Trees

Definition

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

or

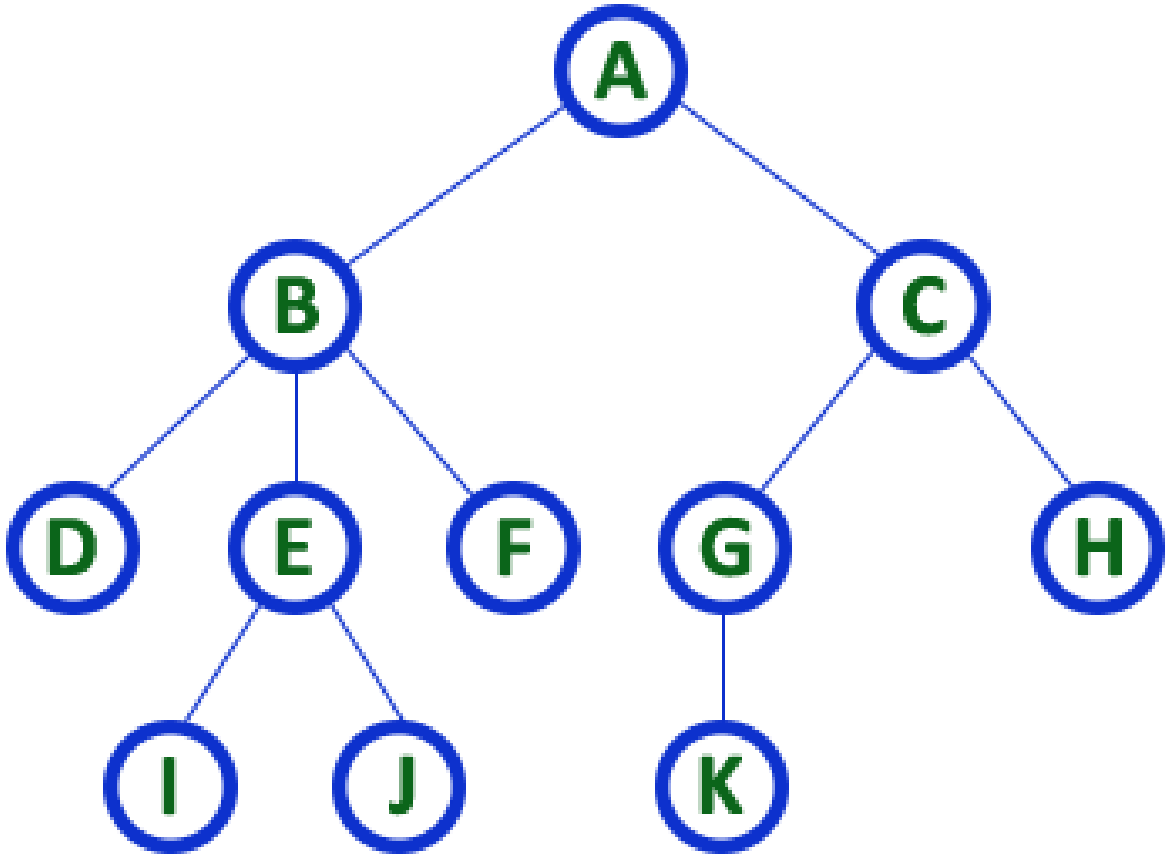
Tree data structure is a collection of data (Node) which is organized in hierarchical structure recursively

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

Source: http://www.btechsmartclass.com/data_structures/tree-terminology.html

Example



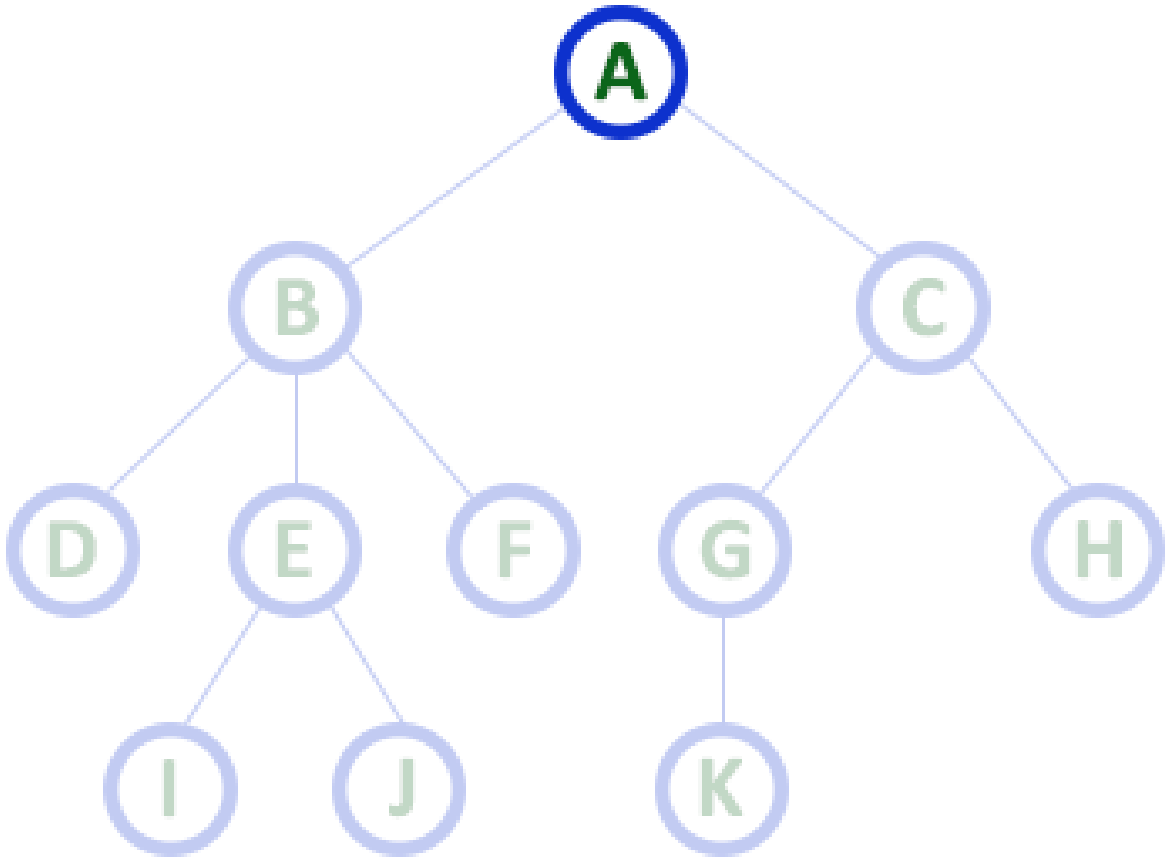
TREE with 11 nodes and 10 edges

- **In any tree with 'N' nodes there will be maximum of 'N-1' edges**
- **In a tree every individual element is called as 'NODE'**

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.

Example



Here ‘A’ is the ‘root’ node

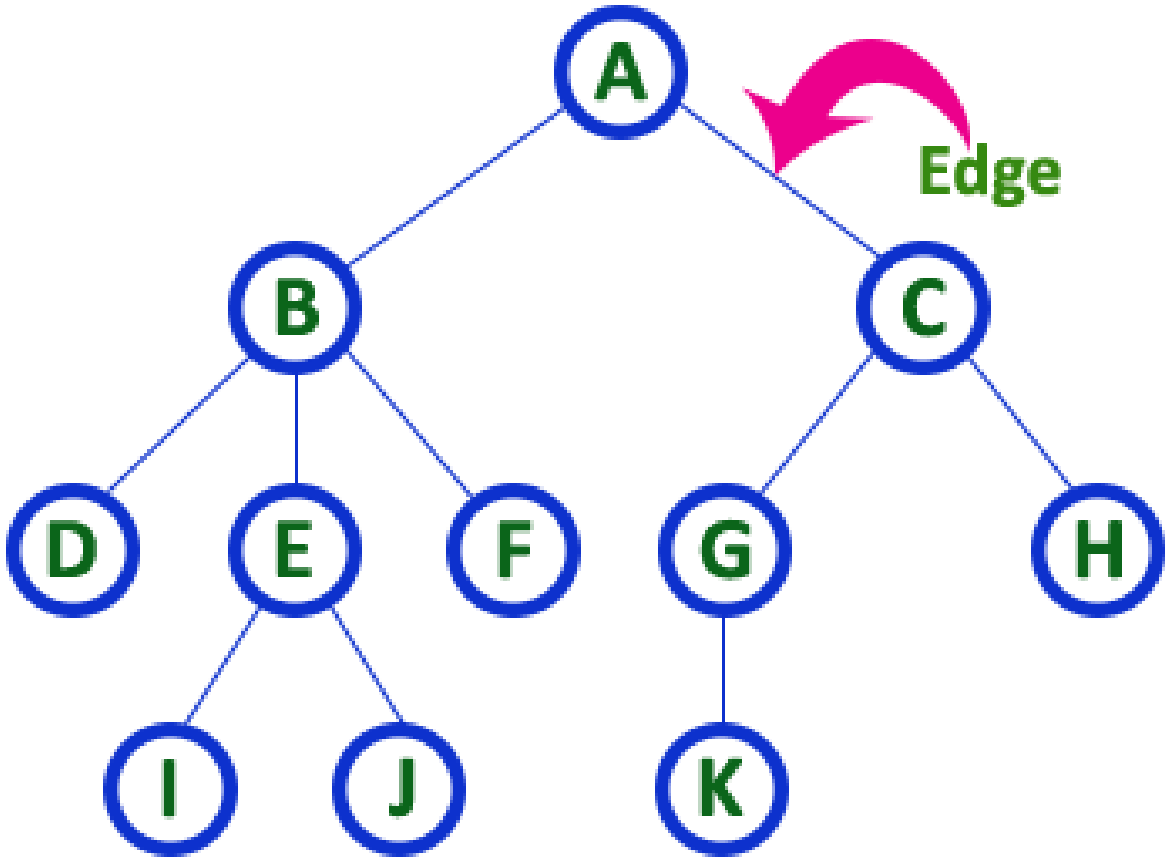
- In any tree the first node is called as ROOT node

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.

