TREE
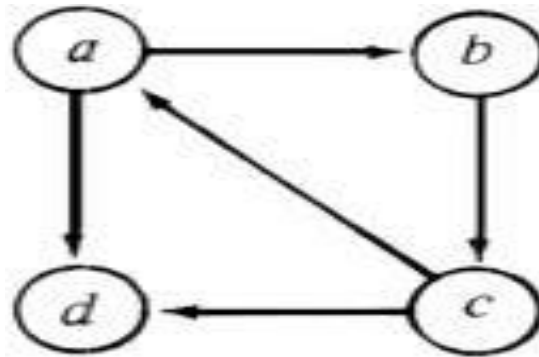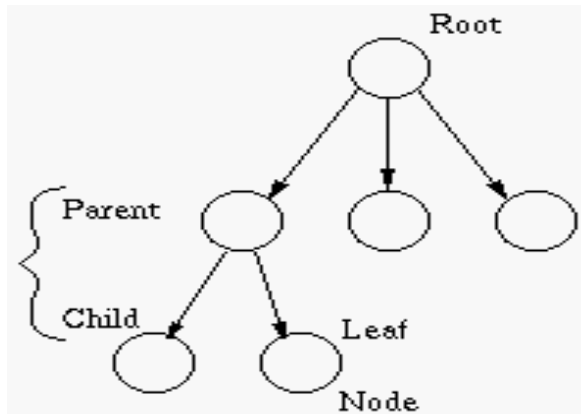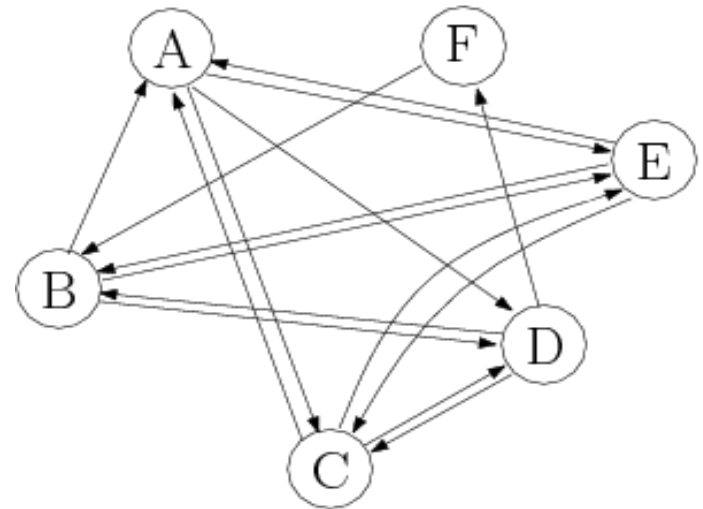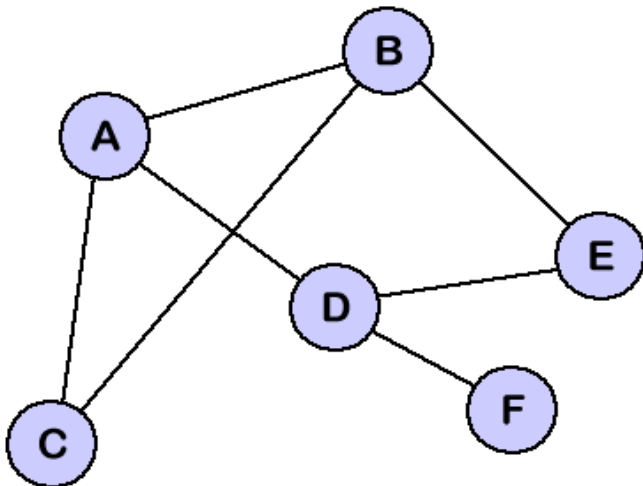
NON-LINEAR DATA STRUCTURE

Graphs

Trees

Figure: Tree data structure

GRAPH

A graph G consists of a nonempty set V called the set of nodes, a set E called the set of edges, and a mapping from E to pairs of elements of V.

$$G : E \rightarrow V \times V$$

It is denoted by G = (V, E).

GRAPH

- **Edge:** Two nodes are connected by edge. It is the connection between two nodes.

- **Cycle:** A path in a graph that starts and ends at the same node is called a cycle, and the graph containing a cycle is called cyclic.

- **Degree of a Node:** The degree of a node is the no of children of that node.

- **In-degree:** Number of edges terminated at given node is called in-degree of node.

- **Out-degree:** Number of edges emerging from given node is called out-degree of node.
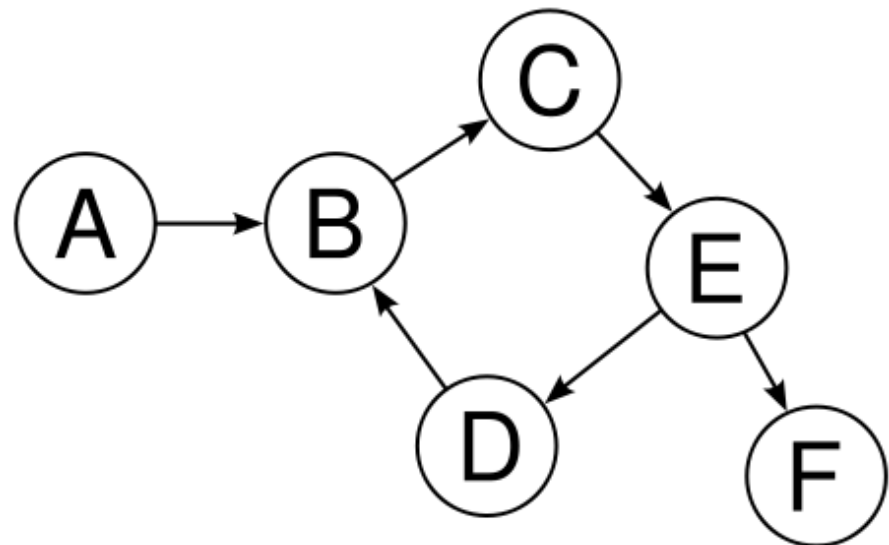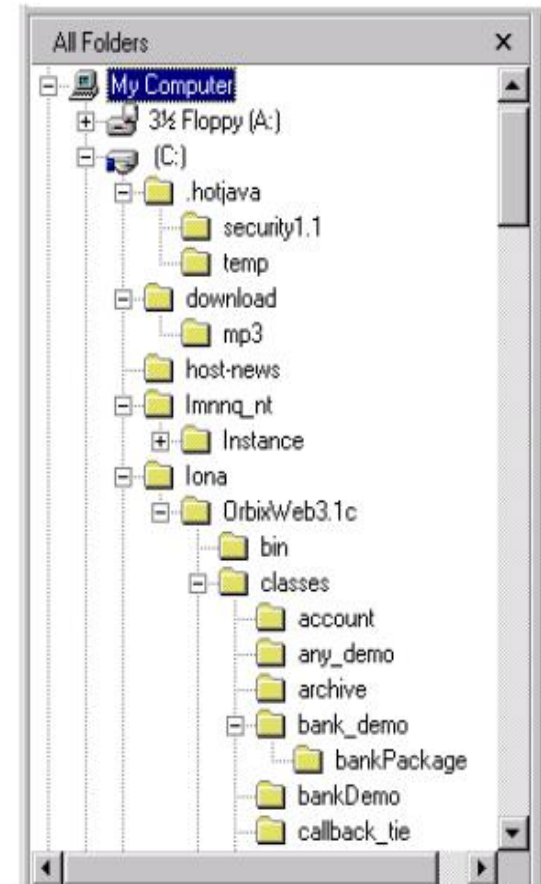
TREE

