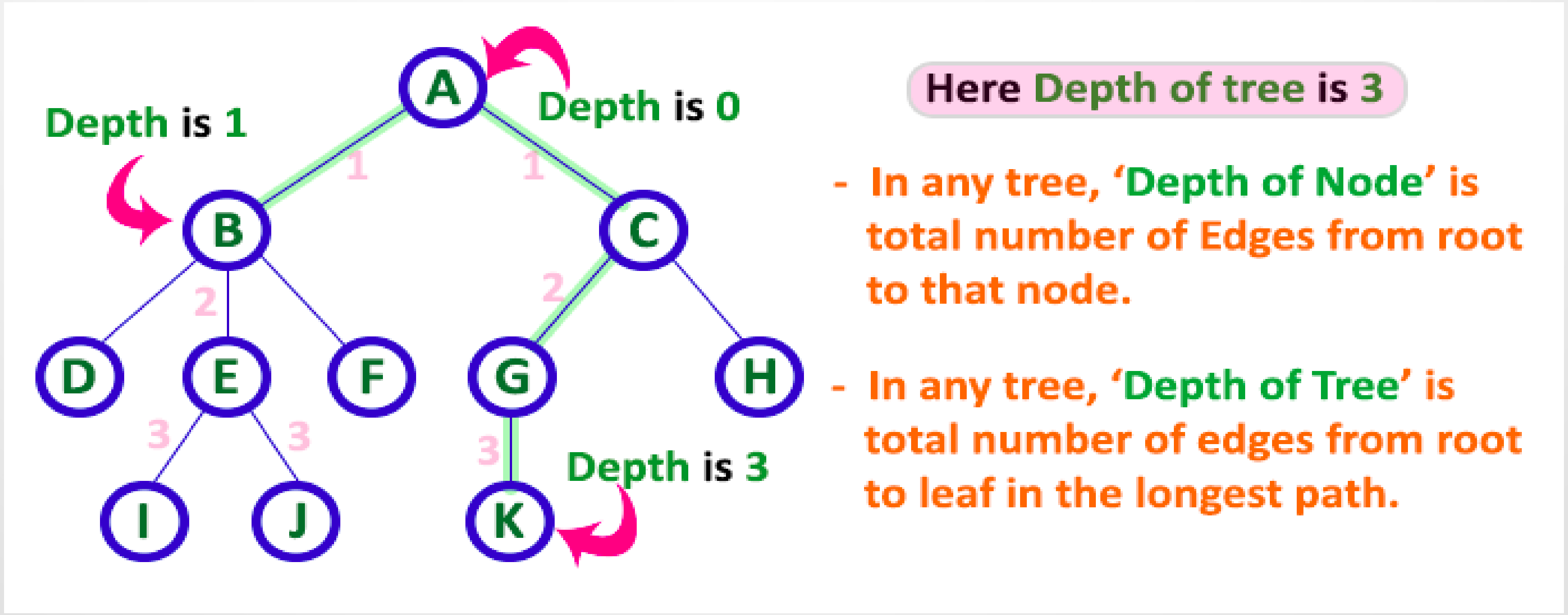


Here Height of tree is 3

- 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.

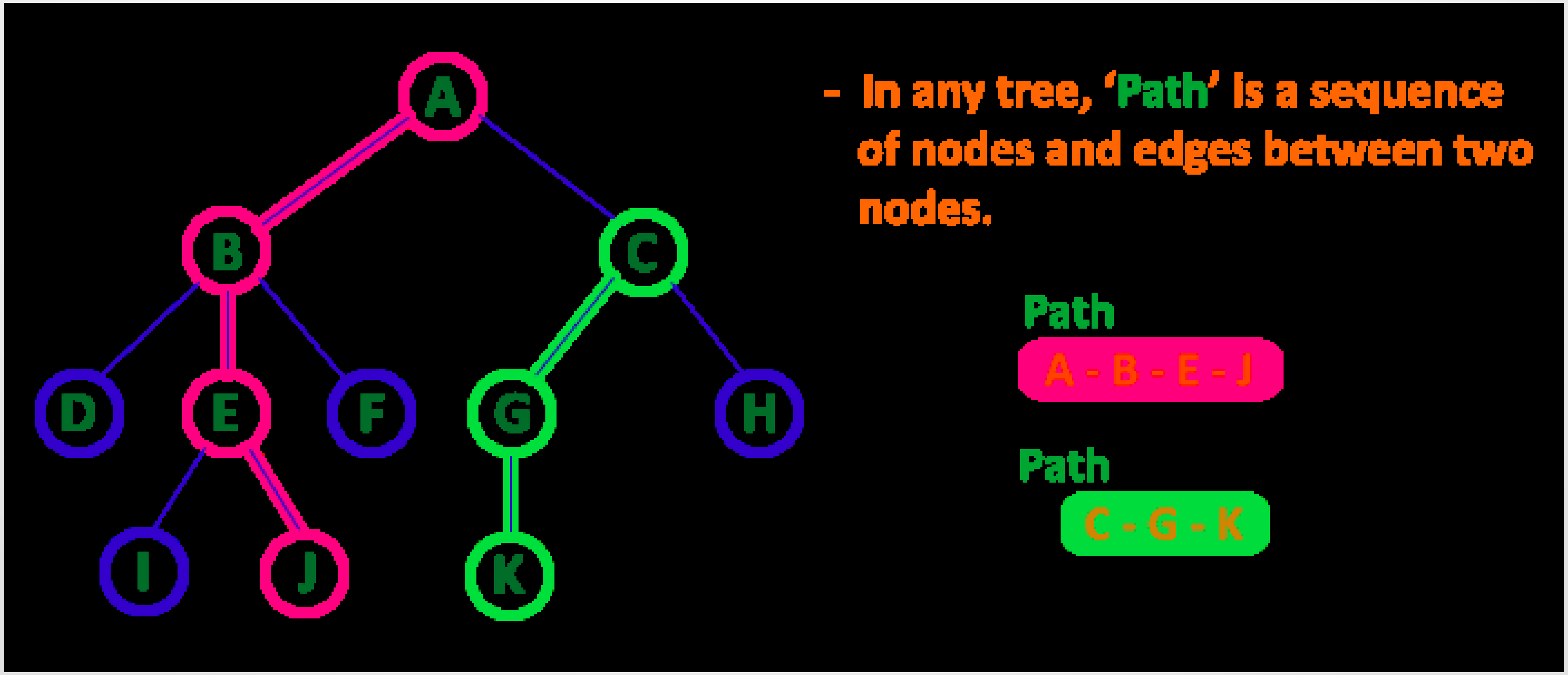
TREE TERMINOLOGY

- **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 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'.


Depth is 1
A
Depth is 0
B
C
D
E
F
G
H
I
J
K
Depth is 3
1
1
2
2
3
3
3
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

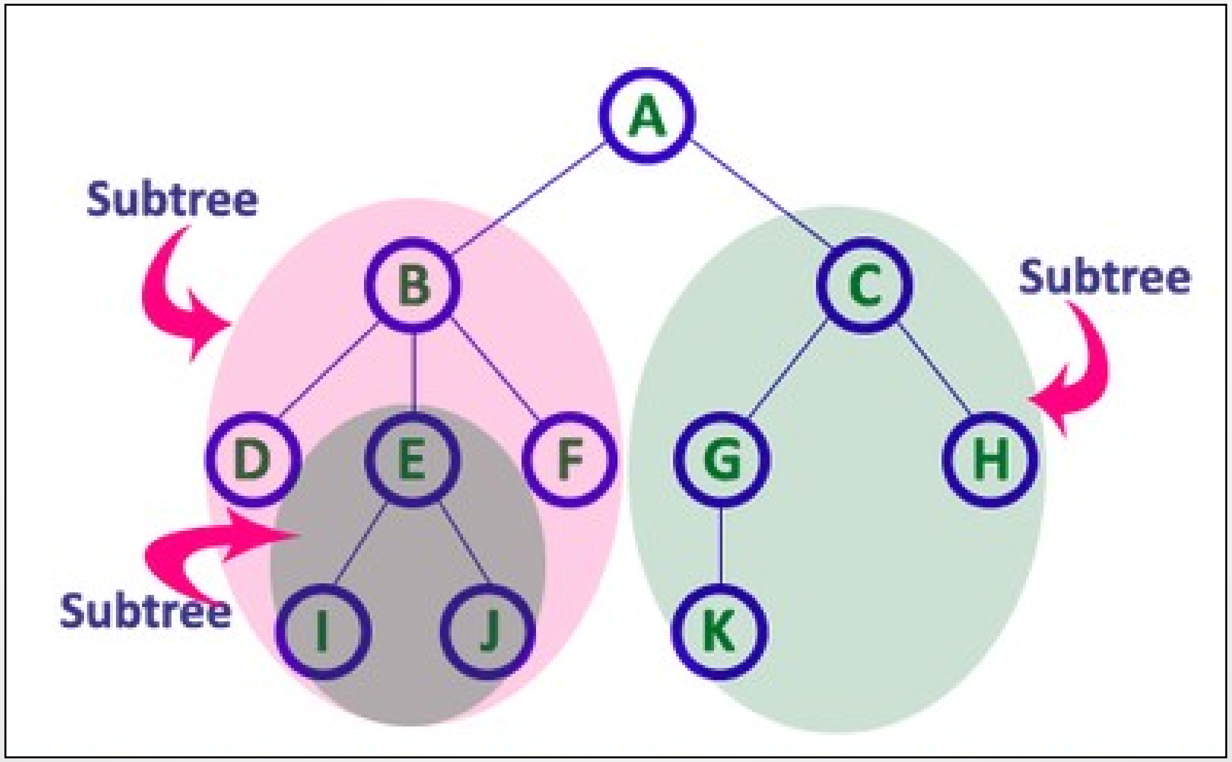
- **Path:** In a tree data structure, the sequence of Nodes and Edges from one node to another node is called as PATH between that two Nodes. Length of a Path is total number of nodes in that path. In below example the path A - B - E - J has length 4.