Example

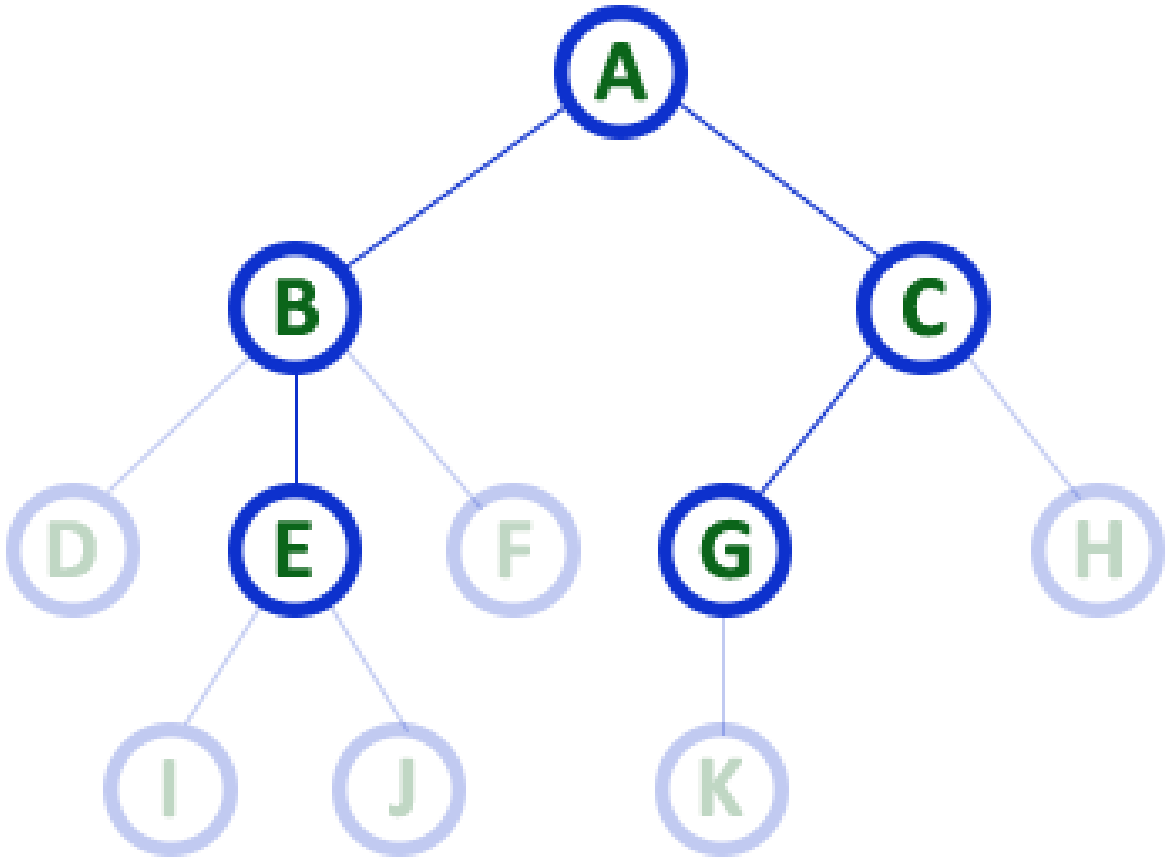


- In any tree, 'Edge' is a connecting link between two nodes.

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".

Example



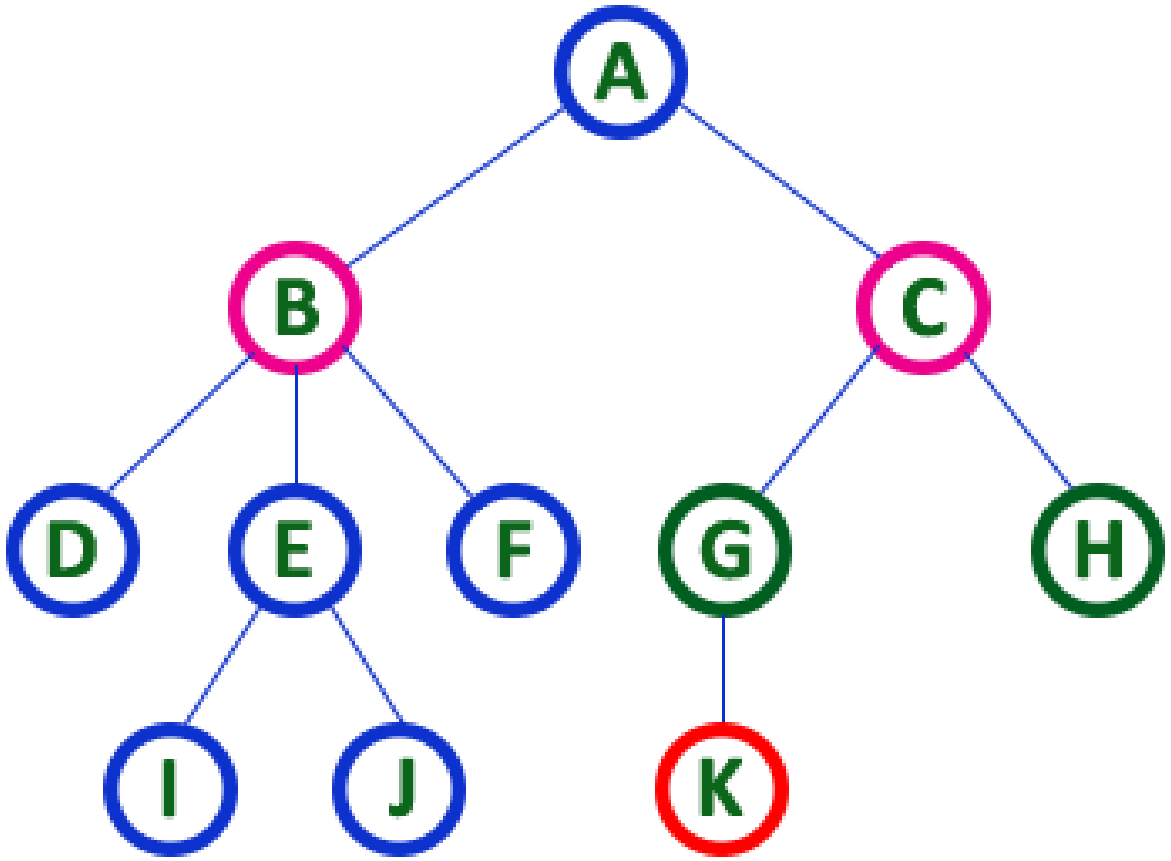
Here A, B, C, E & G are Parent nodes

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

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.

Example



Here B & C are Children of A

Here G & H are Children of C

Here K is Child of G

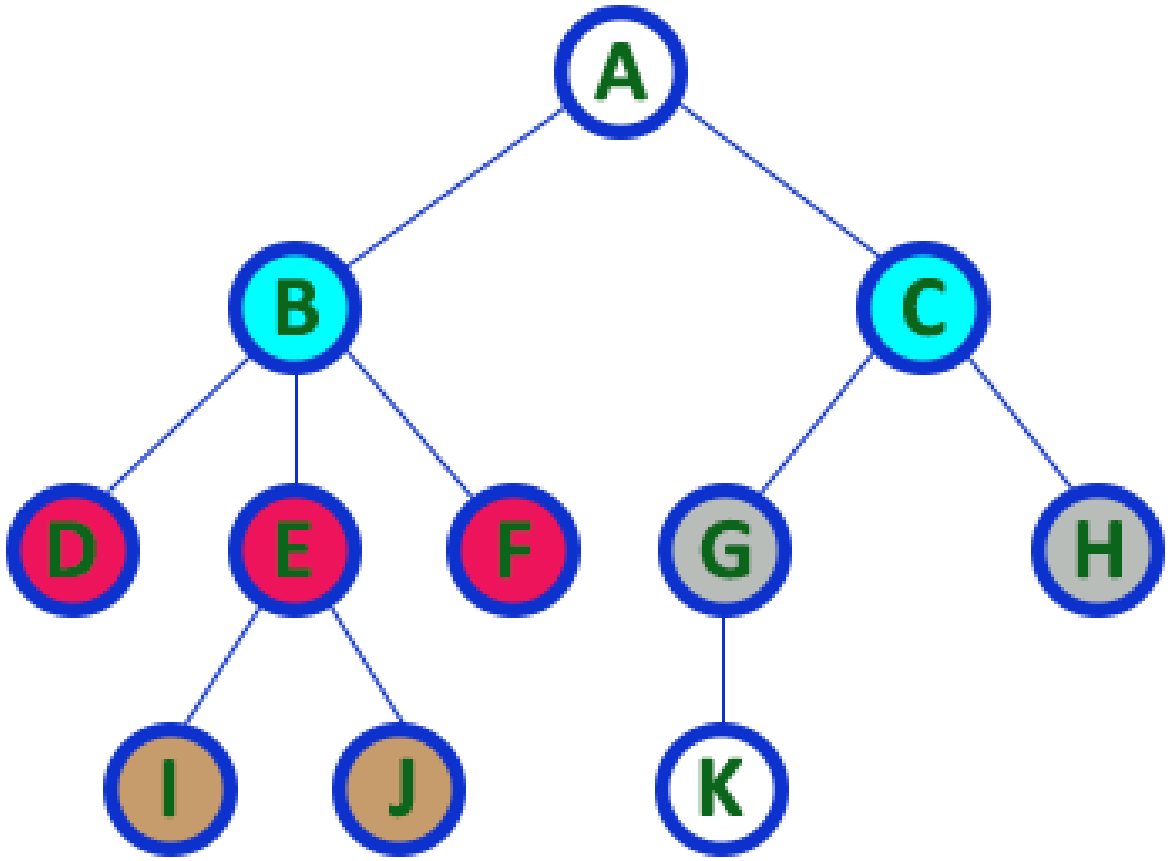
- descendant of any node is called as CHILD Node

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.

Example



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'

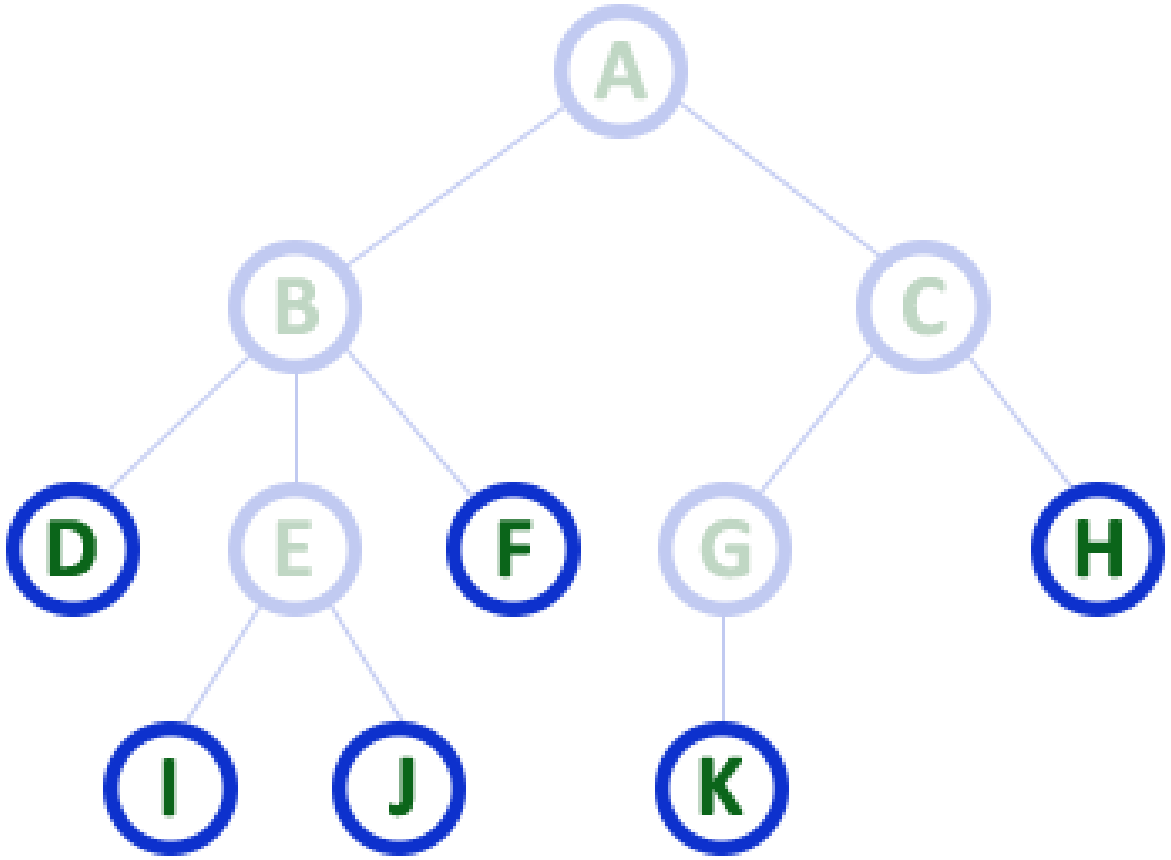
Leaf

- 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.