Definition: A directed tree is an <u>acyclic</u> graph which has one node called its <u>root</u>, with in-degree 0, while all other nodes have in-degree 1.
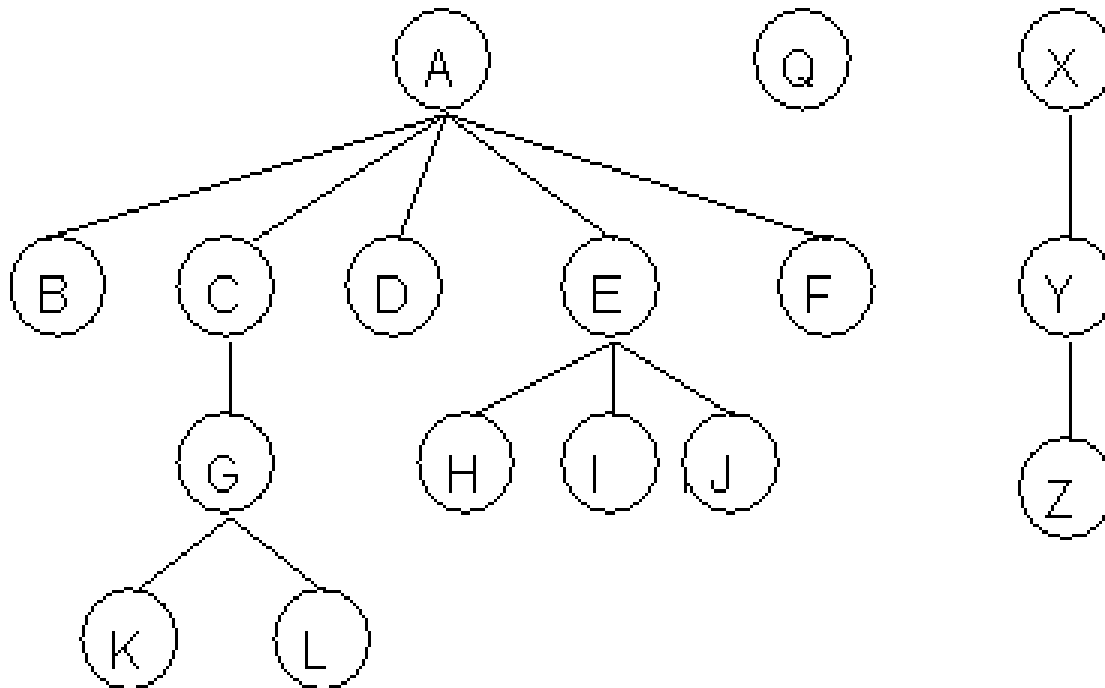
Where have you seen a tree structure before?

- Examples:
  1. Directory Tree
  2. Family Tree
  3. Company Organization Chart
 4. Table of Contents

TREE TERMINOLOGIES

- **Root node:** The root node R is the topmost node in the tree. If R = NULL, then it means the tree is empty. Root node has in-degree 0.

- **Sub-tree:** : A node and all its descendent ignoring the node's parent, this is itself a tree. T1, T2, and T3 are called the sub-trees of R.

- **Path:** A sequence of consecutive edges is called a path.

- **Ancestor node:** An ancestor of a node is any predecessor node on the path from root to that node.

- **Descendant node:** A descendant node is any successor node on any path from the node to a leaf node.


Root node
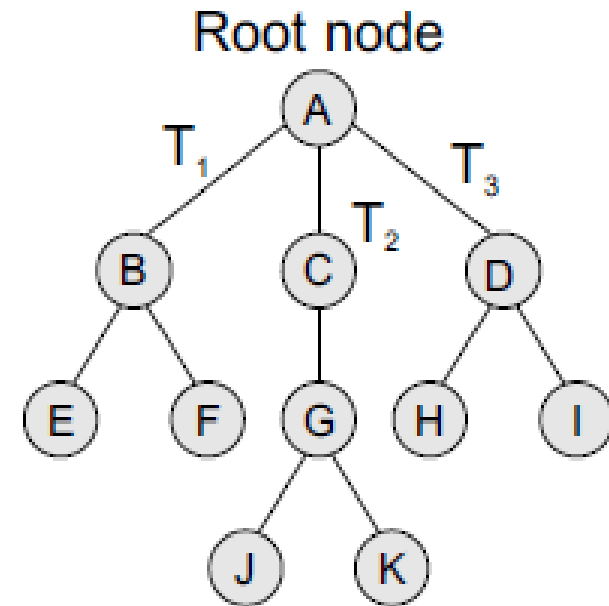
TREE TERMINOLOGIES

- **Parent**: Single node that directly precedes a node. All nodes have 1 parent except root (has 0)

- **Child**: One or more nodes that directly follow a node.

