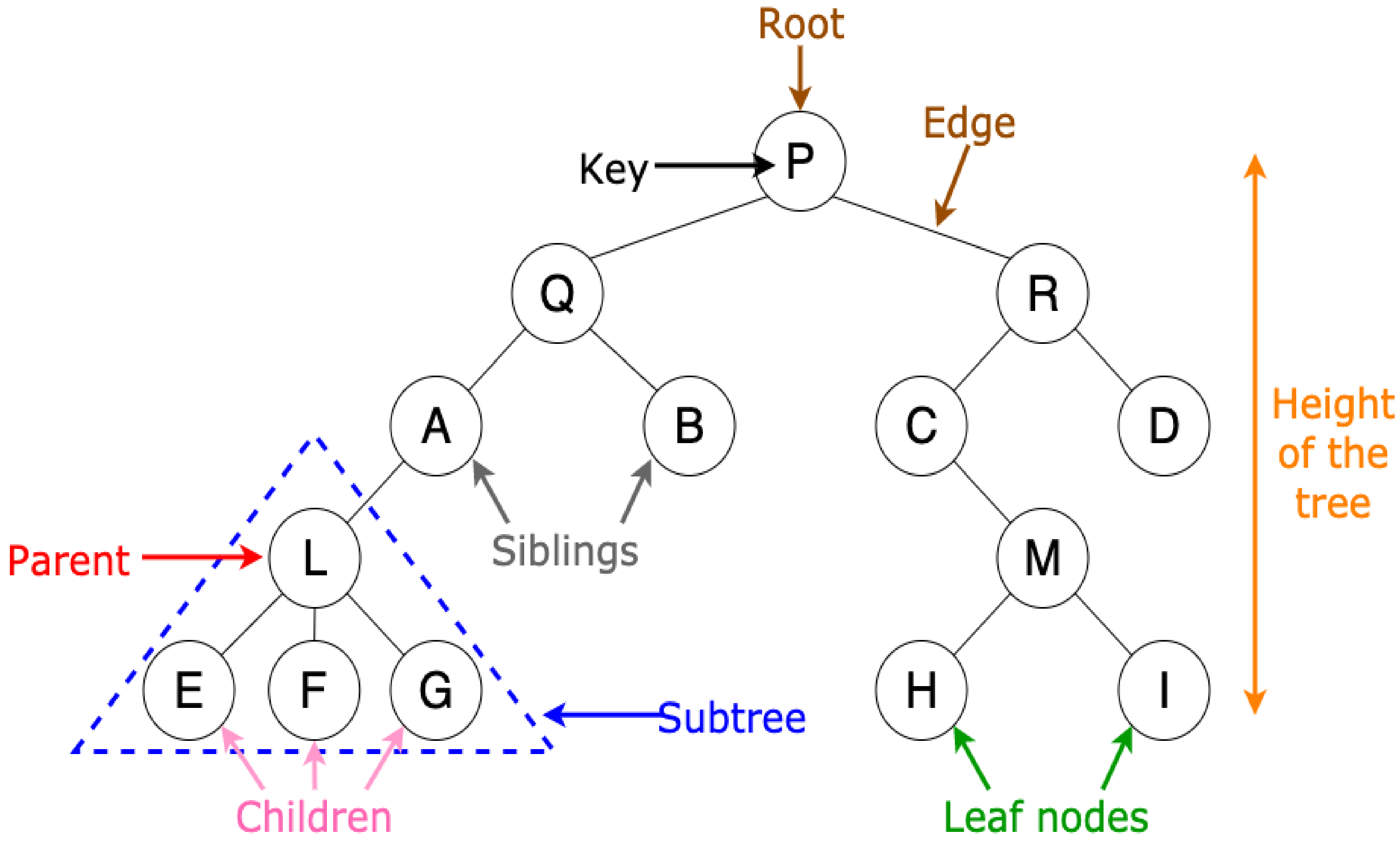
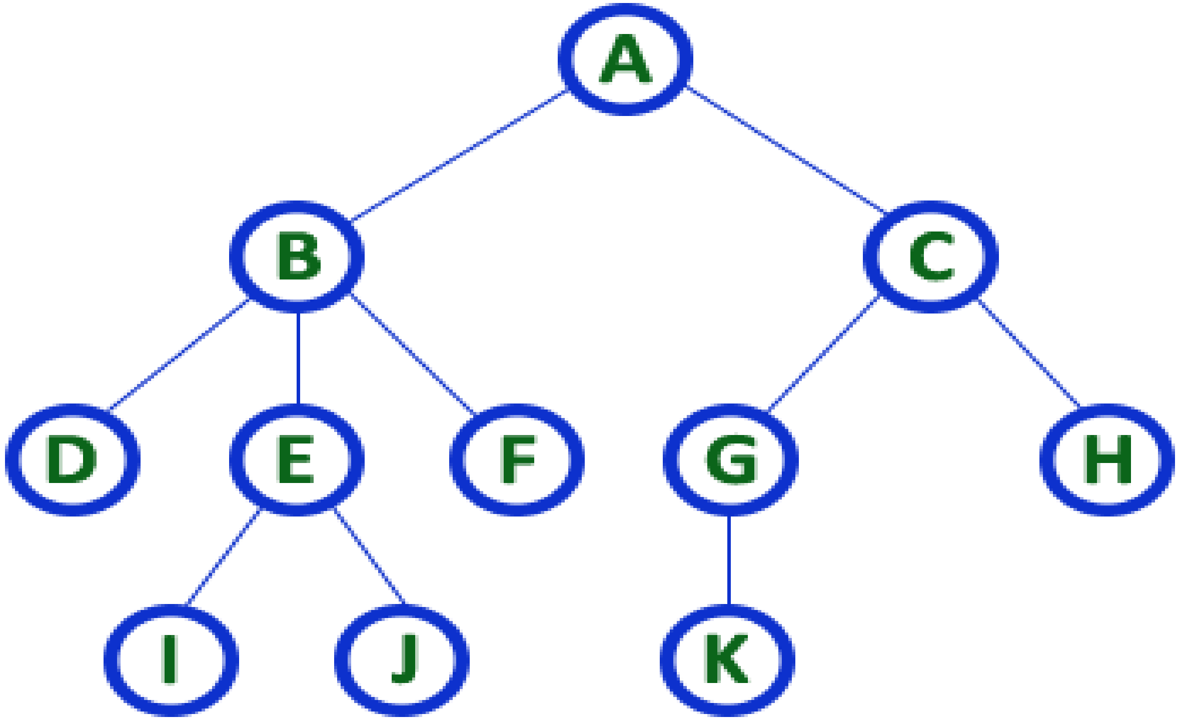
TREES



Root
Edge
Key
P
Q
R
A
B
C
D
L
M
E
F
G
H
I
Parent
Siblings
Subtree
Children
Leaf nodes
Height of the tree

Tree - Terminology

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



TREE with 11 nodes and 10 edges

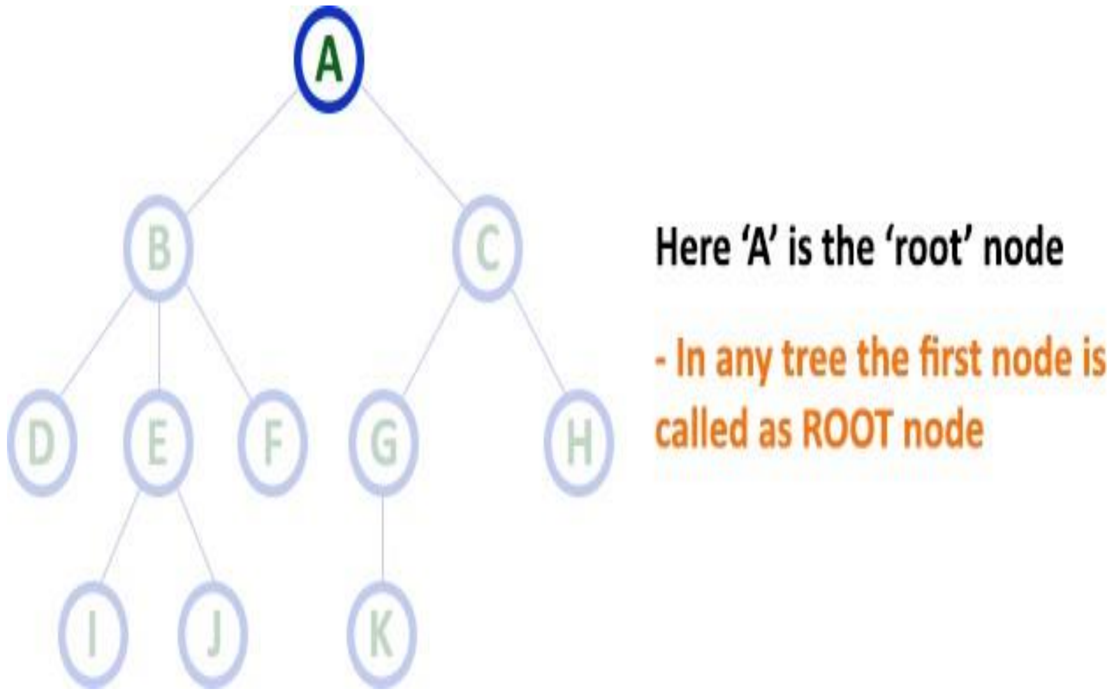
- **In any tree with 'N' nodes there will be maximum of 'N-1' edges**
- **In a tree every individual element is called as 'NODE'**

Terminology

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

1. Root

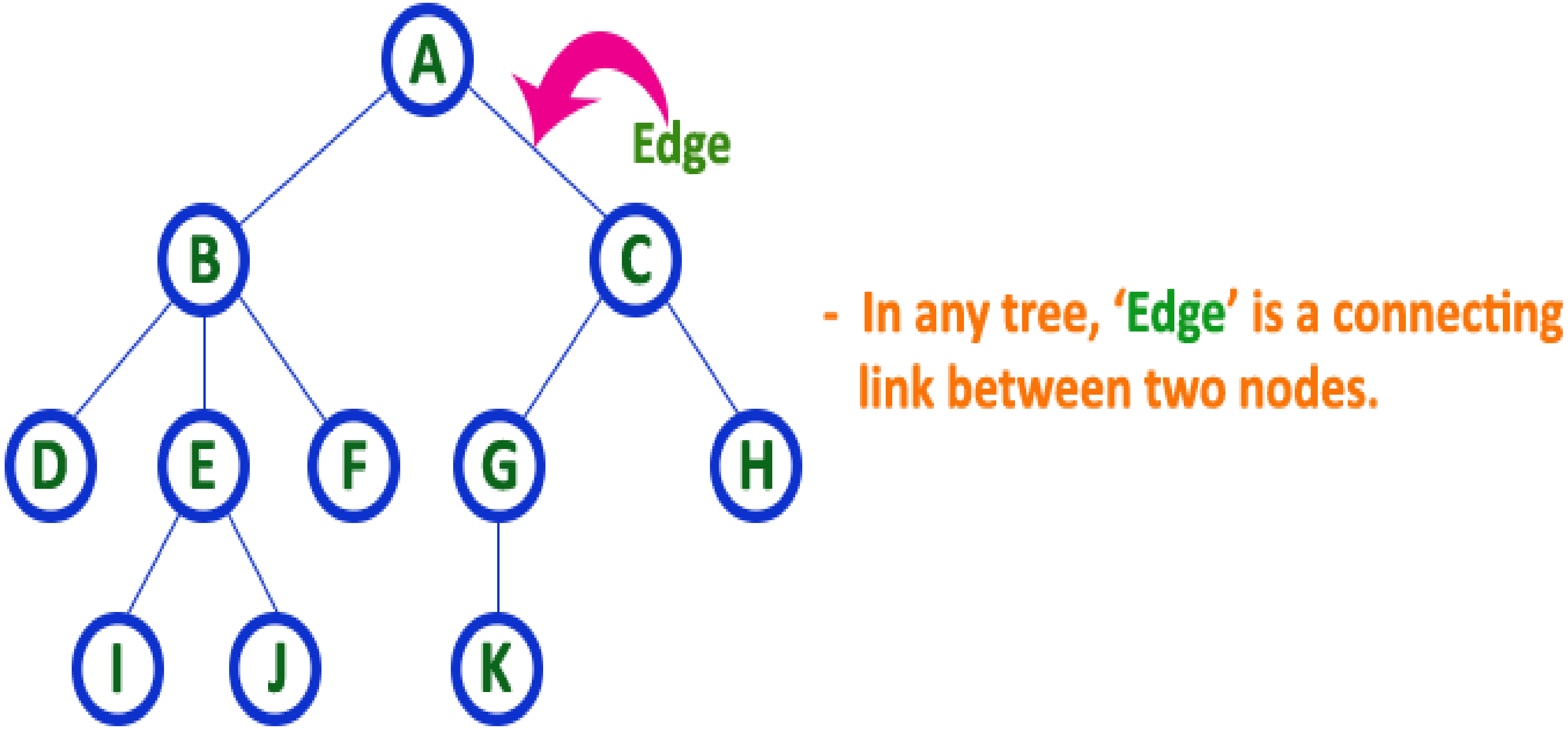
In a tree data structure, the first node is called as **Root Node**.



Null Tree: a tree with no nodes.

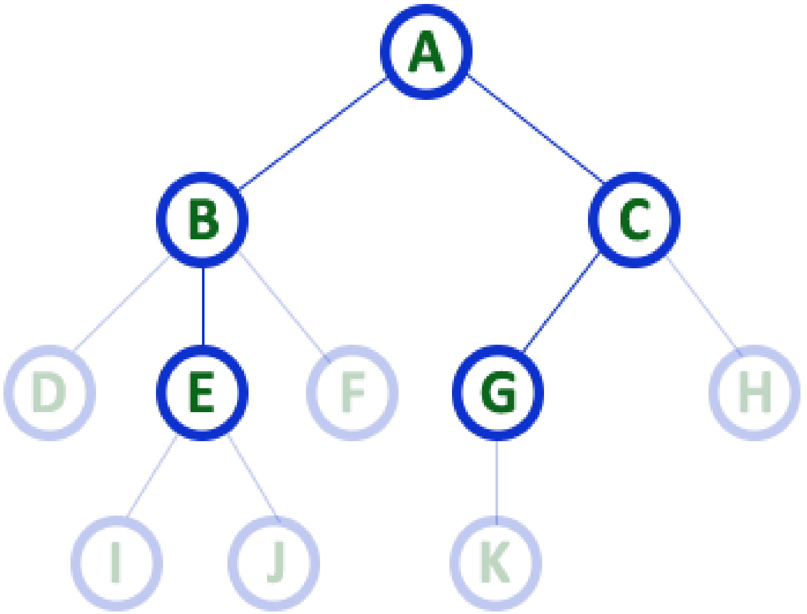
2. Edge

In a tree data structure, the connecting link between any two nodes is called as **EDGE**.



3. Parent

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

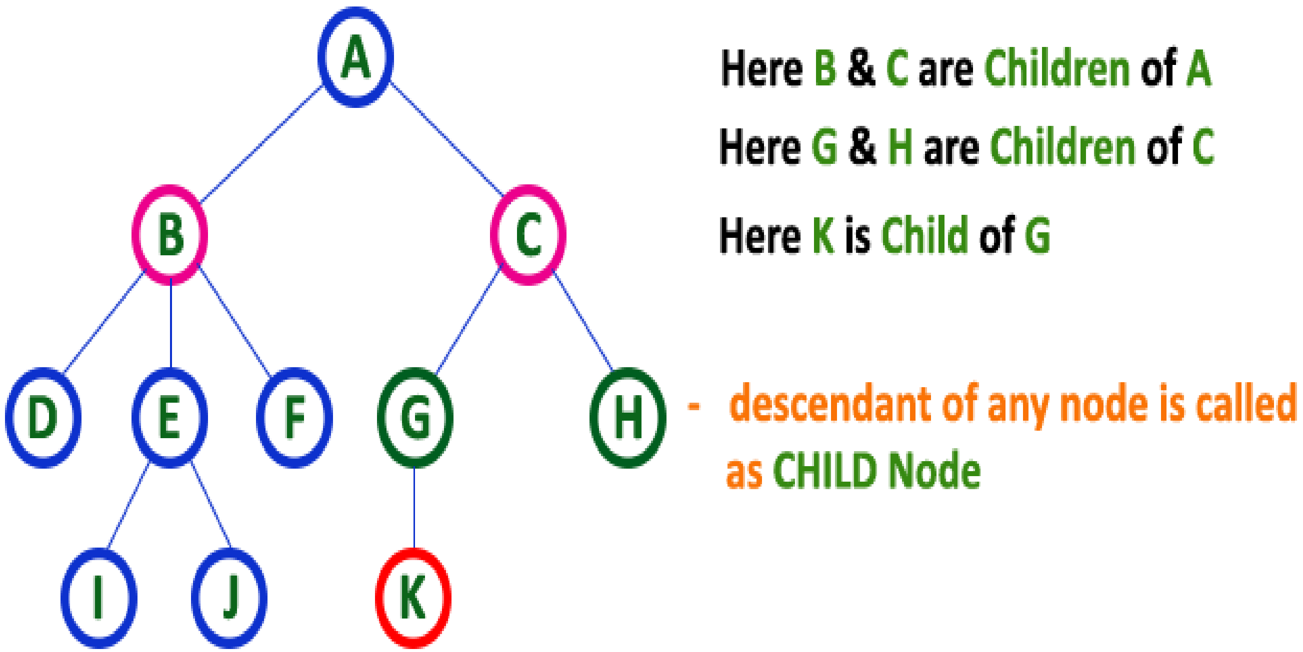


Here A, B, C, E & G are Parent nodes

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

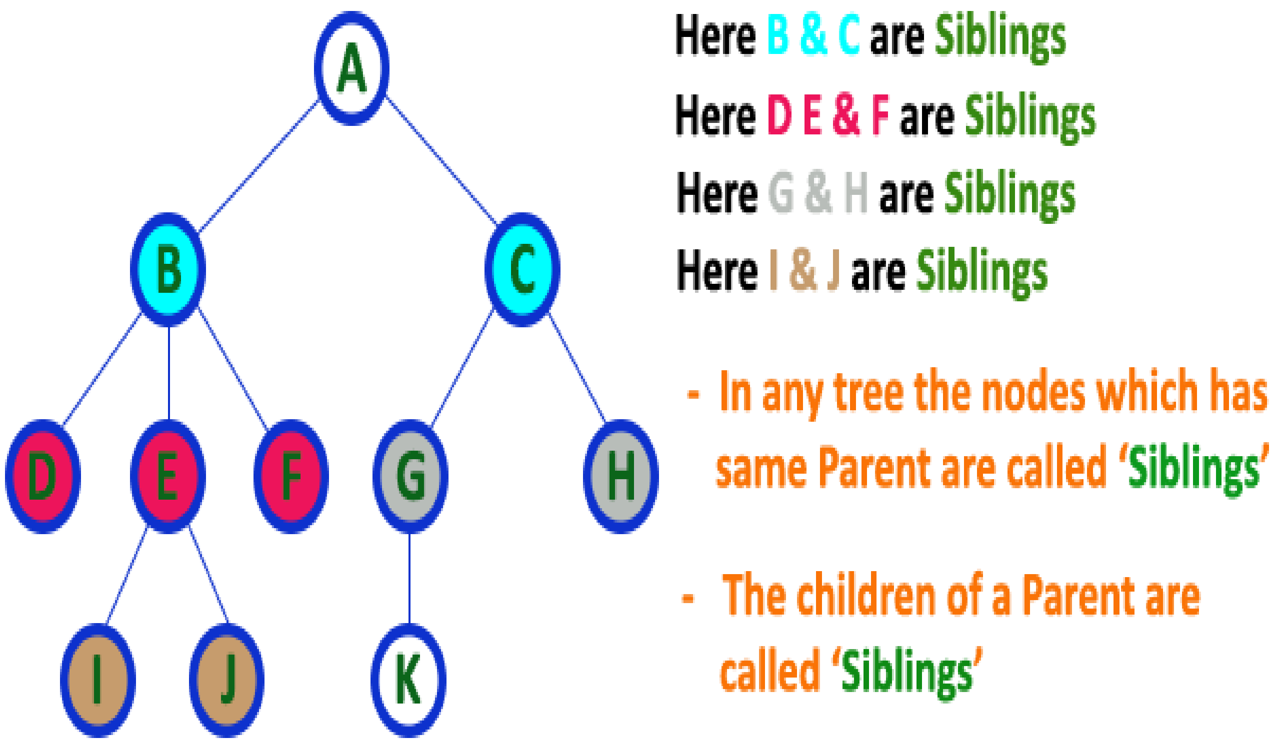
4. Child

In a tree data structure, the node which is descendant of any node is called as **CHILD Node**.



