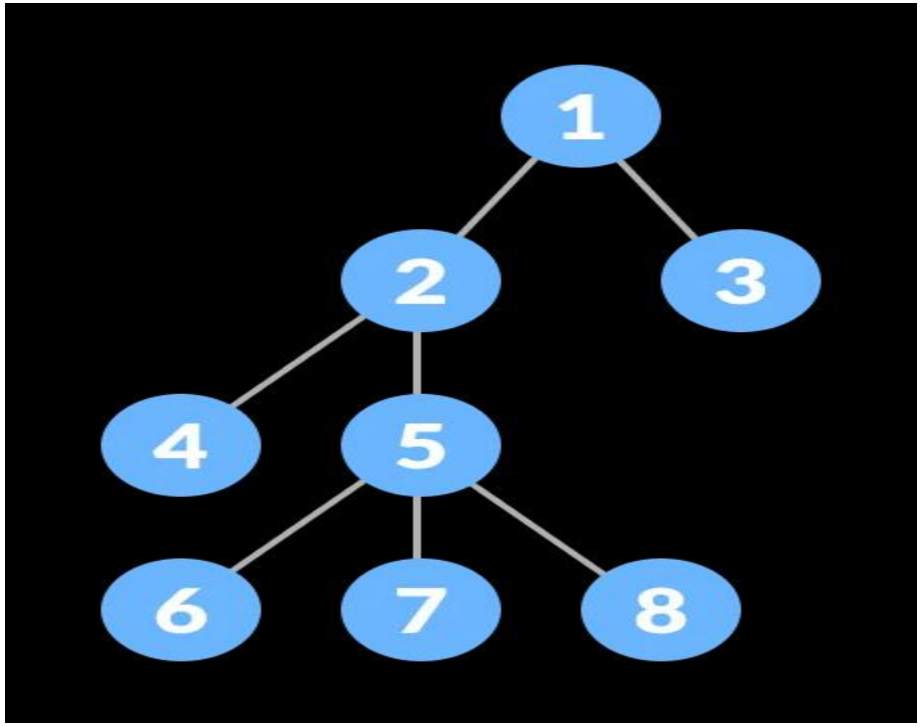
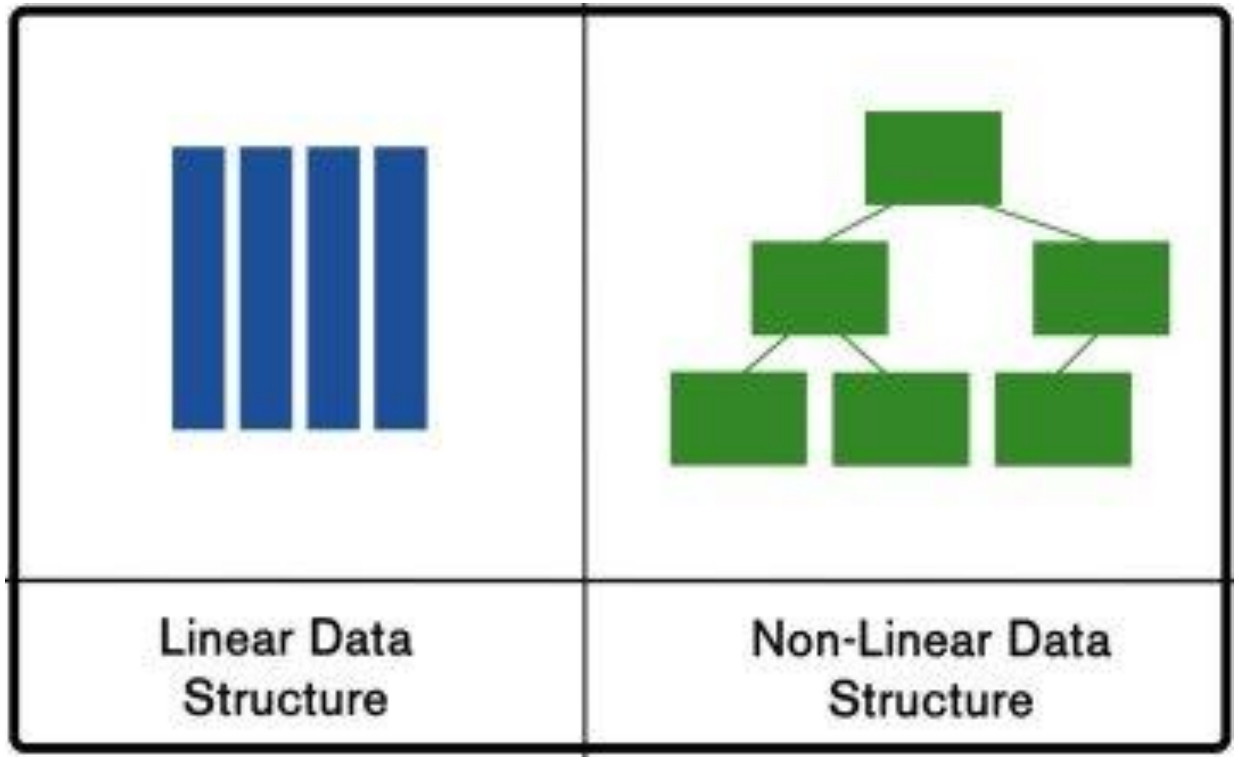
Tree

Tree

- A tree is a nonlinear data structure used to represent entities that are in some hierarchical relationship



Linear vs Non-Linear Data Structures



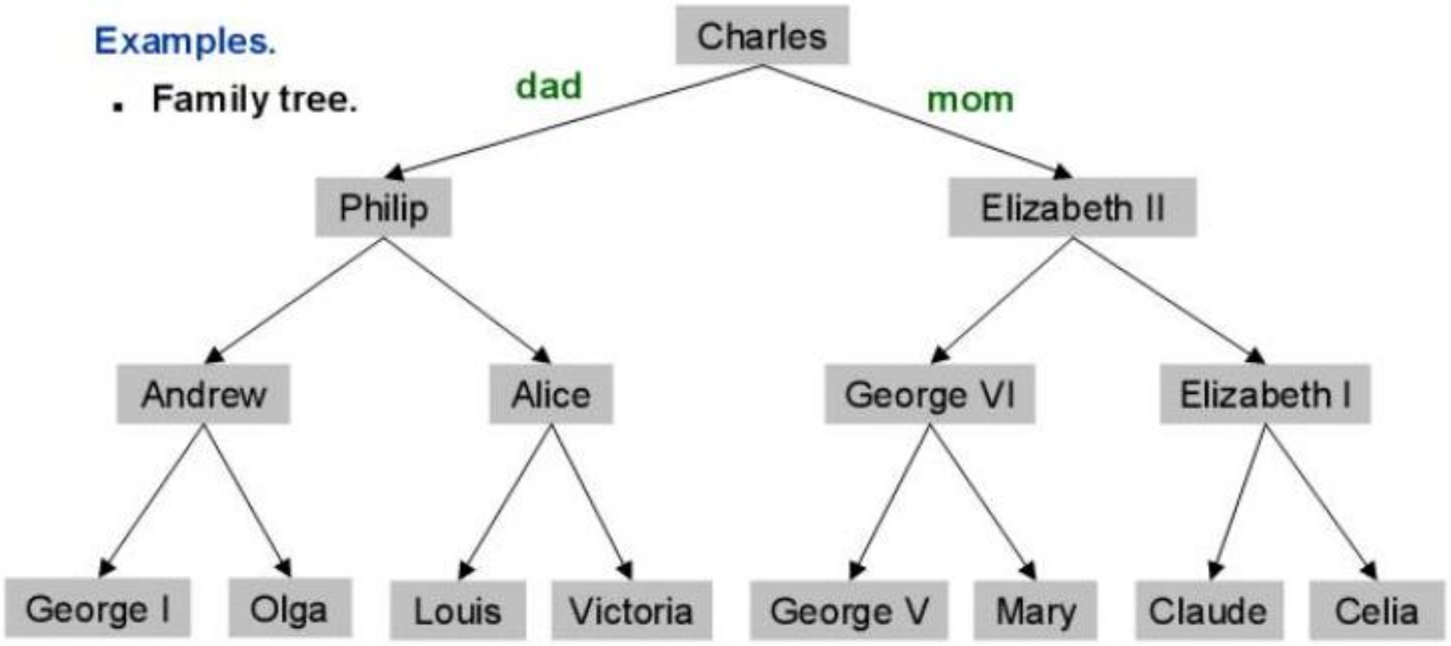
Linear vs Non-Linear Data Structures

S.NO	Linear Data Structure	Non-linear Data Structure
1.	In a linear data structure, data elements are arranged in a linear order where each and every elements are attached to its previous and next adjacent.	In a non-linear data structure, data elements are attached in hierarchically manner.
2.	In linear data structure, single level is involved.	Whereas in non-linear data structure, multiple levels are involved.
3.	Its implementation is easy in comparison to non-linear data structure.	While its implementation is complex in comparison to linear data structure.
4.	In linear data structure, data elements can be traversed in a single run only.	While in non-linear data structure, data elements can't be traversed in a single run only.
5.	In a linear data structure, memory is not utilized in an efficient way.	While in a non-linear data structure, memory is utilized in an efficient way.
6.	Its examples are: array, stack, queue, linked list, etc.	While its examples are: trees and graphs.
7.	Applications of linear data structures are mainly in application software development.	Applications of non-linear data structures are in Artificial Intelligence and image processing.

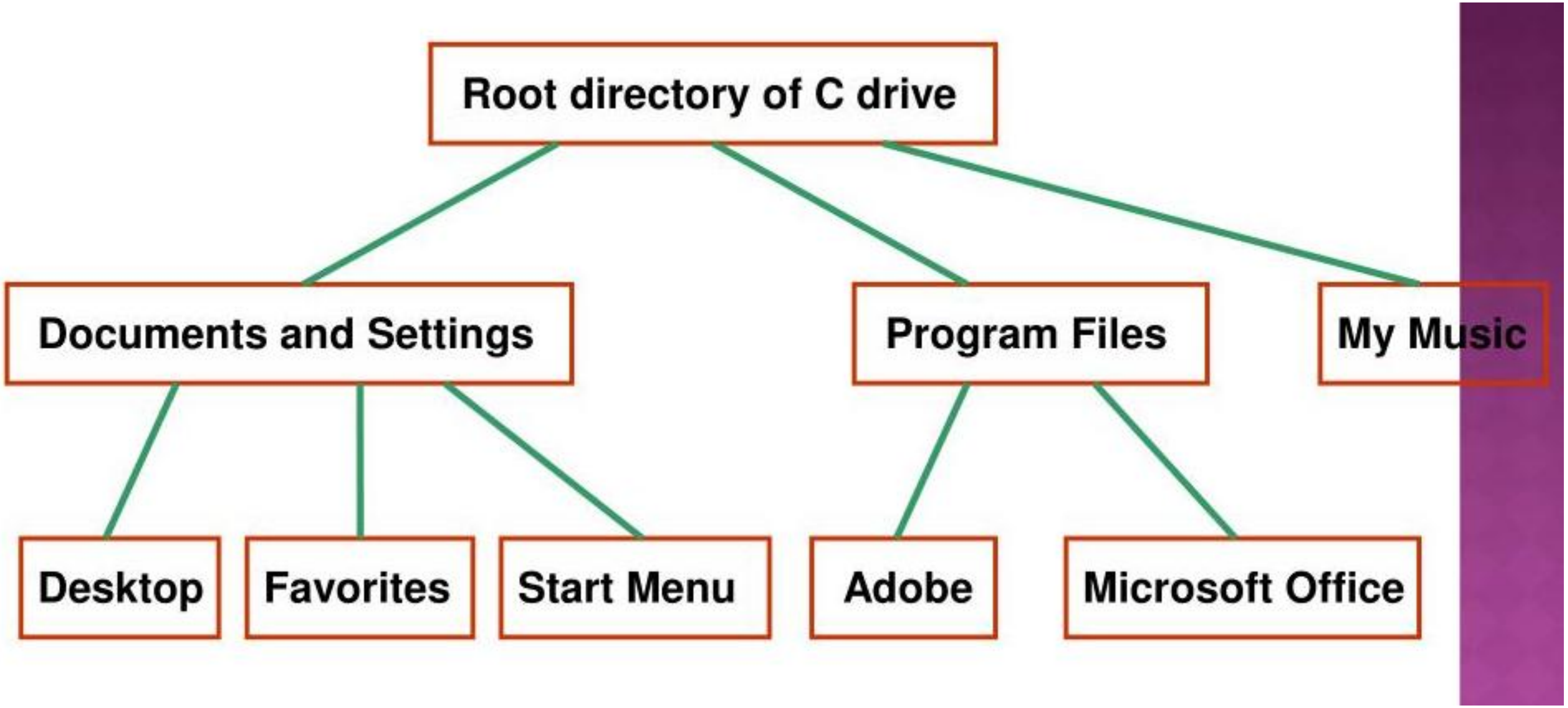
Tree

- A tree is a nonlinear data structure used to represent entities that are in some hierarchical relationship
- Examples in real life:
  - Family tree
  - Table of contents of a book
  - Class inheritance hierarchy in Java
  - Computer file system (folders and subfolders)
  - Decision trees
  - Top-Down Design

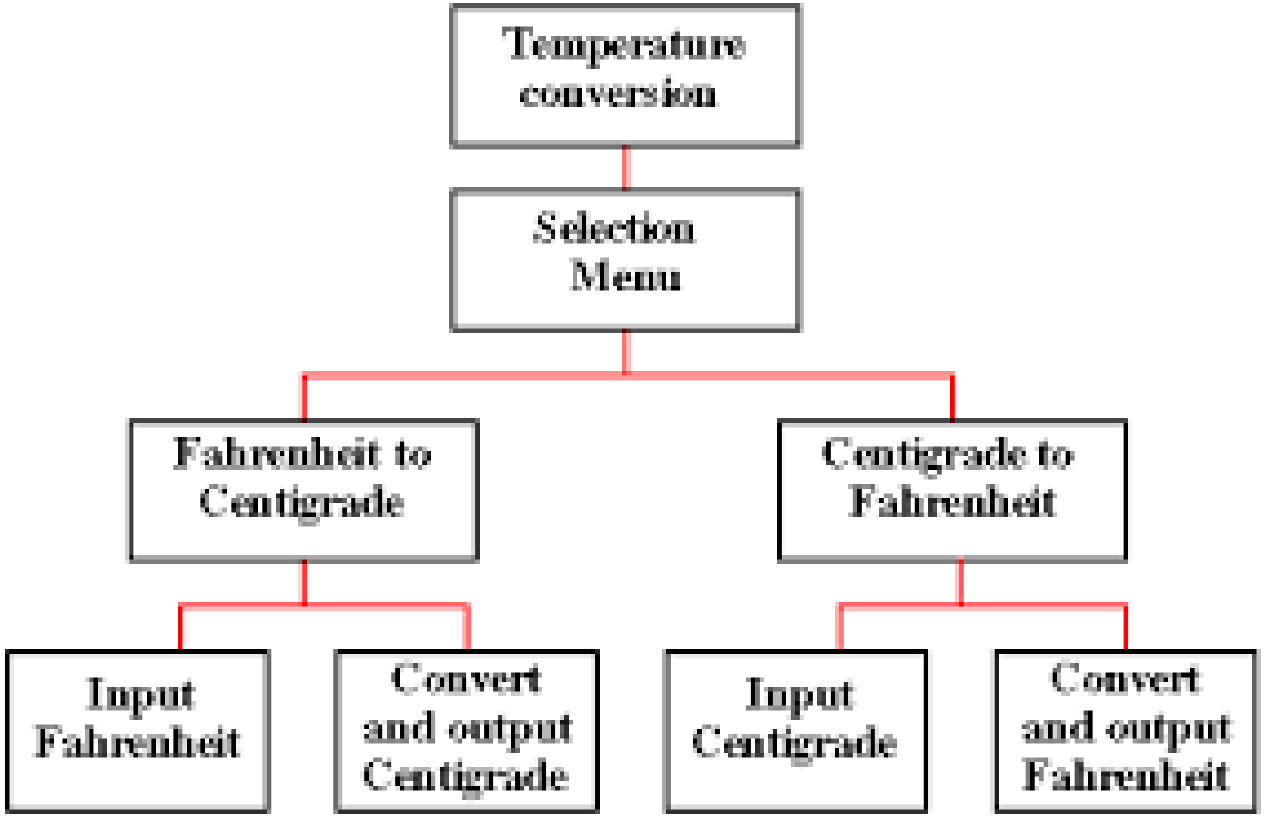
Family Tree


Examples.
. Family tree.
Charles
dad
mom
Philip
Elizabeth II
Andrew
Alice
George VI
Elizabeth I
George I
Olga
Louis
Victoria
George V
Mary
Claude
Celia

