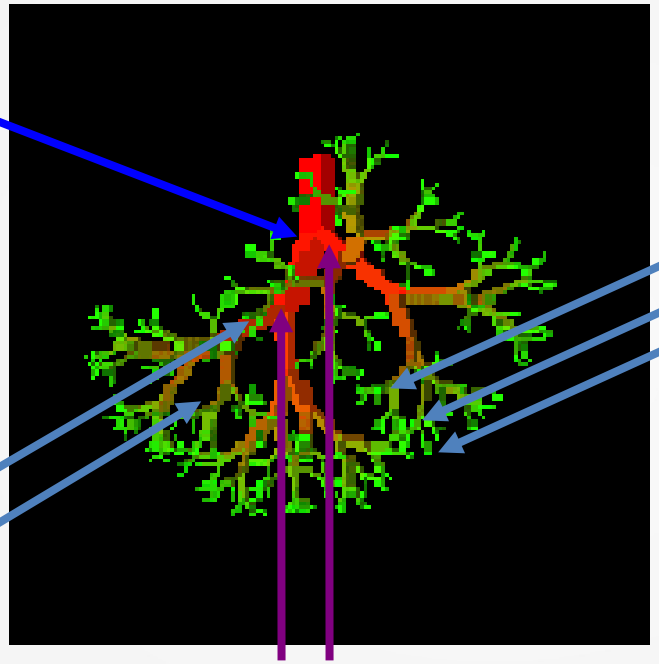
GLS UNIVERSITY

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

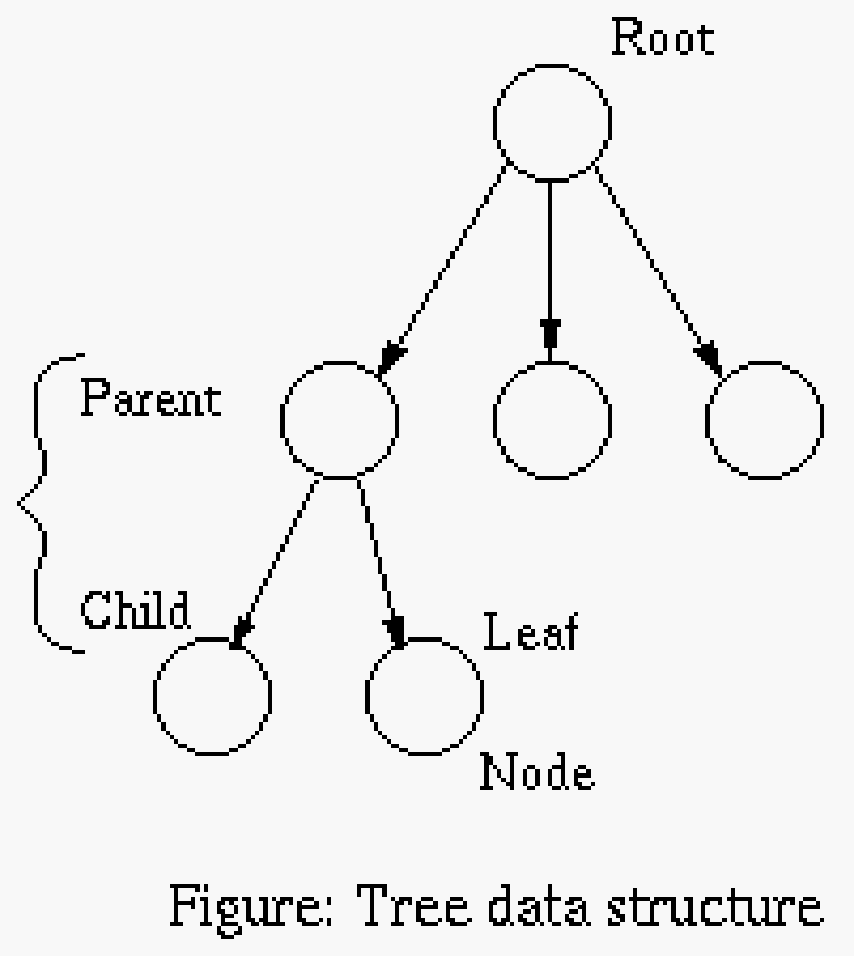
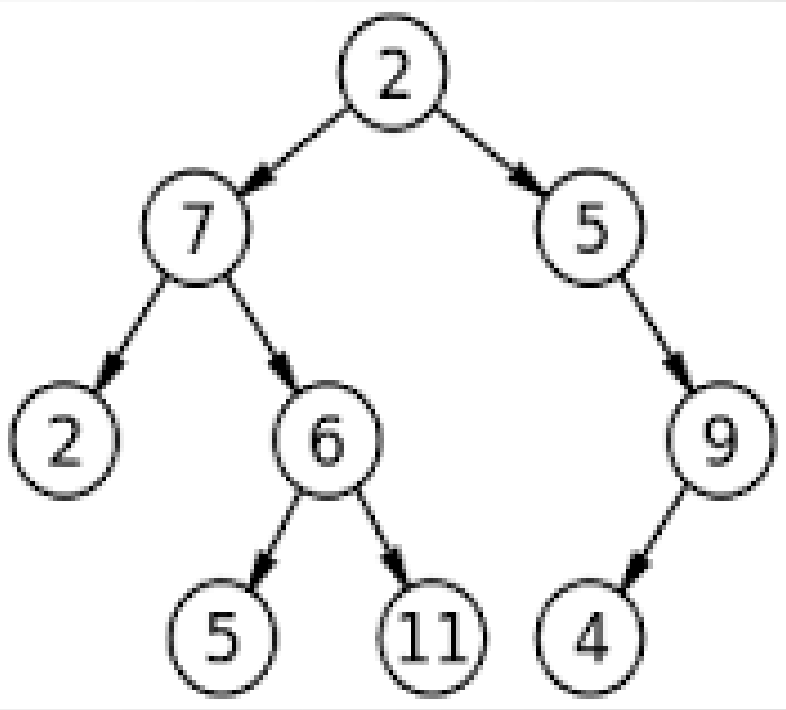


Figure: Tree data structure



Applications of Tree

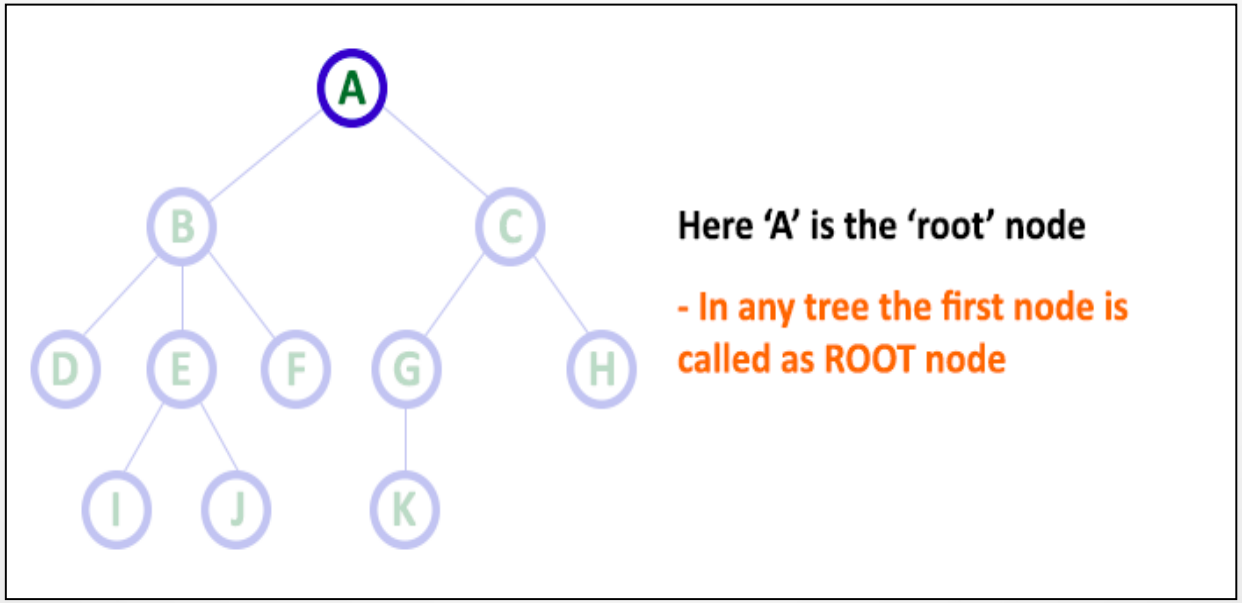
- Company Organization structures
- File Systems

```
  /   <-- root
  /      \
...        home
      /          \
   ugrad        course
    /          /    |    \
  ...        cs101 cs112 cs113
```

- Manipulate sorted lists of data.
- Networking

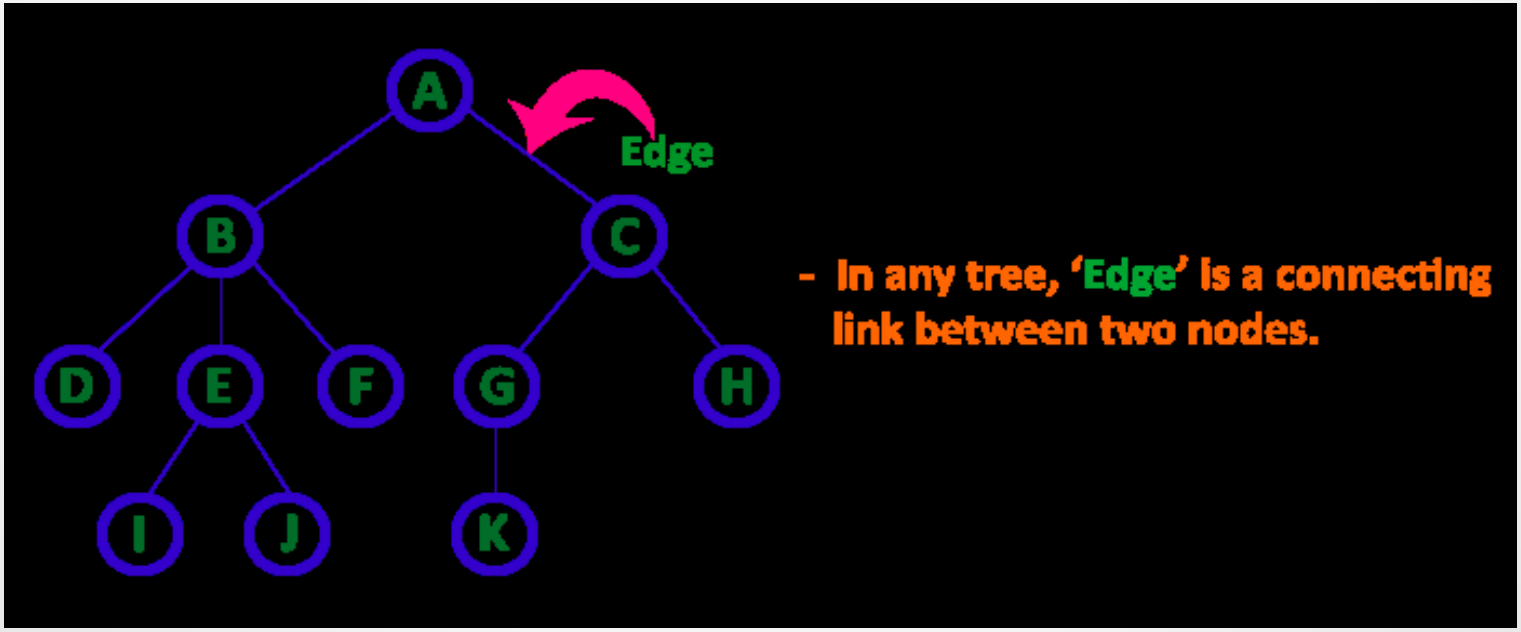
TREE TERMINOLOGY

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.