Example



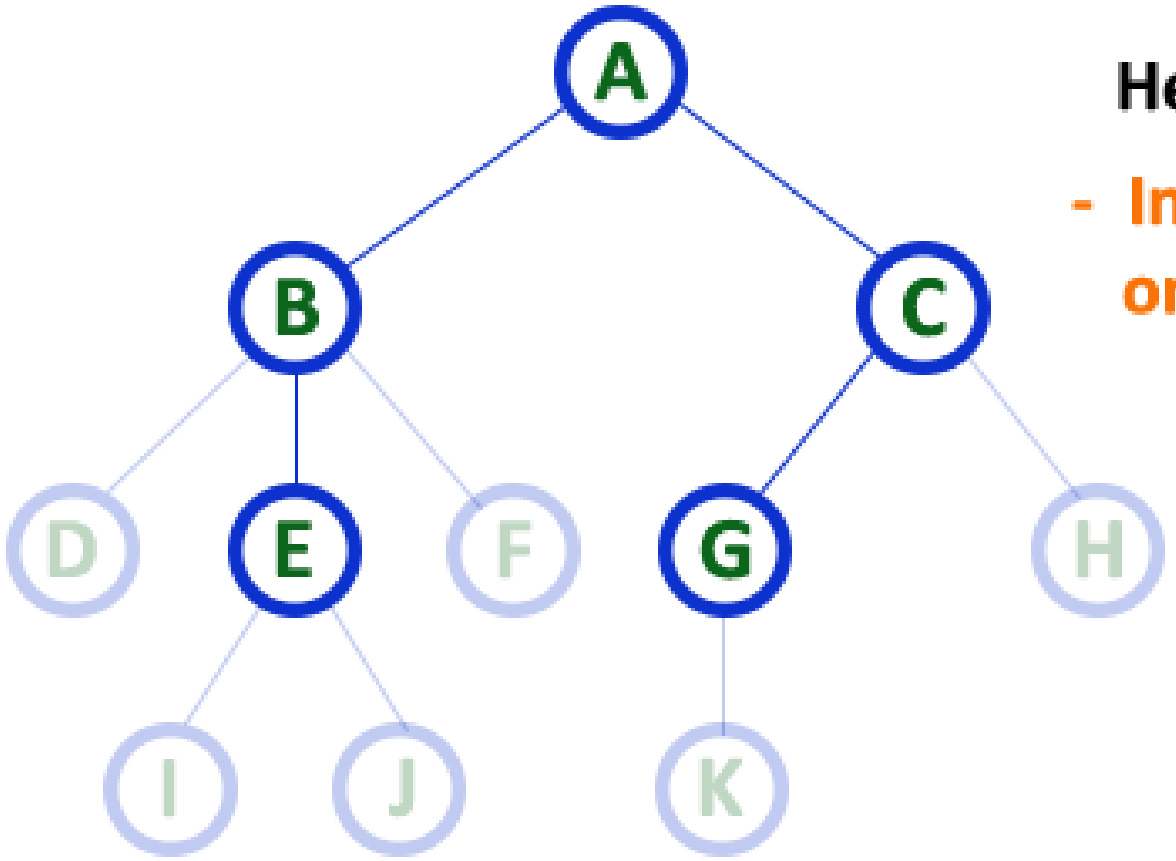
Here D, I, J, F, K & H are Leaf nodes

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

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.

Example



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

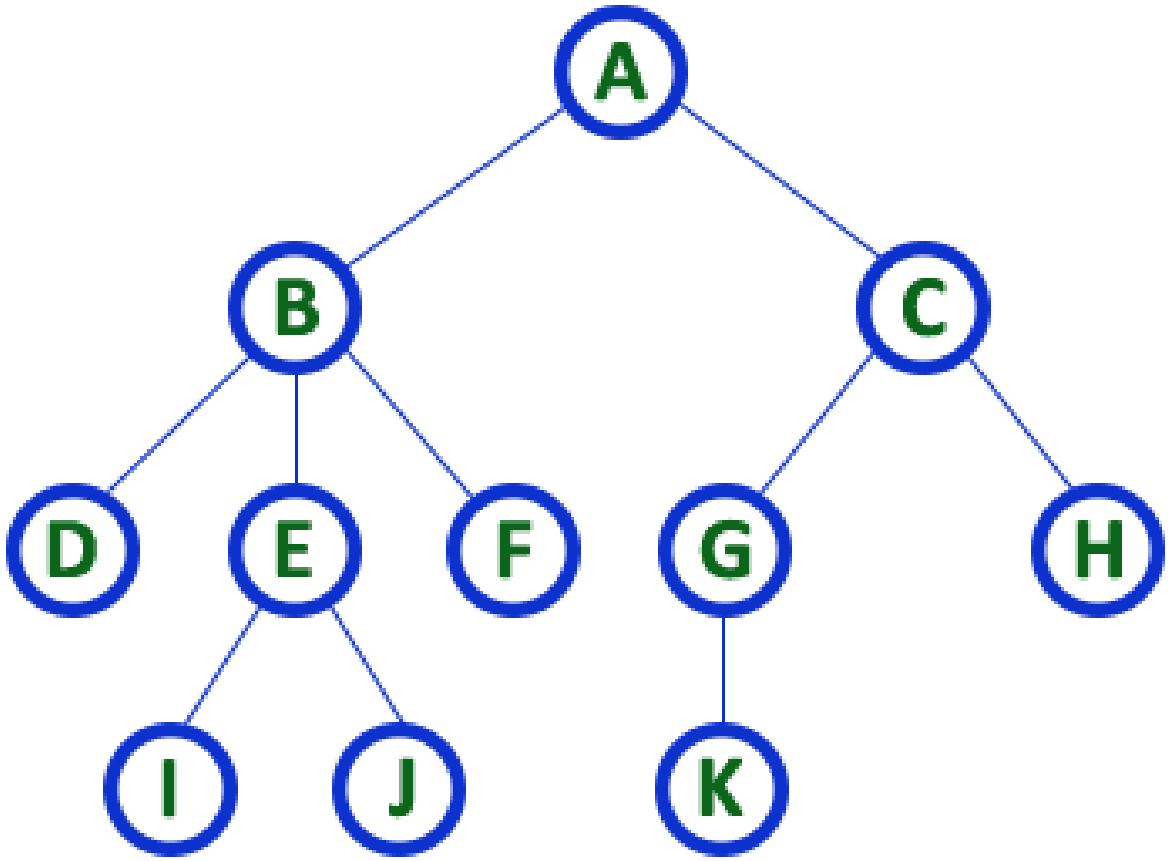
- **In any tree the node which has atleast one child is called 'Internal' node**

- **Every non-leaf node is called as 'Internal' node**

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'

Example



Here Degree of B is 3

Here Degree of A is 2

Here Degree of F is 0

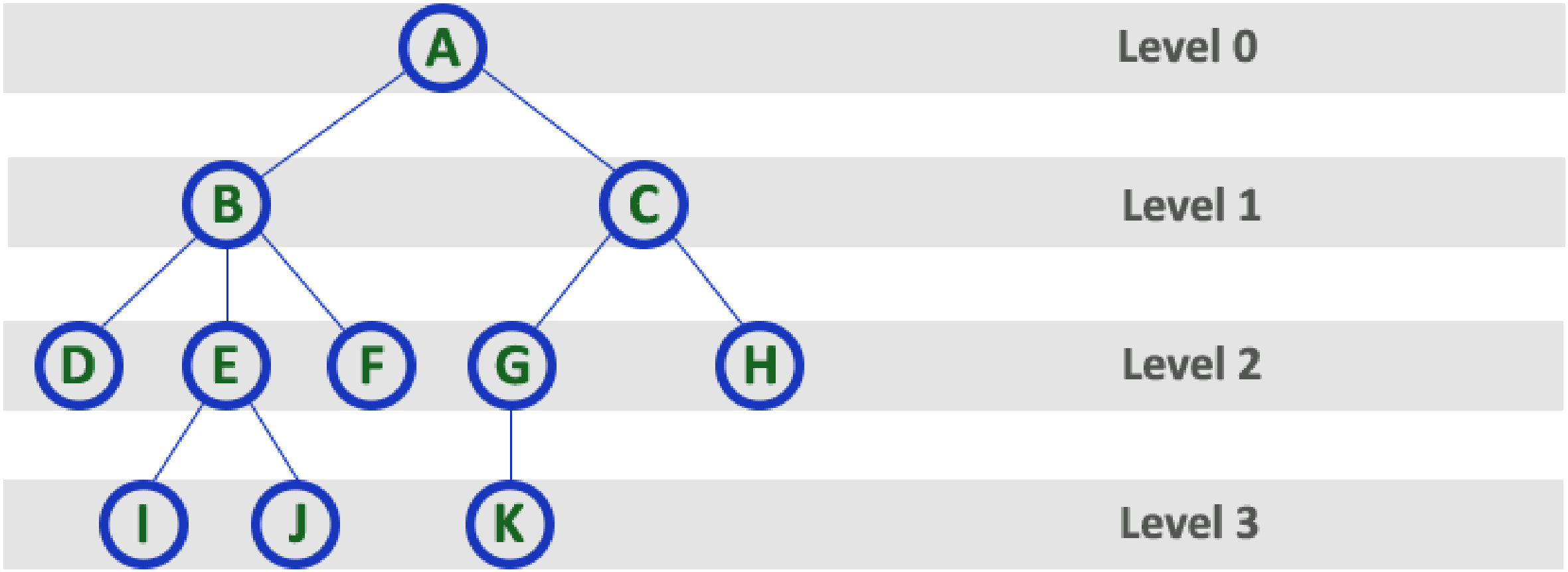
- In any tree, 'Degree' of a node is total number of children it has.

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).

