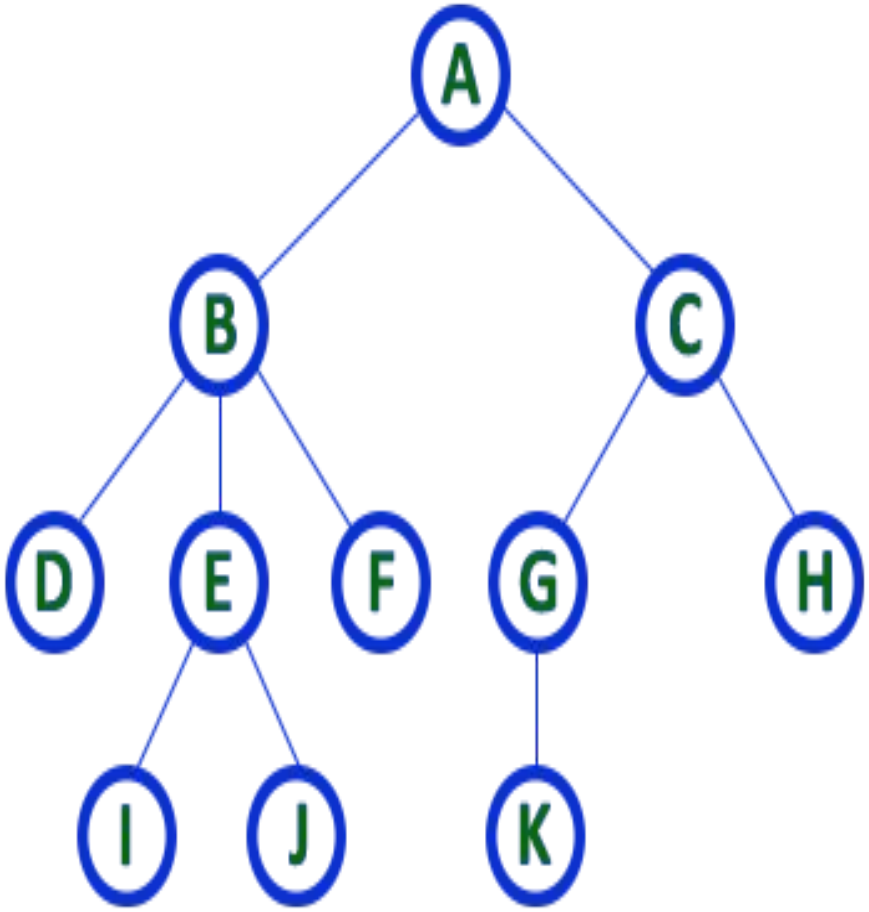
Tree Data Structure

A tree is a nonlinear hierarchical data structure that consists of nodes (data) connected by edges.

Other data structures such as arrays, linked list, stack, and queue are linear data structures that store data sequentially. In order to perform any operation in a linear data structure, the time complexity increases with the increase in the data size.

Example 1.



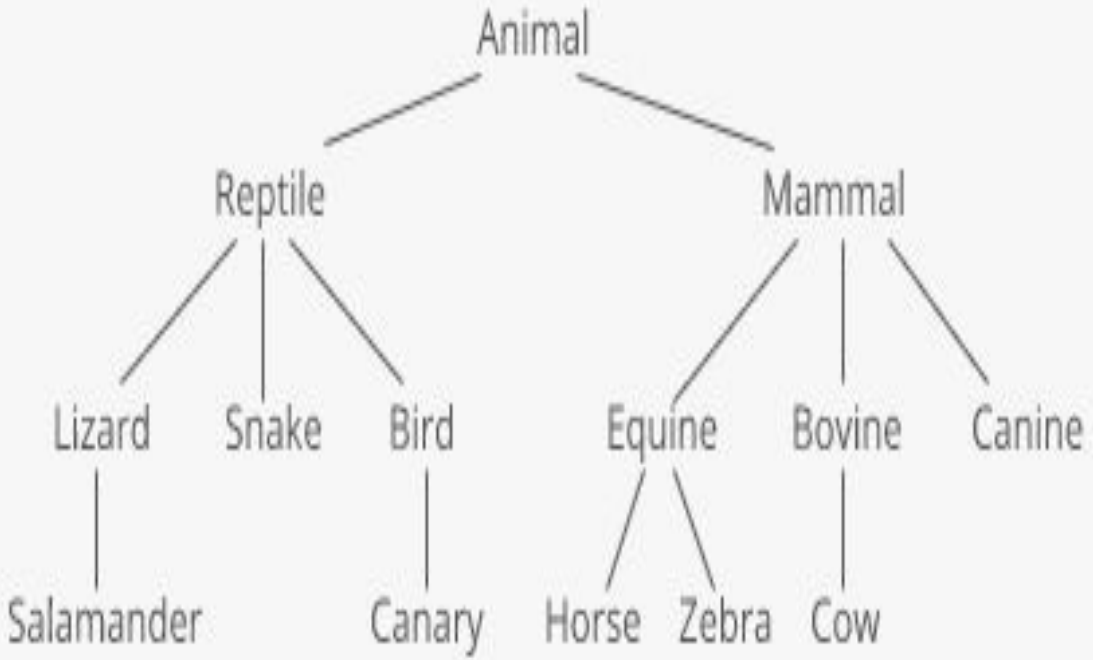
TREE with 11 nodes and 10 edges

- In any tree with 'N' nodes there will be maximum of 'N-1' edges

- In a tree every individual element is called as 'NODE'

Example 2

A **tree** organizes values hierarchically.

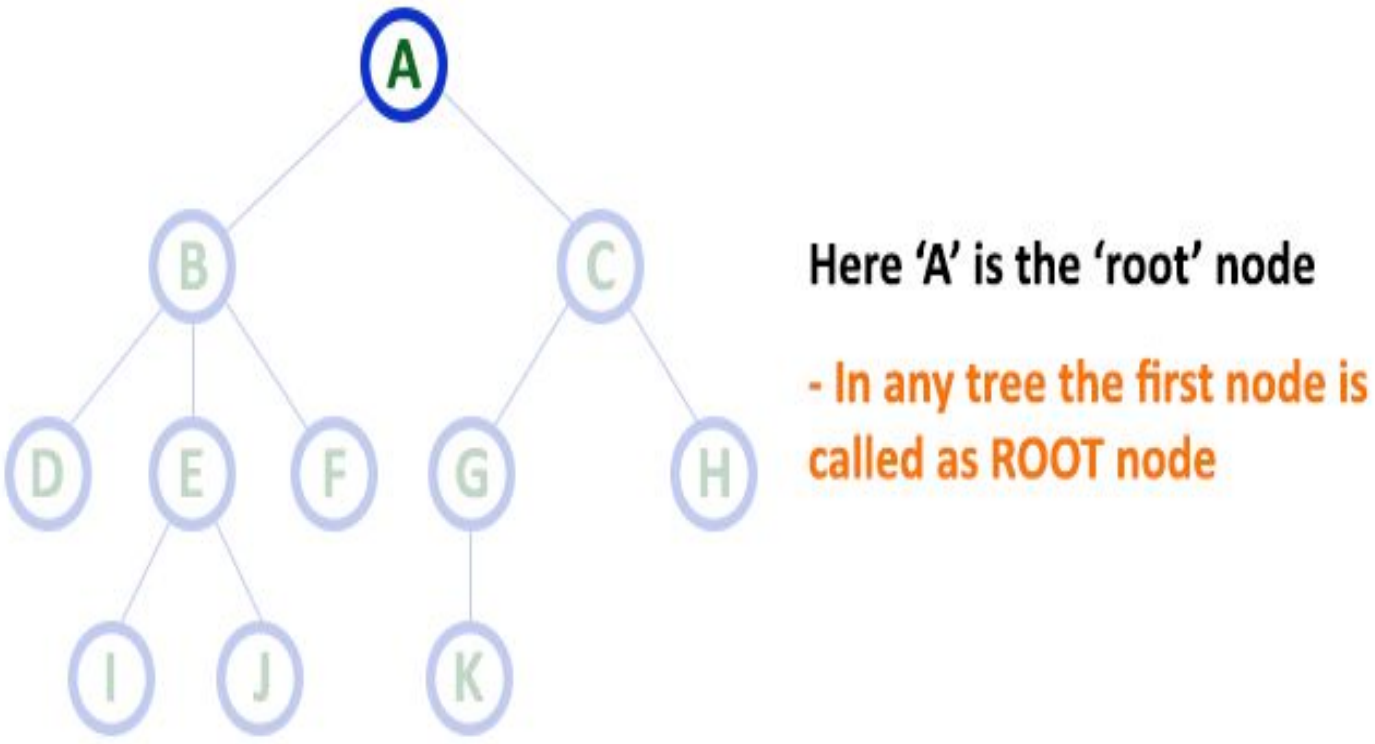


Terminology

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

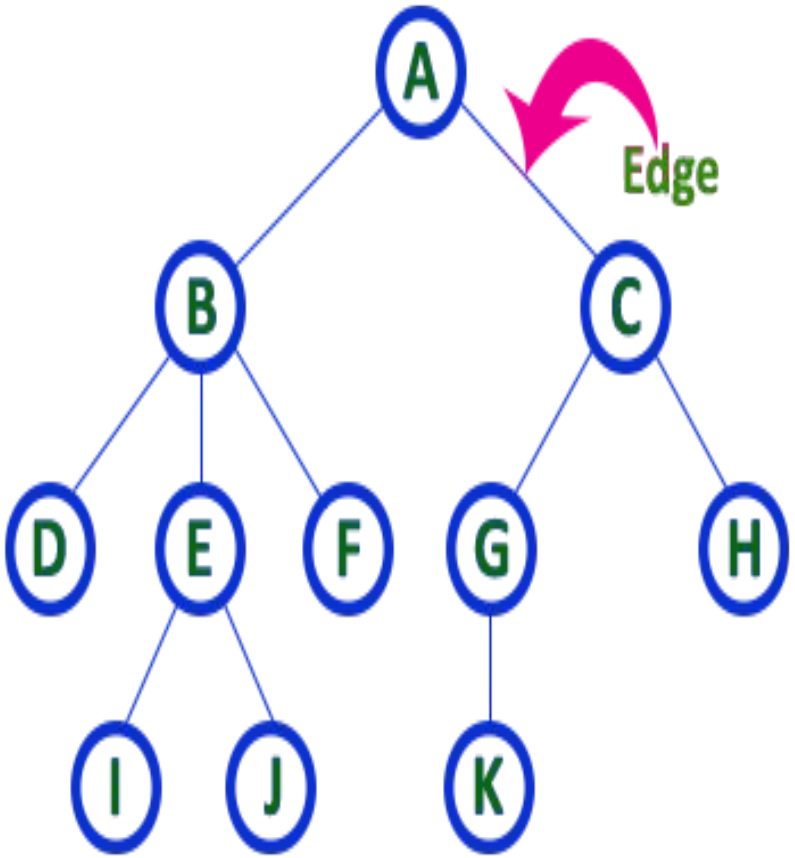
1. Root

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



2. Edge

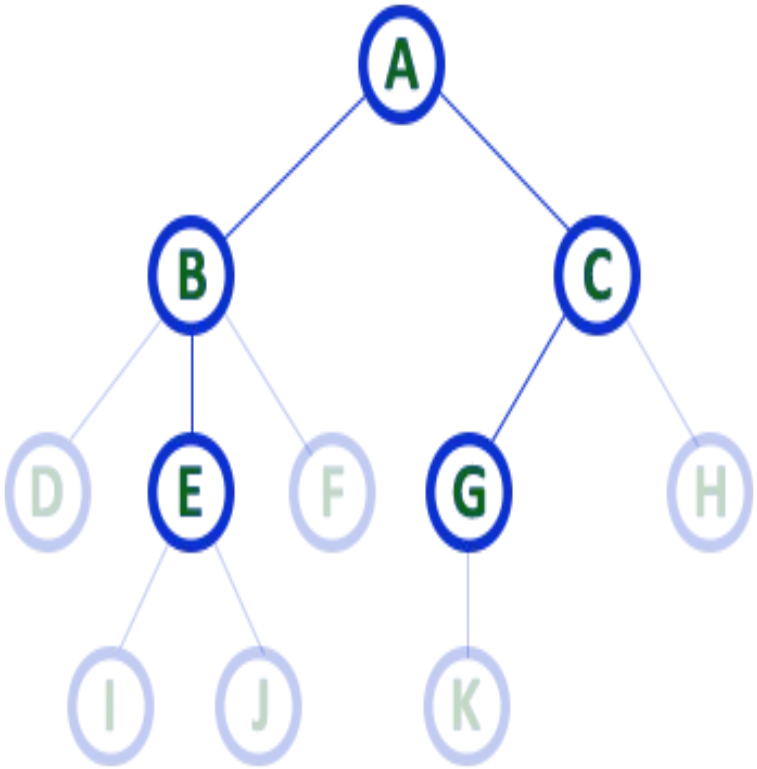
In a tree data structure, the connecting link between any two nodes is called as **EDGE**. In a tree with '**N**' number of nodes there will be a maximum of '**N-1**' number of edges.



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

3. Parent

In a tree data structure, the node which is a predecessor of any node is called as **PARENT NODE**. In simple words, the node which has a branch from it to any other node is called a parent node. Parent node can also be defined as "**The node which has child / children**".

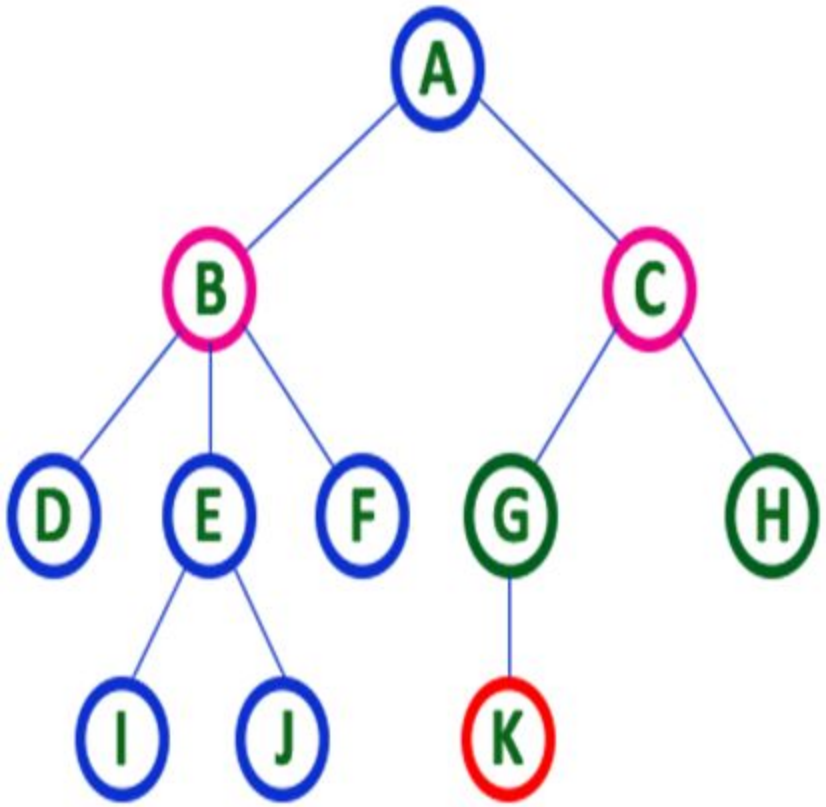


Here A, B, C, E & G are Parent nodes

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

4. Child

In a tree data structure, All immediate successors of a node are its children. 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.



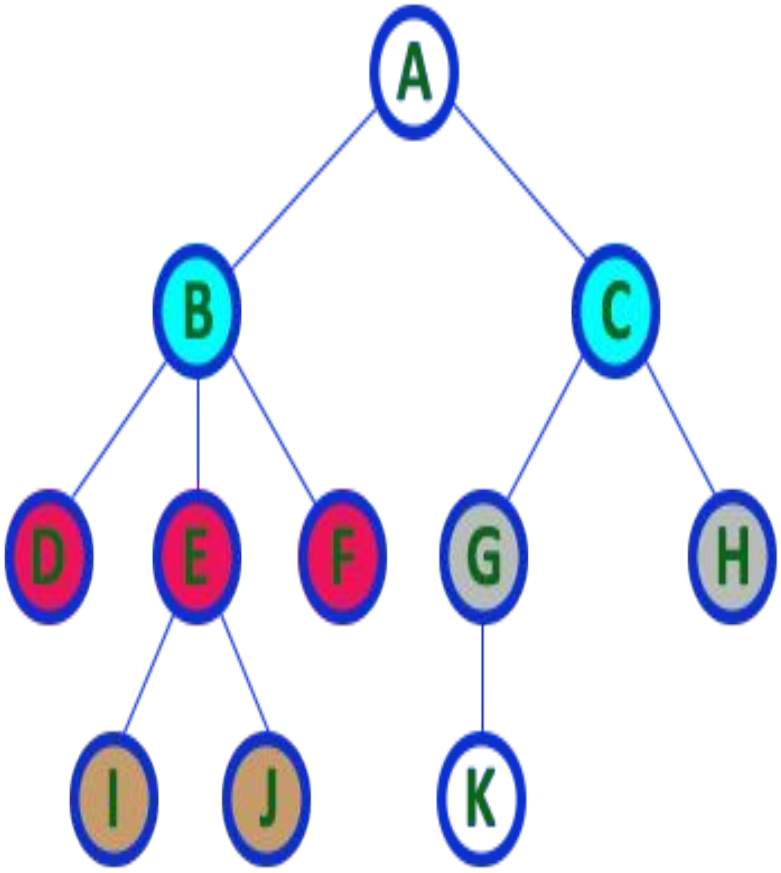
Here B & C are Children of A

Here G & H are Children of C

Here K is Child of G

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.



Here B & C are Siblings

Here D E & F are Siblings

Here G & H are Siblings

Here I & J are Siblings

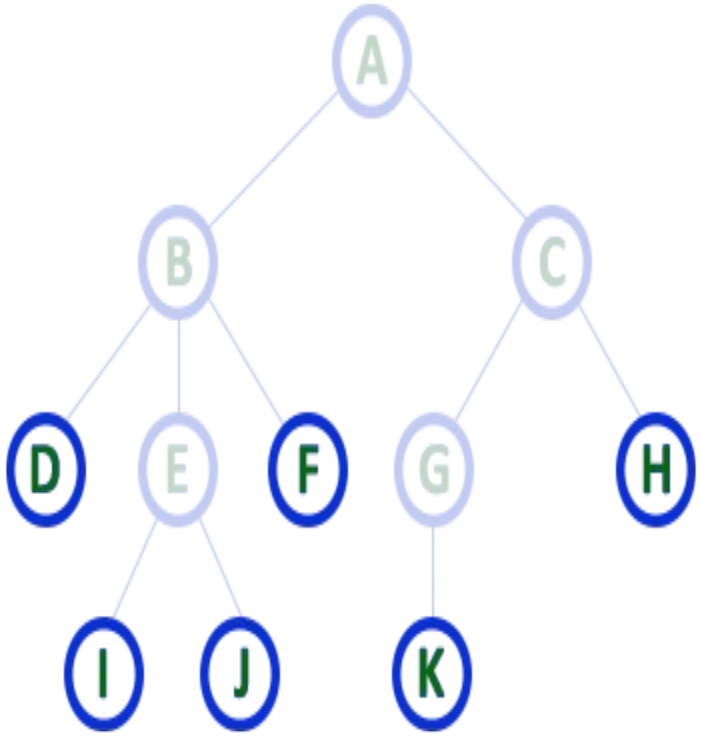
- In any tree the nodes which has same Parent are called 'Siblings'

- The children of a Parent are called 'Siblings'

6. 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.