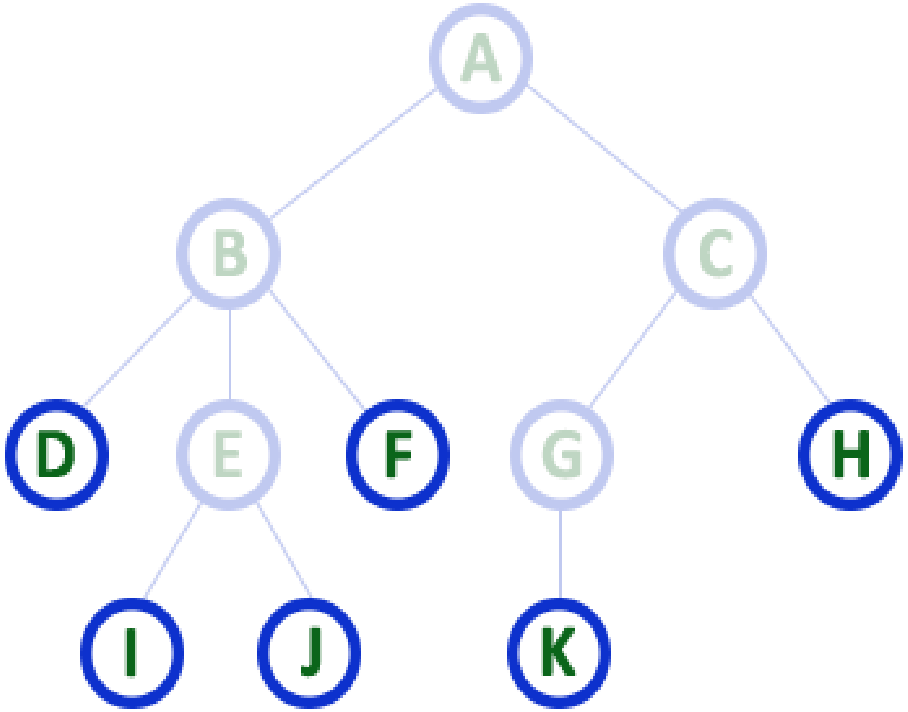
5. Siblings

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.



6. Leaf

In a tree data structure, the node which does not have a child is called as **LEAF Node**.



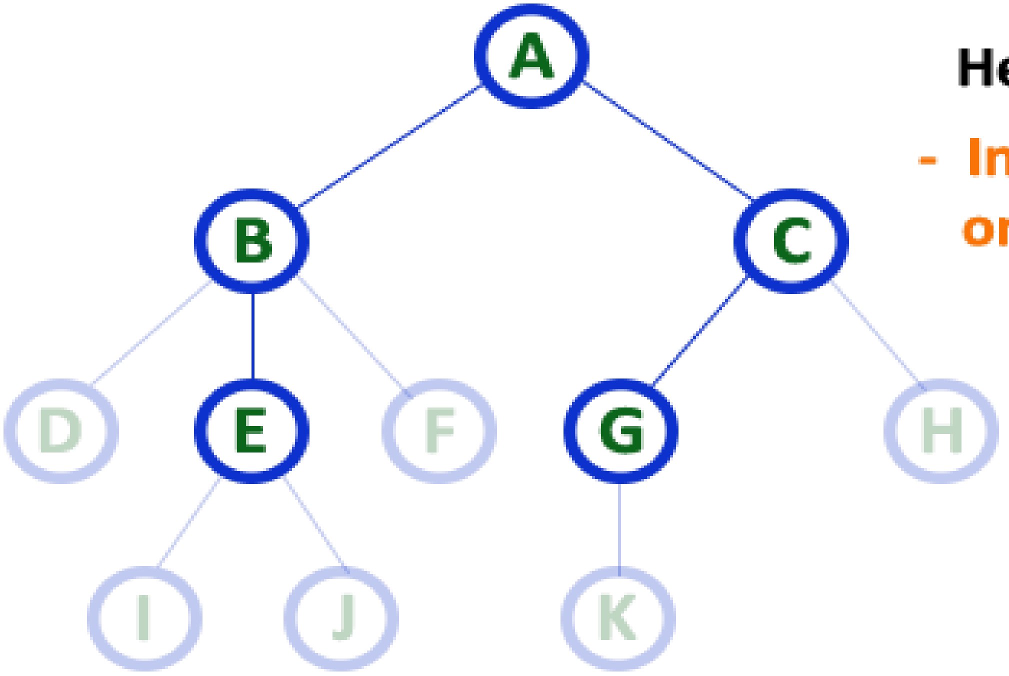
Here D, I, J, F, K & H are Leaf nodes

- In any tree the node which does not have children is called 'Leaf'

- A node without successors is called a 'leaf' node

7. Internal Nodes

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



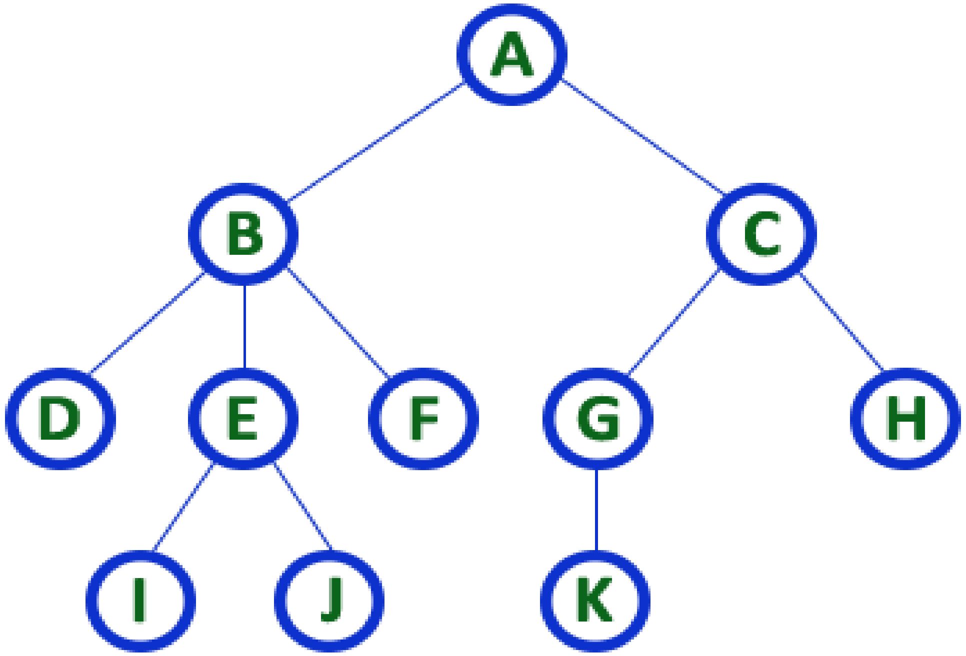
Here A, B, C, E & G are Internal nodes

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

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

8. Degree

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



Here Degree of B is 3

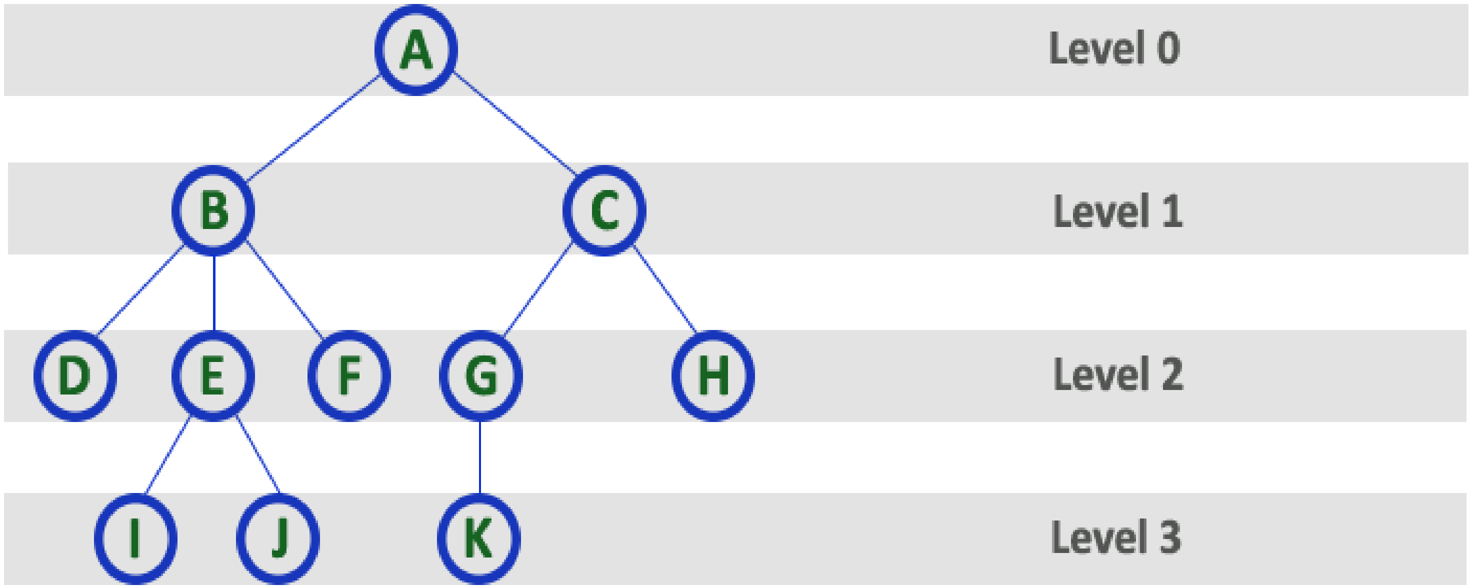
Here Degree of A is 2

Here Degree of F is 0

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

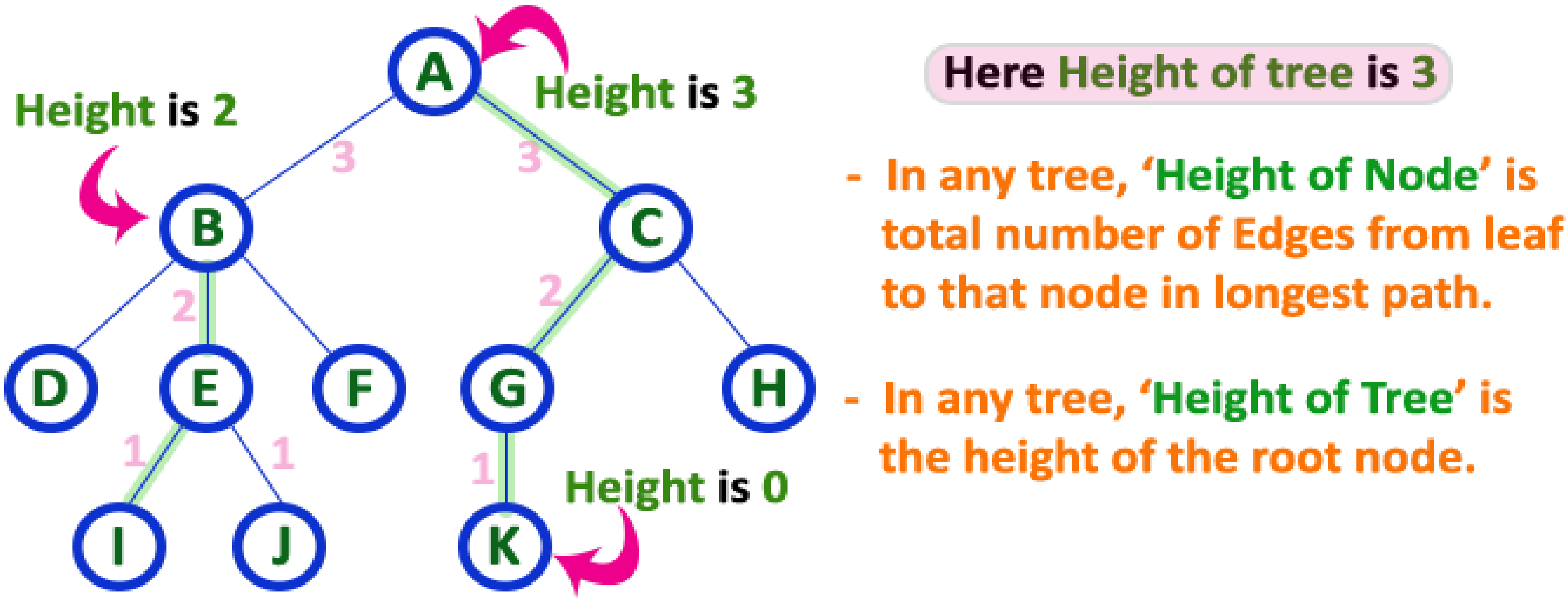
9. Level

In a tree data structure, the root node is said to be at Level 0 and the children of root node are at Level 1 and the children of the nodes which are at Level 1 will be at Level 2 and so on...



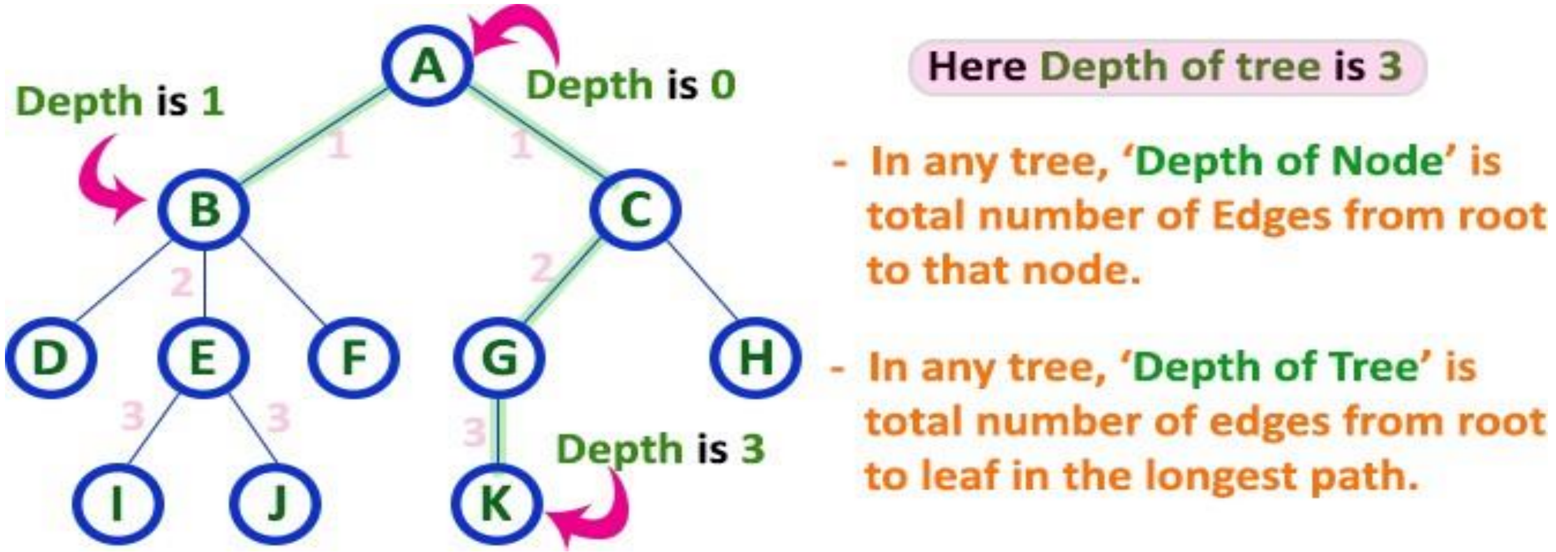
10. Height

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.