Example

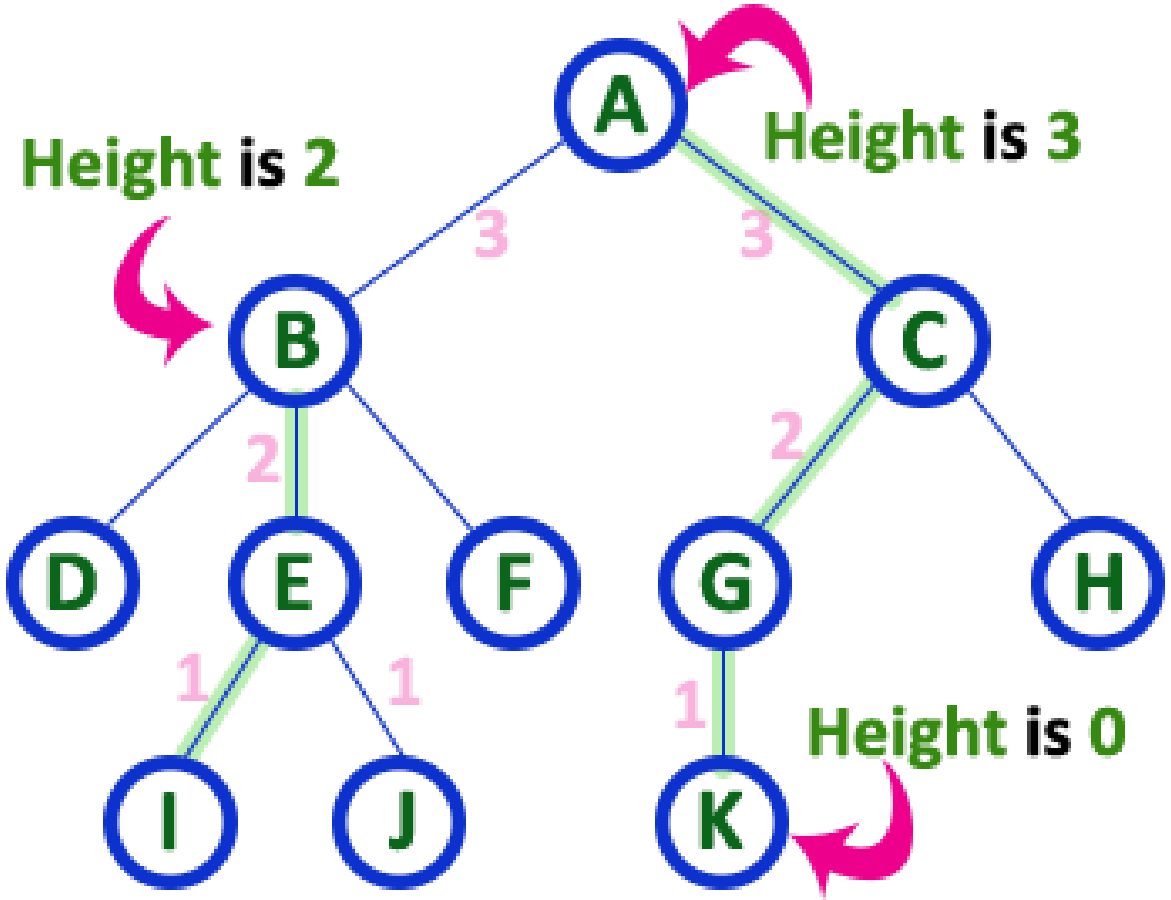


Height

- 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'.

Example



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.**

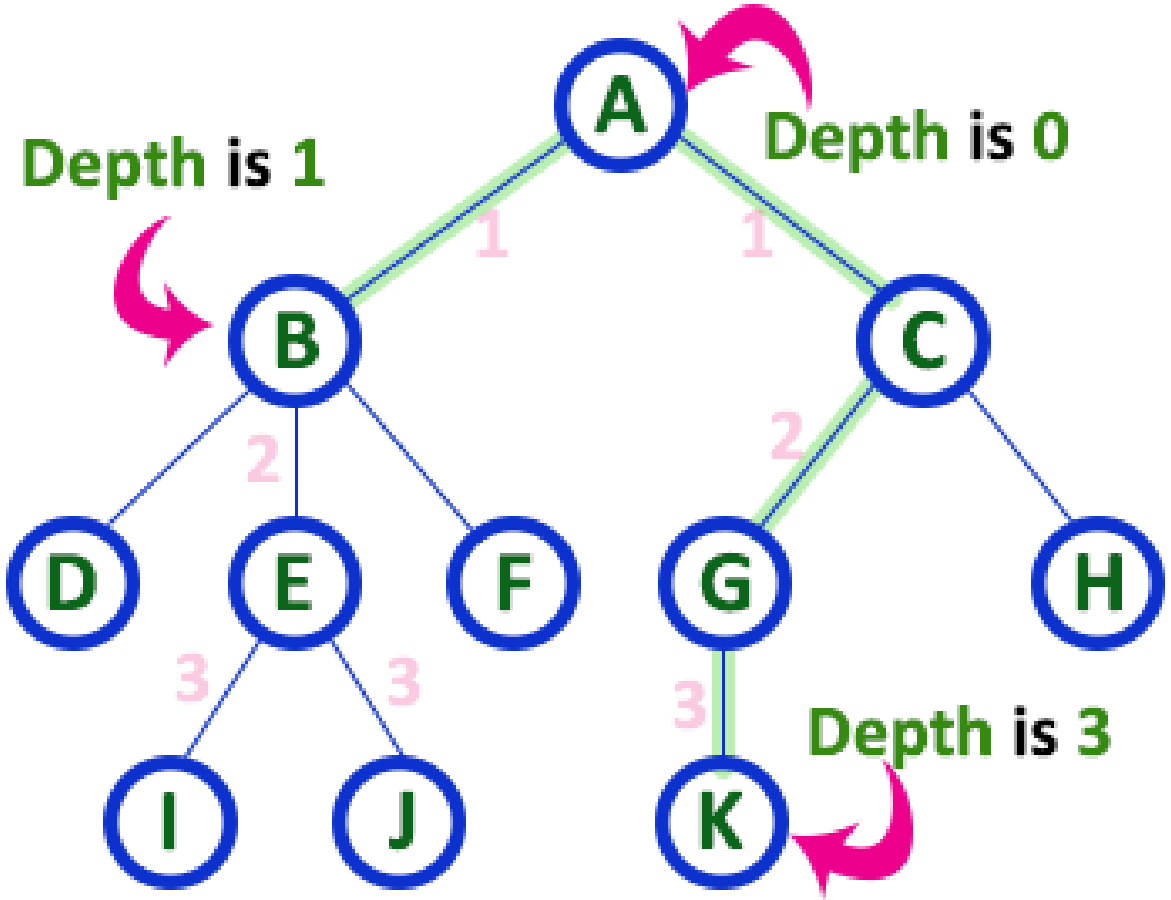
Depth

- 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'.

Example



Here Depth of tree is 3

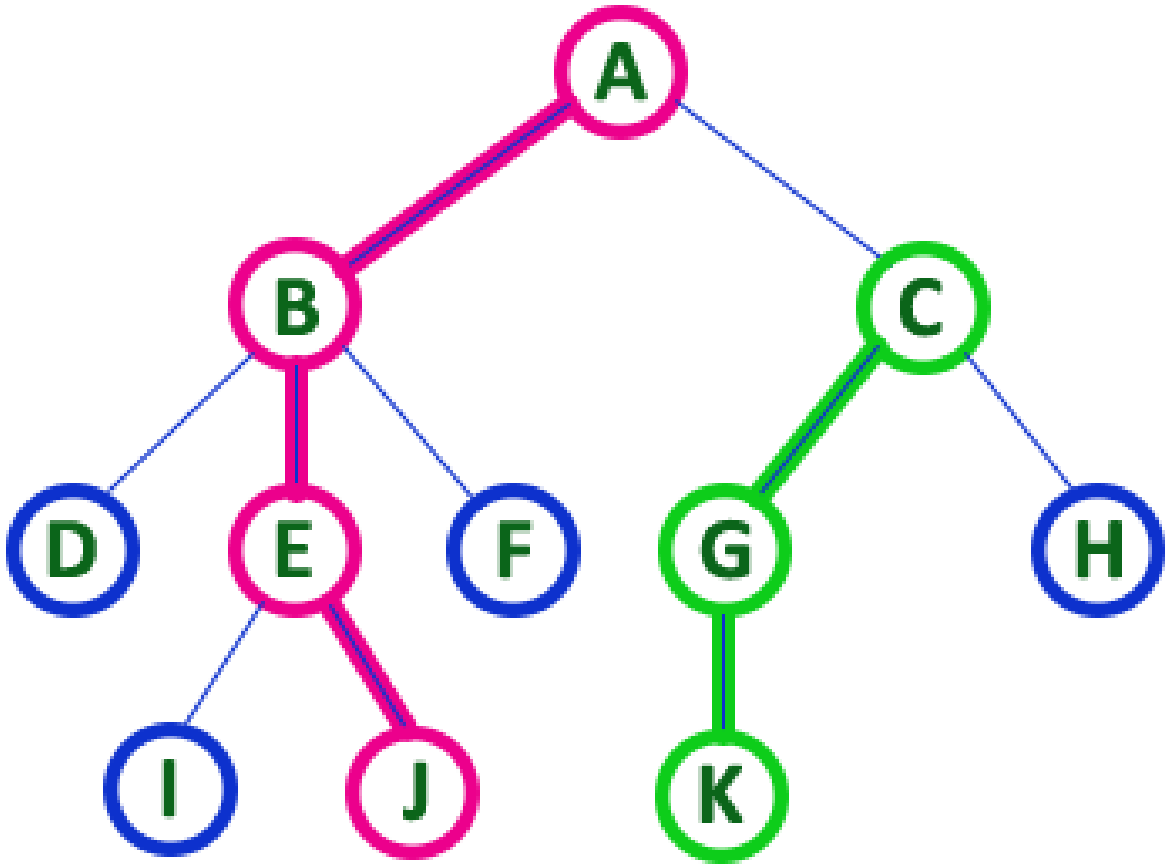
- In any tree, 'Depth of Node' is total number of Edges from root to that node.
- In any tree, 'Depth of Tree' is total number of edges from root to leaf in the longest path.

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.

Example



- 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

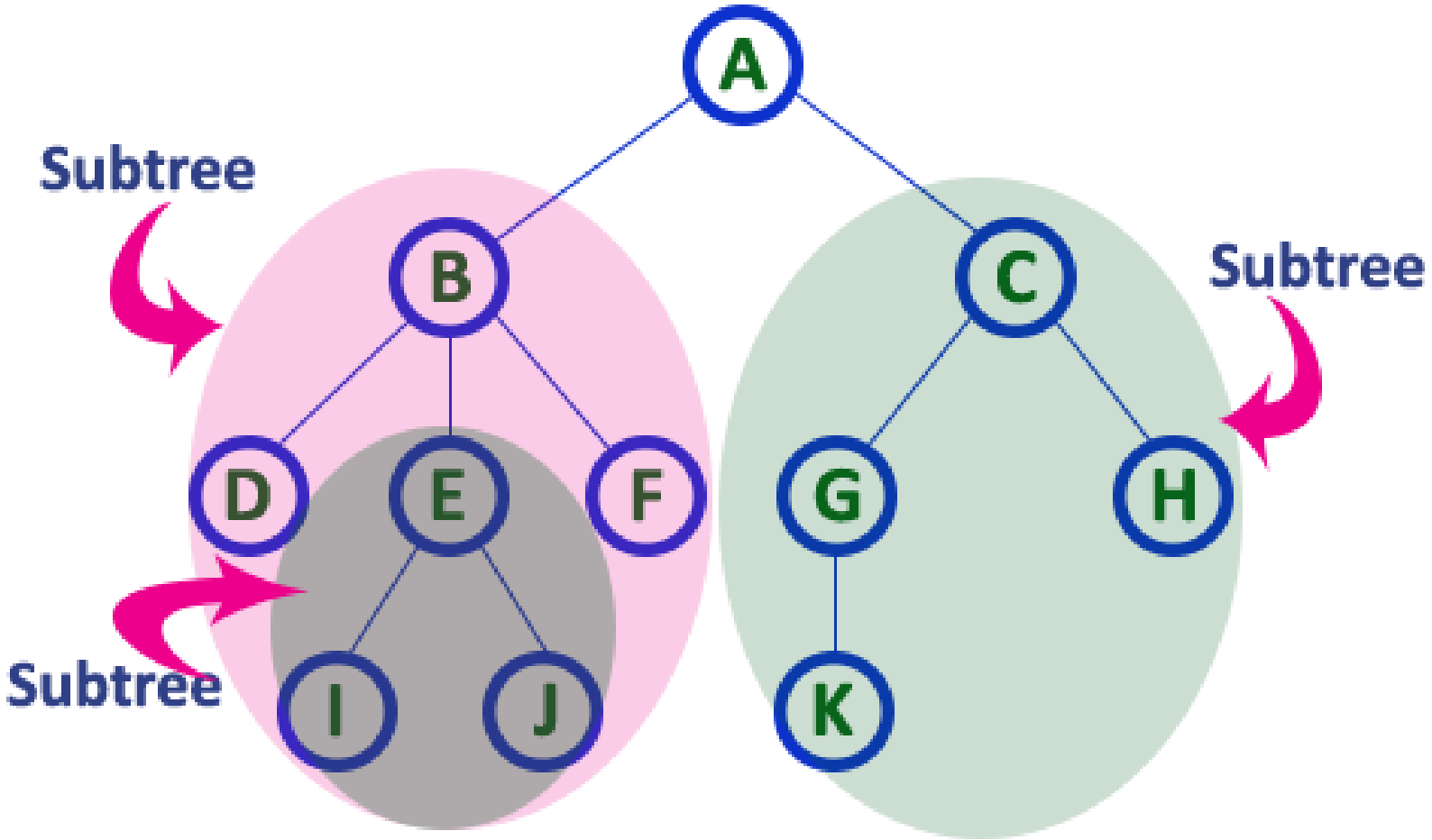
C - G - K

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.