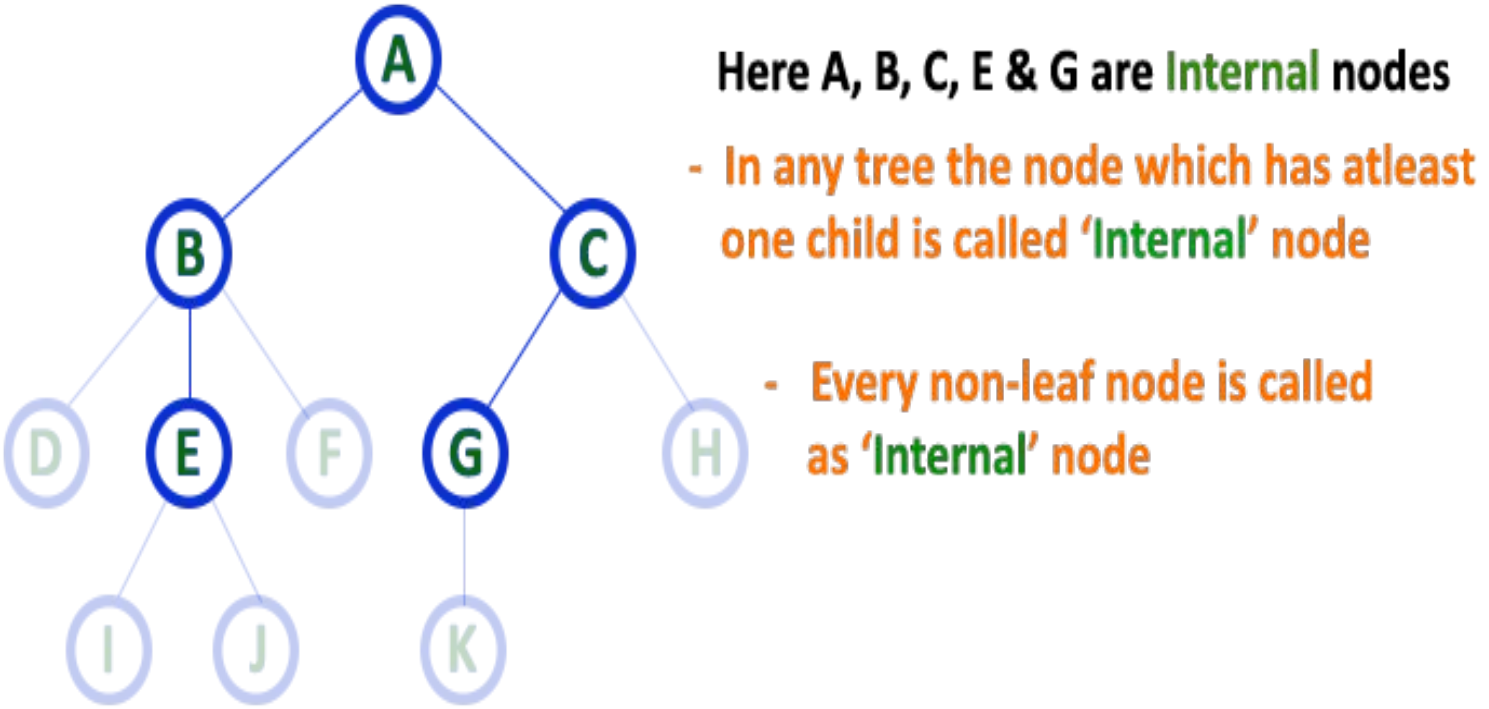
Here D, I, J, F, K & H are Leaf nodes

- 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

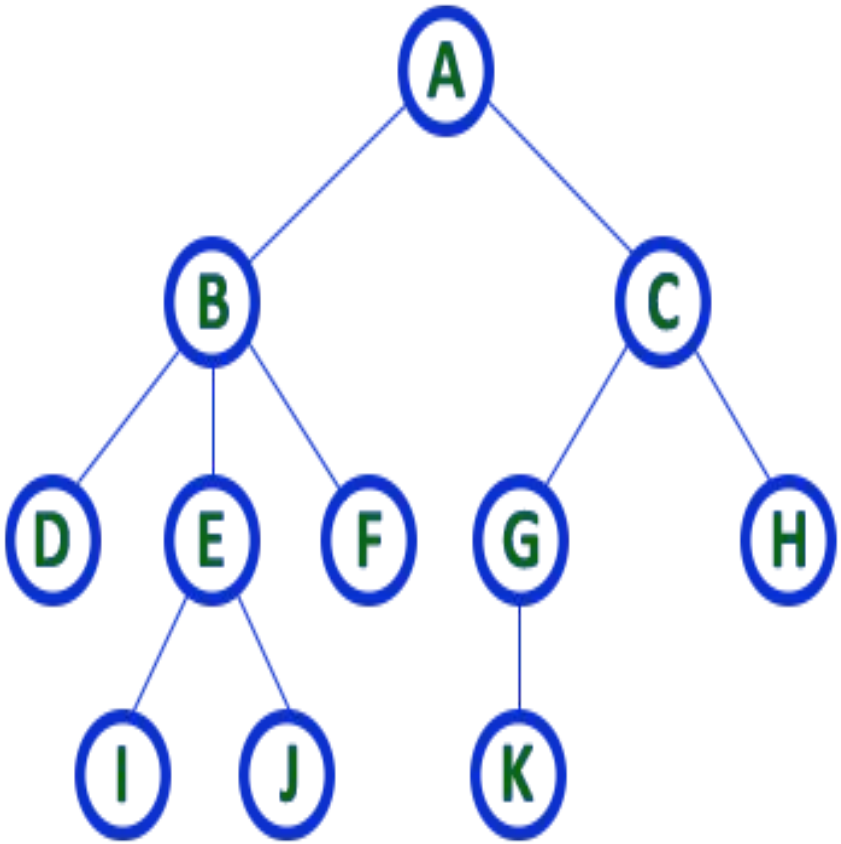
In a tree data structure, the node which has atleast one child is called as **INTERNAL Node**. In simple words, an internal node is a node with atleast one child.

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.



8. 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**'



Here Degree of B is 3

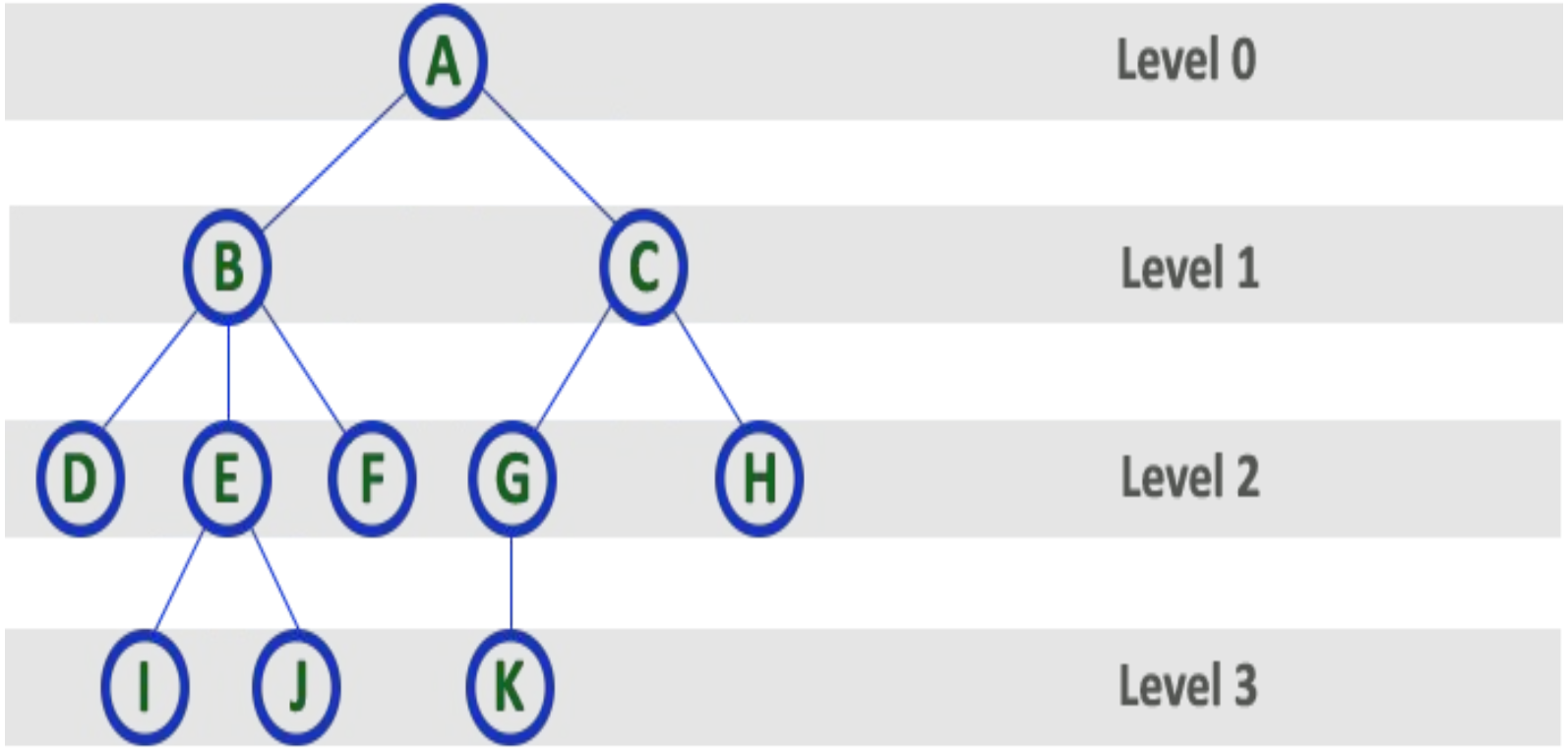
Here Degree of A is 2

Here Degree of F is 0

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

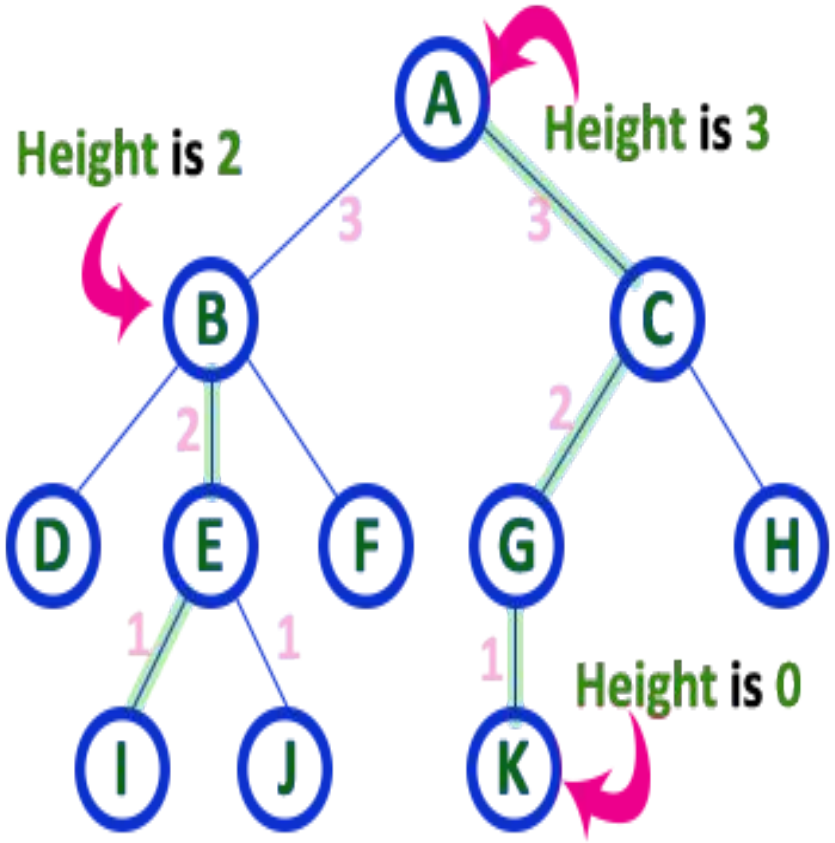
9. Level

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... 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).



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. 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'.**



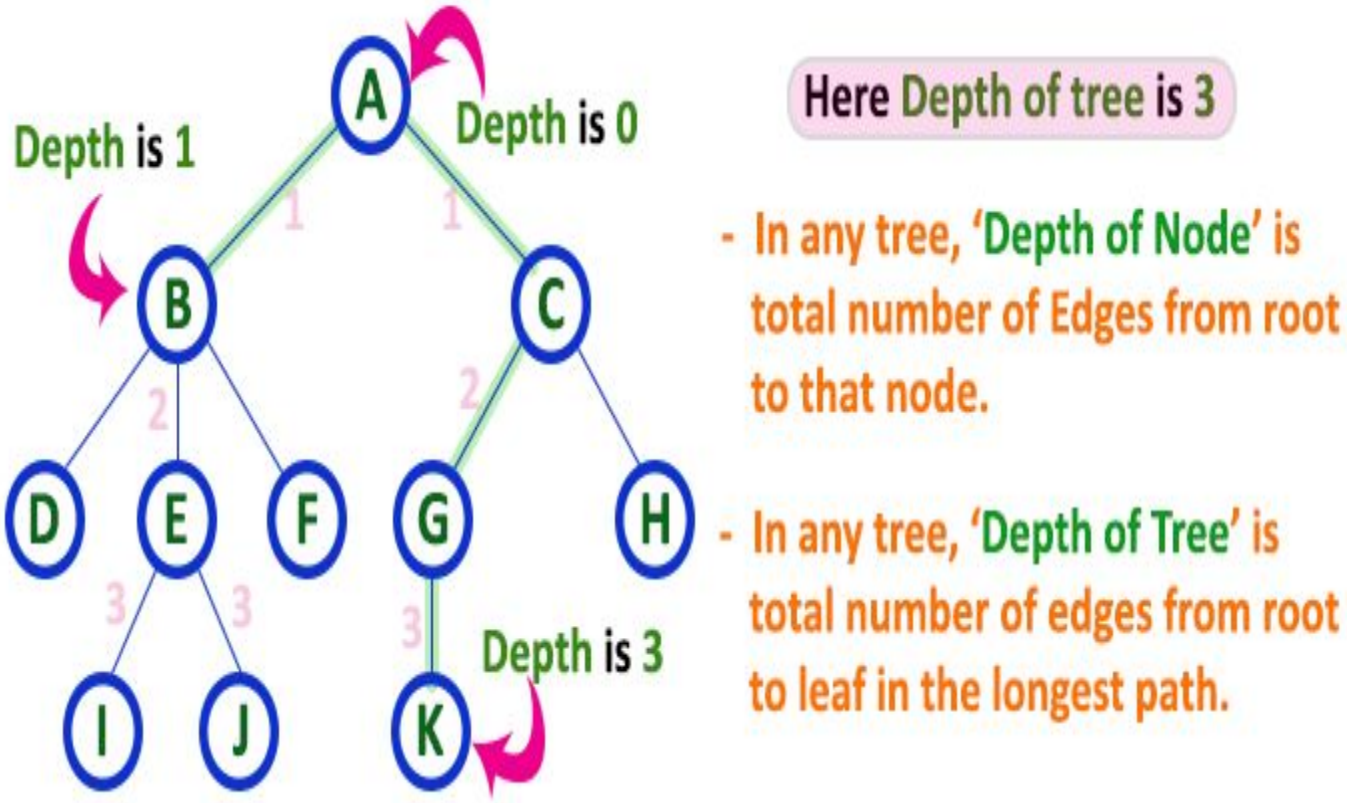
Here Height of tree is 3

- 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.

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. 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'.**



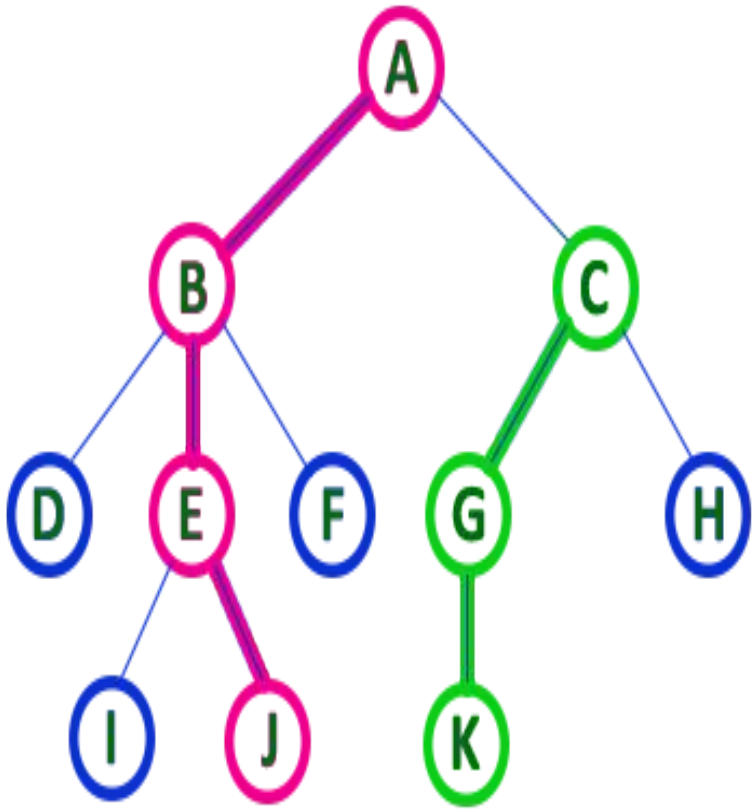
Difference:

Height of a node is the number of edges on the *longest path* from the node to a leaf.

Depth of a node is the number of edges from the node to the tree's root 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. **Length of a Path** is total number of nodes in that path. In below example **the path A - B - E - J has length 4**.



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

Here, 'Path' between A & J is

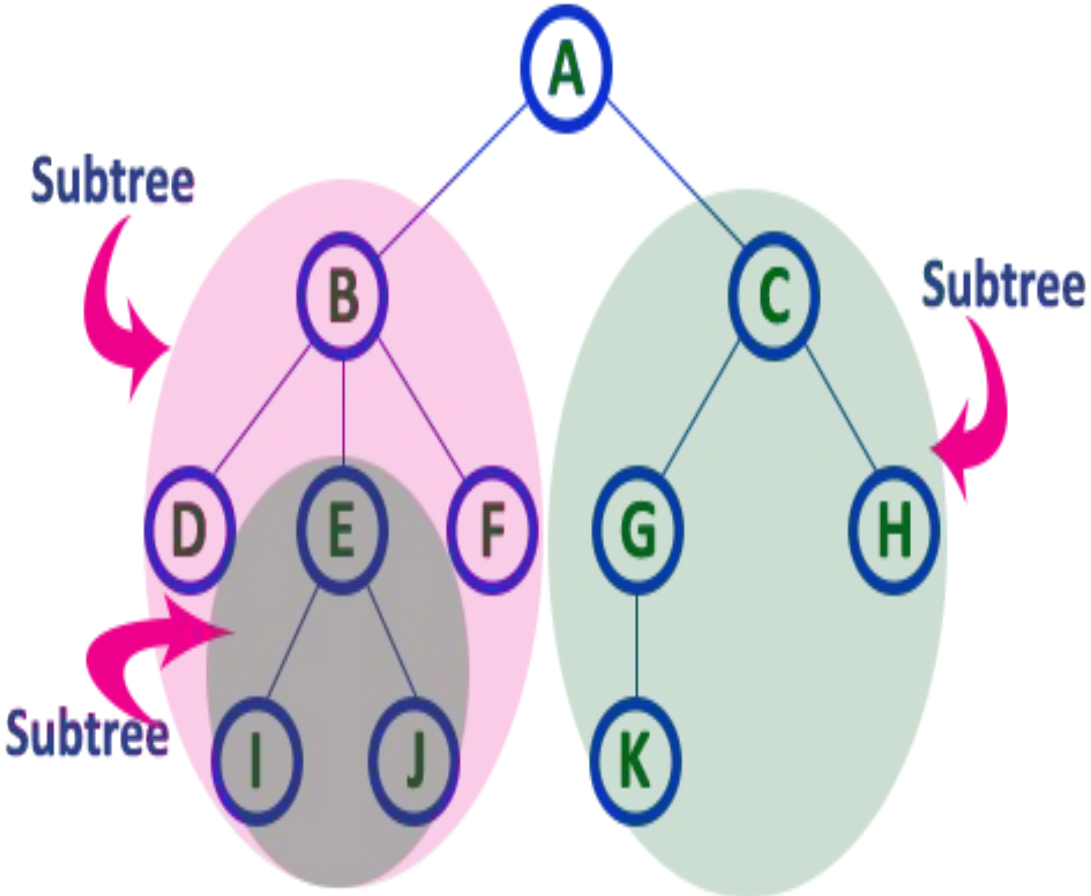
A - B - E - J

Here, 'Path' between C & K is

C - G - K

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.



Binary Tree

A binary tree is a tree where each node has exactly zero, one or two children.

(OR)

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.

(OR)

Binary tree is a tree in which degree of a node is less than or equal to 2.