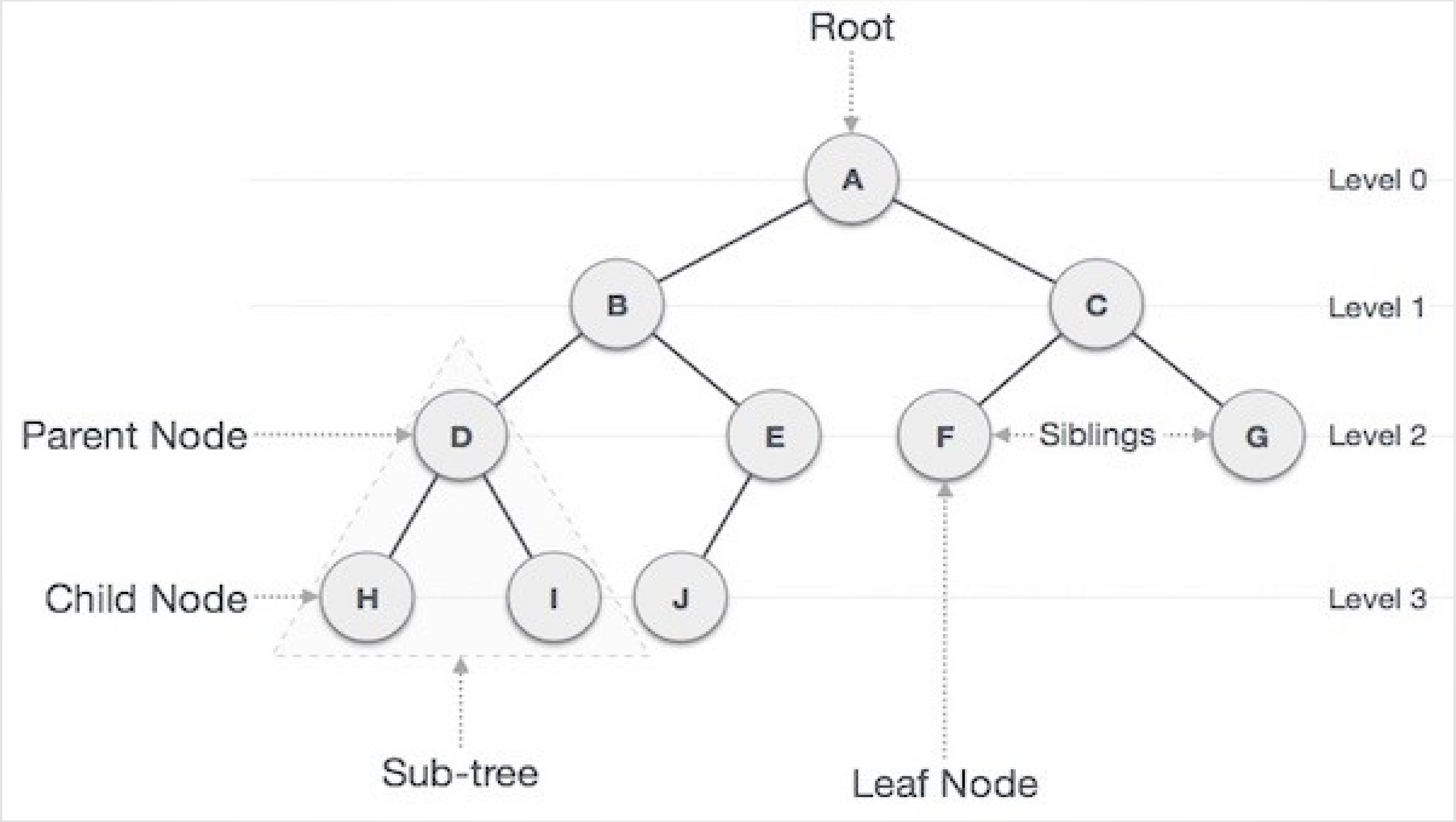


A
B
C
D
E
F
G
H
I
J
K
- In any tree, 'Path' is a sequence of nodes and edges between two nodes.
Path
A - B - E - J
Path
C - G - K

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.



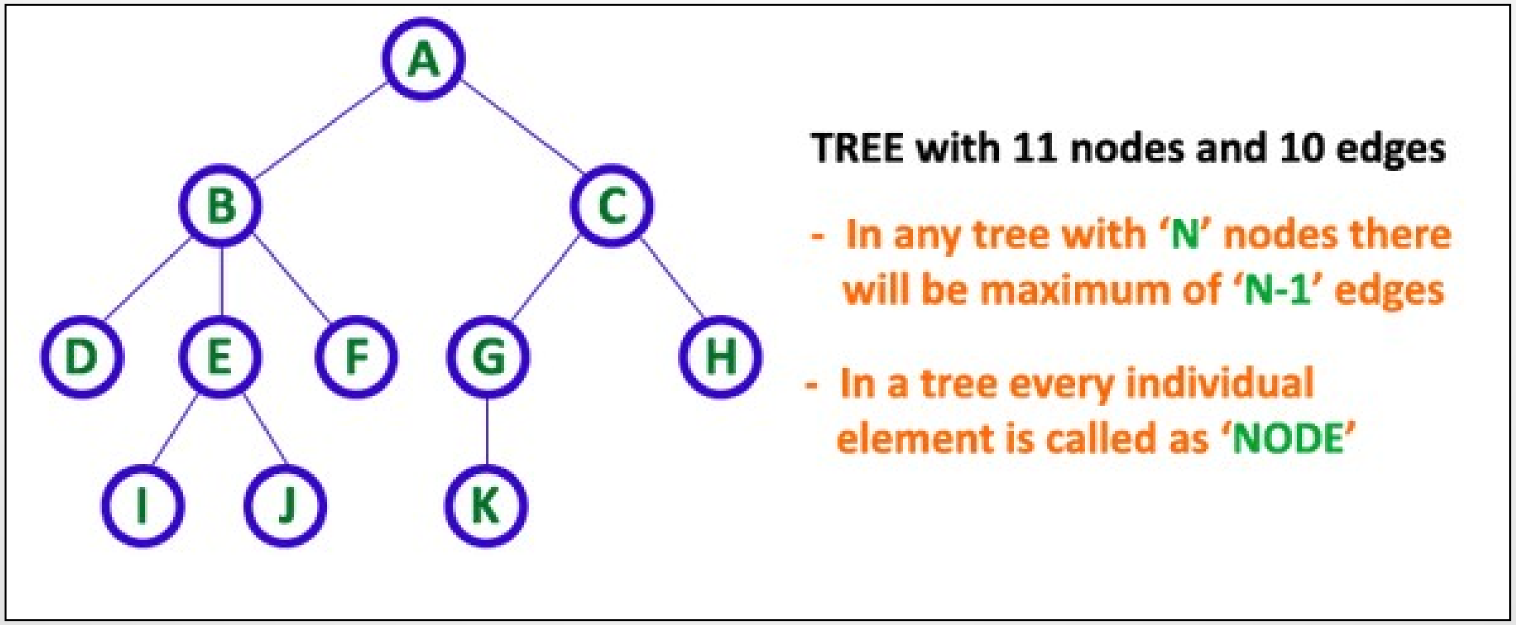

Root
A
B
C
D
E
F
G
H
I
J
Level 0
Level 1
Level 2
Level 3
Parent Node
Child Node
Siblings
Sub-tree
Leaf Node

BINARY TREE

- A tree whose elements have at most 2 children is called a binary tree. Since each element in a binary tree can have only 2 children, we typically name them the left and right child.
- A tree is represented by a pointer to the topmost node in tree. If the tree is empty, then value of root is NULL.

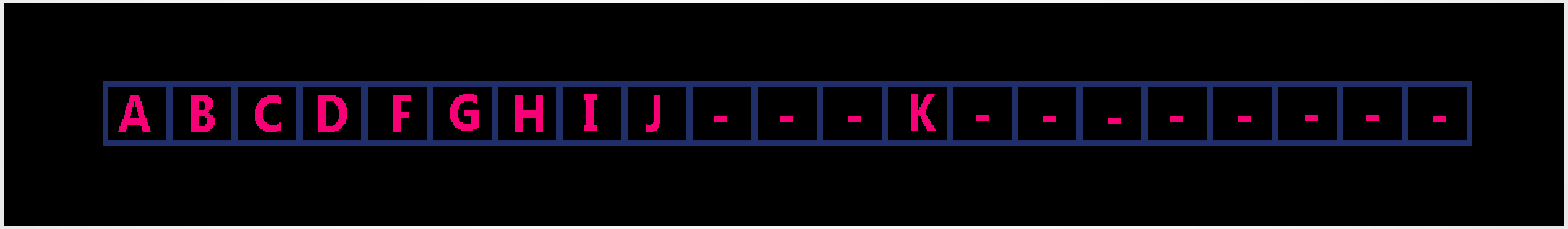
TREE Representation

- A tree data structure can be represented in two methods. Those methods are as follows...
  - Array Representation
  - Linked List Representation