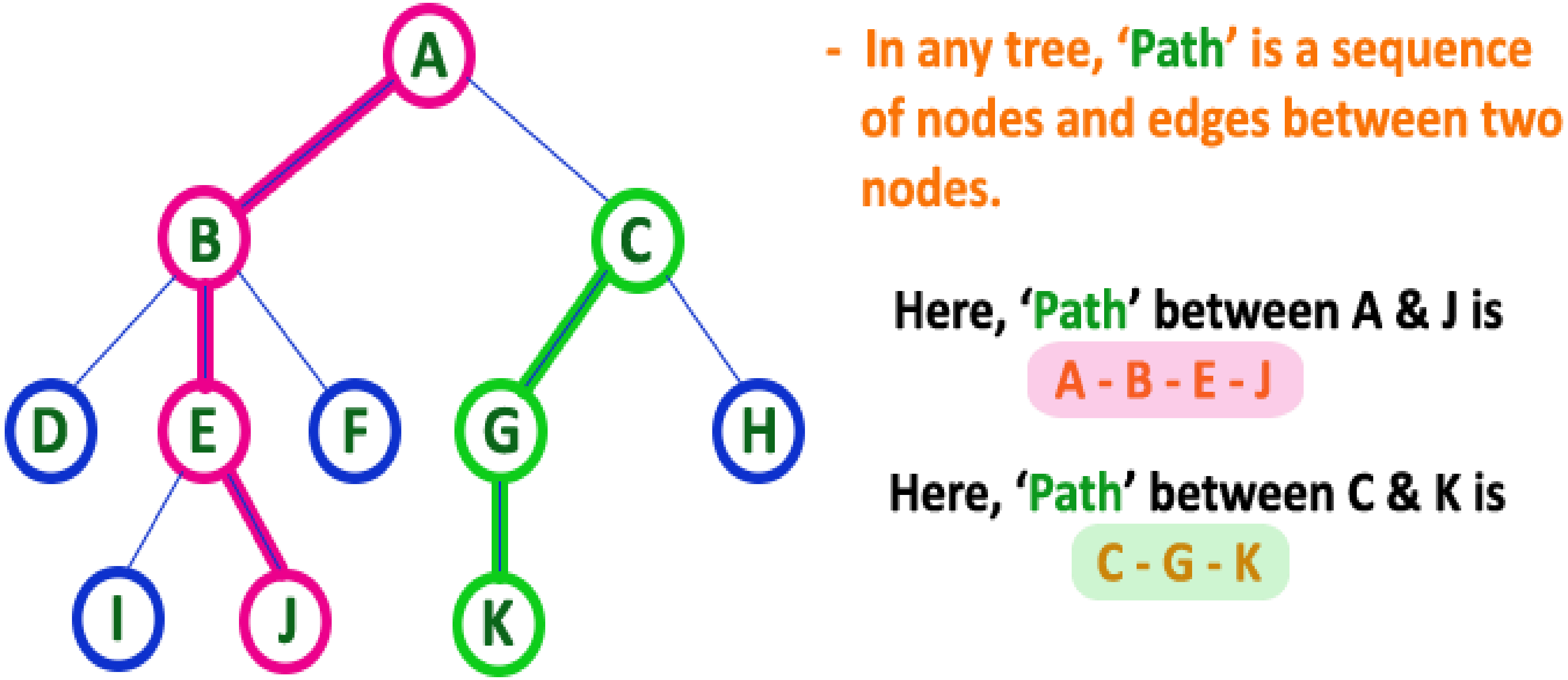
11. Depth

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



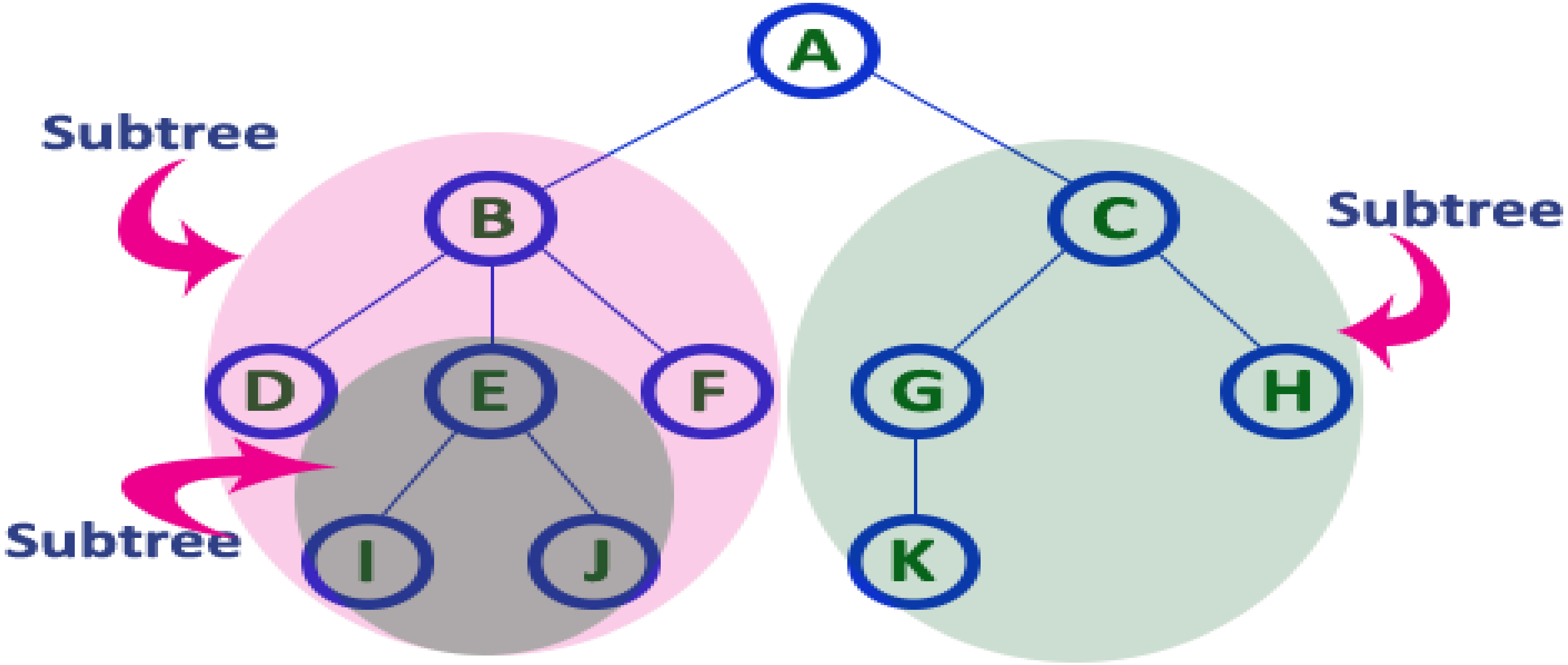
12. Path

In a tree data structure, the sequence of Nodes and Edges from one node to another node is called as **PATH** between that two Nodes.



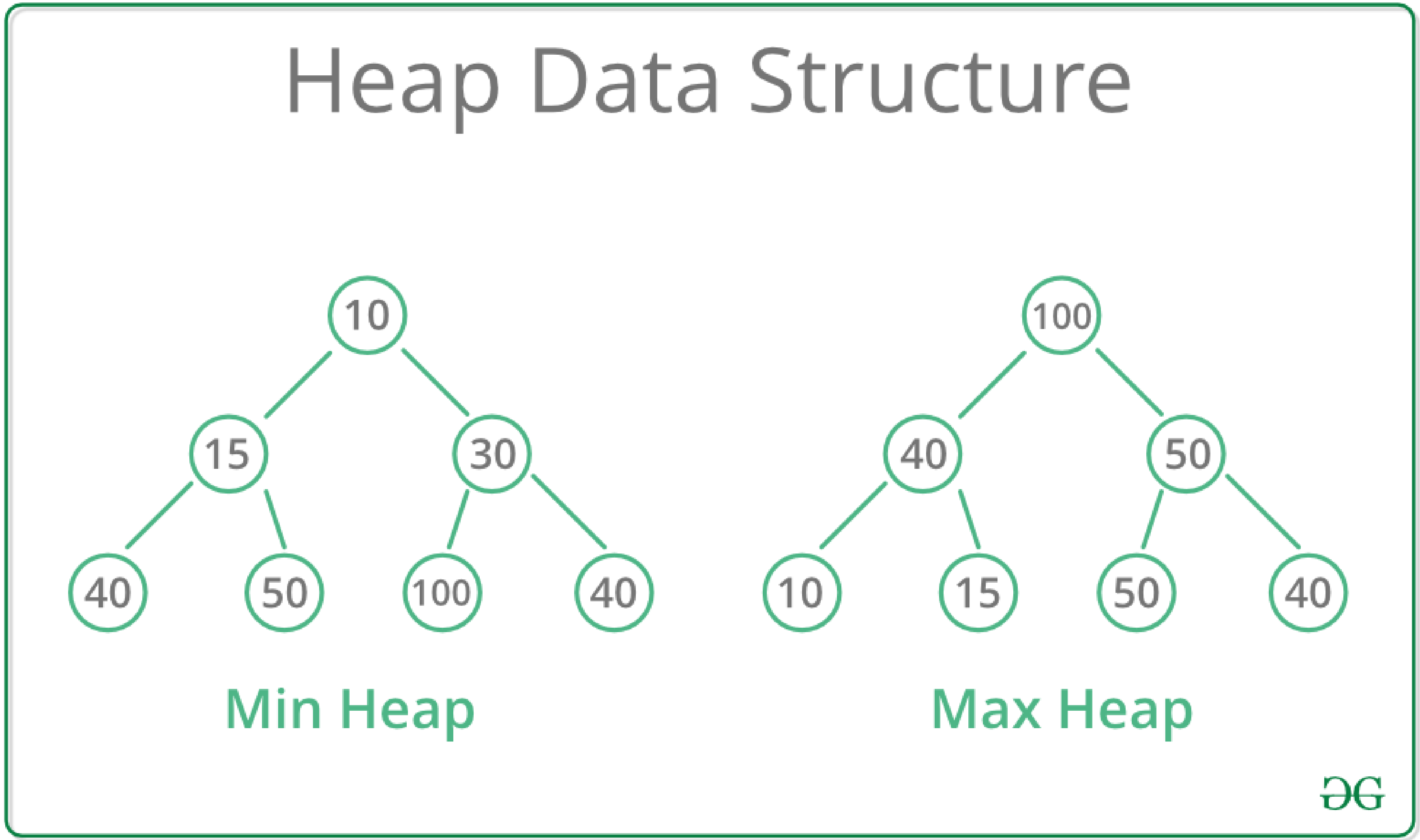
13. Sub Tree

In a tree data structure, each child from a node forms a subtree recursively. Every child node will form a subtree on its parent node.



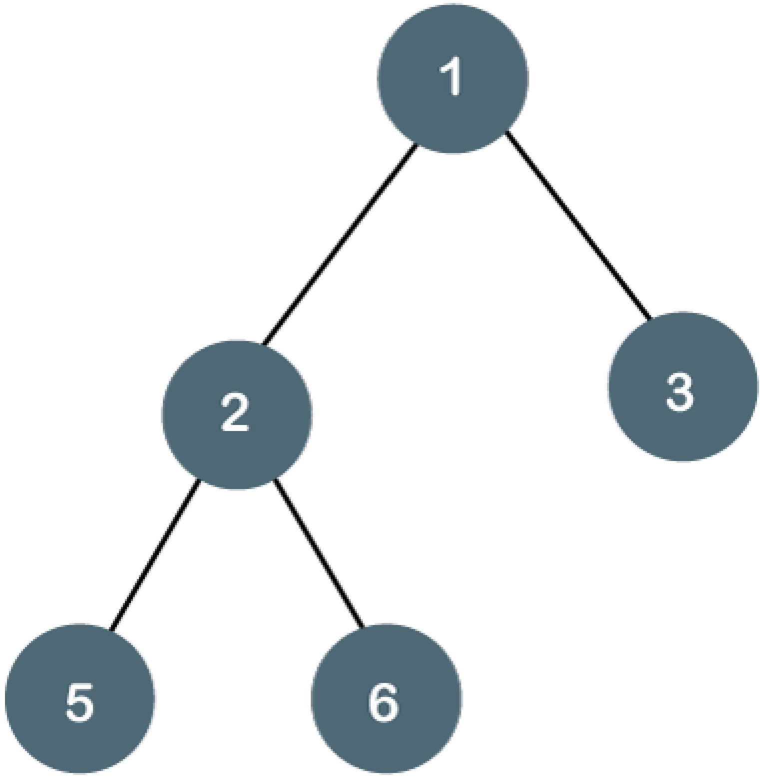
Heap tree

- A heap is a complete binary tree.
- There are two types of the heap:
- Min Heap : The value of the parent node should be less than or equal to either of its children.
- Max heap : The value of the parent node is greater than or equal to its children.


Heap Data Structure
10
15
30
40
50
100
40
Min Heap
100
40
50
10
15
50
40
Max Heap

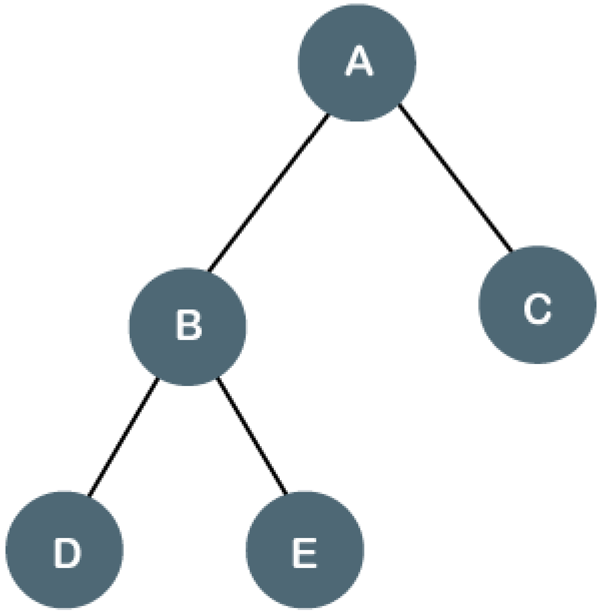
Binary Tree

- The Binary tree means that the node can have maximum two children.



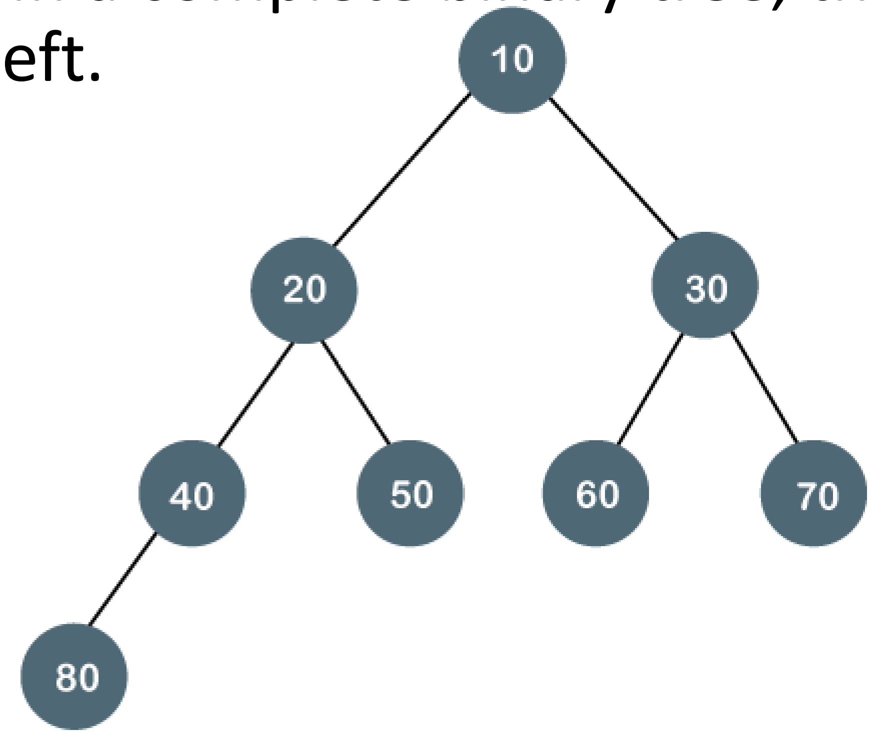
Full/ proper/ strict Binary tree

- 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.



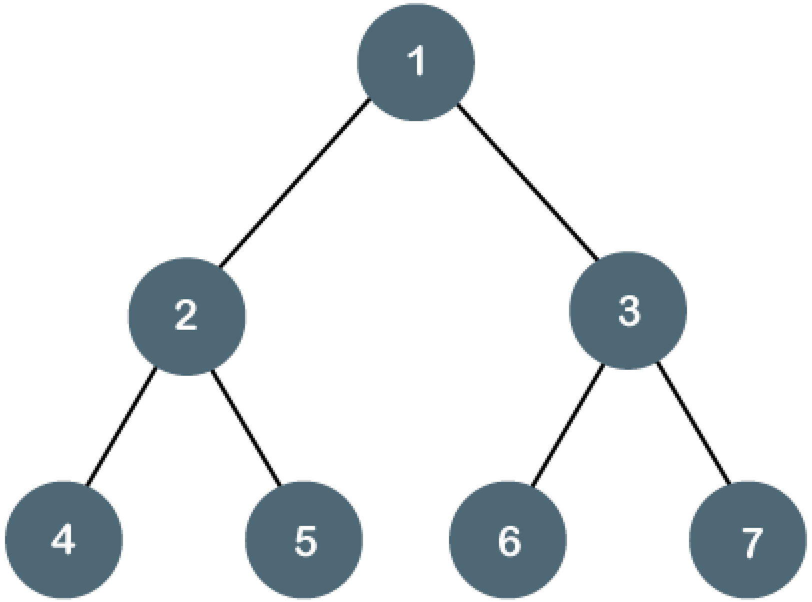
complete binary tree

- The complete binary tree is a tree in which all the nodes are completely filled except the last level. In the last level, all the nodes must be as left as possible. In a complete binary tree, the nodes should be added from the left.