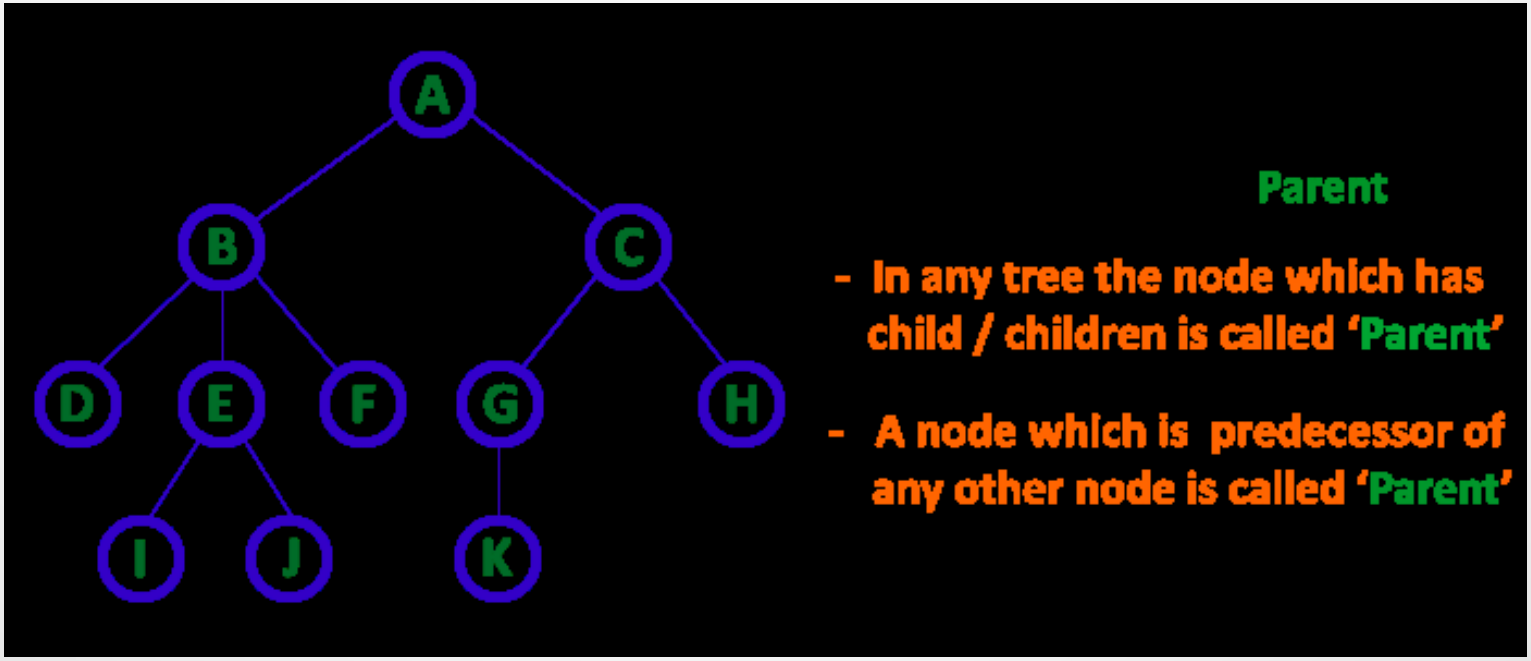
TREE TERMINOLOGY

- **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.



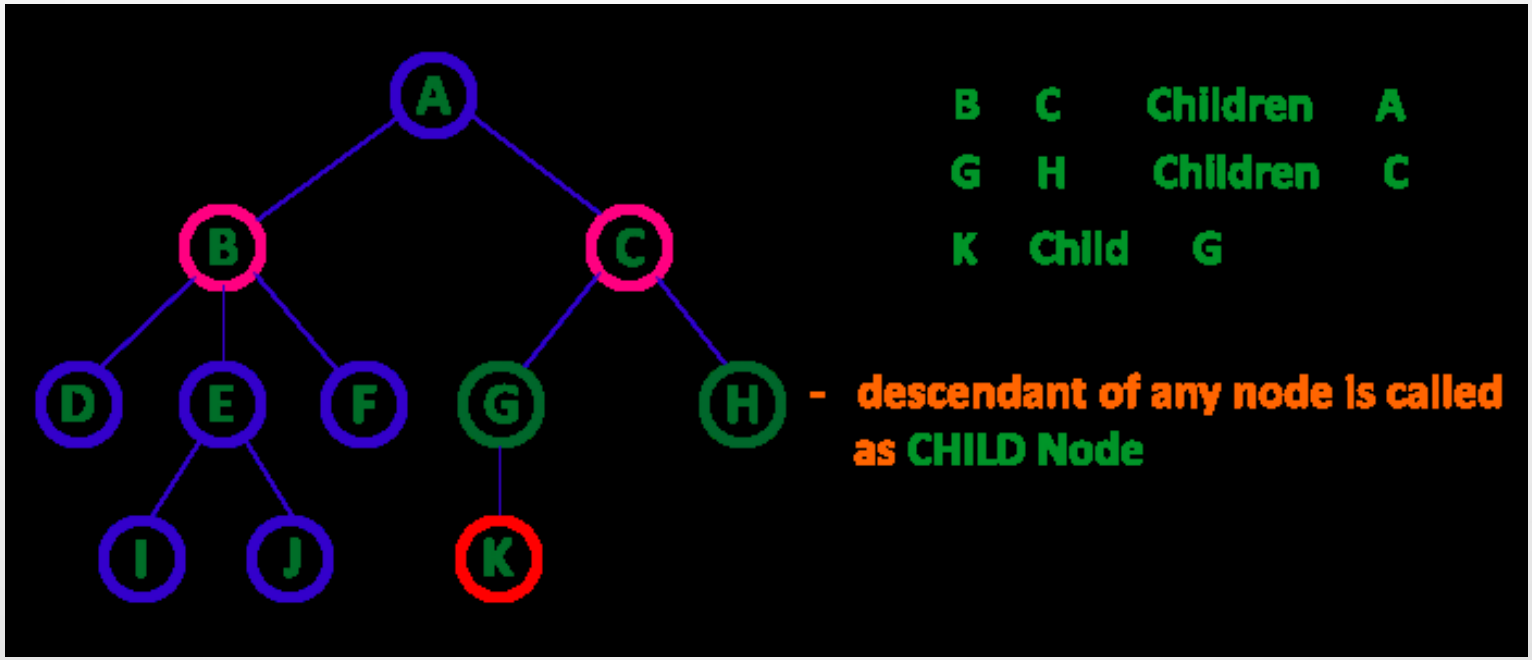
TREE TERMINOLOGY

- **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".



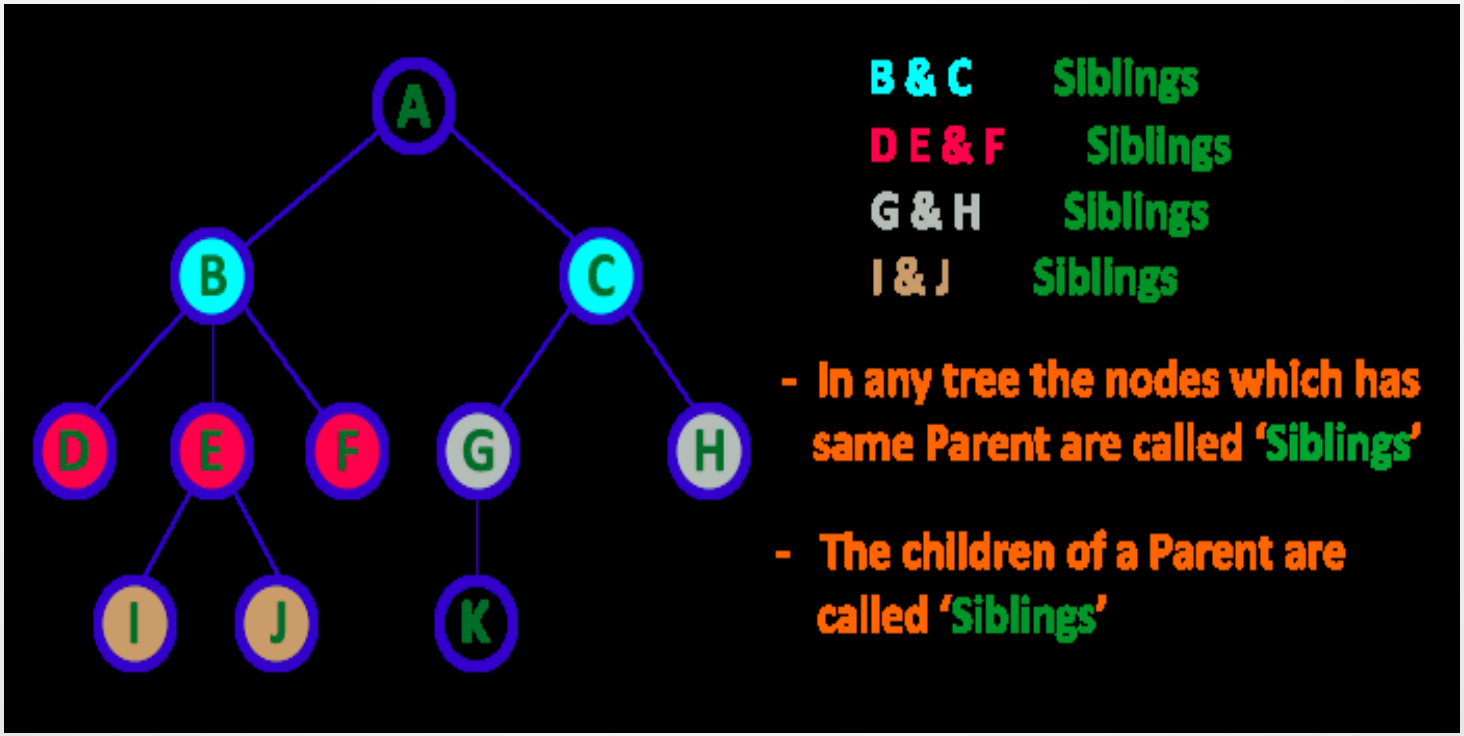
TREE TERMINOLOGY

- **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.



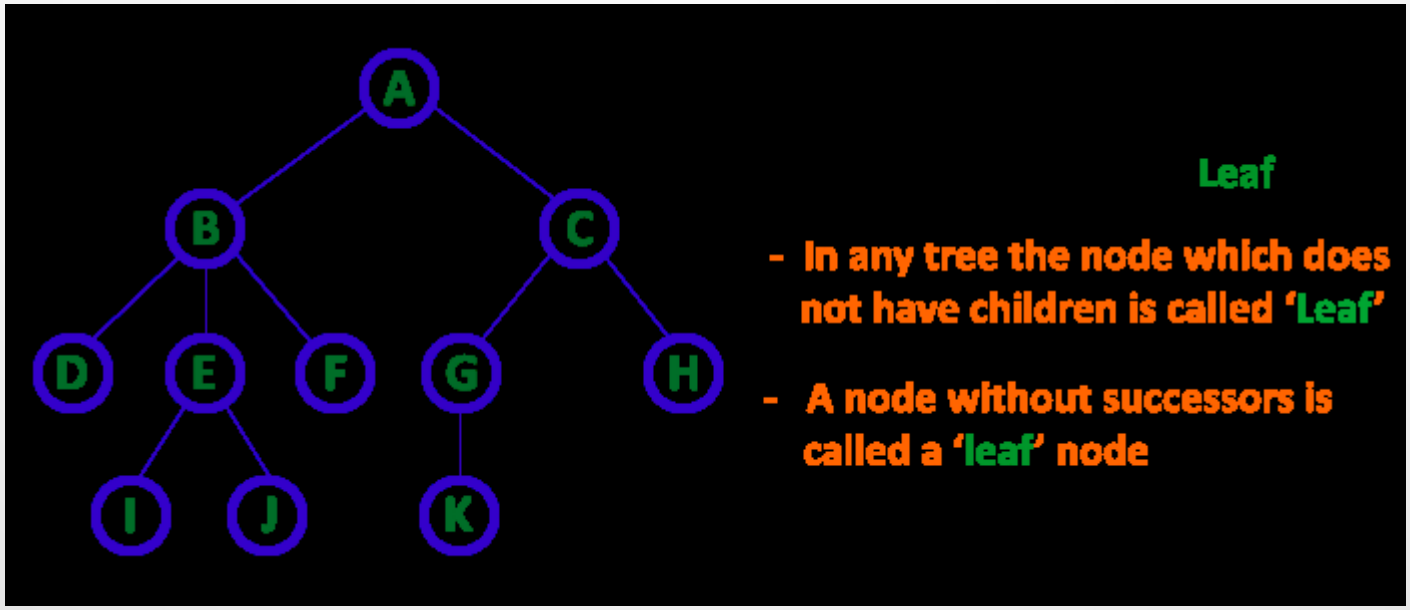
TREE TERMINOLOGY

- **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.



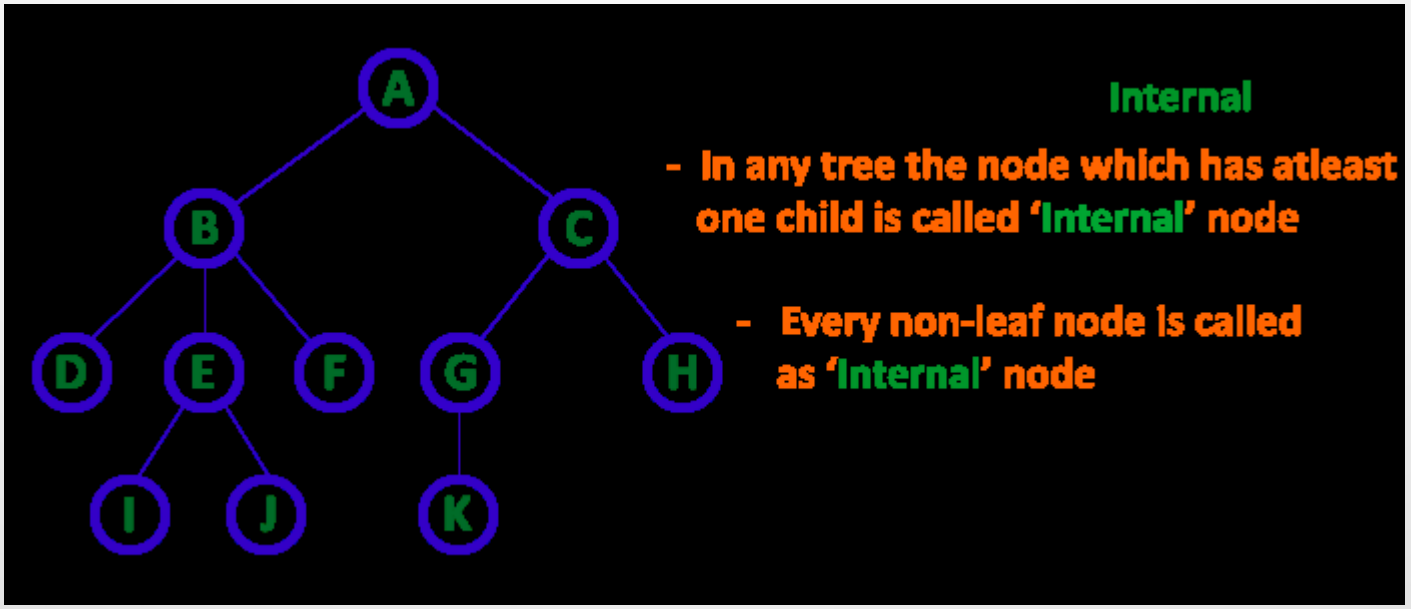
TREE TERMINOLOGY

- **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.