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 Representation

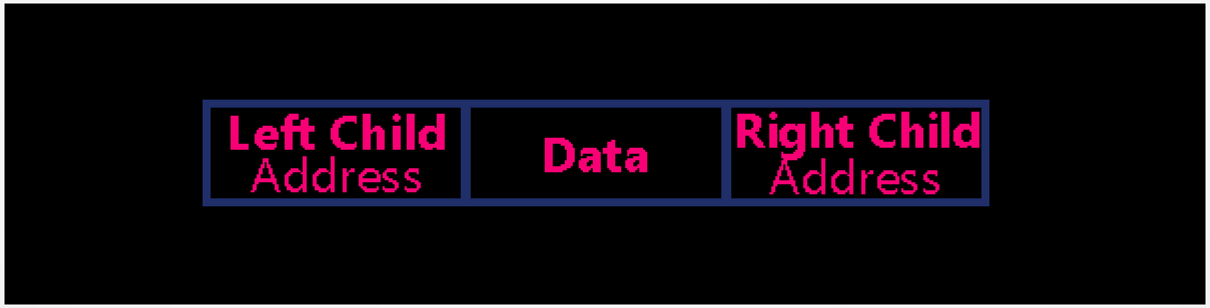
- **Array Representation:** In array representation of binary tree, we use a one dimensional array (1-D Array) to represent a binary tree.
- Consider the above example of binary tree and it is represented as follows



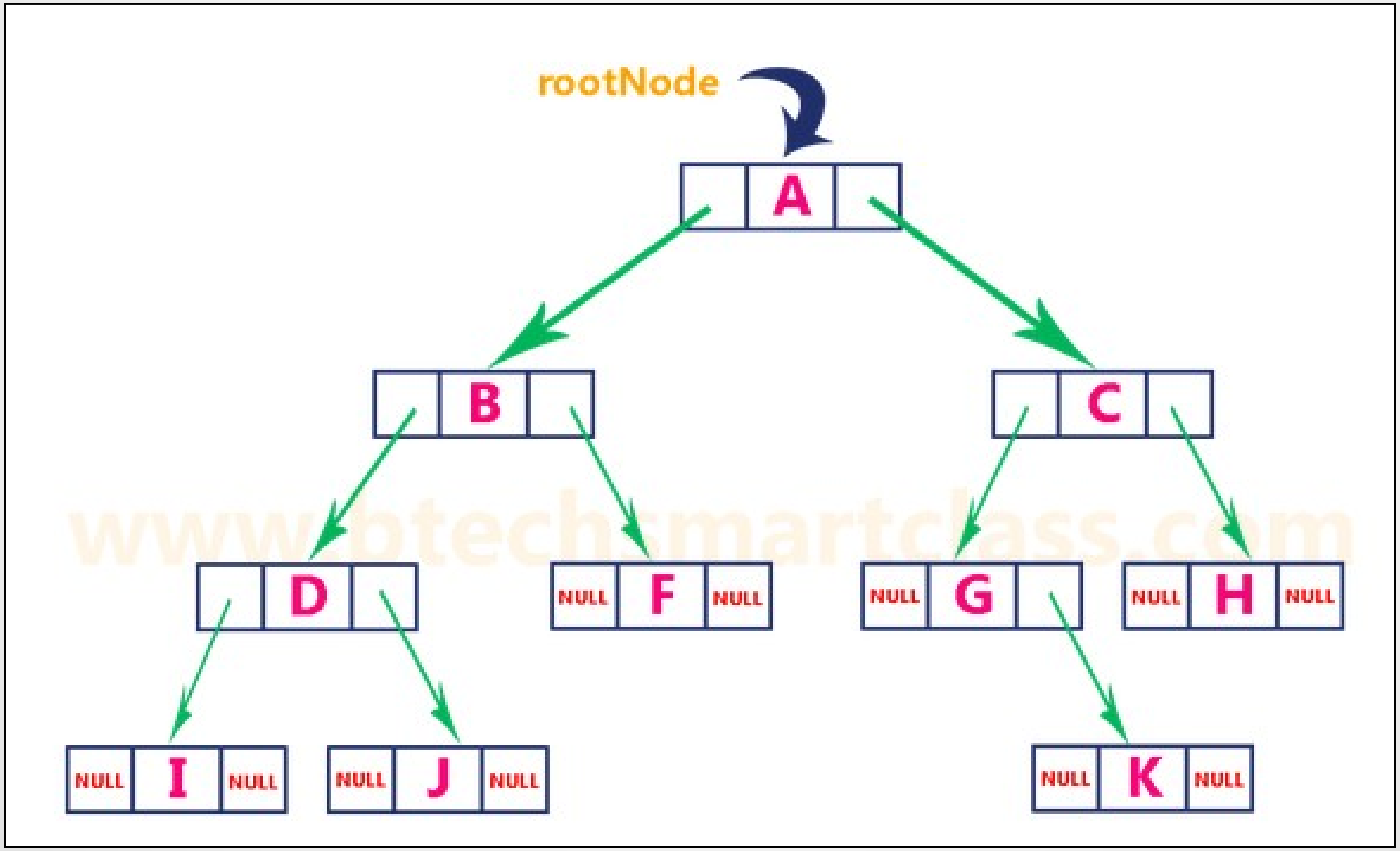
A	B	C	D	F	G	H	I	J	-	-	-	K	-	-	-	-	-	-	-	-

TREE Representation

- **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**

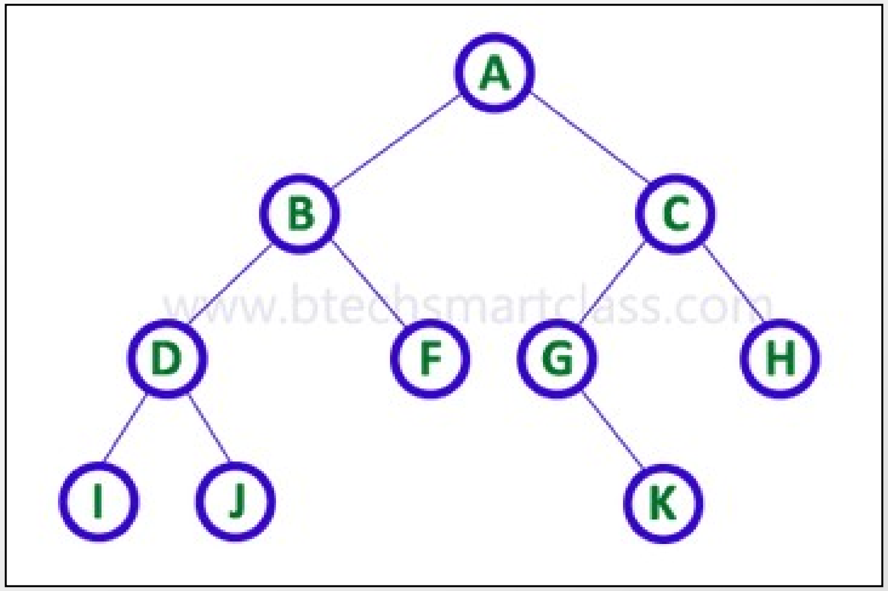


TREE Representation



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



Binary Tree Traversals

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.

Binary Tree Traversals

2. 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.

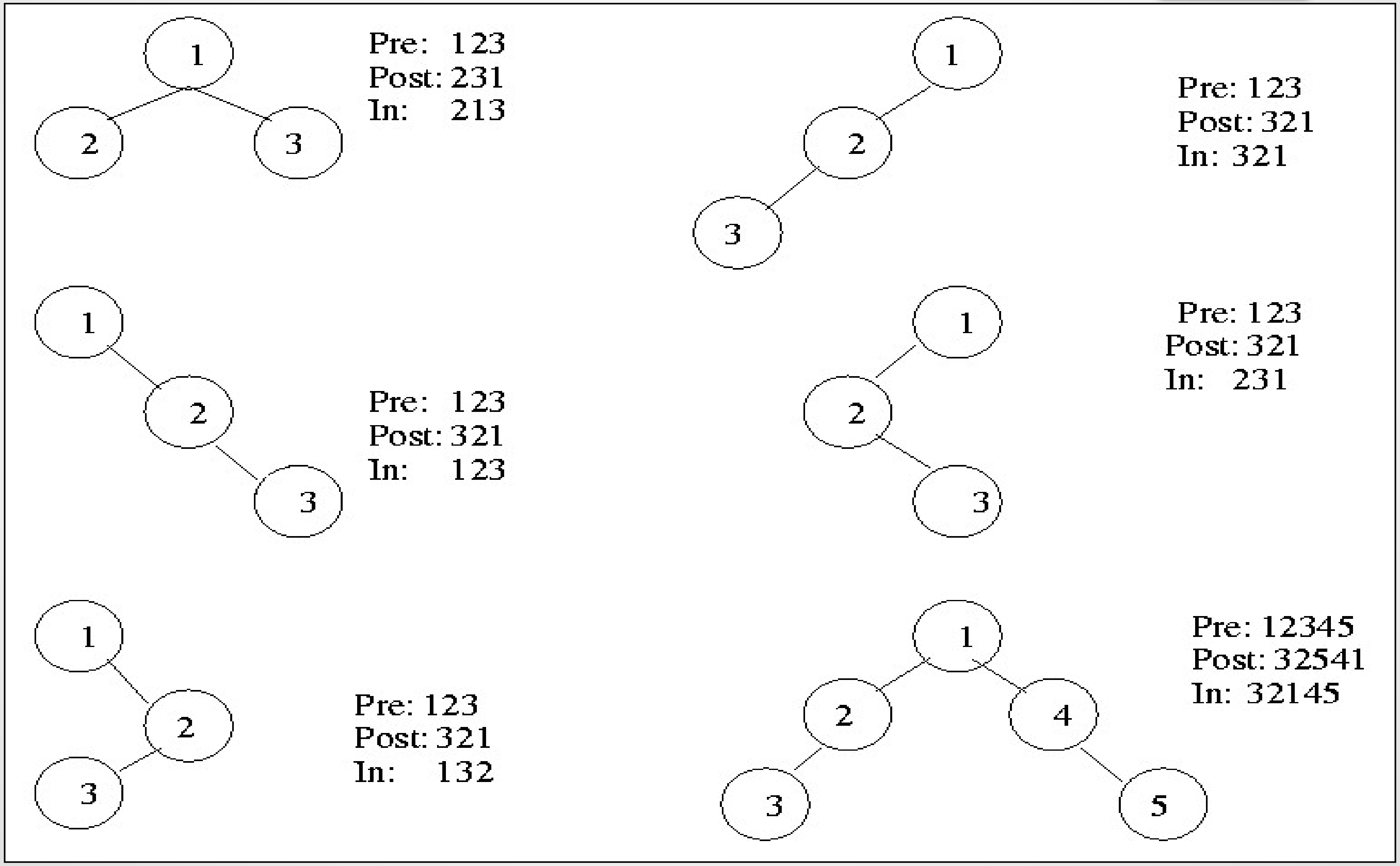
- In the above example of binary tree, first we visit root node 'A' then visit its left child 'B' which is a root for D and F. So we visit B's left child 'D' and again D is a root for I and J. So we visit D's left child 'I' which is the left most child. So next we go for visiting D's right child 'J'. 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.