- **Level number:** Every node in the tree is assigned a level number in such a way that the root node is at level 0, children of the root node are at level number 1. Thus, every node is at one level higher than its parent. So, all child nodes have a level number given by parent's level number + 1.

- **Leaf node:** A node that has no children is called the leaf node or the terminal node. Leaf node has out degree 0.
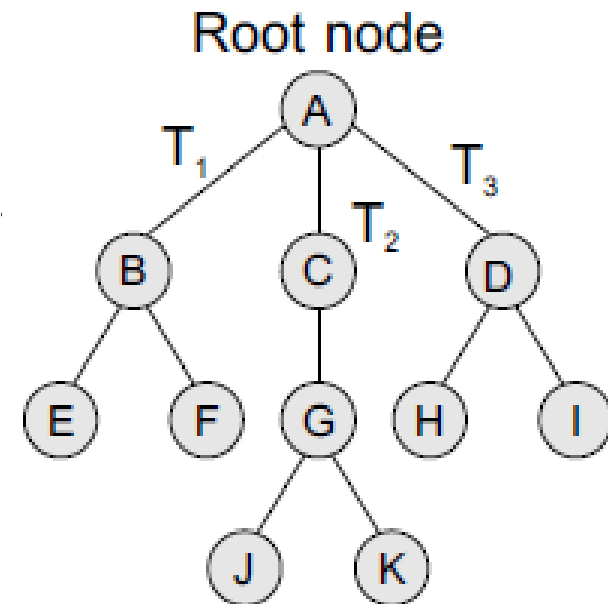
TREE TERMINOLOGIES

- **Degree:** Degree of a node is equal to the number of children that a node has. The degree of a leaf node is zero.

- **Siblings**: All nodes that are at the same level and share the same parent are called siblings (brothers).

- **Depth:** The depth of a node N is given as the length of the path from the root R to the node N. The depth of the root node is zero.

- **Height:** It is the total number of nodes on the path from the root node to the deepest node in the tree. A tree with only a root node has a height of 1.

- **Forest:** A forest is a set of $n \geq 0$ disjoint trees.
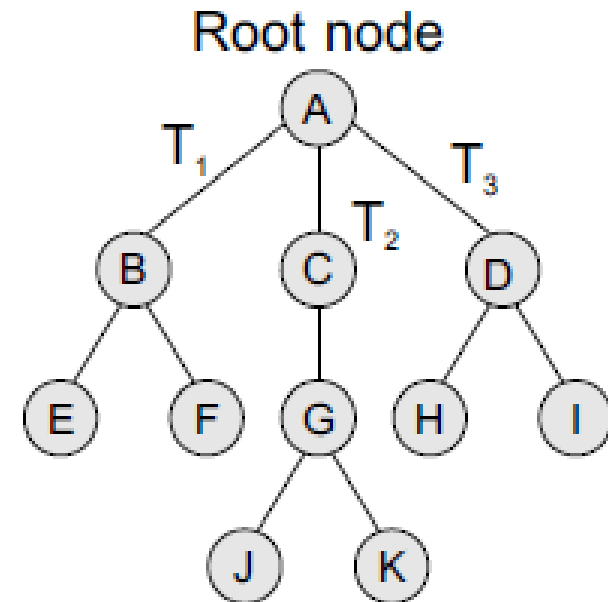
TREE

- A tree is recursively defined as a set of one or more nodes where one node is designated as the root of the tree and all the remaining nodes can be partitioned into non-empty sets each of which is a sub-tree of the root.

- Trees are of following 6 types:
  1. General trees
  2. Forests
  3. Binary trees
  4. Binary search trees
  5. Expression trees
  6. Tournament trees

BINARY TREE

Definition: The tree in which out-degree of every node is less than or equal to 2 (at most 2) is called a binary tree.
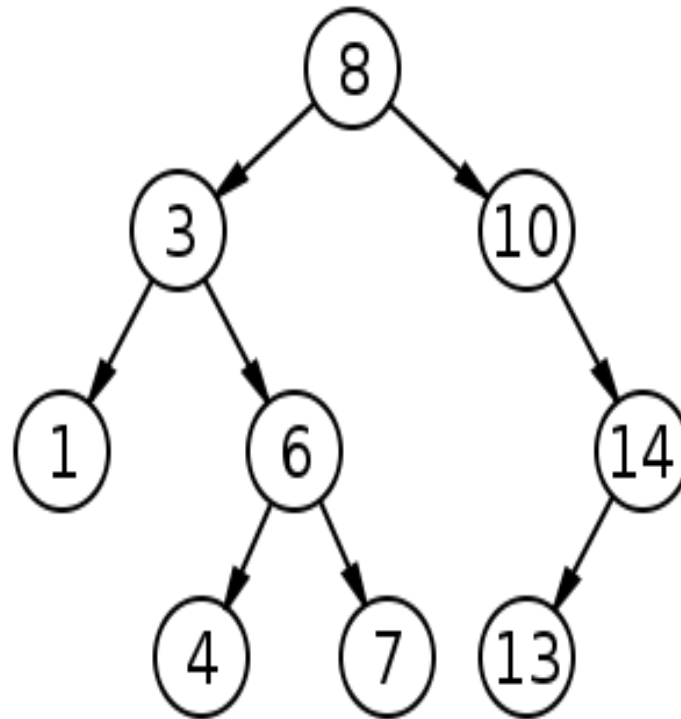
BINARY TREE

<u>Recursive Definition</u>:

A binary tree is a finite set of elements that is either empty or is partitioned into 3 disjoint subsets:

i.   Root of tree

ii.   Left sub-tree which is itself binary tree

iii.   Right sub-tree which is itself binary tree

STRICTLY BINARY TREE

Strictly Binary Tree:

If every non-leaf node in a binary tree has **nonempty left and right sub trees**, then the tree is termed as a strictly binary tree.
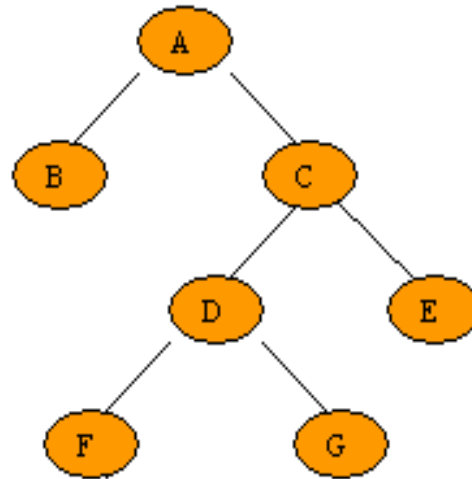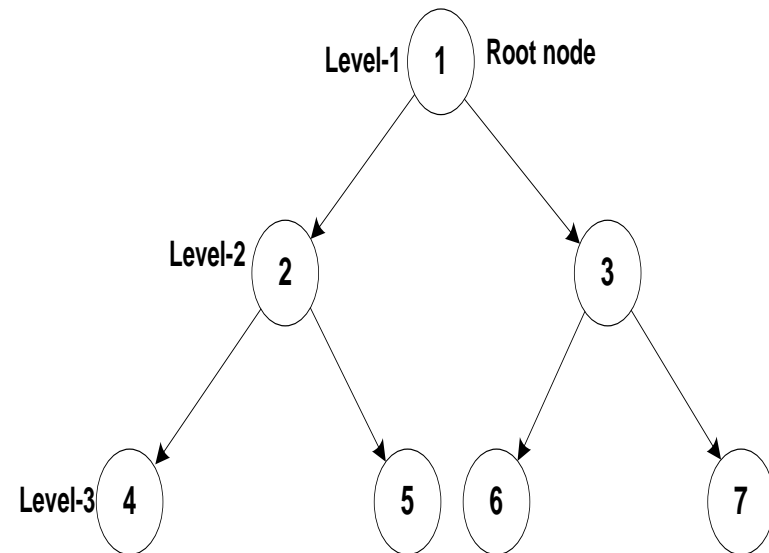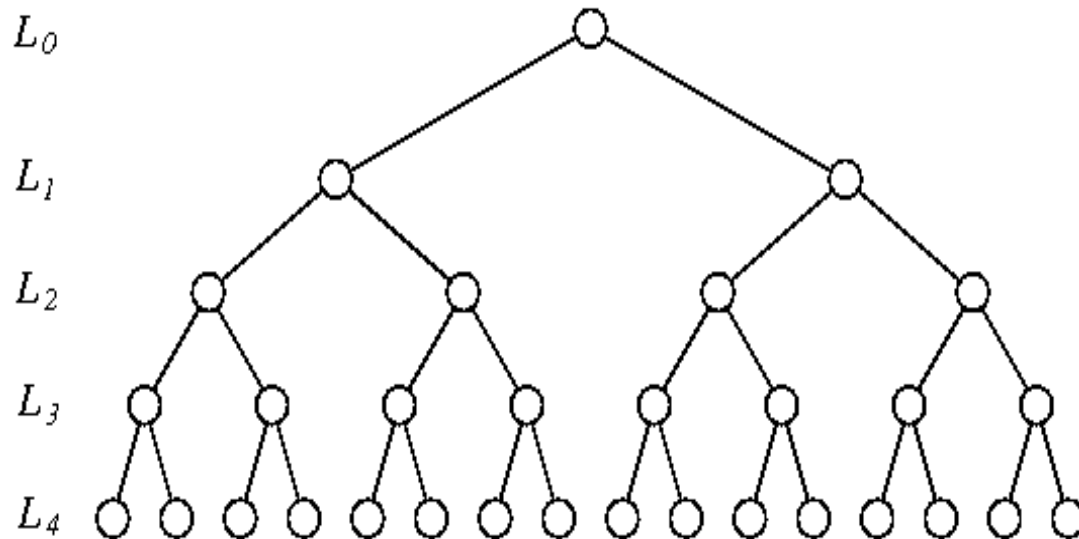

Strictly binary tree

COMPLETE BINARY TREE

Complete Binary Tree :

In a strictly binary tree of depth d, if all the leaves are at level d then it is known as a complete binary tree.
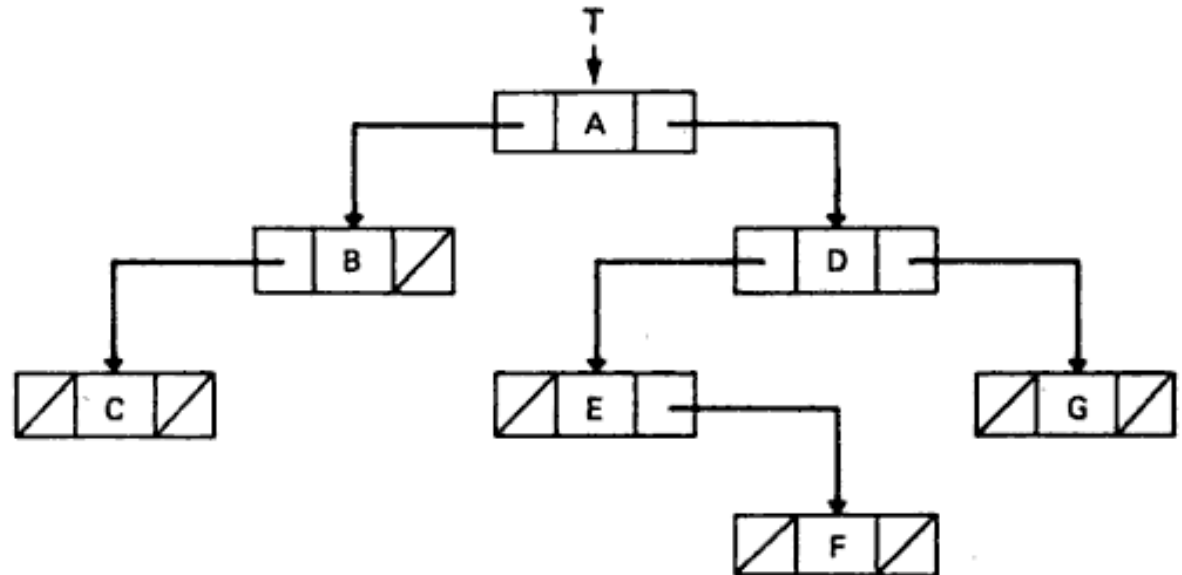
OR

If the out-degree of every node is exactly equal to 2 or 0 and the number of nodes at level i is 2^{i-1} then the tree is called full or complete binary tree.

