Module 3 Non-linear data structures: Tree and Graph

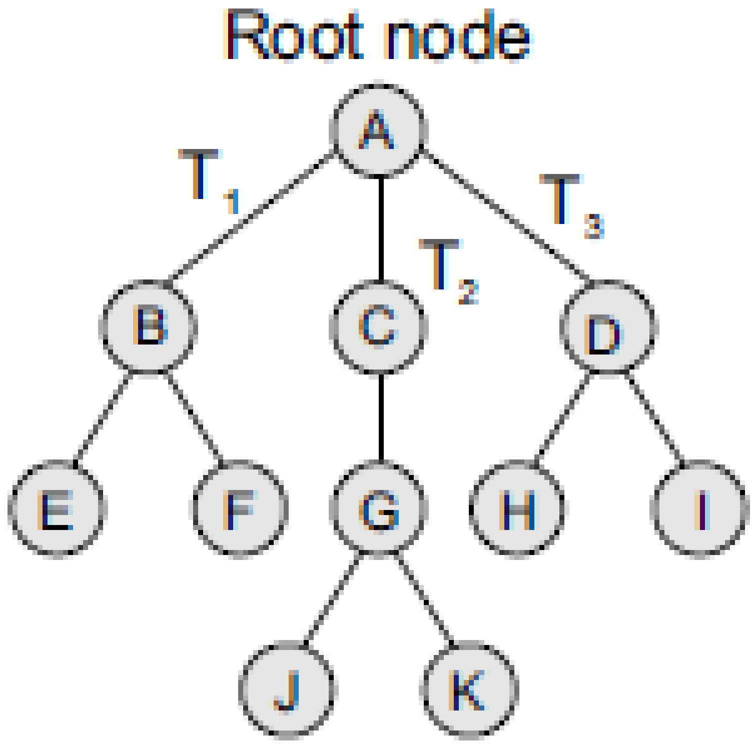
By
Ms. Sarika Dharangaonkar
Assistant Professor,
KJSCE

Contents: Module 3.1

- Basic tree terminologies, Types of trees,
- Binary tree representation, Binary tree operation, Binary tree traversal Binary search tree implementation, Threaded binary trees.
- Different Search Trees -AVL tree, Multiway Search Tree,
- B Tree, B+ Tree, and Trie , Applications/Case study of trees.

Tree

- It is Non-Linear Data Structure
- Trees are set of one or more nodes where one node is designated as the root of the tree
- Remaining nodes can be partitioned into non-empty sets each of which is a sub-tree of the root



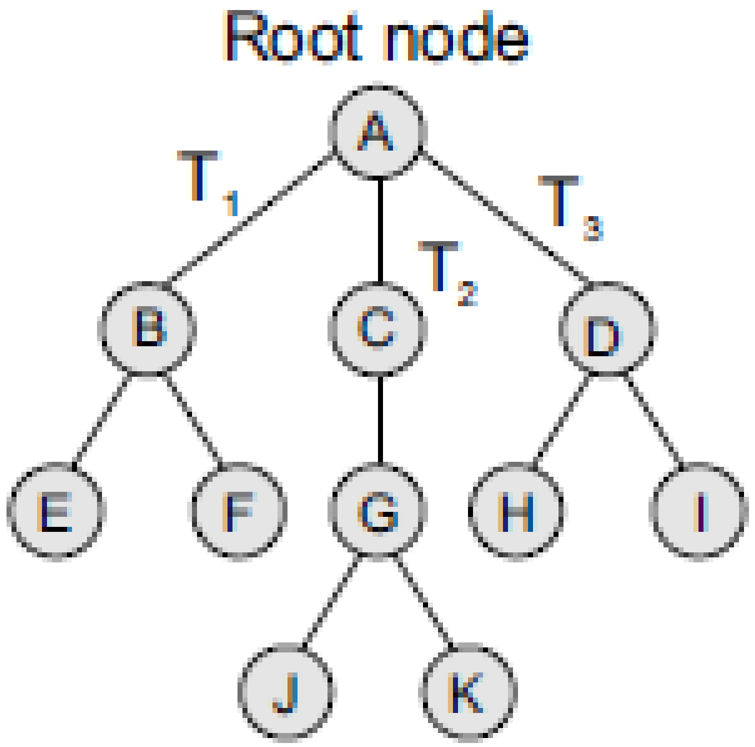
Sub-trees If the root node A is not NULL, then the trees T1, T2, and T3 are called the sub-trees of A.

Leaf node A node that has no children is called the leaf node or the terminal node.

Path A sequence of consecutive edges is called a *path*. For example , the path from the root node A to node I is given as: A, D, and I.

Ancestor node An ancestor of a node is any predecessor node on the path from root to that node. In the tree given in Fig. nodes A, C, and G are the ancestors of node K.

The root node does not have any ancestors



Descendant node : A descendant node is any successor node on any path from the node to a leaf node. Leaf nodes do not have any descendants. In the tree given in Fig. nodes C, G, J, and K are the descendants of node A.

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.

Degree: Degree of a node is equal to the number of children that a node has. The degree of a leaf node is zero.
In-degree: In-degree of a node is the number of edges arriving at that node.
Out-degree: Out-degree of a node is the number of edges leaving that node.

TYPES OF TREES

Trees are of following 6 types:

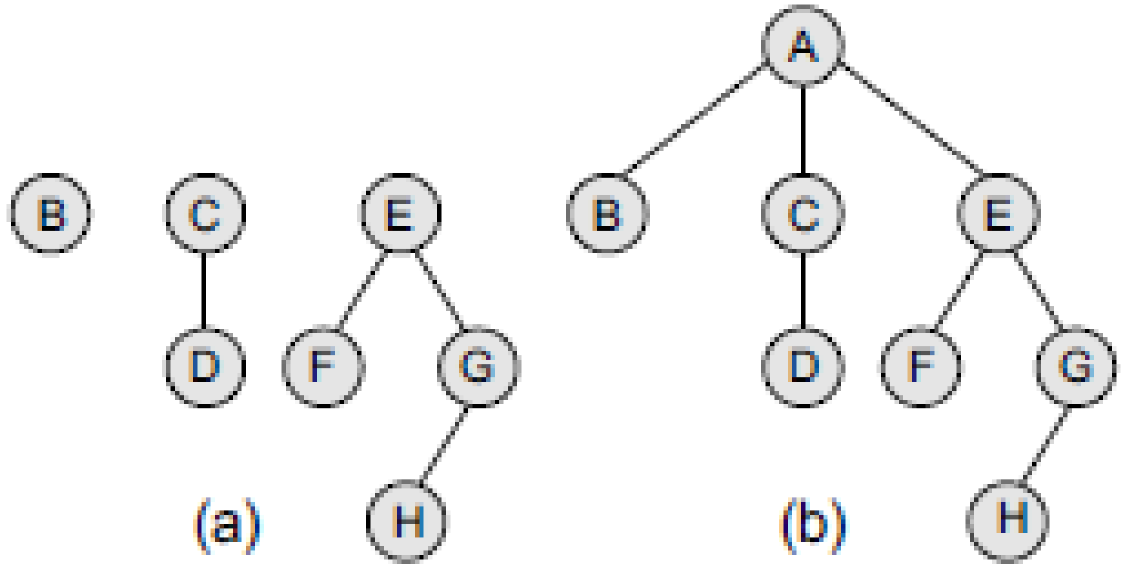
1. General trees
2. Forests
3. Binary trees
4. Binary search trees
5. Expression trees
6. Tournament trees

1. General Trees

- A general tree in data structures is a widely used hierarchical data structure that allows nodes to have any number of children (including zero, making it a possibility for nodes to have no children).
- **Nodes and Structure:**
  - Each node in a general tree can have any number of children.
  - Nodes can have zero or more child nodes.
- **Parent and Child Relationship:**
  - Nodes in a general tree maintain a parent-child relationship.
  - Each node, except the root, has a single parent node.
  - Nodes with the same parent are considered siblings.

2. Forests

- A forest in data structures is a collection of disjoint trees. Each tree in the forest is a separate tree, meaning there are no common nodes or edges between any two trees in the forest.
- In other words, a forest is a set of individual trees.
- A set of disjoint trees (or forests) is obtained by deleting the root and the edges connecting the root node to nodes at level 1.

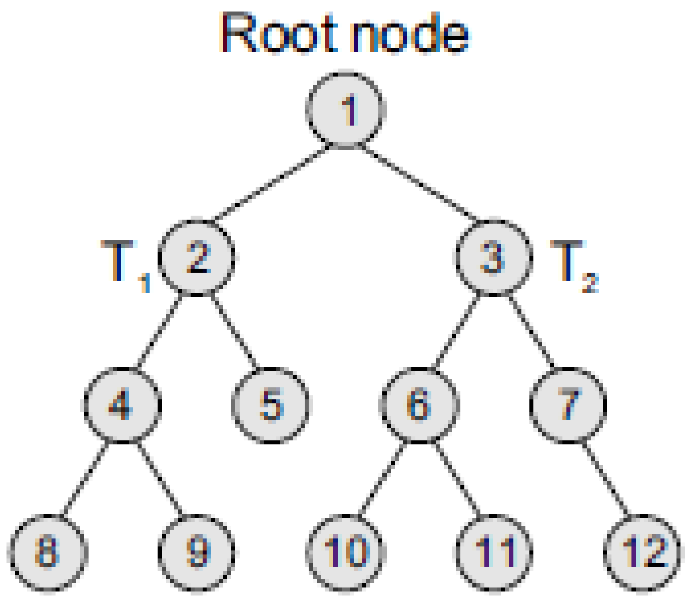


Forest and its corresponding tree

3. Binary trees

- A binary tree is a data structure that is defined as a collection of elements called nodes.
- In a binary tree, the topmost element is called the root node, and each node has 0, 1, or at the most 2 children.
- A node that has zero children is called a leaf node or a terminal node.
- Every node contains a data element, a left pointer which points to the left child, and a right pointer which points to the right child.
- The root element is pointed by a 'root' pointer.
- If root = NULL, then it means the tree is empty.

Binary Trees…



Binary tree

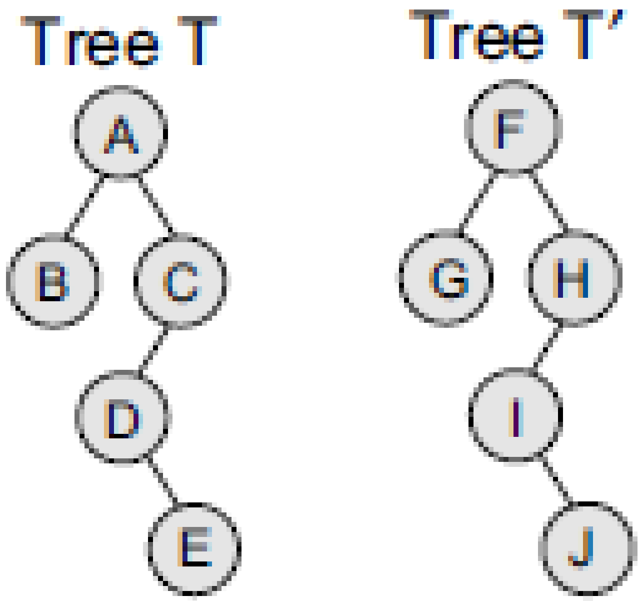
***Parent*:** If N is any node in T that has *left successor* S1 and *right successor* S2, then N is called the *parent* of S1 and S2. Correspondingly, S1 and S2 are called the left child and the right child of N. Every node other than the root node has a parent.

Degree of a node It is equal to the number of children that a node has. The degree of a leaf node is zero.
For example, in the tree, degree of node 4 is 2, degree of node 5 is zero

Sibling All nodes that are at the same level and share the same parent are called *siblings* (brothers).
For example, nodes 2 and 3; nodes 4 and 5; nodes 6 and 7; nodes 8 and 9; and nodes 10 and 11 are siblings.

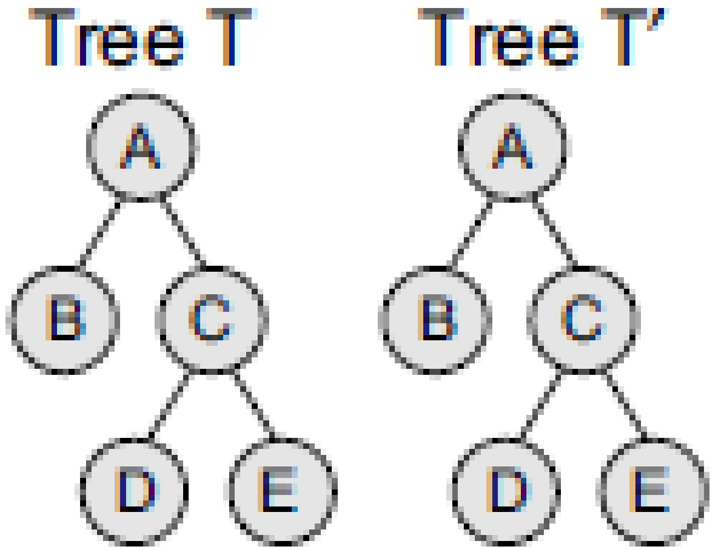
Leaf node A node that has no children is called a leaf node or a terminal node. The leaf nodes in the tree are: 8, 9, 5, 10, 11, and 12.

Binary Trees…



Similar binary trees:

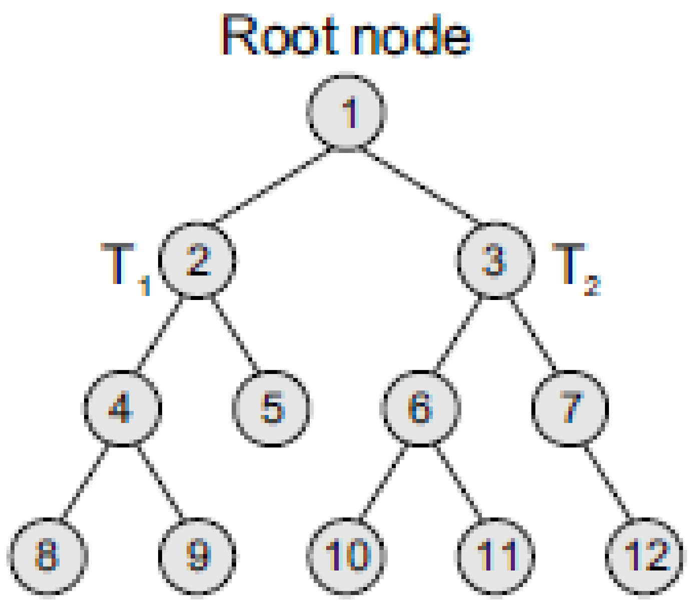
Two binary trees T and T’ are said to be similar if both these trees have the same structure. Figure shows two *similar binary trees*.



Copies :

Two binary trees T and T’ are said to be *copies* if they have similar structure and if they have same content at the corresponding nodes.
Figure shows that T’ is a copy of T.

Binary Trees…



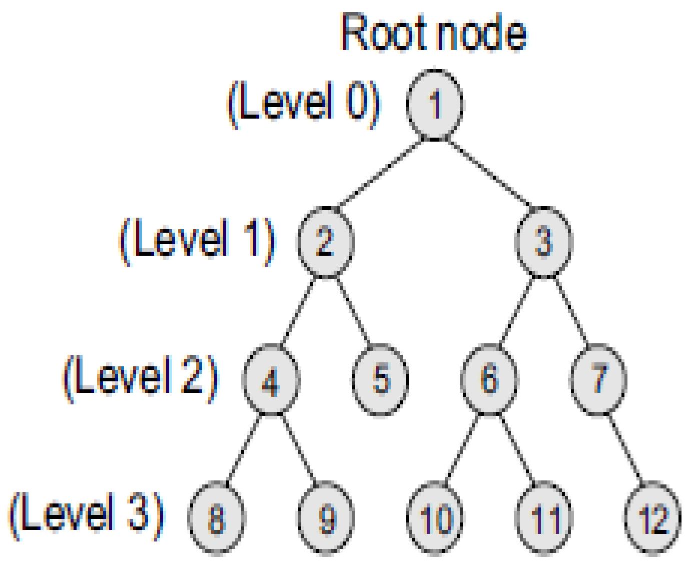
Edge It is the line connecting a node N to any of its successors. A binary tree of n nodes has exactly n – 1 edges because every node except the root node is connected to its parent via an edge.