Binary Tree Traversals

3. IPost - 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.

Binary Tree Traversals



Pre: 123
Post: 231
In: 213

Pre: 123
Post: 321
In: 321

Pre: 123
Post: 321
In: 123

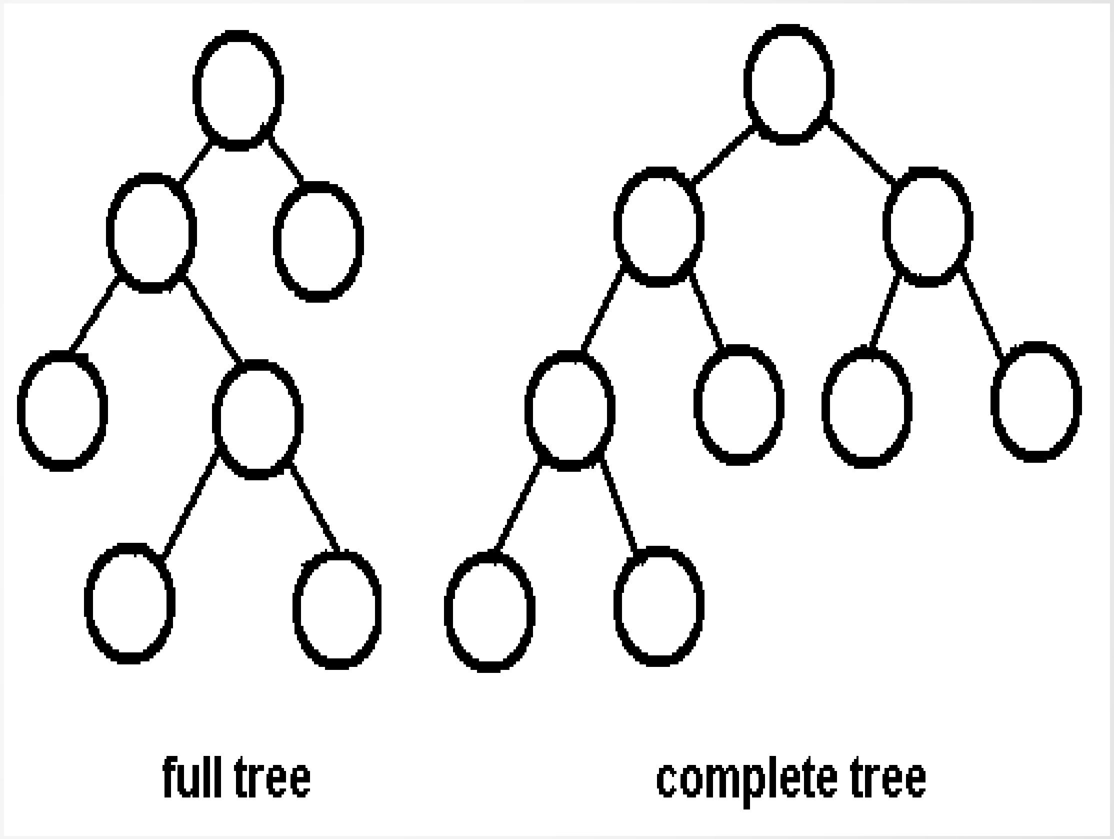
Pre: 123
Post: 321
In: 231

Pre: 123
Post: 321
In: 132

Pre: 12345
Post: 32541
In: 32145

COMPLETE AND FULL TREE

- A full binary tree.is a binary tree in which each node has exactly zero or two children.
- A complete binary tree is a binary tree, which is completely filled, with the possible exception of the bottom level, which is filled from left to right.

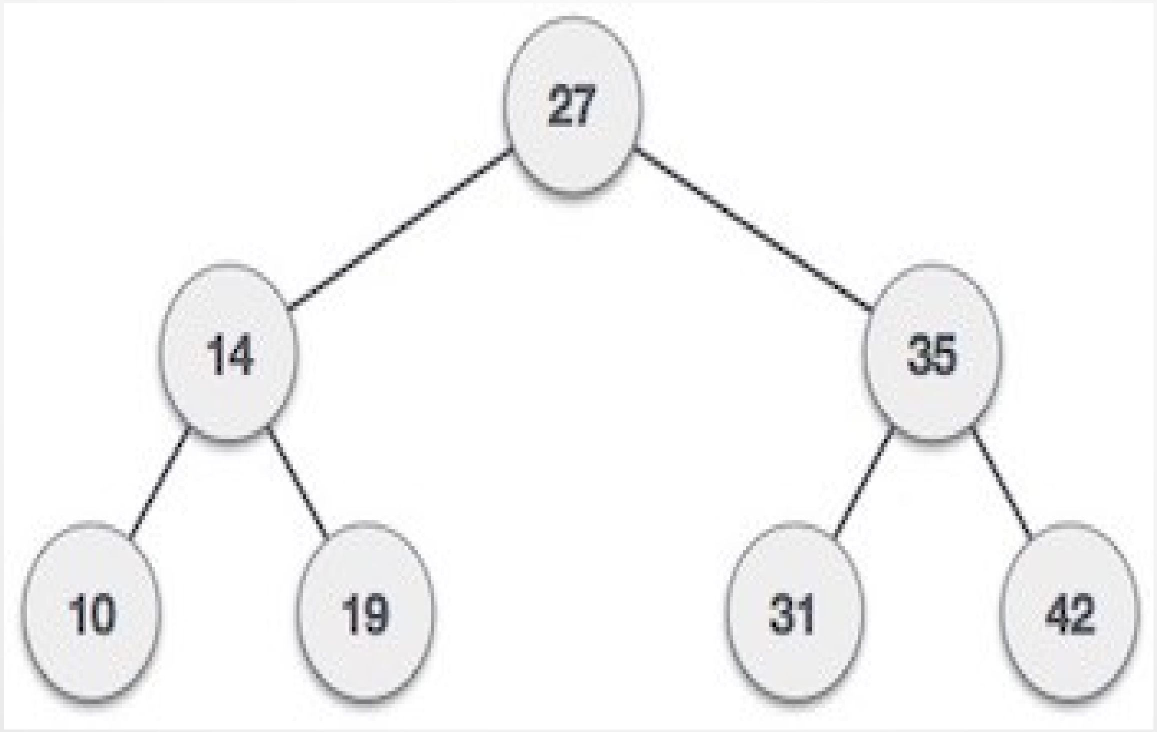


Binary Search Tree

A Binary Search Tree (BST) is a tree in which all the nodes follow the below-mentioned properties −

- The value of the key of the left sub-tree is less than the value of its parent (root) node's key.
- The value of the key of the right sub-tree is greater than or equal to the value of its parent (root) node's key.
- Thus, BST divides all its sub-trees into two segments; the left sub-tree and the right sub-tree and can be defined as −
- left_subtree (keys) < node (key) ≤ right_subtree (keys)

The root node key (27) has all less-valued keys on the left sub-tree and the higher valued keys on the right sub-tree.