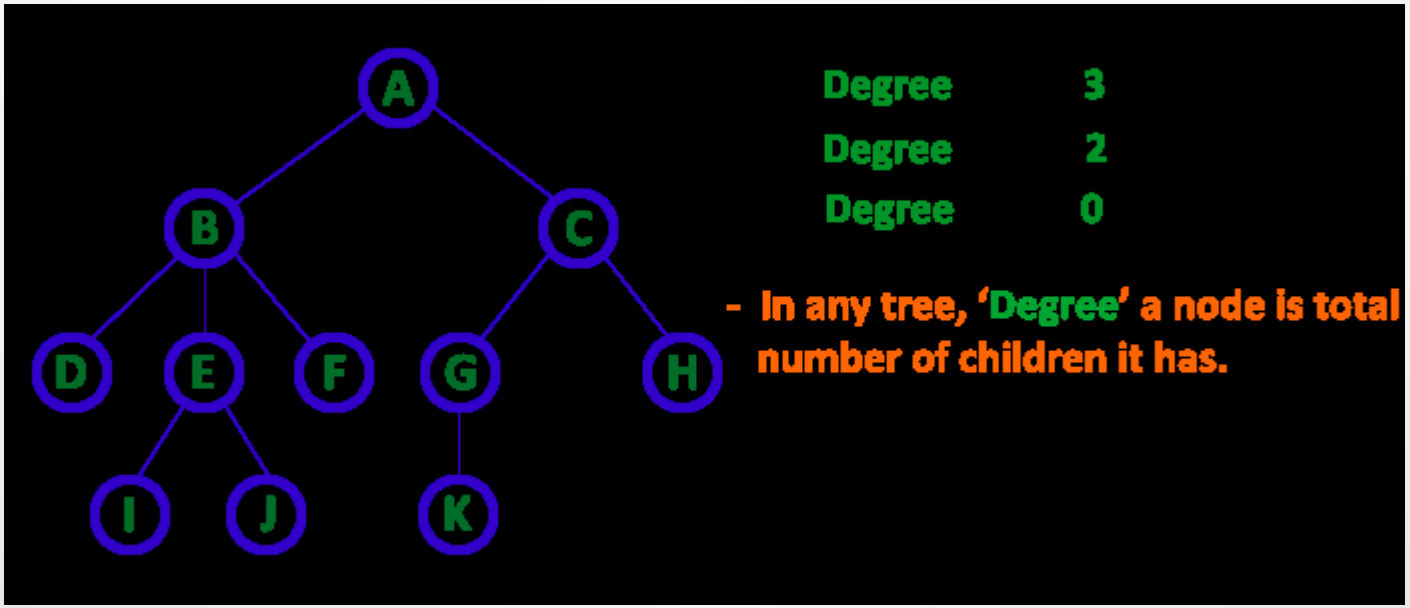
TREE TERMINOLOGY

- **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.



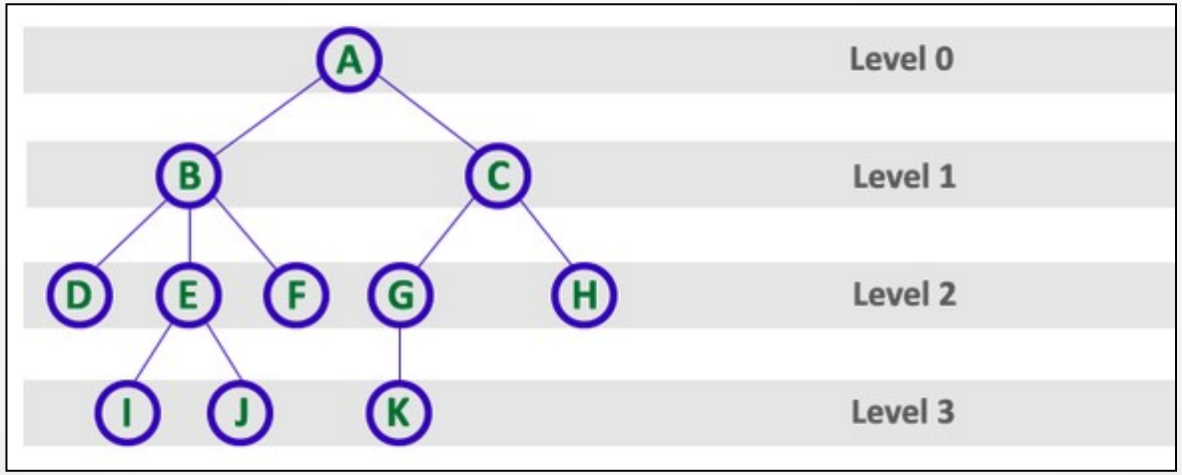
TREE TERMINOLOGY

- **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'



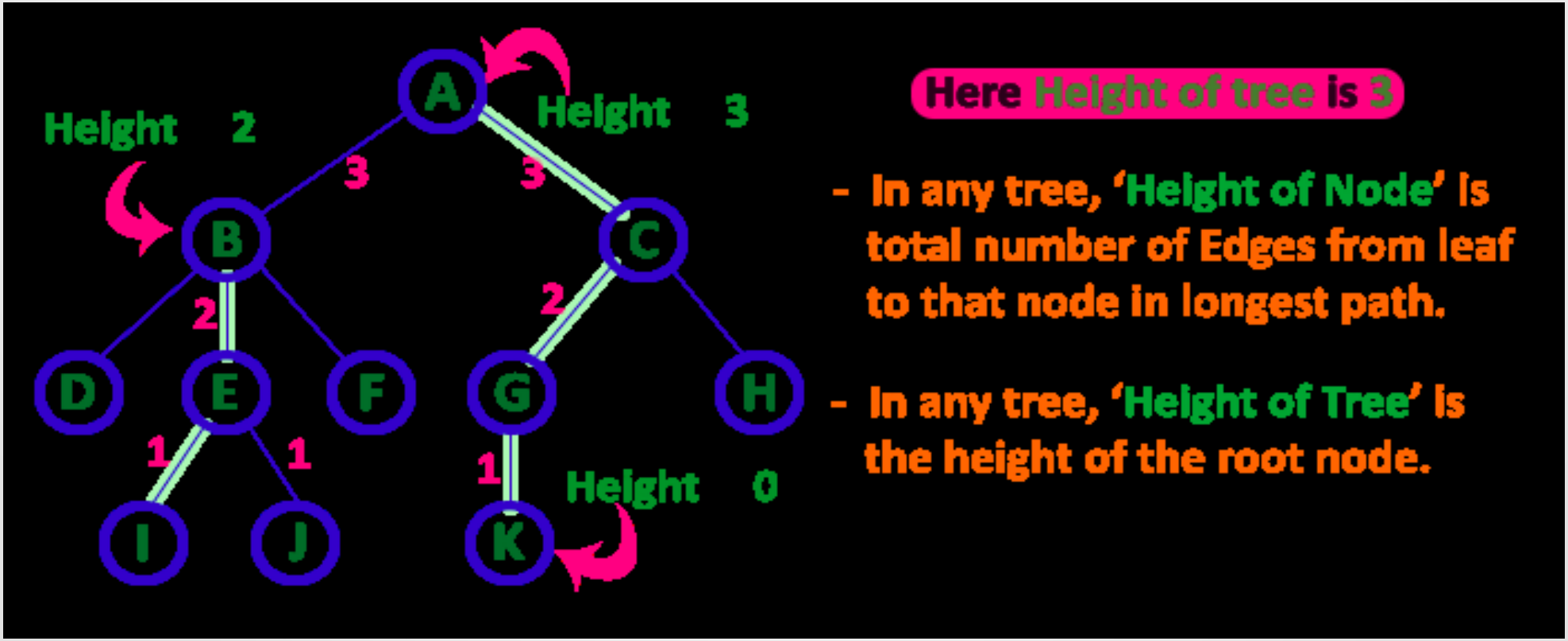
TREE TERMINOLOGY

- **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).



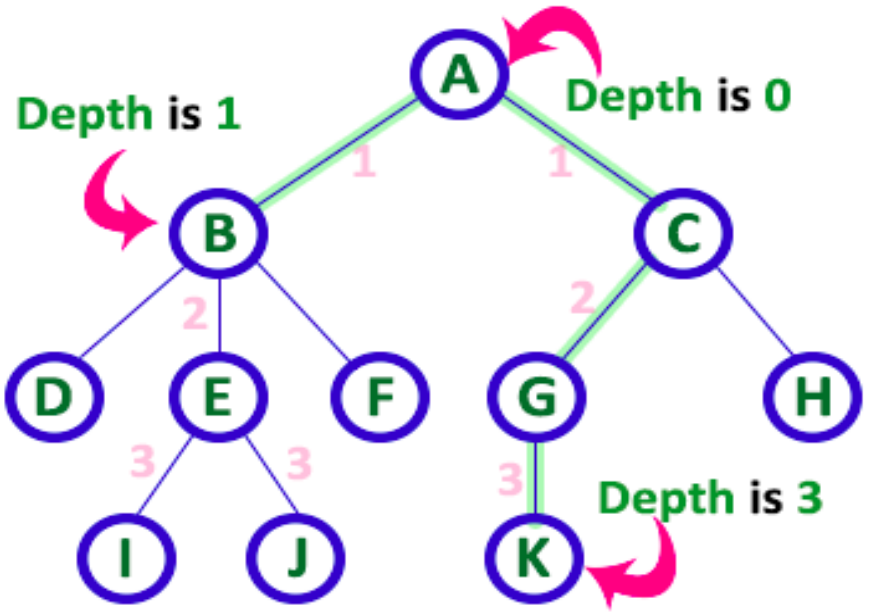
TREE TERMINOLOGY

- **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'.



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'.

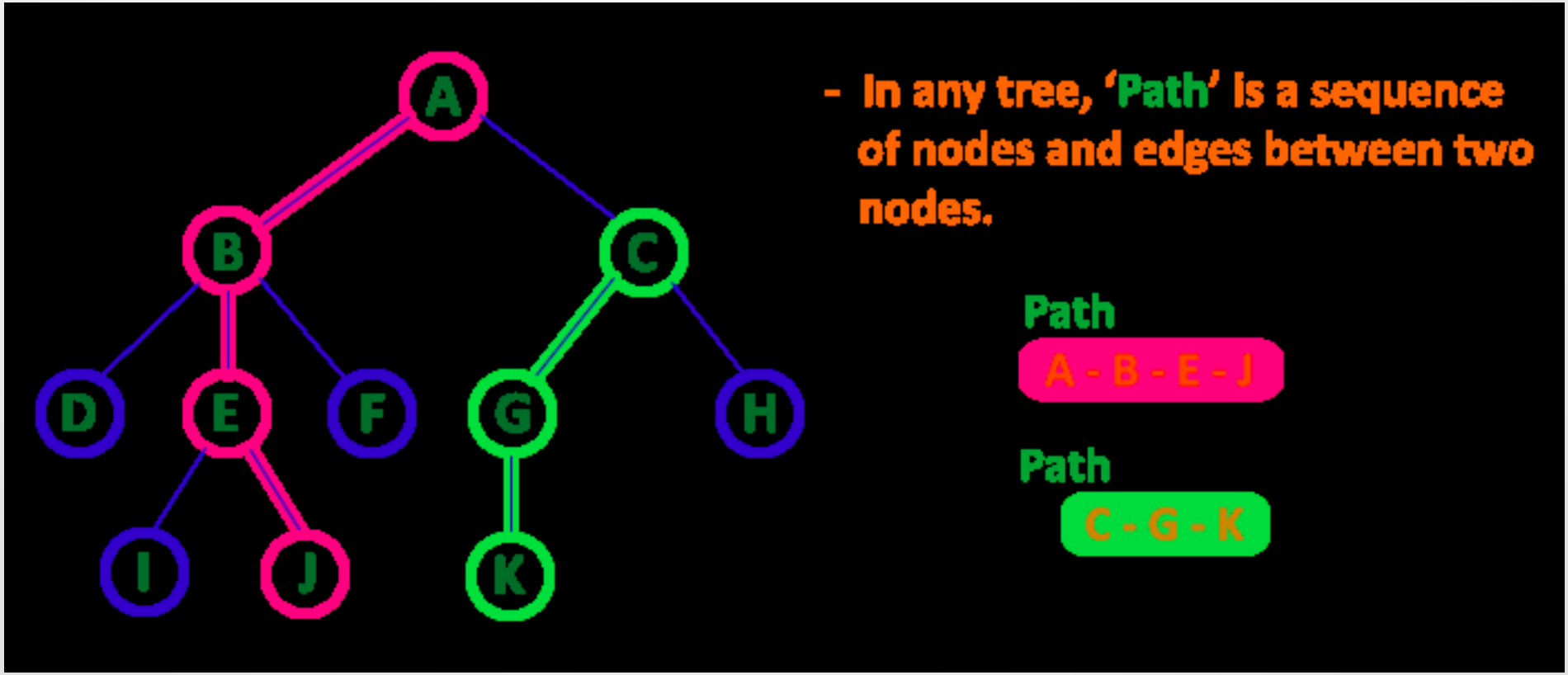


Here Depth of tree is 3

- 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.