Path A sequence of consecutive edges.
For example, in Fig., the path from the root node to the node 8 is given as: 1, 2, 4, and 8.

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 of a tree 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.

Binary Trees…



Levels in binary tree

Level number :

- Every node in the binary tree is assigned a *level number*.
- The root node is defined to be at level 0.
- The left and the right child of the root node have a level number 1.
- Similarly, every node is at one level higher than its parents. So all child nodes are defined to have level number as parent's level number + 1.

In-degree/out-degree of a node:

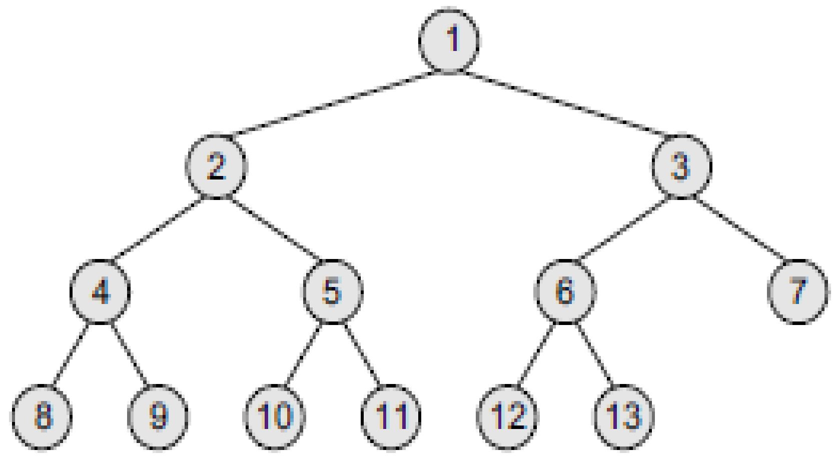
- It is the number of edges arriving at a node.
- The root node is the only node that has an in-degree equal to zero.
- Similarly, *out-degree* of a node is the number of edges leaving that node.

"Binary trees are commonly used to implement binary search trees, expression trees, tournament trees, and binary heaps."

Binary Trees…

- A binary tree of height ‘h’ has at least ‘h’ nodes and at most (2^h – 1) nodes. This is because every level will have at least one node and can have at most 2 nodes.
- So, if every level has two nodes then a tree with height ‘h’ will have at the most (2^h – 1) nodes as at level 0, there is only one element called the root.
- The height of a binary tree with ‘n’ nodes is at least ‘log2(n+1)’ and at most ‘n’.

Complete Binary Trees



Complete binary tree

- A *complete binary tree* is a binary tree that satisfies two properties.
- First, in a complete binary tree, every level, except possibly the last, is completely filled.
- Second, all nodes appear as far left as possible.

- All levels, except possibly the last one, are completely filled with nodes.
- Nodes in the last level are filled from left to right, making it a left-skewed structure.
- A complete binary tree can be efficiently represented using an array.

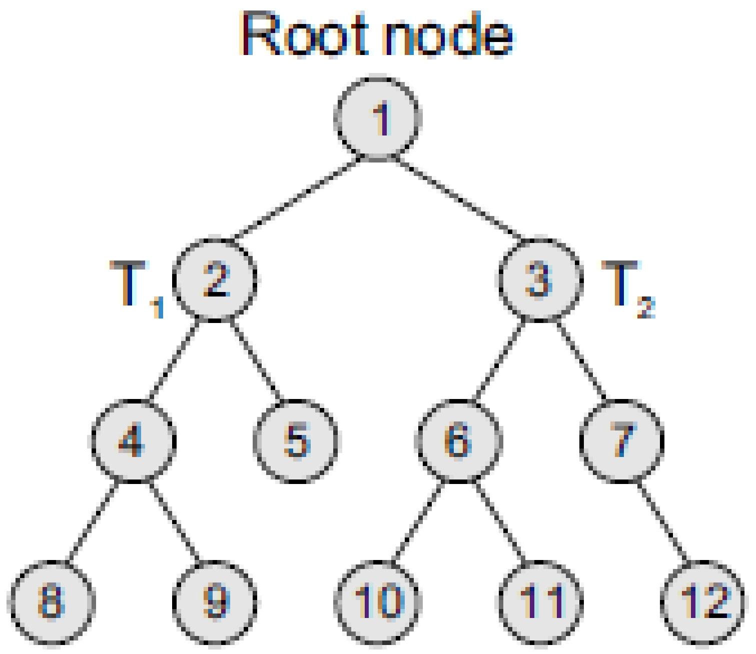
Linked Representation of Binary Trees

- In the linked representation of a binary tree, every node will have three parts: the data element, a pointer to the left node, and a pointer to the right node.
- So in C, the binary tree is built with a node type given below.

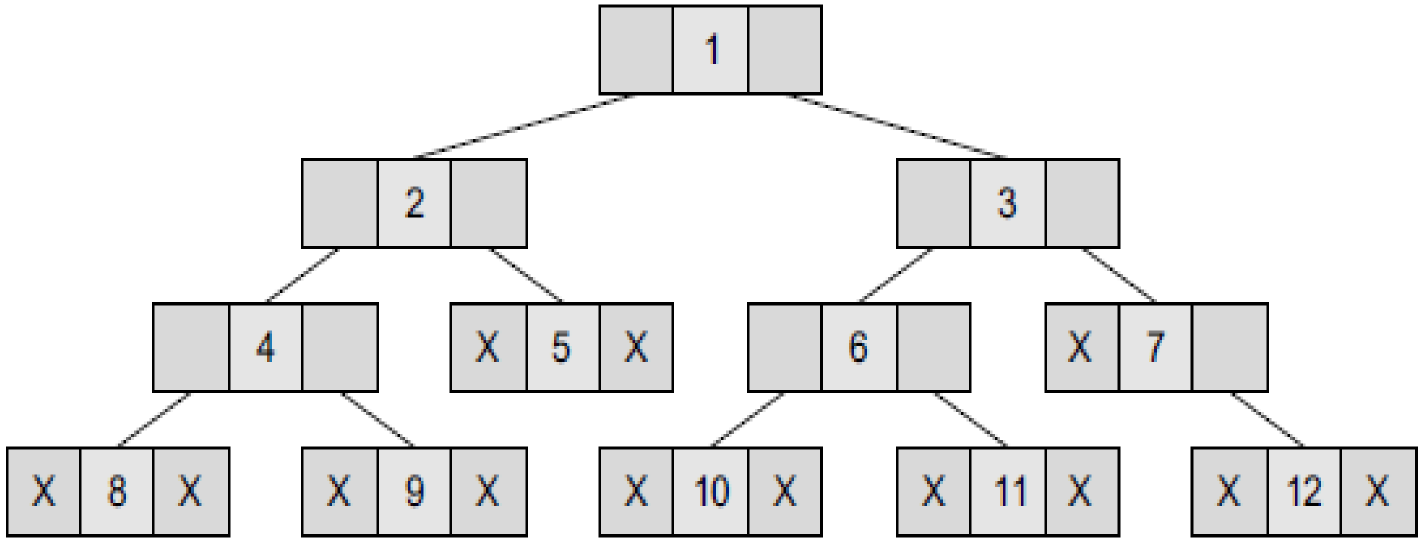
```
struct node {
        struct node *left;
        int data;
        struct node *right;
        };
```

- Every binary tree has a pointer ROOT, which points to the root element (topmost element) of the tree. If ROOT = NULL, then the tree is empty.

Linked Representation of Binary Trees…

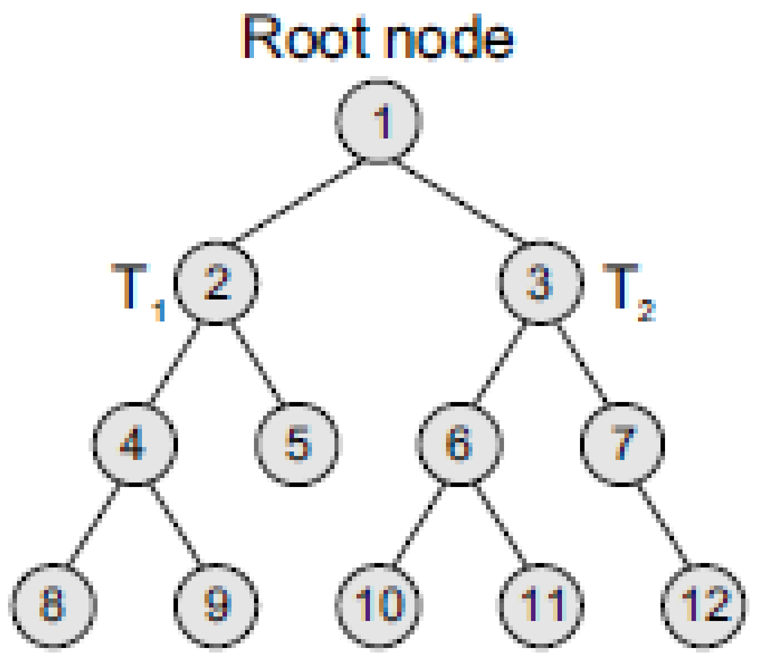


Binary tree

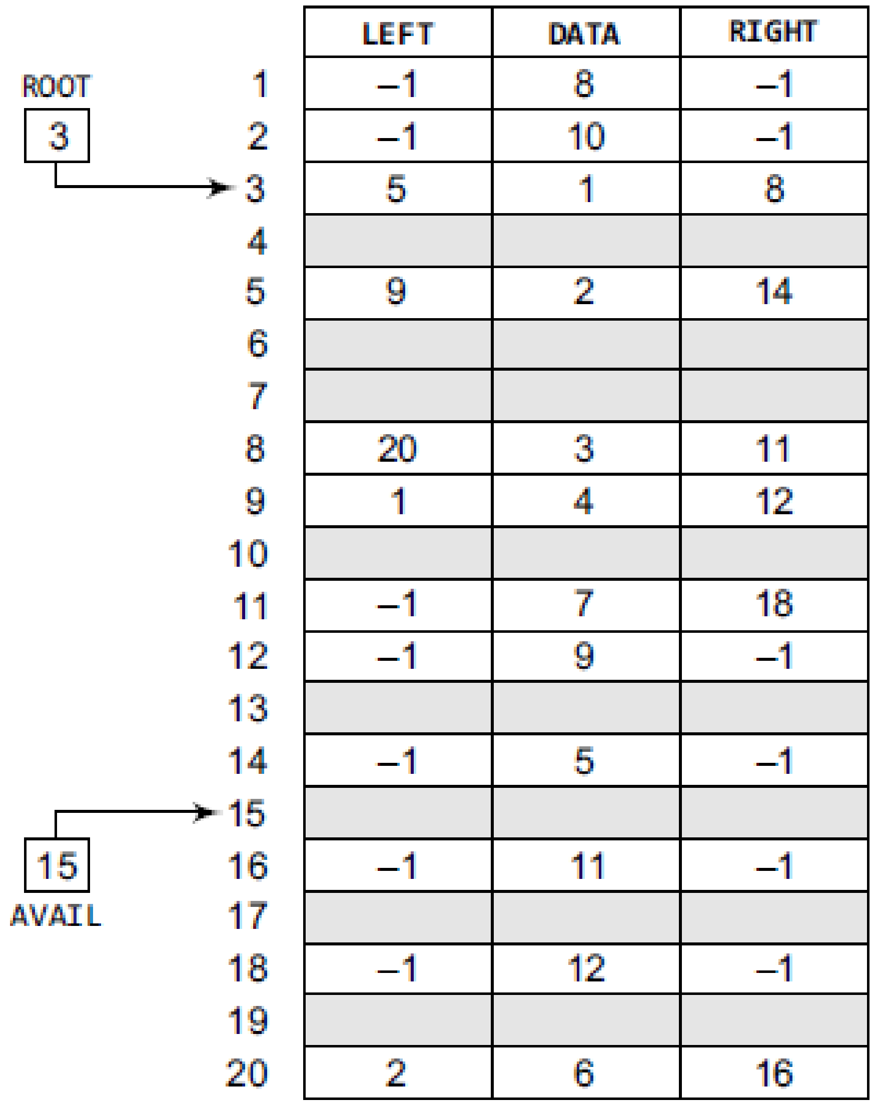


Linked representation of a binary tree

Linked Representation of Binary Trees…



Binary tree



	LEFT	DATA	RIGHT
1	–1	8	–1
2	–1	10	–1
3	5	1	8
4			
5	9	2	14
6			
7			
8	20	3	11
9	1	4	12
10			
11	–1	7	18
12	–1	9	–1
13			
14	–1	5	–1
15			
16	–1	11	–1
17			
18	–1	12	–1
19			
20	2	6	16

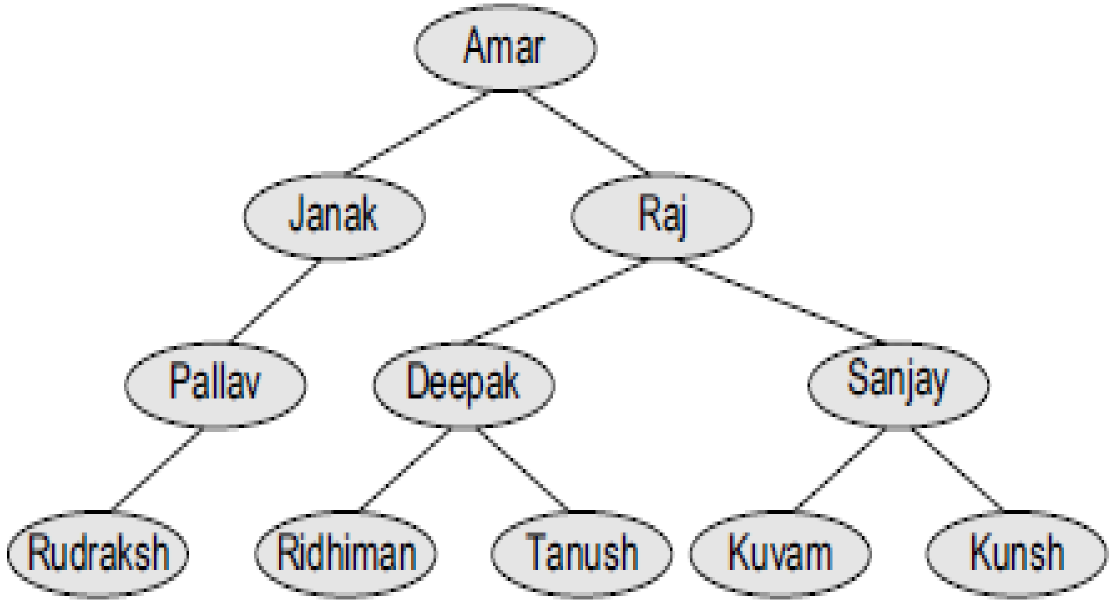
Linked representation of binary tree T in memory

- Given the memory representation of a tree that stores the names of family members, construct the corresponding tree from the given data.

ROOT 3

AVAIL 7

	LEFT	NAMES	RIGHT
1	12	Pallav	–1
2			
3	9	Amar	13
4			
5			
6	19	Deepak	17
7			
8			
9	1	Janak	–1
10			
11	–1	Kuvam	–1
12	–1	Rudraksh	–1
13	6	Raj	20
14			
15	–1	Kunsh	–1
16			
17	–1	Tanush	–1
18			
19	–1	Ridhiman	–1
20	11	Sanjay	15

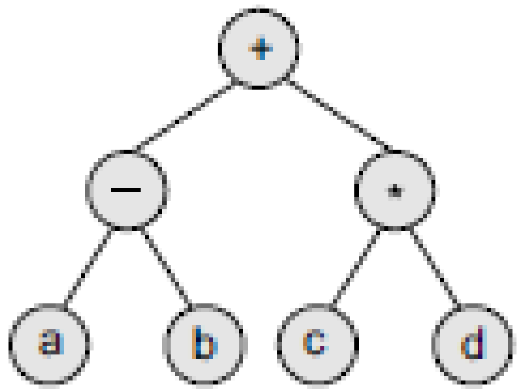


Binary Search Trees

- A binary search tree, also known as an ordered binary tree, is a variant of binary tree in which the nodes are arranged in an order.