Basic Operations

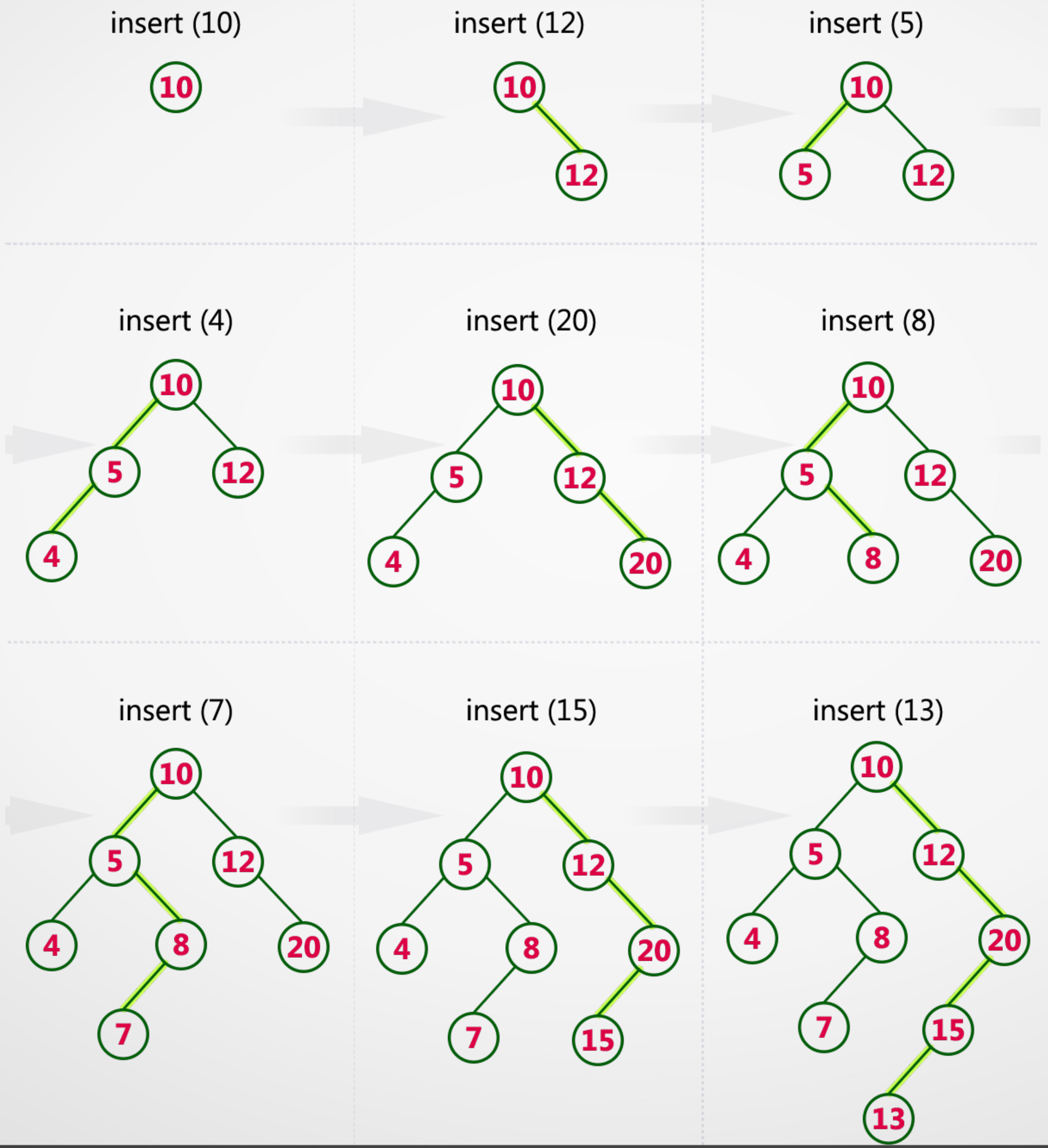
Following are the basic operations of a tree −

- Search − Searches an element in a tree.
- Insert − Inserts an element in a tree.
- Pre-order Traversal − Traverses a tree in a pre-order manner.
- In-order Traversal − Traverses a tree in an in-order manner.
- Post-order Traversal − Traverses a tree in a post-order manner.

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...

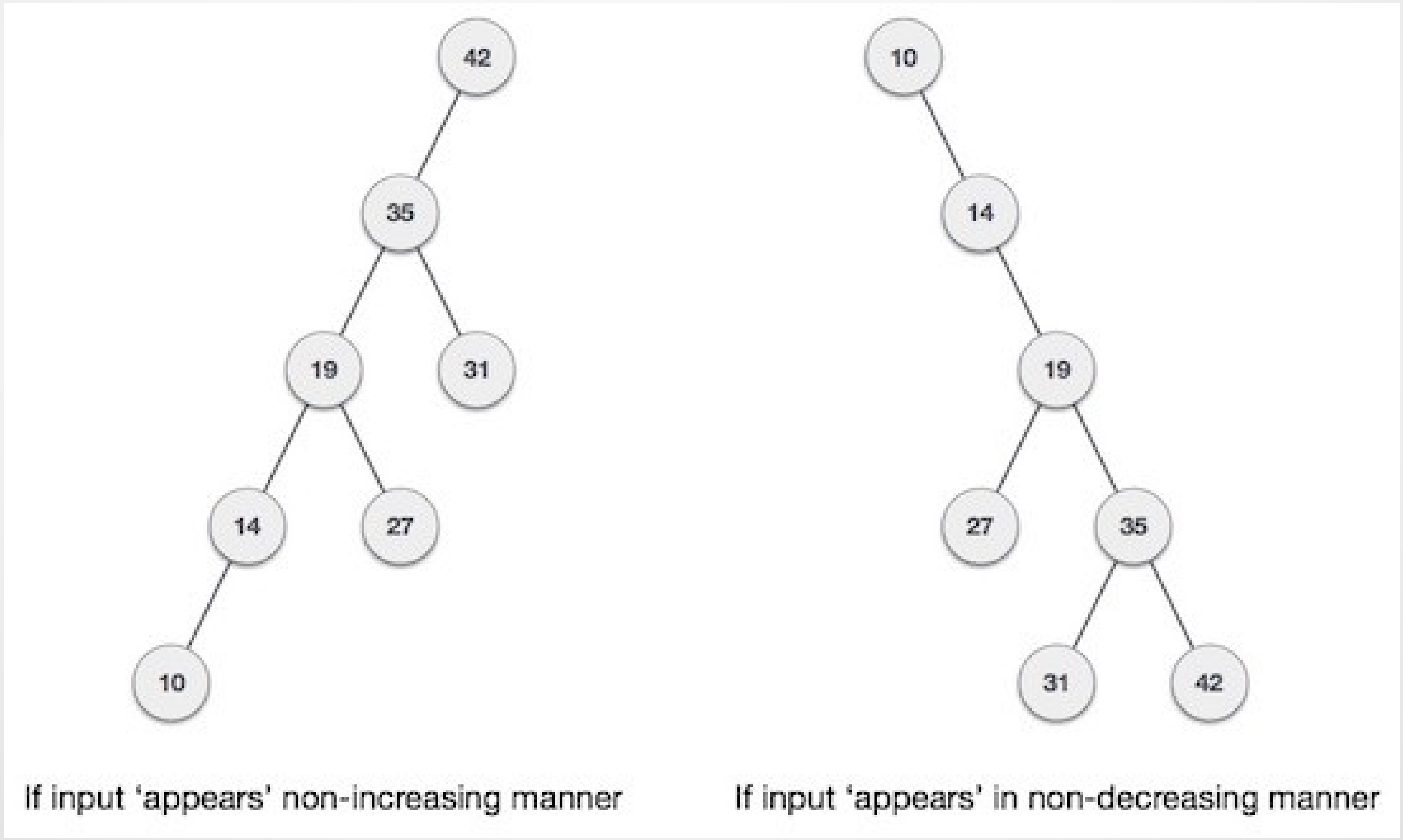

insert (10)
insert (12)
insert (5)
insert (4)
insert (20)
insert (8)
insert (7)
insert (15)
insert (13)

Example

Construct Binary Search Tree

- 50, 70, 60, 20, 90, 10, 40, 100
- 56, 63, 45, 23, 89, 13, 65, 72, 88

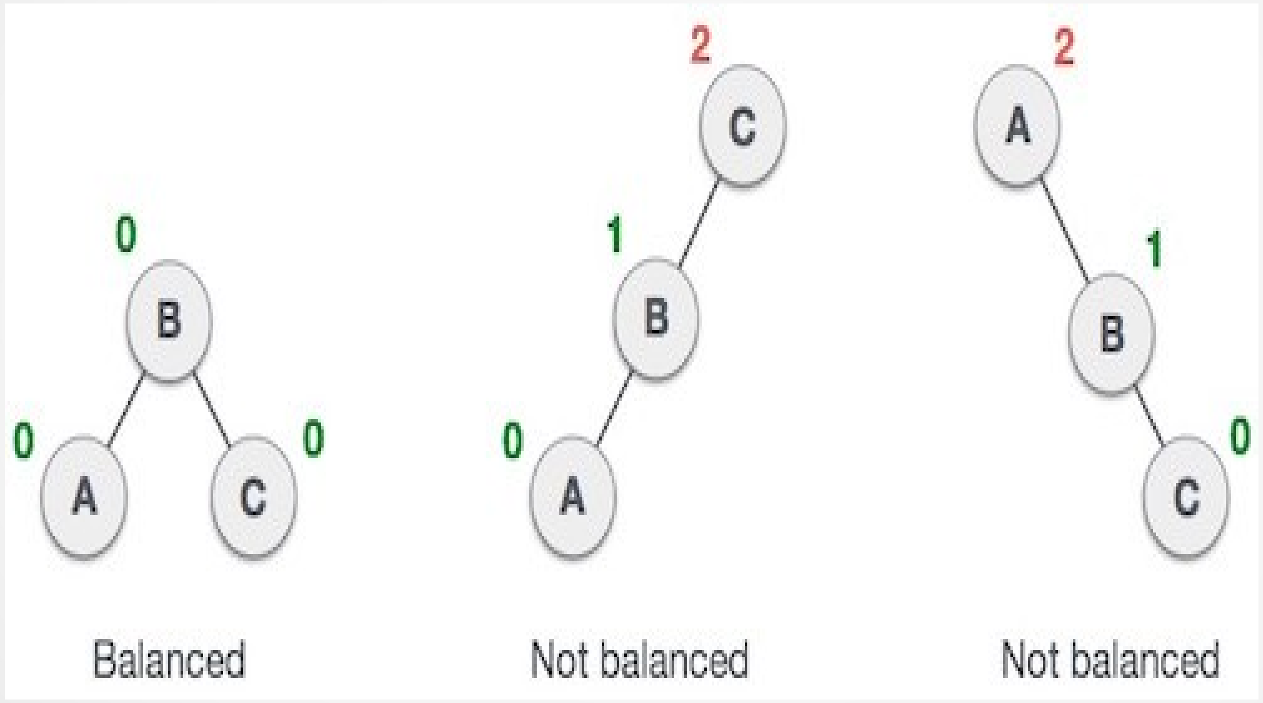
What if the input to binary search tree comes in a sorted (ascending or descending) manner? It will then look like this –


42
35
19
31
14
27
10
If input 'appears' non-increasing manner
10
14
19
27
35
31
42
If input 'appears' in non-decreasing manner

AVL / Height Balanced Tree

- It is observed that BST's worst-case performance is closest to linear search algorithms, that is O(n). In real-time data, we cannot predict data pattern and their frequencies. So, a need arises to balance out the existing BST.
- Named after their inventor Adelson, Velski & Landis, AVL trees are height balancing binary search tree.
- AVL tree checks the height of the left and the right sub-trees and assures that the difference is not more than 1. This difference is called the Balance Factor.