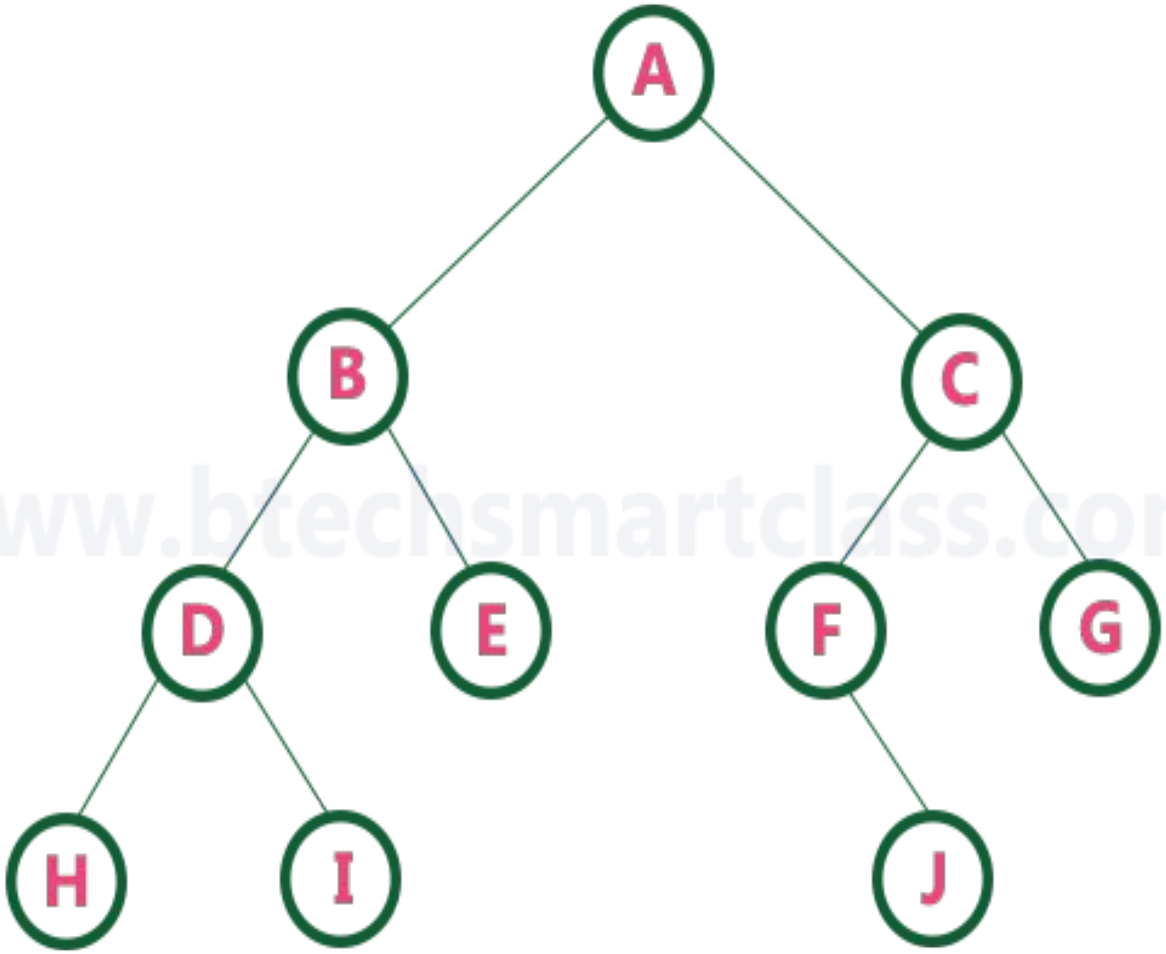
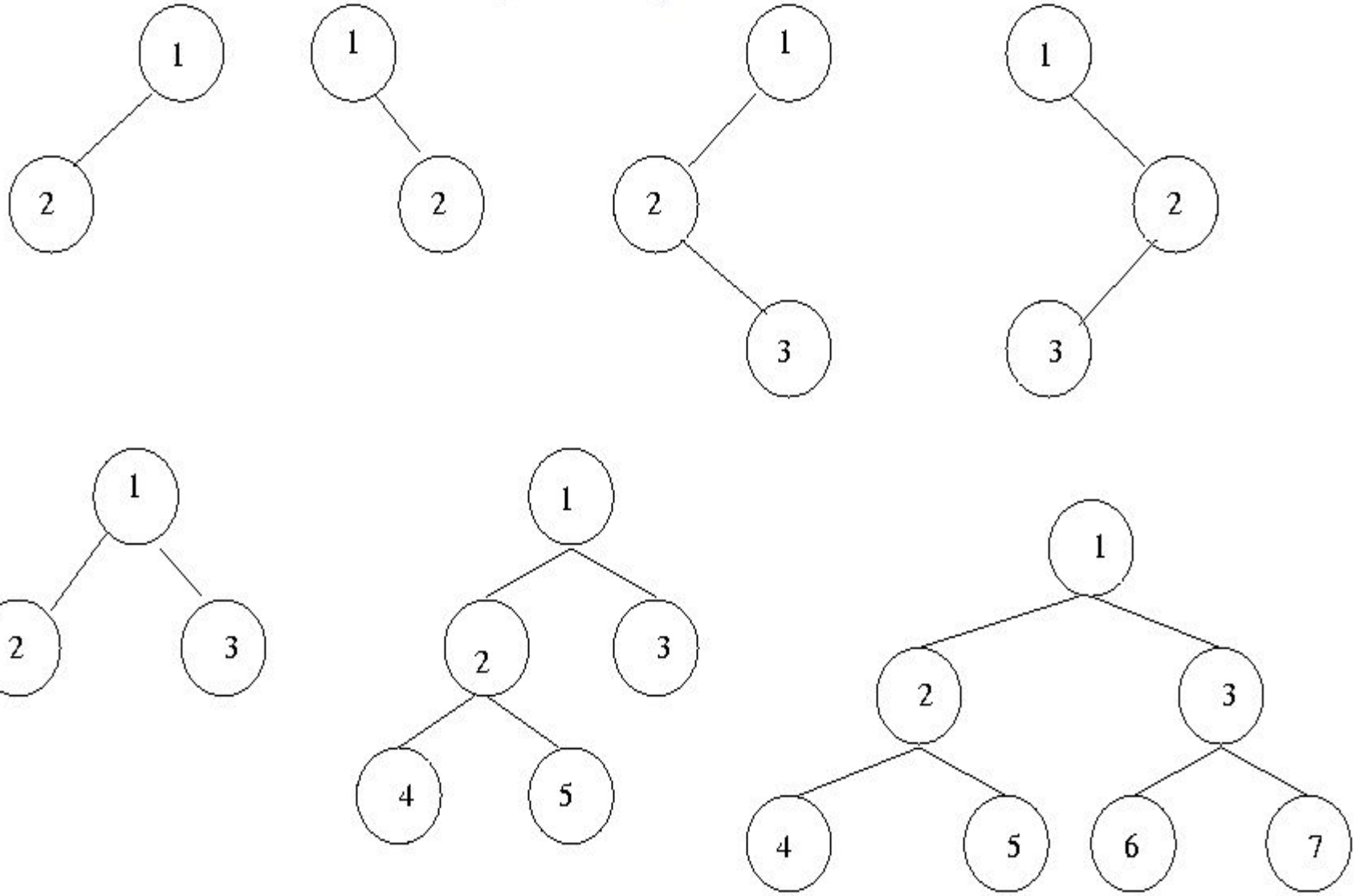
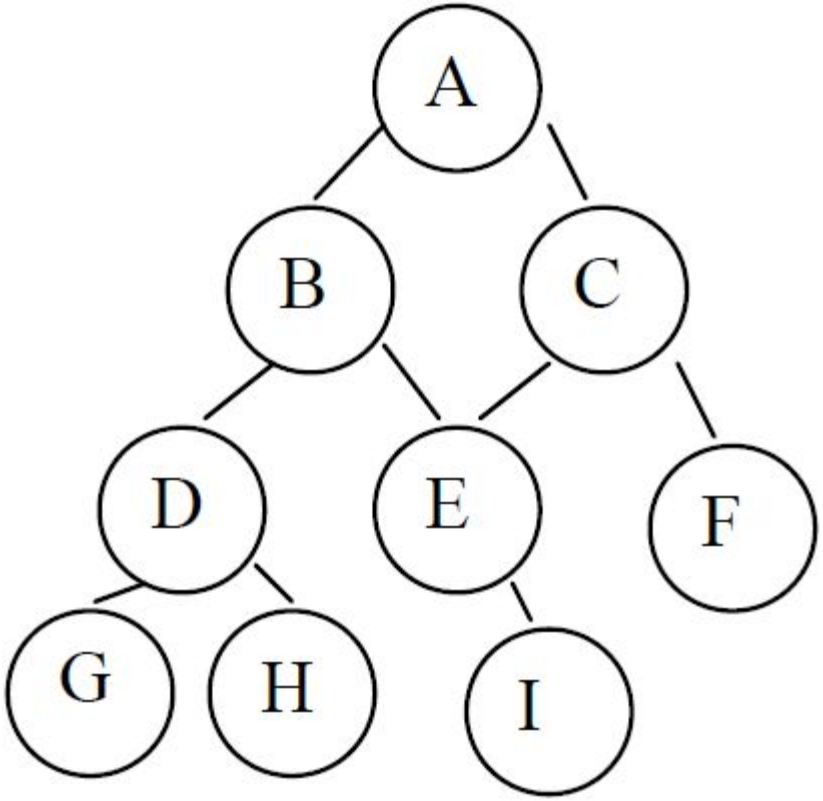


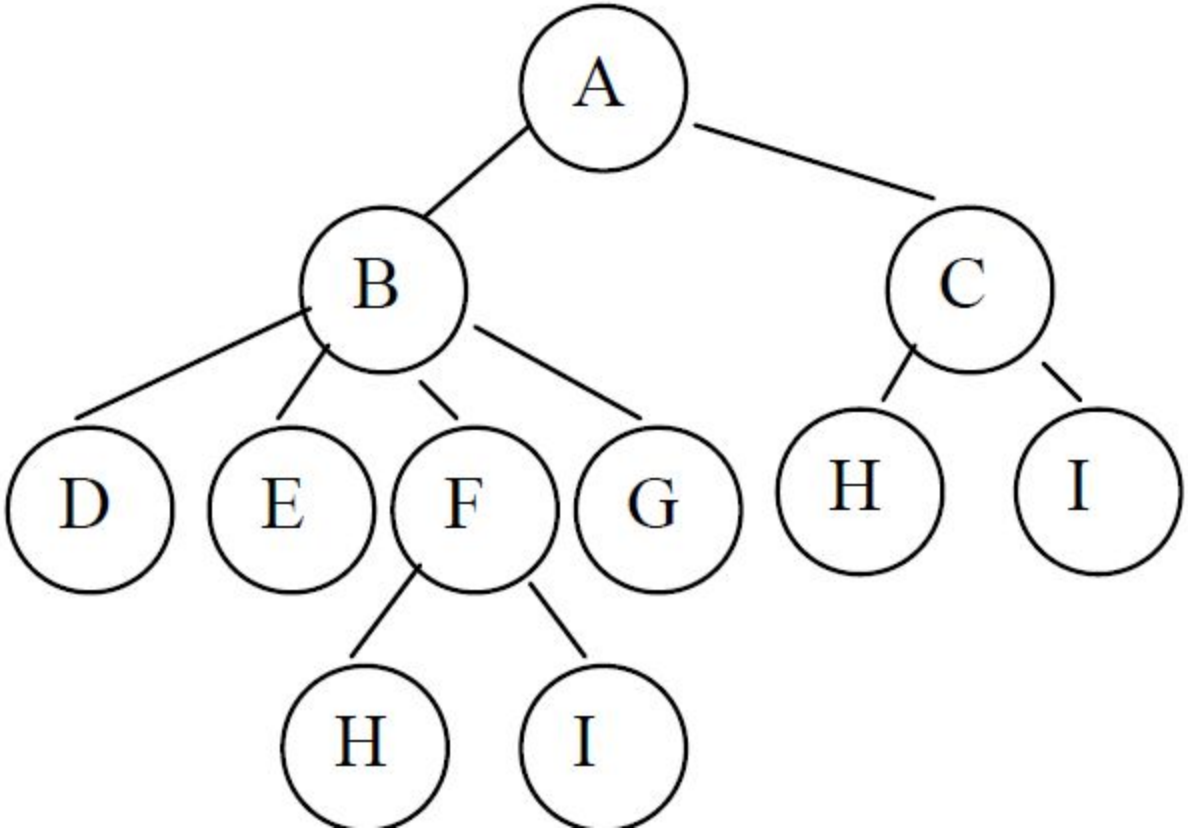
Figure 4.4: Examples of binary trees



Is it a binary tree or not, why?



Is it a binary tree or not, why?



Tree Traversal

- Traversing a tree means visiting each node in a specified order.

There are generally two types of traversal: –
1)breadth first
2) depth first.

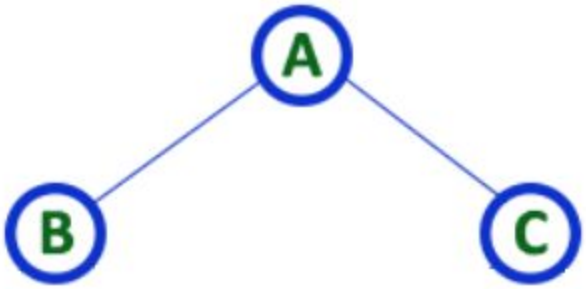
There are three variants for depth first traverse a tree. They're called

1) Inorder
2) Preorder
3)postorder.

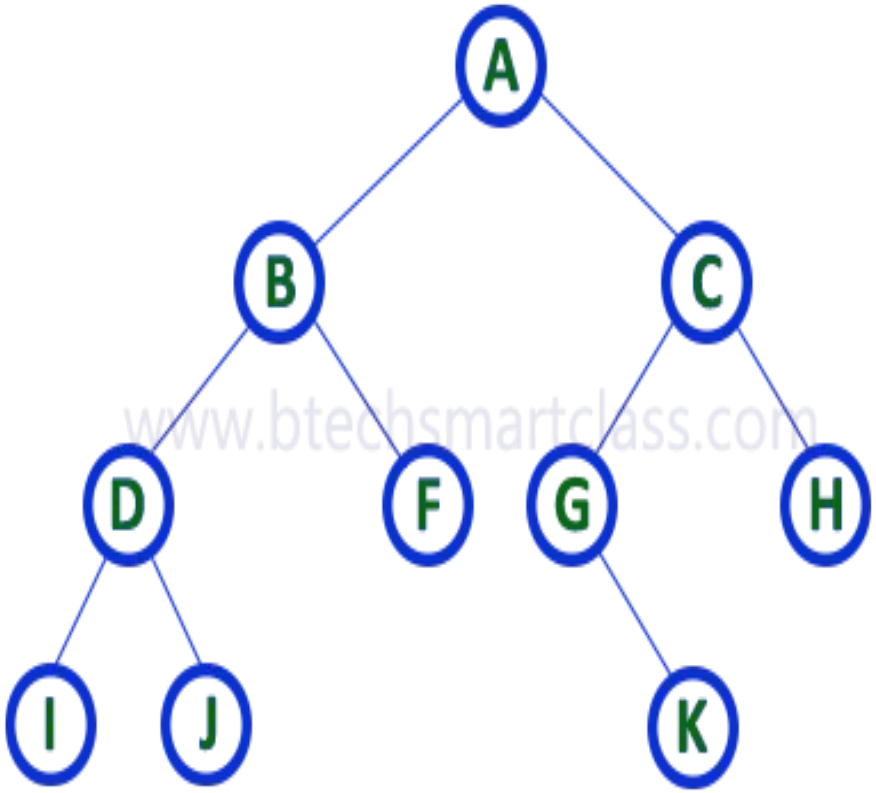
1.) In - Order Traversal

To traverse a binary tree in inorder traversal, following operations are carried out:

Traverse the left most sub tree.
Visit the root.
Traverse the right most sub tree.