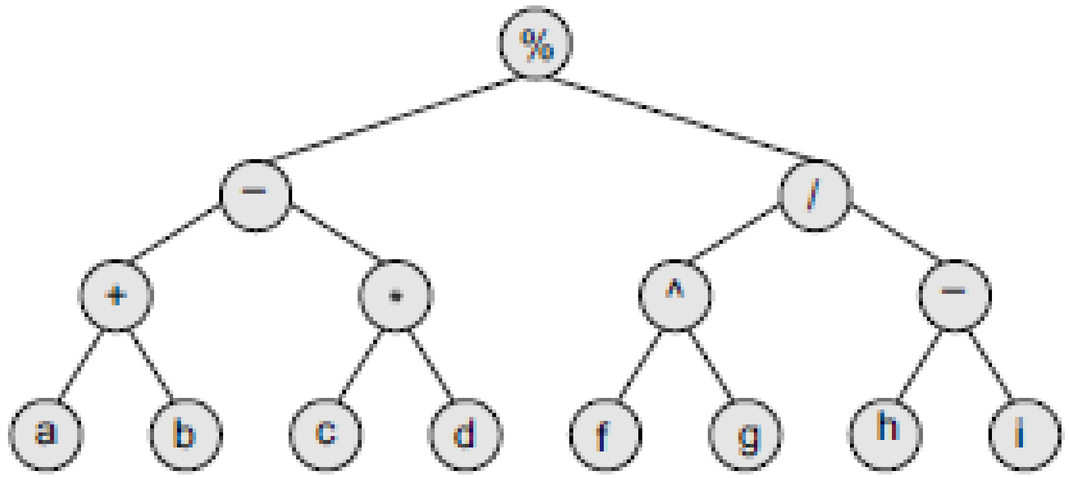
Expression Trees

- Binary trees are widely used to store algebraic expressions. For example,
- consider the algebraic expression given as:
- Exp = (a – b) + (c * d)



Example

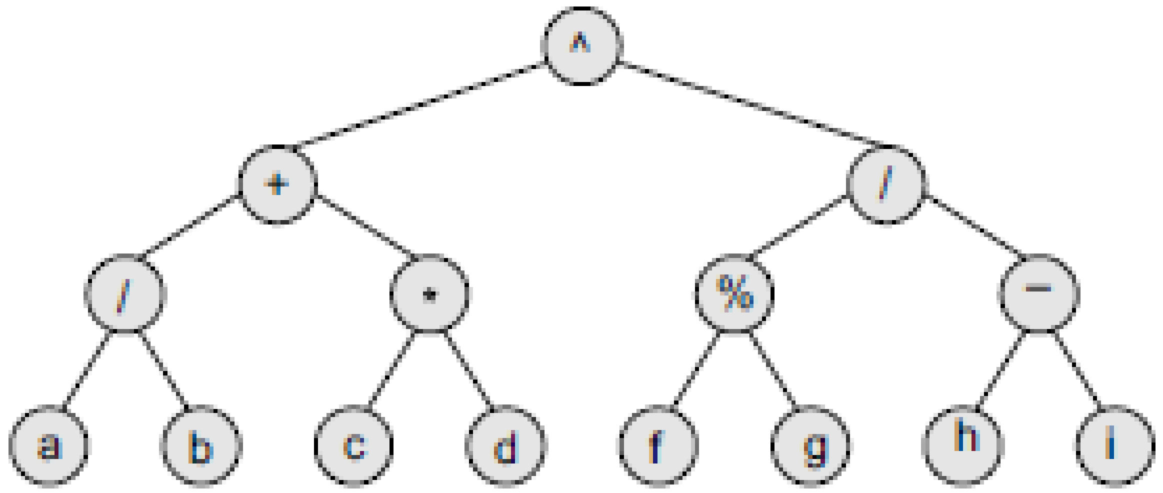
- Given an expression, Exp = ((a + b) – (c * d)) % ((e ^f) / (g – h)), construct the corresponding binary tree.



Expression tree

Example

- Given the binary tree, write down the expression that it represents.



Expression for the above binary tree is
[{(a/b) + (c*d)} ^ {(f % g)/(h – i)}]

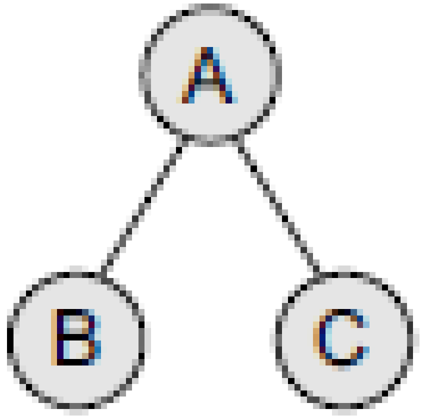
Creating Binary tree from general tree

TRAVERSING A BINARY TREE

- Traversing a binary tree is the process of visiting each node in the tree exactly once in a systematic way.
- Unlike linear data structures in which the elements are traversed sequentially, tree is a nonlinear data structure in which the elements can be traversed in many different ways.
- There are different algorithms for tree traversals.
  - Pre-order Traversal
  - In-order Traversal
  - Post-order Traversal

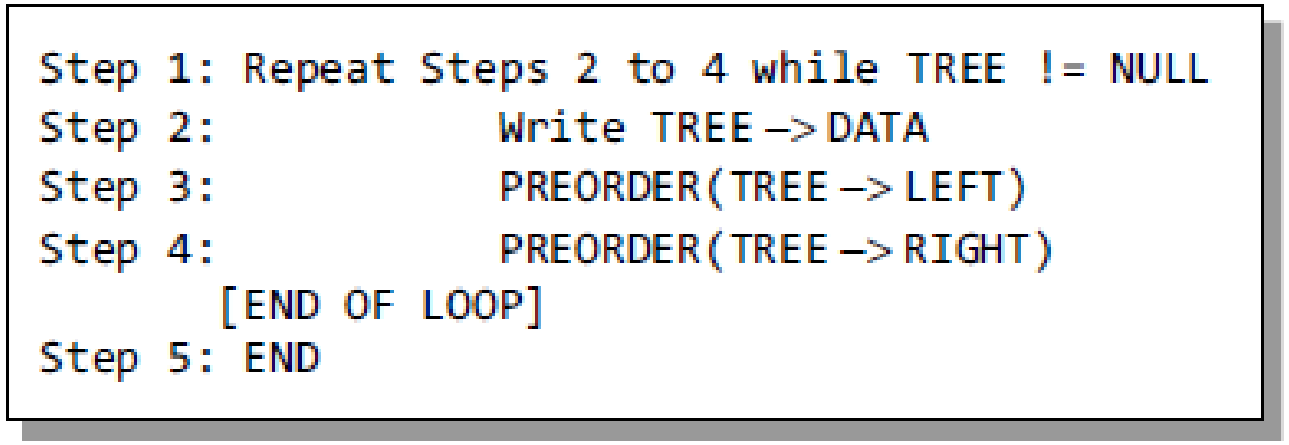
Pre-order Traversal

- To traverse a non-empty binary tree in pre-order, the following operations are performed recursively at each node. The algorithm works by:
  - 1. Visiting the root node,
  - 2. Traversing the left sub-tree, and finally
  - 3. Traversing the right sub-tree.



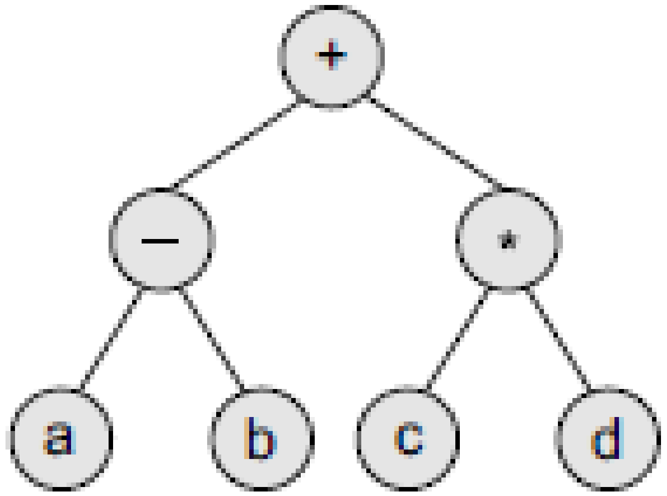
- The pre-order traversal of the tree is given as A, B, C.
- Root node first, the left sub-tree next, and then the right sub-tree.
- Pre-order traversal is also called as ***depth-first traversal*.**

Algorithm



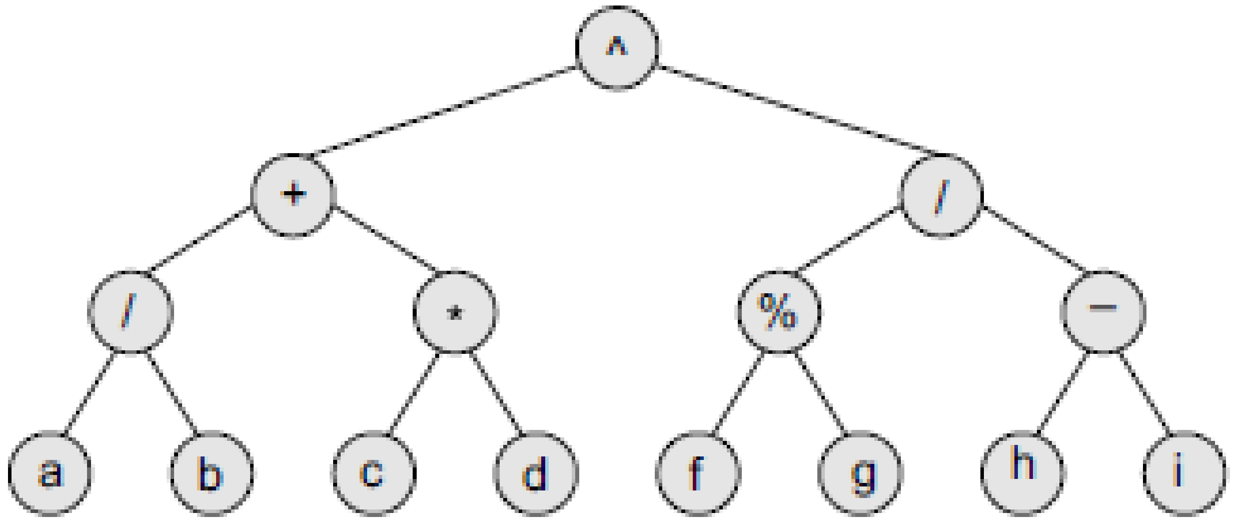
```
Step 1: Repeat Steps 2 to 4 while TREE != NULL
Step 2:             Write TREE -> DATA
Step 3:             PREORDER(TREE -> LEFT)
Step 4:             PREORDER(TREE -> RIGHT)
        [END OF LOOP]
Step 5: END
```

- When we traverse the elements of a tree using the pre-order traversal algorithm, the expression that we get is a prefix expression.



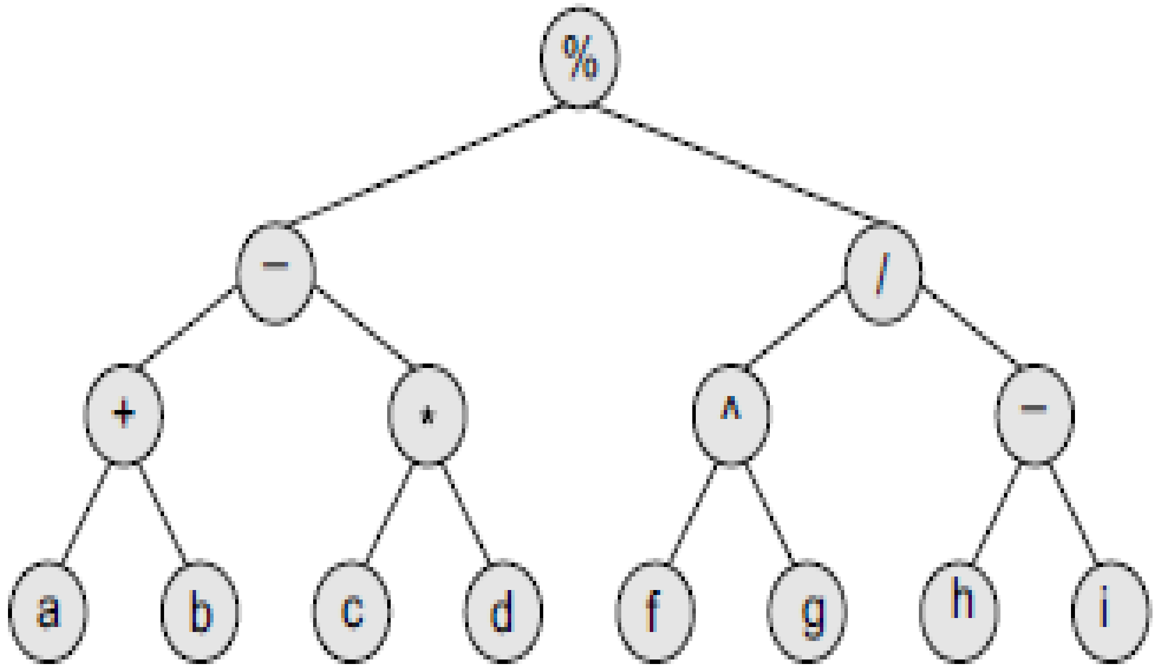
Pre-order traversal: + – a b * c d

Find Pre-order traversal:



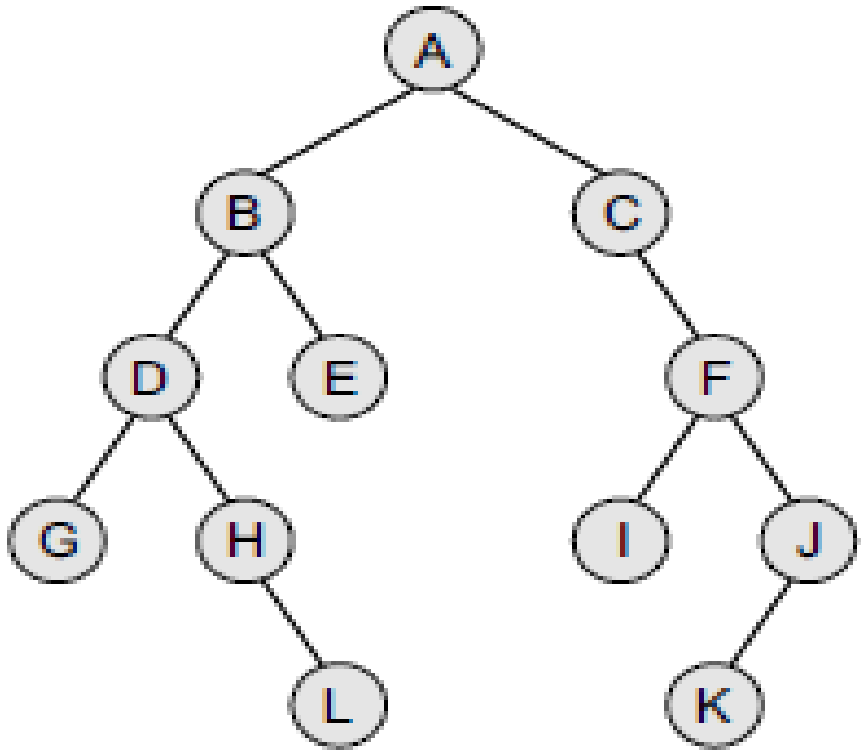
Pre-order: ^ + / a b * c d / % f g – h i

Find Pre-order traversal:



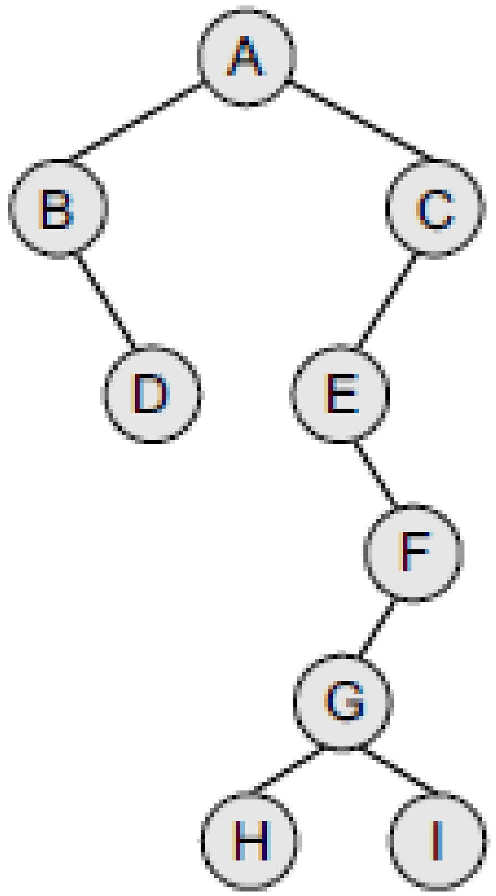
Pre-order: % – + a b * c d / ^ f g – h i

Find Pre-order Traversal:



PRE-ORDER TRAVERSAL: A, B, D, G, H, L, E, C, F, I, J, K

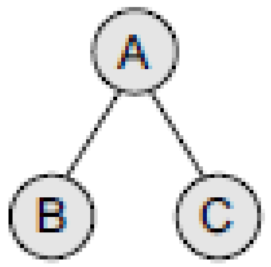
Find Pre-order Traversal:



PRE-ORDER TRAVERSAL: A, B, D, C, D, E, F, G, H, I

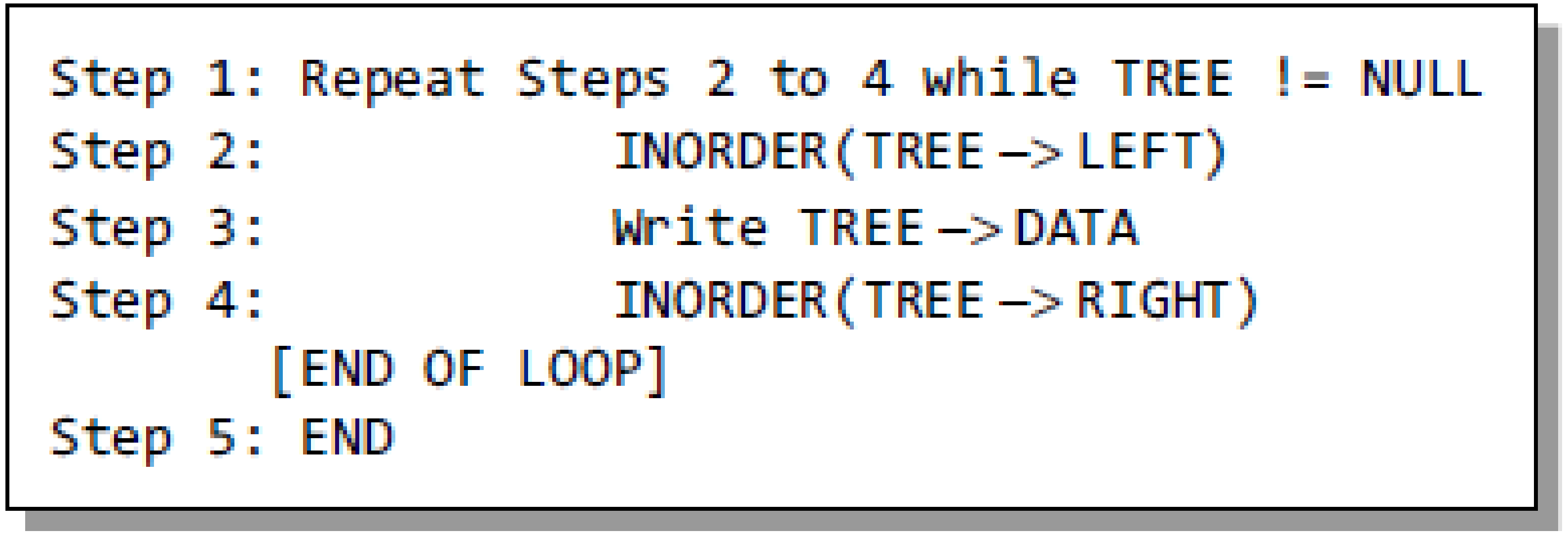
In-order Traversal

- To traverse a non-empty binary tree in in-order, the following operations are performed recursively at each node. The algorithm works by:
  1. Traversing the left sub-tree,
  2. Visiting the root node, and finally
  3. Traversing the right sub-tree.



The in-order traversal of the tree is given as B, A, and C.

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
```

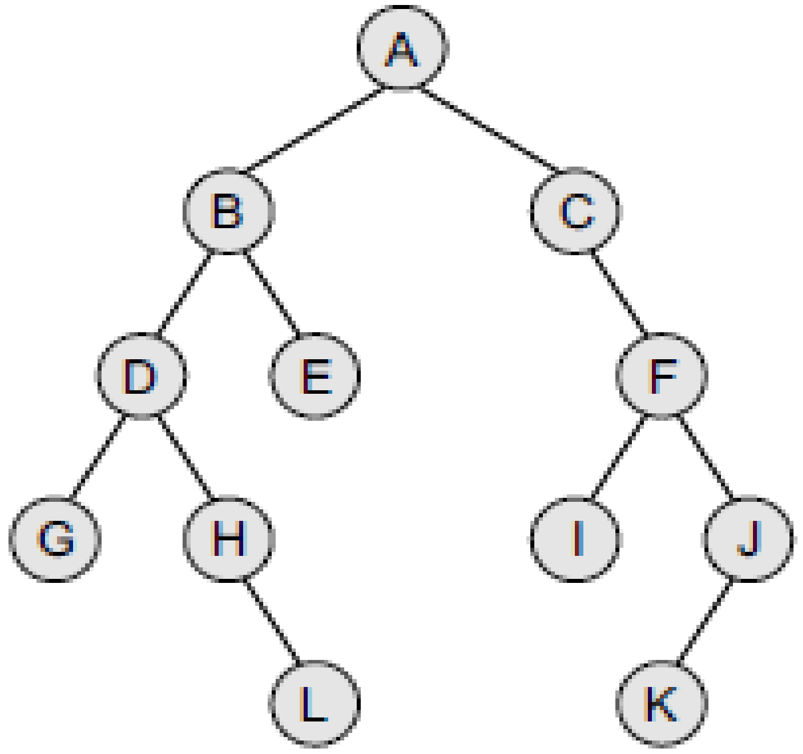
SOMAIYA
VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering
TRUST

In-order Traversal…

- In-order traversal is also called as *symmetric traversal*.
- In this algorithm, the left sub-tree is always traversed before the root node and the right sub-tree.
- The word ‘in’ in the in-order specifies that the root node is accessed in between the left and the right sub-trees.
- In-order algorithm is also known as the LNR traversal algorithm (Left-Node-Right).

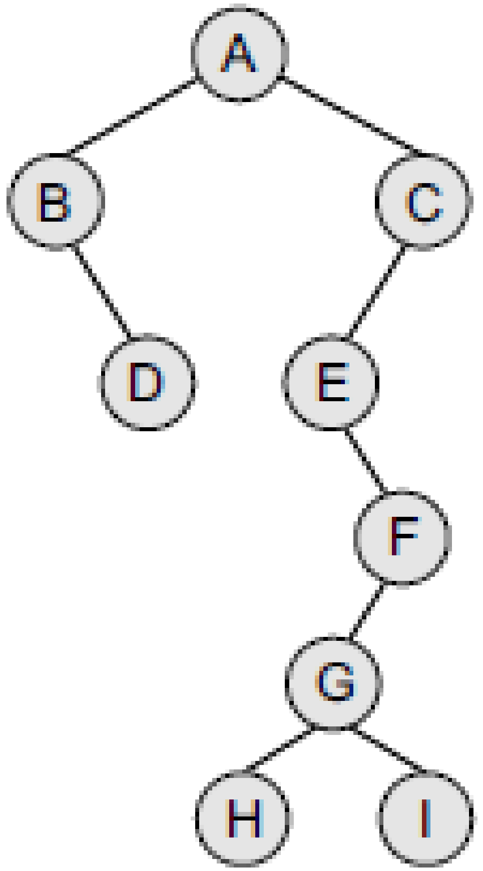
Example

- Find the sequence of nodes that will be visited using in-order traversal algorithm.



IN-ORDER TRAVERSAL:
G, D, H, L, B, E, A, C, I, F, K, and J

Example

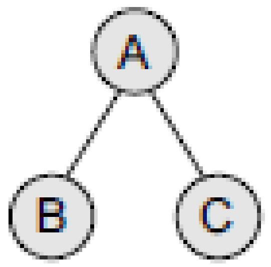


IN-ORDER TRAVERSAL:
B, D, A, E, H, G, I, F, and C

Post-order Traversal

- To traverse a non-empty binary tree in post-order, the following operations are performed recursively at each node. The algorithm works by:

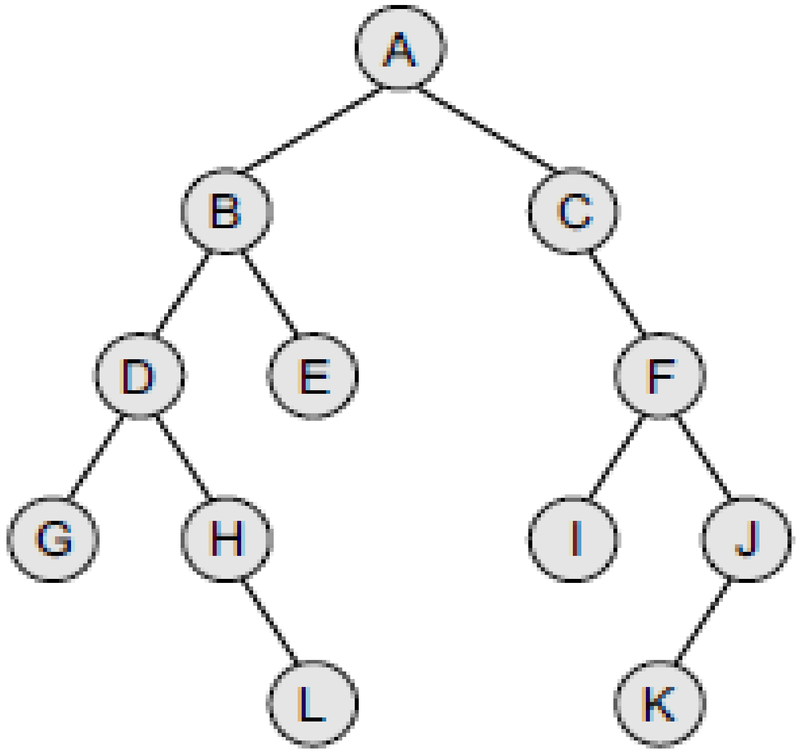
  1. Traversing the left sub-tree,
  2. Traversing the right sub-tree, and finally
  3. Visiting the root node.



The Post-order traversal of the tree is given as B, C, and A.

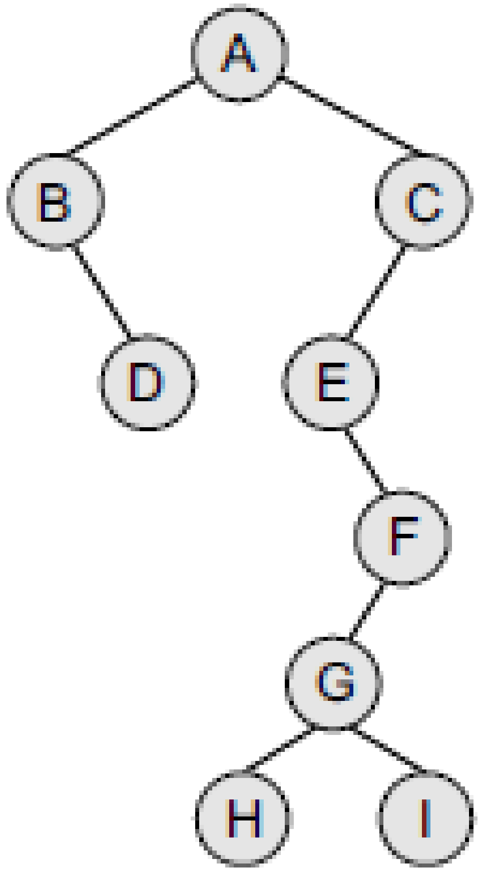
Example

- Find the sequence of nodes that will be visited using Post-order traversal algorithm.



POST-ORDER TRAVERSAL:
G, L, H, D, E, B, I, K, J, F, C,A

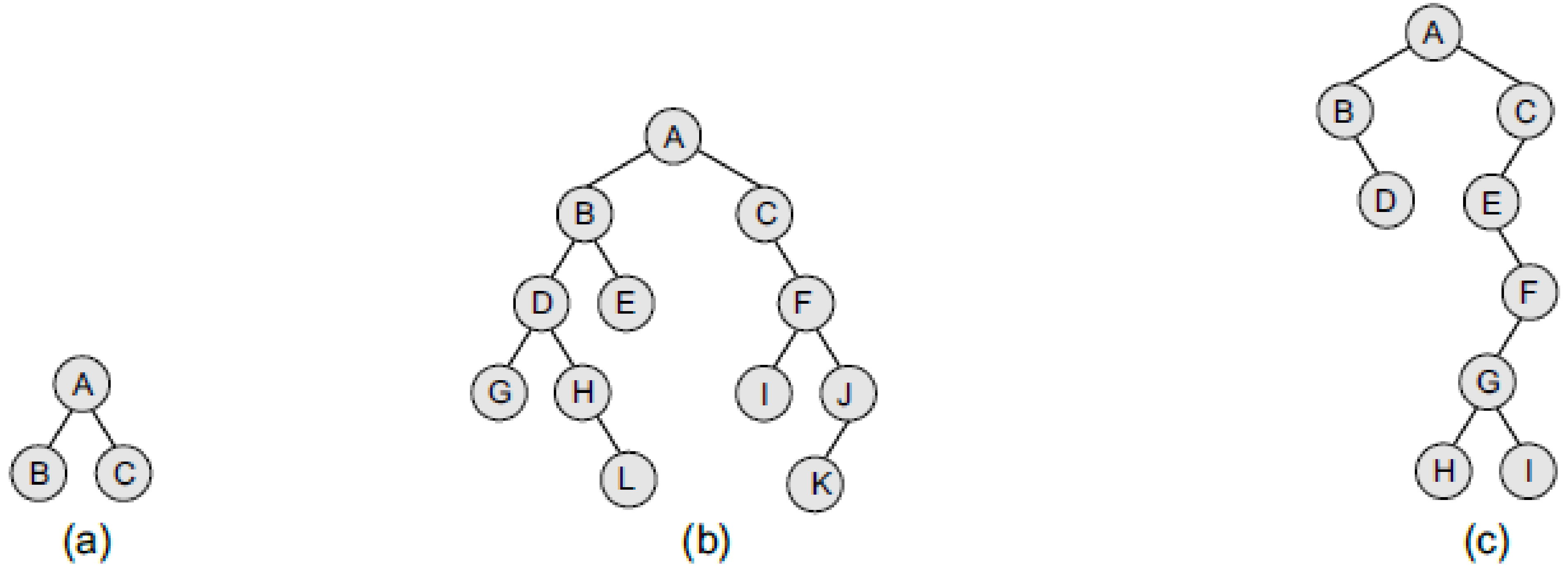
Example



POST-ORDER TRAVERSAL:
D, B, H, I, G, F, E, C, A

Level-order Traversal

- In level-order traversal, all the nodes at a level are accessed before going to the next level.
- This algorithm is also called as the *breadth-first traversal algorithm*.



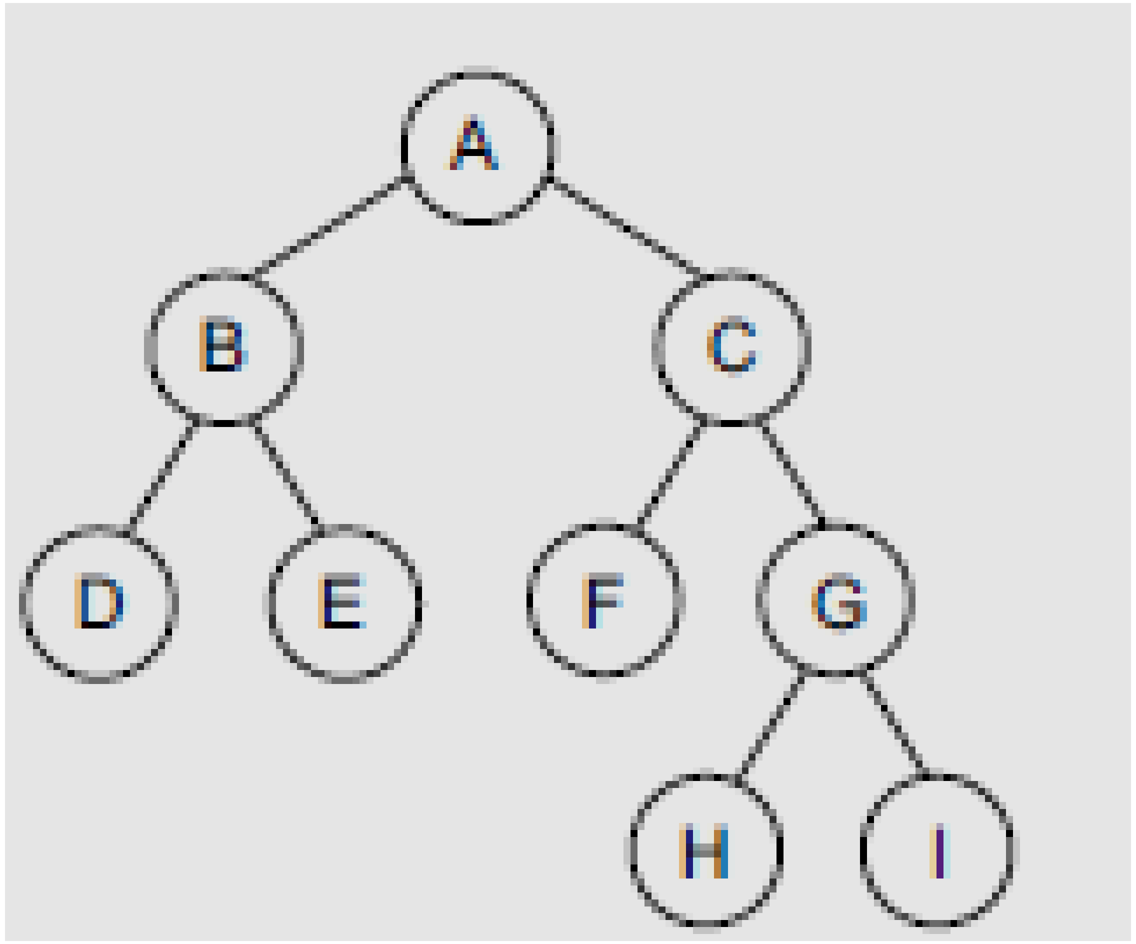
```
TRAVERSAL ORDER:
A, B, and C
```