Example



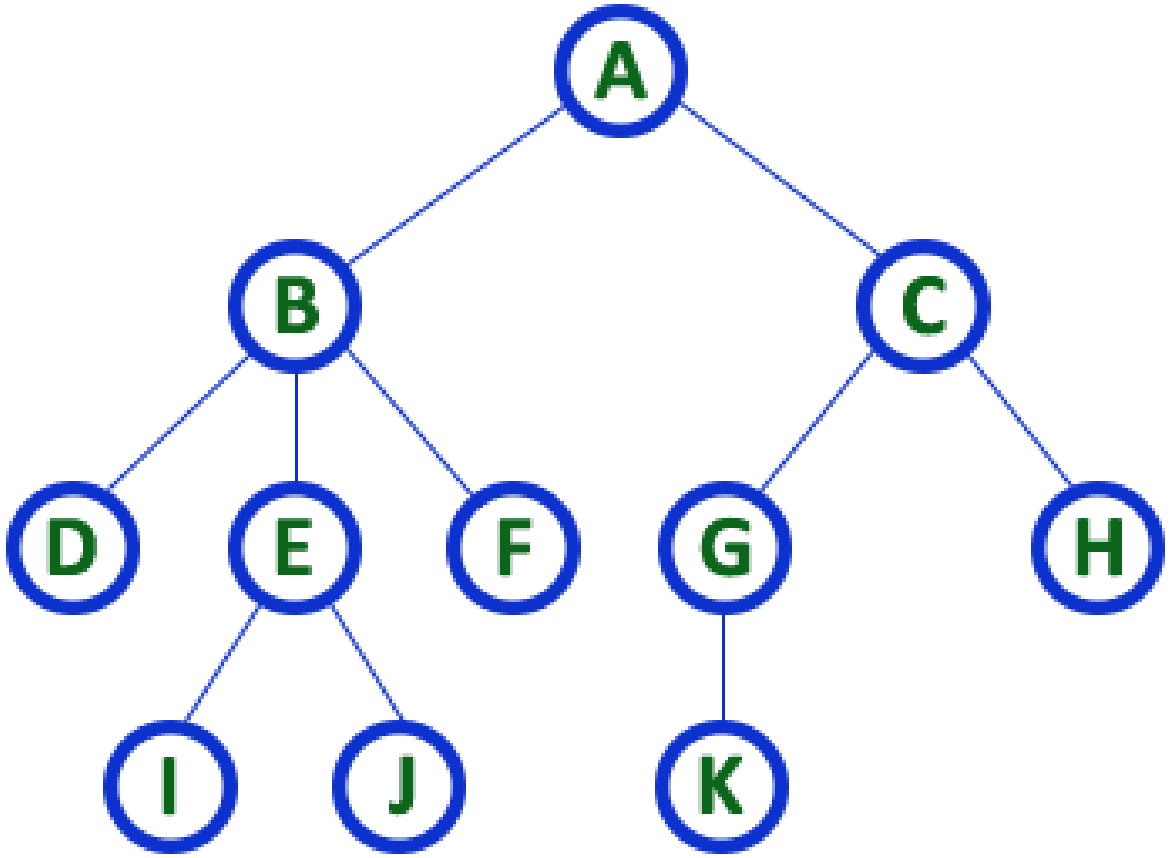
Tree Representation

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

1. List Representation

2. Left Child - Right Sibling Representation

Example



TREE with 11 nodes and 10 edges

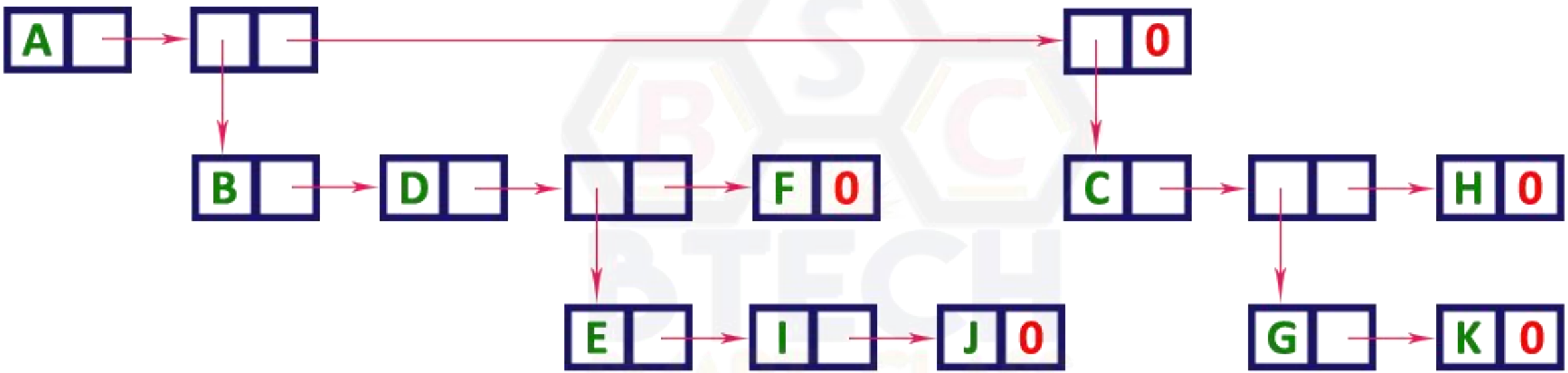
- **In any tree with 'N' nodes there will be maximum of 'N-1' edges**
- **In a tree every individual element is called as 'NODE'**

Linked List Representation

- In this representation, we use two types of nodes one for representing the node with data called 'data node' and another for representing only references called 'reference node’.

- We start with a 'data node' from the root node in the tree. Then it is linked to an internal node through a 'reference node' which is further linked to any other node directly. This process repeats for all the nodes in the tree.

Example

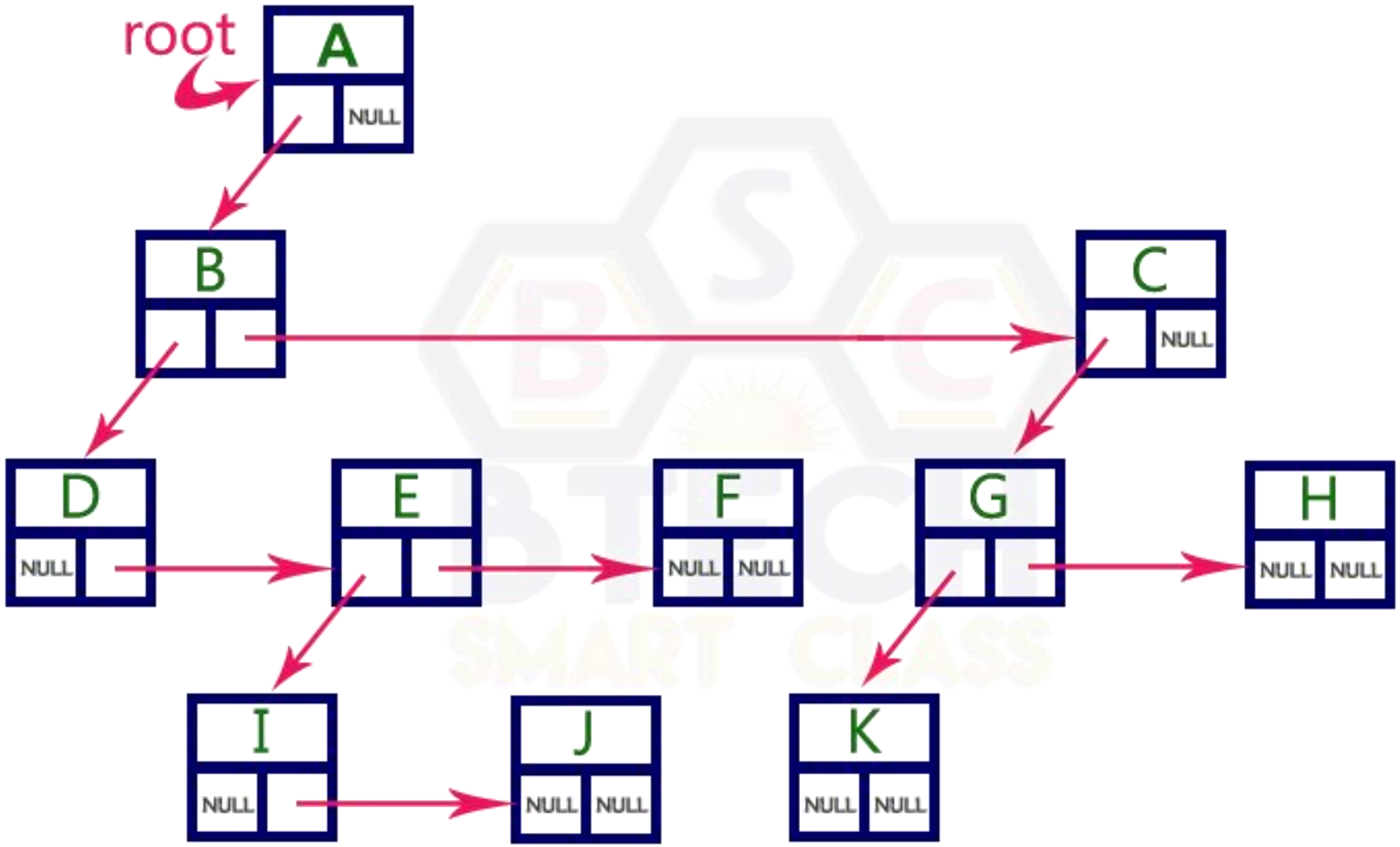


Left Child Right Sibling Representation

- In this representation, we use a 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.