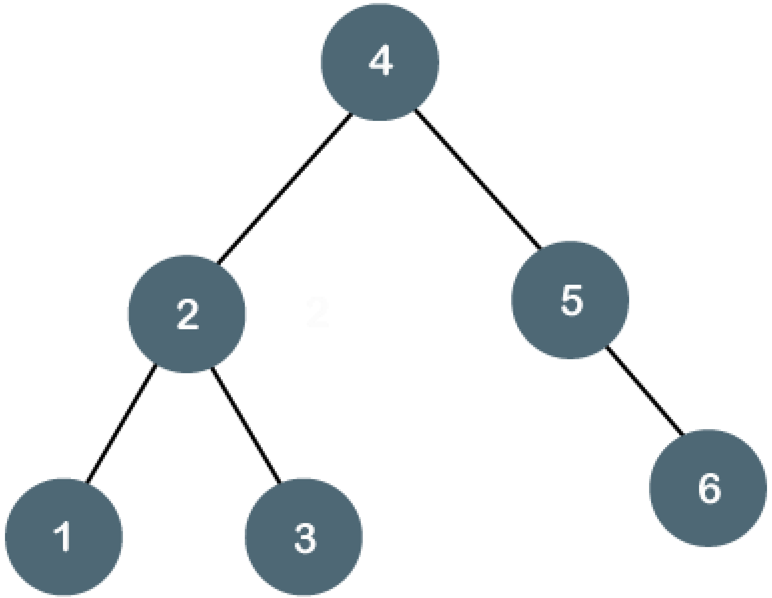
perfect binary tree

- 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.



balanced binary tree

- The balanced binary tree is a tree in which both the left and right trees differ by atmost 1. For example, ***AVL*** and ***Red-Black trees*** are balanced binary tree.

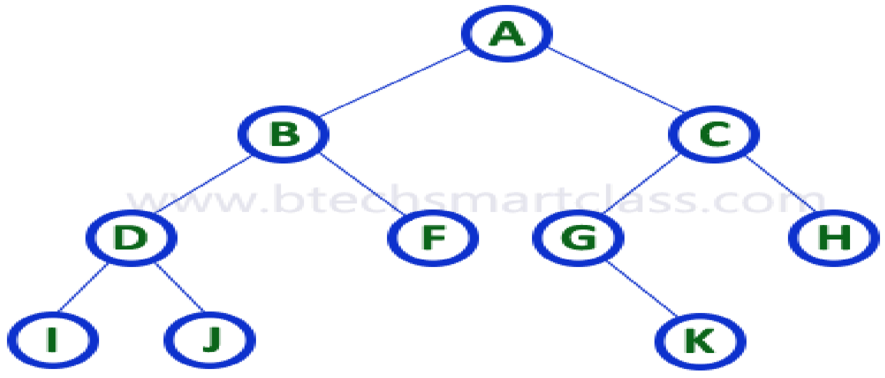


Tree Representations

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

1. **Array Representation**
2. **Linked List Representation**

Consider the following binary tree...

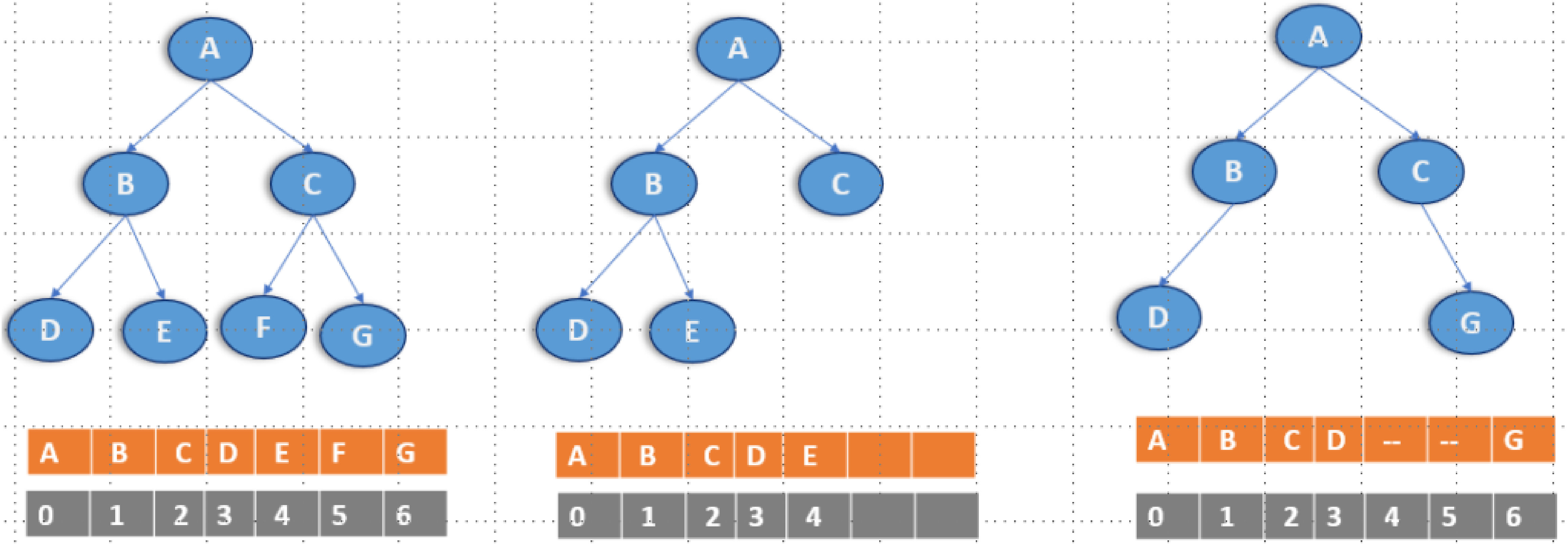


1. Array Representation of Binary Tree

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

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

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



Tree 1 - Complete Tree

Tree 2 - Complete Tree

Tree 3 - Not a Complete Tree

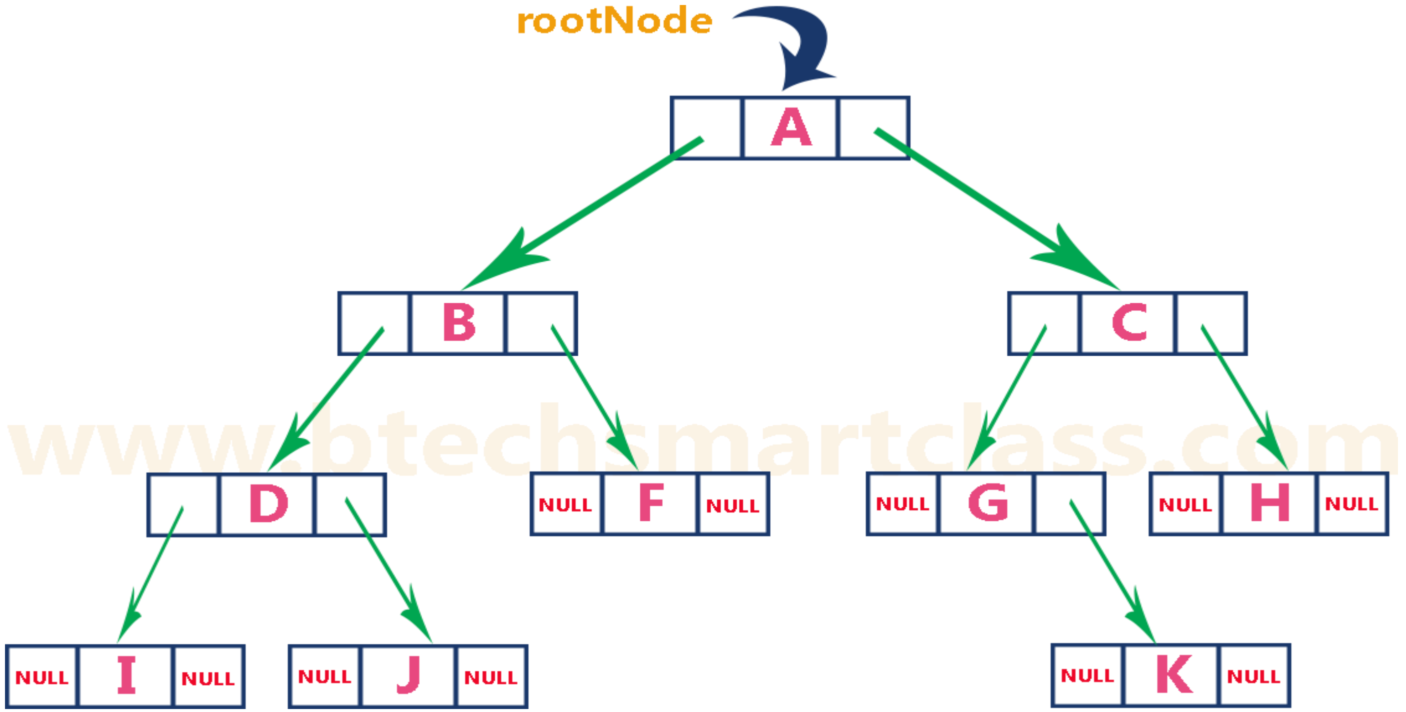
2. Linked List Representation of Binary Tree

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...

Left Child Address	Data	Right Child Address

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



Function to create New Node in Tree:

```
struct node *getNewNode(int val)
{
struct node *newNode = malloc(sizeof(struct node));
newNode->key = val;
newNode->left = NULL;
newNode->right = NULL;
return newNode;
}
```

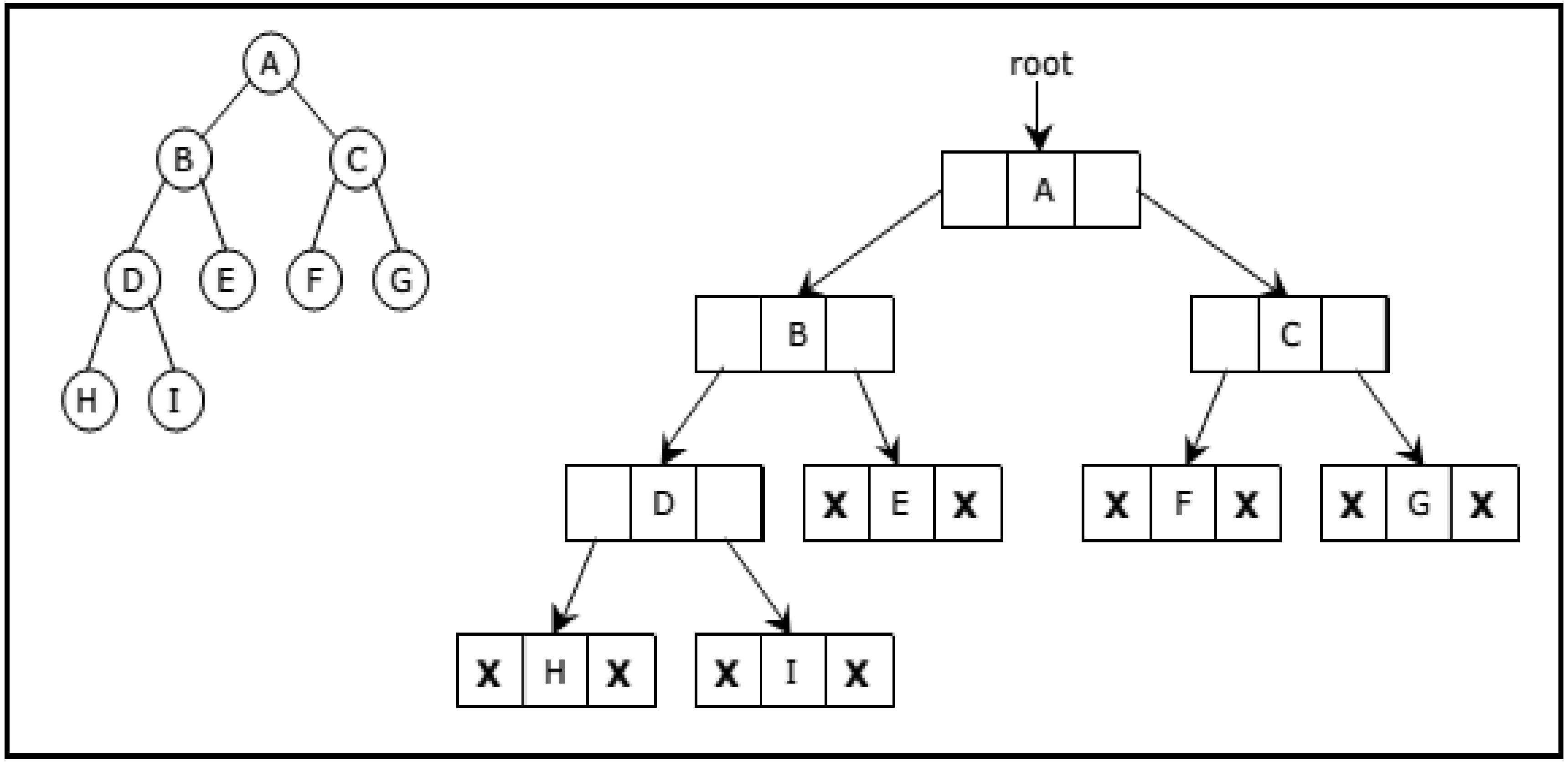
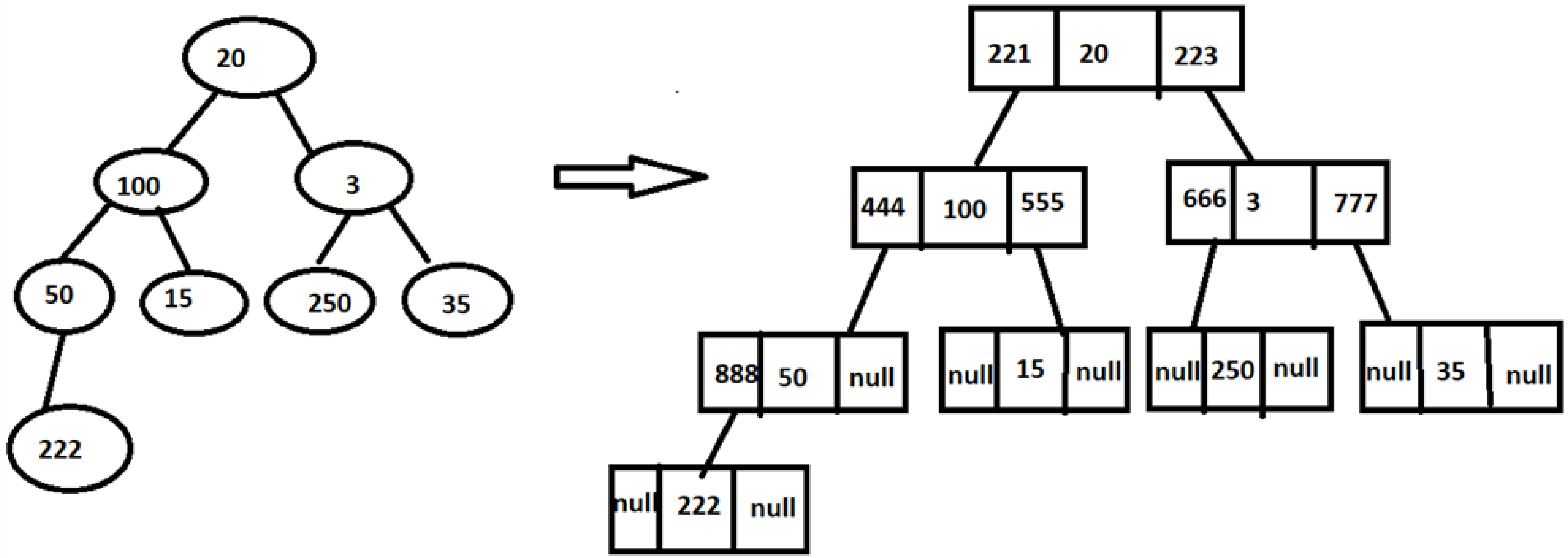