```
TRAVERSAL ORDER:
A, B, C, D, E, F, G, H, I, J, L, and K
```

```
TRAVERSAL ORDER:
A, B, C, D, E, F, G, H, and I
```

Example

- Find the in-order, pre-order, post-order, and level-order traversal



Constructing a Binary Tree from Traversal Results

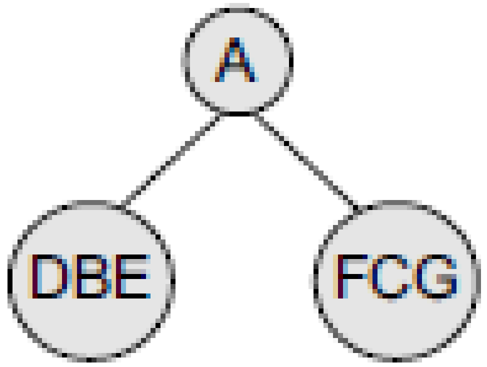
- We can construct a binary tree if we are given at least two traversal results.
- The first traversal must be the in-order traversal and the second can be either pre-order or post-order traversal.
- The in-order traversal result will be used to determine the left and the right child nodes, and the pre-order/post-order can be used to determine the root node.

Example

- Construct a Binary Tree from Traversal Results-

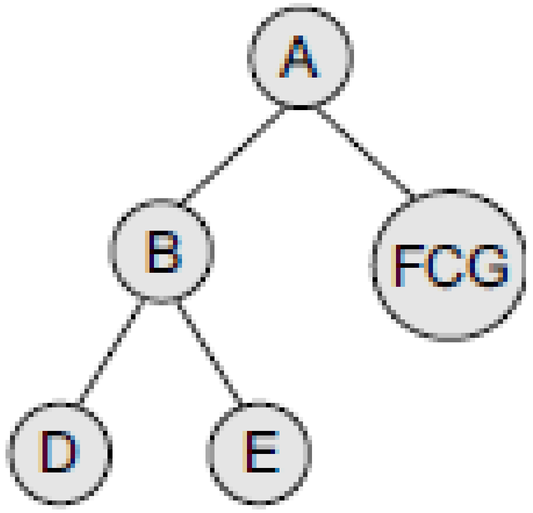
In-order Traversal: **D B E A F C G** and Pre-order Traversal: **A B D E C F G**

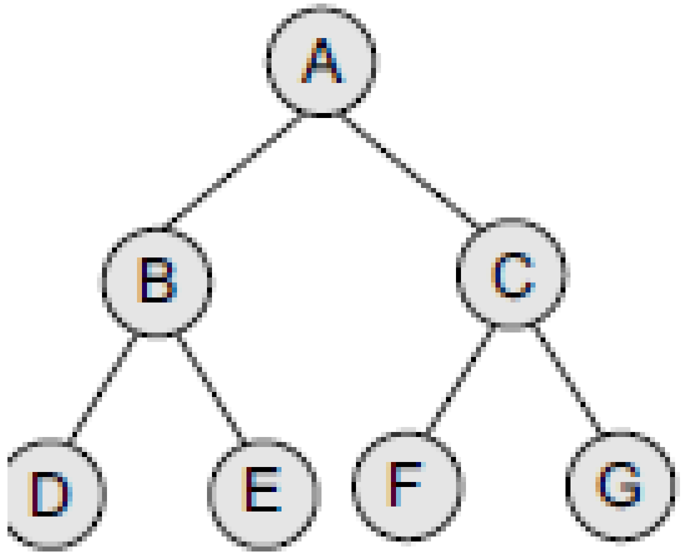
Step 1 Use the pre-order sequence to determine the root node of the tree. The first element would be the root node.
Step 2 Elements on the left side of the root node in the in-order traversal sequence form the left sub-tree of the root node. Similarly, elements on the right side of the root node in the in-order traversal sequence form the right sub-tree of the root node.



Example…

- In-order Traversal: **D B E A F C G** and Pre-order Traversal: **A B D E C F G**
- ***Step 3*** Recursively select each element from pre-order traversal sequence and create its left and right sub-trees from the in-order traversal sequence.





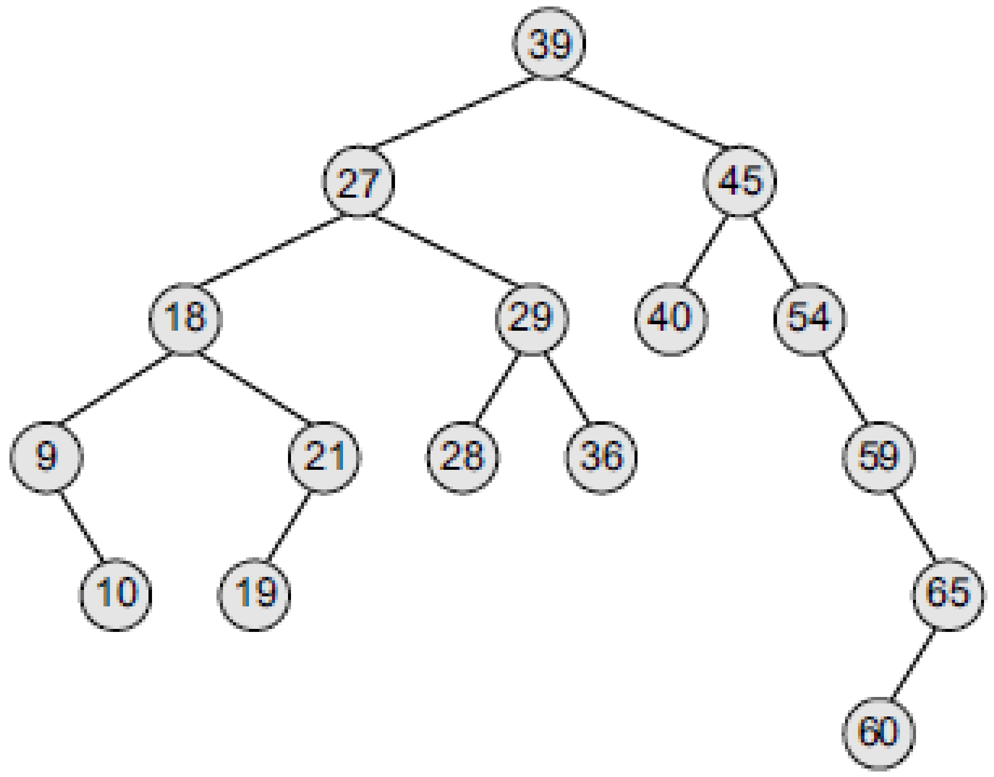
Example 2

- Construct a Binary Tree from Traversal Results-
- In–order Traversal: **D B H E I A F J C G**
- Post order Traversal: **D H I E B J F G C A**

Binary Search Trees(BST)

- A binary search tree, also known as an ordered binary tree, is a variant of binary trees in which the nodes are arranged in an order.
- In a binary search tree, all the nodes in the left sub-tree have a value less than that of the root node.
- Correspondingly, all the nodes in the right sub-tree have a value either equal to or greater than the root node.
- The same rule is applicable to every sub-tree in the tree.
- Since the nodes in a binary search tree are ordered, the time needed to search an element in the tree is greatly reduced. Whenever we search for an element, we do not need to traverse the entire tree.

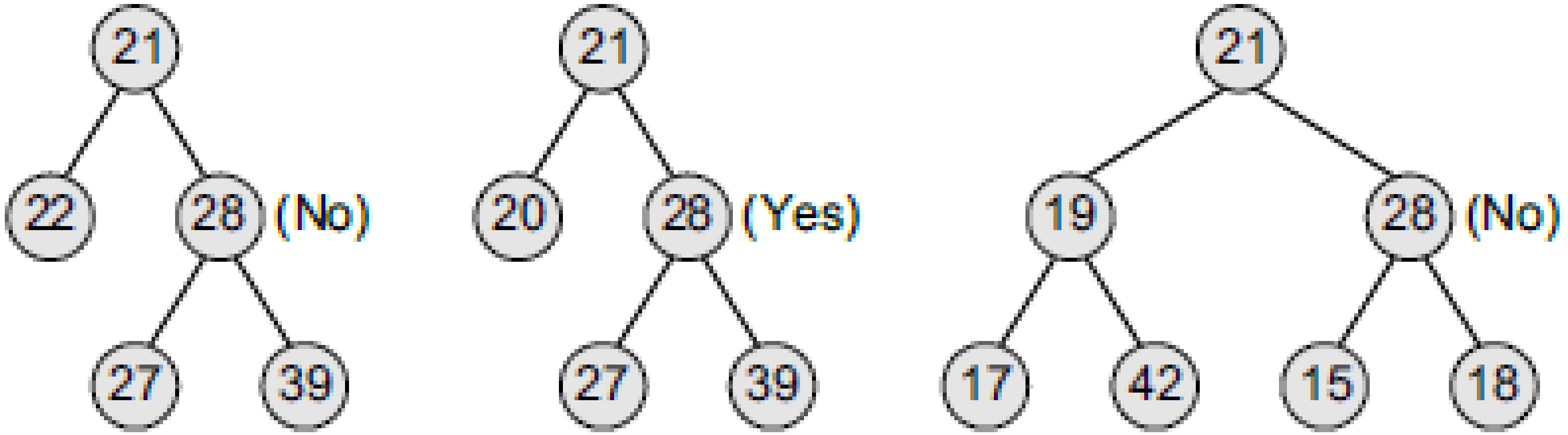
Binary Search Trees



- The root node is 39.
- The left sub-tree of the root node consists of nodes 9, 10, 18, 19, 21, 27, 28, 29, and 36. All these nodes have smaller values than the root node.
- The right sub-tree of the root node consists of nodes 40, 45, 54, 59, 60, and 65.
- Recursively, each of the sub-trees also obeys the binary search tree constraint.
- For example, in the left sub-tree of the root node, 27 is the root and all elements in its left sub-tree (9, 10, 18, 19, 21) are smaller than 27,
- while all nodes in its right sub-tree (28, 29, and 36) are greater than the root node's value.

Binary Search Tree…

State whether the binary trees in Figure are binary search trees or not.



Binary Search Tree…

- a binary search tree is a binary tree with the following properties:
  - The left sub-tree of a node N contains values that are less than N’s value.
  - The right sub-tree of a node N contains values that are greater than N’s value.
  - Both the left and the right binary trees also satisfy these properties and, thus, are binary search trees.

Example

- Create a binary search tree using the following data elements:

  45, 39, 56, 12, 34, 78, 32, 10, 89, 54, 67, 81

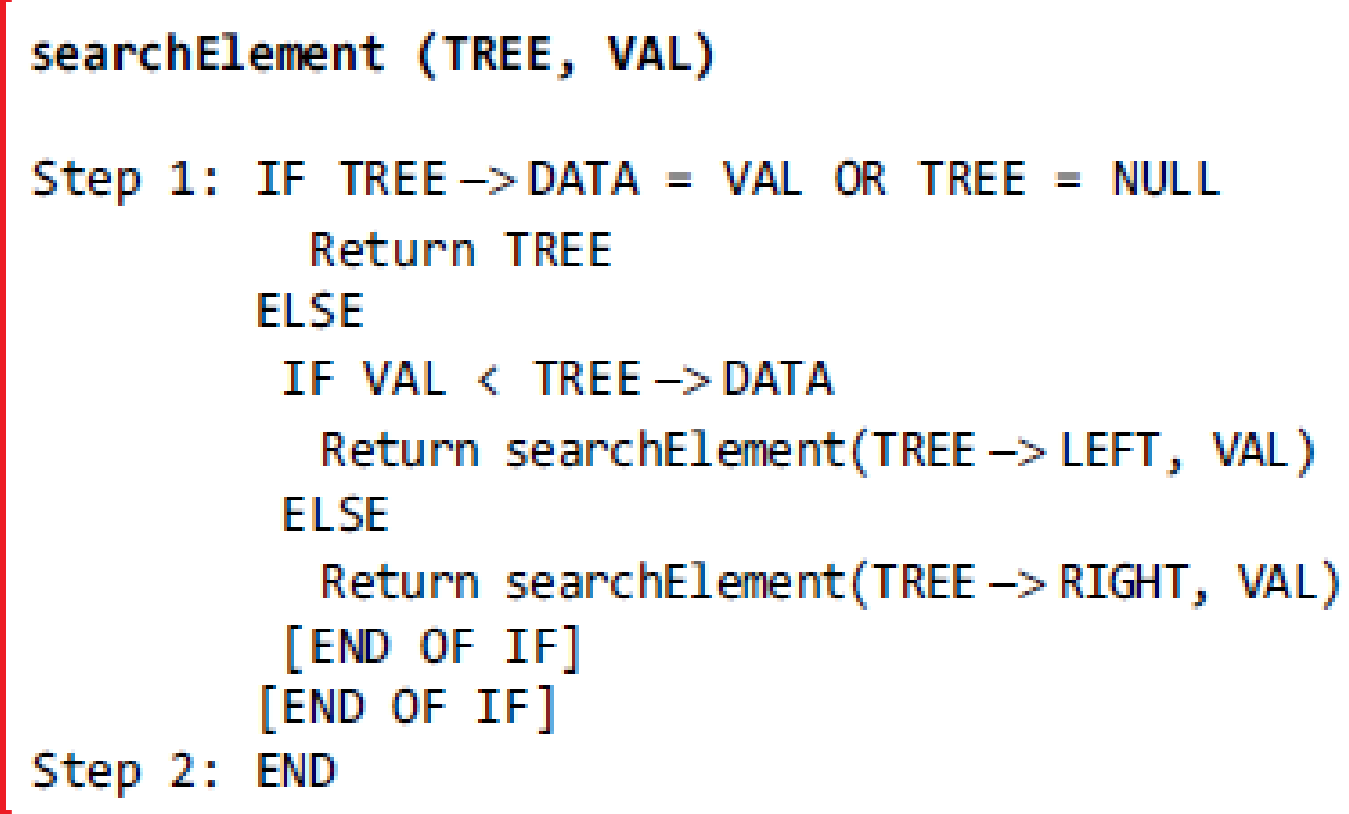
OPERATIONS ON BINARY SEARCH TREES

- Searching for a Node in a Binary Search Tree
- Inserting a New Node in a Binary Search Tree
- Deleting a Node from a Binary Search Tree

Searching for a Node in a Binary Search Tree

- The search function is used to find whether a given value is present in the tree or not.
- The searching process begins at the root node.
- The function first checks if the binary search tree is empty. If it is empty, then the value we are searching for is not present in the tree.
- if there are nodes in the tree, then the search function checks to see if the key value of the current node is equal to the value to be searched.
- If not, it checks if the value to be searched for is less than the value of the current node, in this case it should be recursively called on the left child node.
- In case the value is greater than the value of the current node, it should be recursively called on the right child node.

Algorithm