Computer File Systems



Top-Down Design

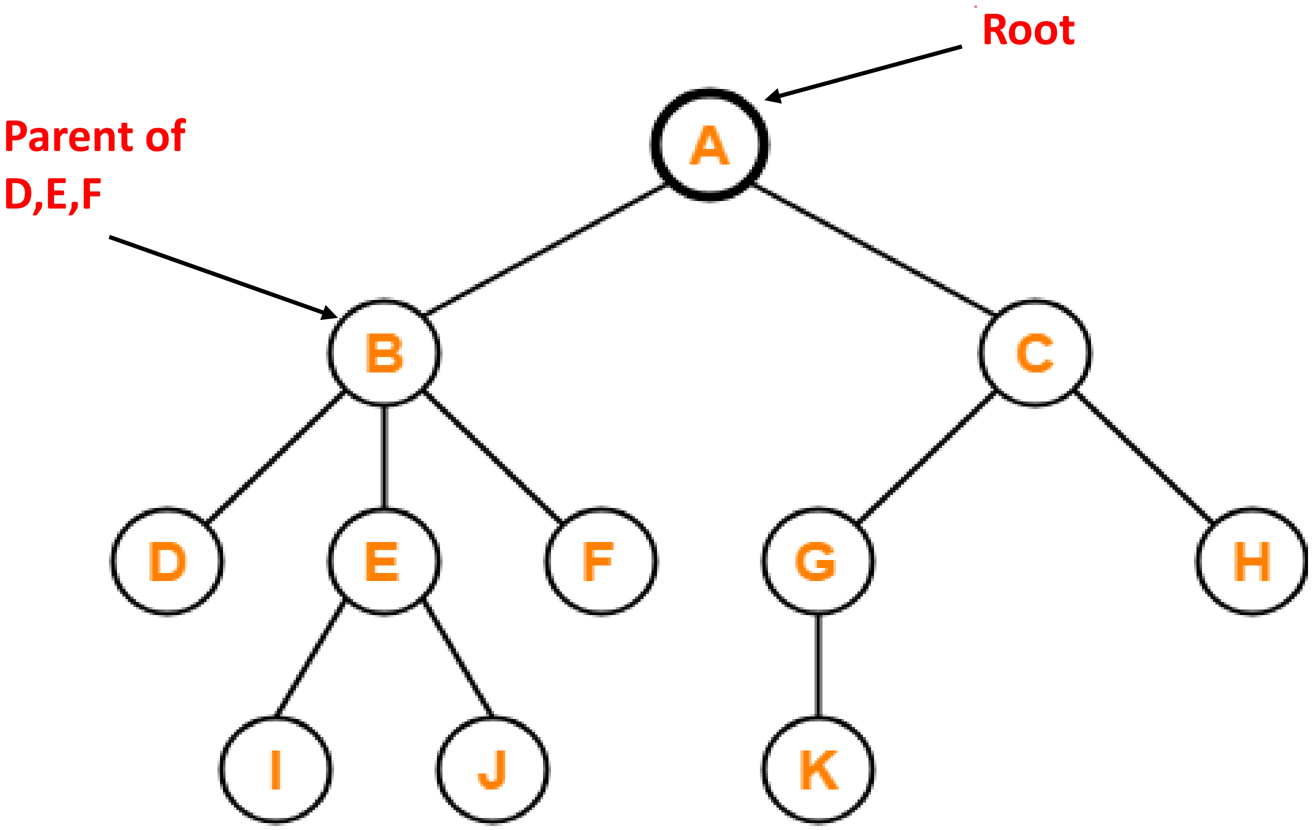


Tree-Definition

- A tree is a hierarchical data structure defined as a collection of nodes and edges. Nodes represent value and nodes are connected by edges.
- A tree has the following properties:
  - The tree has one node called root. The tree originates from this, and hence it does not have any parent.
  - Each node has one parent only but can have multiple children.
  - Each node is connected to its children via edge.

Tree Terminology

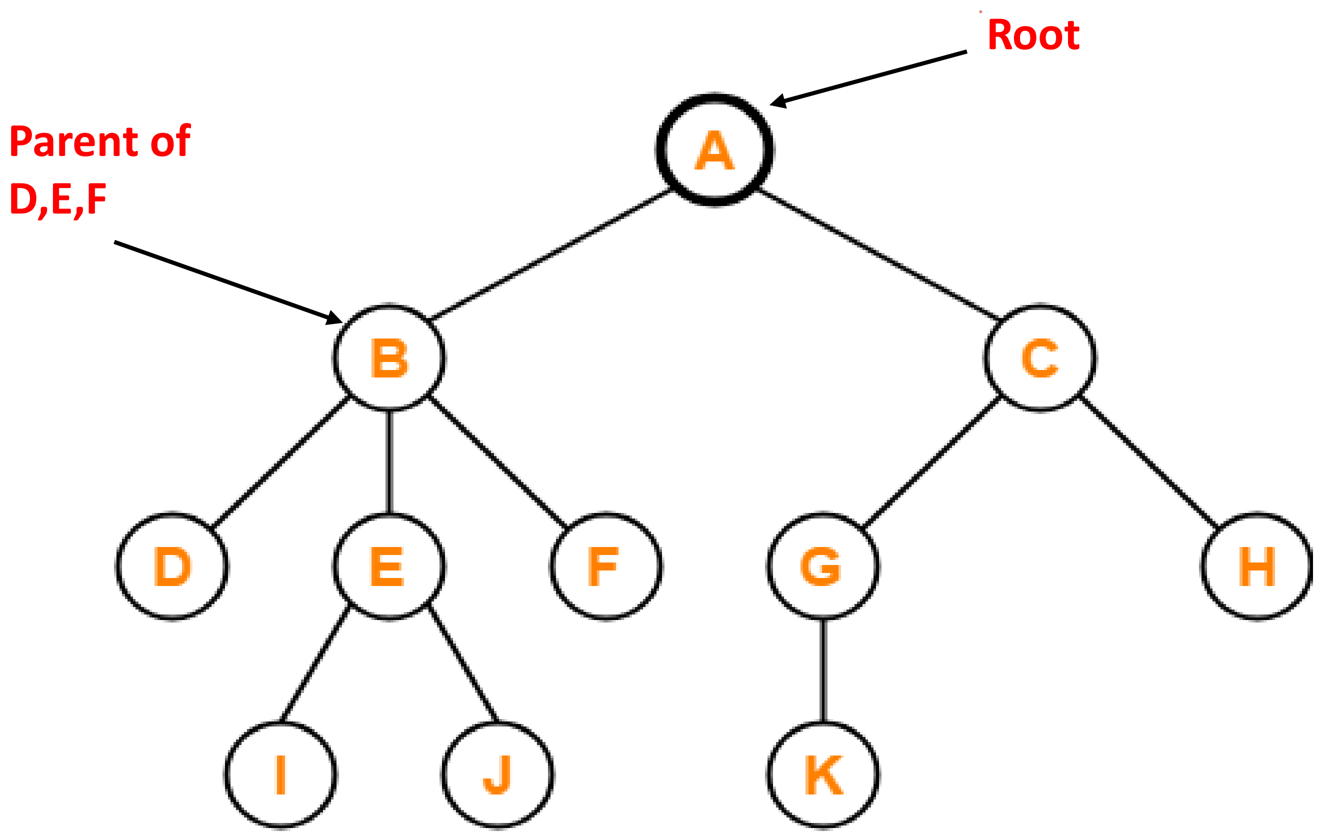
Nodes
Edges
Root
Parent or predecessor
Leaf node
Interior node
Child or successor
Siblings
Ancestors of a node
Descendants of a node
Path/Traversing
Height of Node
Depth of Node
Height of Tree
Levels of Node
Degree of Node
Subtree



Nodes: Individual Elements in Tree

Tree Terminology

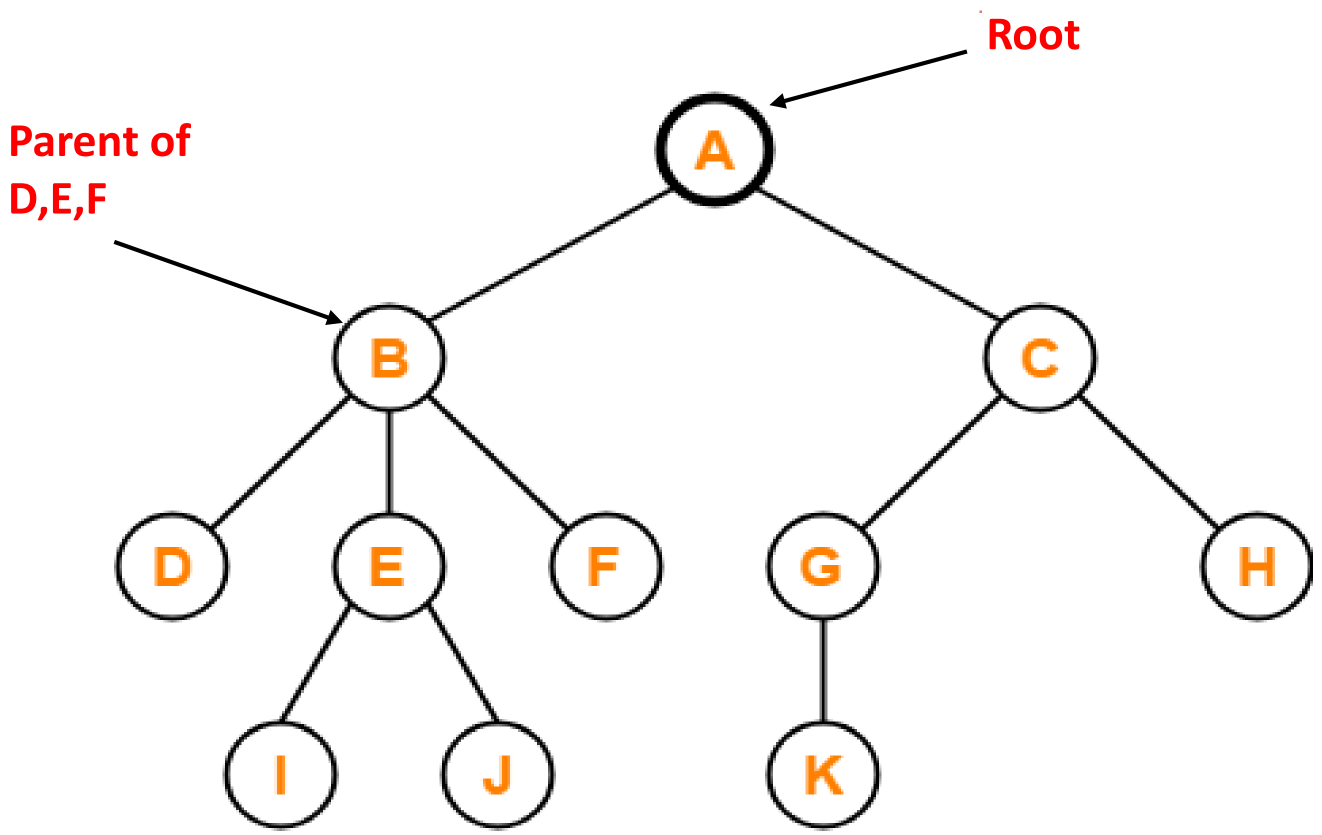
Nodes
Edges
Root
Parent or predecessor
Leaf node
Interior node
Child or successor
Siblings
Ancestors of a node
Descendants of a node
Path/Traversing
Height of Node
Depth of Node
Height of Tree
Levels of Node
Degree of Node
Subtree



The connecting link between any two nodes is called as an **edge**.
In a tree with n number of nodes, there are exactly (n-1) number of edges.

Tree Terminology

- Nodes
- Edges
- Root
- Parent or predecessor
- Leaf node
- Interior node
- Child or successor
- Siblings
- Ancestors of a node
- Descendants of a node
- Path/Traversing
- Height of Node
- Depth of Node
- Height of Tree
- Levels of Node
- Degree of Node
- Subtree

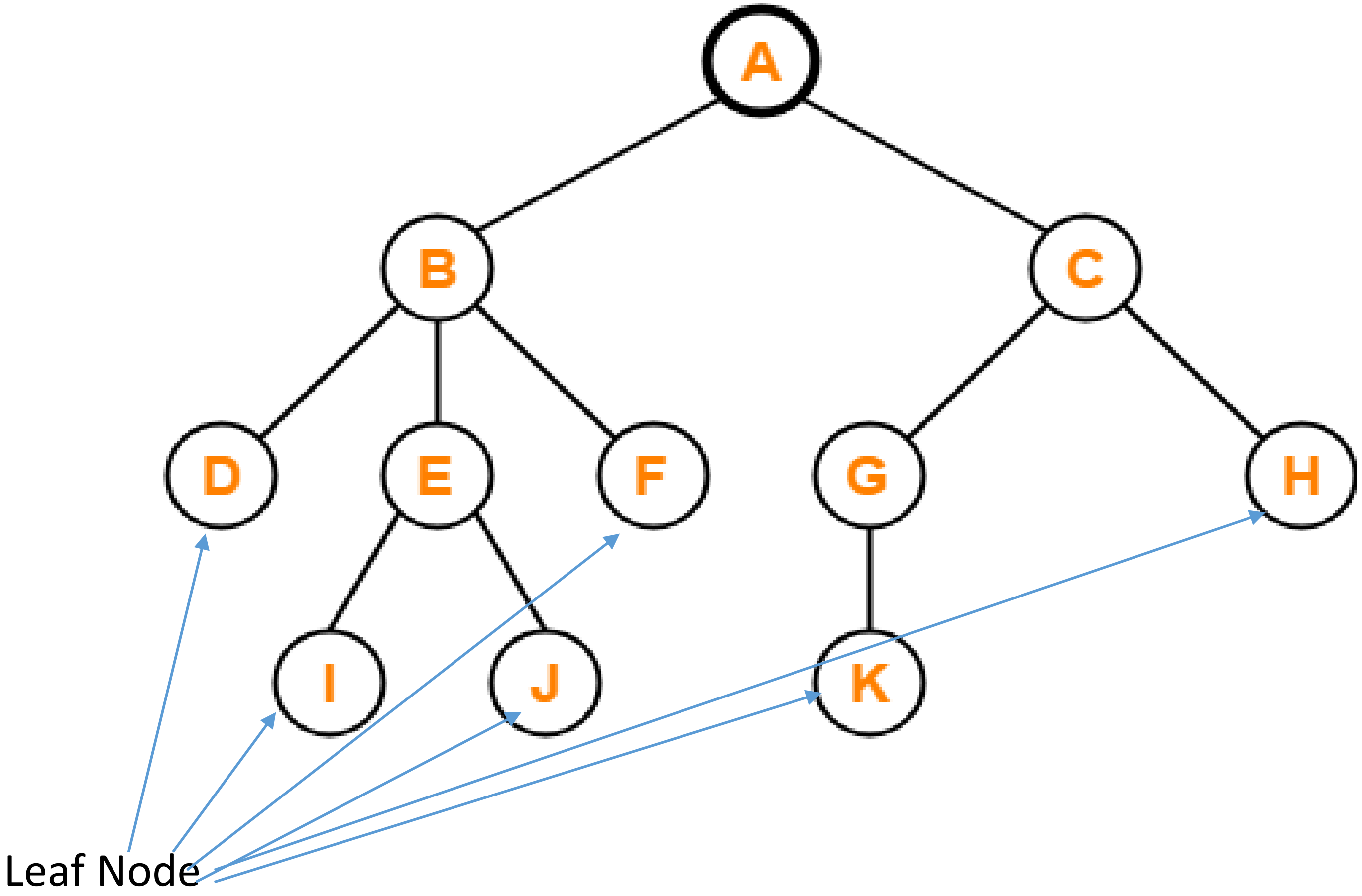


The first node from where the tree originates is called as a **root node**.
In any tree, there must be only one root node.
We can never have multiple root nodes in a tree data structure.

The node which has a branch from it to any other node is called as a **parent node**.
In other words, the node which has one or more children is called as a parent node.
In a tree, a parent node can have any number of child nodes.

Tree Terminology

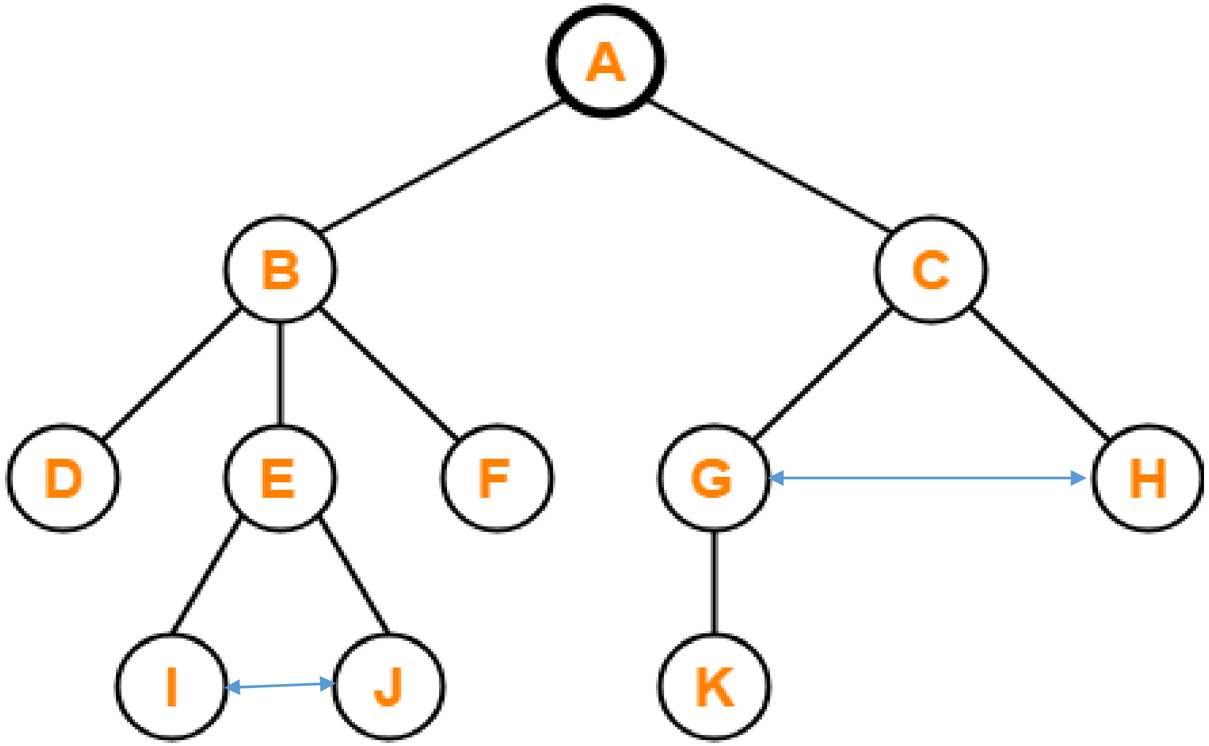
Nodes
Edges
Root
Parent or predecessor
Leaf node
Interior node
Child or successor
Siblings
Ancestors of a node
Descendants of a node
Path/Traversing
Height of Node
Depth of Node
Height of Tree
Levels of Node
Degree of Node
Subtree



The node which does not have any child is called as a **leaf node**.
Leaf nodes are also called as **external nodes** or **terminal nodes**.