Figure 5.2.7. Linked representation for the binary tree



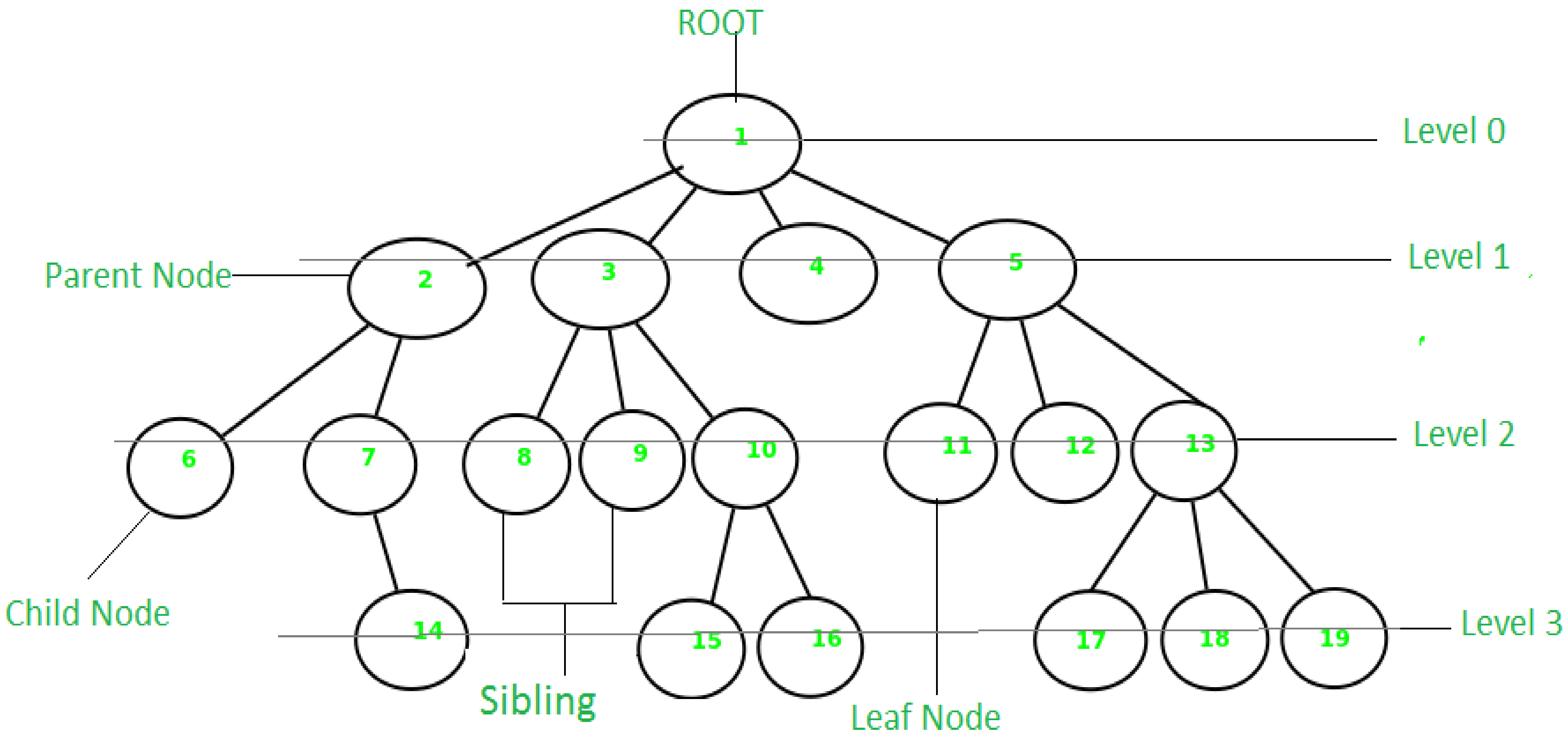
20
100
3
50
15
250
35
222
221
20
223
444
100
555
666
3
777
888
50
null
null
15
null
null
250
null
null
35
null
null
222
null

Types of tree

- General tree
- Binary tree
- Binary search tree

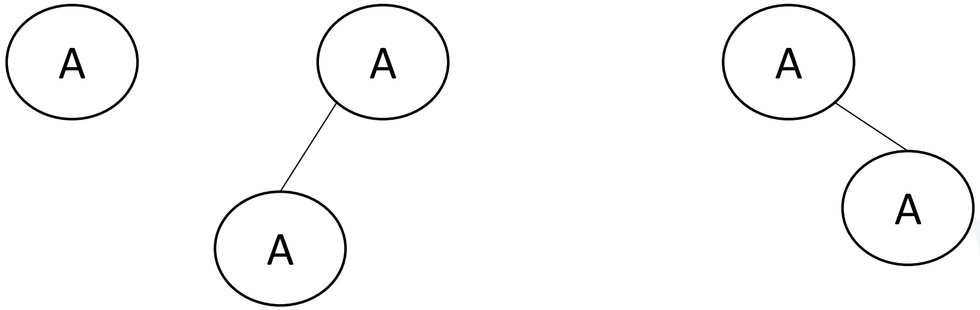
General tree

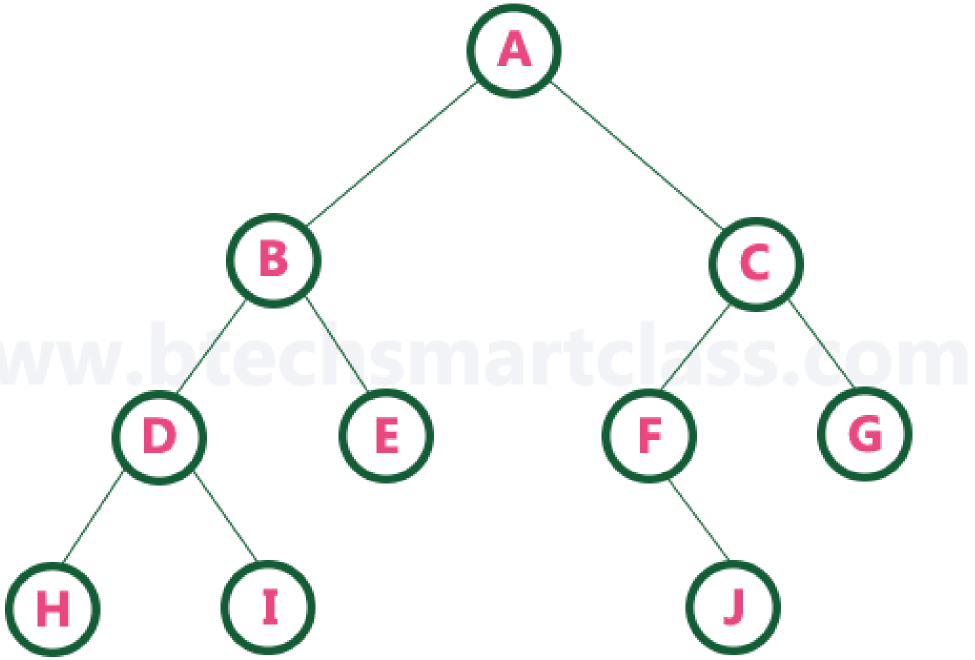
- The general tree is one of the types of tree data structure.
- In the general tree, a node can have either 0 or maximum n number of nodes.
- There is no restriction imposed on the degree of the node (the number of nodes that a node can contain).
- The topmost node in a general tree is known as a root node.
- The children of the parent node are known as ***subtrees***.
- There can be ***n*** number of subtrees in a general tree.
- In the general tree, the subtrees are unordered as the nodes in the subtree cannot be ordered.
- Every non-empty tree has a downward edge, and these edges are connected to the nodes known as ***child nodes***. The root node is labeled with level 0. The nodes that have the same parent are known as ***siblings***.

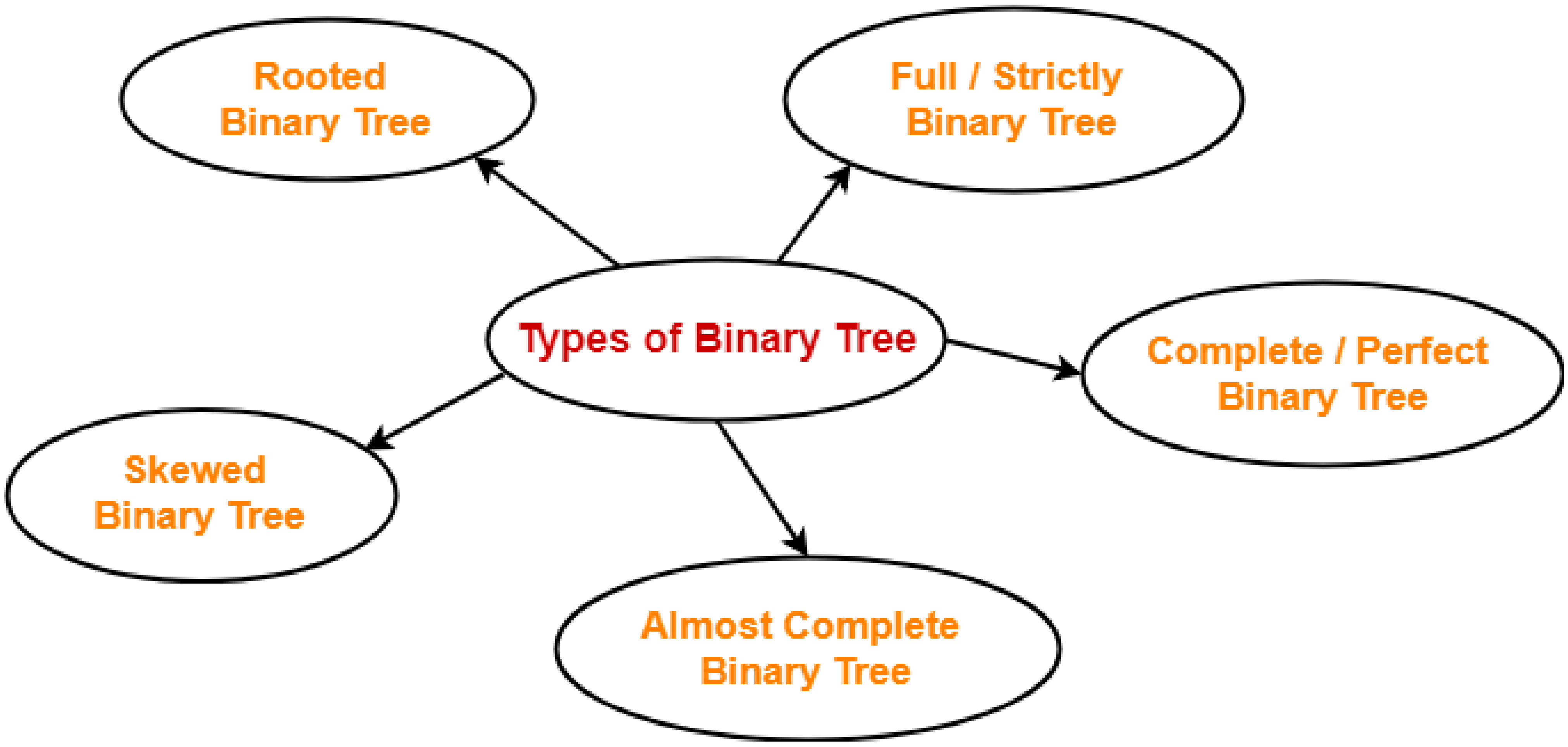

ROOT
Level 0
1
Parent Node
2
3
4
5
Level 1
6
7
8
9
10
11
12
13
Level 2
Child Node
14
15
16
17
18
19
Level 3
Sibling
Leaf Node

binary tree

- Here, binary name itself suggests two numbers, i.e., 0 and 1.
- In a binary tree, each node in a tree can have utmost two child nodes. Here, utmost means whether the node has 0 nodes, 1 node or 2 nodes.






Rooted
Binary Tree
Full / Strictly
Binary Tree
Types of Binary Tree
Complete / Perfect
Binary Tree
Skewed
Binary Tree
Almost Complete
Binary Tree

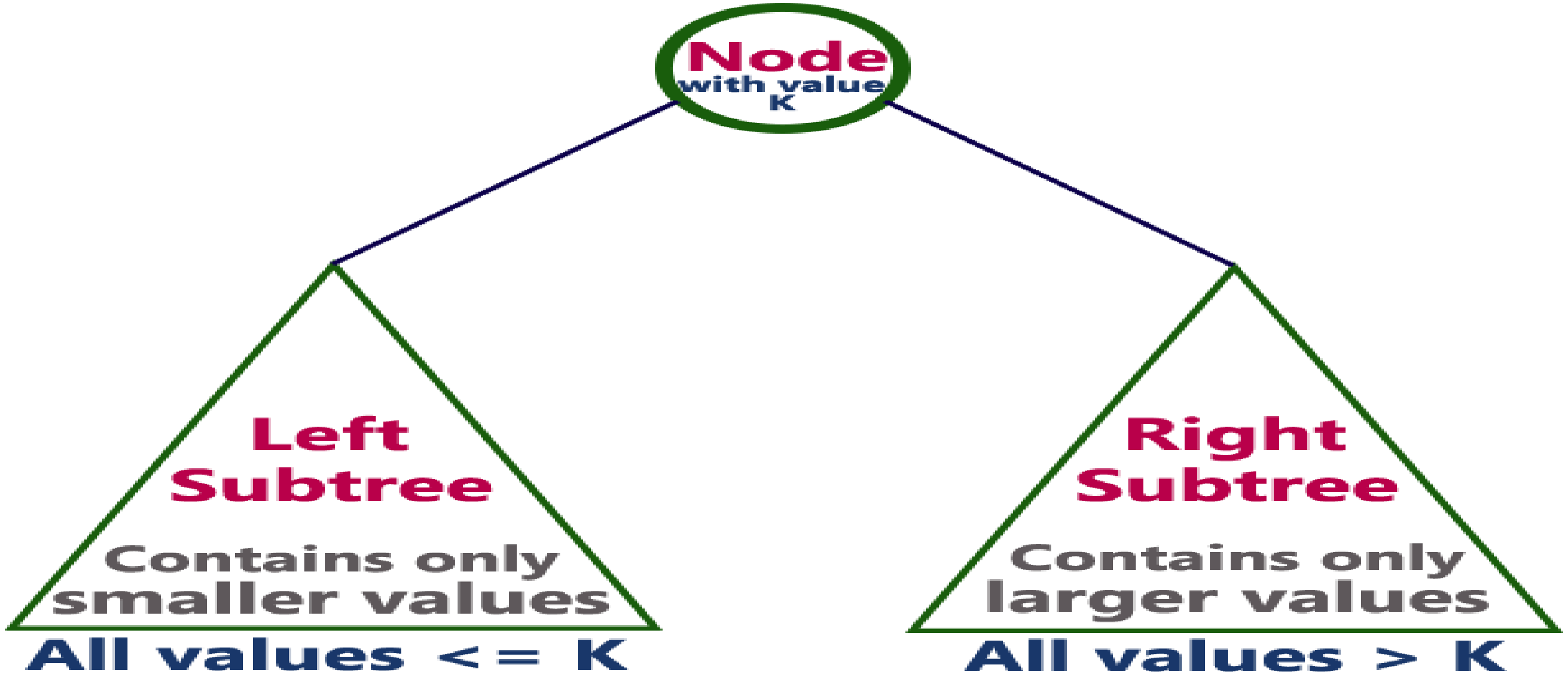
Binary Search Tree

Binary search tree can be defined as follows...

Binary Search Tree is a binary tree in which every node contains only smaller values

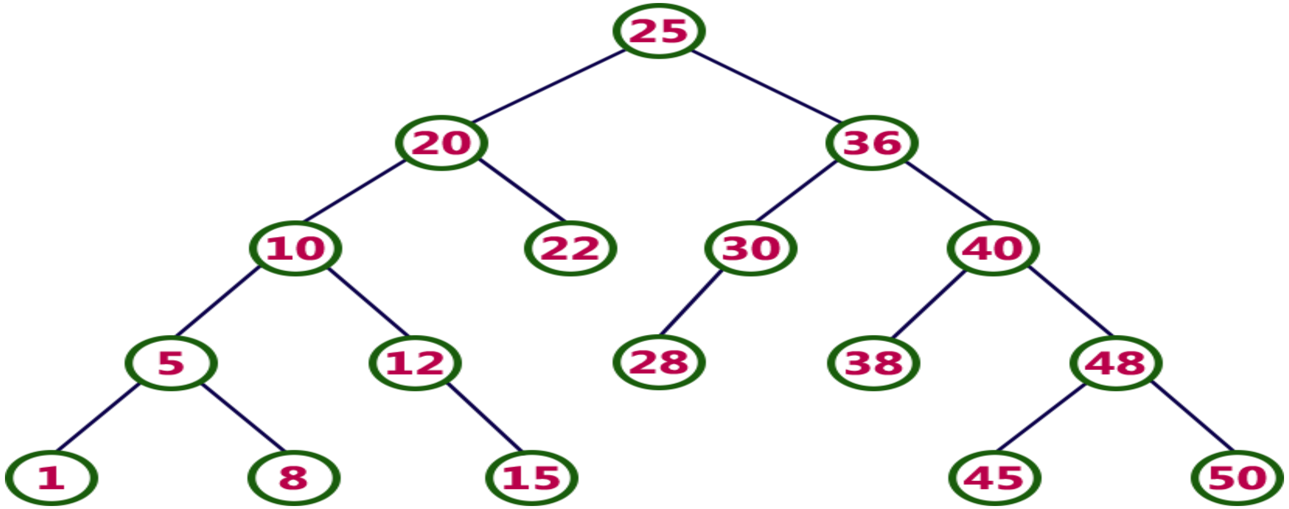
in its left subtree and only larger values in its right subtree.

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



Example

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



E

every binary search tree is a binary tree but every binary tree need

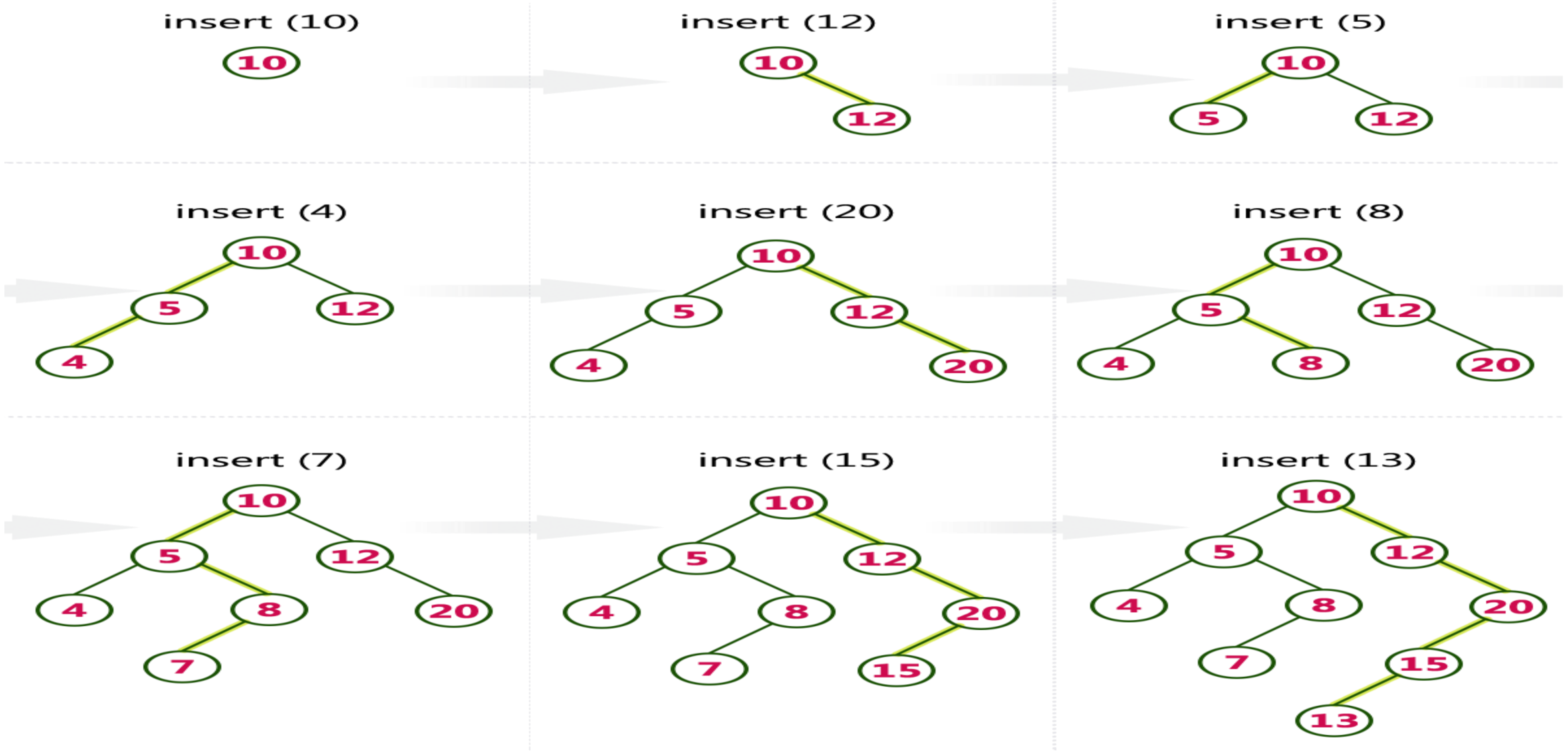
not to be binary search tree.