```
searchElement (TREE, VAL)

Step 1: IF TREE -> DATA = VAL OR TREE = NULL
             Return TREE
        ELSE
         IF VAL < TREE -> DATA
          Return searchElement(TREE -> LEFT, VAL)
         ELSE
          Return searchElement(TREE -> RIGHT, VAL)
         [END OF IF]
        [END OF IF]
Step 2: END
```

Step 1:

- we check if the value stored at the current node of TREE is equal to VAL or if the current node is NULL, then we return the current node of TREE.
- Otherwise, if the value stored at the current node is less than VAL, then the algorithm is recursively called on its right sub-tree,
- else the algorithm is called on its left sub-tree.

Step 2: end

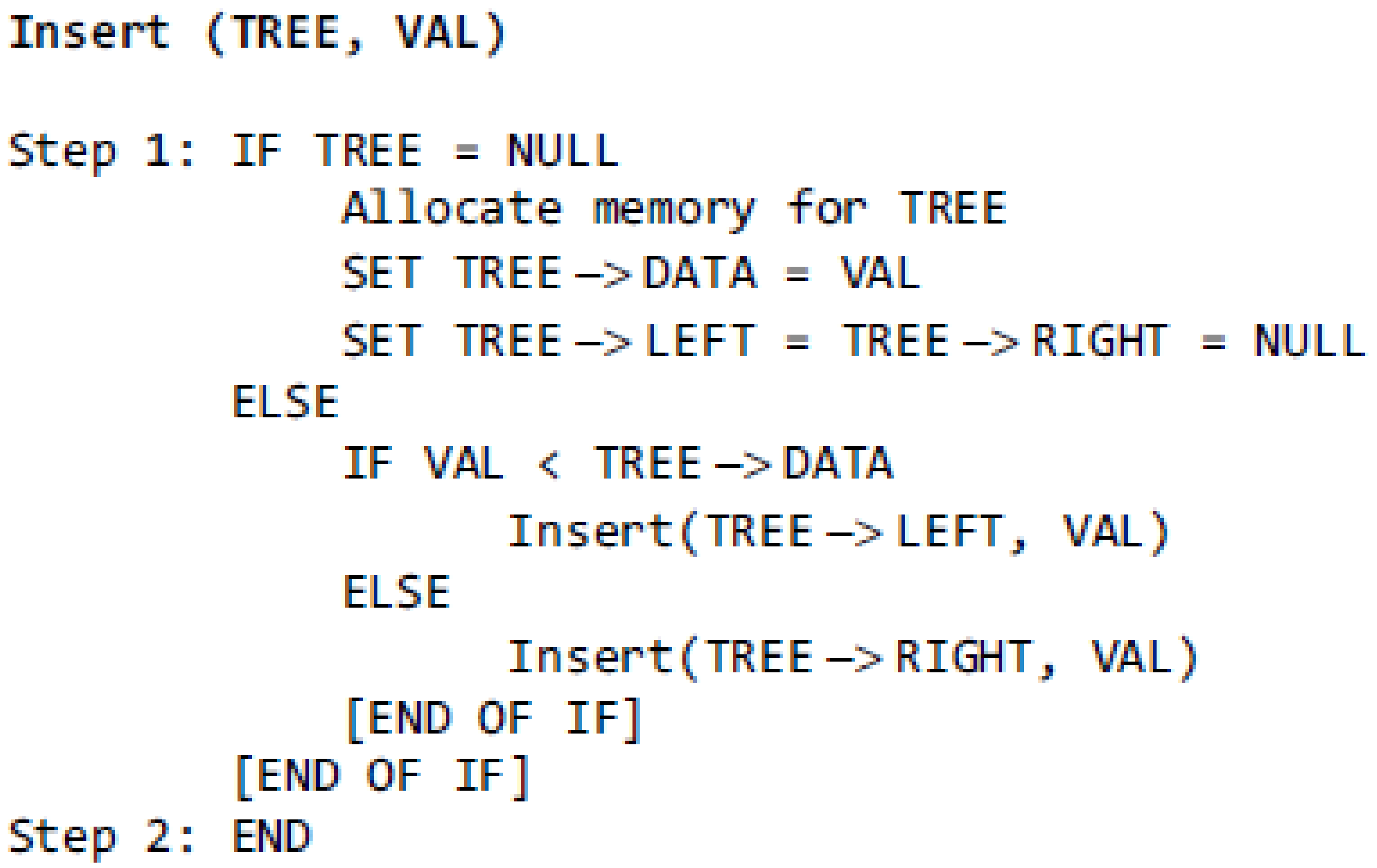
SOMAIYA
VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Somaiya
TRUST

Inserting a New Node in a Binary Search Tree

- The insert function is used to add a new node with a given value at the correct position in the binary search tree.
- Adding the node at the correct position means that the new node should not violate the properties of the binary search tree.
- The initial code for the insert function is similar to the search function. This is because we first find the correct position where the insertion has to be done and then add the node at that position.
- The insertion function changes the structure of the tree. Therefore, when the insert function is called recursively, the function should return the new tree pointer.

Algorithm



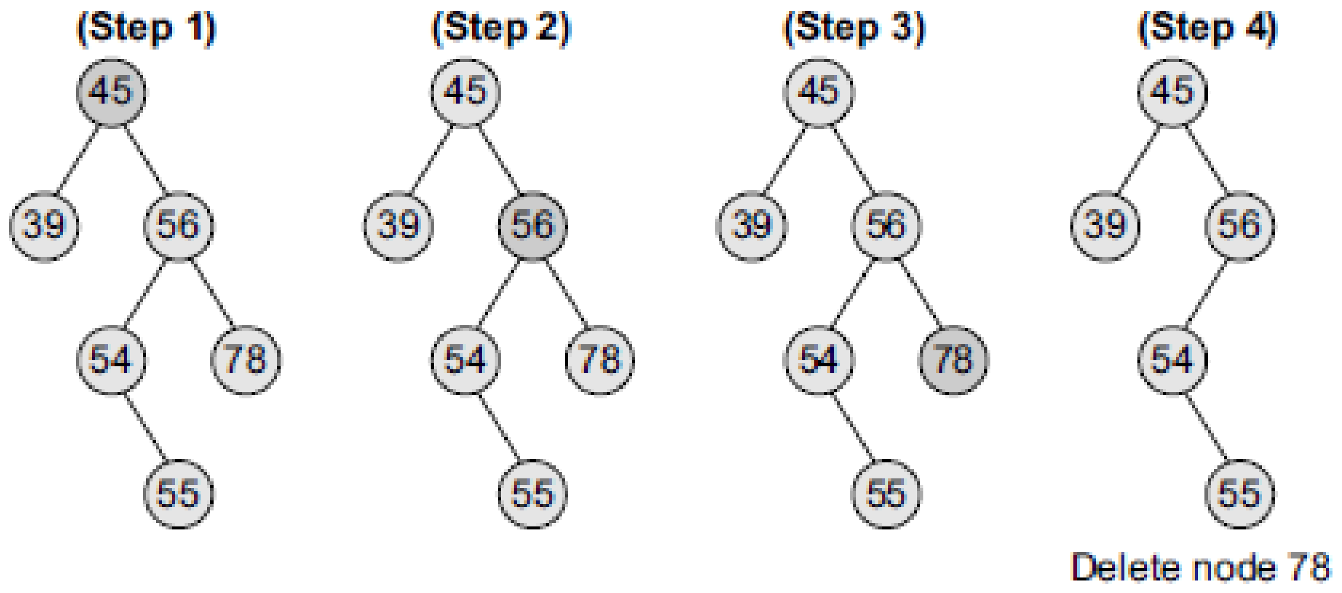
```
Insert (TREE, VAL)