BINARY TREE REPRESENTATION

Typical node of a binary tree implementation:

```
struct node
{
        int info;
        struct node *left;
        struct node *right;
};
```

Linked representation of a binary tree.

BINARY TREE TRAVERSALS

- The most common operation on a binary tree is the traversal of its nodes.

- To traverse a tree means to visit or pass through each node for some kind of processing, viz. printing, searching, updating, insertion, deletion, etc…

- Three types of traversals:

    1. Preorder Traversal – (VLR)

    2. Inorder Traversal – (LVR)

    3. Postorder Traversal – (LRV)

PREORDER TRAVERSAL – (VLR)

Steps:

1. Visit the root Vertex.

2. Traverse the Left sub-tree in preorder.
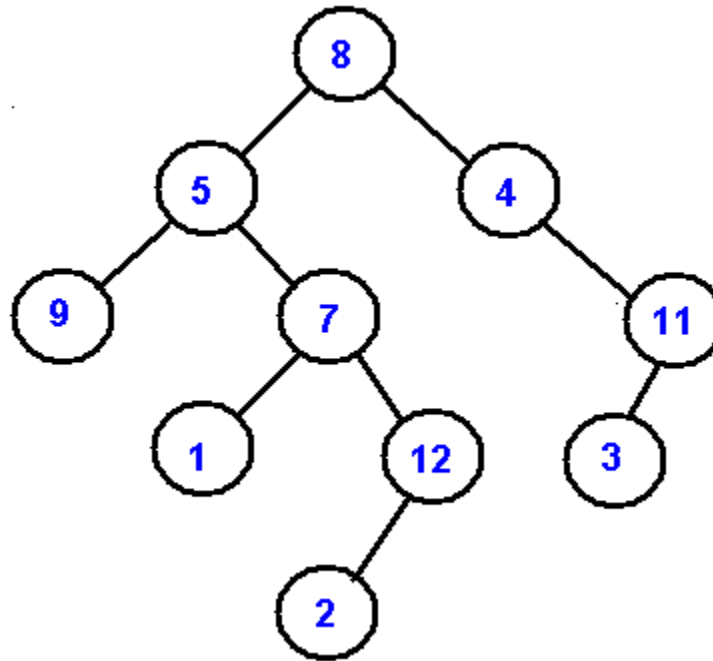
3. Traverse the Right sub-tree in preorder.
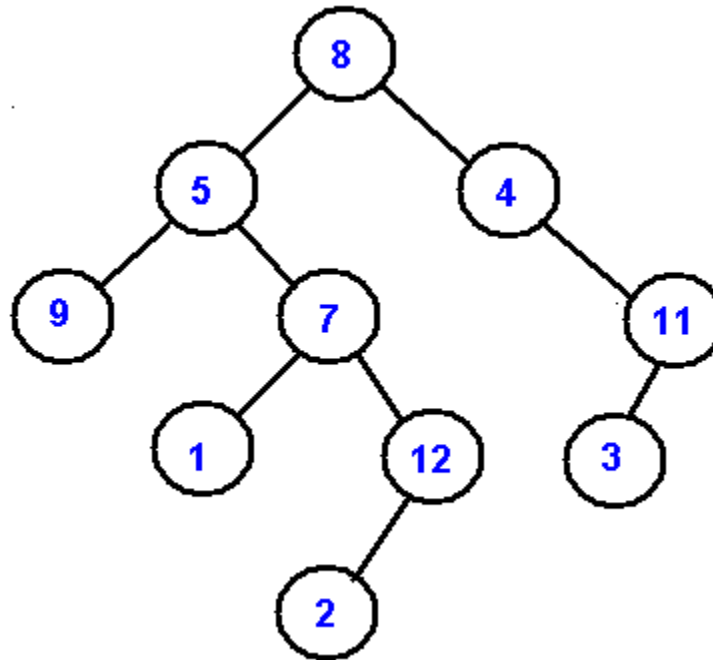
Preorder Traversal: 8, 5, 9, 7, 1, 12, 2, 4, 11, 3

INORDER TRAVERSAL – (LVR)