Example

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

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

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



Construct a Binary Search Tree (BST) for the following sequence of numbers-
50, 70, 60, 20, 90, 10, 40, 100

When elements are given in a sequence,

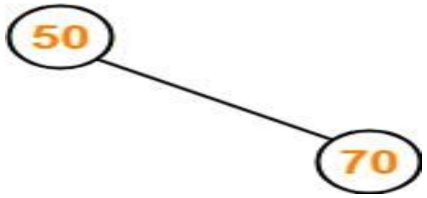
- Always consider the first element as the root node.
- Consider the given elements and insert them in the BST one by one.

The binary search tree will be constructed as explained below-

Insert 50-

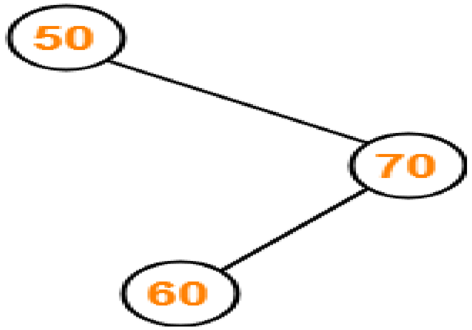
Insert 70-

- As 70 > 50, so insert 70 to the right of 50.



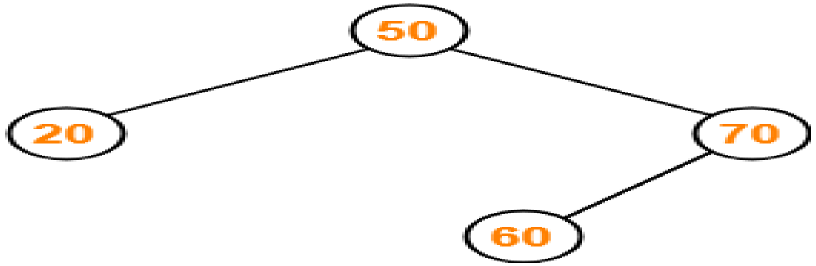
Insert 60-

- As 60 > 50, so insert 60 to the right of 50.
- As 60 < 70, so insert 60 to the left of 70.



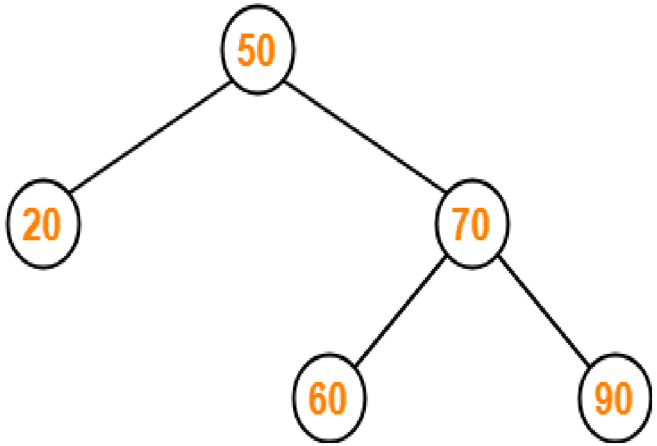
Insert 20-

- As 20 < 50, so insert 20 to the left of 50.



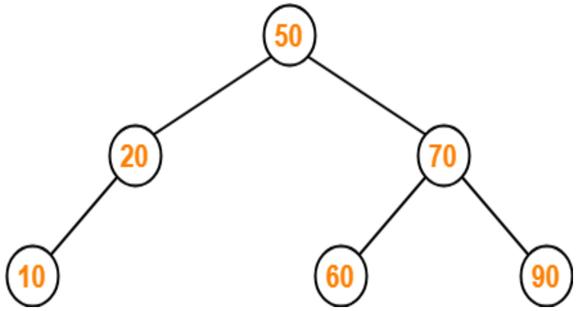
Insert 90-

- As 90 > 50, so insert 90 to the right of 50.
- As 90 > 70, so insert 90 to the right of 70.



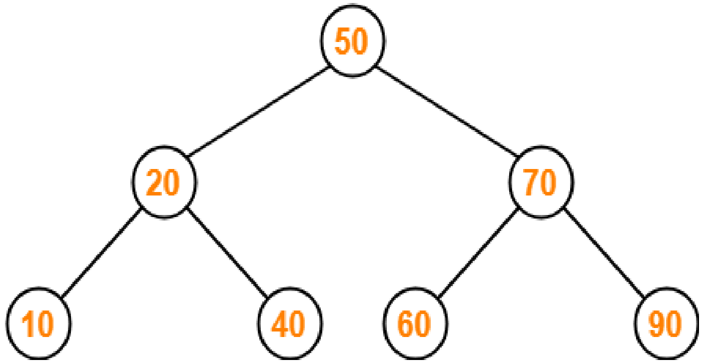
Insert 10-

- As 10 < 50, so insert 10 to the left of 50.
- As 10 < 20, so insert 10 to the left of 20.



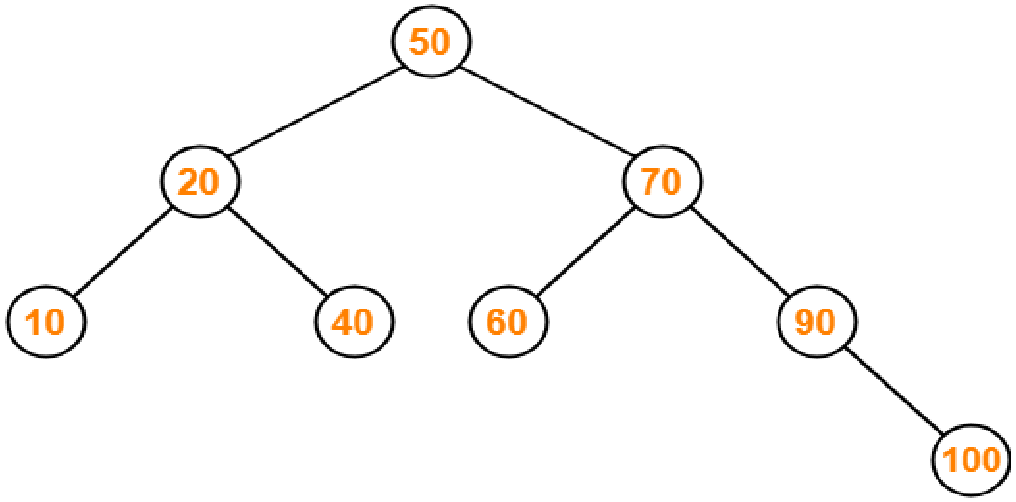
Insert 40-

- As 40 < 50, so insert 40 to the left of 50.
- As 40 > 20, so insert 40 to the right of 20.



Insert 100-

- As 100 > 50, so insert 100 to the right of 50.
- As 100 > 70, so insert 100 to the right of 70.
- As 100 > 90, so insert 100 to the right of 90.



Binary Search Tree

Practice Problems for BST

1. A binary search tree is generated by inserting in order of the following integers-

50, 15, 62, 5, 20, 58, 91, 3, 8, 37, 60, 24

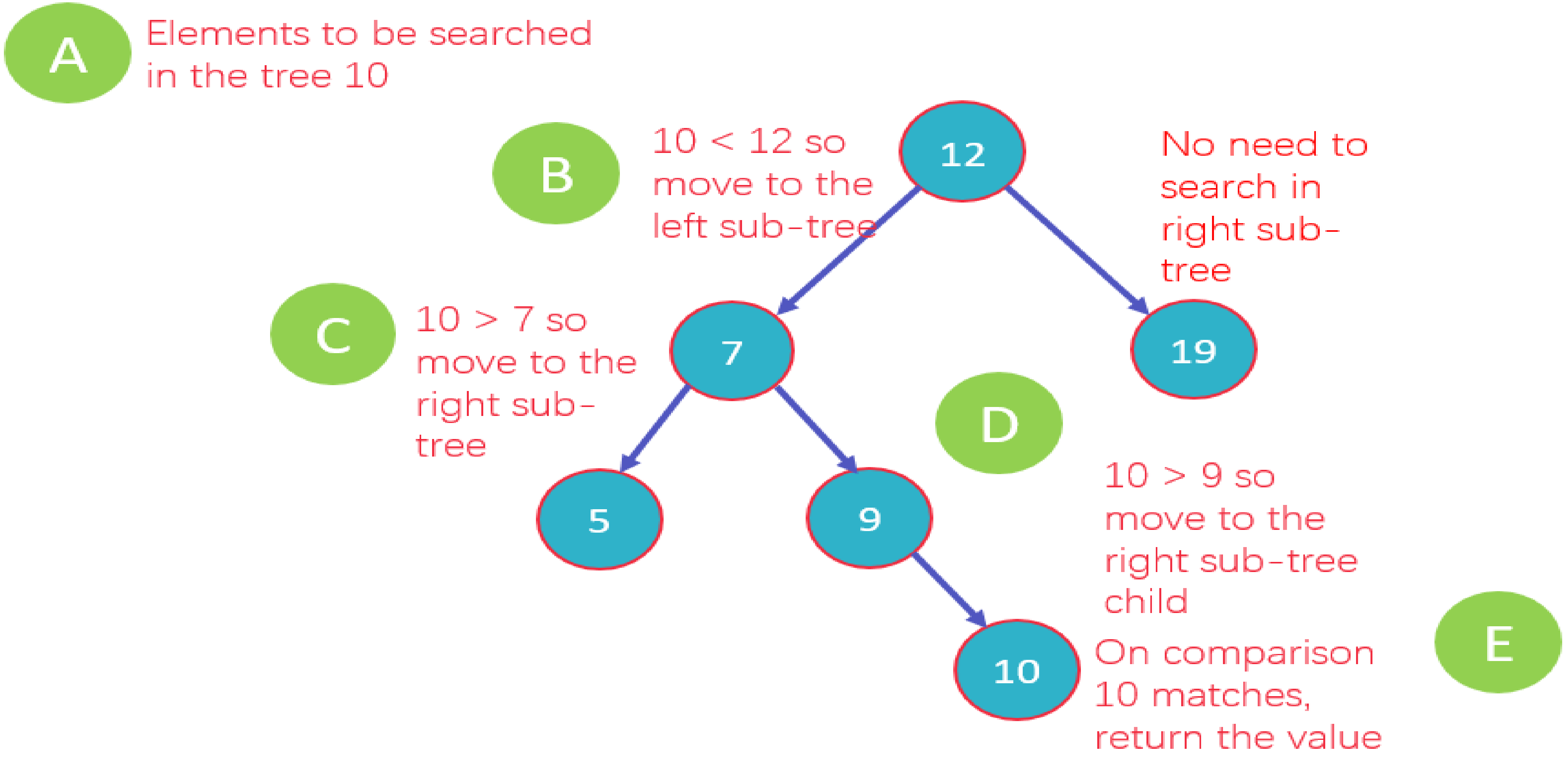
2. A binary search tree is generated by inserting in order of the following integers-

45, 15, 79, 90, 10, 55, 12, 20, 50

Insertion Operation in BST

- In a binary search tree, the insertion operation is performed with **O(log n)** time complexity. In binary search tree, new node is always inserted as a leaf node. The insertion operation is performed as follows...
- **Step 1 -** Create a newNode with given value and set its **left** and **right** to **NULL**.
- **Step 2 -** Check whether tree is Empty.
- **Step 3 -** If the tree is **Empty**, then set **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**.

Search Operation



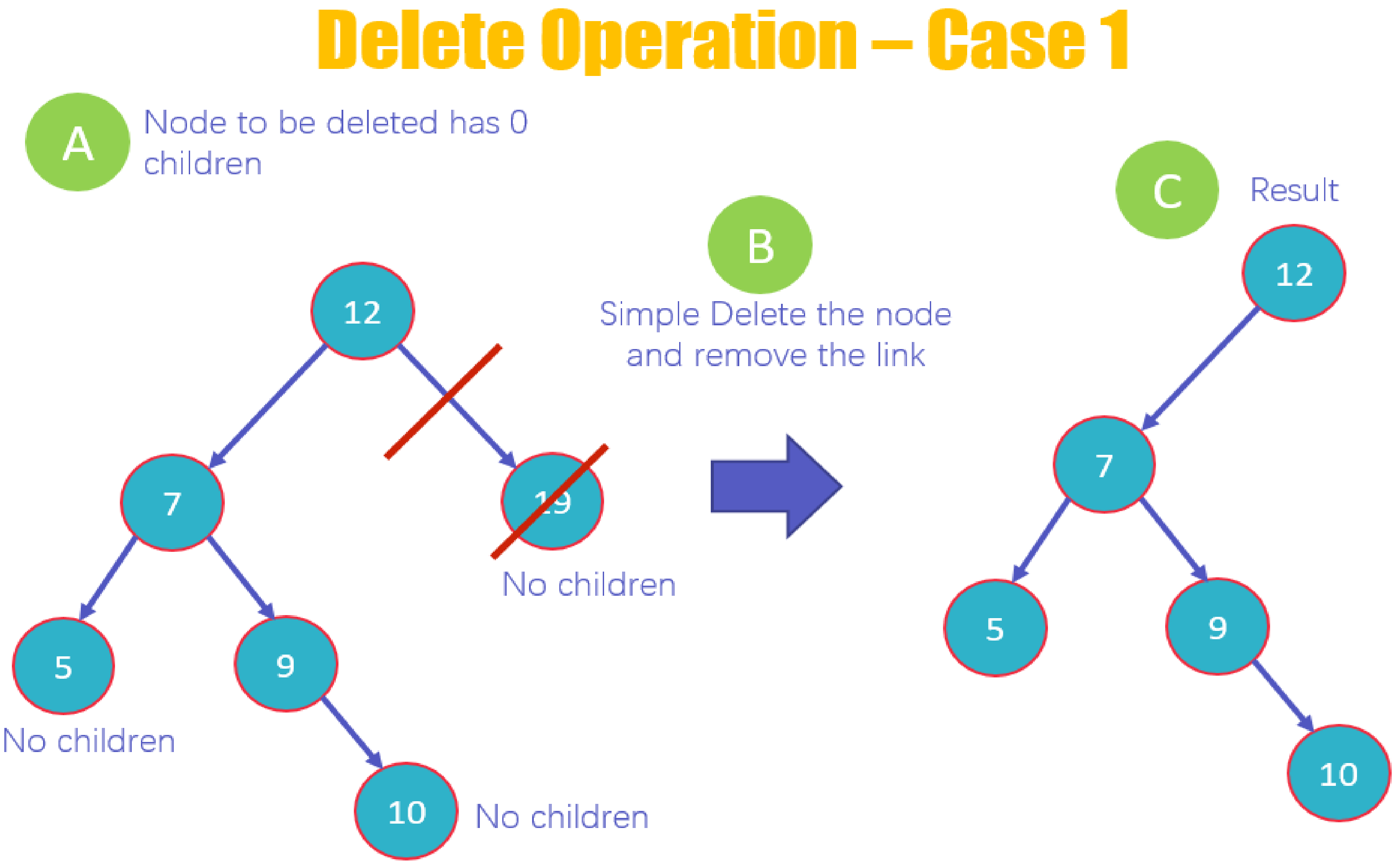
Deletion Operation in BST

- In a binary search tree, the deletion operation is performed with **O(log n)** time complexity. Deleting a node from Binary search tree includes following three cases...
- **Case 1: Deleting a Leaf node (A node with no children)**
- **Case 2: Deleting a node with one child**
- **Case 3: Deleting a node with two children**

Case 1: Deleting a leaf node

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

- **Step 1 - Find** the node to be deleted using **search operation**
- **Step 2 -** Delete the node using **free** function (If it is a leaf) and terminate the function.