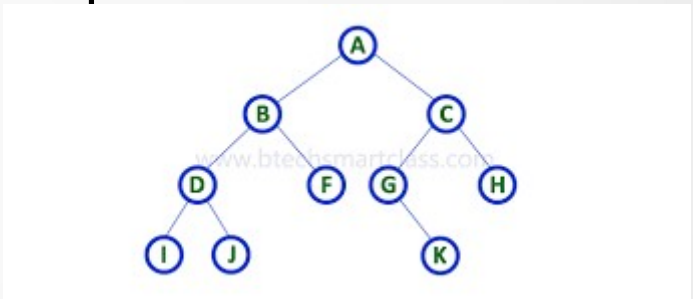
TREE TERMINOLOGY

- **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.

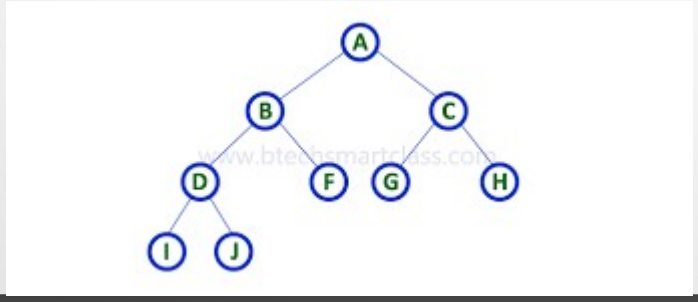


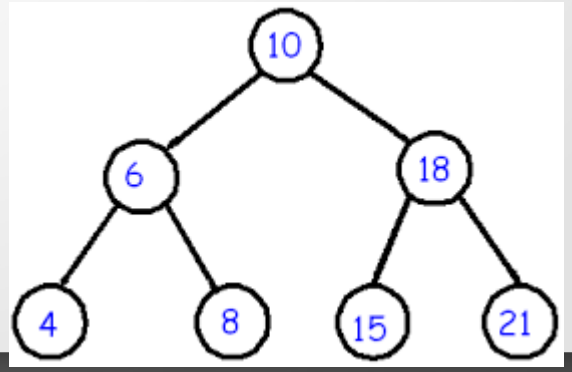
BINARY TREE

- 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.



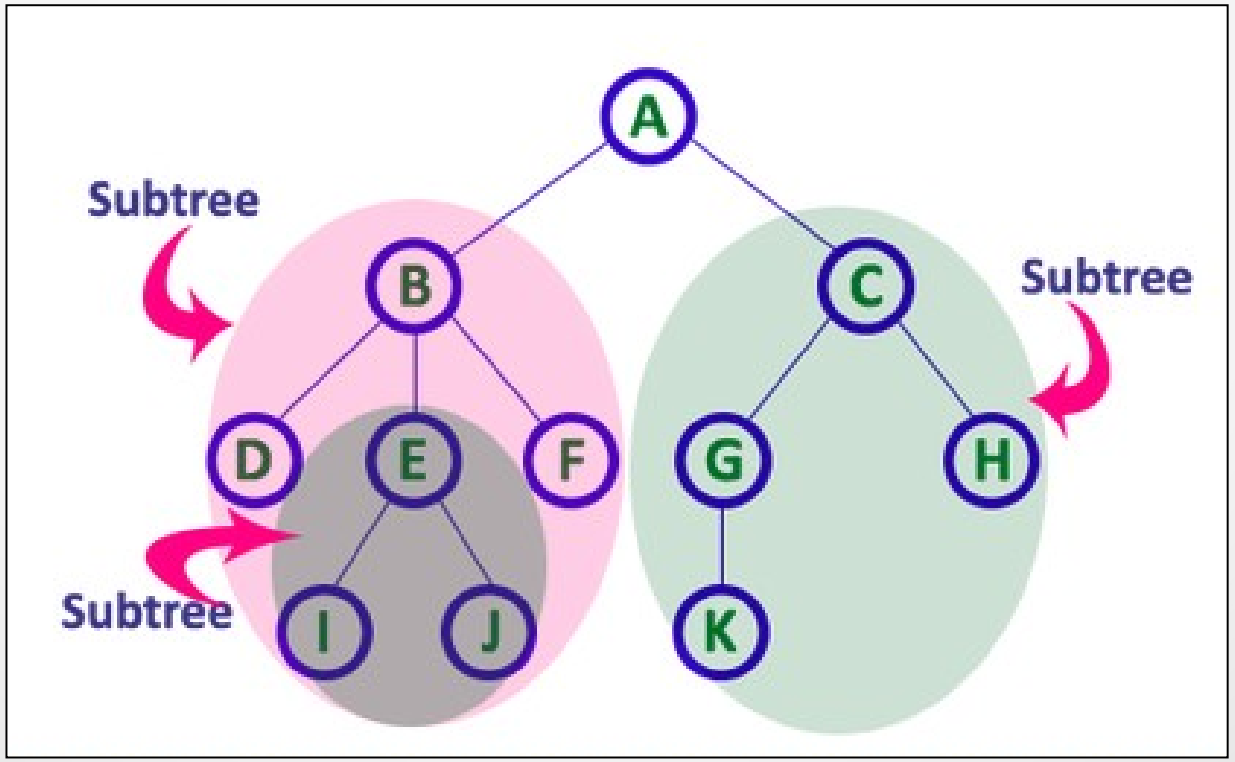
- Full Binary tree or proper binary tree:
- is a tree in which every node has zero or two children. Full binary tree has 2^level number of nodes at each level.





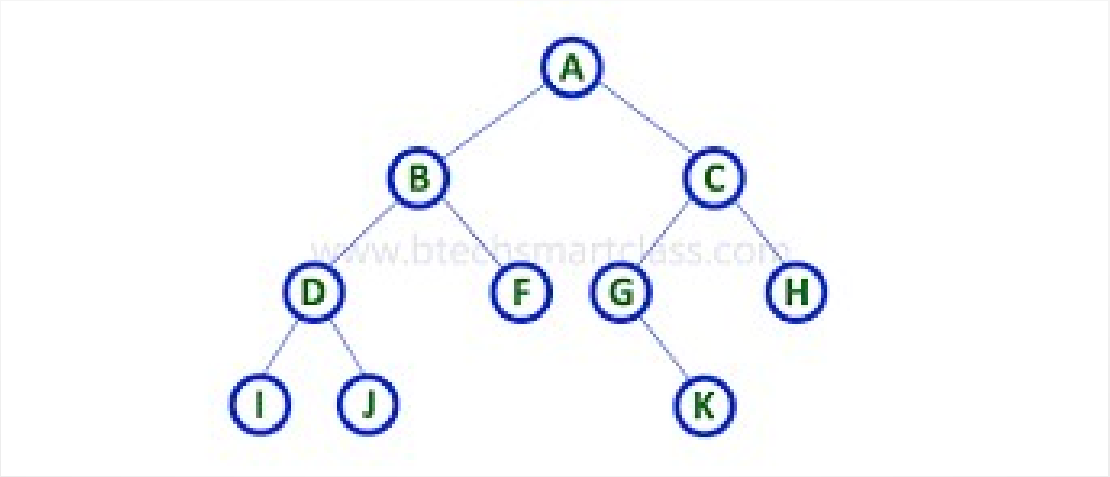
TREE TERMINOLOGY

- **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.



TREE Representation

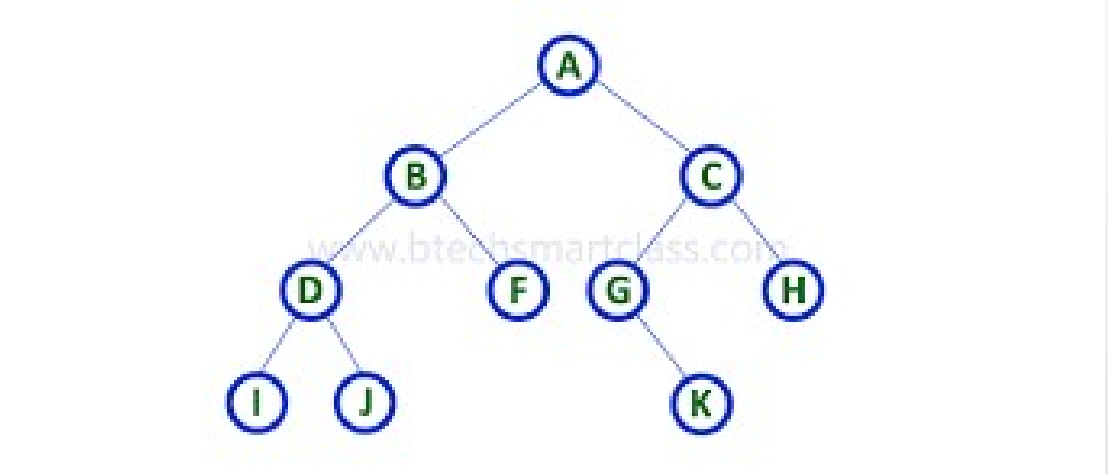
- A tree data structure can be represented in two methods. Those methods are as follows...
  - Array Representation
  - Linked List Representation



TREE Representation

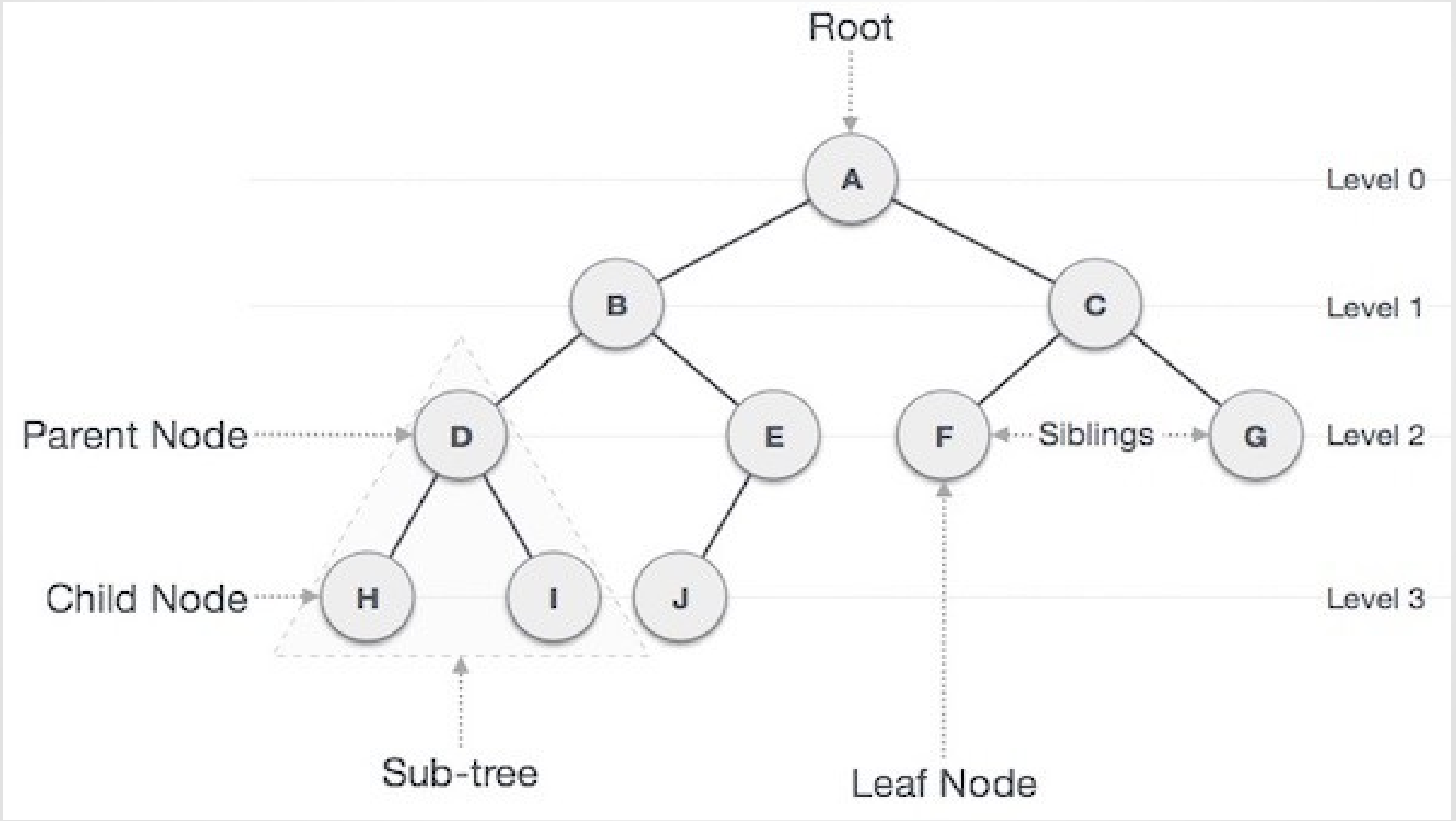
- **Array Representation:** In array representation of binary tree, we use a one dimensional array (1-D Array) to represent a binary tree.
- Consider the above example of binary tree and it is represented as follows