- Here we see that the first tree is balanced and the next two trees are not balanced −



- In the second tree, the left subtree of C has height 2 and the right subtree has height 0, so the difference is 2. In the third tree, the right subtree of A has height 2 and the left is missing, so it is 0, and the difference is 2 again.

- AVL tree permits difference (balance factor) to be only 1.

`BalanceFactor = height(left-sutree) − height(right-sutree)`

- 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

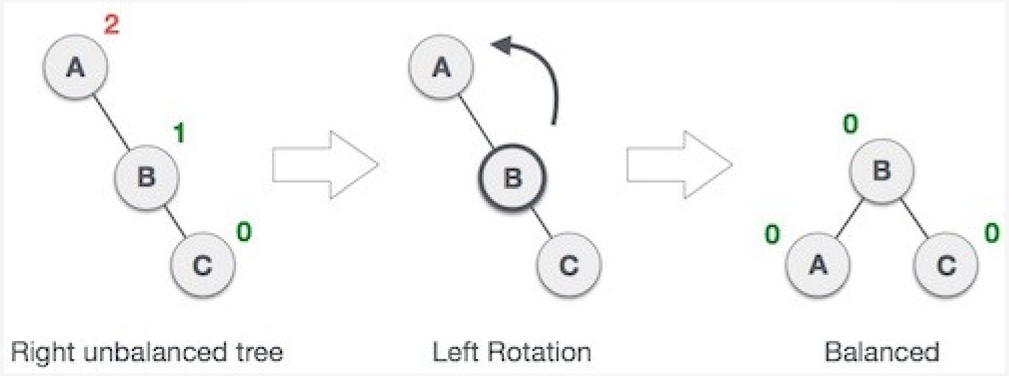
To balance itself, an AVL tree may perform the following four kinds of rotations −

- 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.

Left Rotation

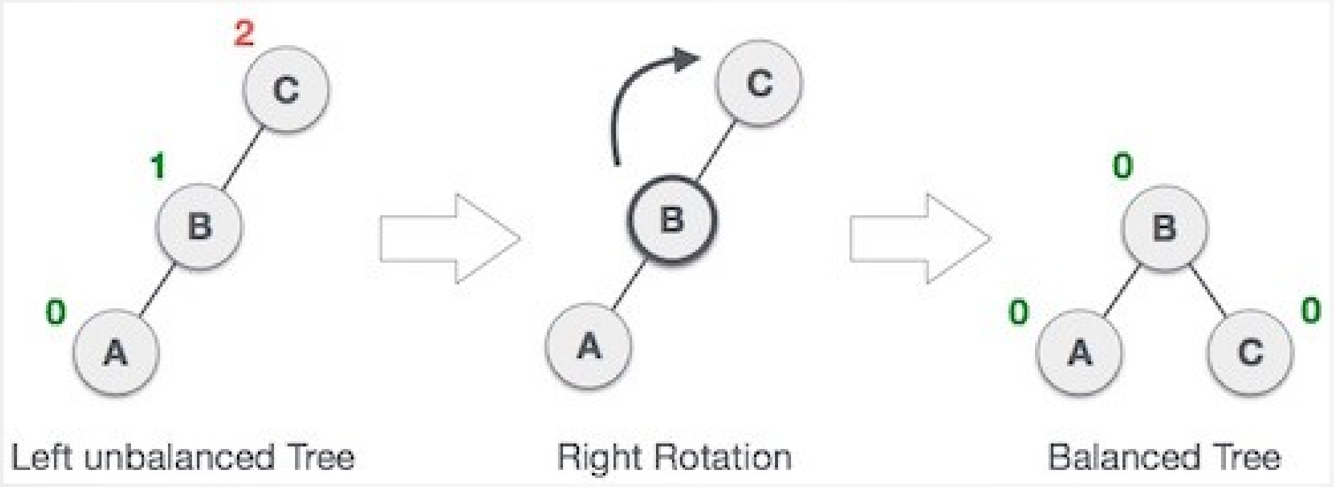
- If a tree becomes unbalanced, when a node is inserted into the right subtree of the right subtree, then we perform a single left rotation −



- 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

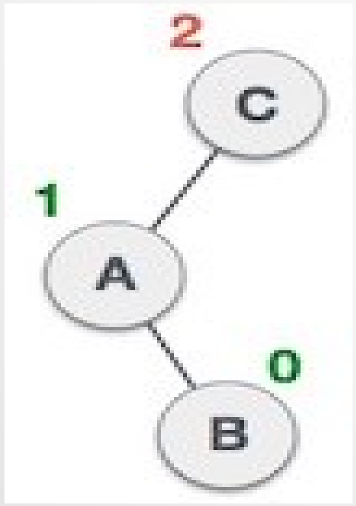
- AVL tree may become unbalanced, if a node is inserted in the left subtree of the left subtree. The tree then needs a right rotation.

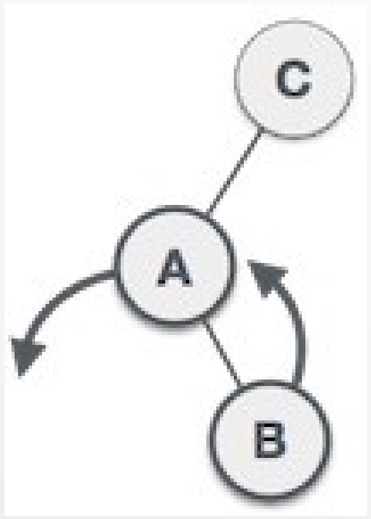


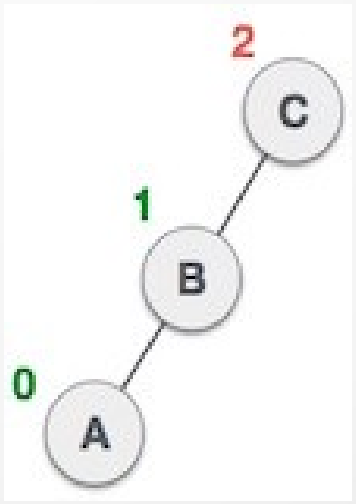
- The unbalanced node becomes the right child of its left child by performing a right rotation.

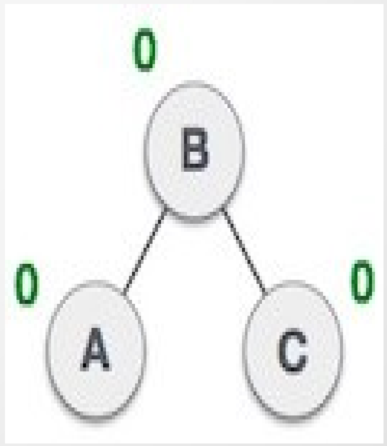
Left Right Rotation

- Double rotations are slightly complex version of already explained versions of rotations. To understand them better, we should take note of each action performed while rotation. Let's first check how to perform Left-Right rotation. A left-right rotation is a combination of left rotation followed by right rotation.



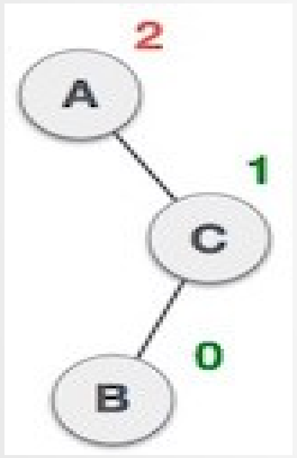


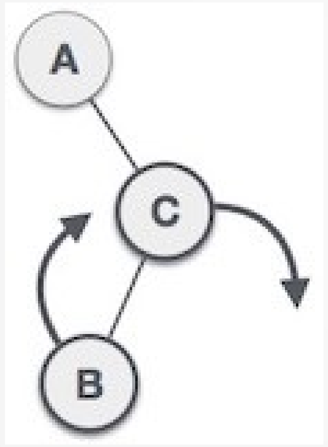


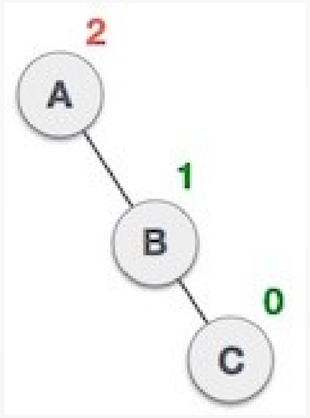


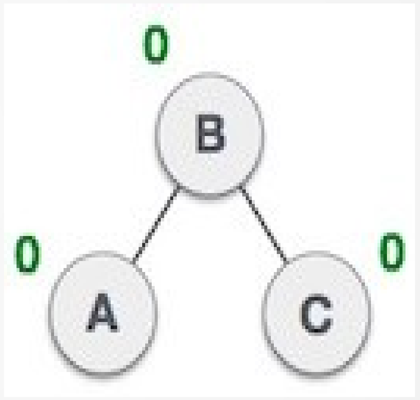
Right Left Rotation

The second type of double rotation is Right-Left Rotation. It is a combination of right rotation followed by left rotation.