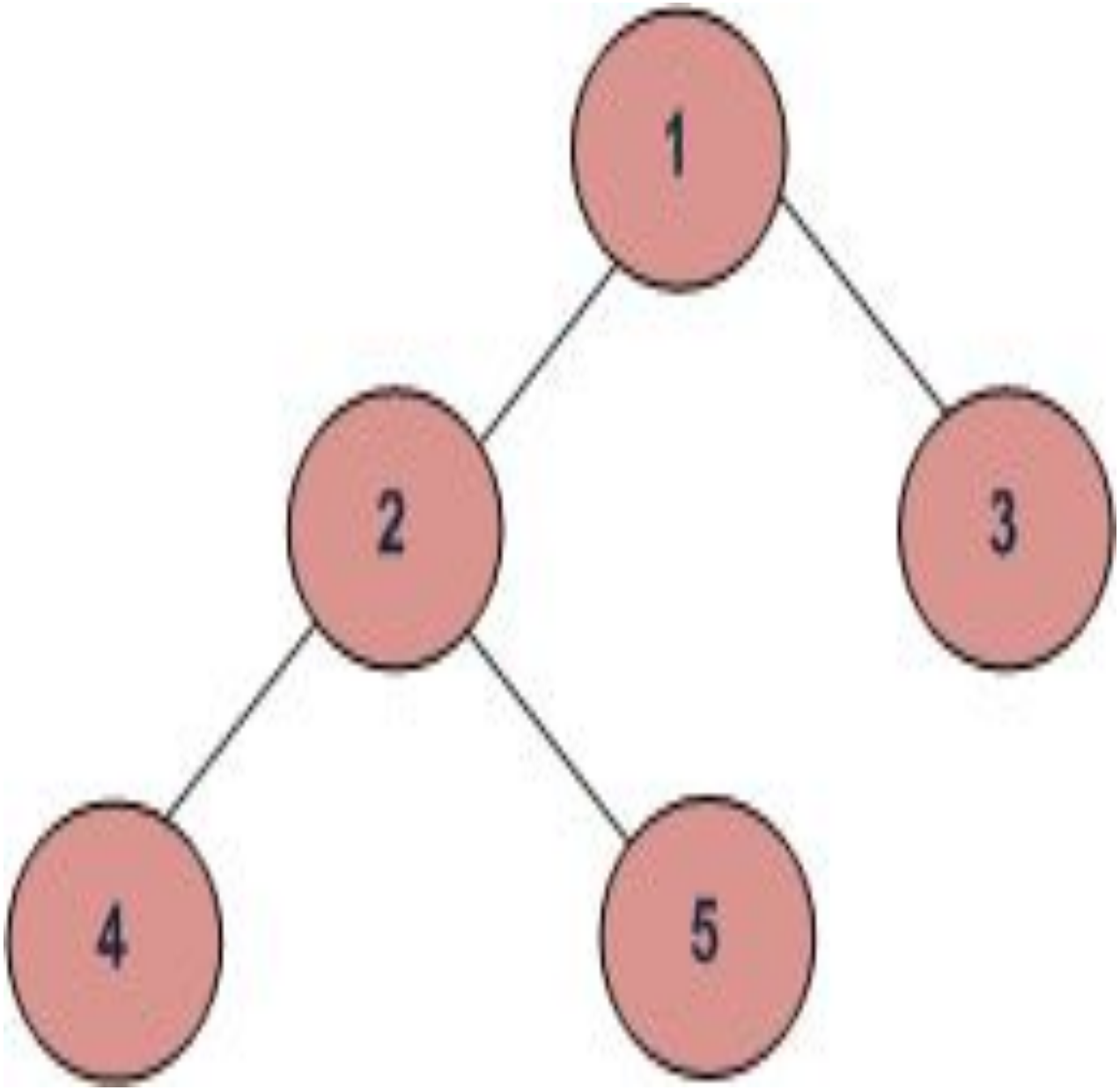


Inorder traversal--------> B,A,C

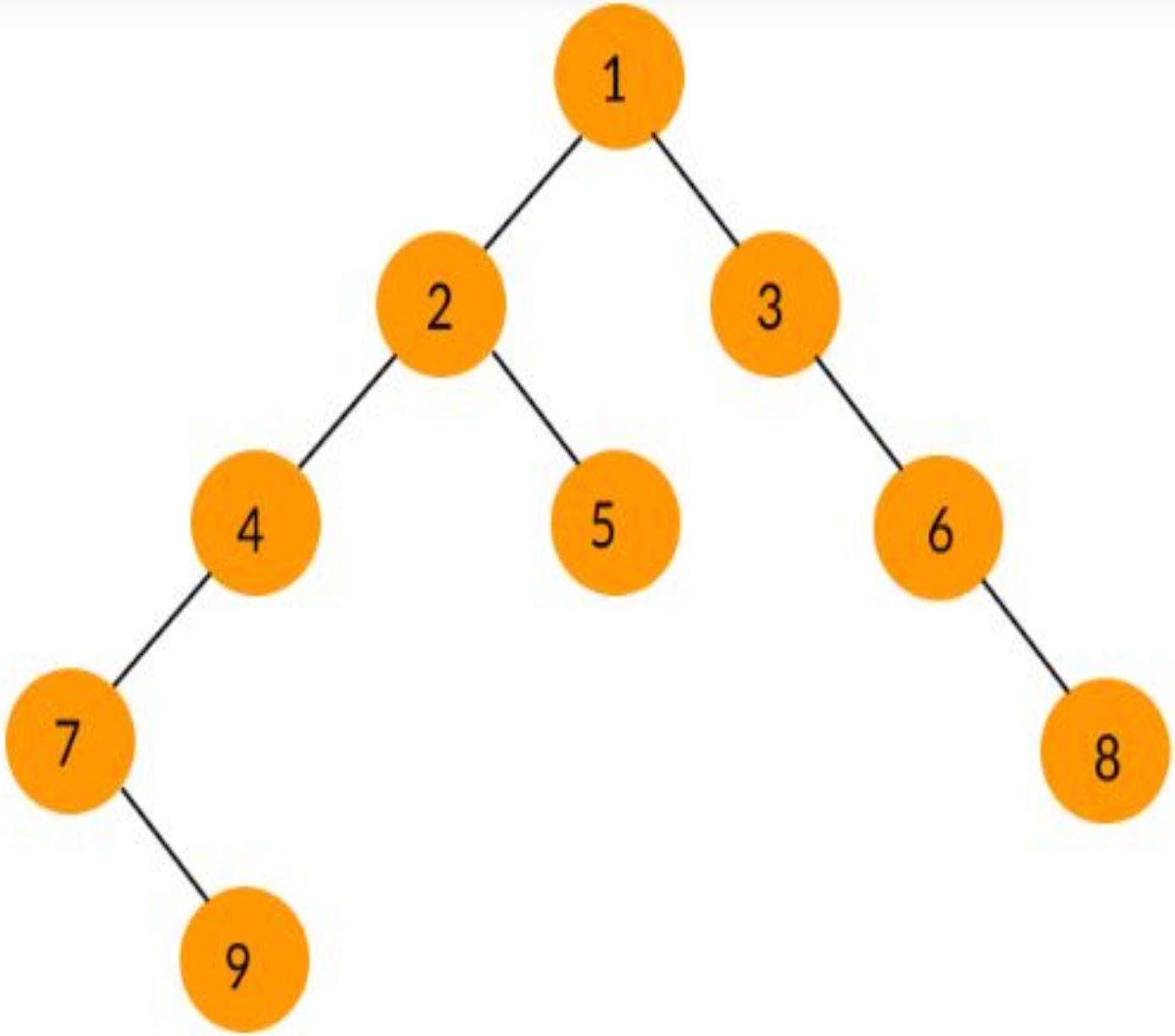


In-Order Traversal for above example of binary tree is

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

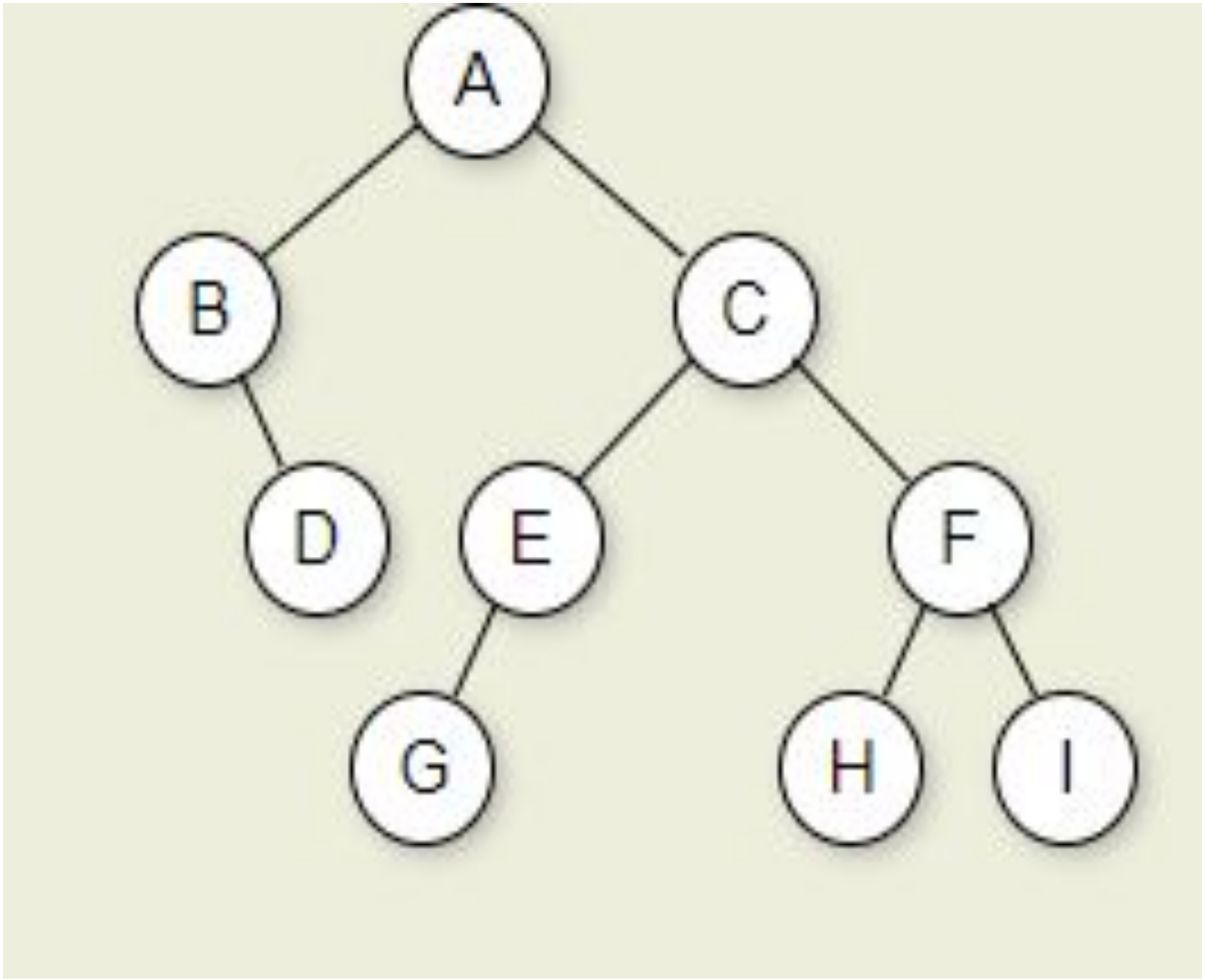

1
2
3
4
5

Inorder: 4 2 5 1 3


1
2
3
4
5
6
7
8
9

Inorder Traversal:

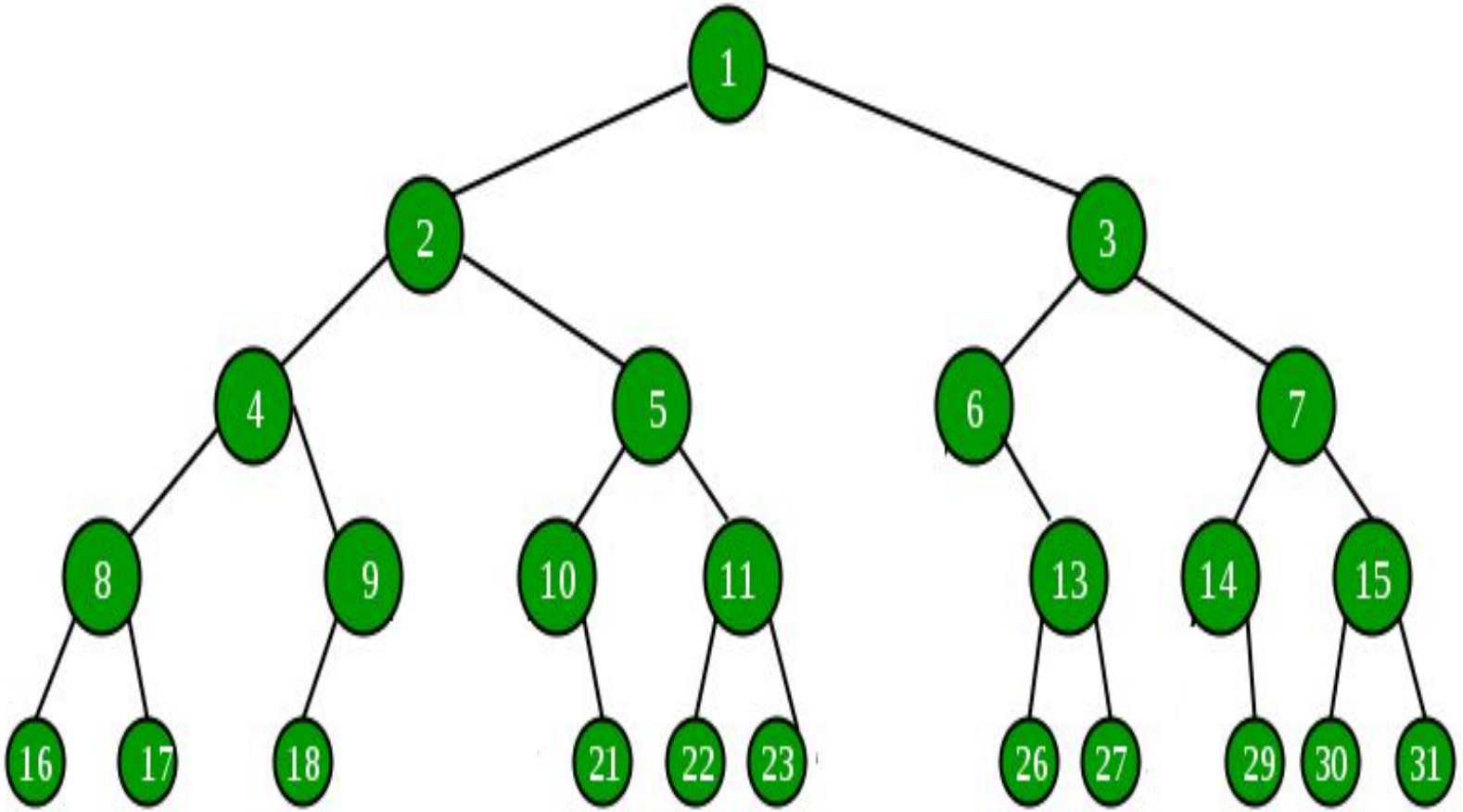
7 9 4 2 5 1 3 6 8


A
B
C
D
E
F
G
H
I

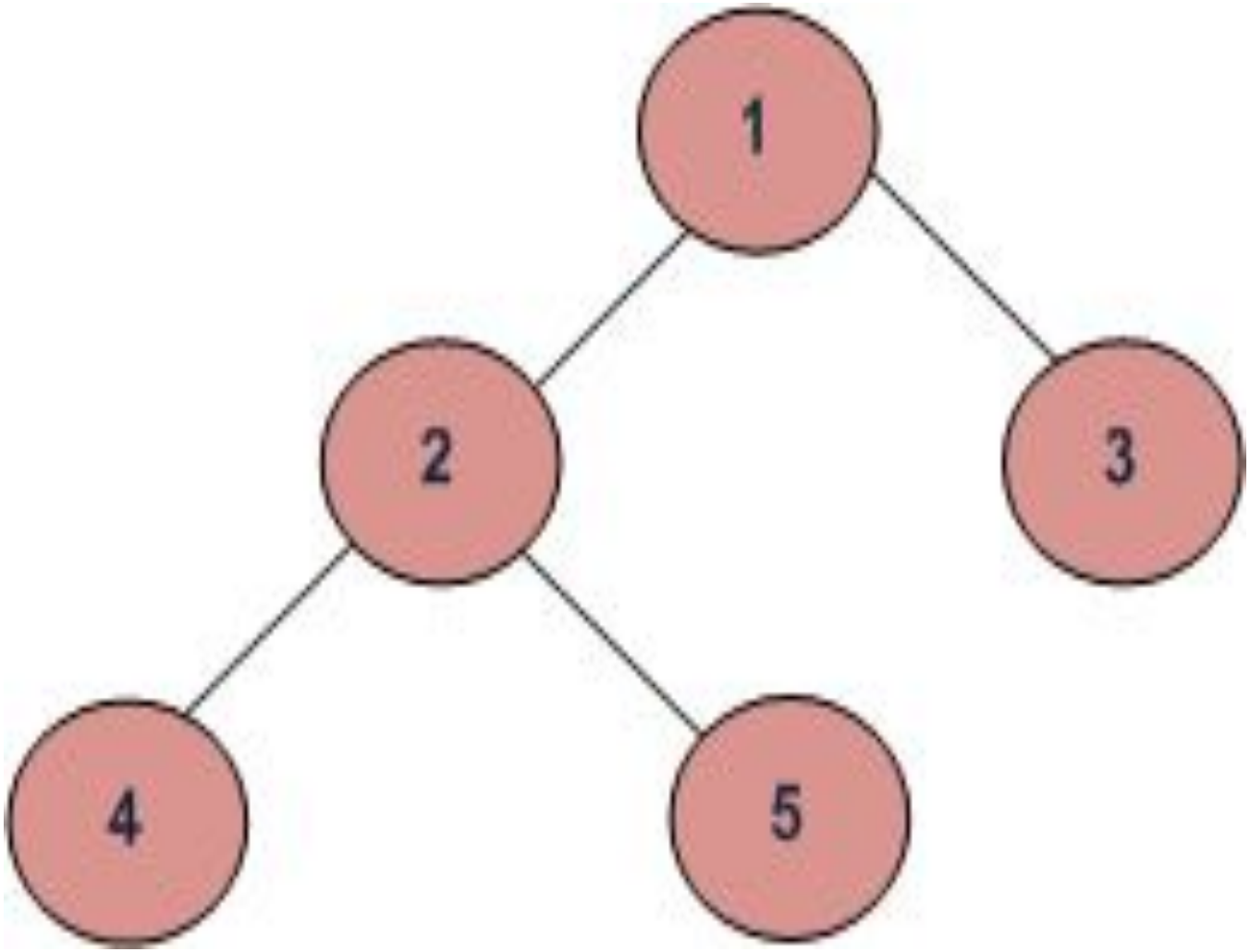
The inorder is

B D A G E C H F I.