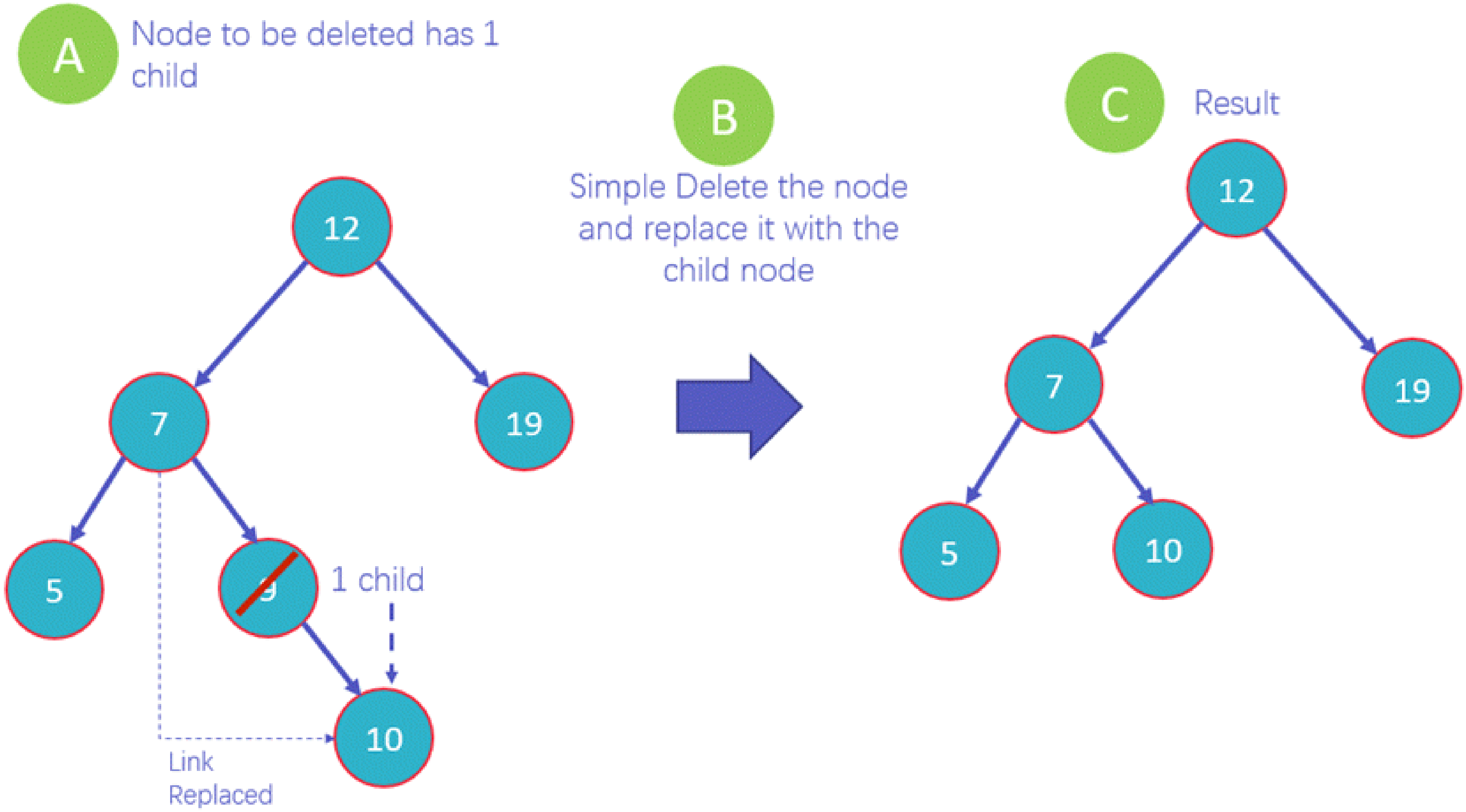


Case 2: Deleting a node with one child

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

- **Step 1 - Find** the node to be deleted using **search operation**
- **Step 2 -** If it has only one child then create a link between its parent node and child node.
- **Step 3 -** Delete the node using **free** function and terminate the function.

Delete Operation – Case 2



Case 3: Deleting a node with two children

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

- **Step 1 - Find** the node to be deleted using **search operation**
- **Step 2** - If it has two children, then find the **largest** node in its **left subtree** (OR) the **smallest** node in its **right subtree**.
- **Step 3 - Swap** both **deleting node** and node which 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.

Delete Operation – Case 3 (a)

A

Node to be deleted has 2 child
Replace Situation: In Order
Predecessor

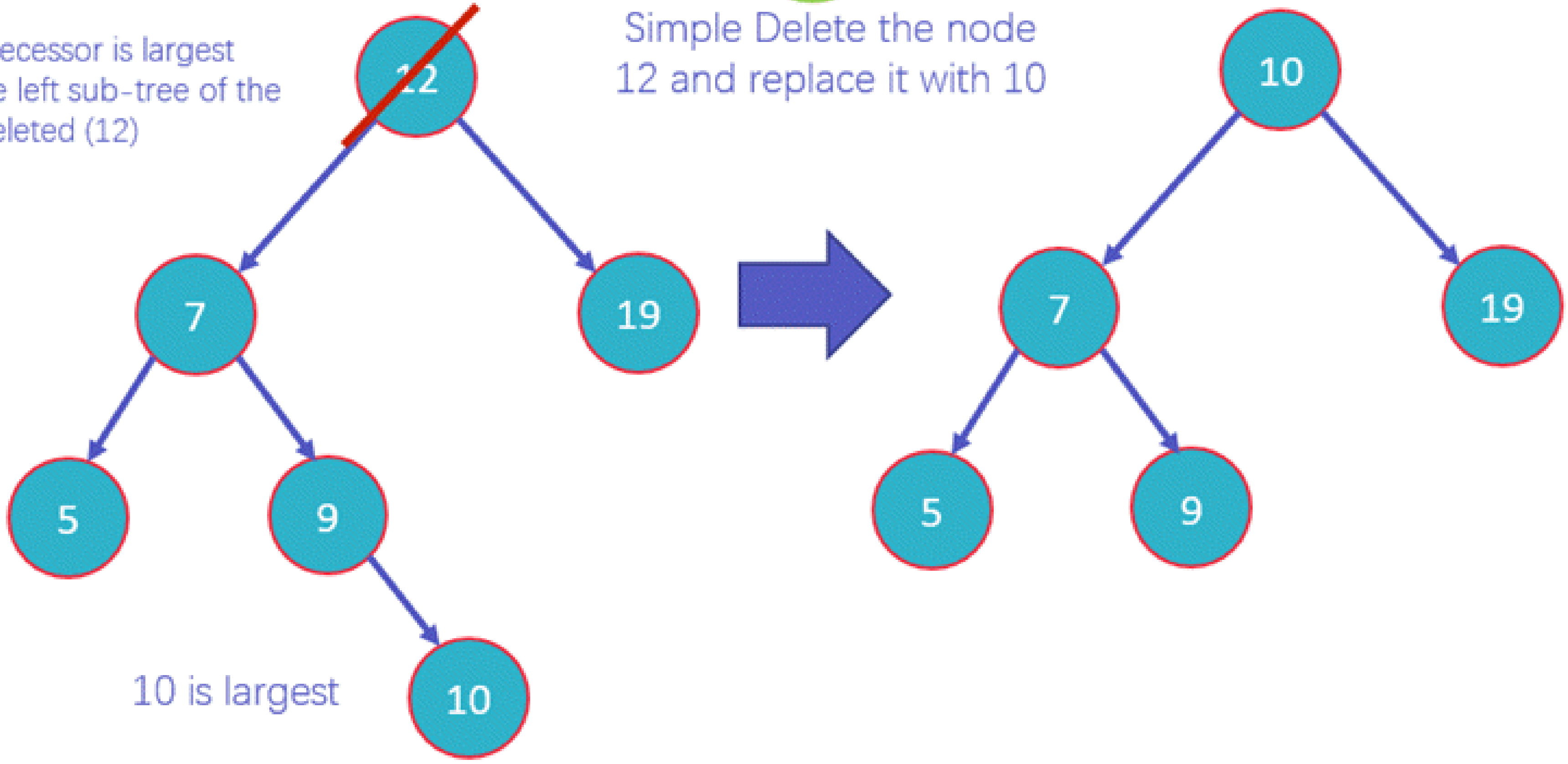
In Order predecessor is largest
element in the left sub-tree of the
node to be deleted (12)

B

C

D

Result



Delete Operation – Case 3 (b)

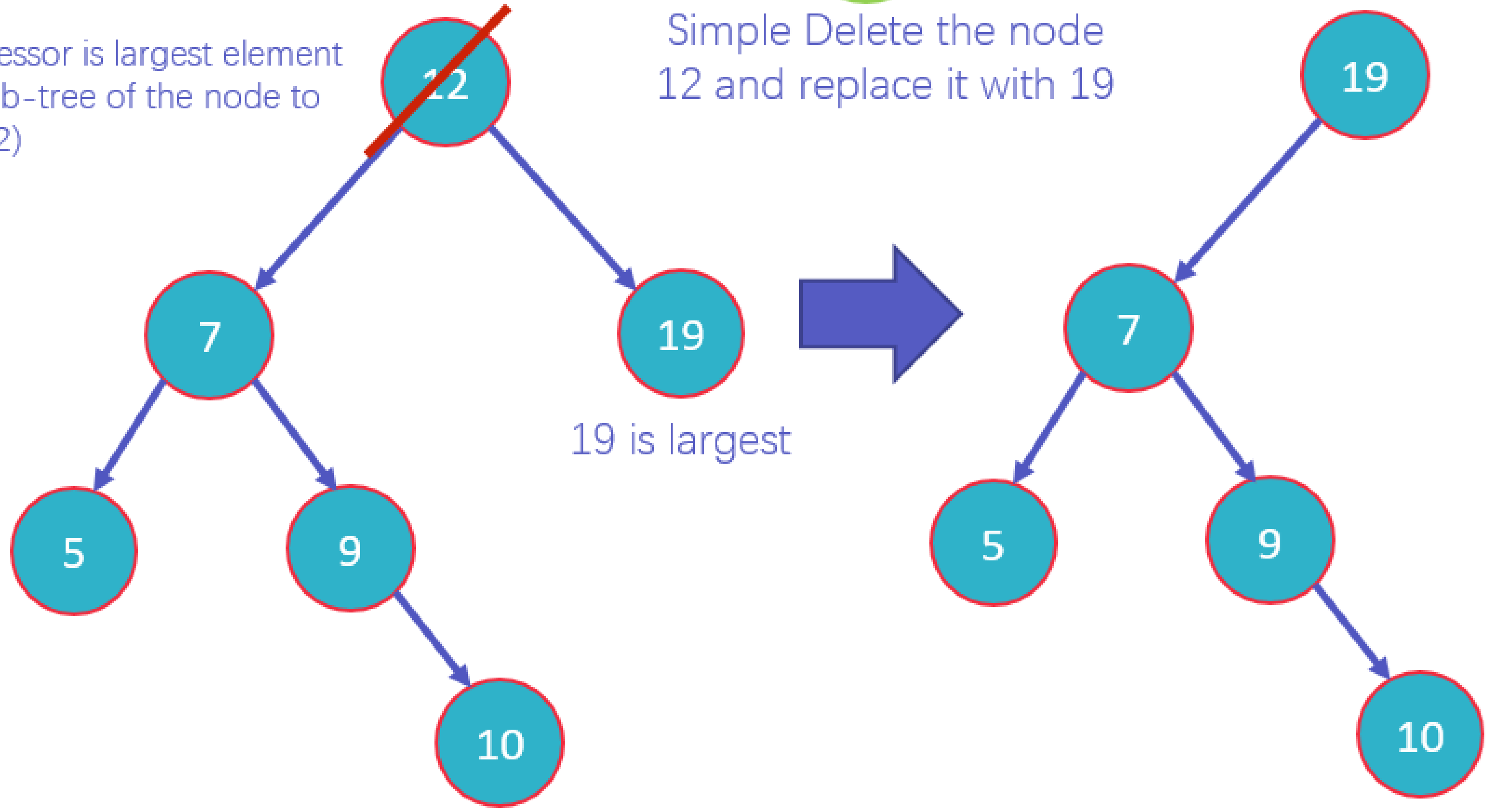
A Node to be deleted has 2 child
Replace Situation: In Order Successor

In Order successor is largest element in the right sub-tree of the node to be deleted (12)

B

C Simple Delete the node 12 and replace it with 19

D Result



Binary Tree Traversals

Displaying (or) visiting order of nodes in a binary tree is called as Binary Tree Traversal.

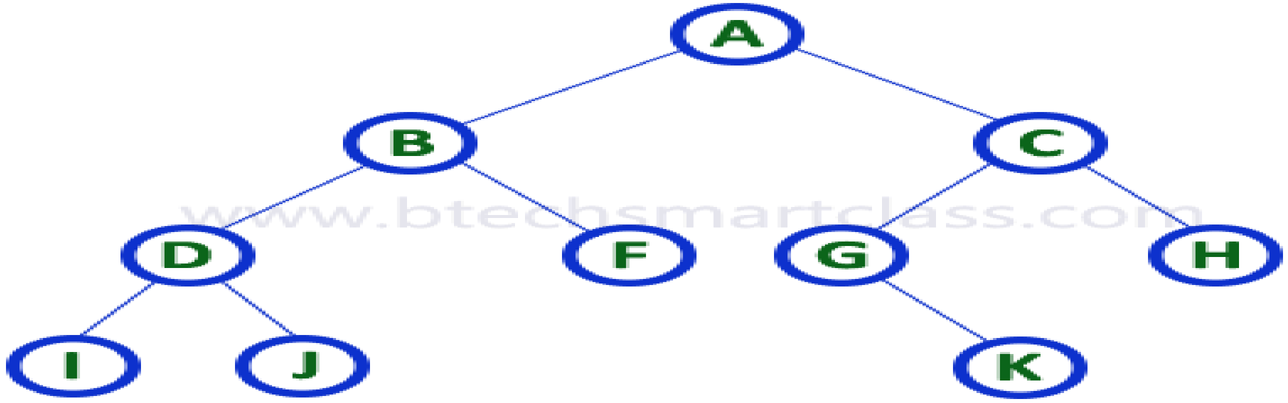
There are three types of binary tree traversals.

1.In - Order Traversal

2.Pre - Order Traversal

3.Post - Order Traversal

Consider the following binary tree...



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

In In-Order traversal, the root node is visited between the left child and right child.

In this traversal,

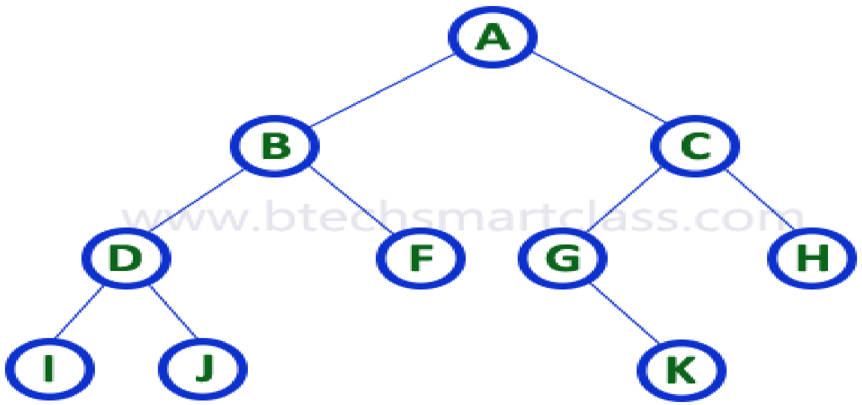
- ✓the left child node is visited first,
- ✓then the root node is visited
- ✓and later we go for visiting the right child node.

This in-order traversal is applicable for every root node of all subtrees in the tree. This is performed recursively for all nodes in the tree.

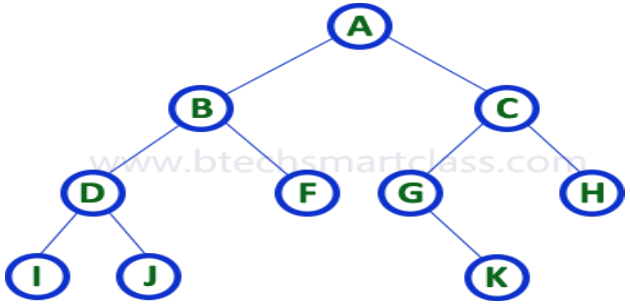
In-Order Traversal for above example of binary tree is

I - D - J - B - F - A - G - K - C – H

}



```
/* Given a binary tree, print its nodes in inorder*/
 void Inorder(struct node* node)
{
  if (node == NULL)  return;

  /* first recur on left child */
  Inorder(node->left);

  /* then print the data of node */
  printf("%d ", node->data);

  /* now recur on right child */
  Inorder(node->right);
  }
```



2.Pre - Order Traversal (root - leftChild - rightChild)

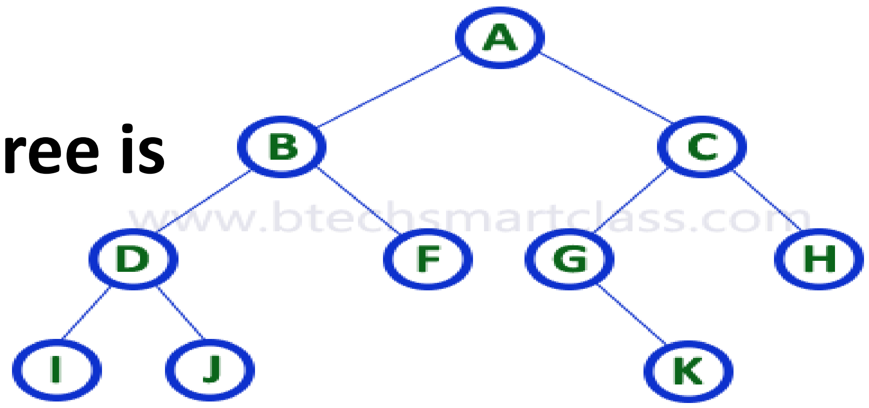
In Pre-Order traversal, the root node is visited before the left child and right child nodes.

In this traversal,

- ✓the root node is visited first,
- ✓then its left child
- ✓and later its right child.

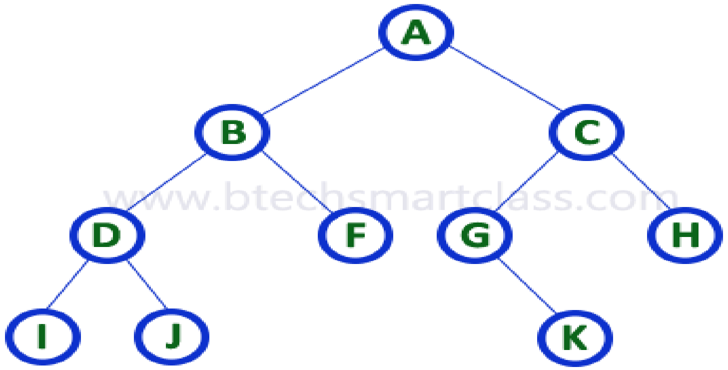
This pre-order traversal is applicable for every root node of all subtrees in the tree.