1. Traverse the Left sub-tree in inorder.

2. Visit the root Vertex.
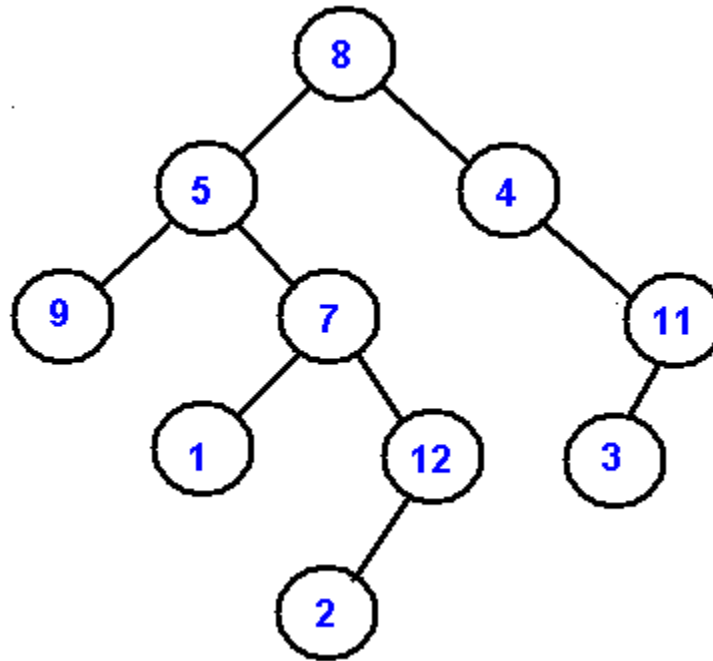
3. Traverse the Right sub-tree in inorder.

Inorder Traversal: 9, 5, 1, 7, 2, 12, 8, 4, 3, 11
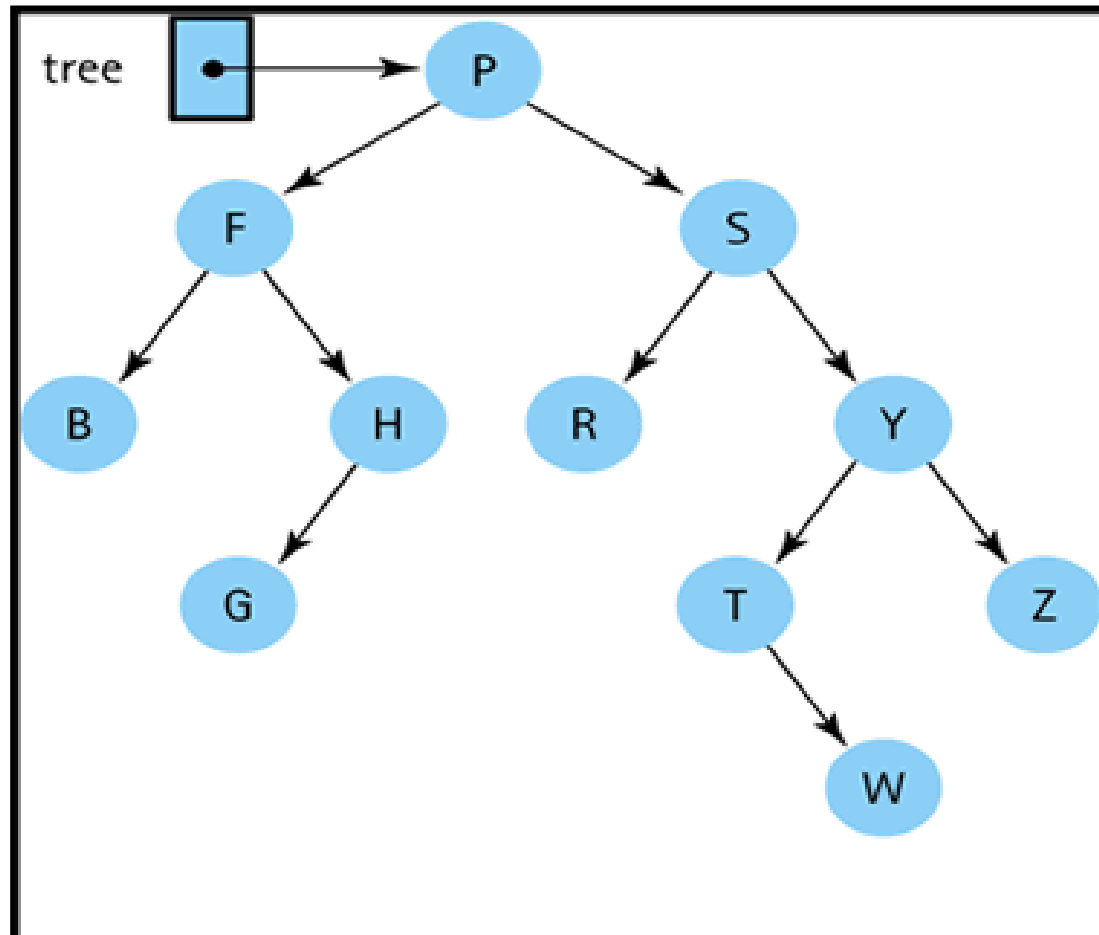
POSTORDER TRAVERSAL – (LRV)

Steps:

1. Traverse the Left sub-tree in postorder.

2. Traverse the Right sub-tree in postorder.

3. Visit the root Vertex.

Postorder Traversal: 9, 1, 2, 12, 7, 5, 3, 11, 4, 8

TRAVERSAL EXAMPLE

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
```

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: END
```

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 SEARCH TREE
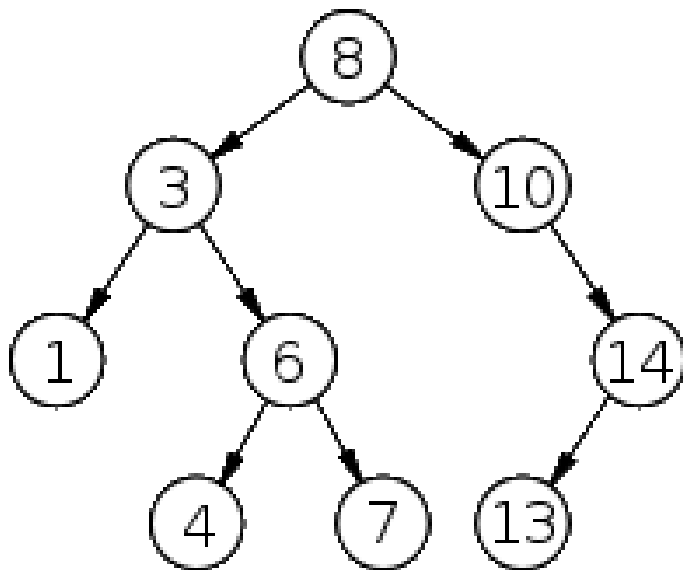
BINARY SEARCH TREE

Definition:

A BST is a binary tree, which is either empty or in which each node contains a key that satisfies the following conditions:

i.   All keys are distinct.

ii.   For every node X, the value of all the keys in its left sub-tree are smaller than X.

iii.   For every node X, the value of all the keys in its right sub-tree are greater than X.