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

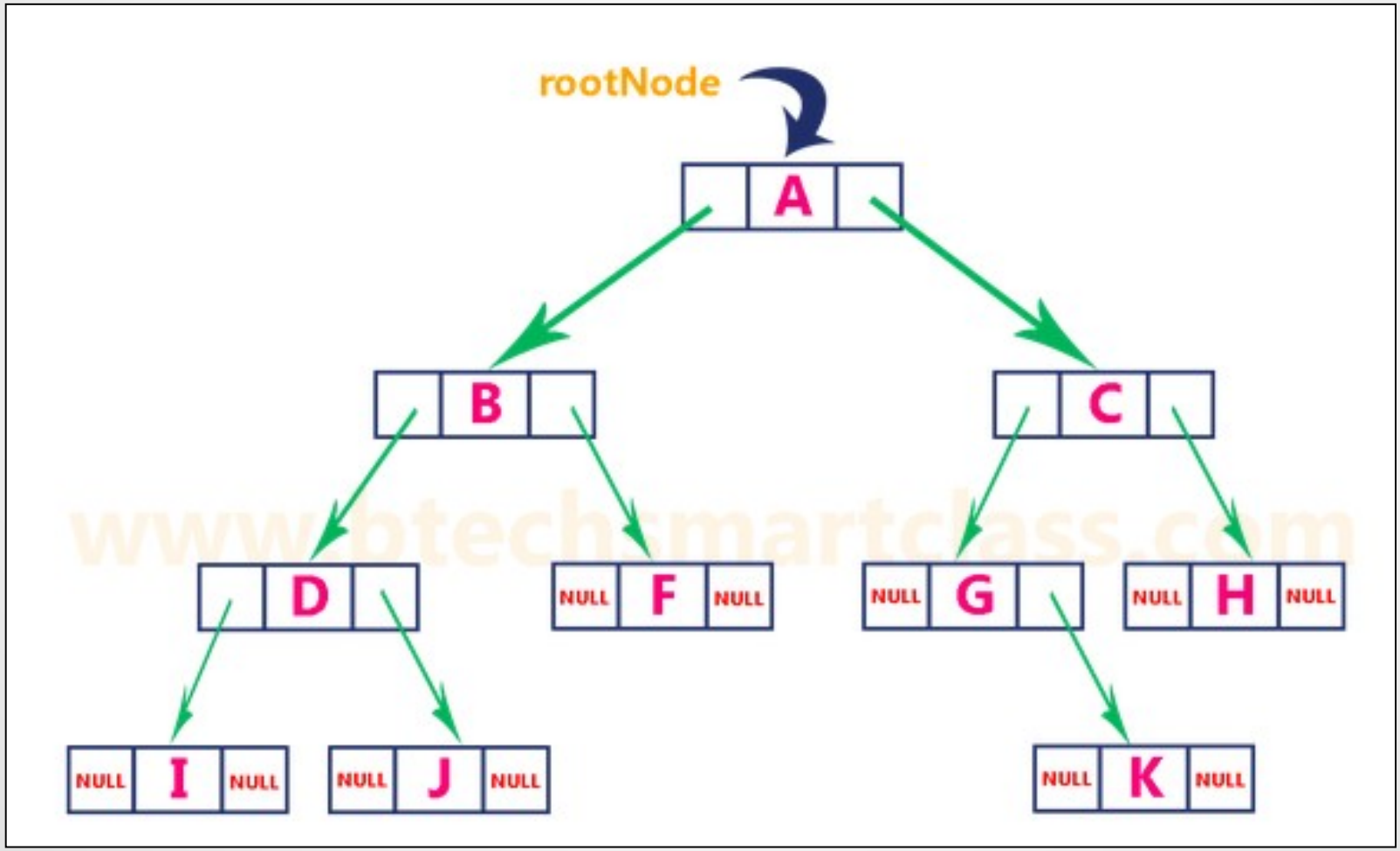


TREE Representation

- **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**


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

TREE Representation



Binary Tree Operations

Insertion/Creation

Traversal

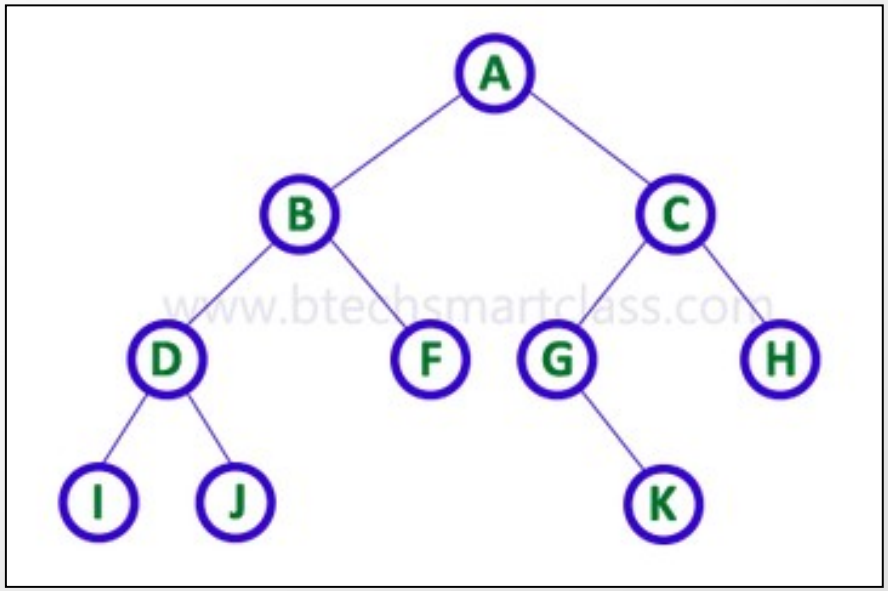
Deletion

Serach

Display

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.

- I - D - J - B - F - A - G - K - C - H - > In-Order Traversal.

- Algorithm Inorder(tree)
- 1. Traverse the left subtree, i.e., call Inorder(left-subtree)
- 2. Visit the root.
- 3. Traverse the right subtree, i.e., call Inorder(right-subtree

```
void printInorder(struct node* node)
{
    if (node == NULL)
        return;
    /* first recur on left child */
    printInorder(node->left);
     /* then print the data of node */
    printf("%d ", node->data);
    /* now recur on right child */
    printInorder(node->right);
}
```

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.

- A-B-D-I-J-F-C-G-K-H - > Pre-Order Traversal.

- Algorithm Preorder(tree)
- 1. Visit the root.
- 2. Traverse the left subtree, i.e., call Preorder(left-subtree)
- 3. Traverse the right subtree, i.e., call Preorder(right-subtree)

```
void printPreorder(struct node* node)
{
     if (node == NULL)
        return;
     // deal with the node
     printf("%d ", node->data);
     // then recur on left subtree
     printPreorder(node->left);
      // then recur on right subtree
     printPreorder(node->right);
          }
```

Binary Tree Traversals

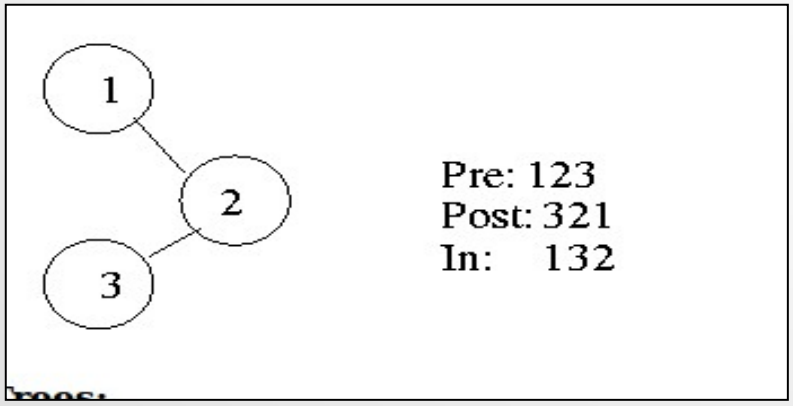
3. Post - Order Traversal (leftChild - rightChild - root)

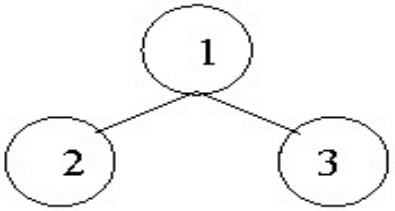
- In Post-Order traversal, the root node is visited after left child and right child. In this traversal, left child node is visited first, then its right child and then its root node. This is recursively performed until the right most node is visited.
- Here we have visited in the order of I - J - D - F - B - K - G - H - C - A using Post-Order Traversal.

- Algorithm Postorder(tree)
- 1. Traverse the left subtree, i.e., call Postorder(left-subtree)
- 2. Traverse the right subtree, i.e., call Postorder(right-subtree)
- 3. Visit the root.

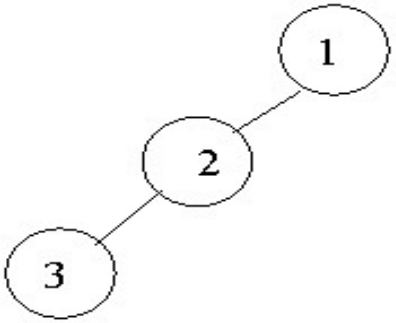
```
void printPostorder(struct node* node)
{
    if (node == NULL)
       return;
    // first recur on left subtree
    printPostorder(node->left);
     // then recur on right subtree
    printPostorder(node->right);
    // now deal with the node
    printf("%d ", node->data);
}
```