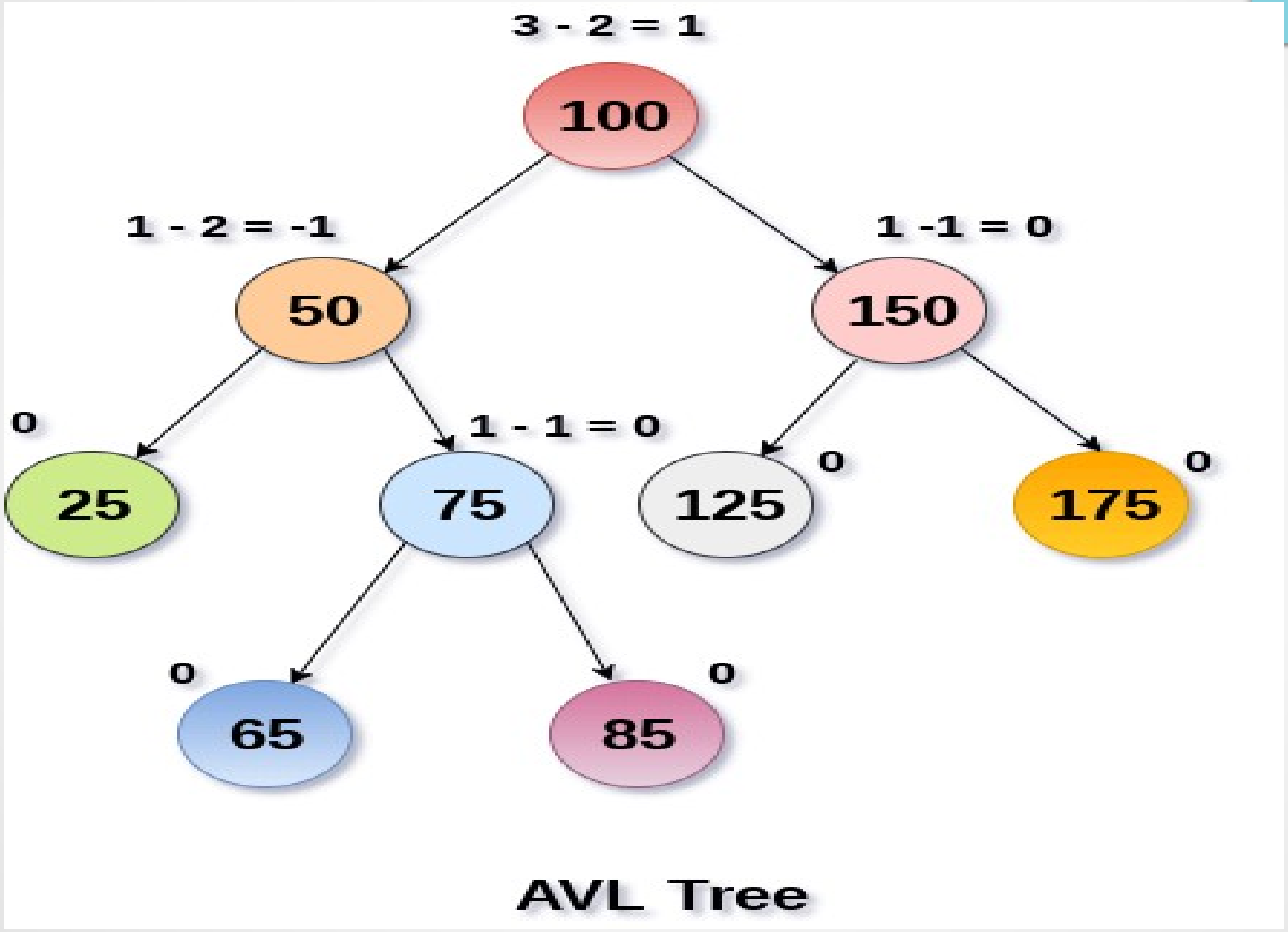






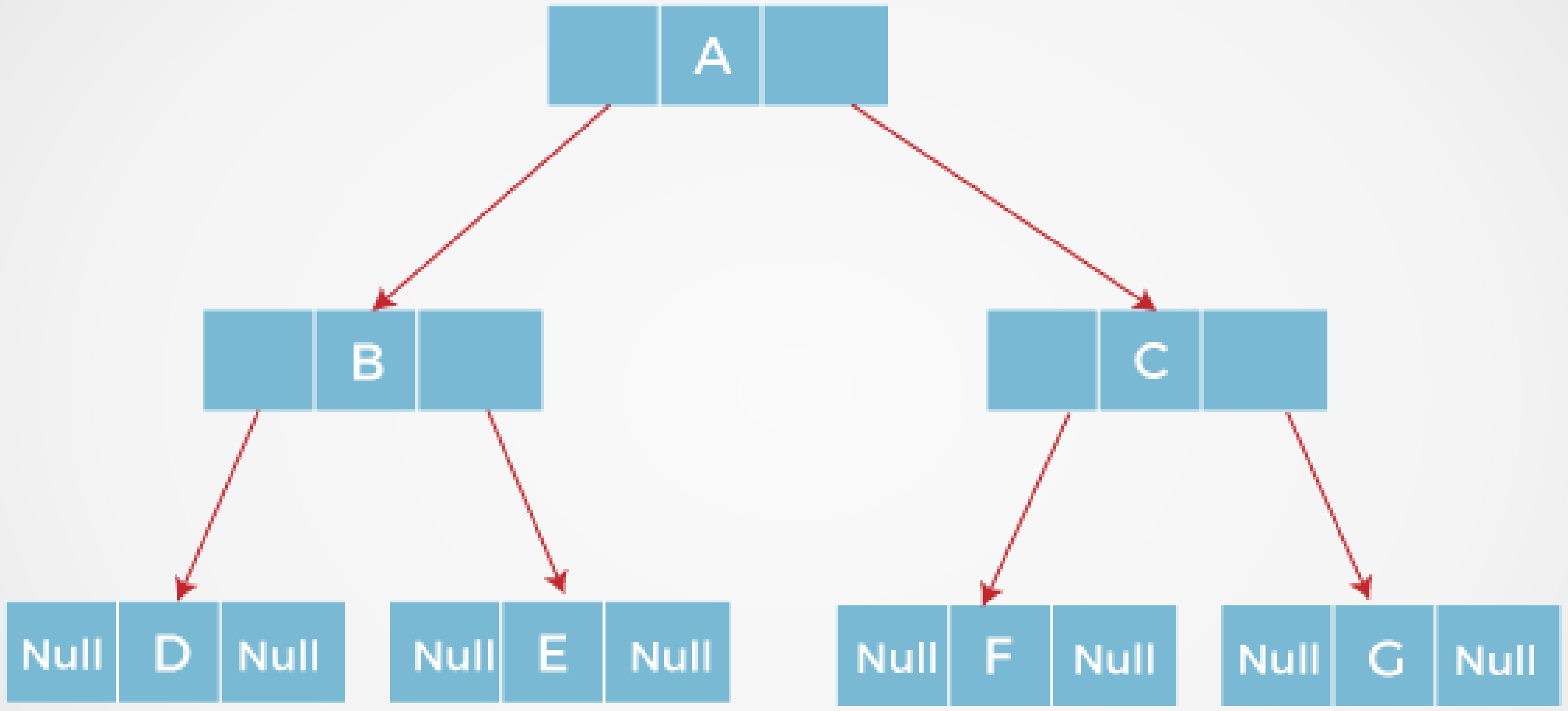


Example



Threaded Binary Tree

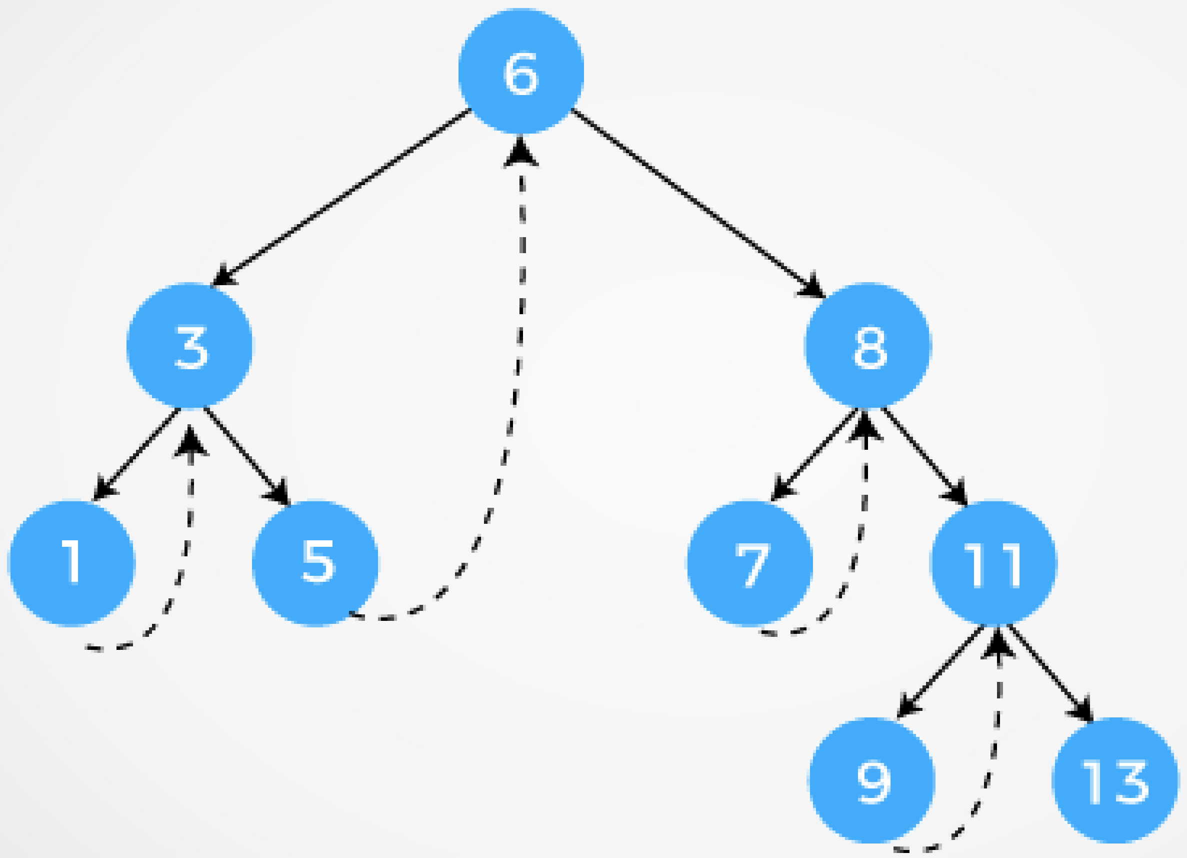
- In the linked representation of binary trees, more than one half of the link fields contain NULL values which results in wastage of storage space.
- If a binary tree consists of n nodes then n+1 link fields contain NULL values. So in order to effectively manage the space, a method was devised by Perlis and Thornton in which the NULL links are replaced with special links known as threads. Such binary trees with threads are known as threaded binary trees.
- Each node in a threaded binary tree either contains a link to its child node or thread to other nodes in the tree.