BINARY SEARCH TREE
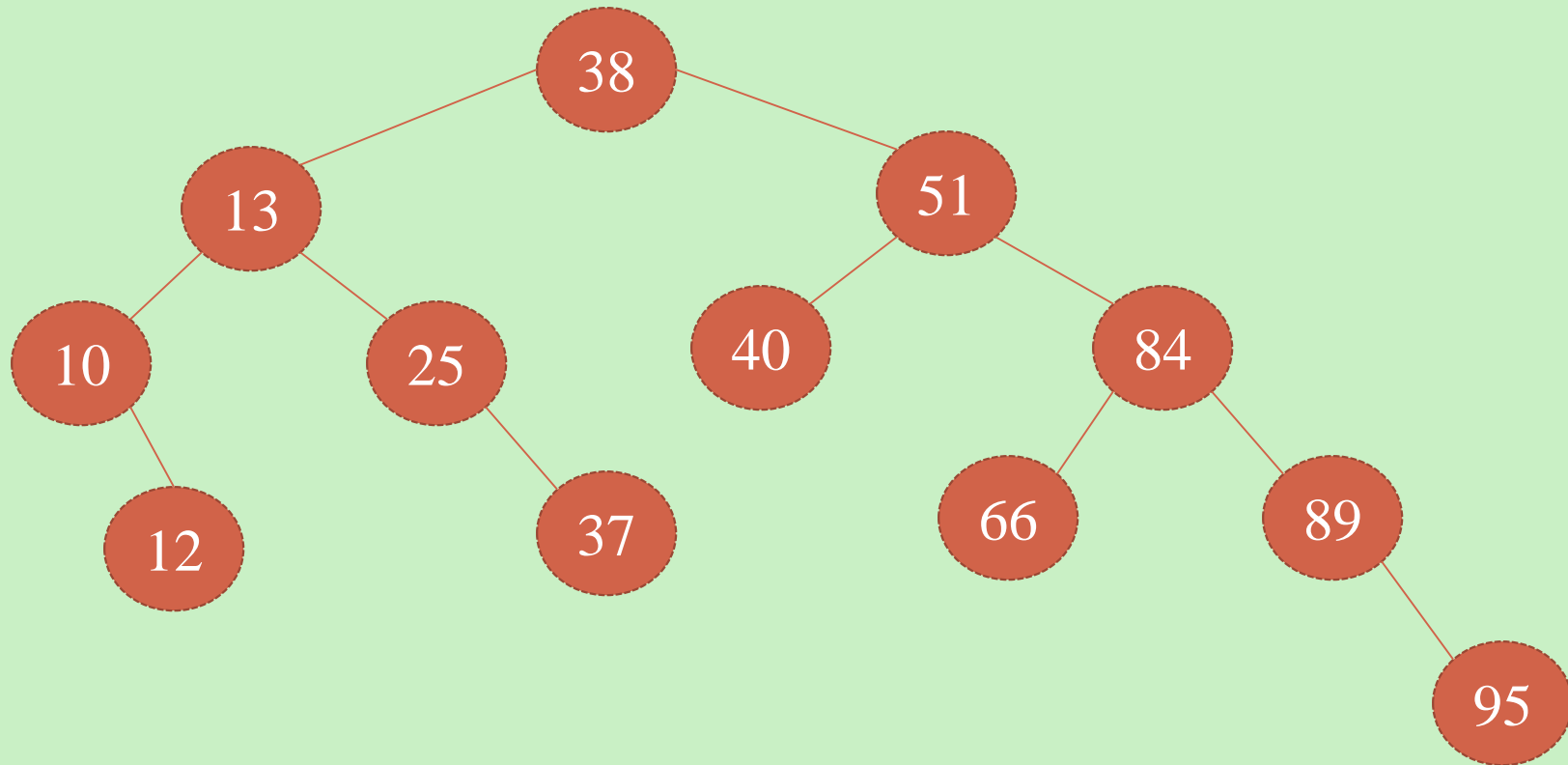
Binary Seach Tree

In-order traversal of tree

1	3	4	6	7	8	10	13	14

OPERATIONS ON BINARY SEARCH TREE

1. Insert

2. Delete

3. Search

Binary Search Tree Example

Tree resulting from the following insertions: 38, 13, 51, 10, 12, 40, 84, 25, 89, 37, 66, 95
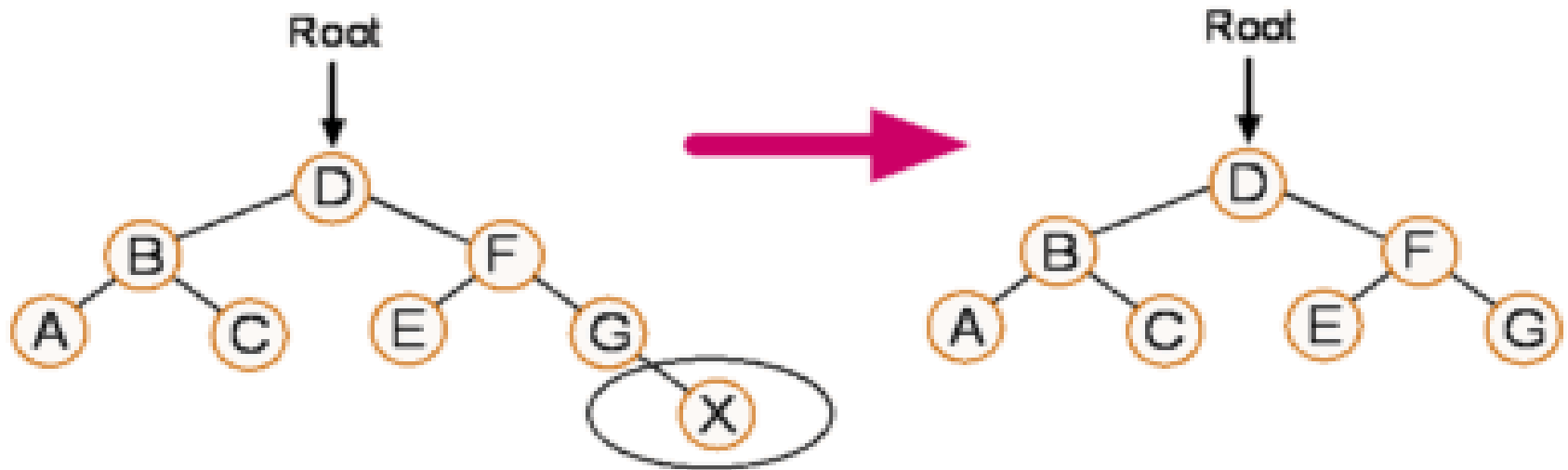
DELETE OPERATION ON BINARY SEARCH TREE

- When the delete operation is performed on a BST, its properties should be maintained.