Binary Tree Traversals

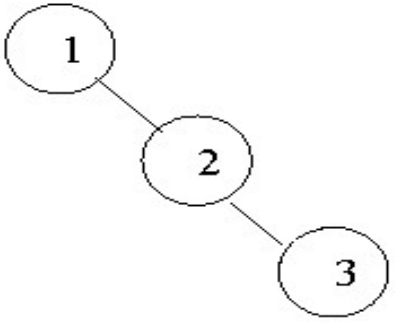




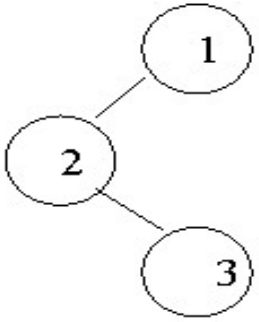
Pre: 123
Post: 231
In: 213



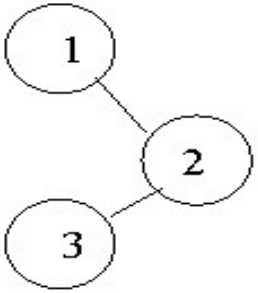
Pre: 123
Post: 321
In: 321



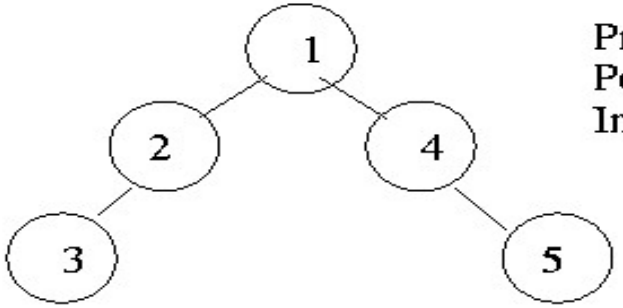
Pre: 123
Post: 321
In: 123



Pre: 123
Post: 321
In: 231

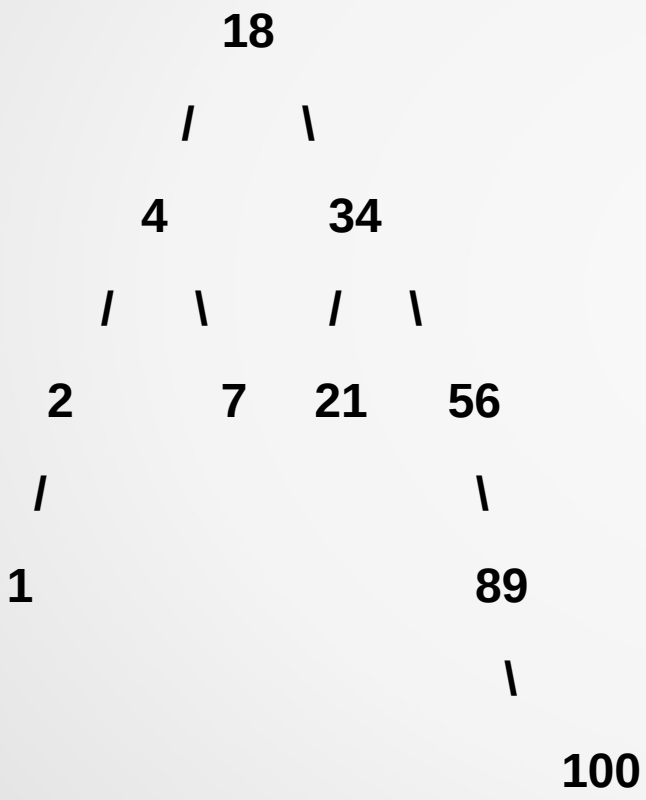


Pre: 123
Post: 321
In: 132



Pre: 12345
Post: 32541
In: 32145

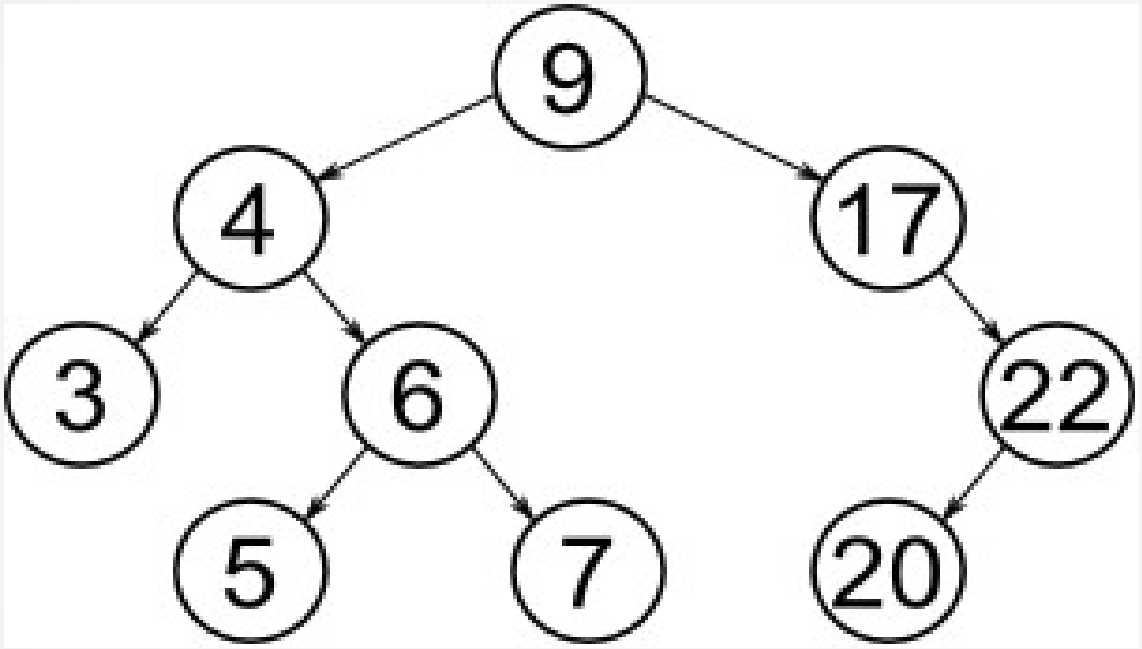
Inorder:1-2-4-7-18-21-34-56-89-100



BINARY SEARCH TREE

- A binary search tree (BST), also known as an ordered binary tree, is a node-based data structure in which each node has no more than two child nodes. Each child must either be a leaf node or the root of another binary search tree.
- The left sub-tree contains only nodes with keys less than the parent node; the right sub-tree contains only nodes with keys greater than the parent node.
- A binary search tree (BST) is a tree with following properties –
  - The left sub-tree of a node has key less than or equal to its parent node's key.
  - The right sub-tree of a node has key greater than or equal to its parent node's key.

BINARY SEARCH TREE

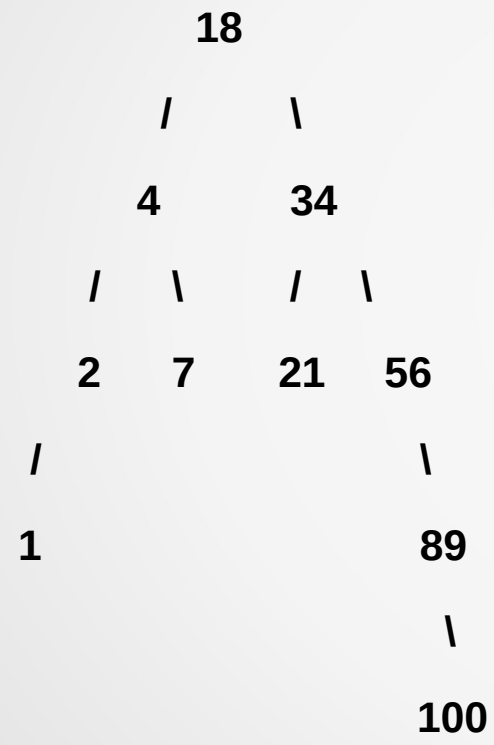


Binary Search Tree, is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys less 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.
- There must be no duplicate nodes.

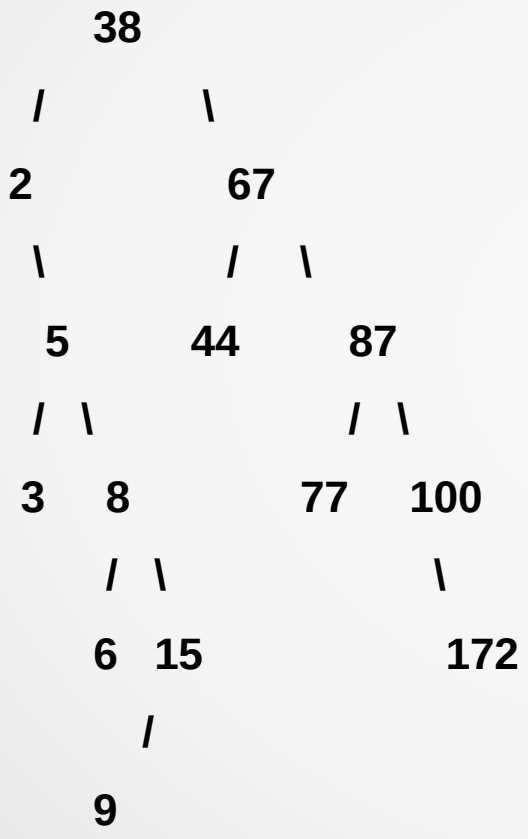
Creation

18 ,34, 56, 89 ,100, 4, 7, 21, 2, 1



Creation

38 67 87 2 5 8 15 100 44 172 77 3 9 6



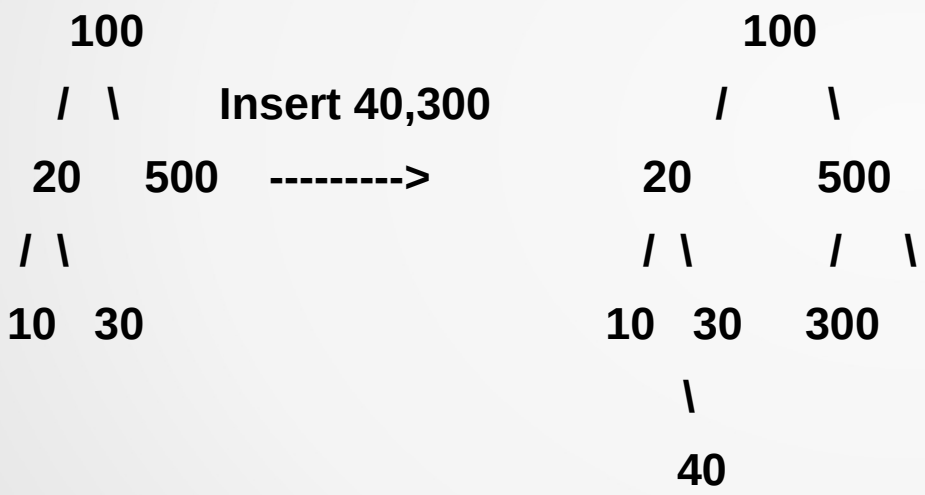
Serach

```
// C function to search a given key in a given BST
struct node* search(struct node* root, int key)
{
    // Base Cases: root is null or key is present at root
    if (root == NULL || root->key == key)
       return root;

    // Key is greater than root's key
    if (root->key < key)
       return search(root->right, key);

    // Key is smaller than root's key
    return search(root->left, key);
}
```

Insertion

Insertion of a key

A new key is always inserted at leaf. We start searching a key from root till we hit a leaf node. Once a leaf node is found, the new node is added as a child of the leaf node.



Insertion

```
* A utility function to insert a new node with given key in BST */
struct node* insert(struct node* node, int key)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(key);

    /* Otherwise, recur down the tree */
    if (key < node->key)
        node->left  = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);

    /* return the (unchanged) node pointer */
    return node;
}
```

Deletion

Binary search tree. Removing a node

Remove operation on binary search tree is more complicated, than add and search. Basically, in can be divided into two stages:

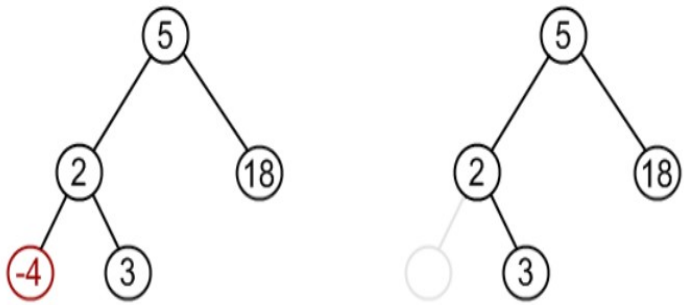
search for a node to remove;

if the node is found, run remove algorithm.

- 1. Node to be removed has no children.
- This case is quite simpl Algorithm sets corresponding link of th parent to NULL and disposes the node.
- Example. Remove -4 from a BST.

This case is quite simple. Algorithm sets corresponding link of the parent to NULL and disposes the node.

Example. Remove -4 from a BST.

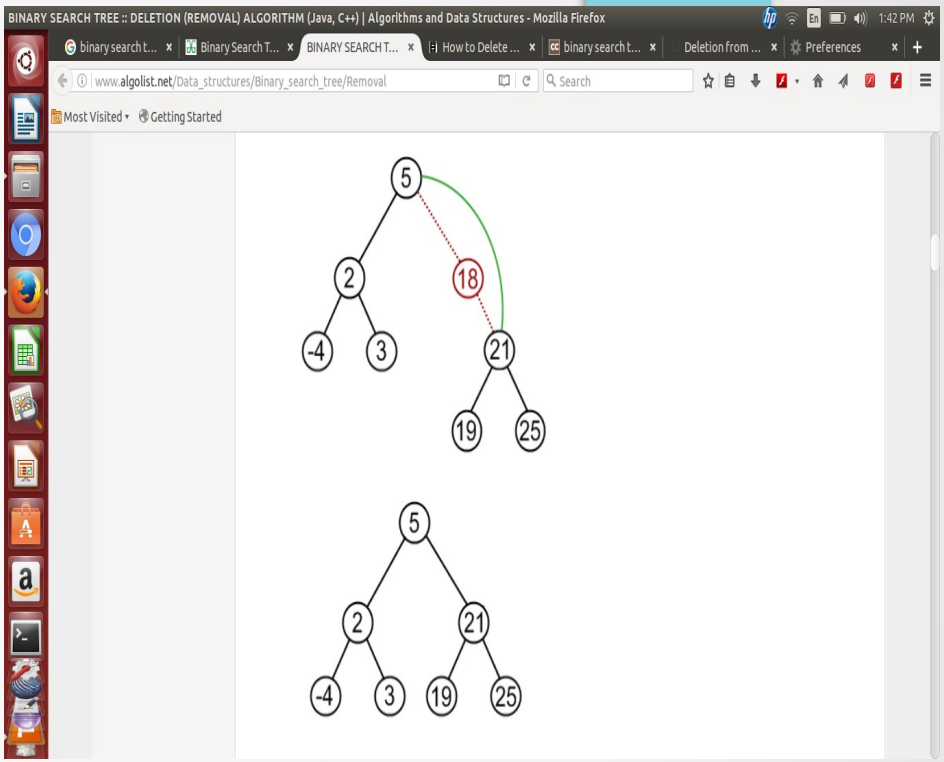


2. Node to be removed has one child.

It this case, node is cut from the tree and algorithm links single child (with it's subtree) directly to the parent of the removed node.