Step 1: IF TREE = NULL
              Allocate memory for TREE
              SET TREE -> DATA = VAL
              SET TREE -> LEFT = TREE -> RIGHT = NULL
        ELSE
              IF VAL < TREE -> DATA
                     Insert(TREE -> LEFT, VAL)
              ELSE
                     Insert(TREE -> RIGHT, VAL)
              [END OF IF]
        [END OF IF]
Step 2: END
```

SOMAIYA
VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering
Somaiya
TRUST

Deleting a Node from a Binary Search Tree

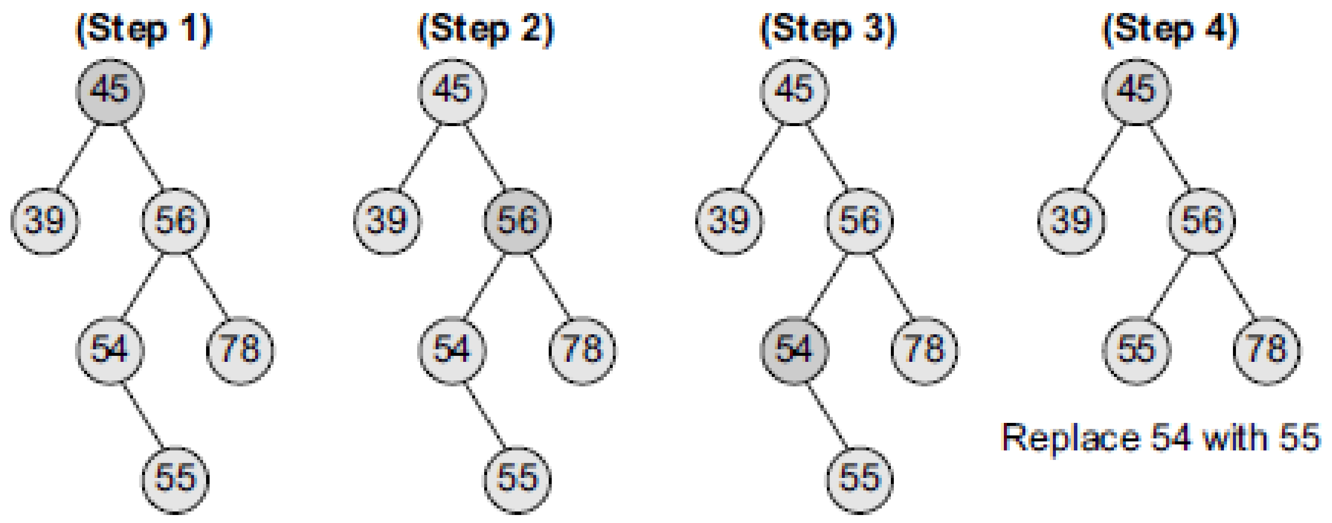
- The delete function deletes a node from the binary search tree.
- However, utmost care should be taken that the properties of the binary search tree are not violated and nodes are not lost in the process.
- ***Case 1: Deleting a Node that has No Children***
- ***Case 2: Deleting a Node with One Child***
- ***Case 3: Deleting a Node with Two Children***

Case 1: Deleting a Node that has No Children



If we have to delete node 78, we can simply remove this node without any issue. This is the simplest case of deletion.

Case 2: Deleting a Node with One Child



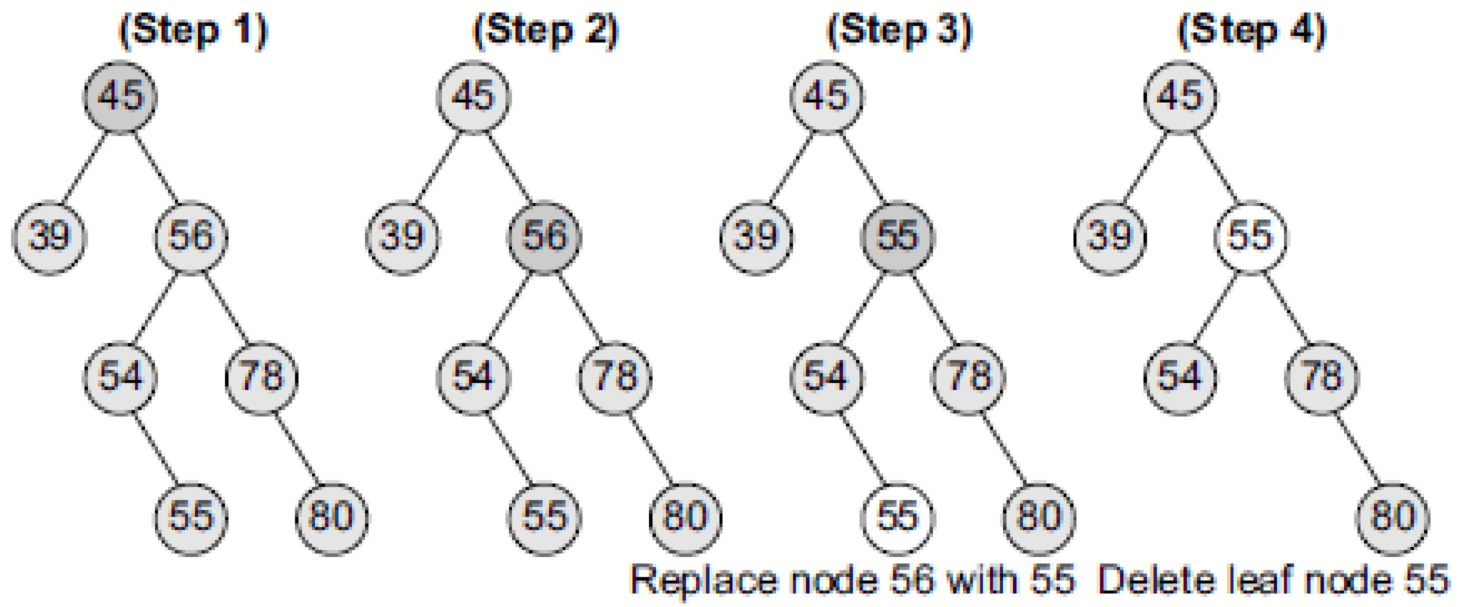
Deleting node 54 from the given binary search tree

- The node’s child is set as the child of the node’s parent. In other words, replace the node with its child.
- If the node is the left child of its parent, the node’s child becomes the left child of the node’s parent.
- Correspondingly, if the node is the right child of its parent, the node’s child becomes the right child of the node’s parent.

Case 3: Deleting a Node with Two Children

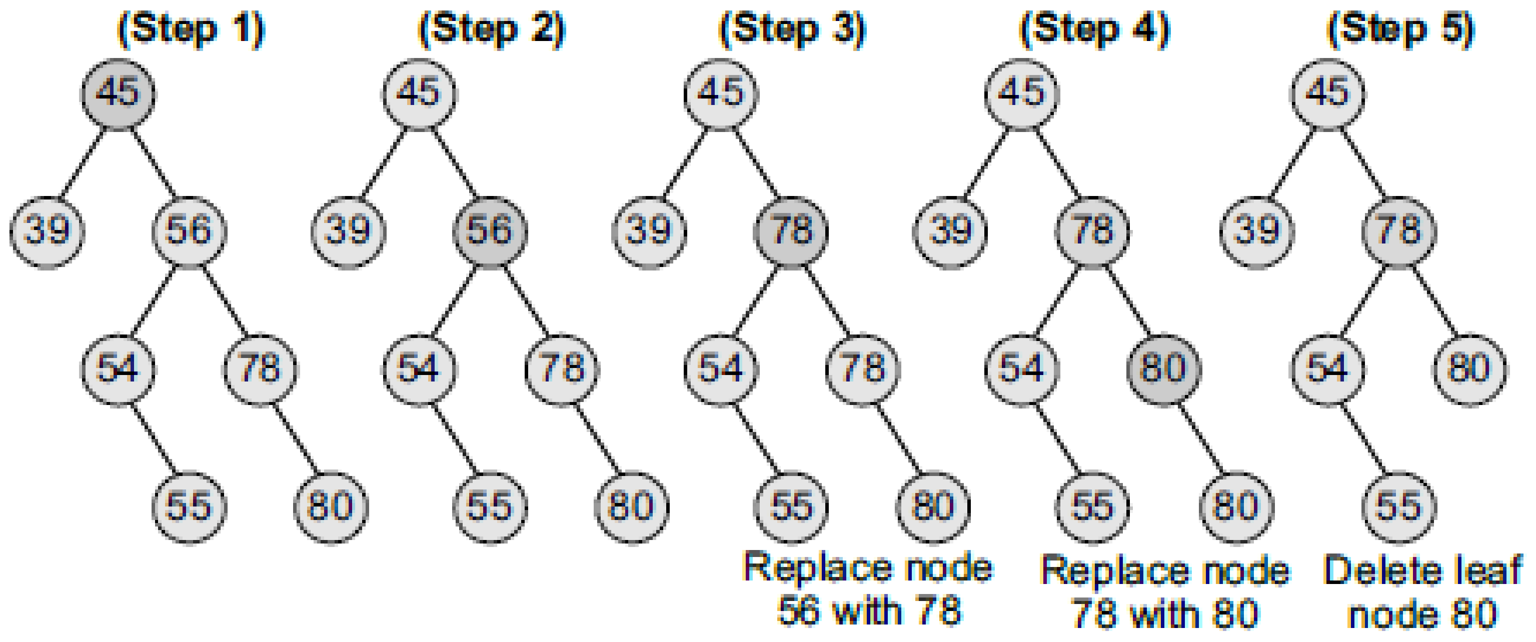
- To delete a node with two children replace the node’s value with its *in-order predecessor* (largest valu in the left sub-tree) or *in-order successor* (smallest value in the right sub-tree).
- The in-order predecessor or the successor can then be deleted using any of the cases: Deleting a node with no children OR Deleting a node with one children

Case 3: Deleting a Node with Two Children



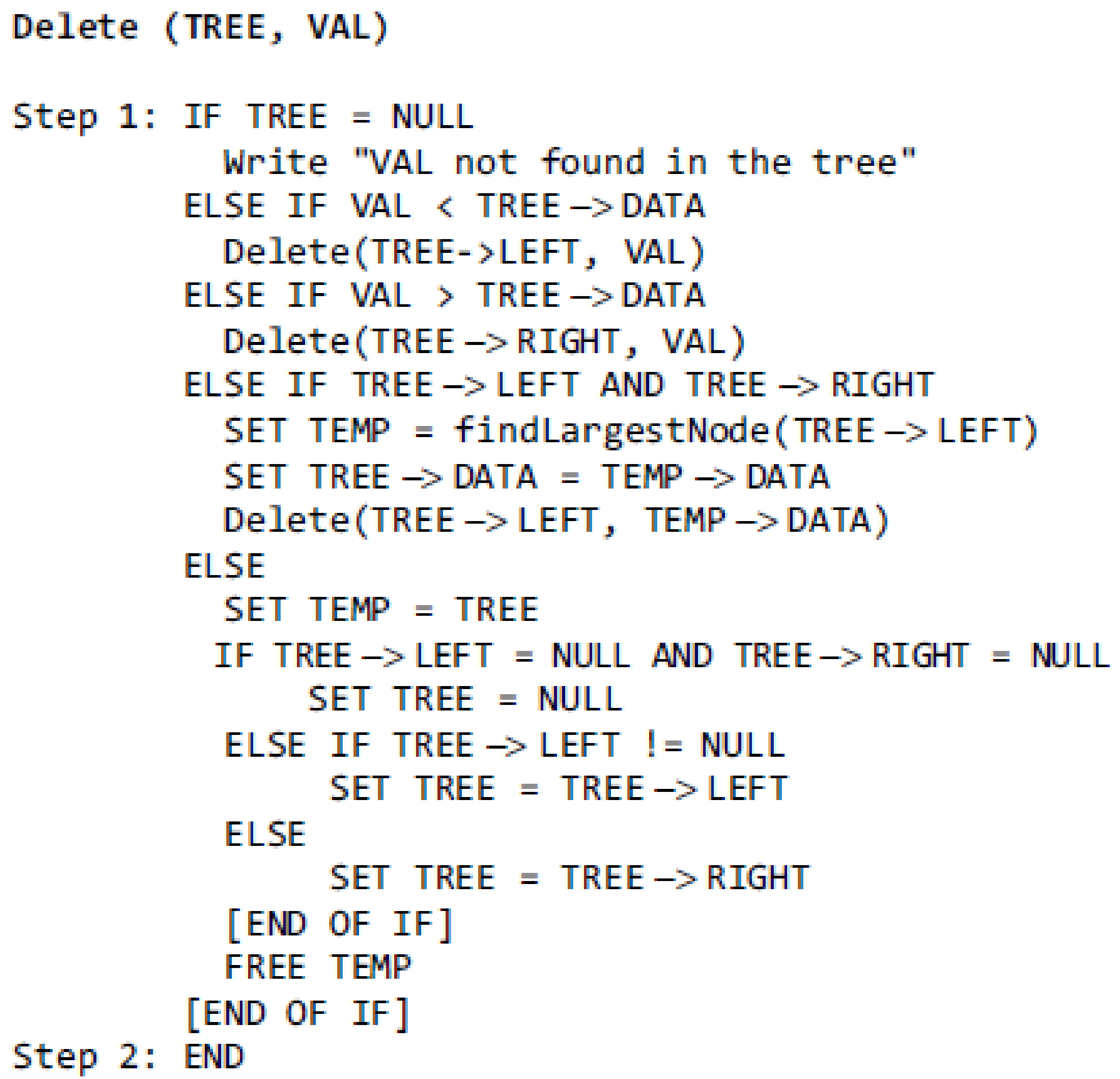
Deleting node 56 from the given binary search tree with its In-order Predecessor

Case 3: Deleting a Node with Two Children



Deleting node 56 from the given binary search tree with its In-order Successor

Algorithm



```
Delete (TREE, VAL)

Step 1: IF TREE = NULL
          Write "VAL not found in the tree"
        ELSE IF VAL < TREE -> DATA
          Delete(TREE->LEFT, VAL)
        ELSE IF VAL > TREE -> DATA
          Delete(TREE -> RIGHT, VAL)
        ELSE IF TREE -> LEFT AND TREE -> RIGHT
          SET TEMP = findLargestNode(TREE -> LEFT)
          SET TREE -> DATA = TEMP -> DATA
          Delete(TREE -> LEFT, TEMP -> DATA)
        ELSE
          SET TEMP = TREE
         IF TREE -> LEFT = NULL AND TREE -> RIGHT = NULL
              SET TREE = NULL
          ELSE IF TREE -> LEFT != NULL
               SET TREE = TREE -> LEFT
          ELSE
               SET TREE = TREE -> RIGHT
          [END OF IF]
          FREE TEMP
        [END OF IF]
Step 2: END
```

- if TREE=NULL, if it is true, then the node to be deleted is not present in the tree.

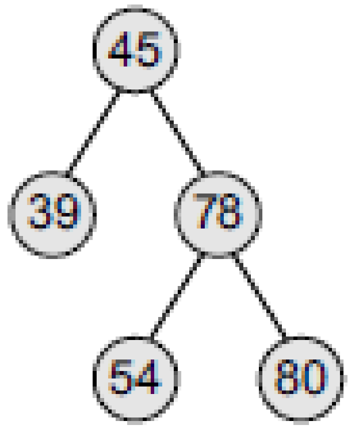
- if that is not the case, then we check if the value to be deleted is less than the current node's data.

- In case the value is less, we call the algorithm recursively on the node's left sub-tree, otherwise the algorithm is called recursively on the node's right sub-tree.

- if we have found the node whose value is equal to VAL, then we check which case of deletion it is.

Determining the Height of a Binary Search Tree

- To Calculate the height of the left sub-tree and the right sub-tree.
- Whichever height is greater, 1 is added to it.
- For example, if the height of the left sub-tree is greater than that of the right sub-tree, then 1 is added to the left sub-tree, else 1 is added to the right sub-tree.



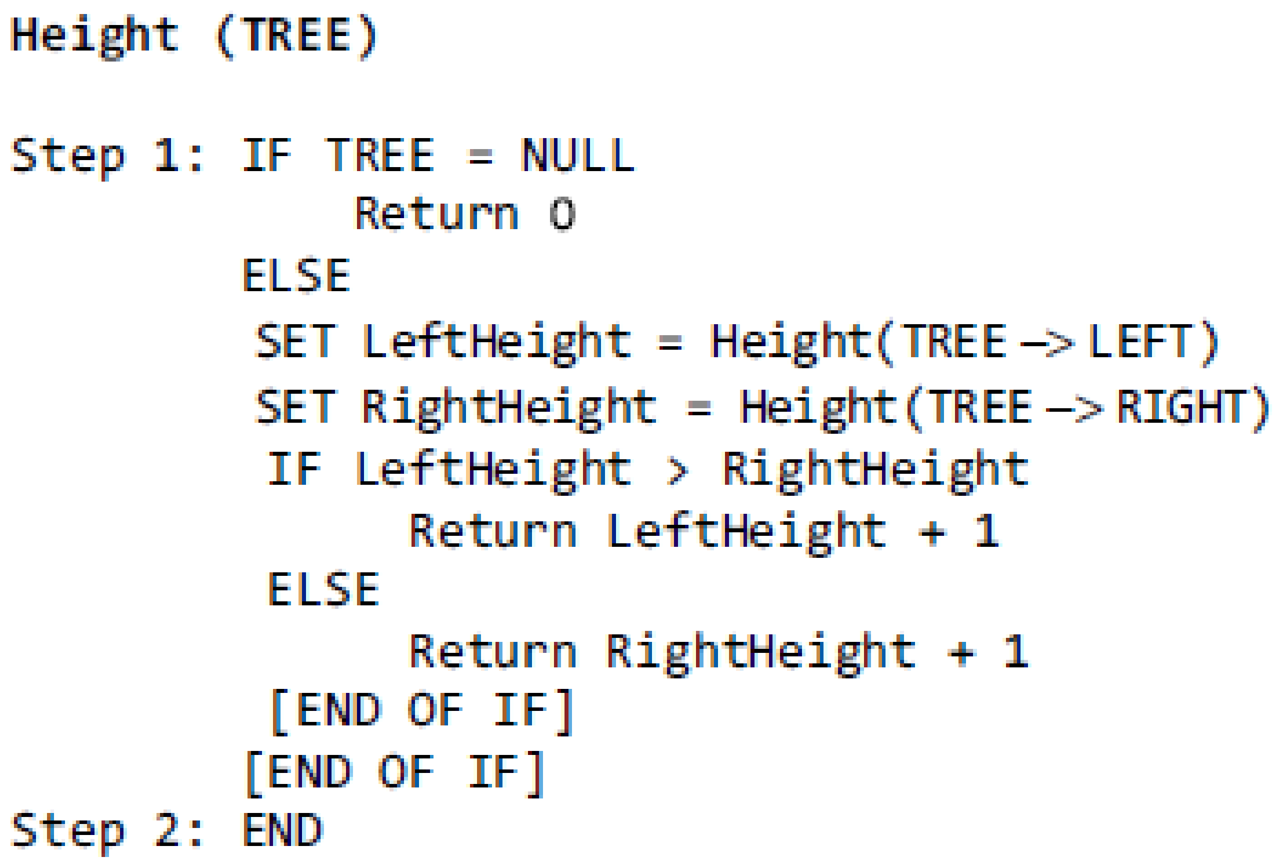
Since the height of the right sub-tree is greater than the height of the left sub-tree,

The height of the tree = height (right sub-tree) + 1

= 2 + 1

= 3.

Algorithm to determine the height of a binary search tree



```
Height (TREE)