A
B
C
Null
D
Null
Null
E
Null
Null
F
Null
Null
G
Null

Types of Threaded Binary Tree

One-way threaded Binary Tree

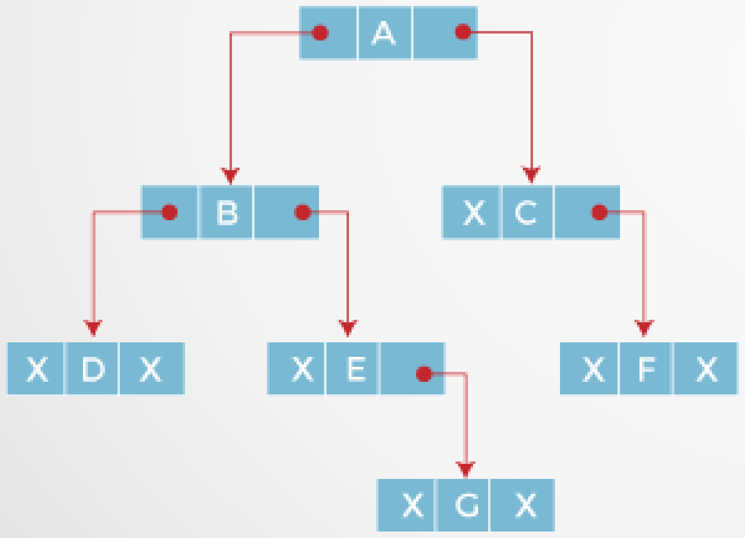
- In one-way threaded binary trees, a thread will appear either in the right or left link field of a node. If it appears in the right link field of a node then it will point to the next node that will appear on performing **in order** traversal.Such trees are called Right threaded binary trees.
- If thread appears in the left field of a node then it will point to the nodes inorder predecessor. Such trees are called Left threaded binary trees.
- Left threaded binary trees are used less often as they don't yield the last advantages of right threaded binary trees. In one-way threaded binary trees, the right link field of last node and left link field of first node contains a NULL. In order to distinguish threads from normal links they are represented by dotted lines.



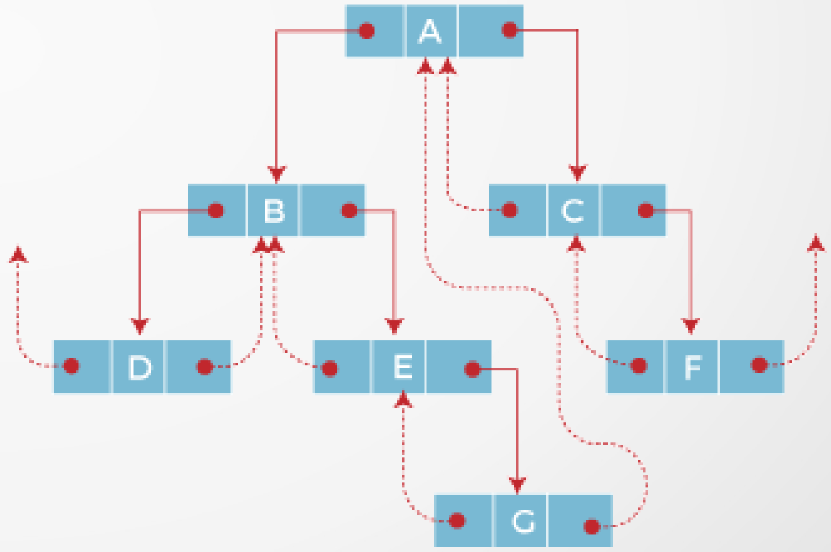
Single Threaded Binary Tree

Two – way Threaded Binary Tree

- 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.

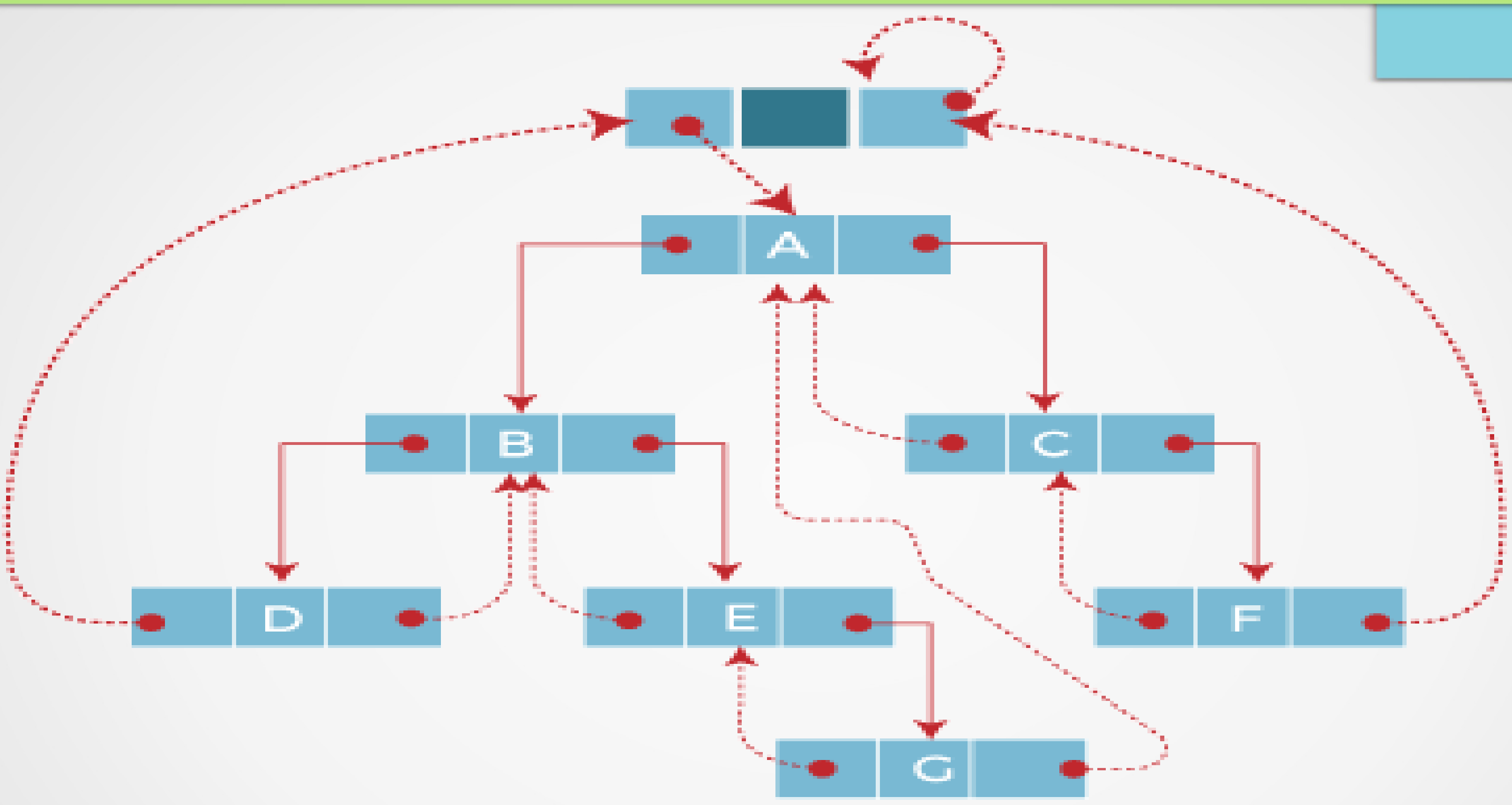


A binary tree (Inorder traversal - D, B, E, G, A, C, F)



A two - way threaded binary tree

- Two-way threaded Binary tree, we noticed that no left thread is possible for the first node and no right thread is possible for the last node.
- This is because they don't have any inorder predecessor and successor respectively. This is indicated by threads pointing nowhere.
- So in order to maintain the uniformity of threads, we maintain a special node called the header node. The header node does not contain any data part and its left link field points to the root node and its right link field points to itself.
- If this header node is included in the two-way threaded Binary tree then this node becomes the inorder predecessor of the first node and inorder successor of the last node. Now threads of left link fields of the first node and right link fields of the last node will point to the header node.



Two-way threaded - tree with header node