Tree Terminology

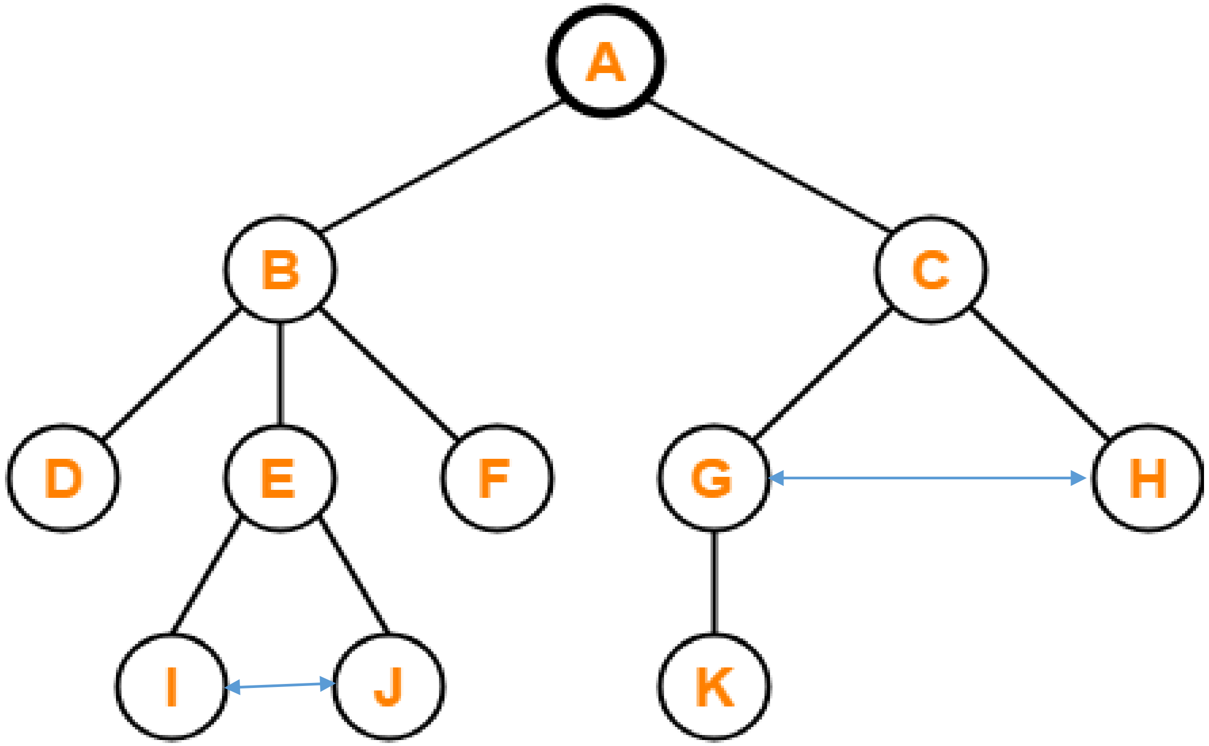
Nodes
Edges
Root
Parent or predecessor
Leaf node
Interior node
Child or successor
Siblings
Ancestors of a node
Descendants of a node
Path/Traversing
Height of Node
Depth of Node
Height of Tree
Levels of Node
Degree of Node
Subtree



The node which has at least one child is called as an **internal node**.
Internal nodes are also called as **non-terminal nodes**.
Every non-leaf node is an internal node.

Tree Terminology

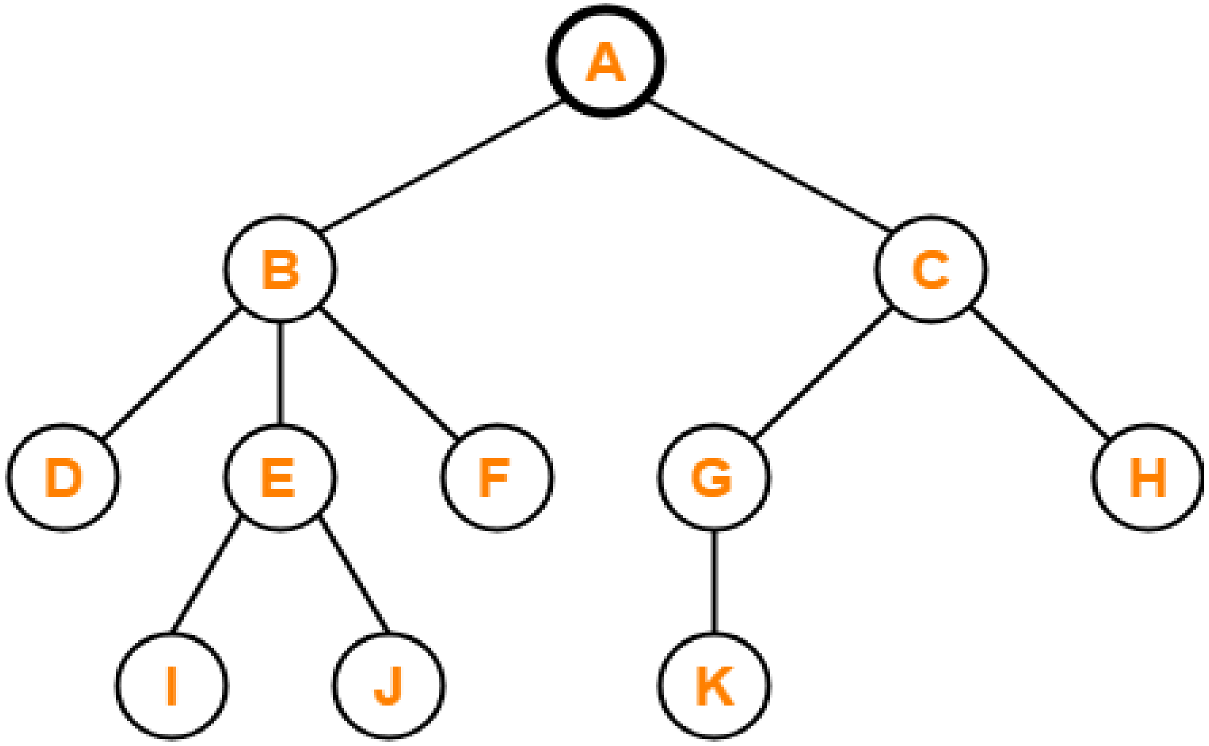
Nodes
Edges
Root
Parent or predecessor
Leaf node
Interior node
Child or successor
Siblings
Ancestors of a node
Descendants of a node
Path/Traversing
Height of Node
Depth of Node
Height of Tree
Levels of Node
Degree of Node
Subtree



Sibling: The nodes that have the same parent are known as siblings.

Tree Terminology

Nodes
Edges
Root
Parent or predecessor
Leaf node
Interior node
Child or successor
Siblings
Ancestors of a node
Descendants of a node
Path/Traversing
Height of Node
Depth of Node
Height of Tree
Levels of Node
Degree of Node
Subtree

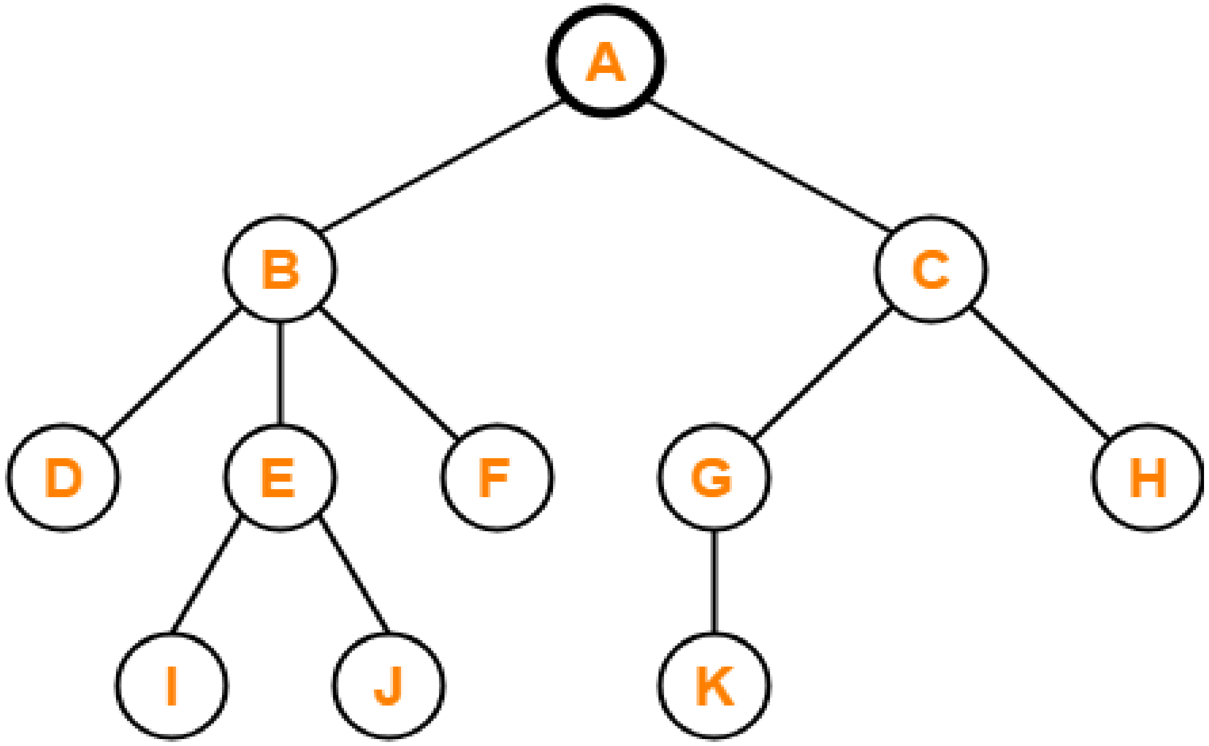


Ancestor node:- An ancestor of a node is any predecessor node on a path from the root to that node. The root node doesn't have any ancestors.

In the tree shown in the above image, nodes A, B, and E are the ancestors of node J.

Tree Terminology

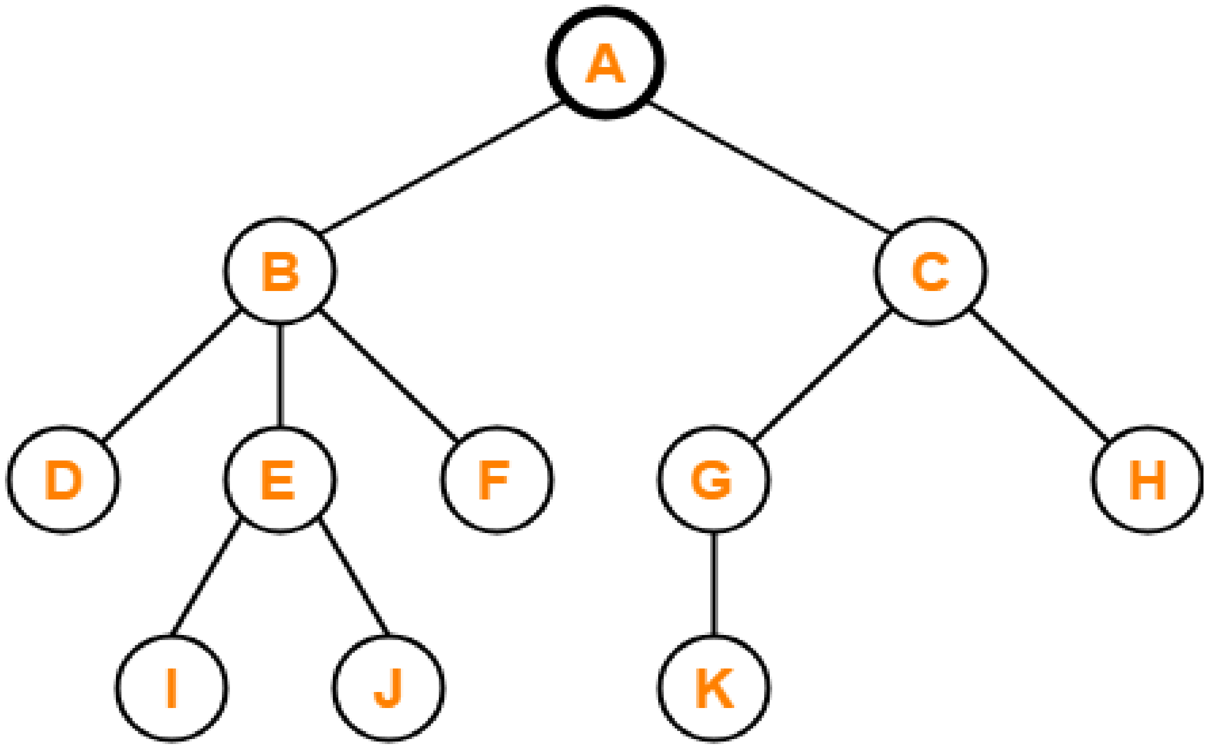
- Nodes
- Edges
- Root
- Parent or predecessor
- Leaf node
- Interior node
- Child or successor
- Siblings
- Ancestors of a node
- Descendants of a node
- Path/Traversing
- Height of Node
- Depth of Node
- Height of Tree
- Levels of Node
- Degree of Node
- Subtree



Descendant: An descendent of a node is any successor node on a path from the node to the leaf.
The leaf node doesn't have any descendants
In the tree shown in the above image, nodes C,G and K are the descendants of node A.

Tree Terminology

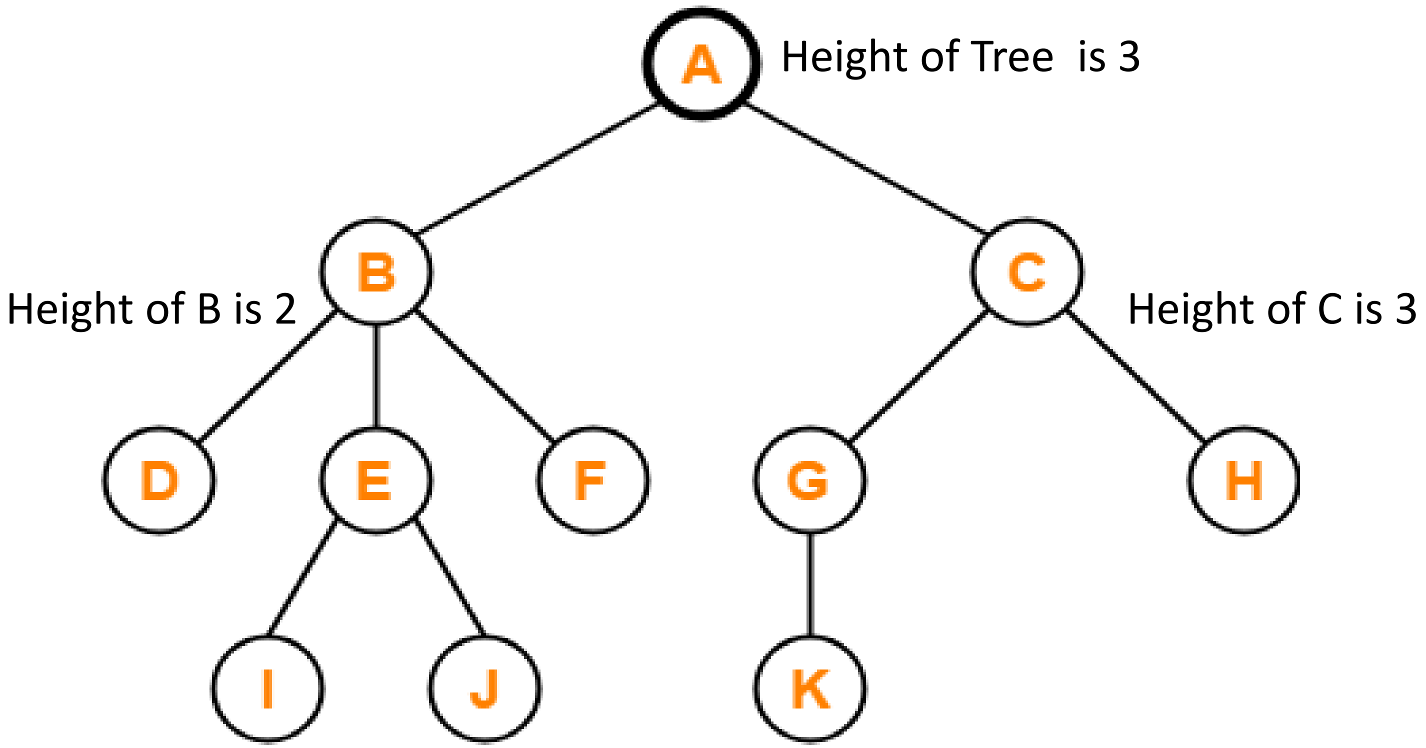
Nodes
Edges
Root
Parent or predecessor
Leaf node
Interior node
Child or successor
Siblings
Ancestors of a node
Descendants of a node
Path/Traversing
Height of Node
Height of Tree
Depth of Node
Levels of Node
Degree of Node
Subtree



Path is a number of successive edges from source node to destination node.