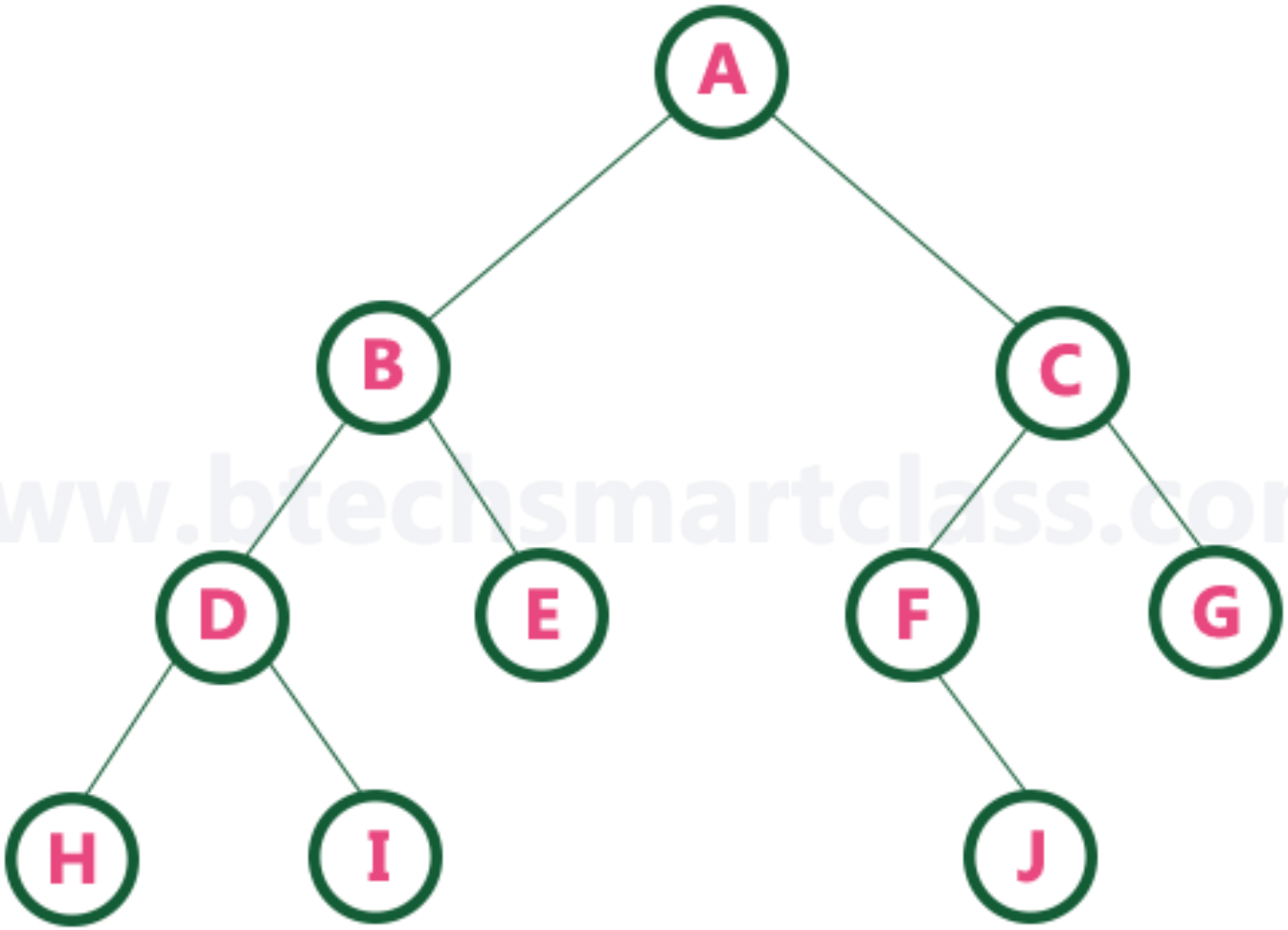
Example


root
A
NULL
B
C
NULL
D
NULL
E
F
NULL
NULL
G
H
NULL
NULL
I
NULL
J
NULL
NULL
K
NULL
NULL

Binary Tree

- 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.
- Definition: **A tree in which every node can have a maximum of two children is called Binary Tree.**