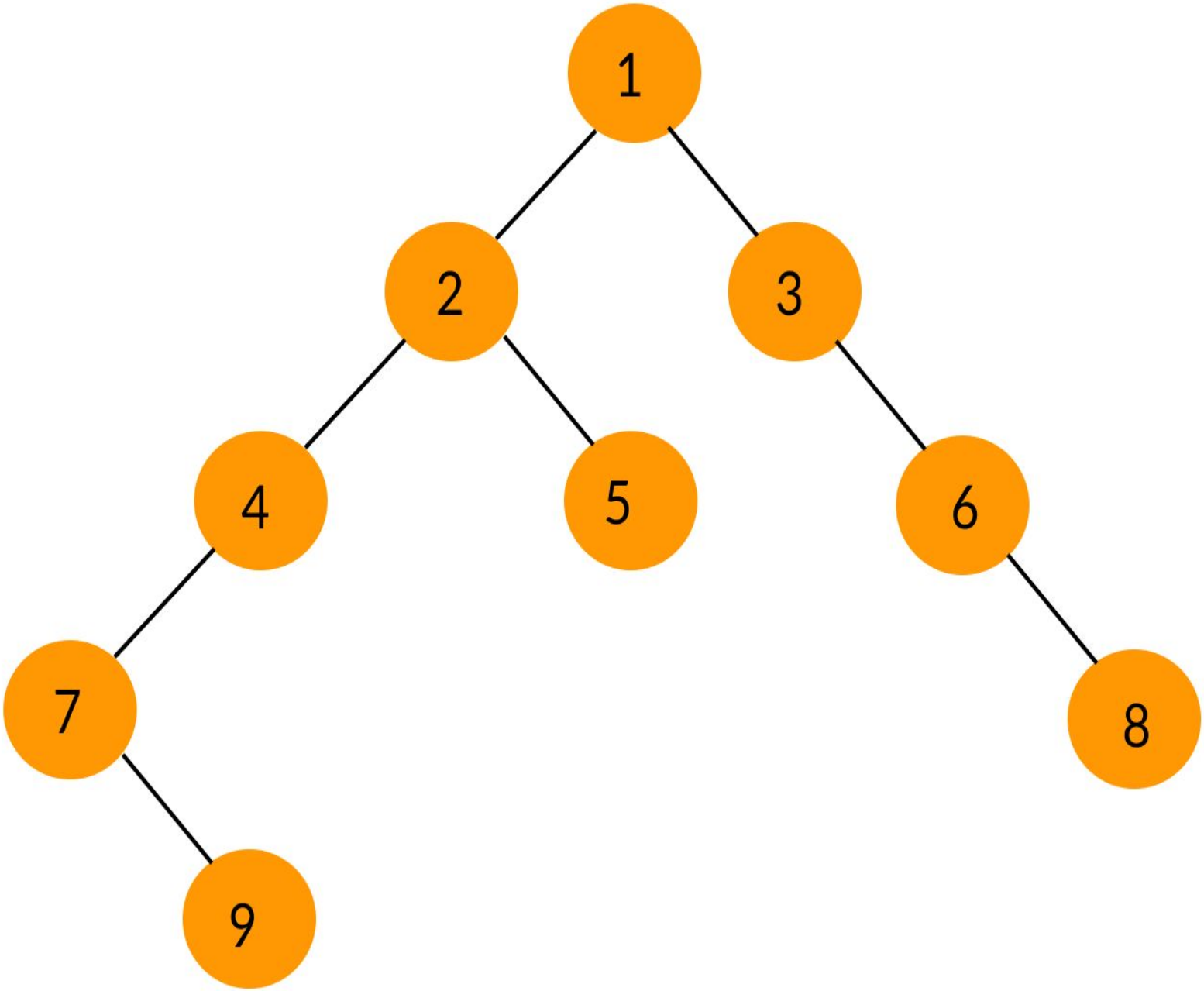
Perform inorder traversal



2) Preorder

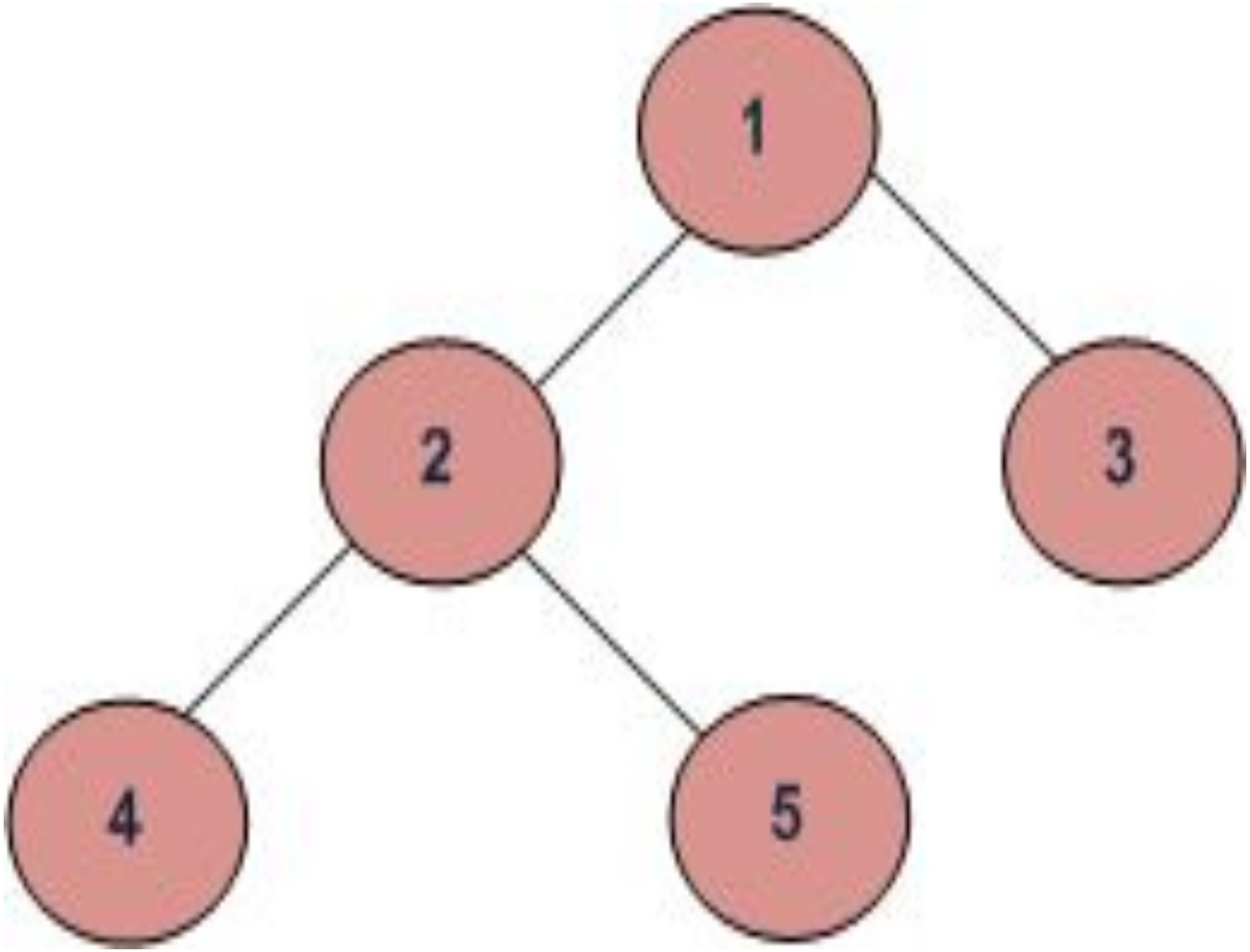


1 2 4 5 3

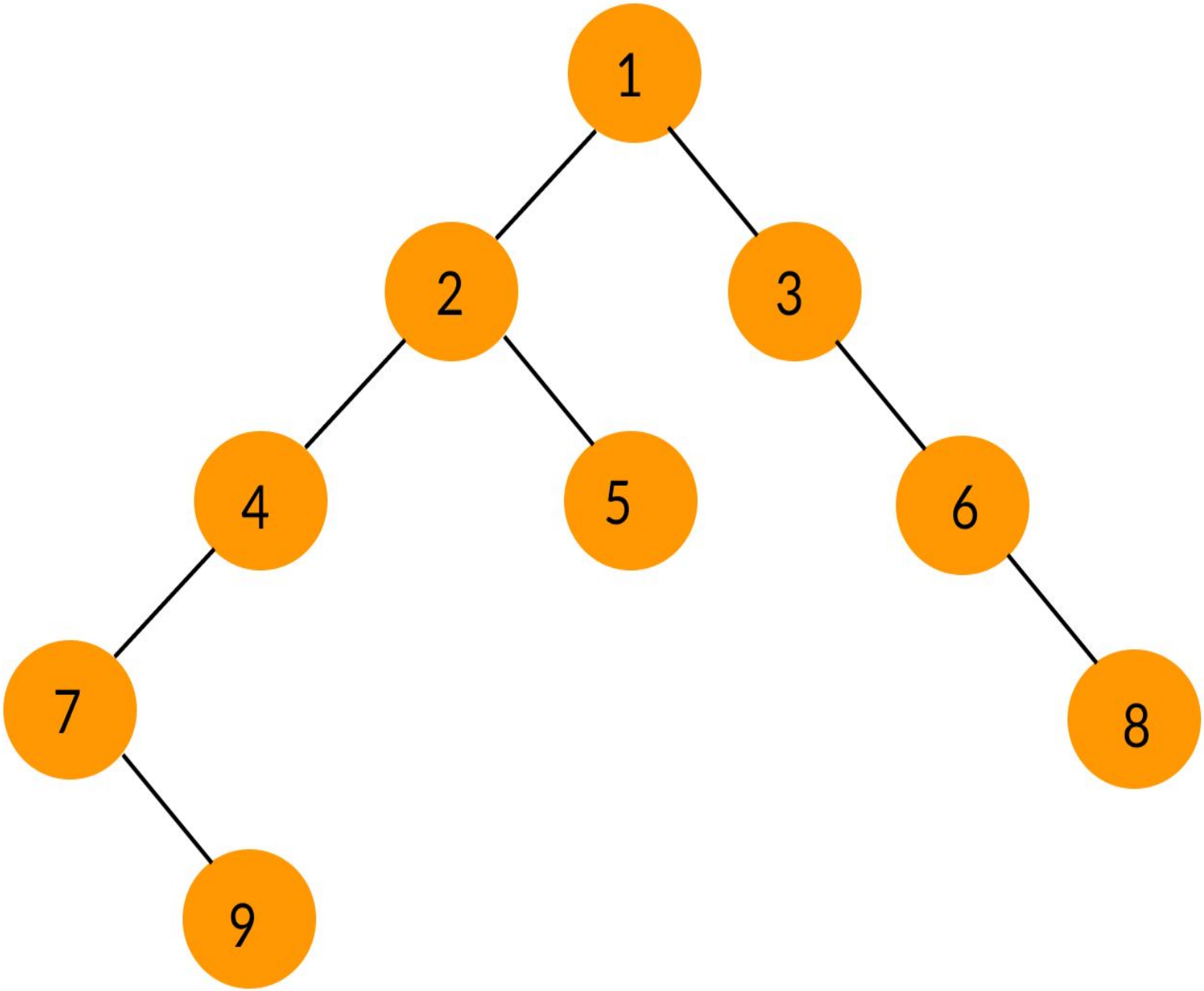

1
2
3
4
5
6
7
8
9

1 2 4 7 9 5 3 6 8

3) Postorder



4 5 2 3 1


1
2
3
4
5
6
7
8
9

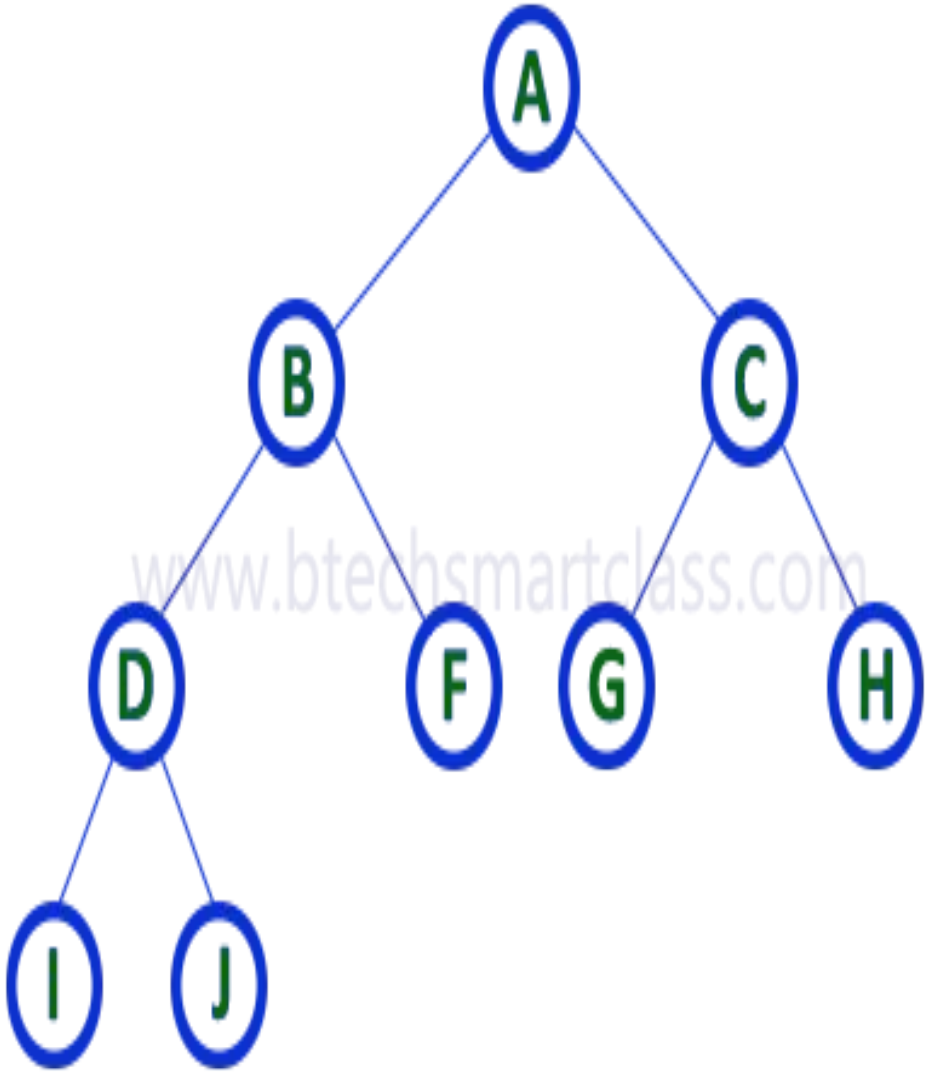
9 7 4 5 2 8 6 3 1

Strictly Binary Tree

A binary tree in which every node has either two or zero number of children is called Strictly Binary Tree. That means every internal node must have exactly two children.

Strictly binary tree is also called as **Full Binary Tree** or **Proper Binary Tree** or **2-Tree**

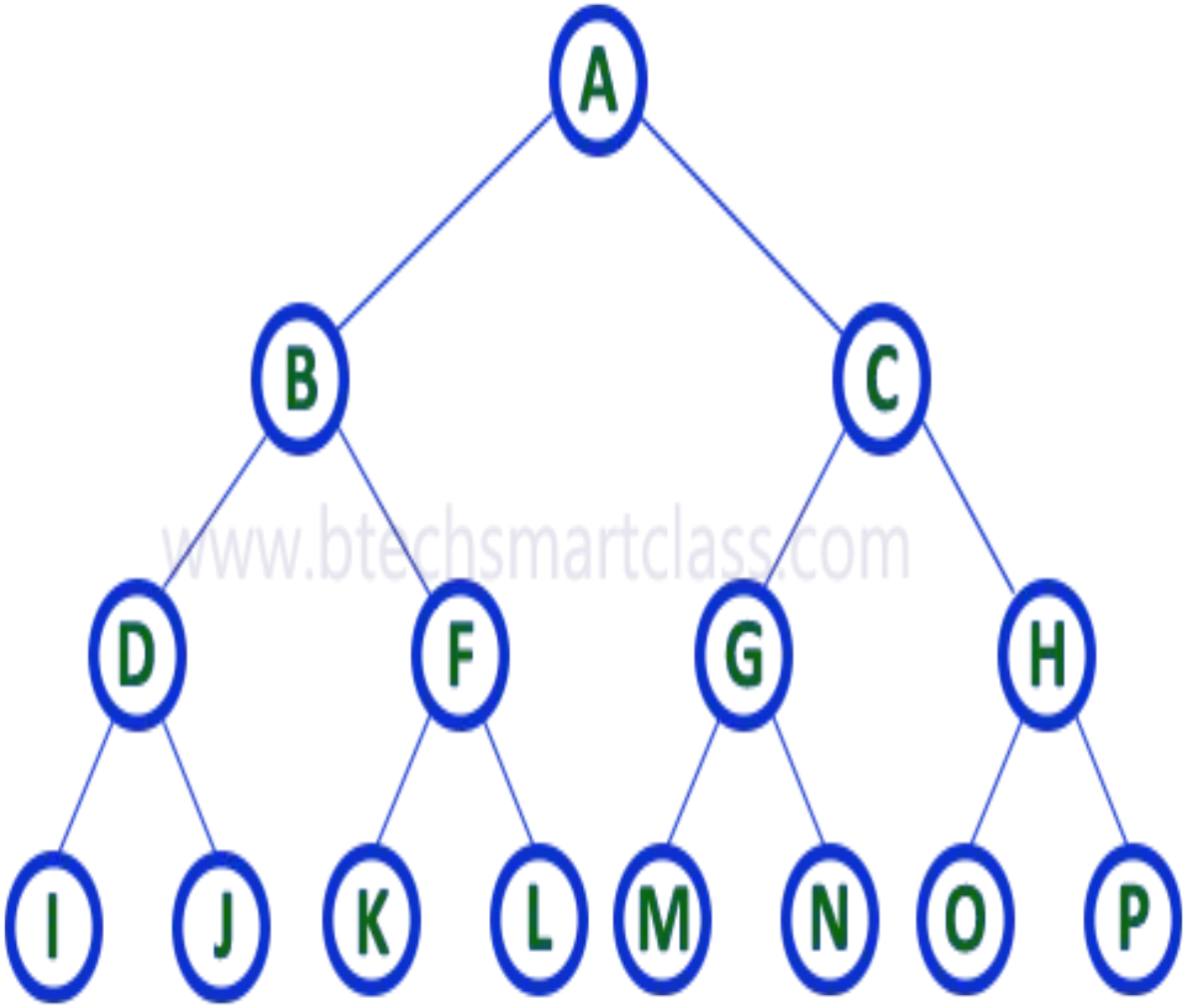
Strictly binary tree data structure is used to represent mathematical expressions.



Complete Binary Tree

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.

Complete binary tree is also called as **Perfect Binary Tree**



The **number of nodes** at **depth *d*** in a ***perfect*** **binary tree** = $\mathbf{2^d}$

The **number** of nodes **doubles** every time the **depth** increases by **1**

Skewed Binary tree

Elements are arranged only one side of the tree is known as skewed binary tree

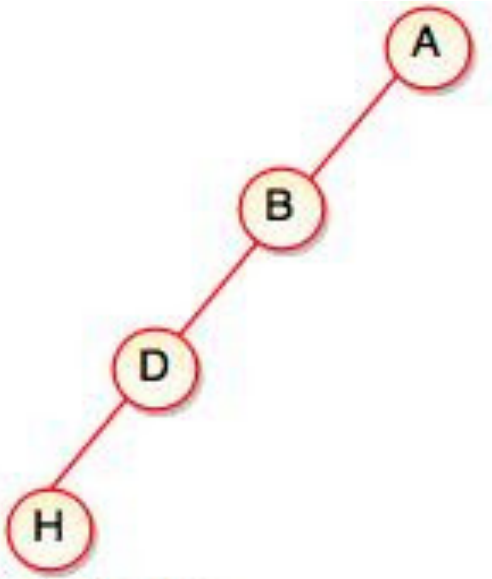


Fig. Left Skewed Binary Tree

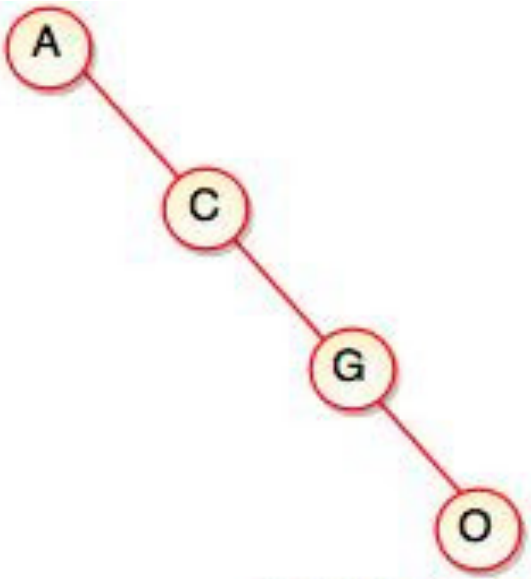
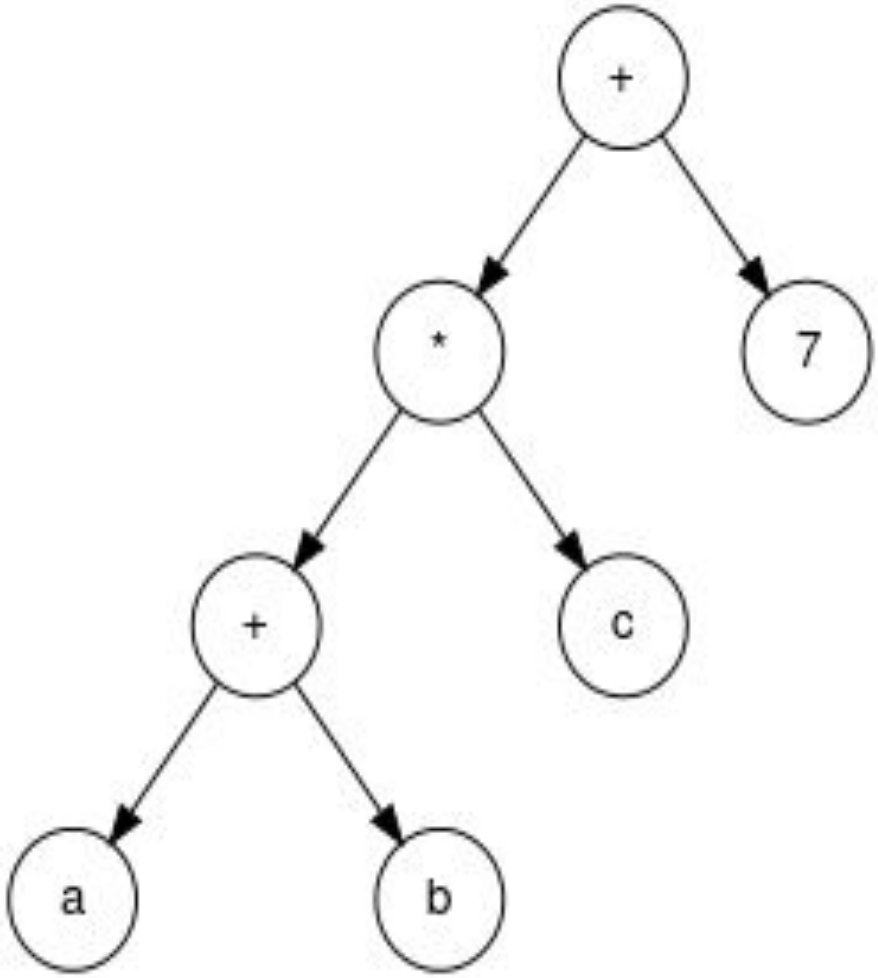


Fig. Right Skewed Binary Tree

Binary Expression Tree

Is a srictly binary tree in which leaf node contains operands and non leaf node contains operators



Construction of Expression Tree

We consider postfix expression as an input:

Create a Stack

- Read next input symbol
- If the symbol is an operand, then push it into the stack.
- If the symbol is an operator, pop out two trees (T1 and T2) from the stack. Create a new tree with the operator as the root and T1 and T2 as the right and left sub trees. Push this new tree back into the stack.
- Repeat this procedure until the whole input is read.
- At the end, the stack will contain a single tree which would be the output.

a+b*c =a b + c *

a*b/c-d = a b * c / d -

Create a binary expression tree

1. A * B + C * D = A B * C D * +

2. A + B * C + D = A B C * + D +

3. A + B + C + D = A B + C + D +

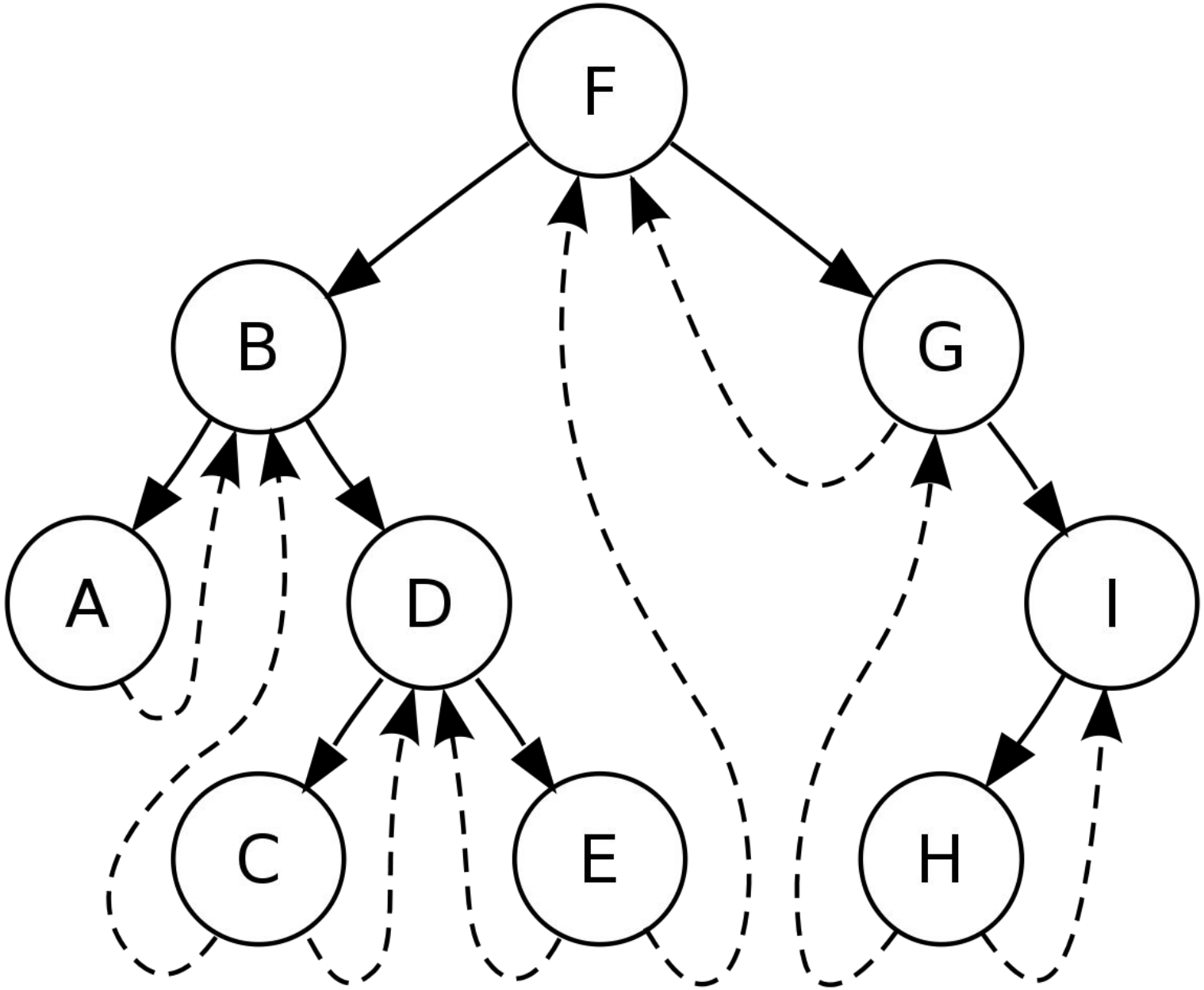
Threaded binary tree

A thread is a pointer that replaces the null pointer in a node

A threaded binary tree is a binary tree in which:

A null left pointer in a BT node is replaced by a pointer to the predecessor of the node, if the node has an inorder predecessor, and

A null right pointer in a BT node is replaced by a pointer to the successor of the node, if the node has an inorder successor.