- There are 3 possibilities for node to be deleted:

  Case I: leaf node

  Case II: node with one child

  Case III: node with two children

DELETE OPERATION ON BINARY SEARCH TREE
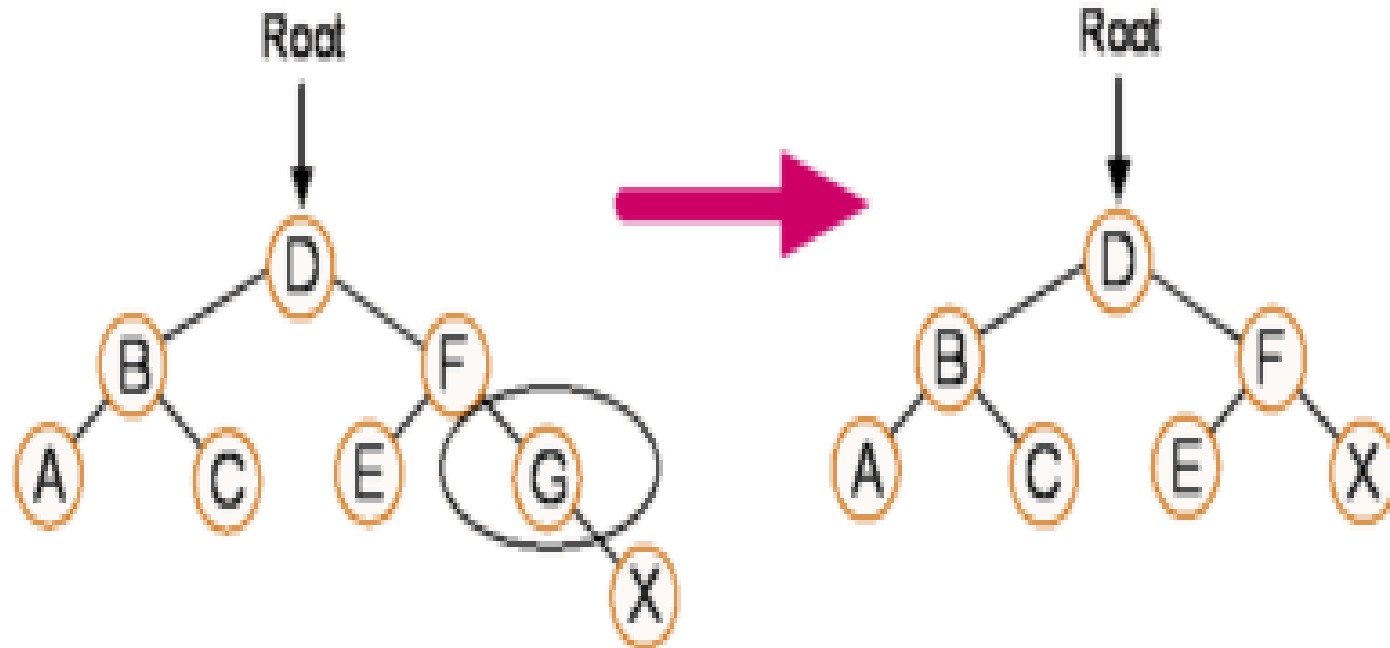
Case I: Leaf Node

Simply set the parent node pointer of the node to be deleted to NULL.

DELETE OPERATION ON BINARY SEARCH TREE

<u>Case II: node with one child</u>

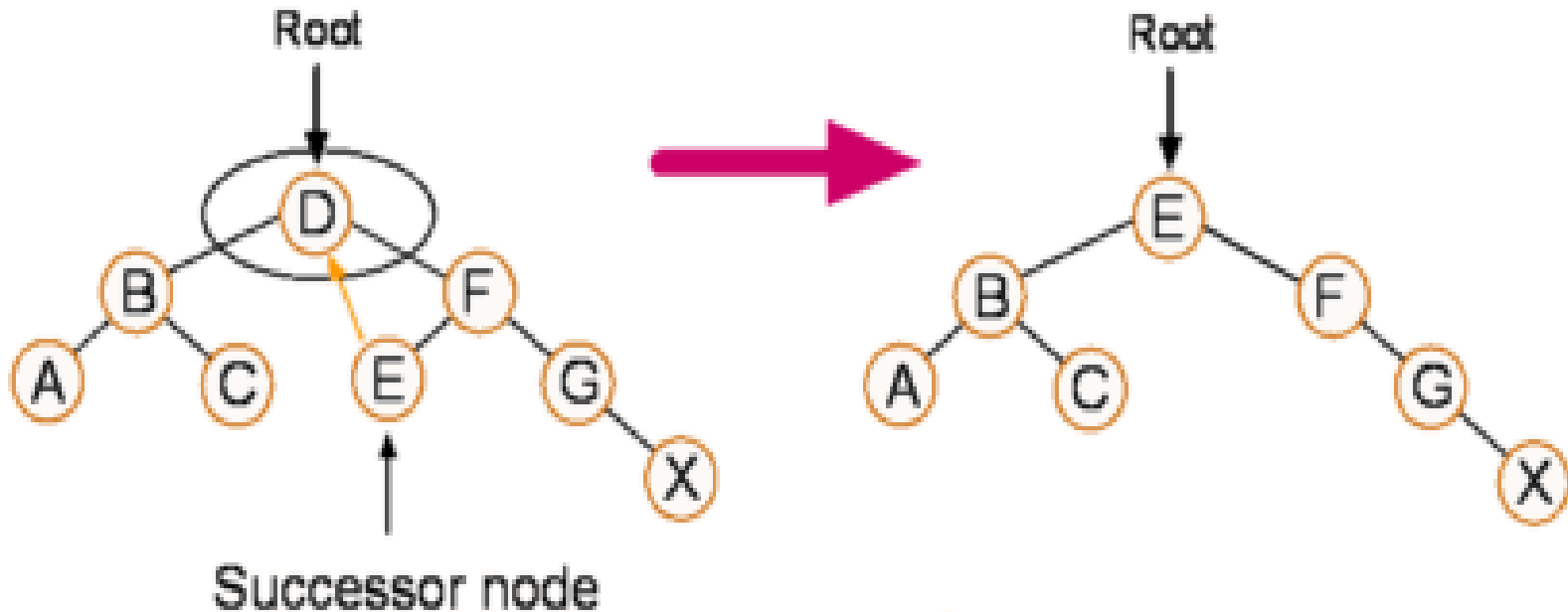Set the parent node pointer of node to be deleted to point to its child.

DELETE OPERATION ON BINARY SEARCH TREE

<u>Case III: node with two children</u>

Steps: (assume node 'p' is to be deleted)

1) Replace p with the smallest node 's' in its right sub-trsee.

2) Replace 's' with its right son, if any.

APPLICATION OF TREE

Some of the applications of a tree are:

- Binary Search Trees for Efficient Searching

- Dictionary

- Disk Storage

- Expression Tree

THANK YOU