Tree Terminology

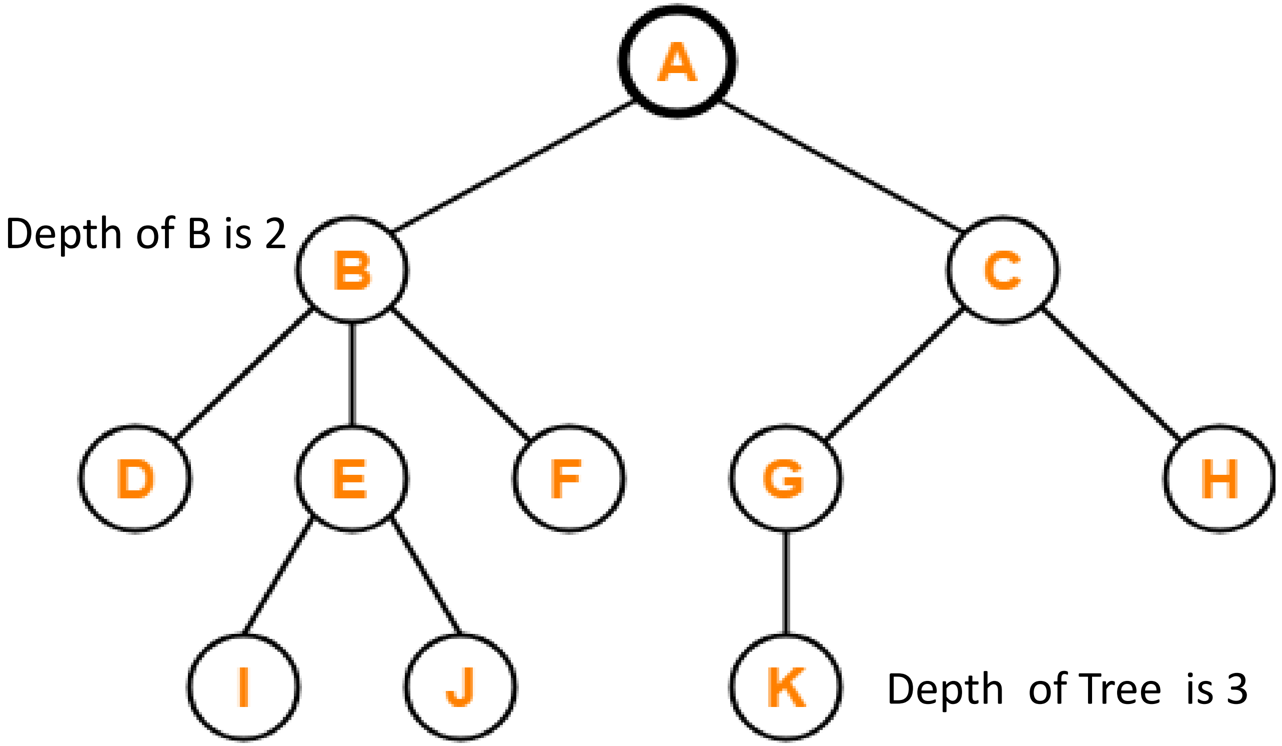
Nodes
Edges
Root
Parent or predecessor
Leaf node
Interior node
Child or successor
Siblings
Ancestors of a node
Descendants of a node
Path/Traversing
Height of Node
Height of Tree
Depth of Node
Levels of Node
Degree of Node
Subtree



Height of a node represents the number of edges on the longest path between that node and a leaf.
Height of a root node is **Height of a Tree**

Tree Terminology

- Nodes
- Edges
- Root
- Parent or predecessor
- Leaf node
- Interior node
- Child or successor
- Siblings
- Ancestors of a node
- Descendants of a node
- Path/Traversing
- Height of Node
- Height of Tree
- Depth of Node
- Levels of Node
- Degree of Node
- Subtree



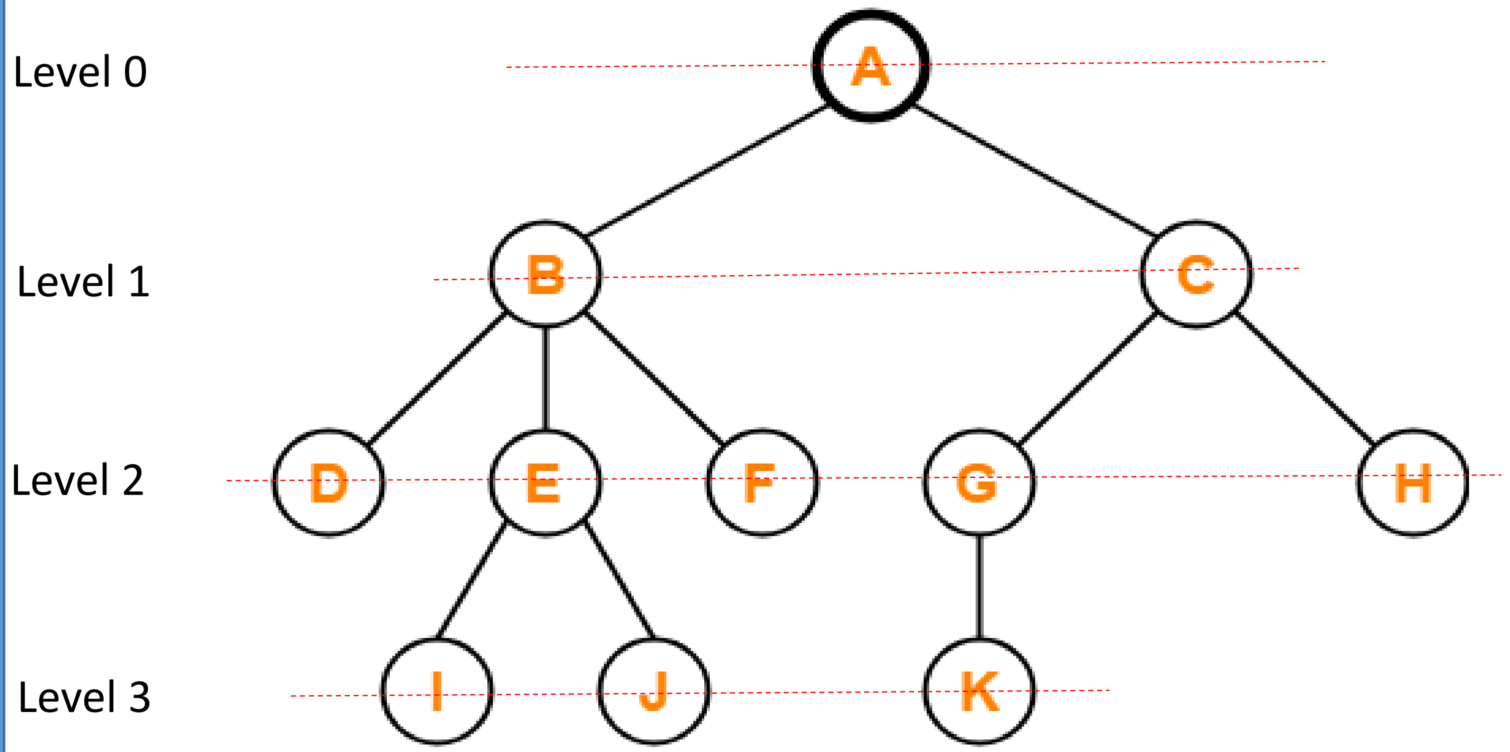
Total number of edges from root node to a particular node is called as **depth of that node**.

Depth of a tree is the total number of edges from root node to a leaf node in the longest path.

Depth of the root node = 0

Tree Terminology

Nodes
Edges
Root
Parent or predecessor
Leaf node
Interior node
Child or successor
Siblings
Ancestors of a node
Descendants of a node
Path/Traversing
Height of Node
Height of Tree
Depth of Node
Levels of Node
Degree of Node
Subtree

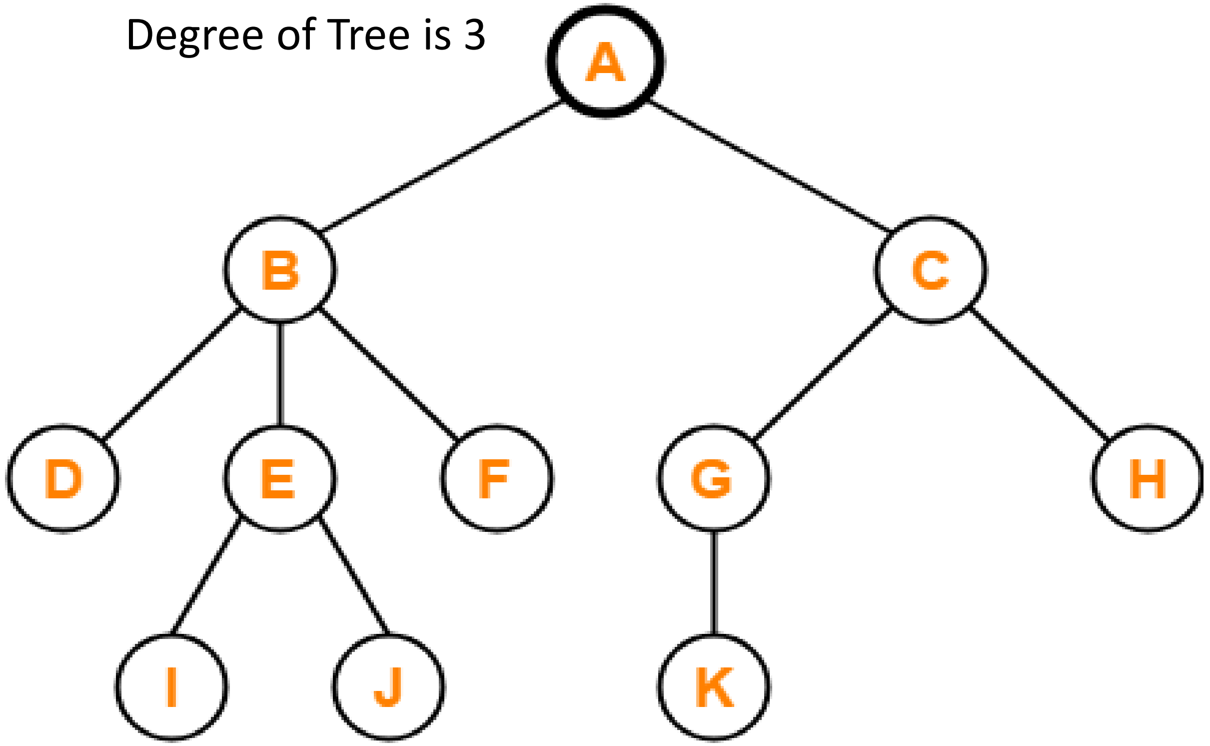


In a tree, each step from top to bottom is called as **level of a tree**. The level count starts with 0 and increments by 1 at each level or step.

Tree Terminology

Nodes
Edges
Root
Parent or predecessor
Leaf node
Interior node
Child or successor
Siblings
Ancestors of a node
Descendants of a node
Path/Traversing
Height of Node
Height of Tree
Depth of Node
Levels of Node
Degree of Node
Subtree

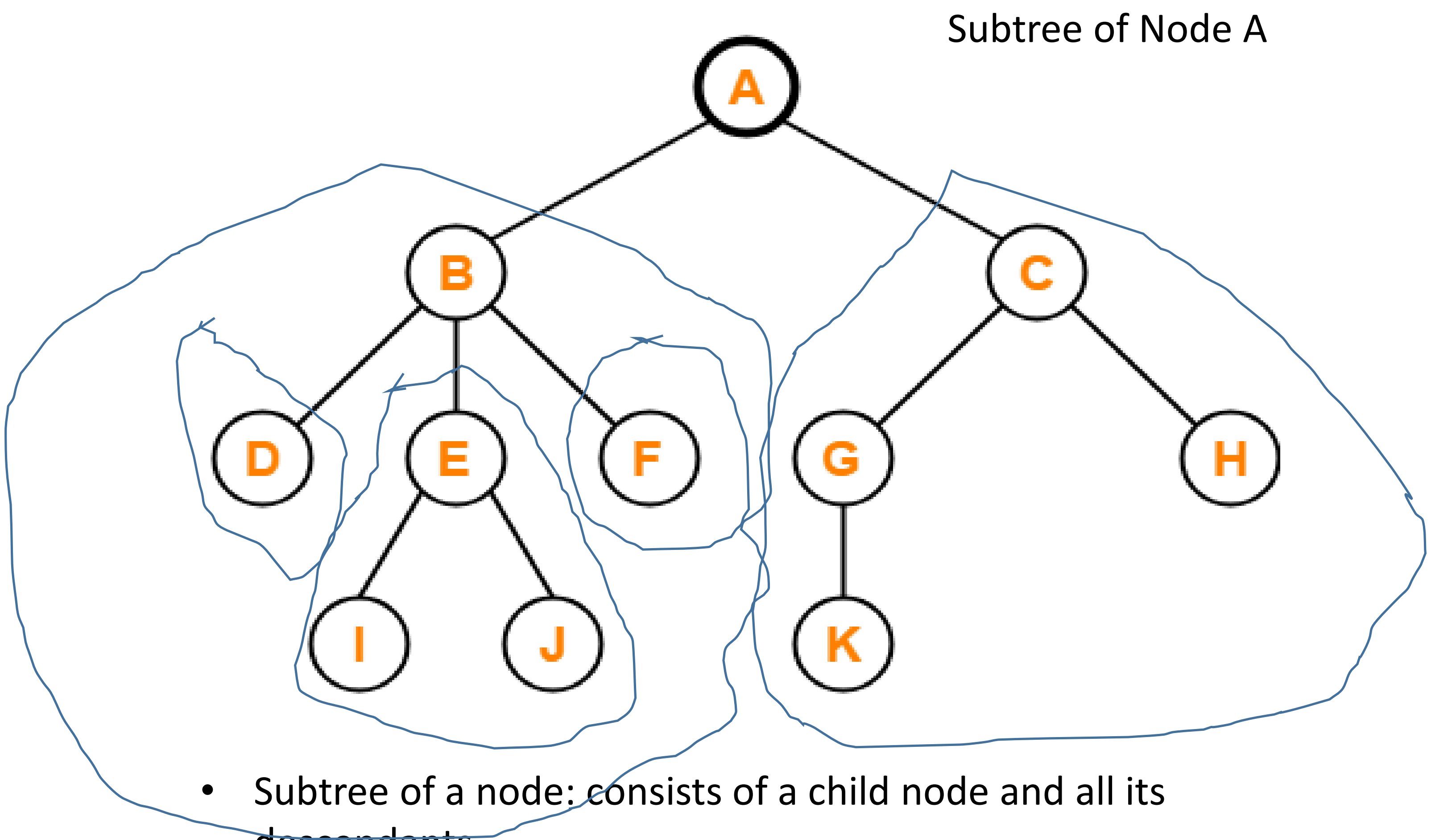
Degree of Tree is 3



Degree of a node is the total number of children of that node.
Degree of a tree is the highest degree of a node among all the nodes in the tree.

Tree Terminology

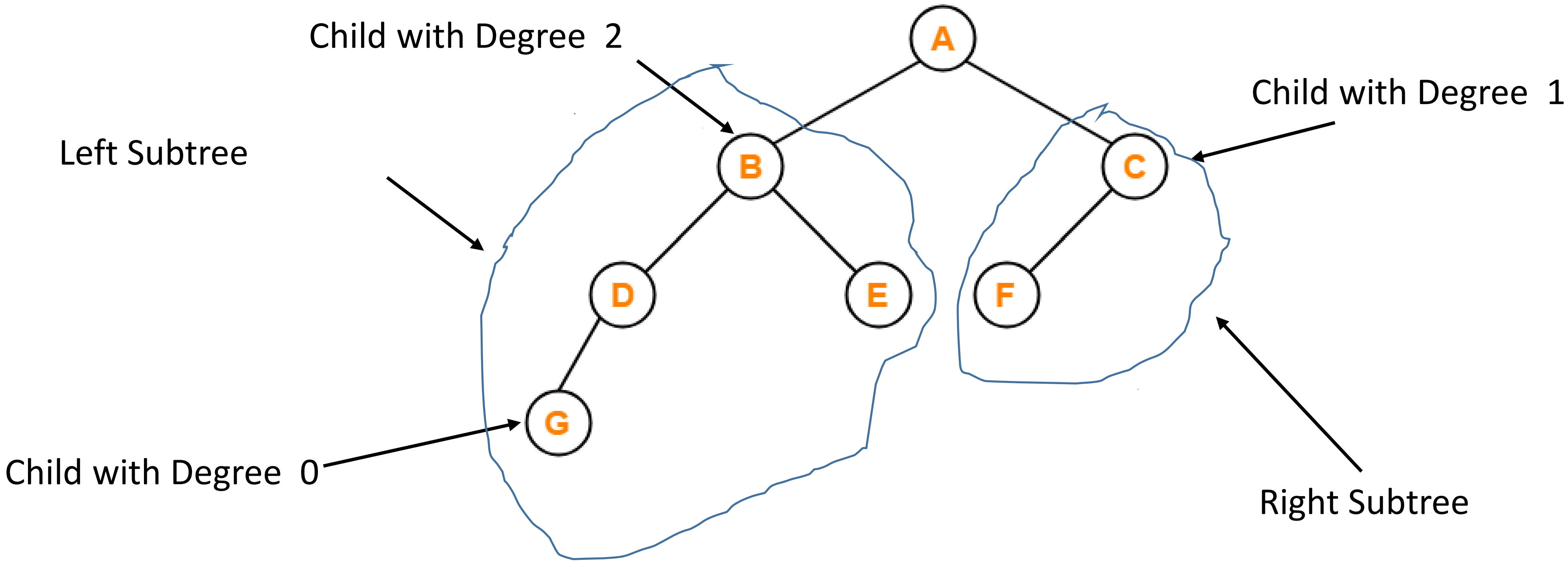
Nodes
Edges
Root
Parent or predecessor
Leaf node
Interior node
Child or successor
Siblings
Ancestors of a node
Descendants of a node
Path/Traversing
Height of Node
Height of Tree
Depth of Node
Levels of Node
Degree of Node
Subtree



- Subtree of a node: consists of a child node and all its descendants.
  - A subtree is itself a tree
  - A node may have many subtrees

Binary Tree

- Binary tree is a special tree data structure in which each node can have at most 2 children.
- Thus, in a binary tree,
  - Each node has either 0 child or 1 child or 2 children.
  - i.e Binary tree is tree with degree 2
  - The children(if present) are called left child and right child.