F
B
G
A
D
I
C
E
H

Complete and submitt the problem on today before 5.pm on sibucash@gmail.com

1. Create a binary expression tree using the expression
   A/b+c*d/g-c then convert it into threaded binary tree.

2. Create a binary expression tree using the expression
   2 ^ 3 * 4 / 5 – 6 +7 – 8 + 9 * 10
   then convert it into threaded binary tree.

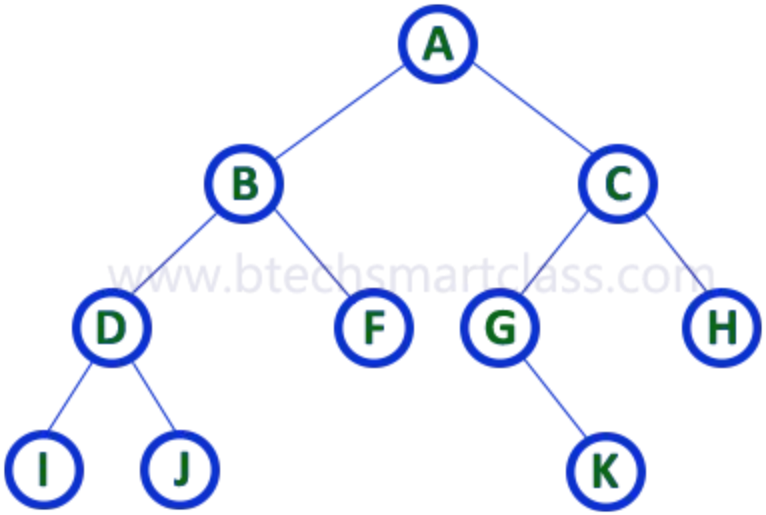
Binary Tree Representations

A binary tree 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.

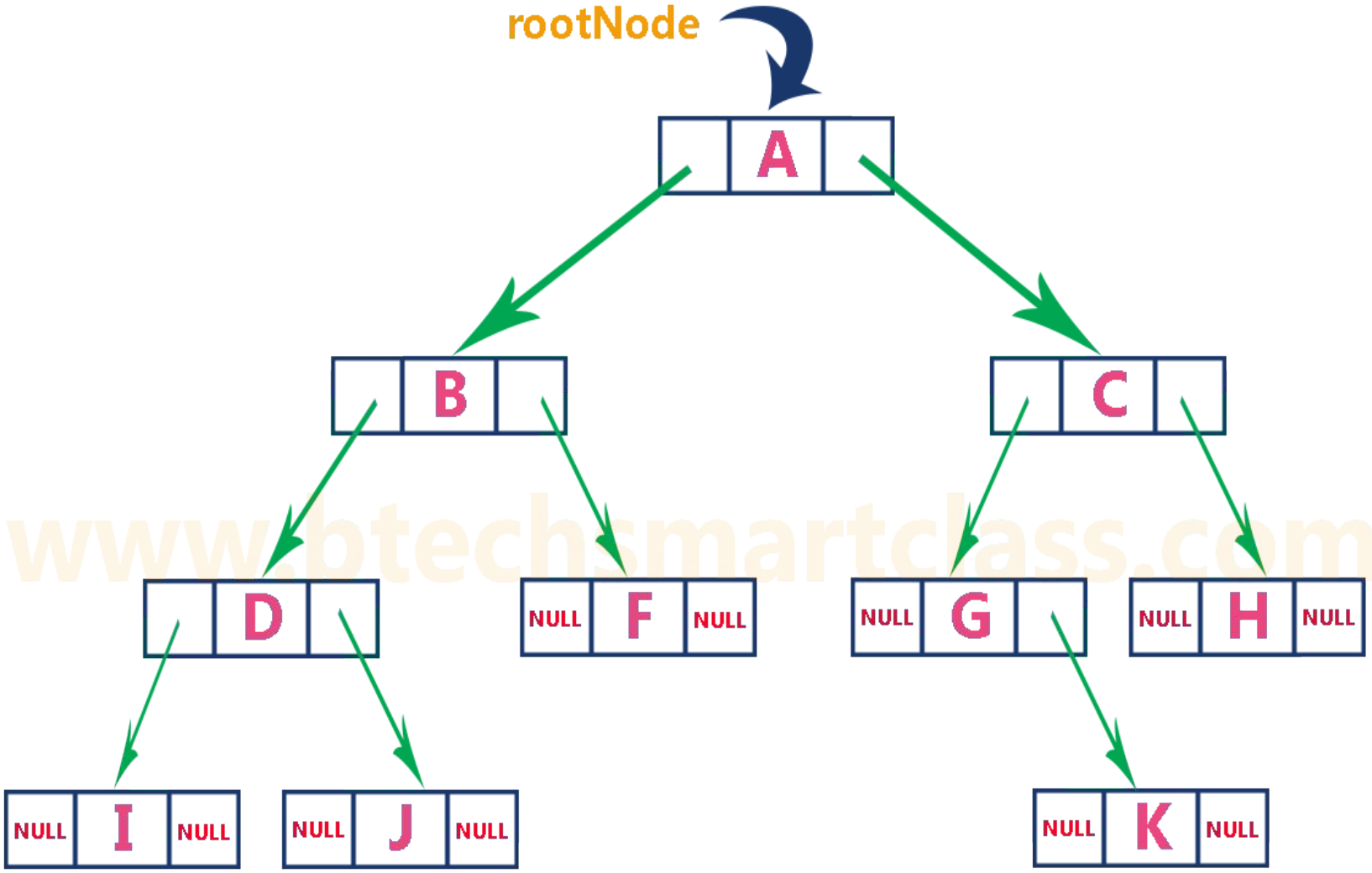
A	B	C	D	F	G	H	I	J	-	-	-	K	-	-	-	-	-	-	-	-

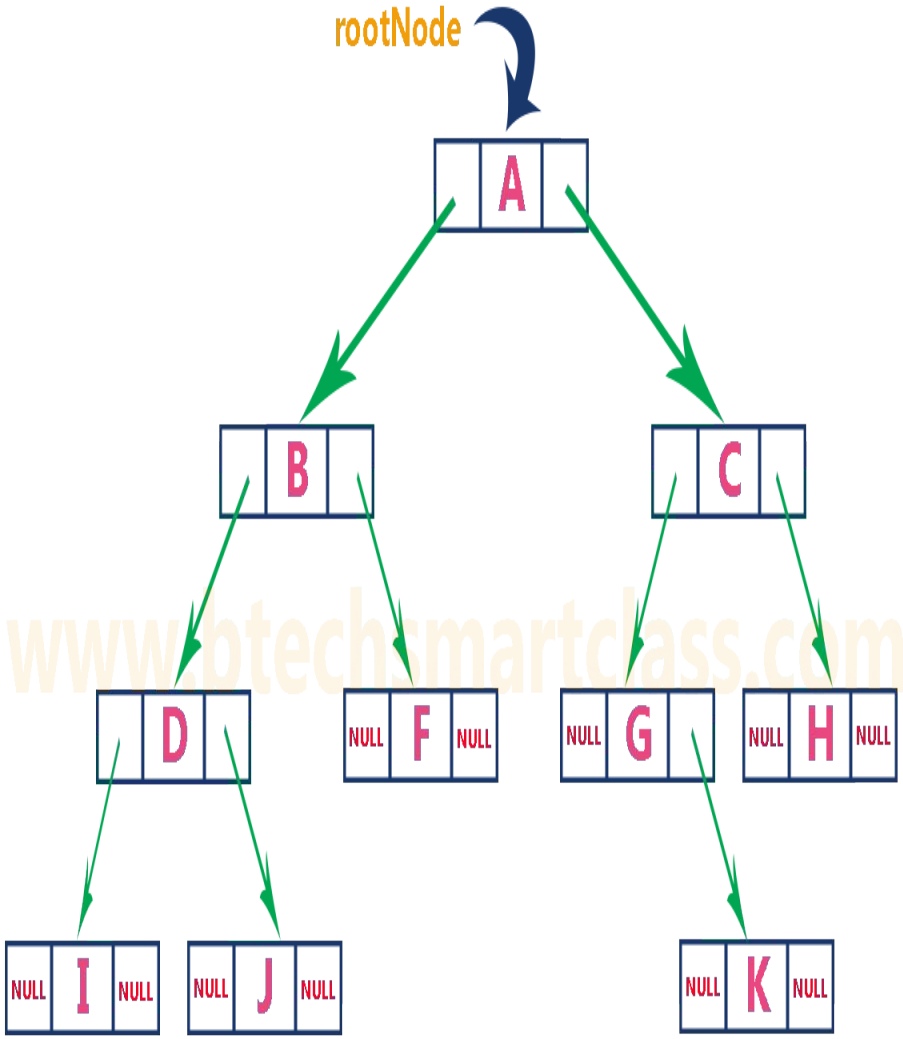
2. Linked List Representation of Binary Tree

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...





START

1

Avail

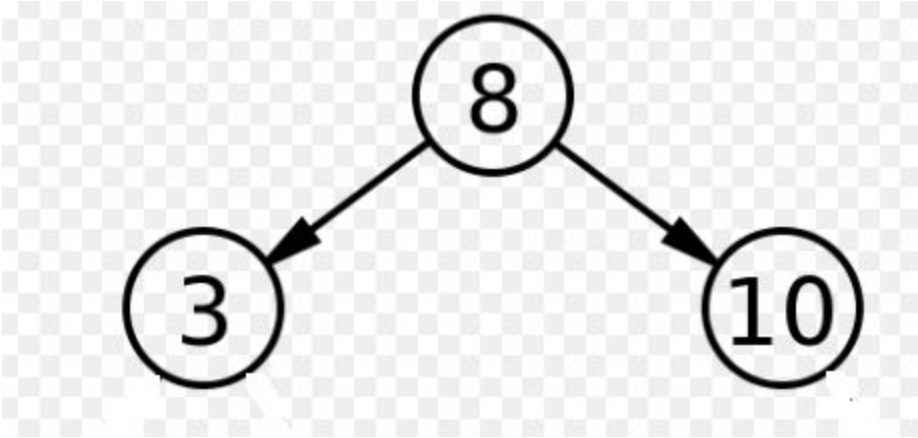
11

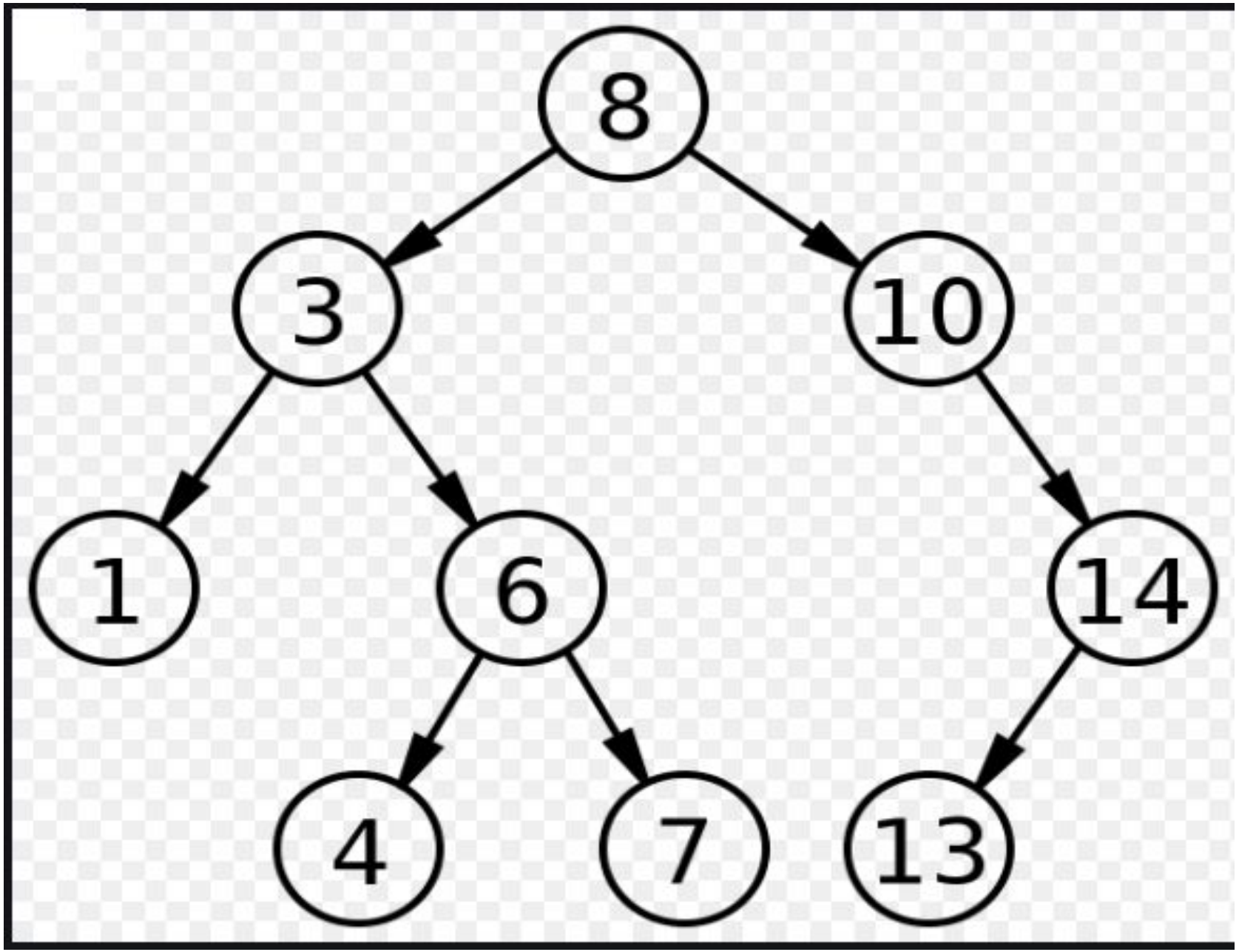
	INfo	Left	Right
1	A	2	3
2	B	4	5
3	C	6	7
4	D	8	9
5	F	null	null
6	G	null	10
7	H	null	null
8	I	null	null
9	J	null	null
10	K	null	null
11		12	
12		null	

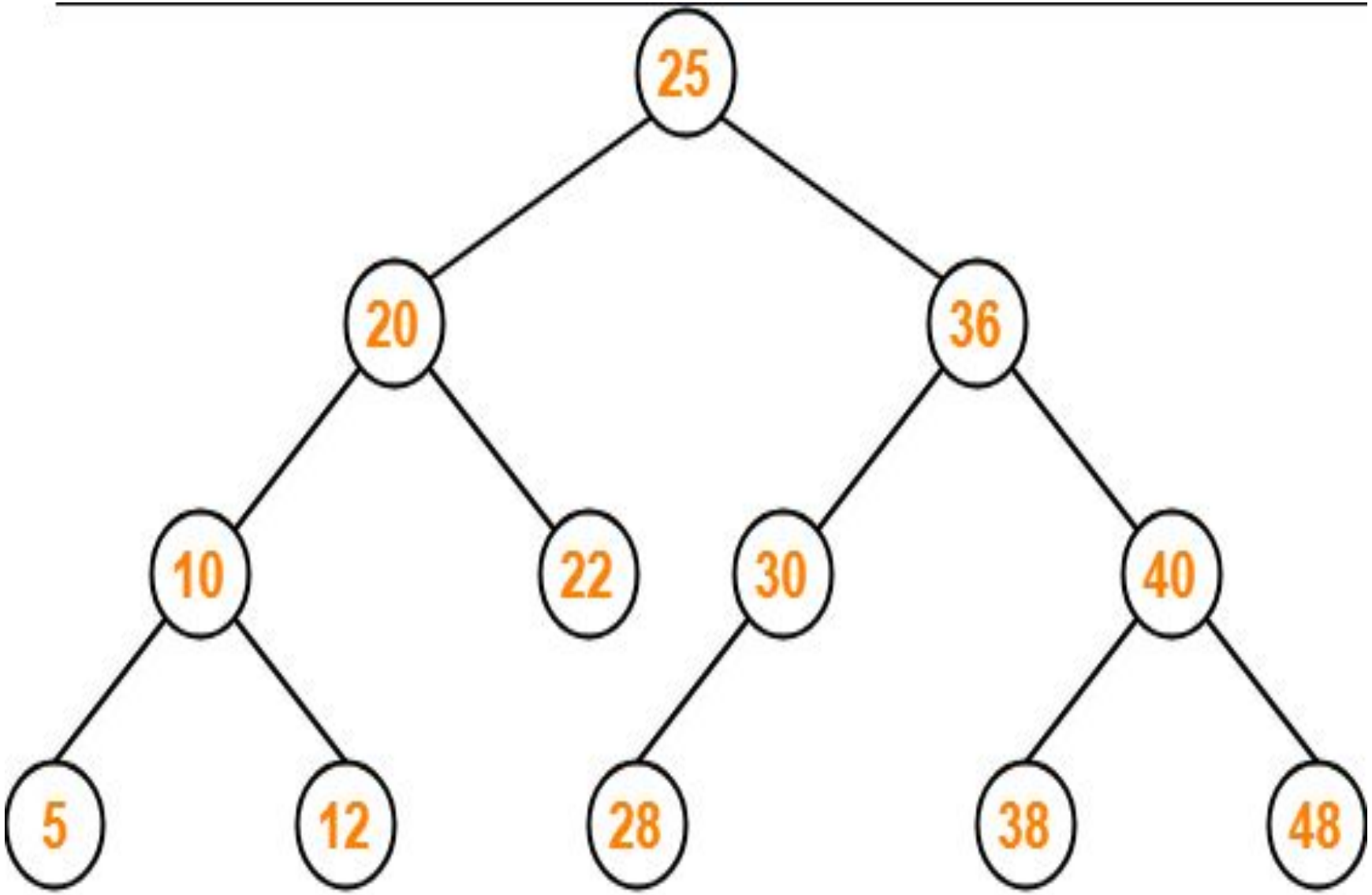
Binary Search Tree

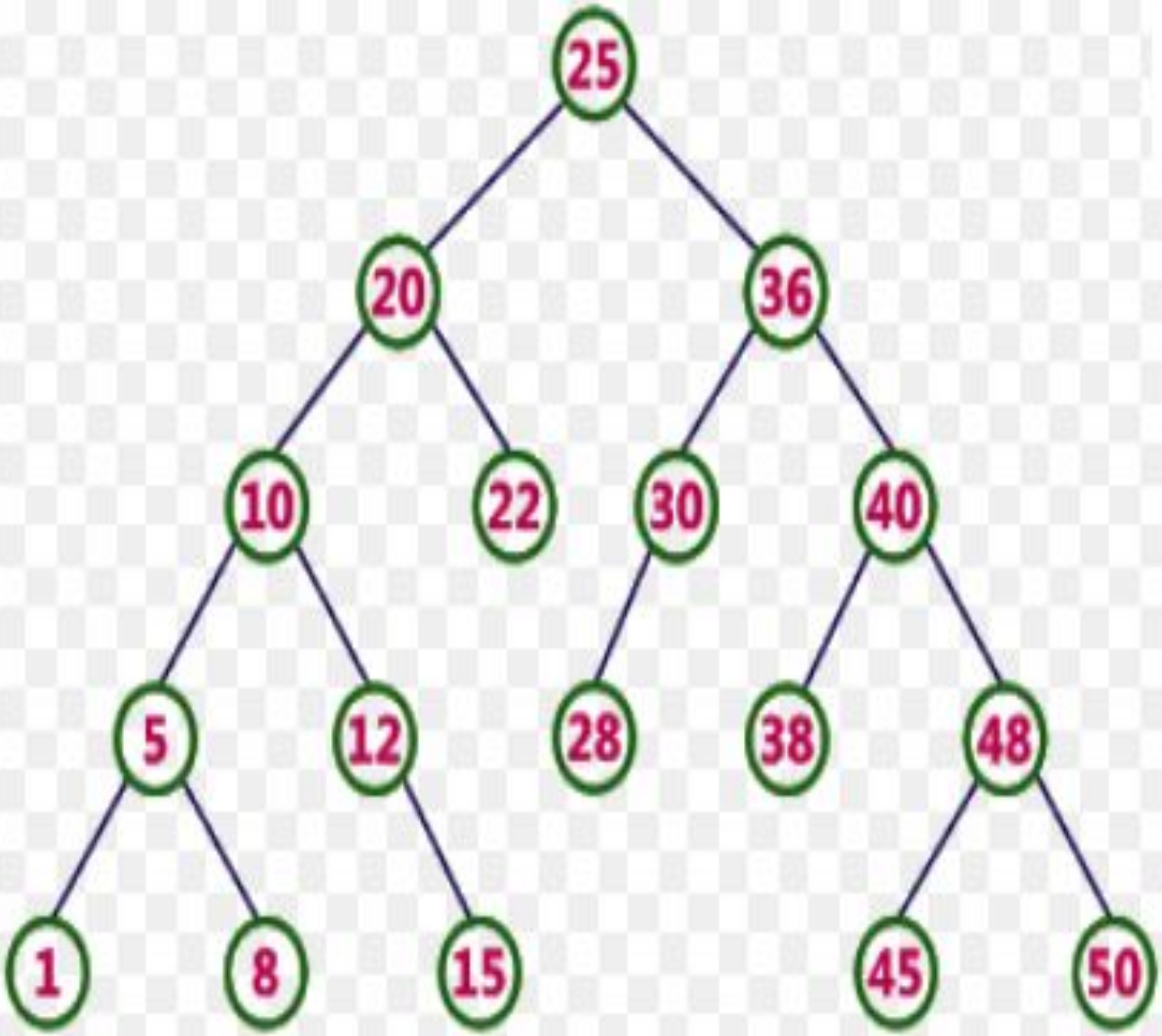
Is a tree with following rules :