Example

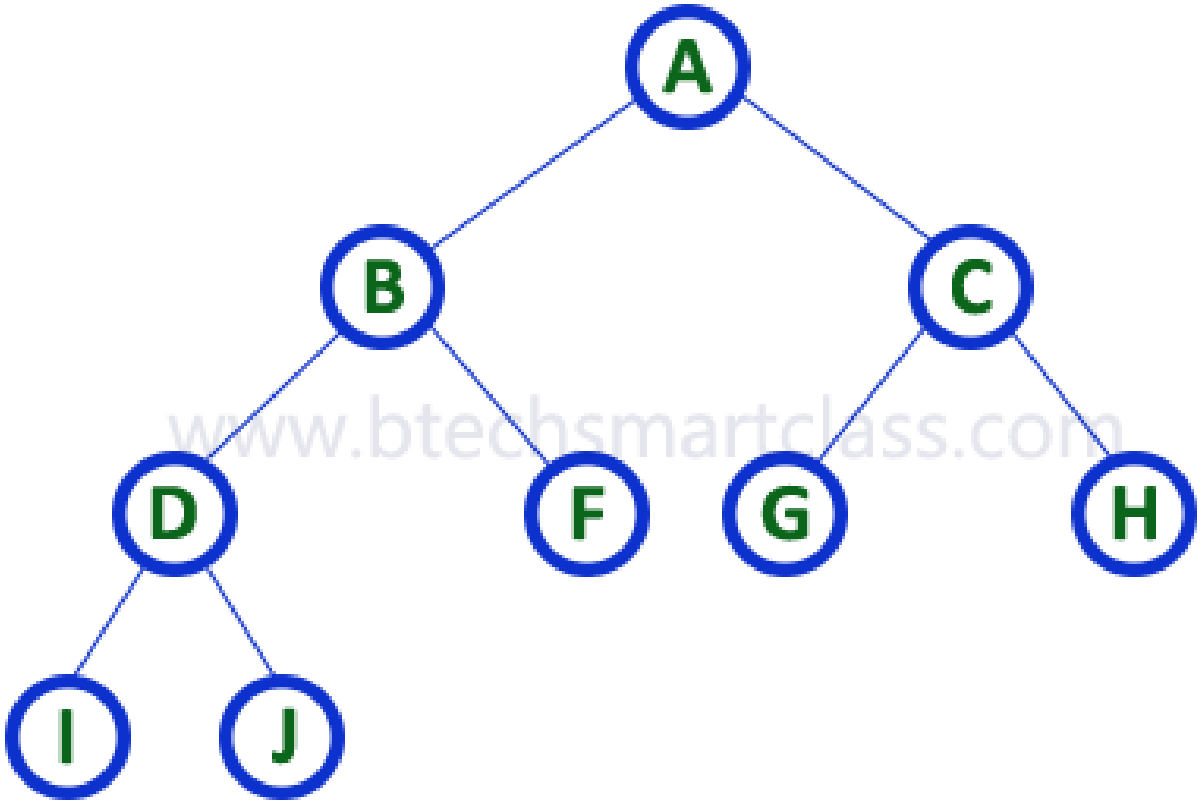


Strictly Binary Tree

- 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

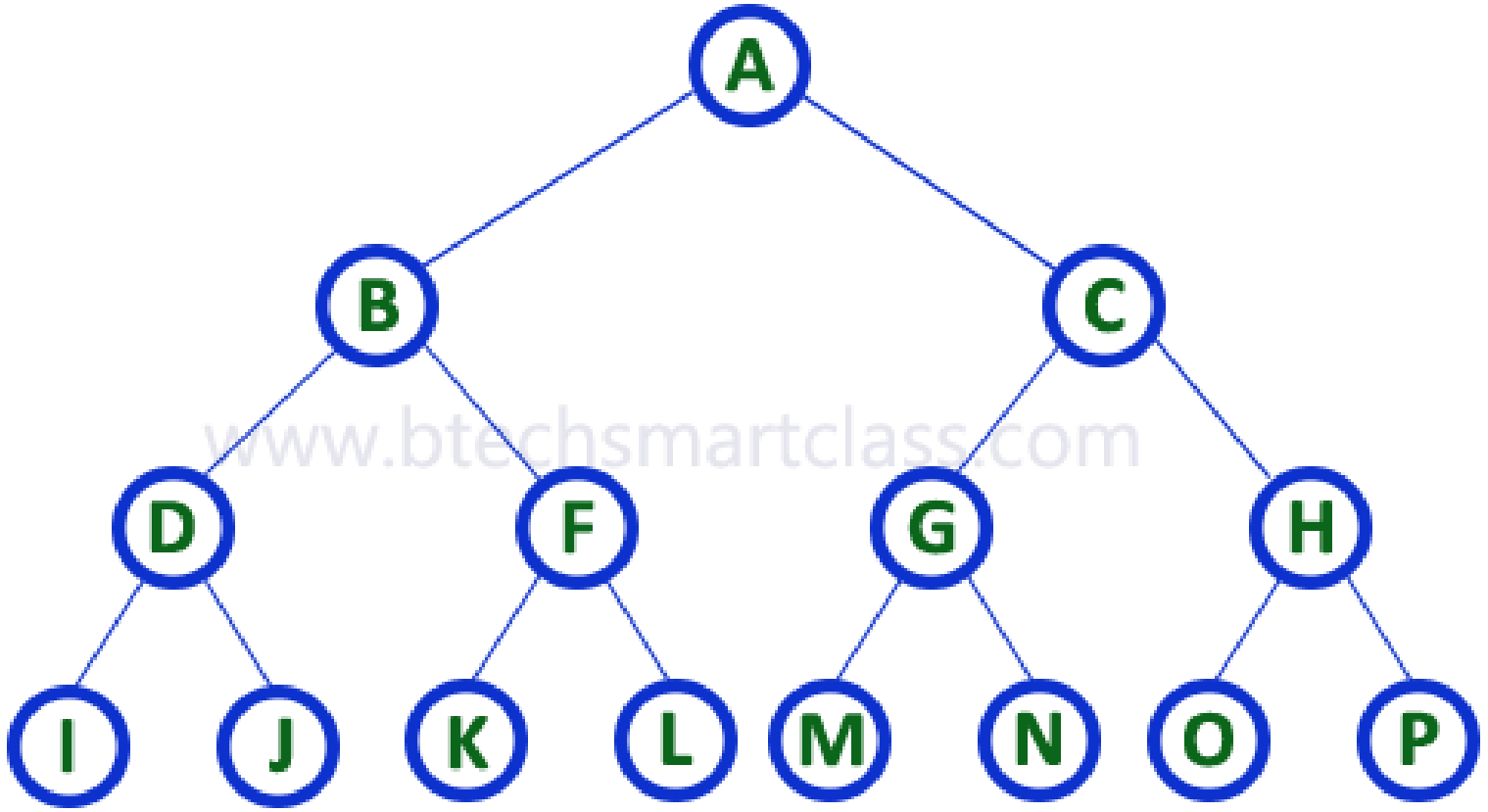
Example



Complete Binary Tree

- Definition: **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.**

Example



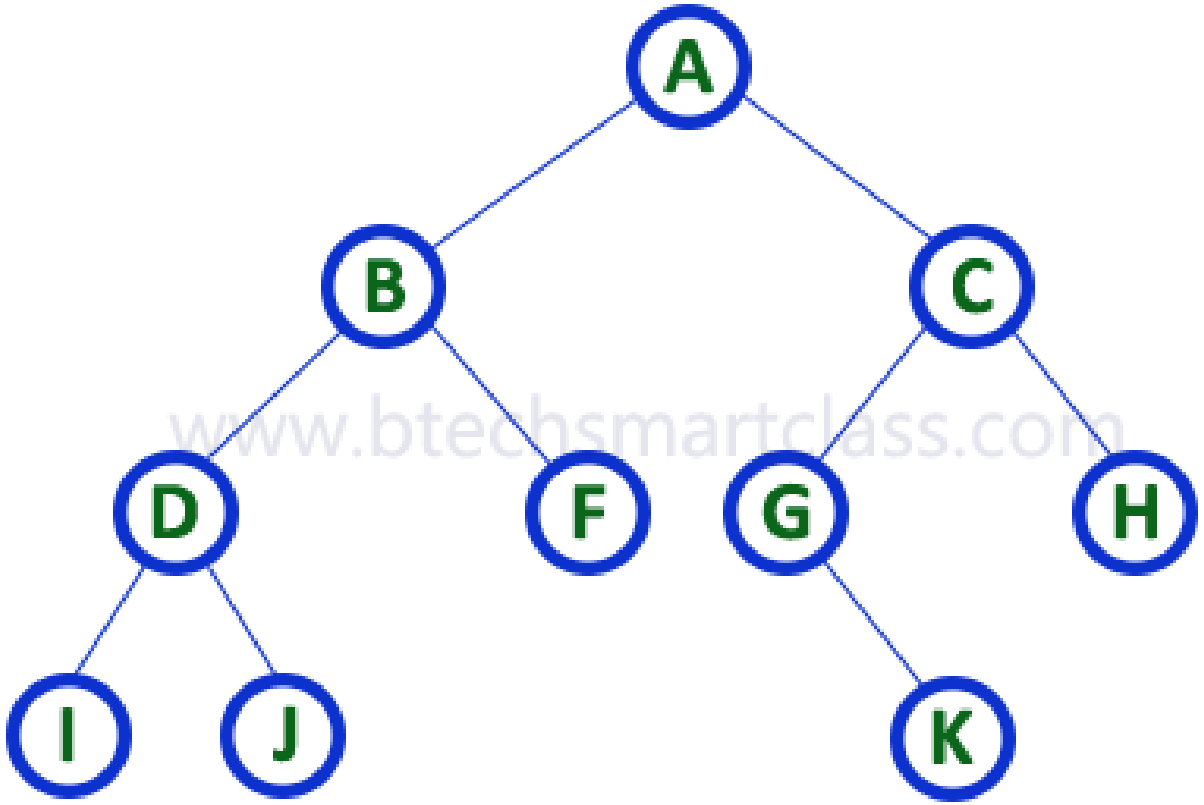
Binary Tree Representation

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

1. Array Representation

2. Linked List Representation

Example



Array Representation

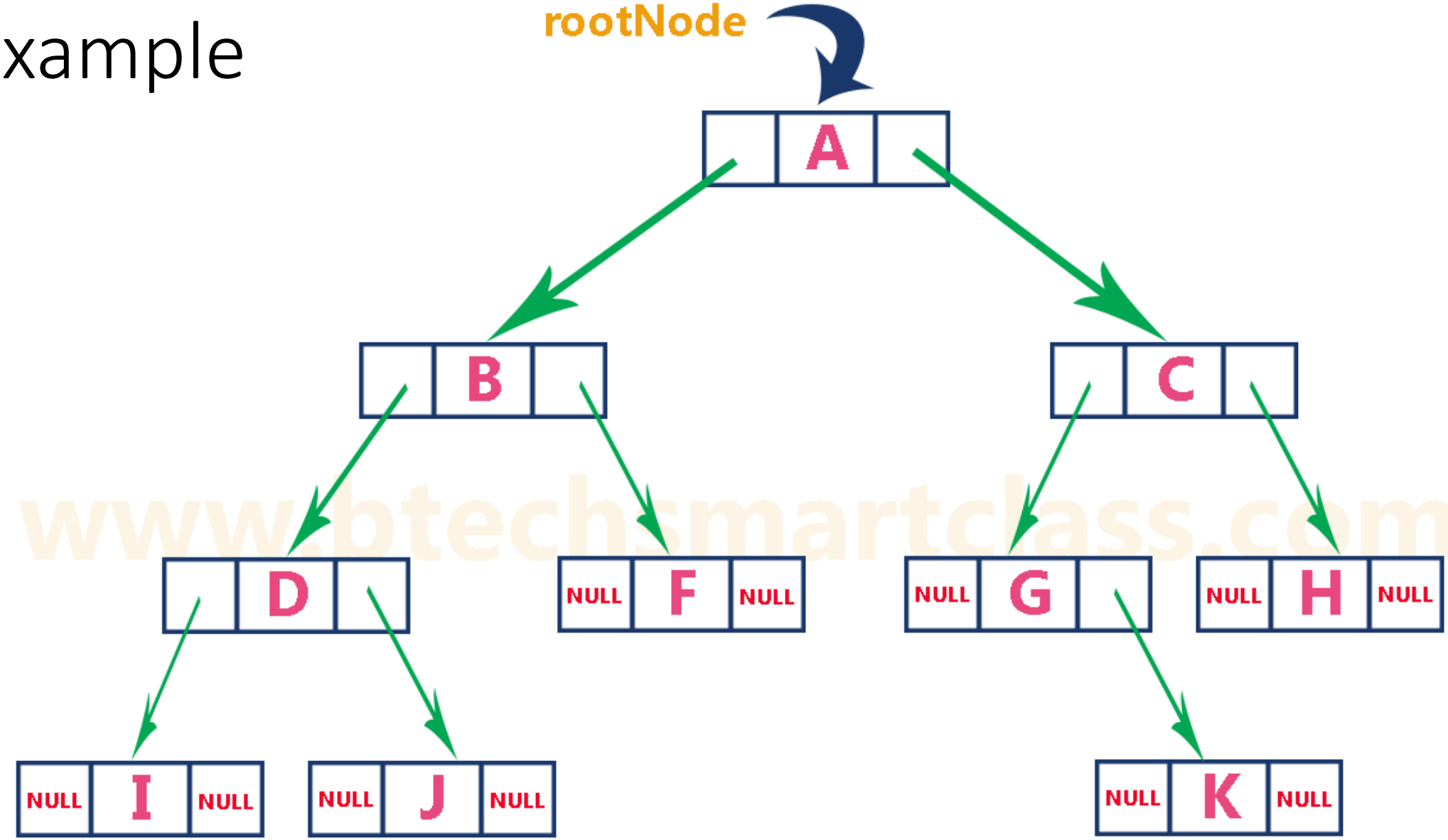
- To represent a binary tree of depth 'n' using array representation, we need one dimensional array with a maximum size of 2n + 1.

- If Root index is n, the left child index is 2*n + 1 and right child index is 2*n + 2.

Linked list Representation

- 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.

Example


rootNode
A
B
C
D
NULL
F
NULL
NULL
G
NULL
NULL
H
NULL
NULL
I
NULL
NULL
J
NULL
NULL
K
NULL

Binary Tree Traversal

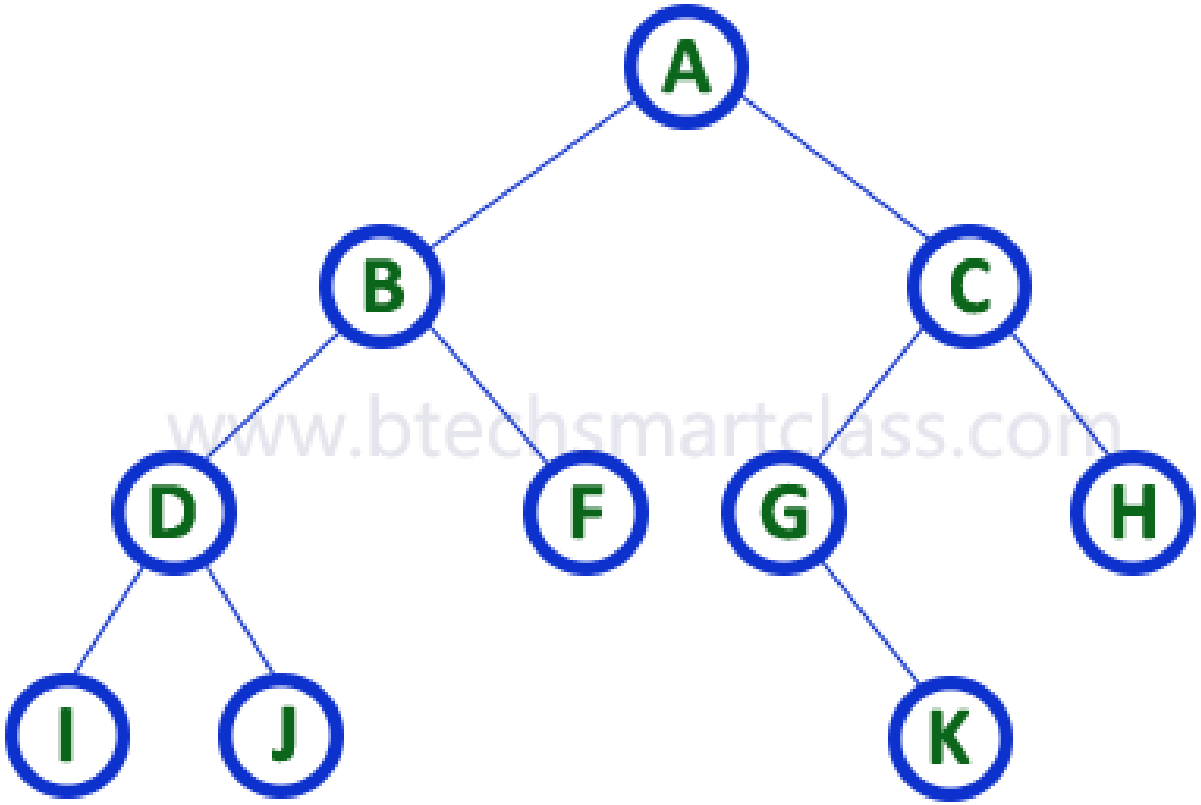
- When we wanted to display a binary tree, we need to follow some order in which all the nodes of that binary tree must be displayed.

- In any binary tree, displaying order of nodes depends on the traversal method.

- There are three types of binary tree traversals.

1. In - Order Traversal
2. Pre - Order Traversal
3. Post - Order Traversal

Example



In-Order (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.

Example

- In-Order Traversal for above example of binary tree is

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

Explanation

- 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.

Explanation (Contd)

- 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.

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.

Example

- Pre-Order Traversal for above example binary tree is

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

Explanation

- 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’.

Explanation

- 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.

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.