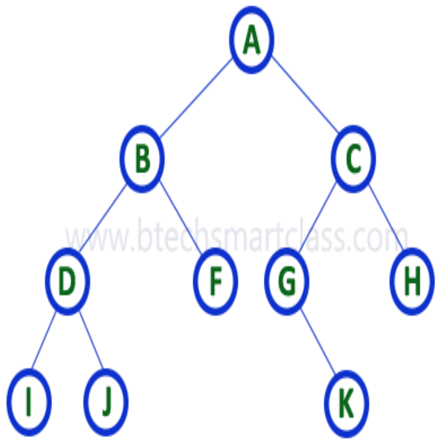


Representation of Binary Tree

- Array Representation
- Linked Representation

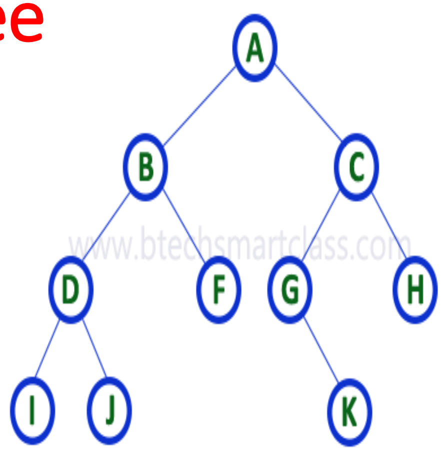
Representation of Binary Tree

- Array Representation
  - Node is at index i
  - Left Child at 2*i+1
  - Right Child at 2*i+2
  - Parent at floor(i-1/2)



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Representation of Binary Tree

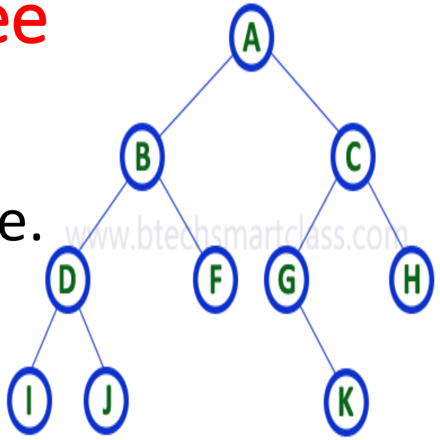


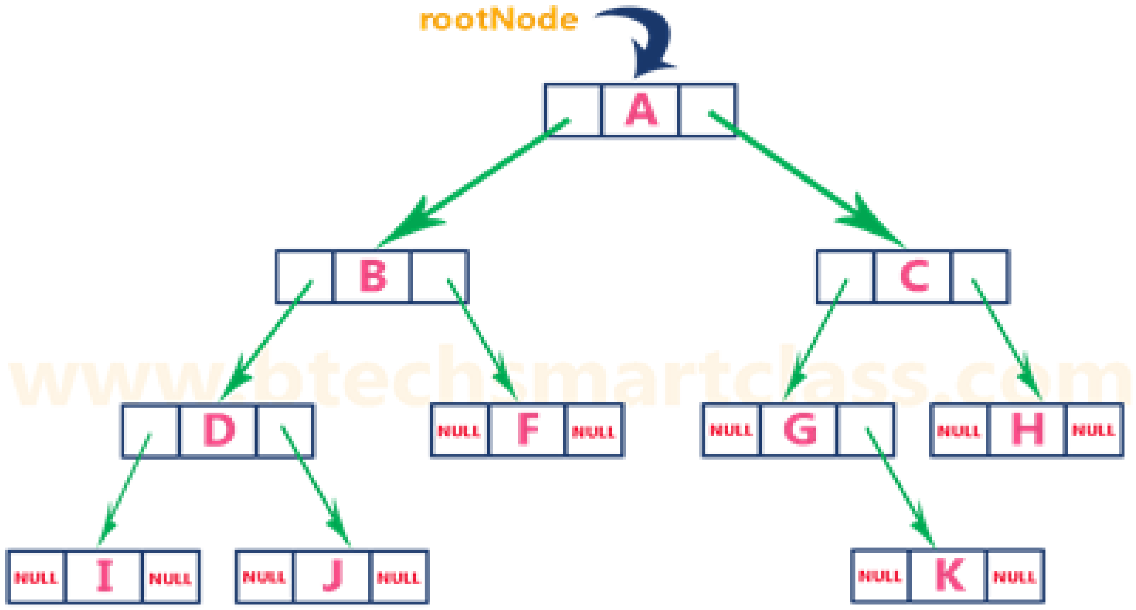
- Array Representation
  - Node is at index i
  - Left Child at 2*i+1
  - Right Child at 2*i+2
  - Parent at floor(i-1/2)
- This approach is good, and easily we can find the index of parent and child, but it is not memory efficient. It will occupy many spaces that has no use.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	B	C	D	F	G	H	I	J	--	--	--	K			

Representation of Binary Tree

- Linked Representation
  - We use a double linked list to represent a binary tree.



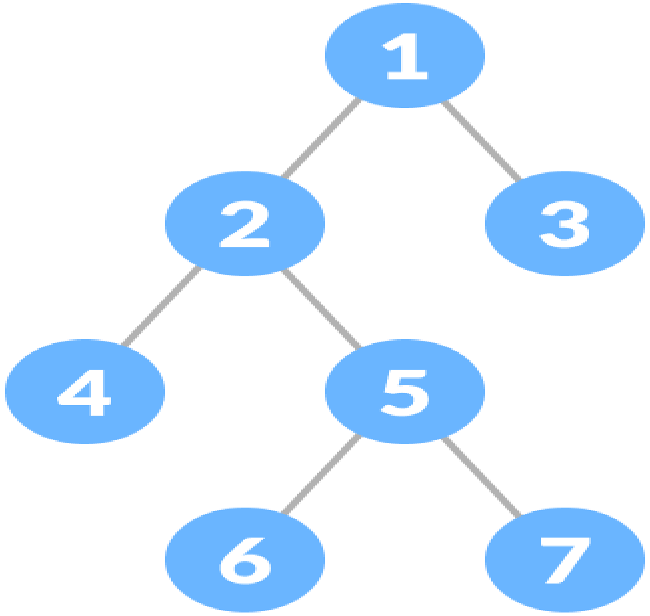


Types of Binary Tree

- **The following are types of Binary tree:**
  - **Full/ proper/ strict Binary tree**
  - **Complete Binary tree**
  - **Perfect Binary tree**
  - **Skewed Binary tree**
  - **Balanced Binary tree**

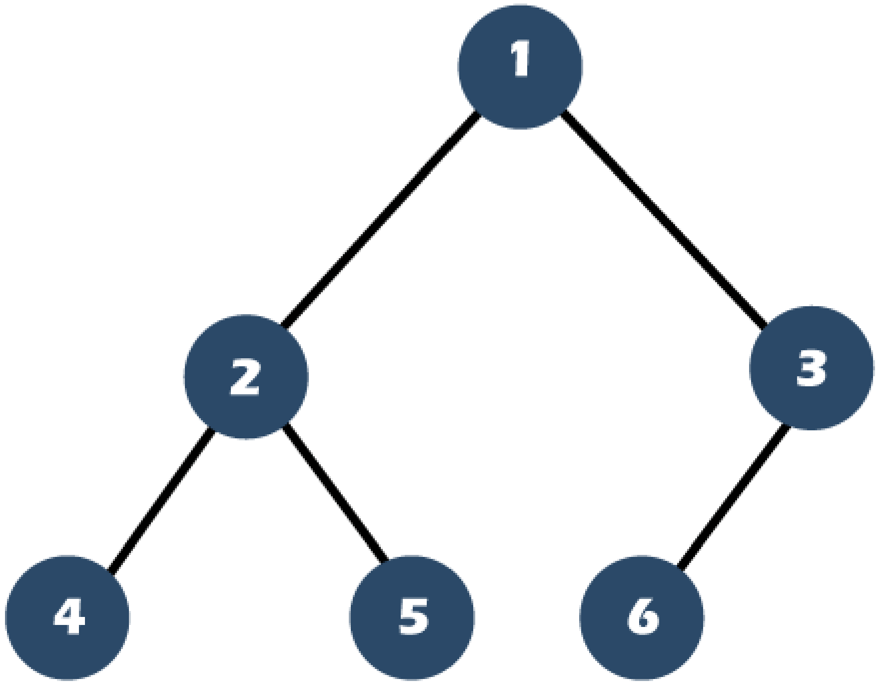
Full Binary Tree

- The full binary tree is a binary tree in which all the nodes have two children except the leaf nodes.



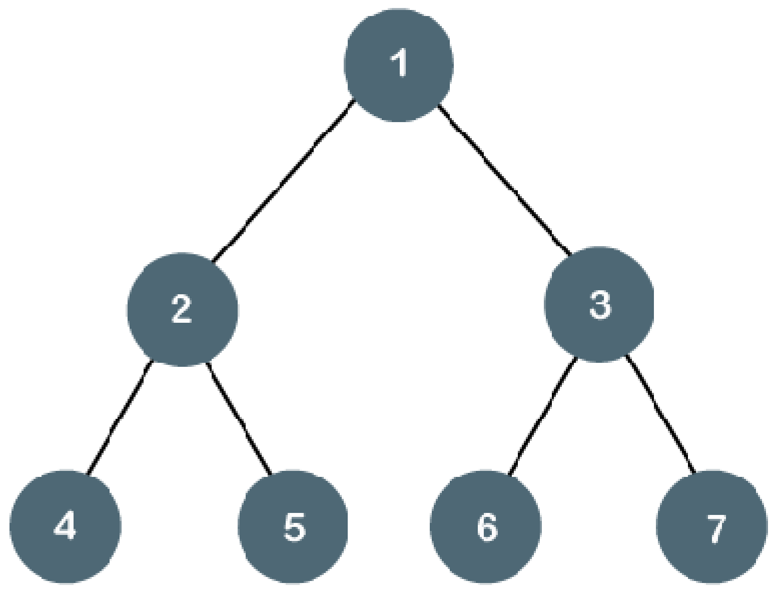
Complete Binary Tree

- A binary tree is said to be a complete binary tree when all the levels are completely filled except the last level, which is filled from the left.



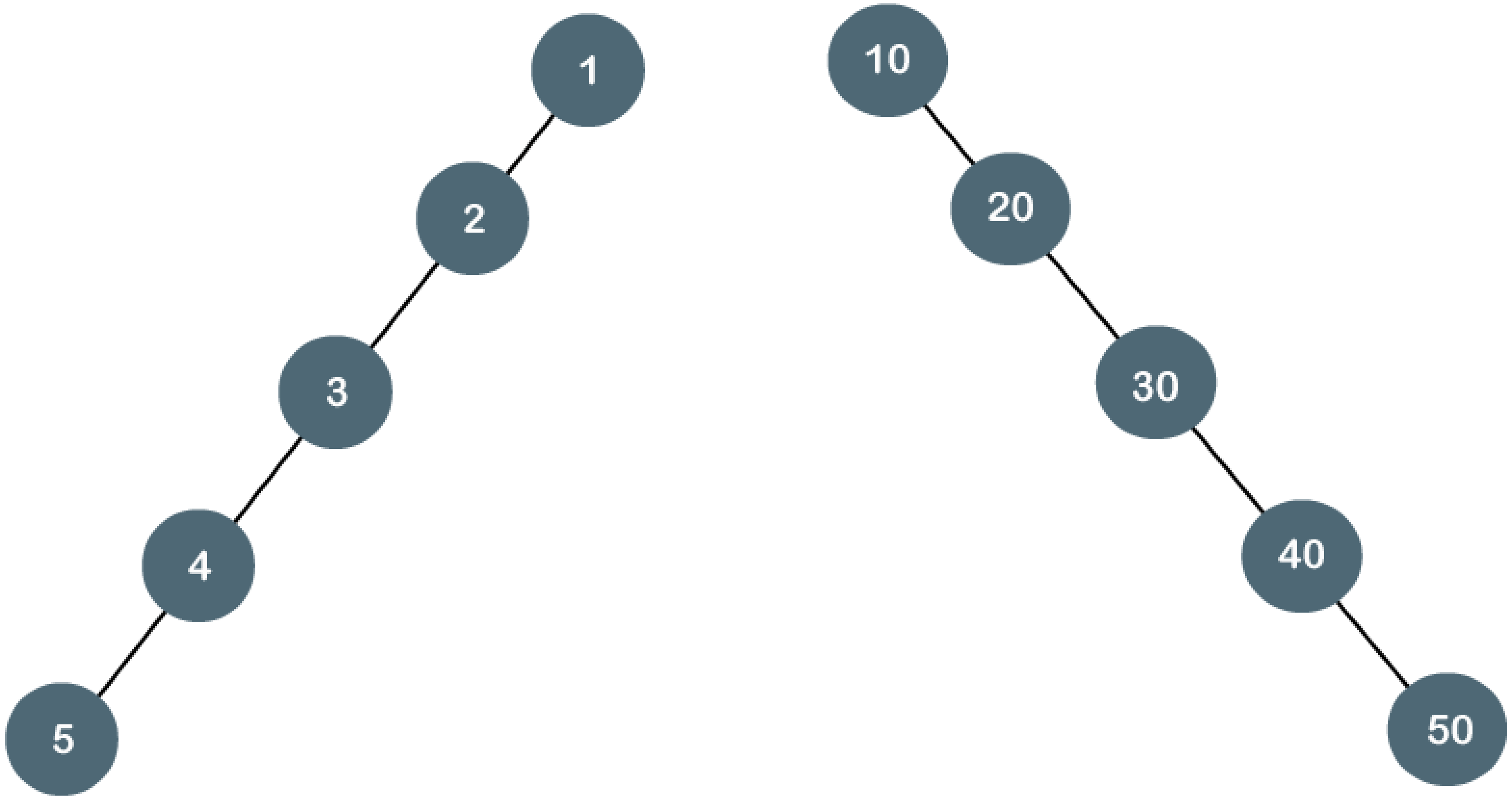
Perfect Binary Tree

- 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.



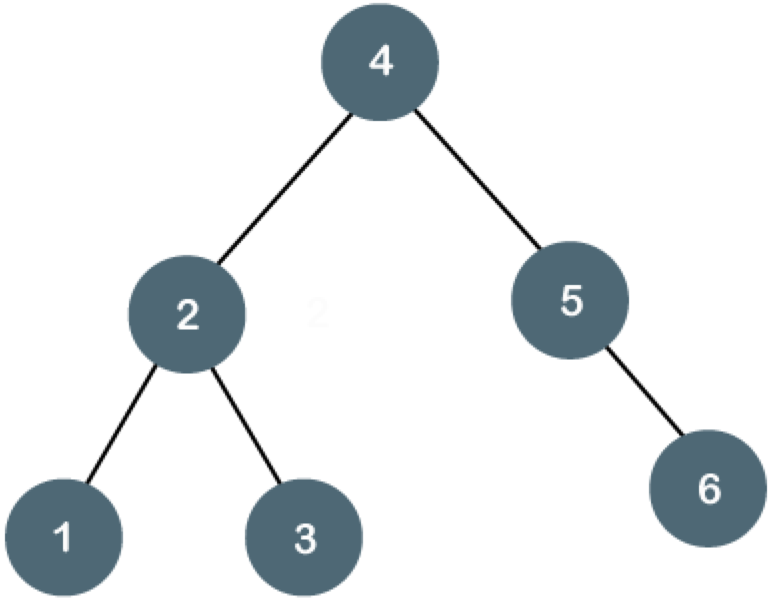
Skewed Binary Tree

- The degenerate binary tree is a tree in which all the internal nodes have only one children.



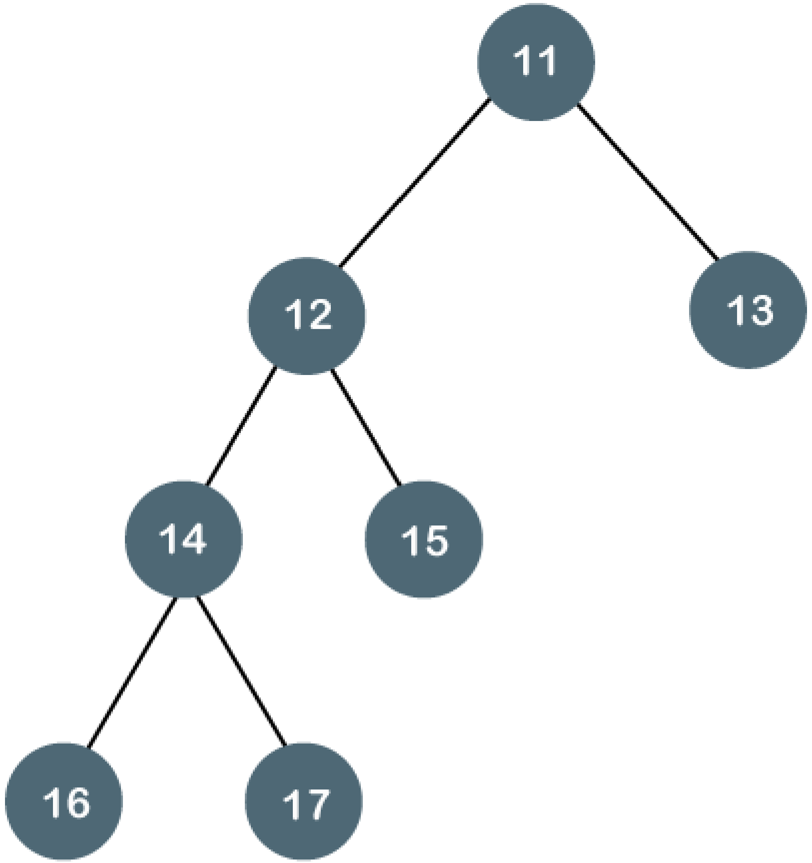
Balanced Binary Tree

- 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.