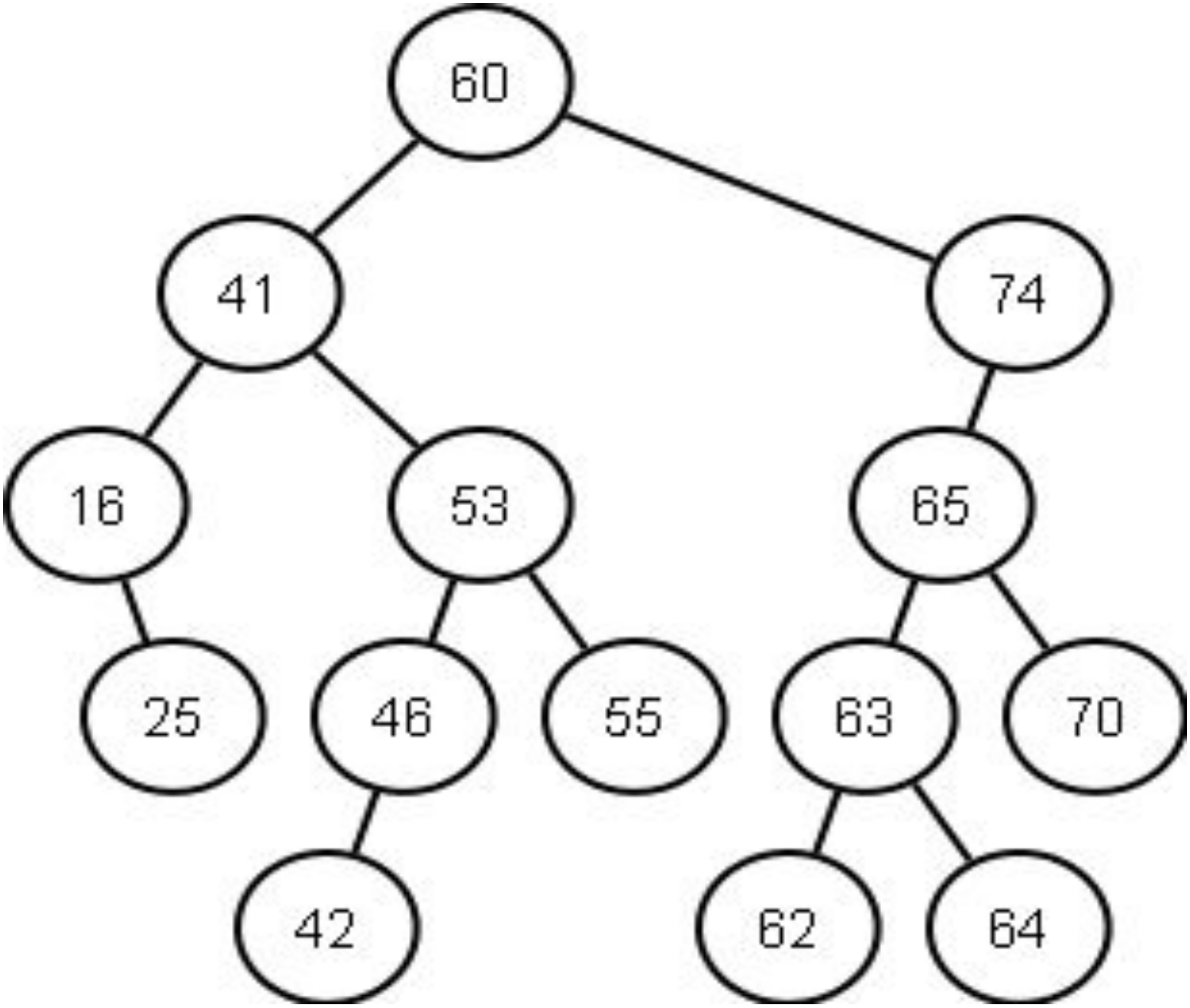
- The value of key in the left child or left sub tree is less than the value of the root .
- The value of key in the right child or right sub tree is greater than or equal to the value of the root

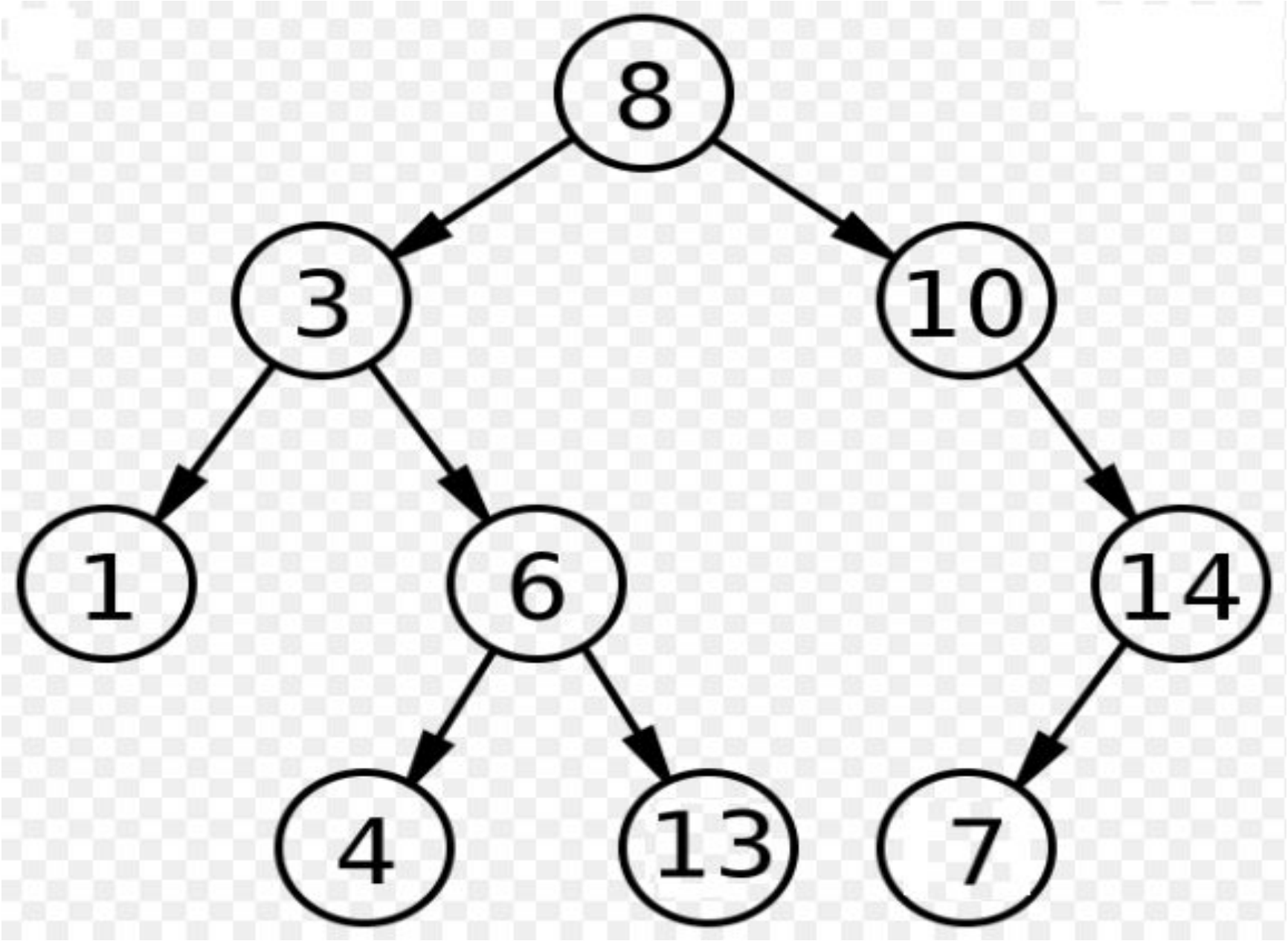



8
3
10
1
6
14
4
7
13


25
20
36
10
22
30
40
5
12
28
38
48


25
20
36
10
22
30
40
5
12
28
38
48
1
8
15
45
50


60
41
74
16
53
65
25
46
55
63
70
42
62
64


8
3
10
1
6
14
4
13
7

Binary Search Tree – Search

Step 1: Compare the element with the root/current node of the tree.

Step 2: If the item is matched then return the location of the node.

Step 3: if item is less than the element present on root/current node, then move to the left sub-tree.

Step 4: if item is greater than or equal to the element present on root/current node , then move to the right sub-tree.

Step 5: Repeat step 1 to 5 until match found. If element is not found then return NULL.

Binary Search Tree – Insertion

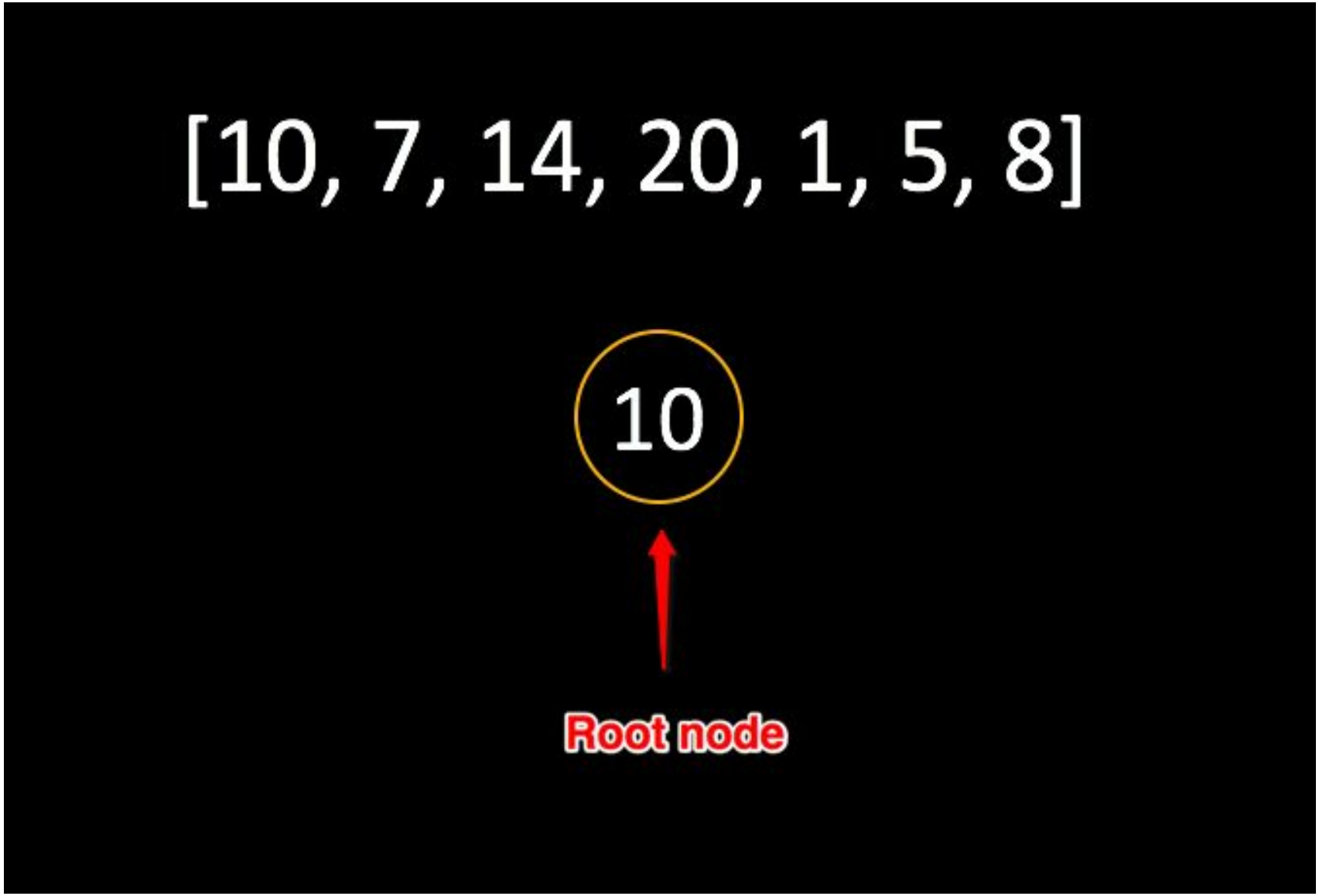
Step 1: Compare the element with the root/current node of the tree.

Step 2: if item is less than the element present on root/current node, then move to the left sub-tree. If that location is free, that is the positinon of the new node

Step 3: if item is greater than or equal to the element present on root/current node , then move to the right sub-tree. If that location is free, that is the position of the new node

Step 4: Repeat step 1 to 3.

Create/Insert a binary search tree

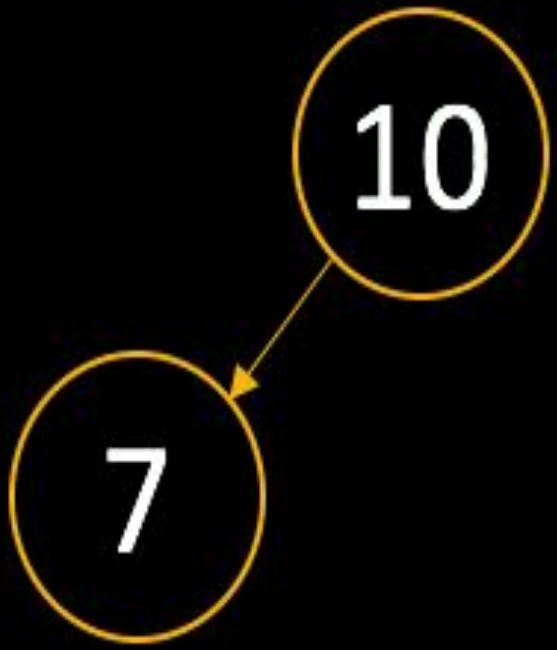


[10, 7, 14, 20, 1, 5, 8]

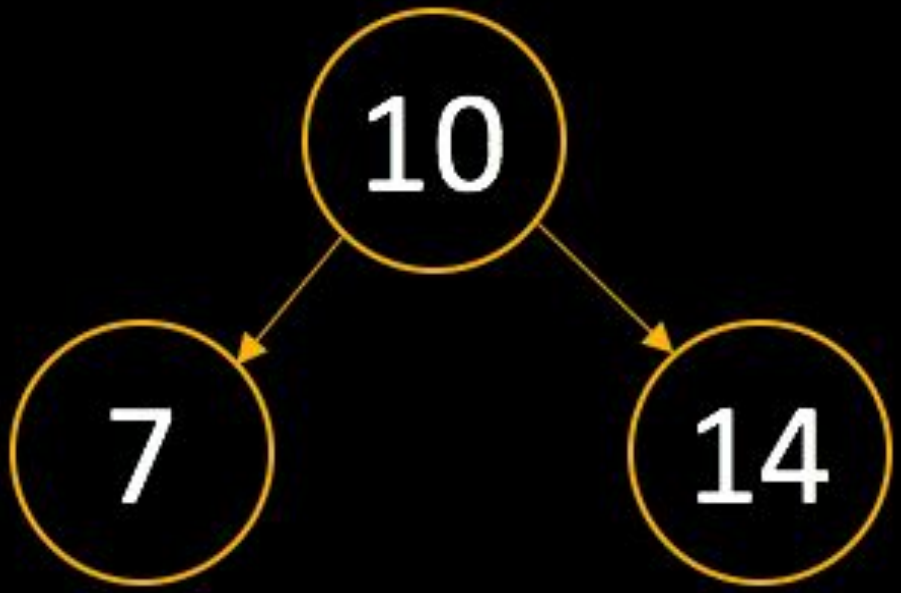
7 < 10

7 >= 10

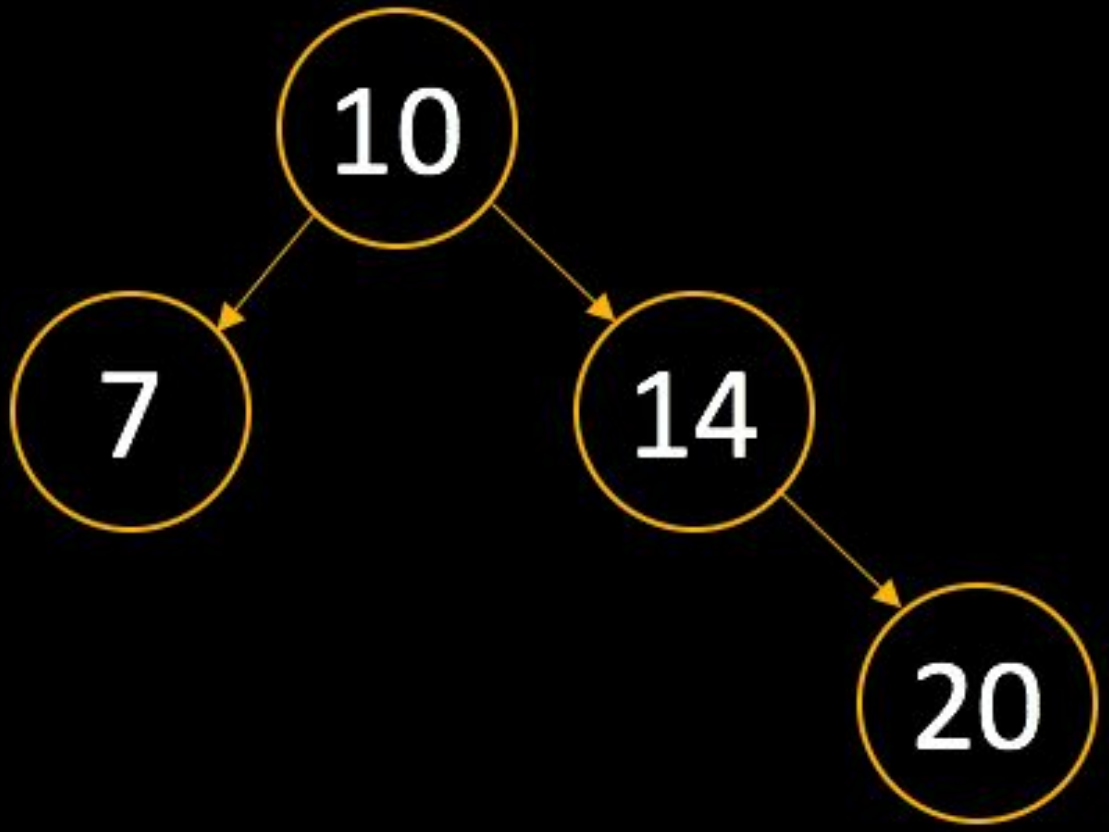
[10, 7, 14, 20, 1, 5, 8]