Example

- Post-Order Traversal for above example binary tree is

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

Binary Tree Construction

- Let us consider the below traversals:

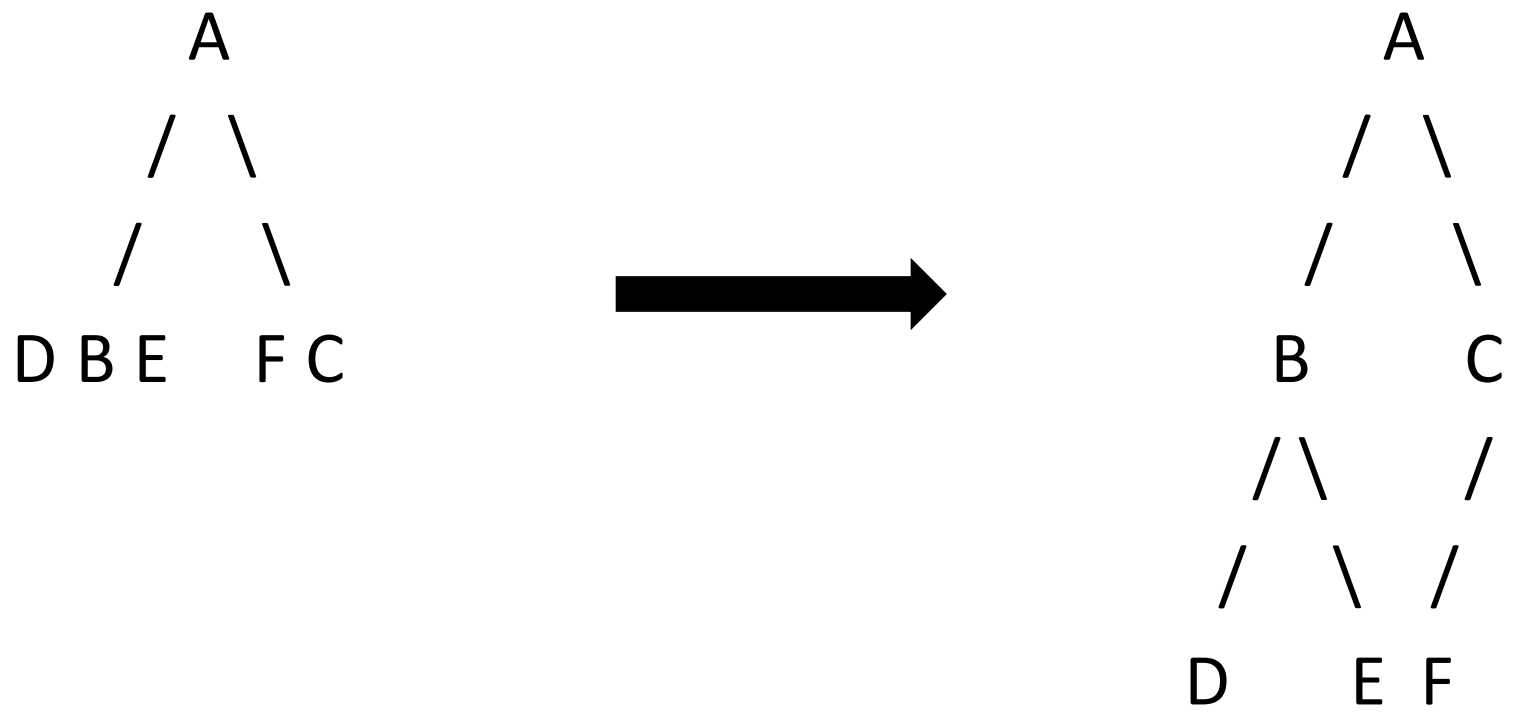
Inorder sequence: D B E A F C

Preorder sequence: A B D E C F

Algorithm

1. Pick an element from Preorder. Increment a Preorder Index Variable (preIndex in below code) to pick the next element in the next recursive call.
2. Create a new tree node tNode with the data as the picked element.
3. Find the picked element's index in Inorder. Let the index be inIndex.
4. Call buildTree for elements before inIndex and make the built tree as a left subtree of tNode.
5. Call buildTree for elements after inIndex and make the built tree as a right subtree of tNode.
6. return tNode.

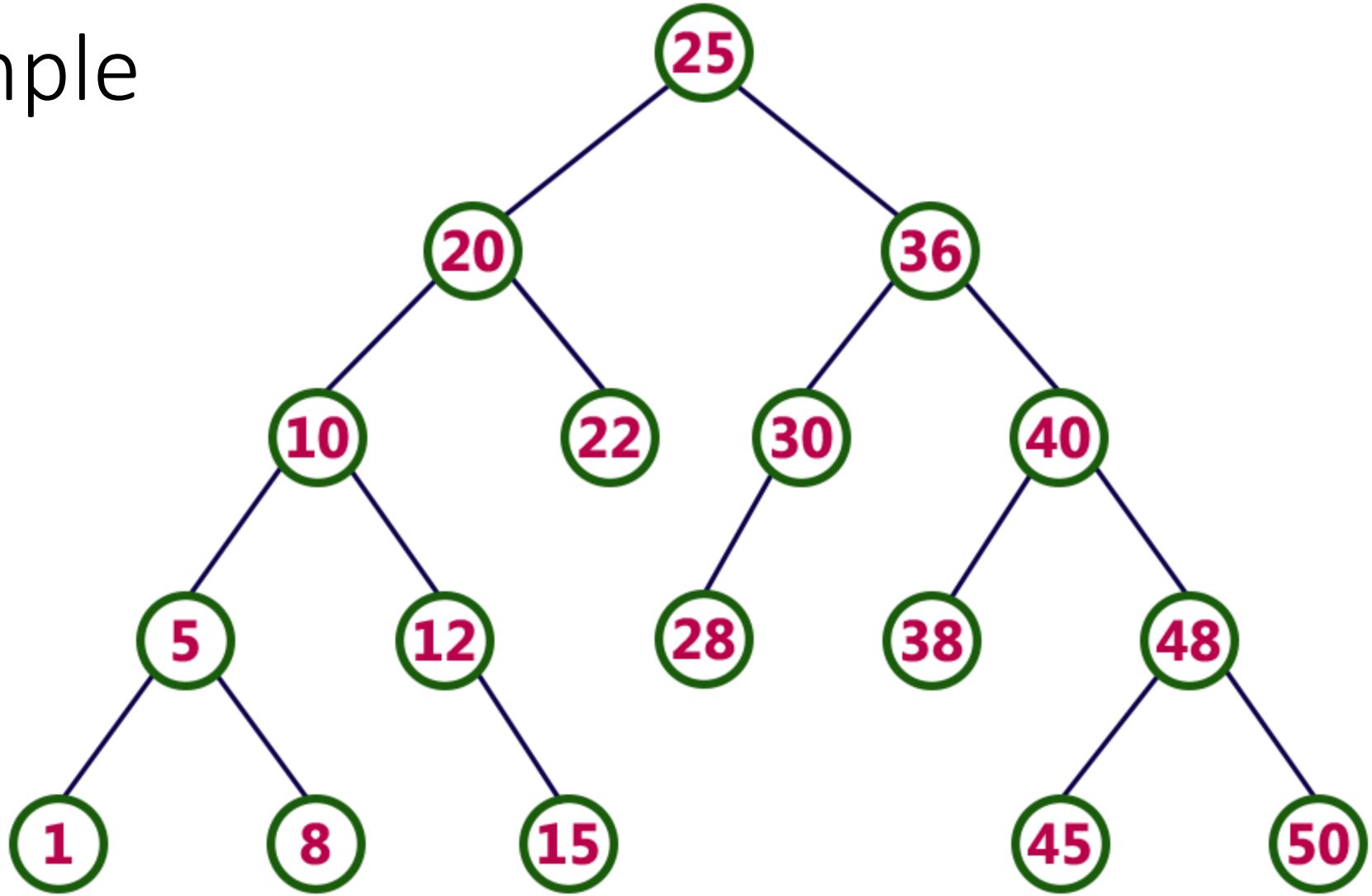
Example



Binary Search Tree (BST)

- Definition: **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.**

Example



Operations

- Search
- Insertion
- Deletion

Search Operation

- **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 matched, then display "Given node is found!!!" and terminate the function
- **Step 4 -** If both are not matched, 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 find the exact element or until the search element is compared with the leaf node
- **Step 8 -** If we reach to the node having the value equal to the search value then display "Element is found" and terminate the function.
- **Step 9 -** If we reach to the leaf node and if it is also not matched with the search element, then display "Element is not found" and terminate the function.

Insertion Operation

- **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 **root** to **newNode**.
- **Step 4 -** If the tree is **Not Empty**, then check whether the 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 steps until we reach to the **leaf** node (i.e., reaches to NULL).
- **Step 7 -** After reaching the leaf node, insert the newNode as **left child** if the newNode is **smaller or equal** to that leaf node or else insert it as **right child**.

Deletion Operation

- 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

- 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

- 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 node and child node.

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

Case 3

- 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 is found in the above step.

Step 4 - Then check whether deleting node came to case 1 or case 2 or else goto step 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.

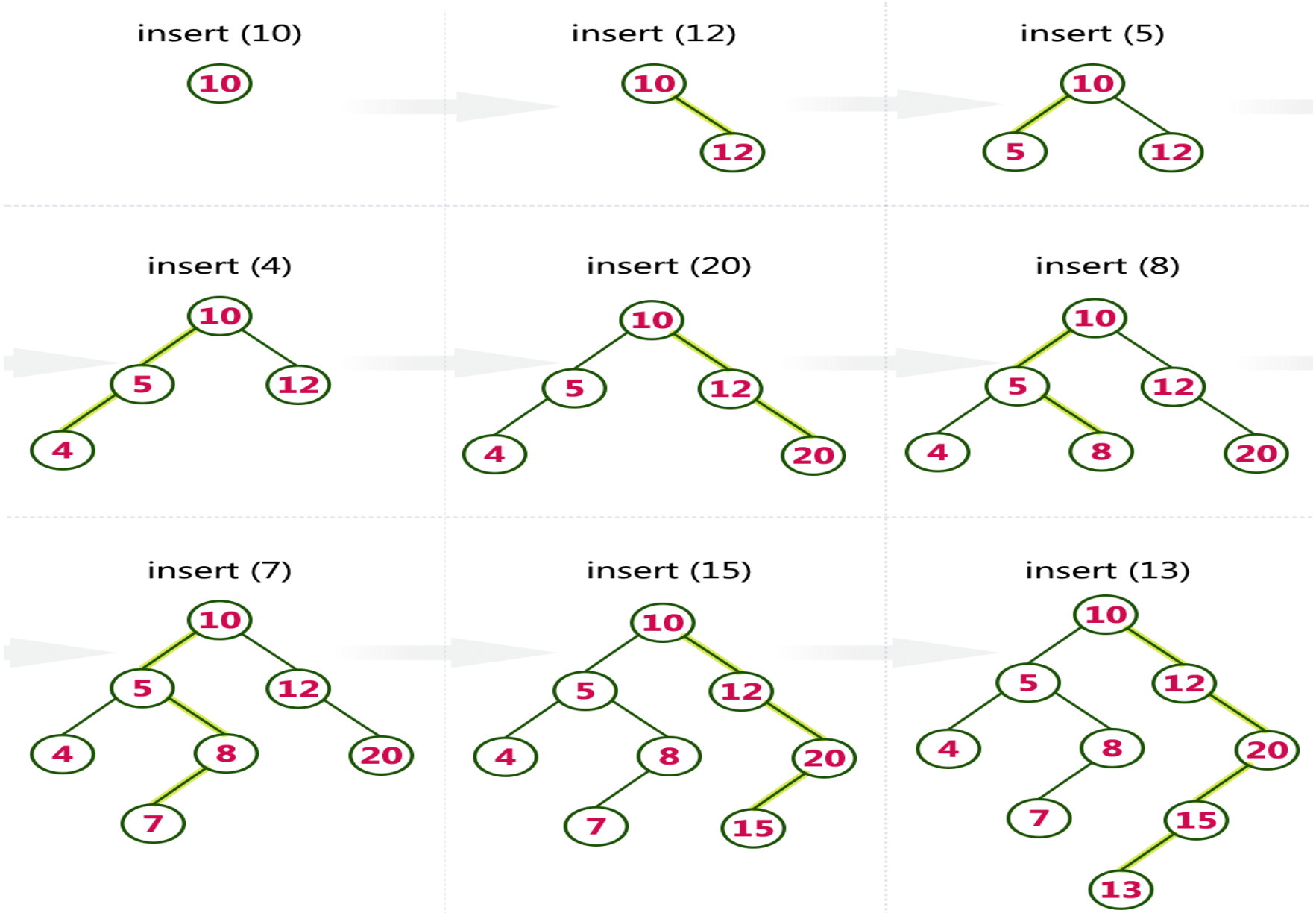
Step 7 - Repeat the same process until the node is deleted from the tree.

Construct BST

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

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

Solution


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