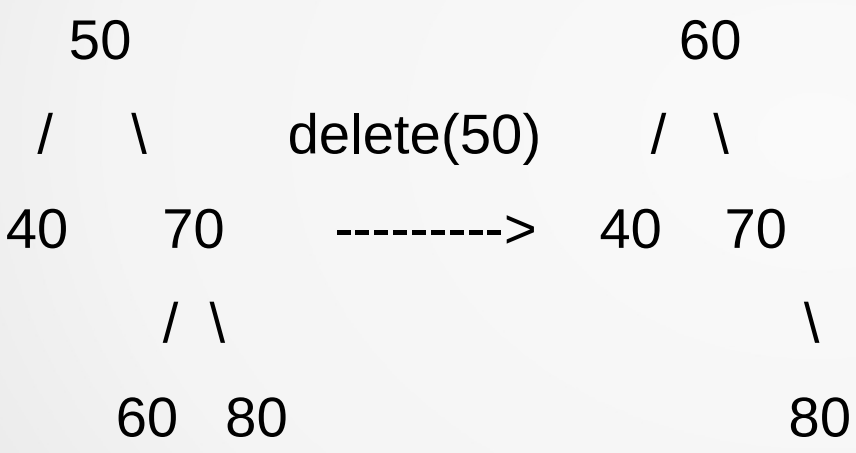
Example. Remove 18 from a BST.

- 2. Node to be removed has one child.
- 'In this case, node is cut from the tree and algorithm links single child (with it's subtree) directly to the parent of the removed node.
- Example. Remove 18 from a BST.



3. Node to be removed has two children.

This is the most complex case.



In-order left – root- right

40-50-60-70-80

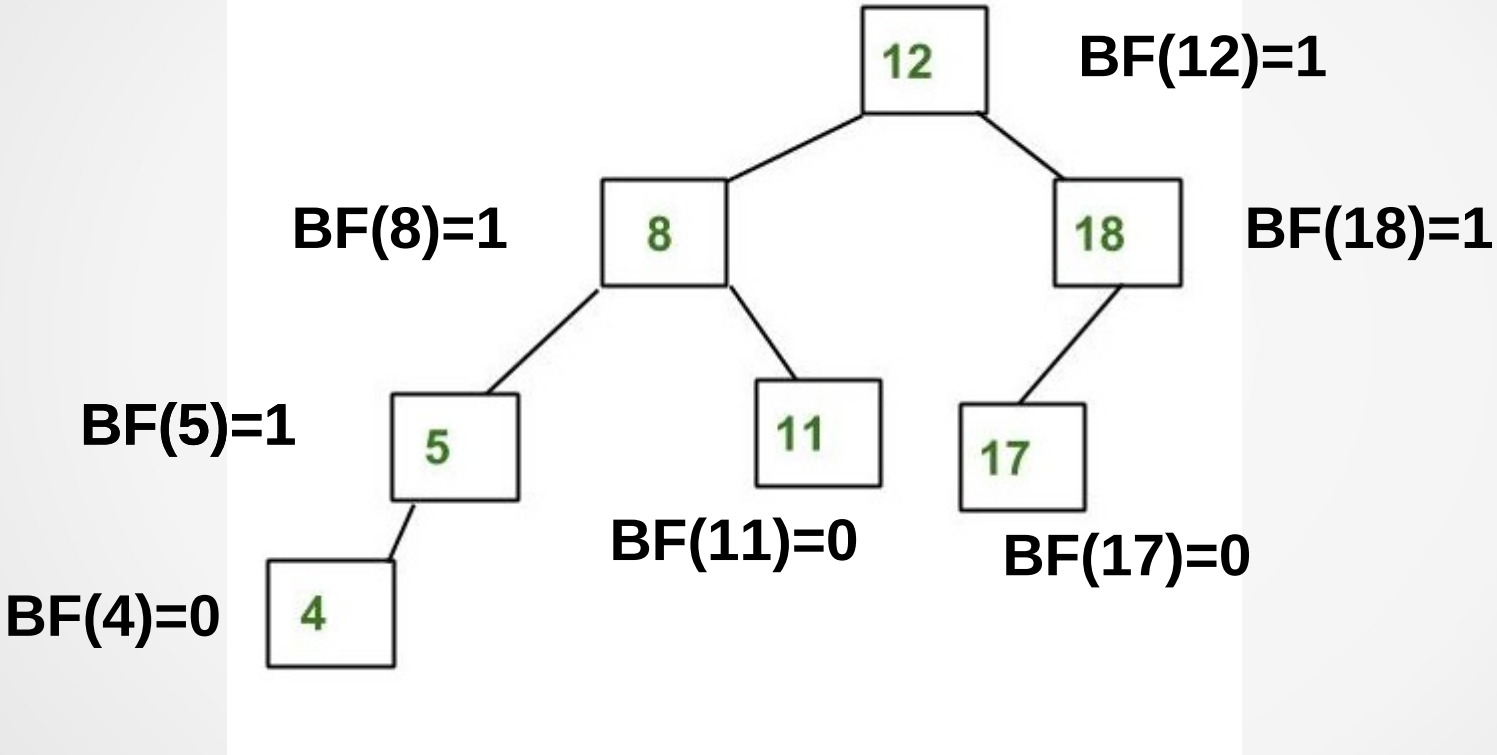
AVL/Height Balanced Tree

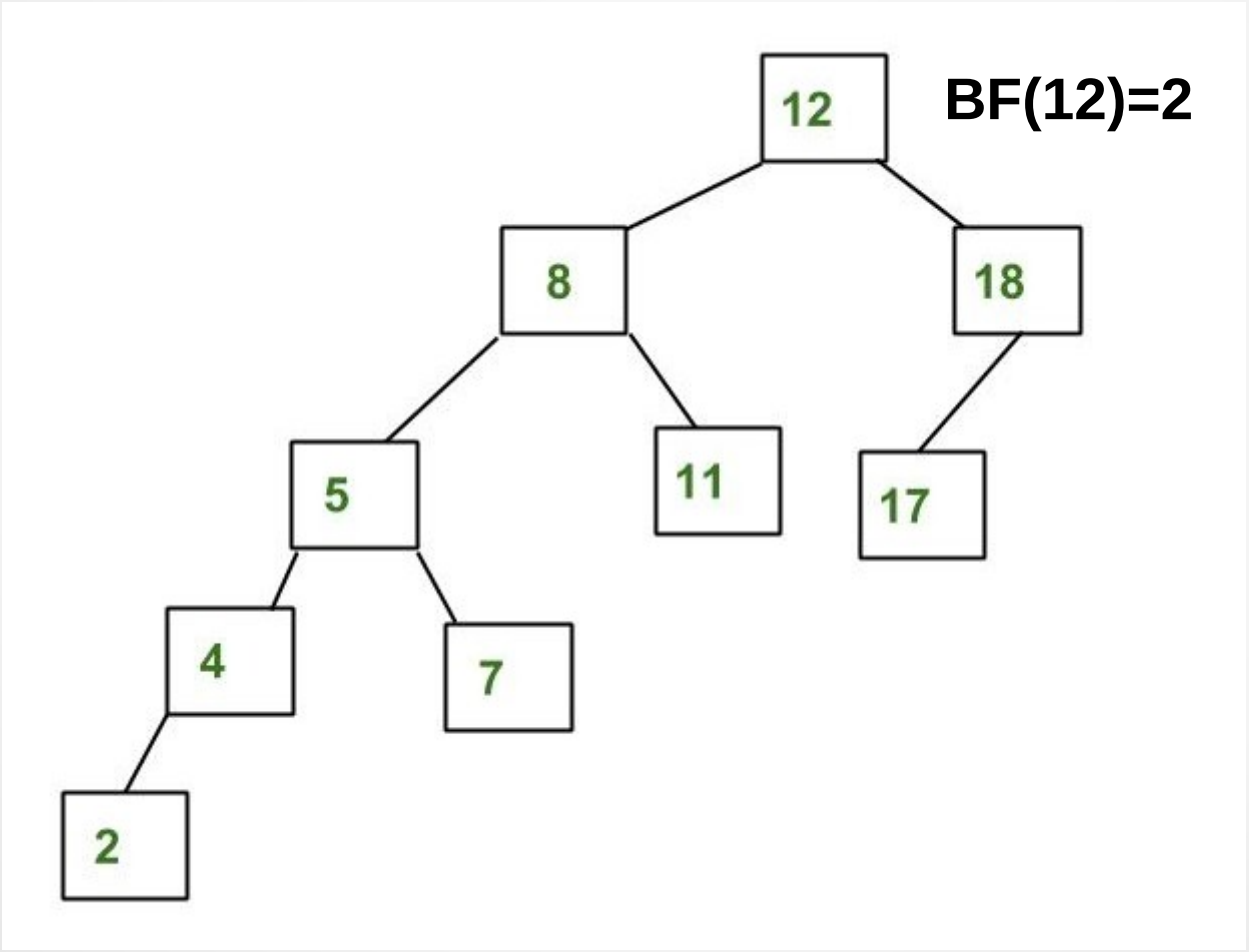
- In computer science, an AVL tree is a self-balancing binary search tree. It was the first such data structure to be invented.
- Named after their inventors, Adelson-Velskii and Landis, they were the first dynamically balanced trees to be proposed.
- AVL tree is a self-balancing Binary Search Tree (BST) where the difference between heights of left and right subtrees cannot be more than one for all nodes.
- An AVL tree is a binary search tree which has the following properties:
  - The sub-trees of every node differ in height by at most one.
  - Every sub-tree is an AVL tree.

AVL/Height Balanced Tree

- The structure of AVL tree is same as that of bianry tree but it stores an additional variable called Balance Factor.
- Thus every node has a balance factor associated with it. The balance factor of a node is calculated by subtracting the height of its right sub tree from the height of its left sub-tree.
- A binary search tree in which every node has a balance factor of -1, 0, or 1 is said to be height balanced.
- A node with any other balance factor is considered to be unbalanced and requires rebalancing of the tree.
- **Balance factor = Height (left sub-tree) - Height (right sub-tree)**
- **BF(12)=1**

Balance factor = Height (left sub-tree) - Height (right sub-tree)

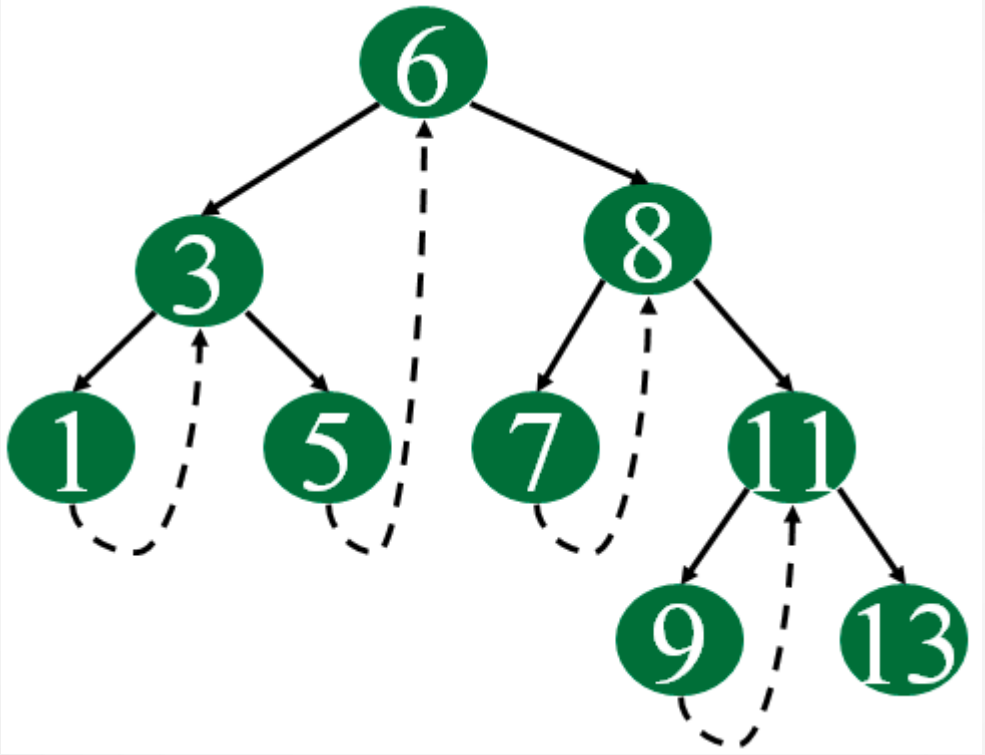



12
BF(12)=2
8
18
5
11
17
4
7
2

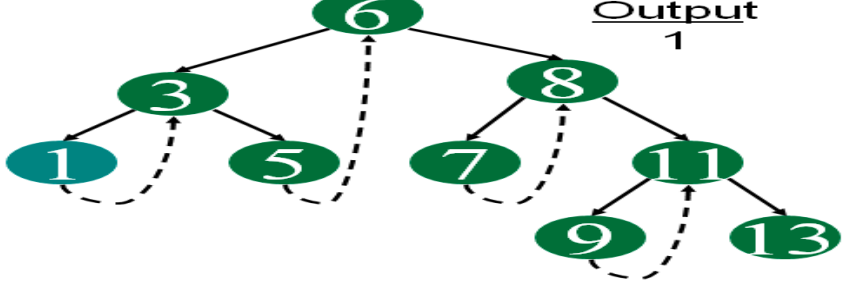
Threaded Binary Tree

Inorder traversal of a Binary tree is either be done using recursion or with the use of a auxiliary stack. The idea of threaded binary trees is to make inorder traversal faster and do it without stack and without recursion. A binary tree is made threaded by making all right child pointers that would normally be NULL point to the inorder successor of the node (if it exists).