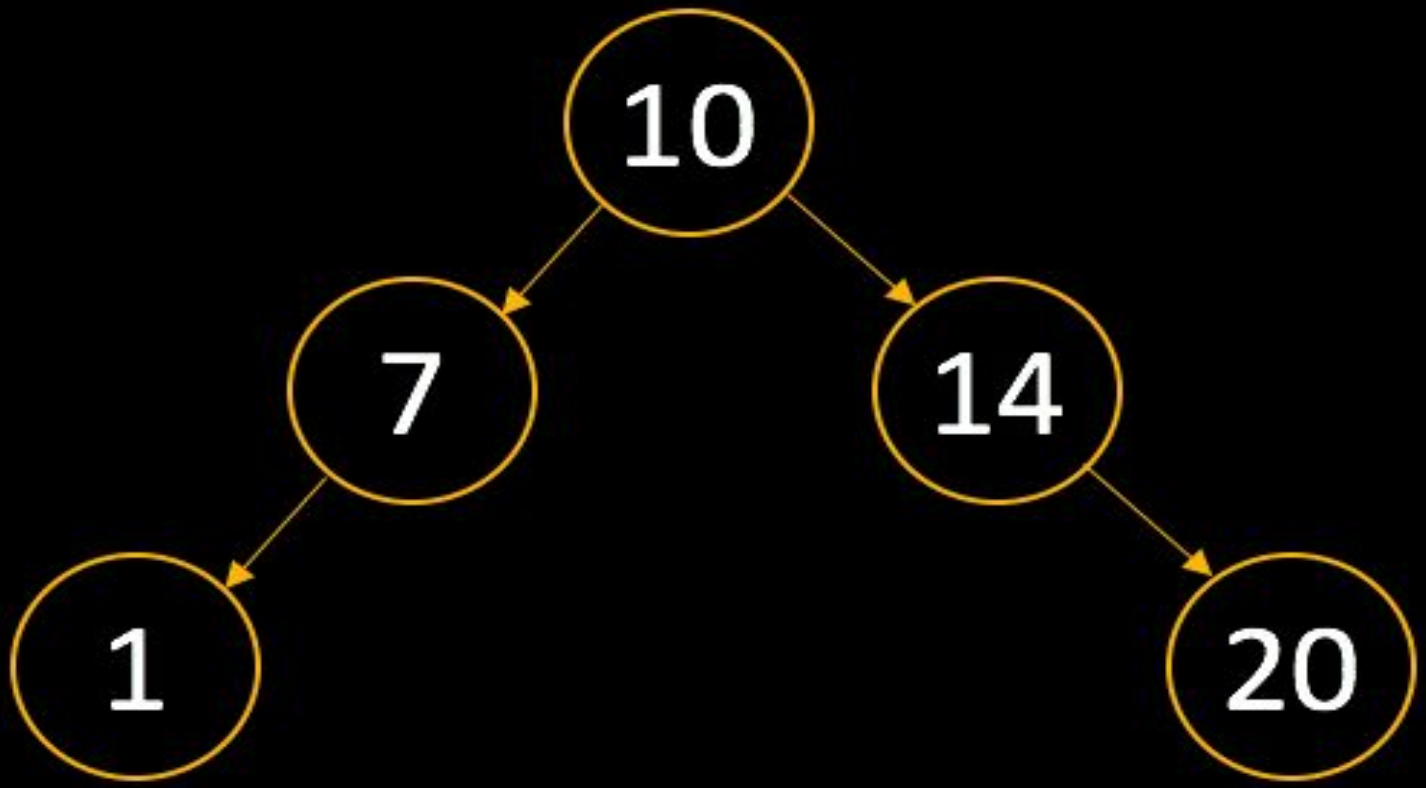
[10, 7, 14, 20, 1, 5, 8]



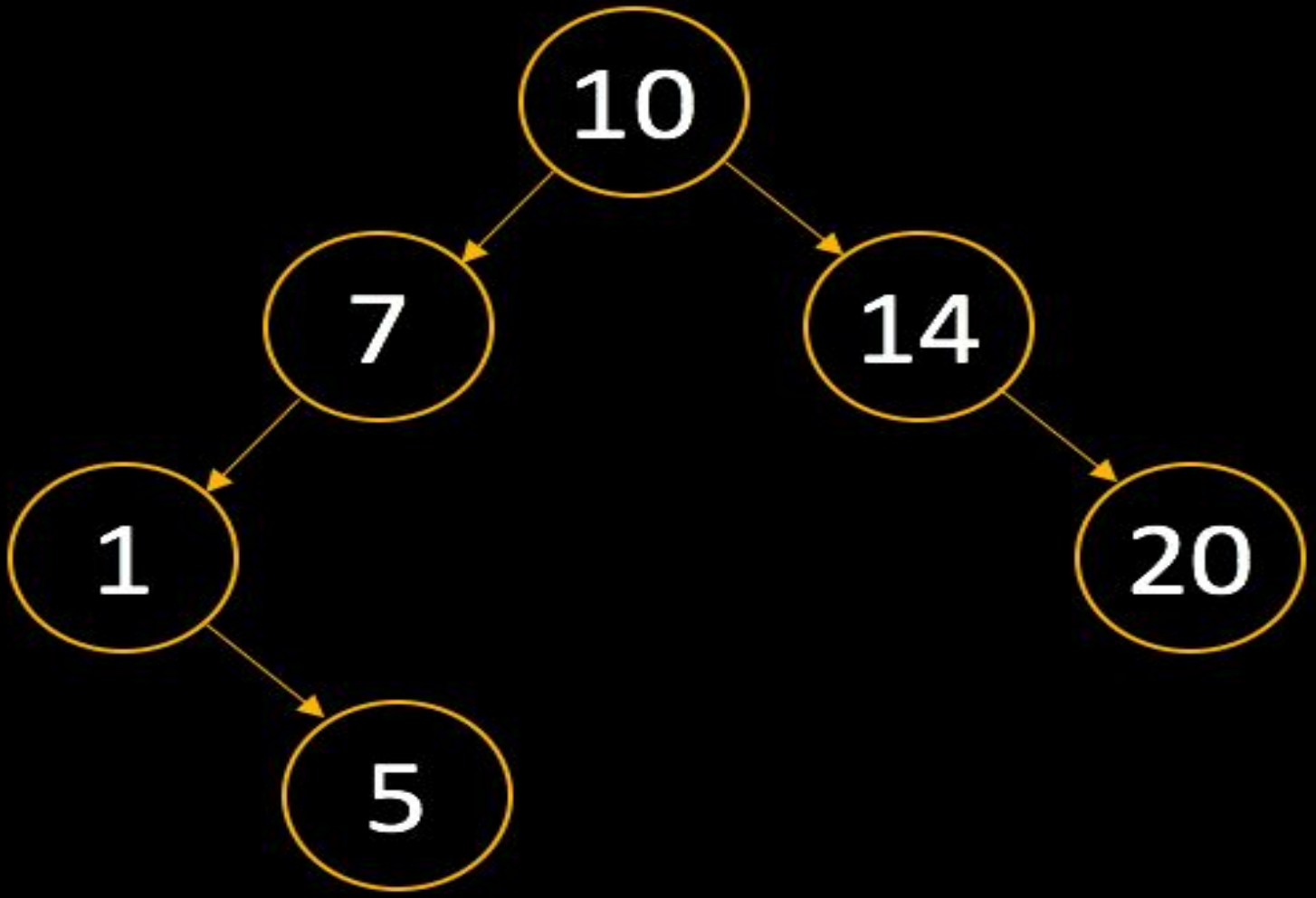
[10, 7, 14, 20, 1, 5, 8]



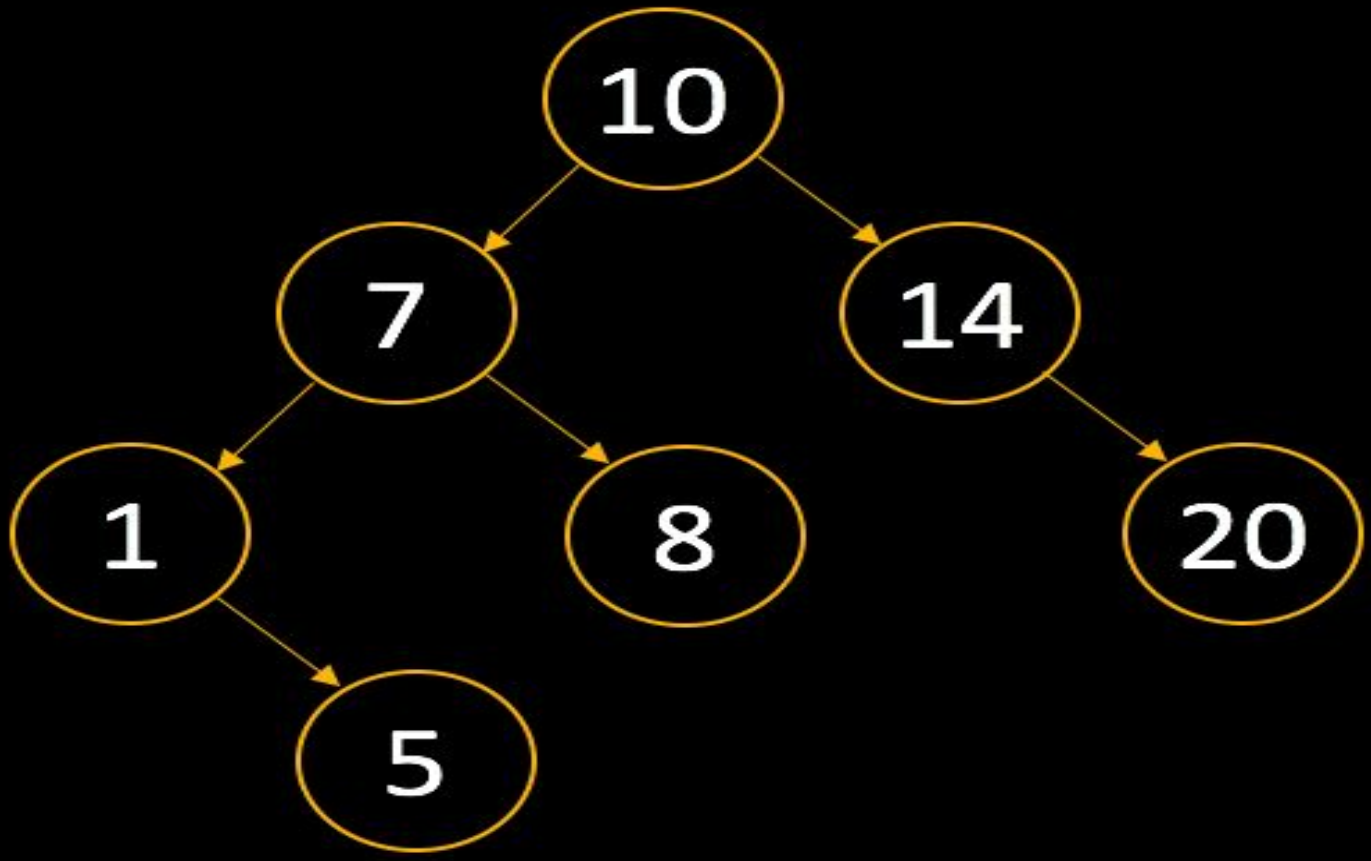
[10, 7, 14, 20, 1, 5, 8]



[10, 7, 14, 20, 1, 5, 8]



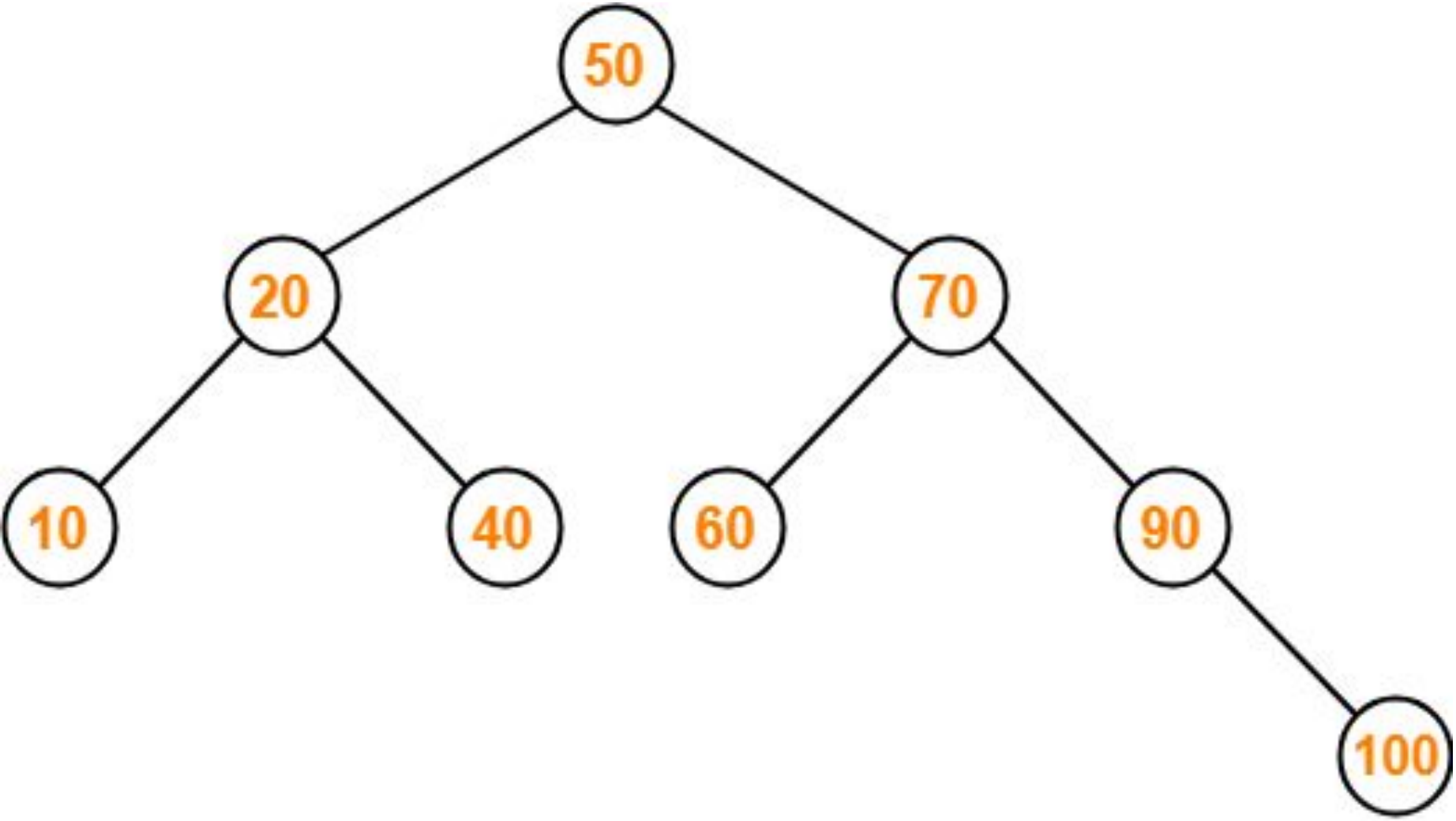
[10, 7, 14, 20, 1, 5, 8]



Done

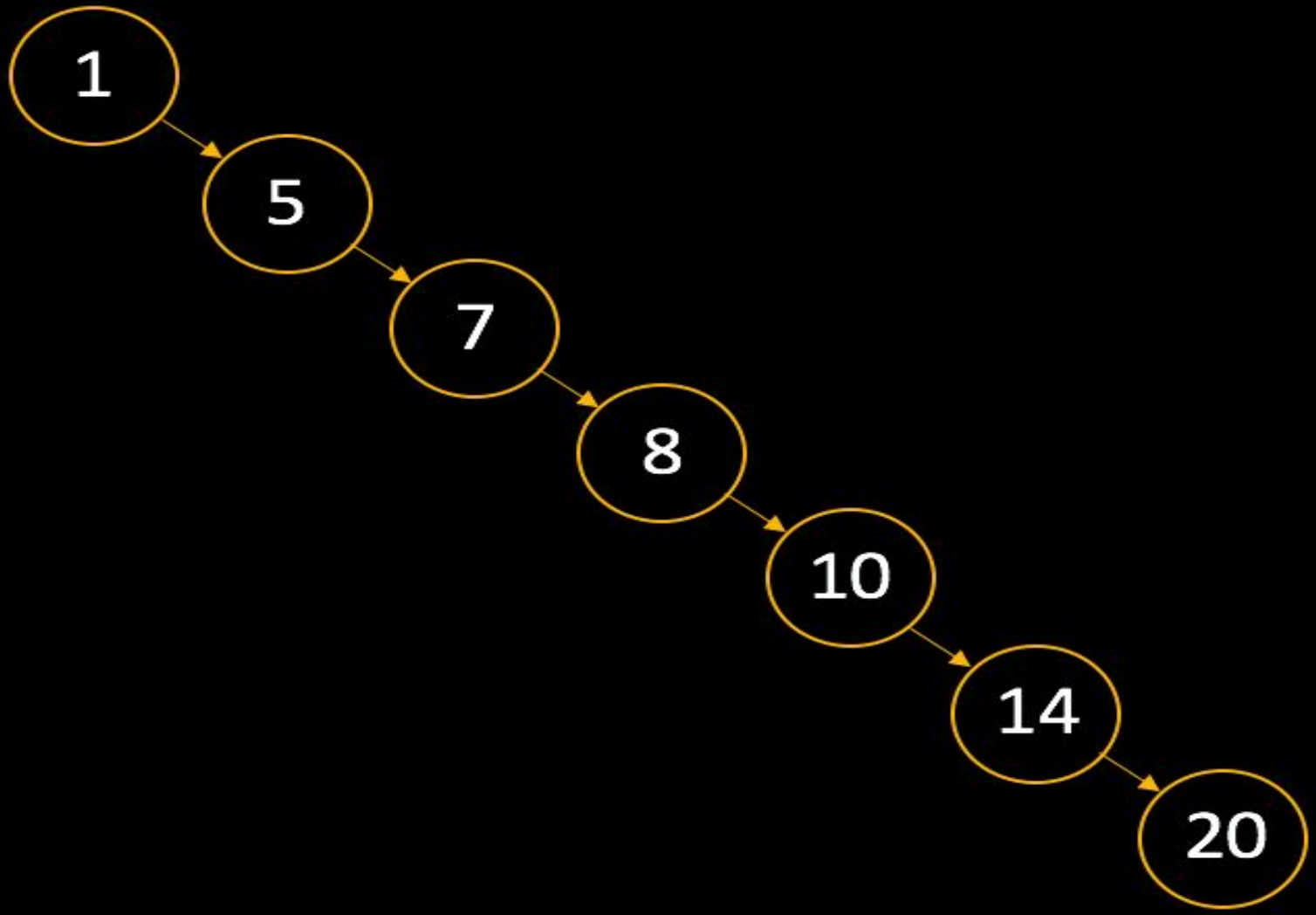
Construct a Binary Search Tree (BST) for the following sequence of numbers-

50, 70, 60, 20, 90, 10, 40, 100



Binary Search Tree

[1, 5, 7, 8, 10, 14, 20]



Deleating in a binary search tree

The node ‘n’ is deleted from the tree depends pramarly on the no : of children on node n . There are three cases .

CASE 1 : N has no children . then n is simply deleted from T .

.

CASE 2 : N has exactly one child . Then n is deleted from the tree by placing the child on that position .

CASE 3 : N has 2 children . Let s(n) denote the inorder successor of N . then N is deleted from T by first deleting s (n) and place that position