Pre-Order Traversal for above example binary tree is



A - B - D - I - J - F - C - G - K – H

```
/* Given a binary tree, print its nodes in
preorder*/

void Preorder(struct node* node)
{
  if (node == NULL)  return;

   /* first print data of node */

   printf("%d ", node->data);

   /* then recur on left sutree */

   Preorder(node->left);

   /* now recur on right subtree */

   Preorder(node->right);
}
```



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

In Post-Order traversal, the root node is visited after left child and right child.

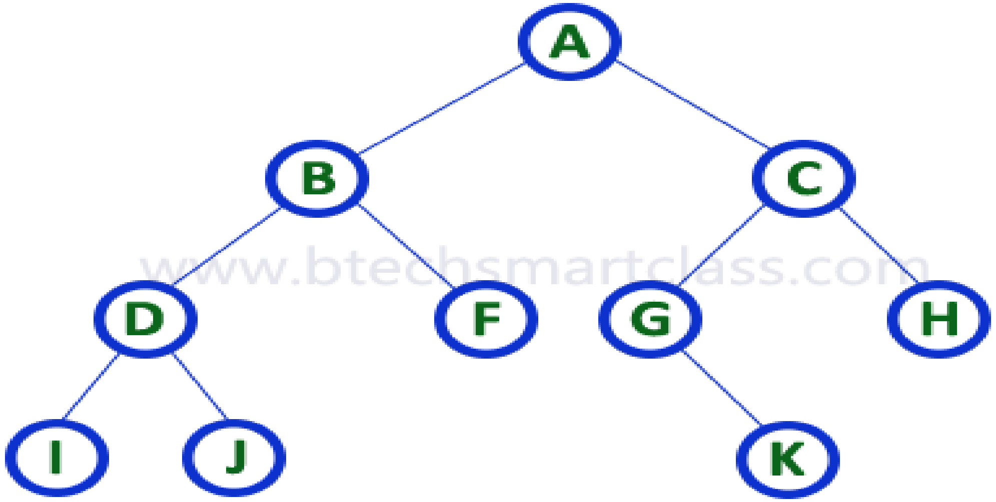
In this traversal,

- ✓left child node is visited first,
- ✓then its right child
- ✓and then its root node.

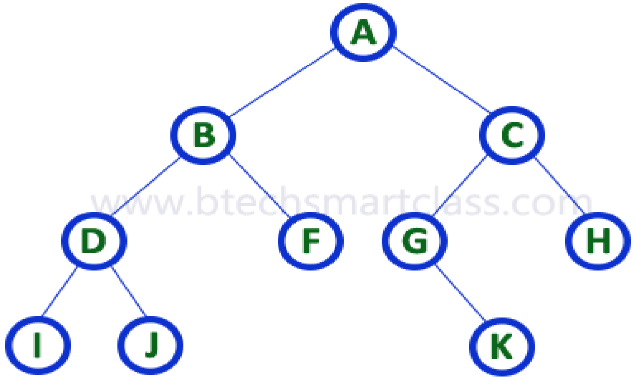
This is recursively performed until the right most node is visited.

Post-Order Traversal for above example binary tree is

I - J - D - F - B - K - G - H - C – A



```
void Postorder(struct node* node)
{
  if (node == NULL)
  return;

  // first recur on left subtree
   Postorder(node->left);

  // then recur on right subtree
    Postorder(node->right);

  // now deal with the node

  printf("%d ", node->data);
  }
```

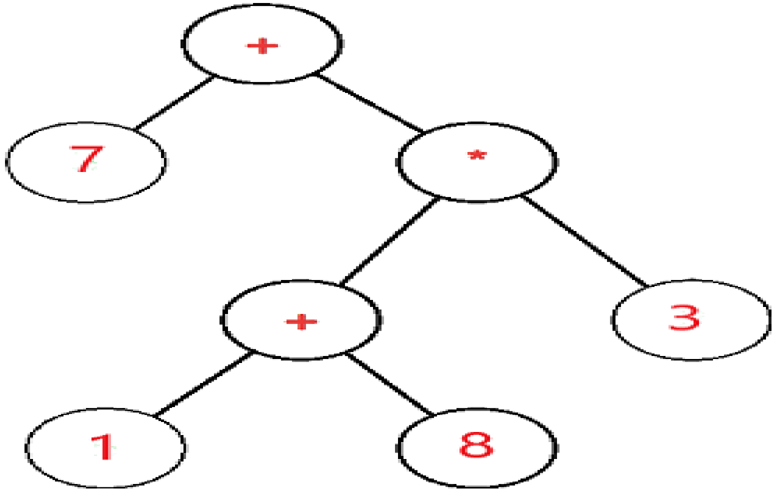


Expression Tree

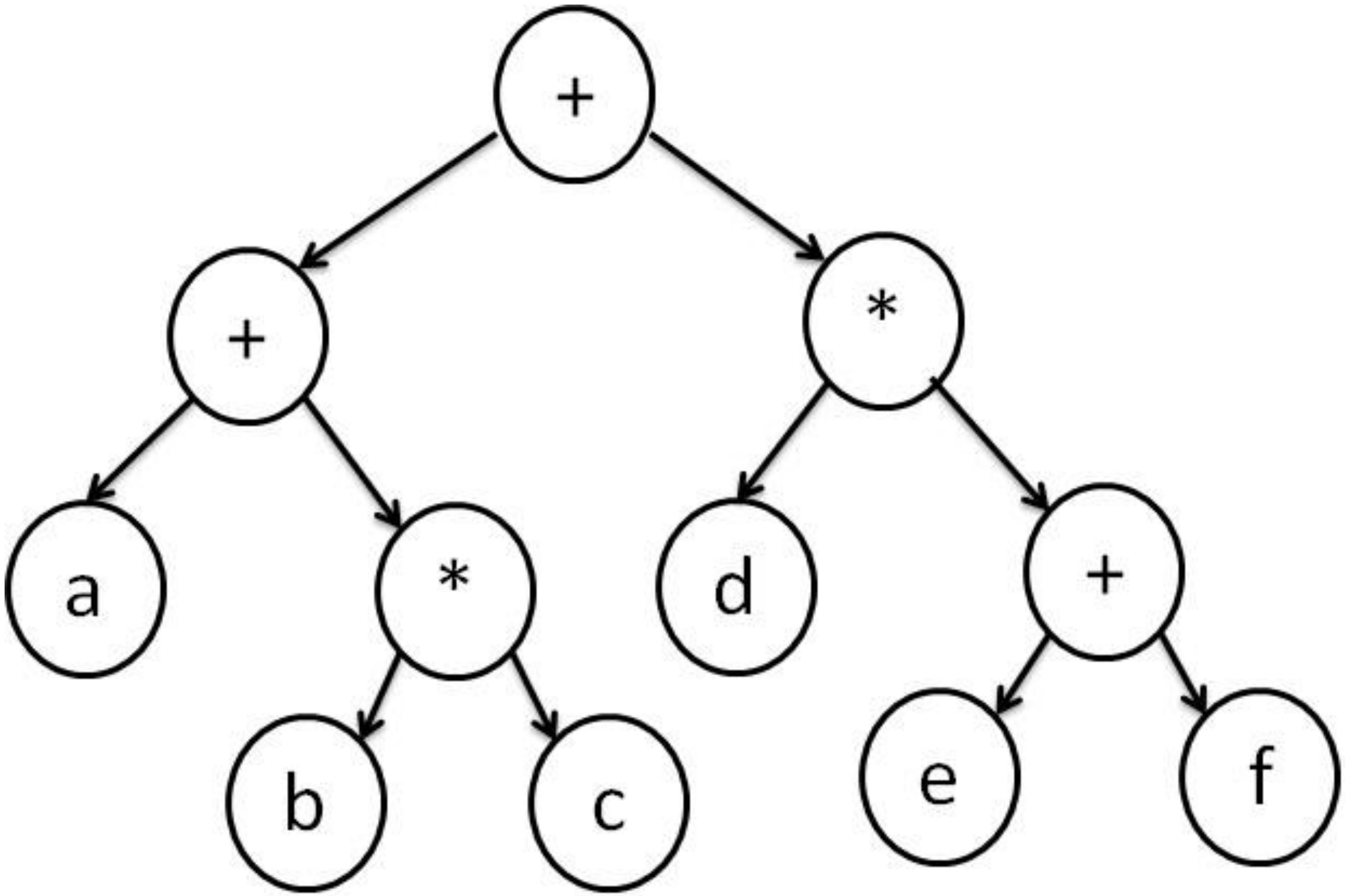
An expression tree is one form of binary tree that is used to represent the expressions.

A binary expression tree can represent two types of expressions i.e., algebraic expressions and Boolean expressions.

- Expression Tree is used to represent expressions.



a + (b * c) + d * (e + f)



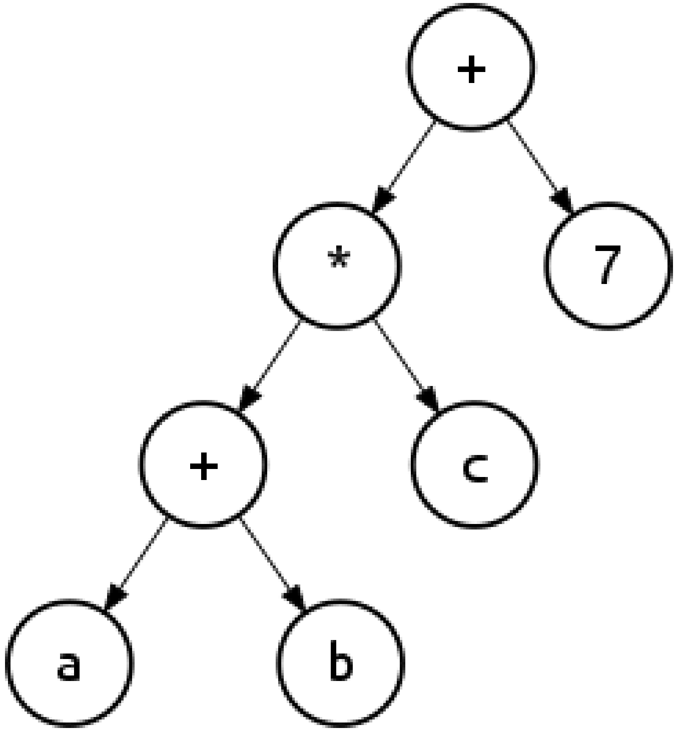
In-order Traversal:
(a+(b*c))+(d*(e + f))

Post-order Traversal:
a b c * + d e f + * +

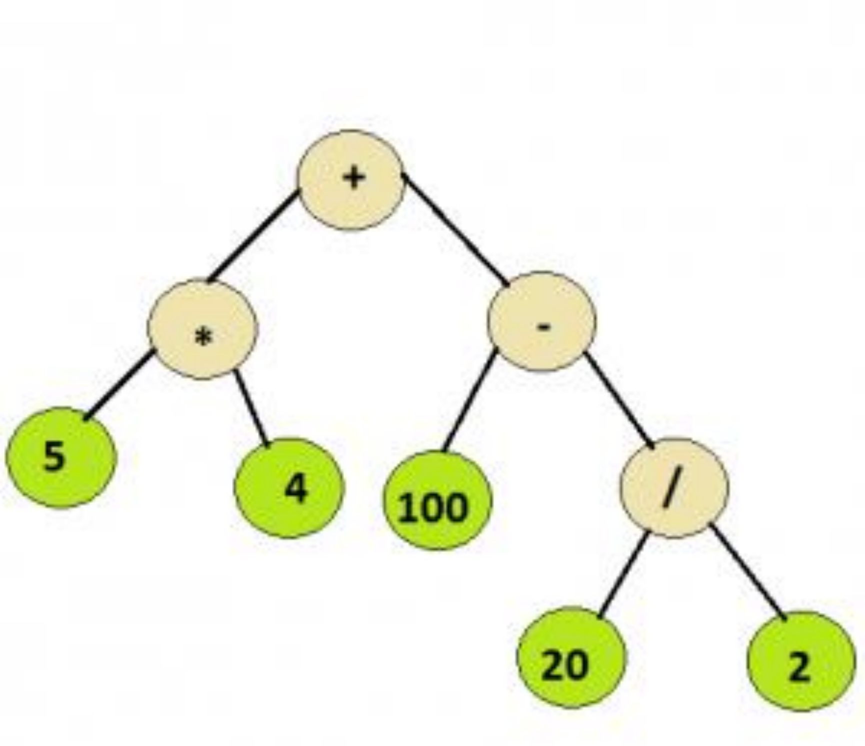
Pre-order Traversal:
+ + a * b c * d + e f

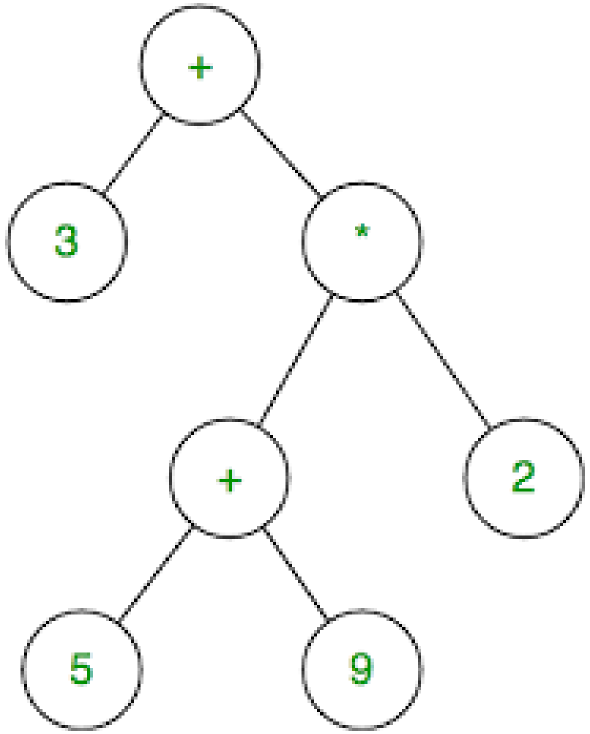
Problems on expression tree

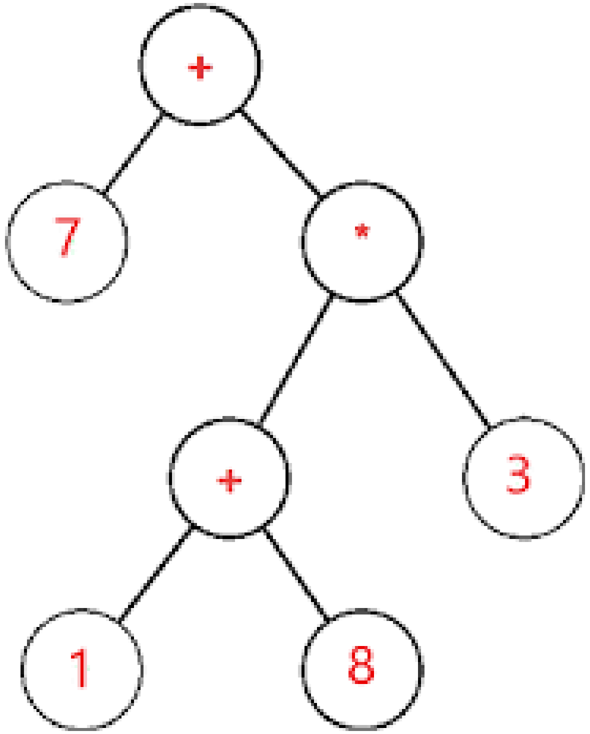
Traverse given expression tree in Inorder, preorder and postorder.

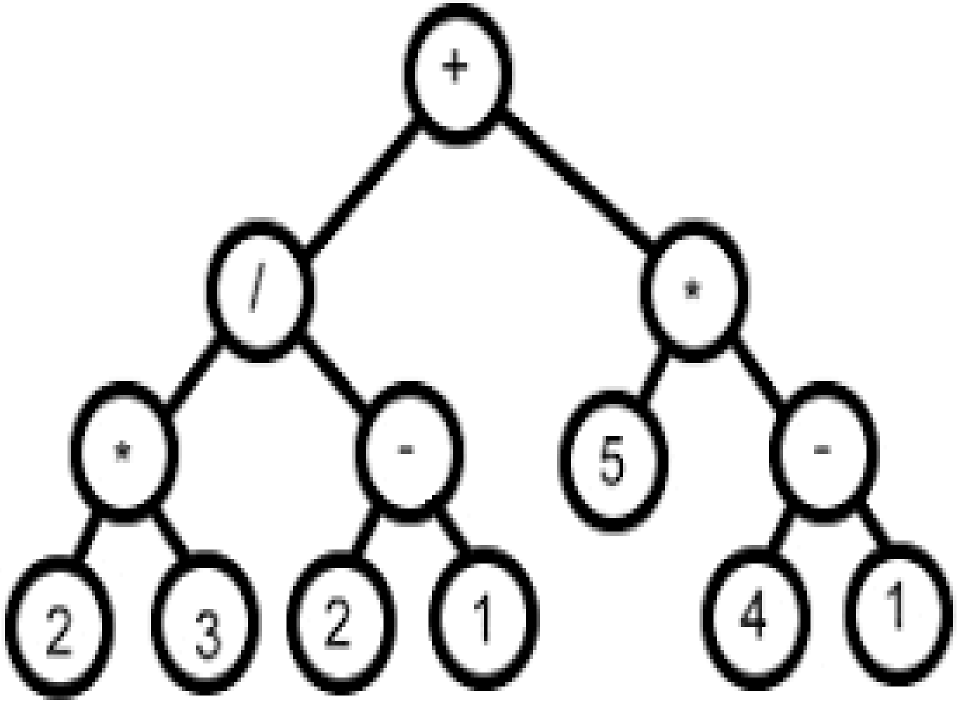


Evaluate given expression tree.









Expression tree for 2*3/(2-1)+5*(4-1)