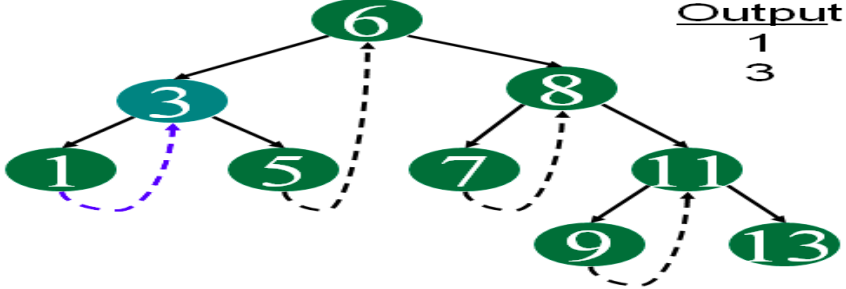
Following diagram shows an example Single Threaded Binary Tree. The dotted lines represent threads.



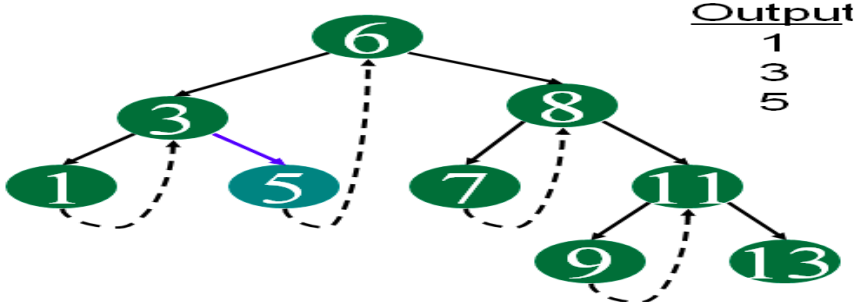
IN ORDER: 1-3-5-6-7-8-9-11-13



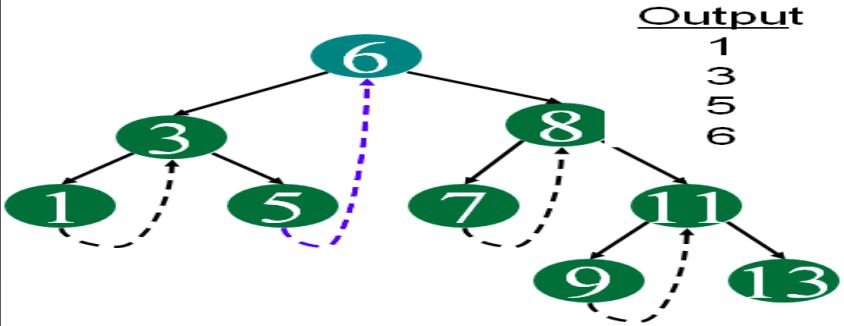
Start at leftmost node, print it



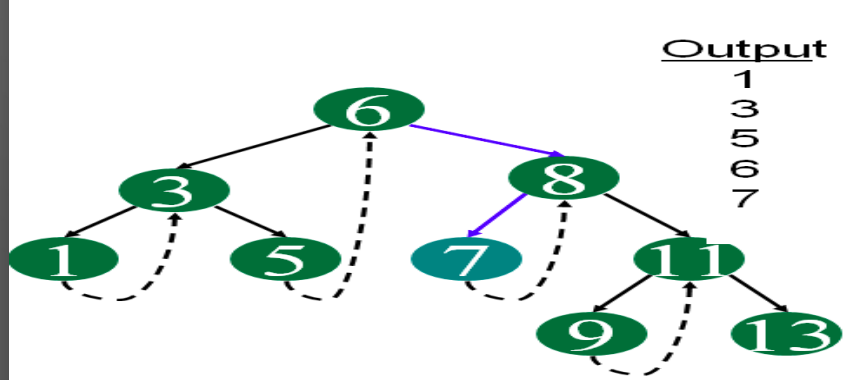
Follow thread to right, print node



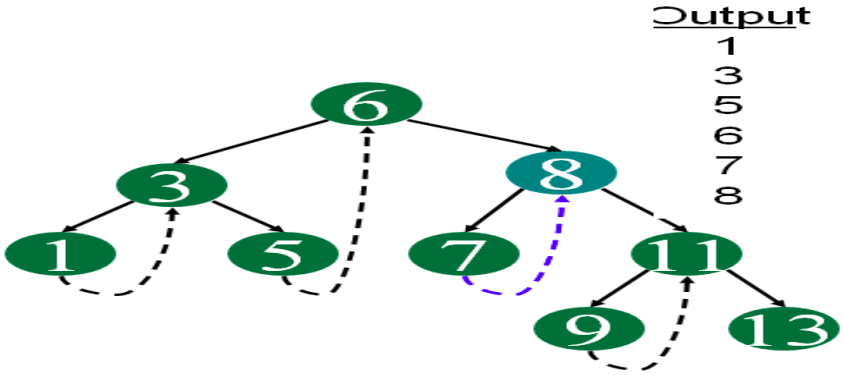
Follow link to right, go to leftmost node and print



Follow thread to right, print node



Follow link to right, go to leftmost node and print



Follow thread to right, print node

continue same way for remaining node.....

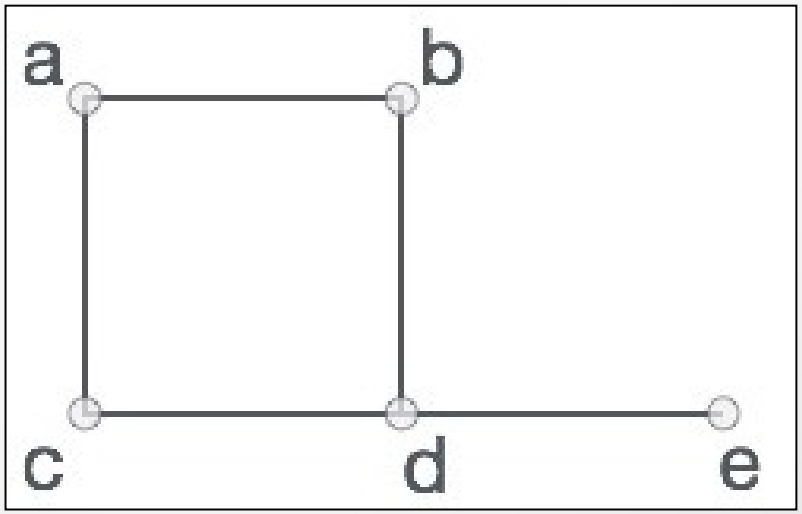
There are two types of threaded binary trees.

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. The predecessor threads are useful for reverse inorder traversal and postorder traversal.

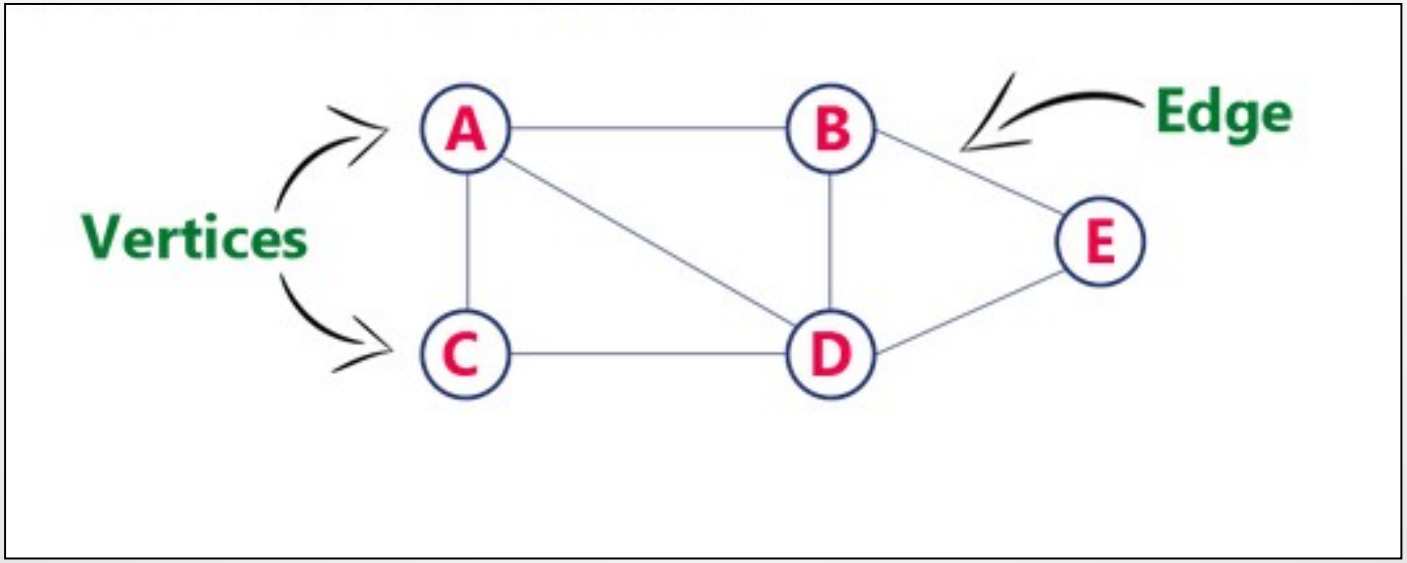
GRAPH

- A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by points termed as vertices, and the links that connect the vertices are called edges.
- Formally, a graph is a pair of sets (V, E), where V is the set of vertices and E is the set of edges, connecting the pairs of vertices.
- In the below graph,
- V = {a, b, c, d, e}
- E = {ab, ac, bd, cd, de}



GRAPH: Example

- The following is a graph with 5 vertices and 6 edges.
- This graph G can be defined as G = (V , E)
- Where V = {A,B,C,D,E} and E = (A,B),(A,C)(A,D),(B,D),(C,D),(B,E),(E,D)}.



GRAPH: Terms

- **Undirected Graph:** A graph with only undirected edges is said to be undirected graph.
- **Directed Graph:** A graph with only directed edges is said to be directed graph.
- **Mixed Graph:** A graph with undirected and directed edges is said to be mixed graph.
- **End vertices or Endpoints:** The two vertices joined by an edge are called the end vertices (or endpoints) of the edge.
- **Origin:** If an edge is directed, its first endpoint is said to be origin of it.
- **Destination:** If an edge is directed, its first endpoint is said to be origin of it and the other endpoint is said to be the destination of the edge.

GRAPH: Terms

- **Adjacent:** If there is an edge between vertices A and B then both A and B are said to be adjacent. In other words, Two vertices A and B are said to be adjacent if there is an edge whose end vertices are A and B.
- **Outgoing Edge:** A directed edge is said to be outgoing edge on its orign vertex.
- **Incoming Edge:** A directed edge is said to be incoming edge on its destination vertex.
- **Degree:** Total number of edges connected to a vertex is said to be degree of that vertex.
- **Indegree:** Total number of incoming edges connected to a vertex is said to be indegree of that vertex.
- **Outdegree:** Total number of outgoing edges connected to a vertex is said to be outdegree of that vertex.

GRAPH: Terms

- **Parallel edges or Multiple edges:** If there are two undirected edges to have the same end vertices, and for two directed edges to have the same origin and the same destination. Such edges are called parallel edges or multiple edges.
- **Self-loop:** An edge (undirected or directed) is a self-loop if its two endpoints coincide. (is an edge with the end vertices)
- **Simple Graph:** A graph is said to be simple if there are no parallel and self-loop edges.
- **Path:** A path is a sequence of alternating vertices and edges that starts at a vertex and ends at a vertex such that each edge is incident to its predecessor and successor vertex.

GRAPH: Representation

- Graph data structure is represented using following representations.
  - Adjacency Matrix
  - Adjacency List

GRAPH: Terms

- **Vertex:** A individual data element of a graph is called as Vertex. Vertex is also known as node. In above example graph, A, B, C, D & E are known as vertices.
- **Edge:** An edge is a connecting link between two vertices. Edge is also known as Arc. An edge is represented as (startingVertex, endingVertex). For example, in above graph, the link between vertices A and B is represented as (A,B). In above example graph, there are 7 edges (i.e., (A,B), (A,C), (A,D), (B,D), (B,E), (C,D), (D,E))
- **Edges are three types: Undirected Edge -** An undirected egde is a bidirectional edge. If there is a undirected edge between vertices A and B then edge (A , B) is equal to edge (B , A).
- **Directed Edge -** A directed egde is a unidirectional edge. If there is a directed edge between vertices A and B then edge (A , B) is not equal to edge (B , A).
- **Weighted Edge -** A weighted egde is an edge with cost on it.