Balanced Binary Tree

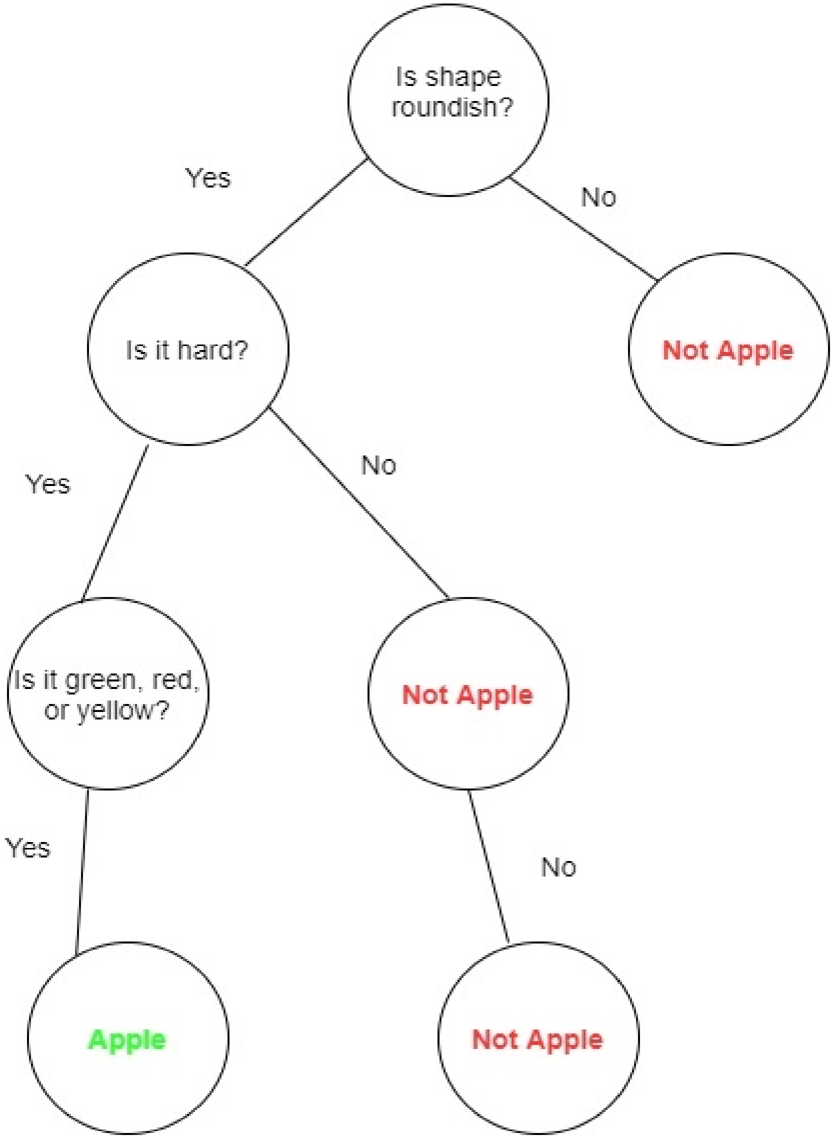
- The balanced binary tree is a tree in which both the left and right trees differ by atmost 1.
- For example, ***AVL*** and ***Red-Black trees*** are balanced binary tree.



Applications of Trees

- Routing Tables
- Decision Trees
- Expression Evaluation
- Sorting
- Database Indexing
- Data Compression(Huffman Coding)

Decision Tree

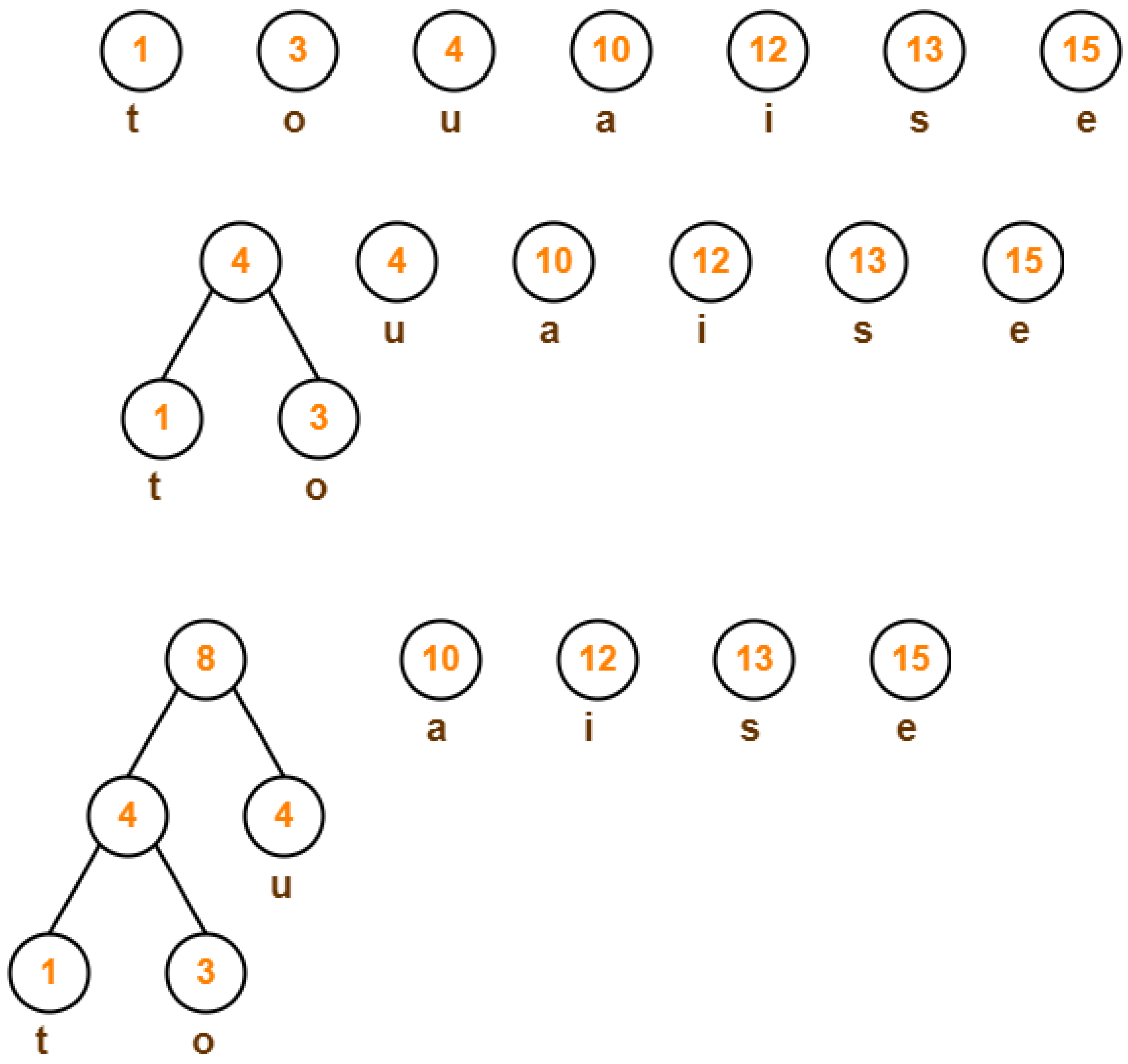


Syntax Tree

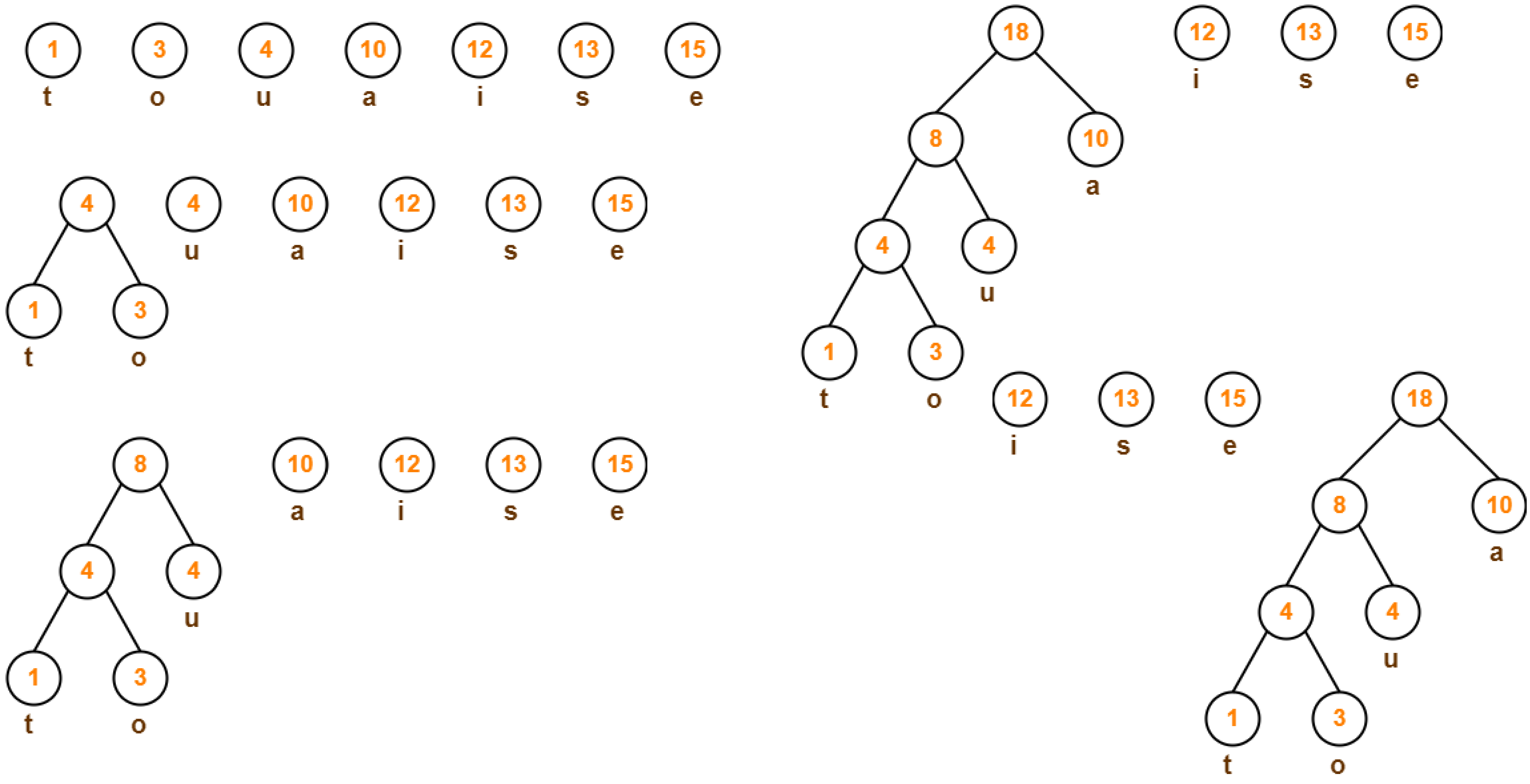
Sr.No.	Infix Notation	Prefix Notation	Postfix Notation
1	a + b	+ a b	a b +
2	(a + b) ∗ c	∗ + a b c	a b + c ∗
3	a ∗ (b + c)	∗ a + b c	a b c + ∗
4	a / b + c / d	+ / a b / c d	a b / c d / +
5	(a + b) ∗ (c + d)	∗ + a b + c d	a b + c d + ∗
6	((a + b) ∗ c) - d	- ∗ + a b c d	a b + c ∗ d -

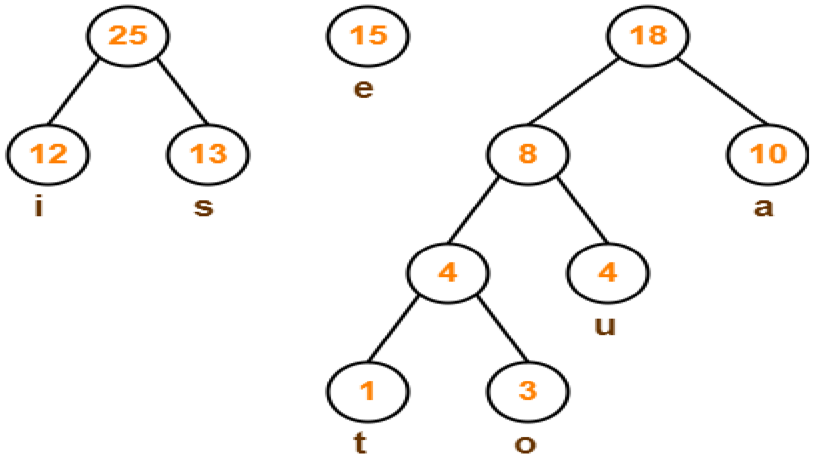
Huffman Coding

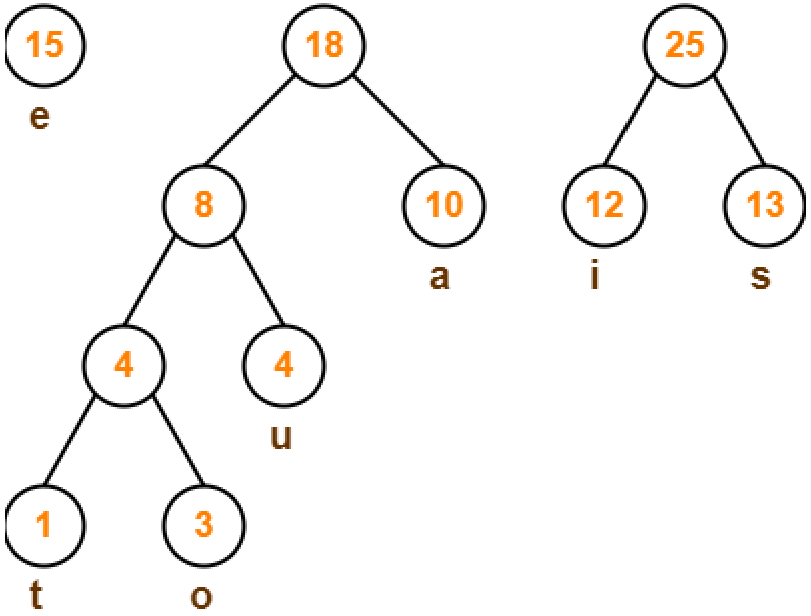
Characters	Frequencies
a	10
e	15
i	12
o	3
u	4
s	13
t	1

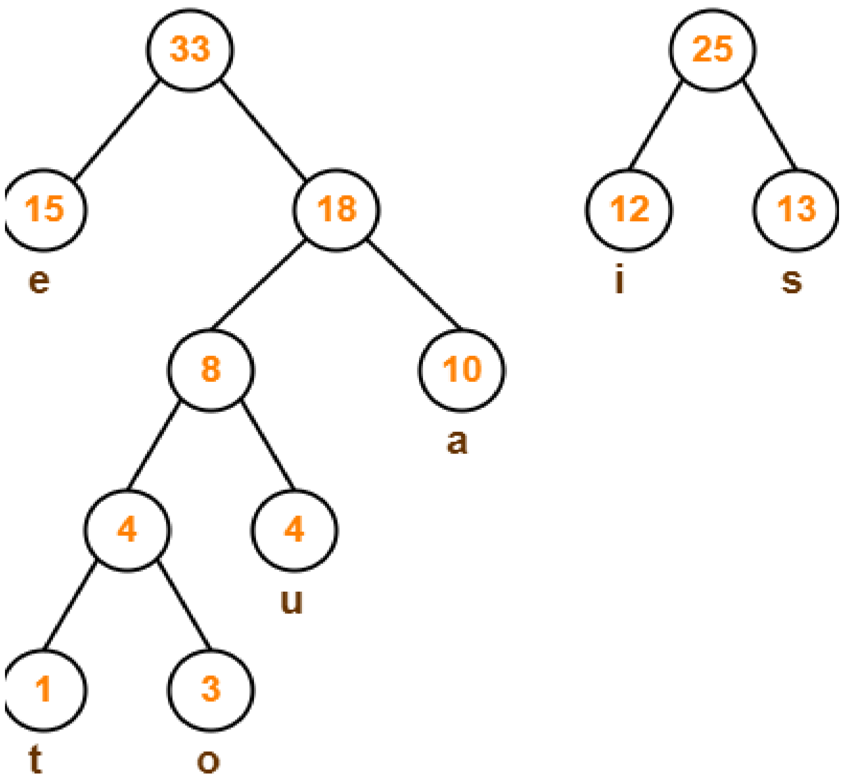


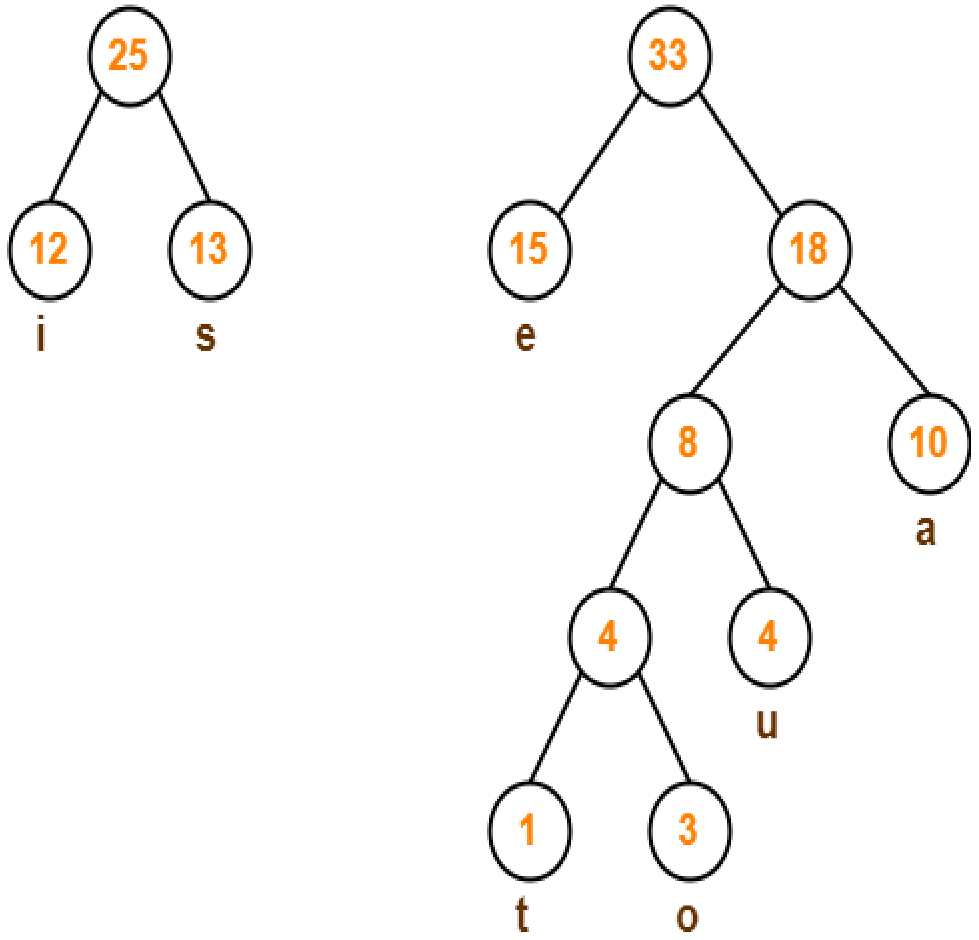
Huffman Coding

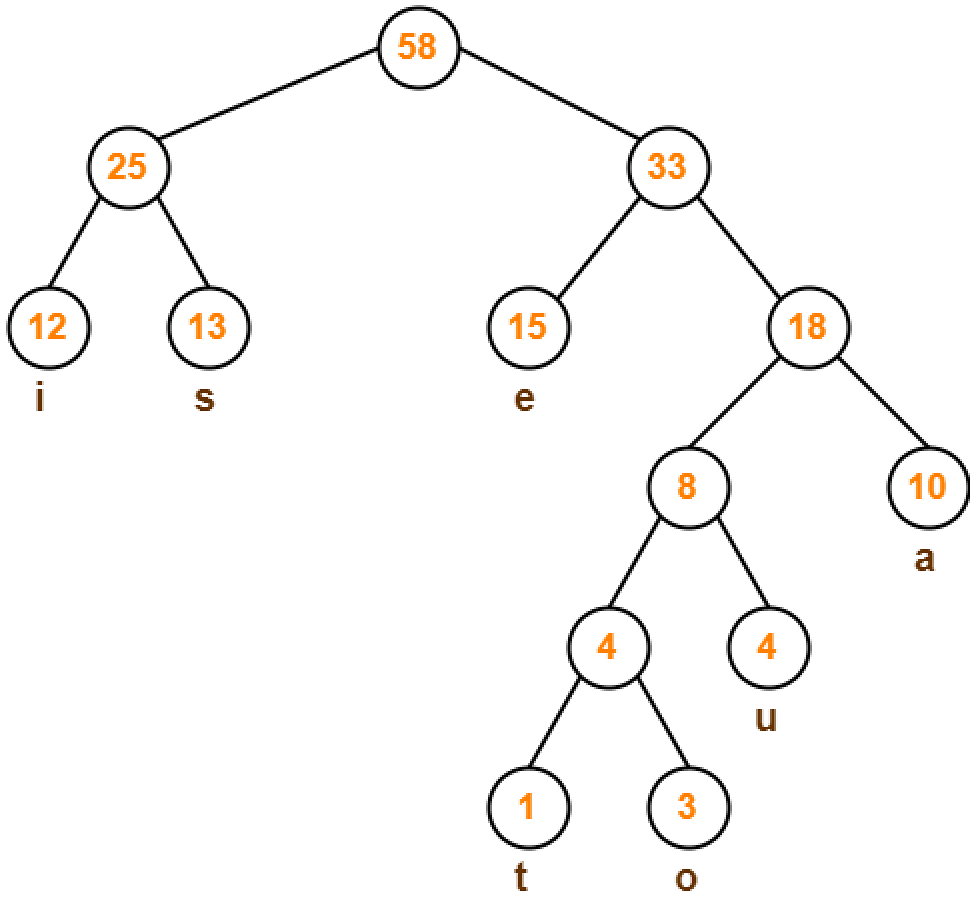



25
12
i
13
s
15
e
18
8
10
a
4
4
u
1
t
3
o

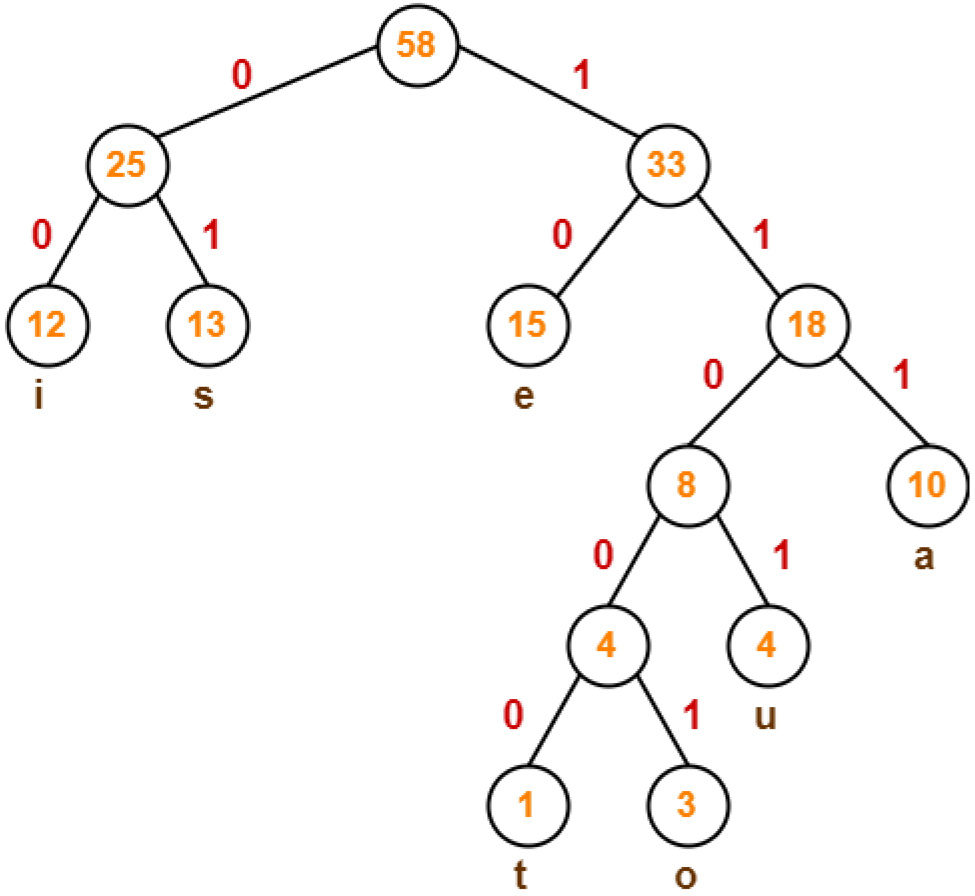

15
e
18
8
10
a
4
4
u
1
t
3
o
25
12
i
13
s


33
15
e
18
8
10
a
4
4
u
1
t
3
o
25
12
i
13
s


25
12
13
i
s
33
15
18
e
8
10
a
4
4
u
1
3
t
o


58
25
33
12
13
15
18
i
s
e
8
10
a
4
4
u
1
3
t
o

a	
e	
i	
o	
u	
s	
t	



Binary Tree Traversal

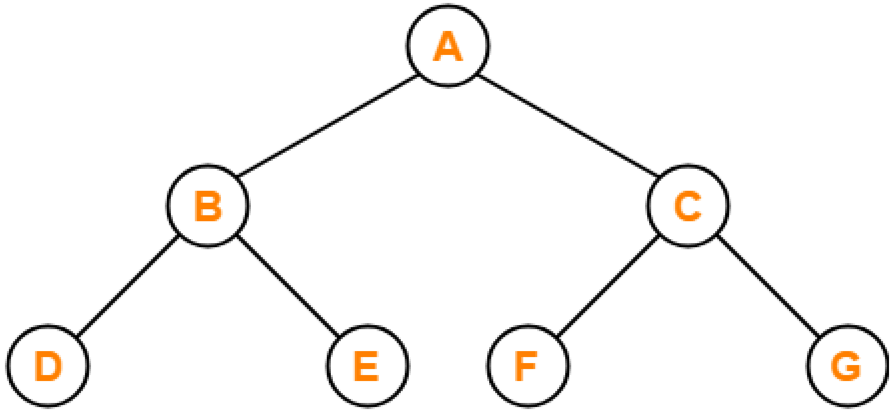
- **Displaying (or) visiting order of all nodes in a binary tree is called as Binary Tree Traversal.**
- **Tree Traversal Techniques**
  - **Depth First Traversal**
  - **Breadth First Traversal**

Binary Tree Traversal –Depth First Traversal

- **Displaying (or) visiting order of all nodes in a binary tree is called as Binary Tree Traversal.**
- **Tree Traversal Techniques**
  - **Depth First Traversal**
    - **Preorder Traversal**
    - **Inorder Traversal**
    - **Postorder Traversal**

Binary Tree Traversal –Preorder Traversal

- The preorder traversal of a nonempty binary tree is defined as follows:
  - Visit the root node
  - Traverse the left sub-tree in preorder
  - Traverse the right sub-tree in preorder
- Root → left → right
- A B D E C F G



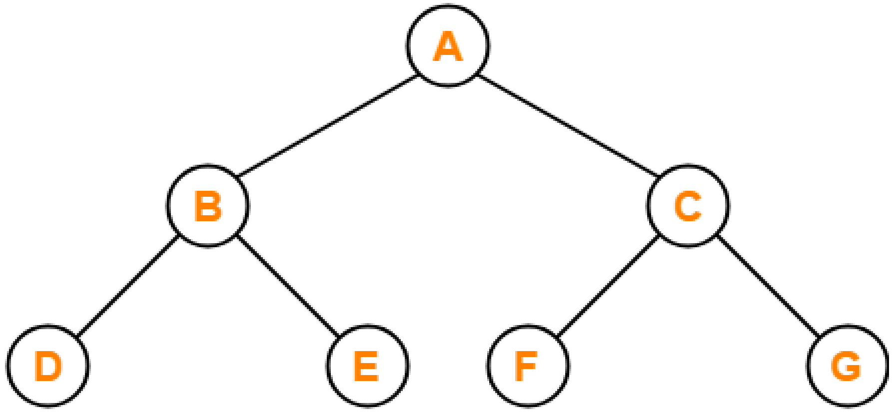
- Preorder traversal is used to get prefix expressic
- It is use to copy the Tree

Binary Tree Traversal –Preorder Traversal-Algorithm

- Step 1: Repeat Steps 2 to 4 while TREE != NULL
- Step 2: Write TREE DATA
- Step 3: PREORDER(TREE LEFT)
- Step 4: PREORDER(TREE RIGHT) [END OF LOOP]
- Step 5: END

Binary Tree Traversal –Inorder Traversal

- The in-order traversal of a nonempty binary tree is defined as follows:
  - Traverse the left sub-tree in in-order
  - Visit the root node
  - Traverse the right sub-tree in inorder
- Left → Root → right
- D B E A F C G



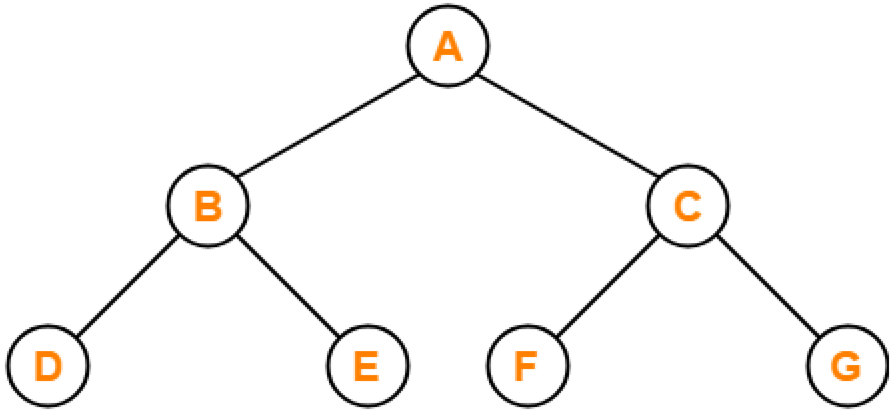
- Inorder traversal is used to get infix expression of an expression tree.

Binary Tree Traversal –Inorder Traversal-Algorithm

- Step 1: Repeat Steps 2 to 4 while TREE != NULL
- Step 2: INORDER(TREE LEFT)
- Step 3: Write TREE DATA
- Step 4: INORDER(TREE RIGHT) [END OF LOOP]
- Step 5: ENDtree.

Binary Tree Traversal –Postorder Traversal

- The Post-order traversal of a nonempty binary tree is defined as follows:
  - Traverse the left sub-tree in post-order
  - Traverse the right sub-tree in post-order
  - Visit the root node
- Left → Right → Root
- D E B F G C A



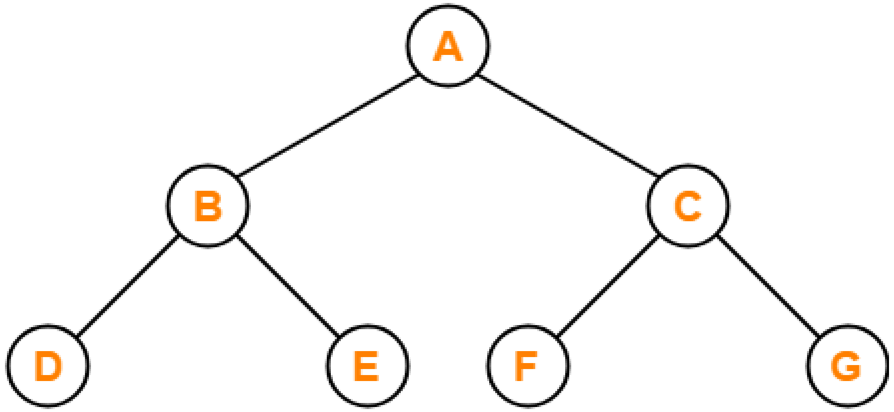
- Postorder traversal is used to get postfix expression of an expression tree.
- Postorder traversal is used to delete the tree.
- This is because it deletes the children first and then it deletes the parent.

Binary Tree Traversal –Postorder Traversal-Algorithm

- Step 1: Repeat Steps 2 to 4 while TREE != NULL
- Step 2: POSTORDER(TREE LEFT)
- Step 3: POSTORDER(TREE RIGHT)
- Step 4: Write TREE DATA [END OF LOOP]
- Step 5: END

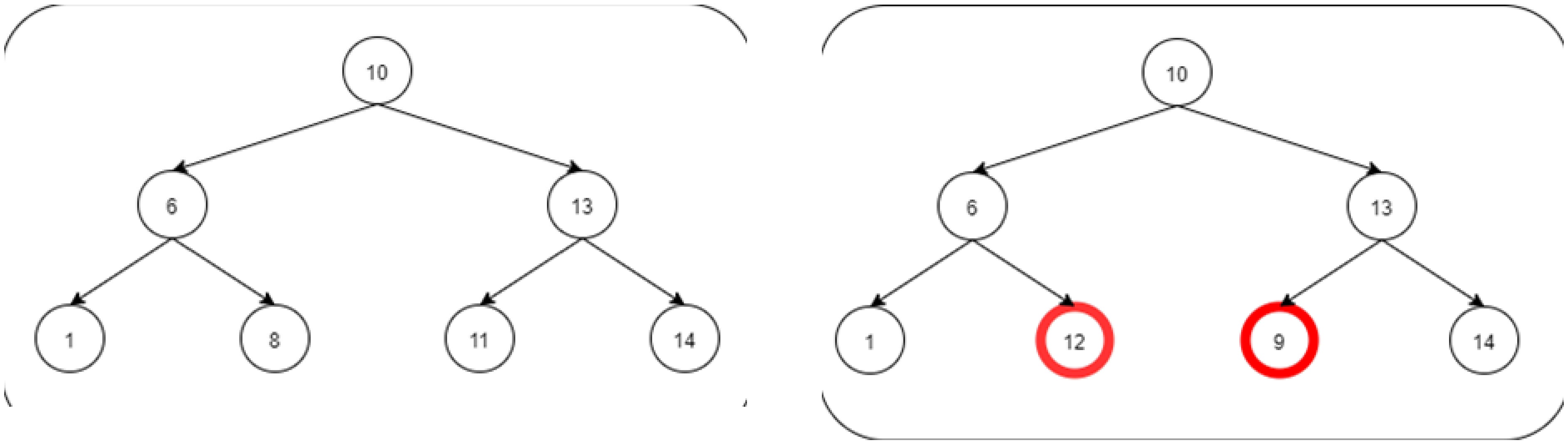
Binary Tree Traversal –Breadth First Traversal

- Breadth First Traversal of a tree prints all the nodes of a tree level by level.
- Breadth First Traversal is also called as **Level Order Traversal**.
- A B C D E F G
- Level order traversal is used to print the data in the same order as stored in the array representation of a complete binary tree.



Binary Search Tree

- Binary Search Tree is a special type of binary tree that has a specific order of elements in it. It follows three basic properties:-
  - All elements in the **left subtree** of a node should have a value **lesser than** the node's value.
  - All elements in the **right subtree** of a node should have a value **greater than** the node's value
  - Both the left and right subtrees should be binary search trees too.



Binary Search Tree-Construction

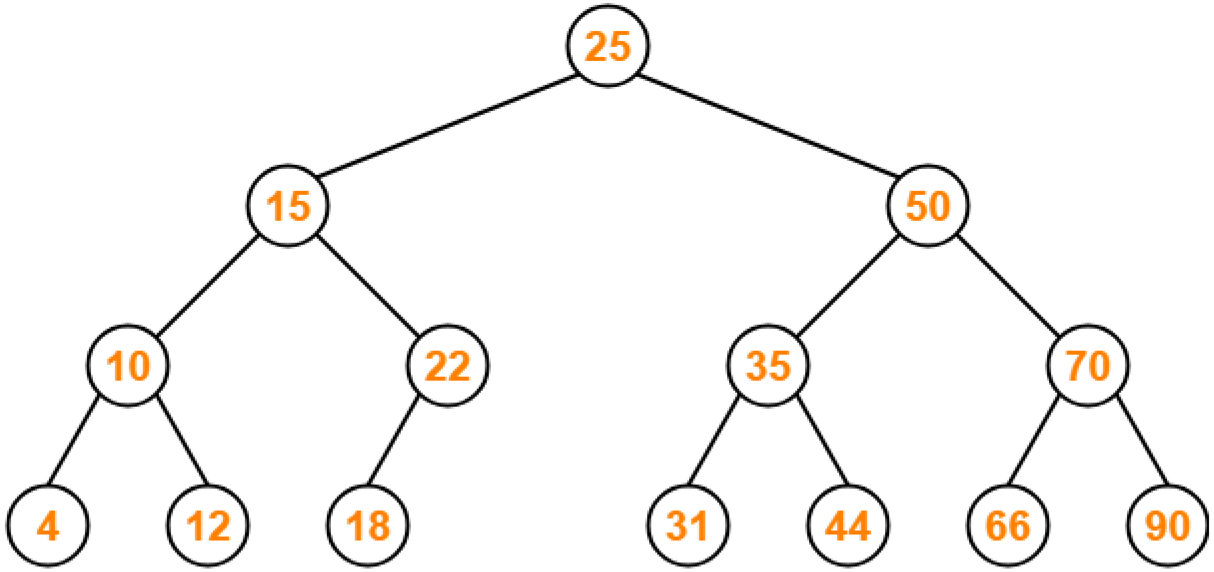
- 50, 70, 60, 20, 90, 10, 40, 100

Binary Search Tree-Operations

- Search
- Insert
- Delete

Binary Search Tree-Search

- Search Operation is performed to search a particular element in the Binary Search Tree.



Binary Search Tree

Binary Search Tree-Search

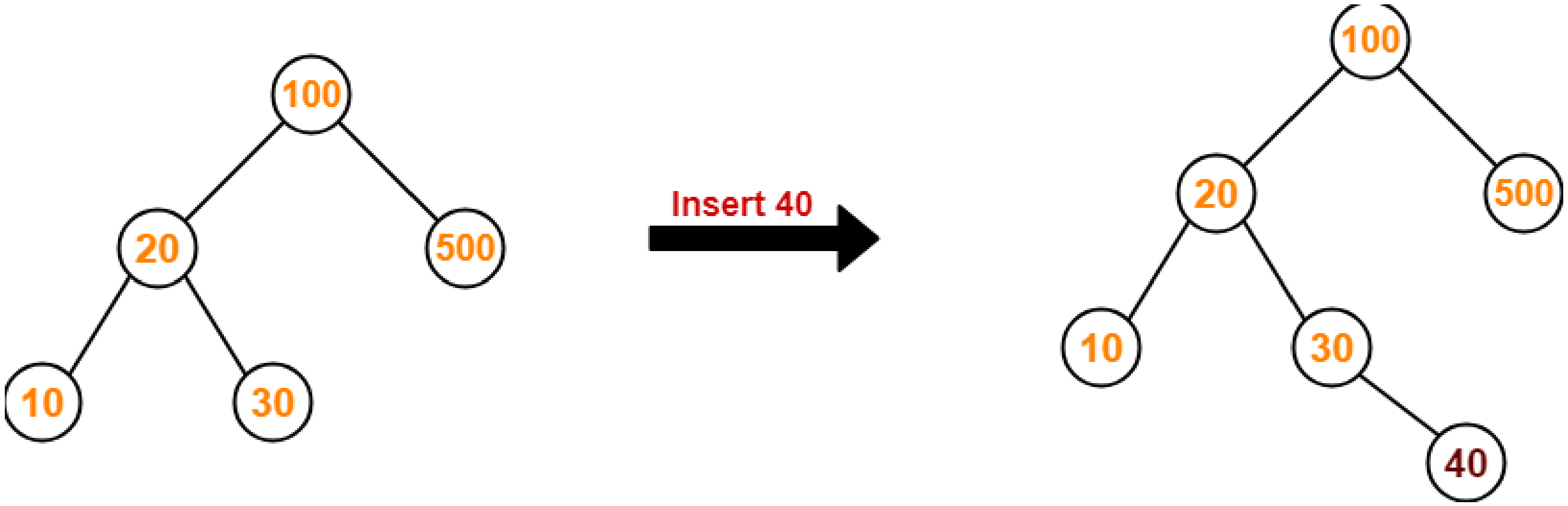
- Search Operation is performed to search a particular element in the Binary Search Tree.
- For searching a given key in the BST,
  - Compare the key with the value of root node.
  - If the key is present at the root node, then return the root node.
  - If the key is greater than the root node value, then recur for the root node's right subtree.
  - If the key is smaller than the root node value, then recur for the root node's left subtree.

Binary Search Tree-Search-Algorithm

- Our objective is to return **True** if a node exists with the value equal to the **item,** else return **False**.
  - Check if the root is NULL, return False if it is NULL.
  - Else, Compare **root.val** with **item**
  - **item == root.val:** return True
  - **item > root.val:** recurse for right subtree
  - **item < root.val:** recurse for left subtree

Binary Search Tree-Insert

- Insertion Operation is performed to insert an element in the Binary Search Tree.



Binary Search Tree-Insert

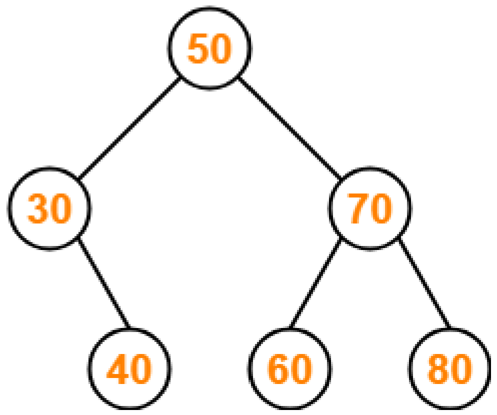
- Insertion Operation is performed to insert an element in the Binary Search Tree.
- **Rules-**
  - The insertion of a new key always takes place as the child of some leaf node.
  - For finding out the suitable leaf node,
    - Search the key to be inserted from the root node till some leaf node is reached.
    - Once a leaf node is reached, insert the key as child of that leaf node.

Binary Search Tree-Insert-Algorithm

- We need to insert a node in BST with value **item** and return the root of the new modified tree.
  - If the root is NULL, create a new node with value **item** and return it.
  - Else, Compare **item** with **root.val**
  - If **root.val < item**, recurse for right subtree
  - If **root.val > item**, recurse for left subtree

Binary Search Tree-Delete

- Deletion Operation is performed to delete a particular element from the Binary Search Tree.
- **Case-01: Deletion Of A Node Having No Child (Leaf Node)**
- **Case-02: Deletion Of A Node Having Only One Child**
- **Case-03: Deletion Of A Node Having Two Children**



Case-01: Deletion Of A Node Having No Child (Leaf Node)

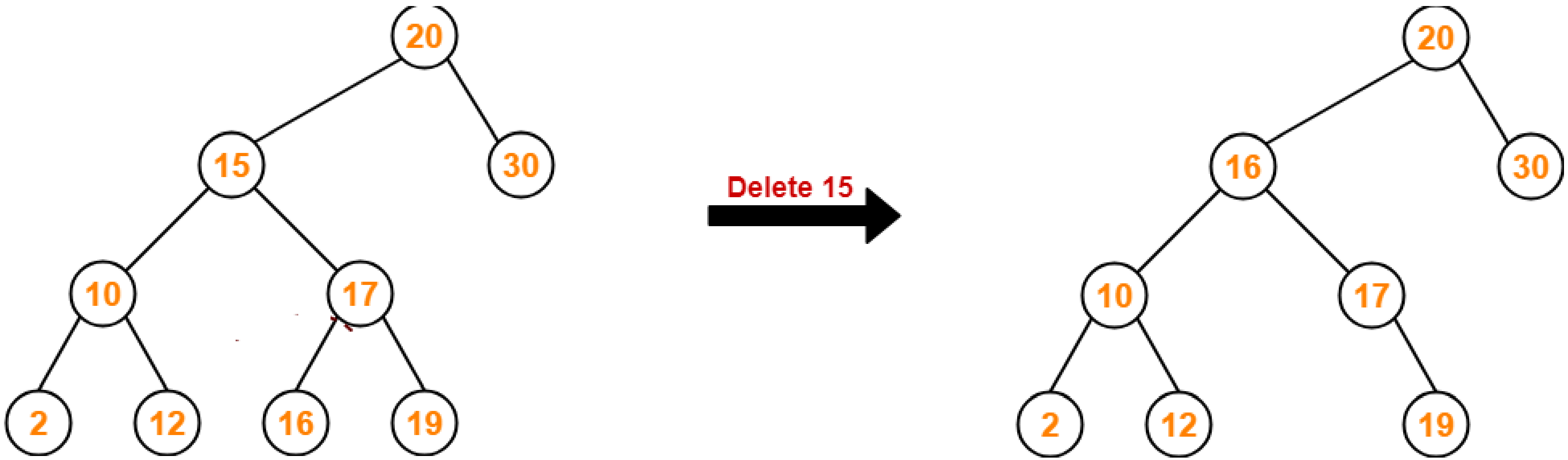
- Just remove / disconnect the leaf node that is to deleted from the tree.

Case-02: Deletion Of A Node Having Only One Child

- Just make the child of the deleting node, the child of its grandparent.

Case-03: Deletion Of A Node Having Two Children

- A node with two children may be deleted from the BST in the following two ways-



Case-03: Deletion Of A Node Having Two Children

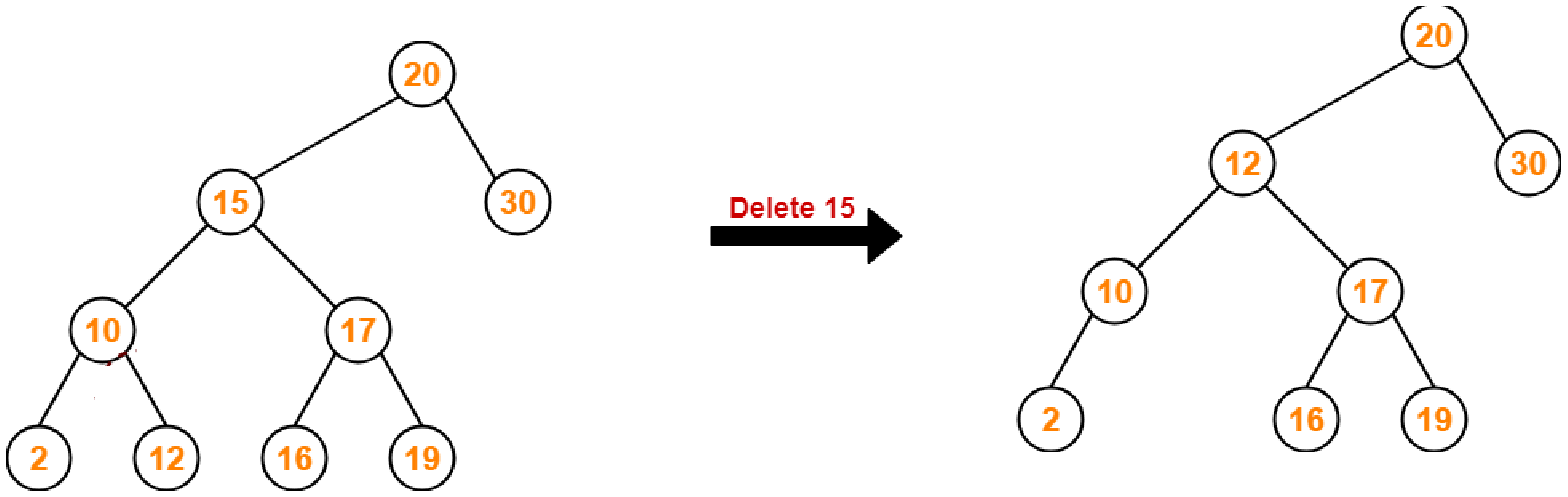
- A node with two children may be deleted from the BST in the following two ways-
- **Method-01:**
  - Visit to the right subtree of the deleting node.
  - Pluck the least value element in the right subtree.
  - Replace the deleting element with this element.

Case-03: Deletion Of A Node Having Two Children

- A node with two children may be deleted from the BST in the following two ways-
- **<u>Method-02:</u>**
  - Visit to the left subtree of the deleting node.
  - Pluck the greatest value element in the right subtree.
  - Replace the deleting element with this element.

Case-03: Deletion Of A Node Having Two Children

- A node with two children may be deleted from the BST in the following two ways-

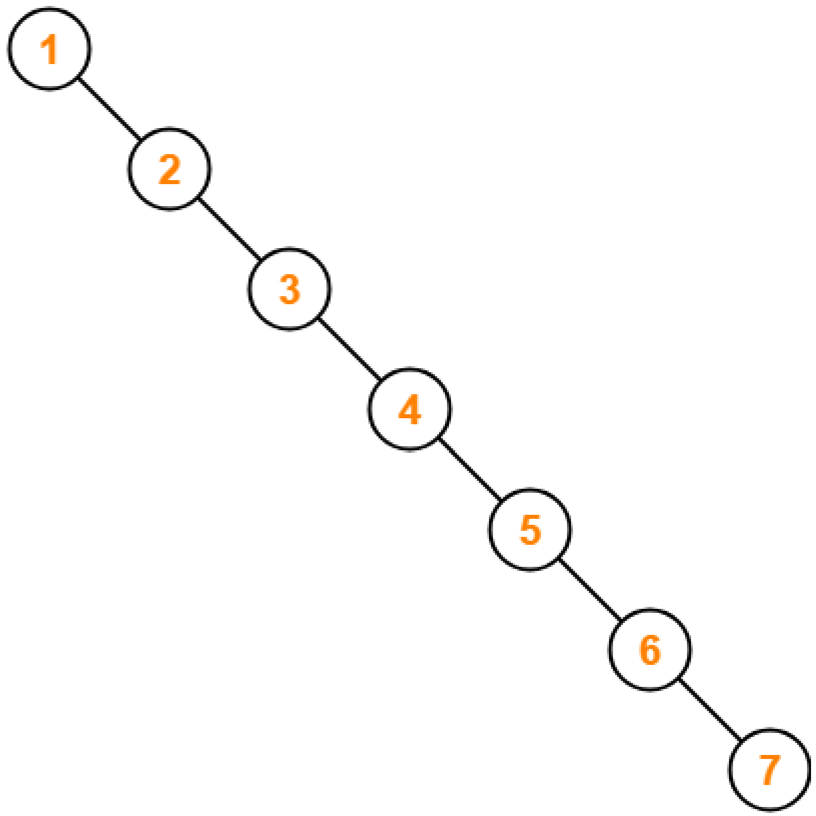


Binary Search Tree-Delete

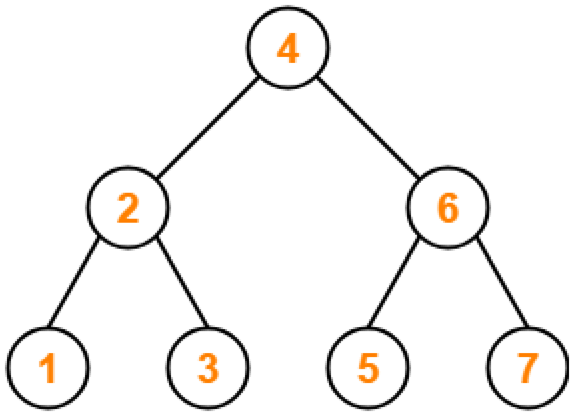
- You need to delete the node with value **item** and then return the **root** of the modified tree. First, we need to find the node to be deleted and then replace it by the appropriate node if needed.
- Check if the root is NULL, if it is, just return the root itself. It's an empty tree!
- If **root.val < item**, recurse the right subtree.
- If **root.val > item**, recurse the left subtree.
- If both above conditions above false, this means **root.val == item**.
- Now we first need to check how many child did **root** have.
- **CASE 1: No Child →** Just delete **root** or deallocate space occupied by it
- **CASE 2: One Child** →Replace **root** by its child
- **CASE 3: Two Children**
- Find the inorder successor of the **root** (Its the smallest element of its right subtree). Let's call it **new_root.**
- Replace **root** by its inorder successor
- Now recurse to the right subtree and delete **new_root.**
- Return the **root**.

Time Complexity of Binary Search Tree

- Time Complexity depends on Height of a Tree.
- i.e O(n) to O(logn)



Skewed Binary Search Tree



Balanced Binary Search Tree