Step 1: IF TREE = NULL
             Return 0
        ELSE
         SET LeftHeight = Height(TREE -> LEFT)
         SET RightHeight = Height(TREE -> RIGHT)
          IF LeftHeight > RightHeight
               Return LeftHeight + 1
          ELSE
               Return RightHeight + 1
          [END OF IF]
        [END OF IF]
Step 2: END
```

SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering
Somaiya TRUST

Determining the number of nodes in a binary search tree

- To find number of nodes in binary search tree first count the number of nodes in the left sub-tree and the right sub-tree and then add root node i.e +1

  Number of nodes =

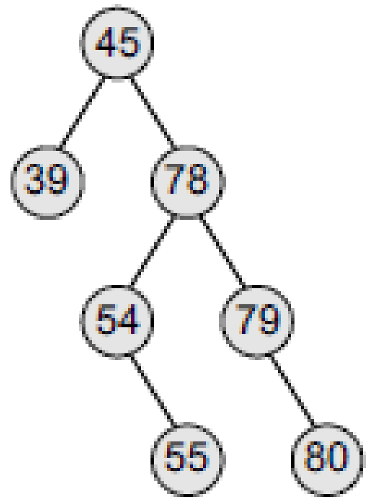
  totalNodes(left sub–tree) + totalNodes(right sub–tree) + 1

- Example: Total nodes of left sub–tree = 1

  Total nodes of right sub–tree = 5

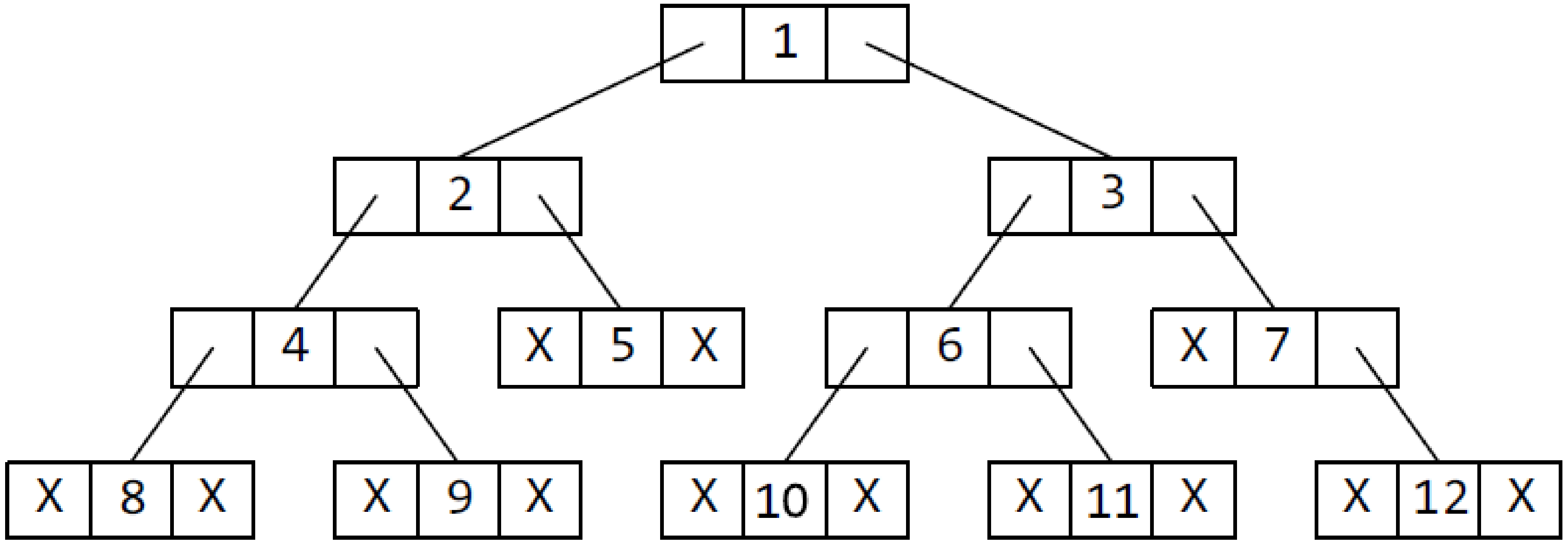
  Total nodes of tree = (1 + 5) + 1

  Total nodes of tree = 7



THREADED BINARY TREEs

- A threaded binary tree is the same as that of a binary tree but with a difference in storing the NULL pointers.
- In the linked representation, a number of nodes contain a NULL pointer, either in their left or right fields or in both.
- This space that is wasted in storing a NULL pointer can be efficiently used to store some other useful piece of information.



Linked representation of a binary tree

Threaded Binary Trees…

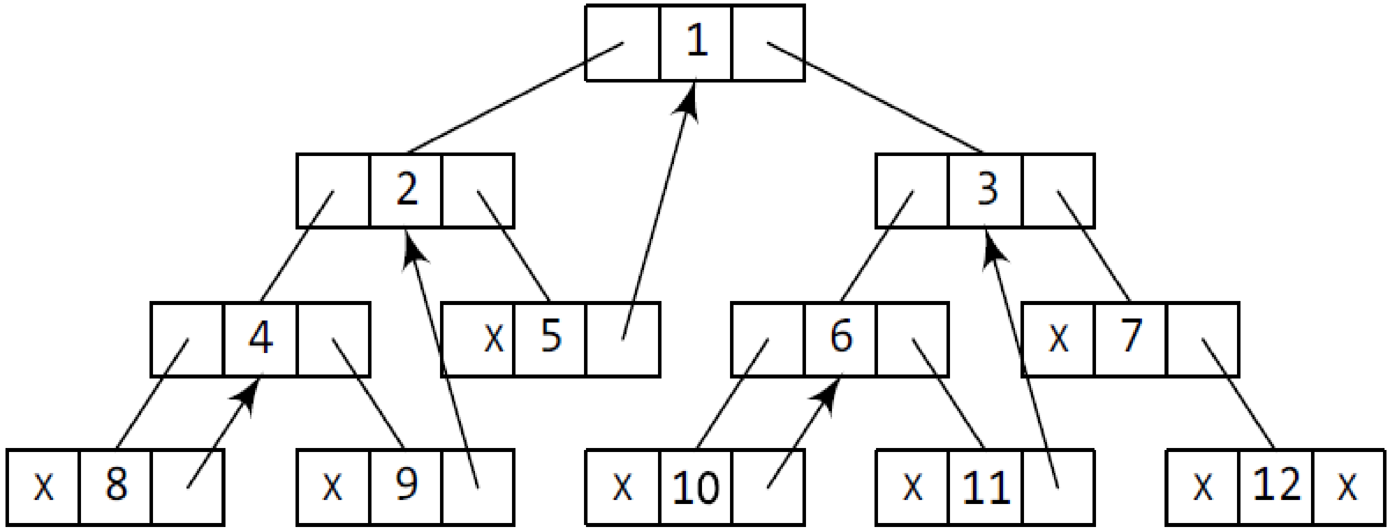
- The NULL entries can be replaced to store a pointer to the in-order predecessor or the in-order successor of the node.
- These special pointers are called *threads* and binary trees containing threads are called *threaded trees*.
- In the linked representation of a threaded binary tree, threads will be denoted using arrows.
- Two Ways:
  - One-way Threaded Binary Tree
  - Two-way Threaded Binary Tree

One-way Threaded Binary Tree

- In one-way threading, a thread will appear either in the right field or the left field of the node.
- A one-way threaded tree is also called a single-threaded tree.
- If the thread appears in the left field, then the left field will be made to point to the in-order predecessor of the node. Such a one-way threaded tree is called a left-threaded binary tree.
- On the contrary, if the thread appears in the right field, then it will point to the in-order successor of the node. Such a one-way threaded tree is called a right threaded binary tree.

One-way Threaded Binary Tree

- Node 5 contains a NULL pointer in its RIGHT field, so it will be replaced to point to node 1, which is its in-order successor.
- Similarly, the RIGHT field of node 8 will point to node 4, the RIGHT field of node 9 will point to node 2, the RIGHT field of node 10 will point to node 6, the RIGHT field of node 11 will point to node 3, and the RIGHT field of node 12 will contain NULL because it has no in-order successor.



(a) Linked representation of the binary tree with one-way threading

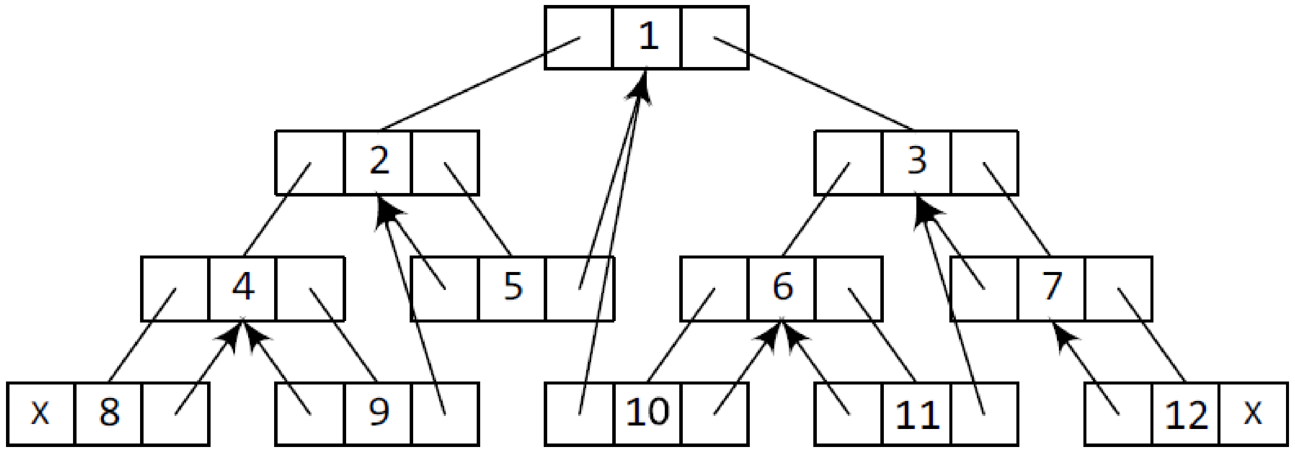
The in-order traversal of the tree is given as 8, 4, 9, 2, 5, 1, 10, 6, 11, 3, 7, 12

Two-way Threaded Binary Tree

- In a two-way threaded tree, also called a double-threaded tree, threads will appear in both the left and the right field of the node.
- While the left field will point to the in-order predecessor of the node, the right field will point to its successor. A two-way threaded binary tree is also called a fully threaded binary tree.

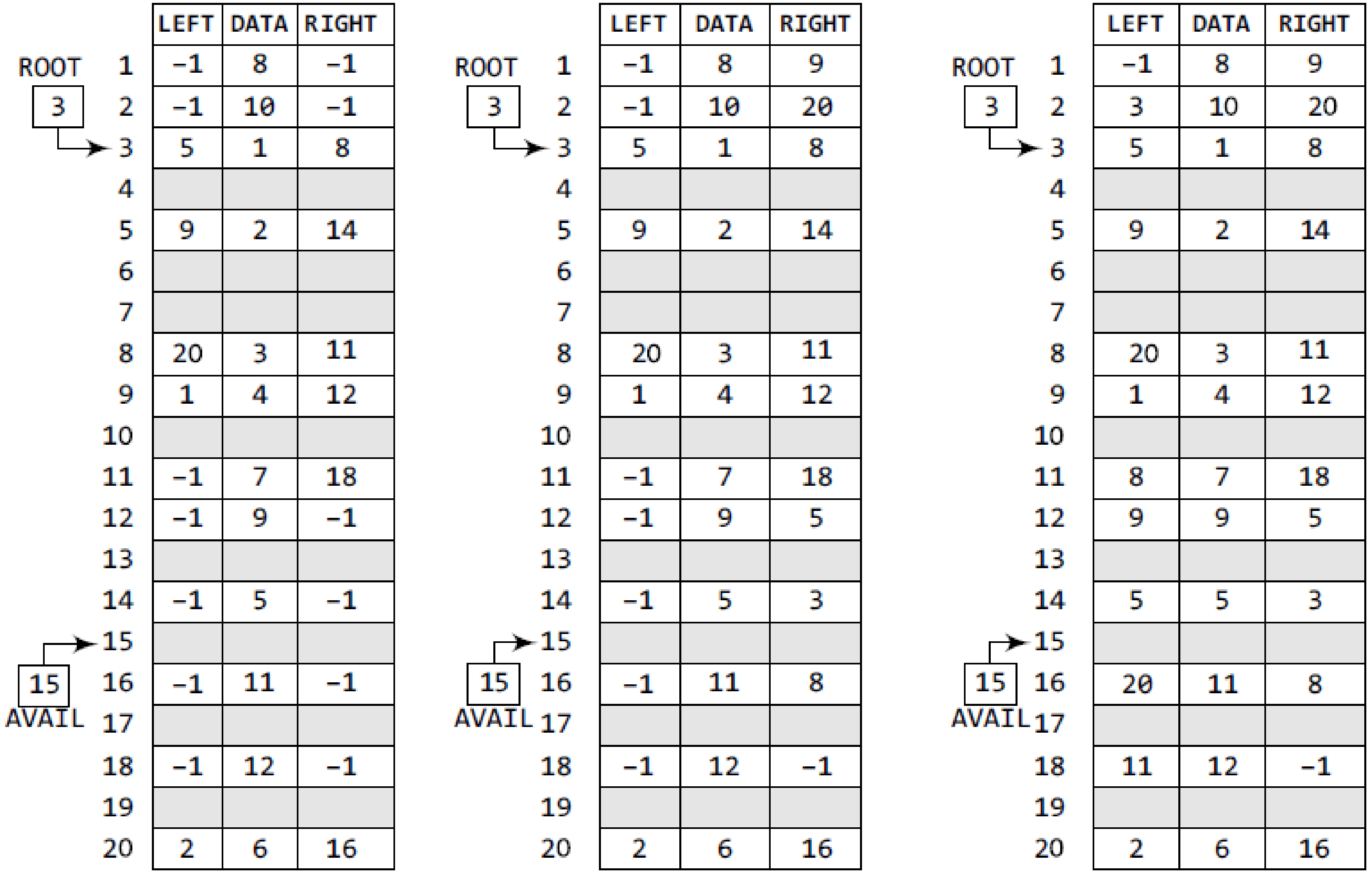
Two-way Threaded Binary Trees

- Node 5 contains a NULL pointer in its LEFT field, so it will be replaced to point to node 2, which is its in-order predecessor.
- Similarly, the LEFT field of node 8 will contain NULL because it has no in-order predecessor, the LEFT field of node 7 will point to node 3, the LEFT field of node 9 will point to node 4, the LEFT field of node 10 will point to node 1, the LEFT field of node 11 will contain 6, and the LEFT field of node 12 will point to node 7.



The in-order traversal of the tree is given as 8, 4, 9, 2, 5, 1, 10, 6, 11, 3, 7, 12

Threaded Binary Tree Representation in Memory


ROOT
3
AVAIL
15

(a)

	LEFT	DATA	RIGHT
1	–1	8	–1
2	–1	10	–1
3	5	1	8
4			
5	9	2	14
6			
7			
8	20	3	11
9	1	4	12
10			
11	–1	7	18
12	–1	9	–1
13			
14	–1	5	–1
15			
16	–1	11	–1
17			
18	–1	12	–1
19			
20	2	6	16

(b)

	LEFT	DATA	RIGHT
1	–1	8	9
2	–1	10	20
3	5	1	8
4			
5	9	2	14
6			
7			
8	20	3	11
9	1	4	12
10			
11	–1	7	18
12	–1	9	5
13			
14	–1	5	3
15			
16	–1	11	8
17			
18	–1	12	–1
19			
20	2	6	16

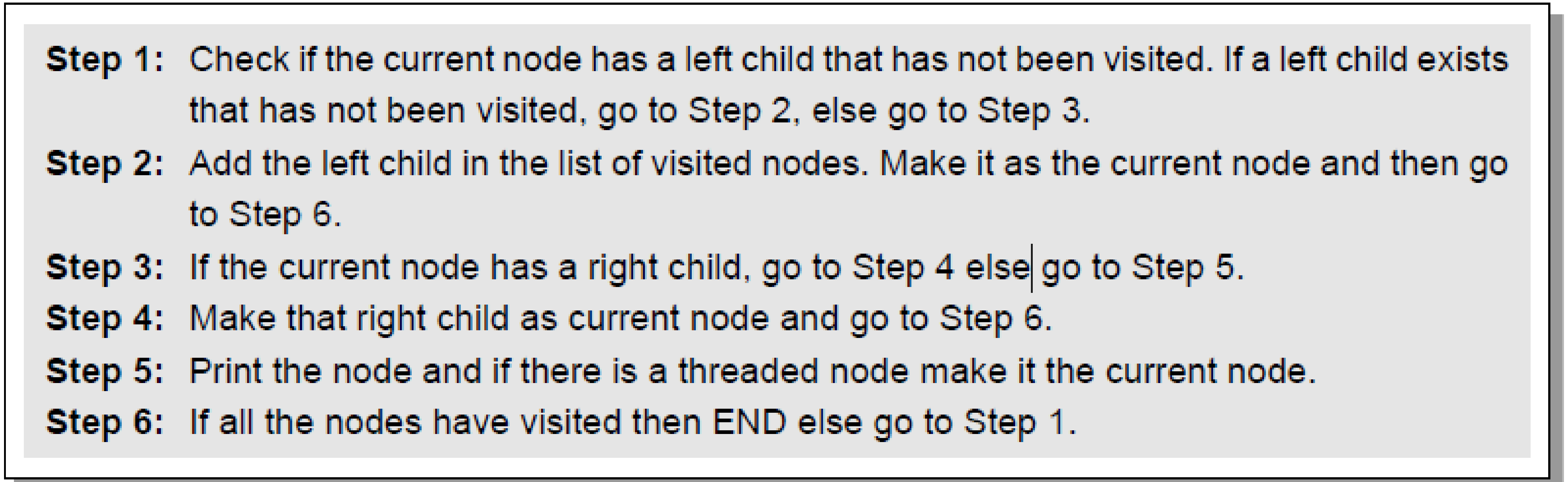
(c)

	LEFT	DATA	RIGHT
1	–1	8	9
2	3	10	20
3	5	1	8
4			
5	9	2	14
6			
7			
8	20	3	11
9	1	4	12
10			
11	8	7	18
12	9	9	5
13			
14	5	5	3
15			
16	20	11	8
17			
18	11	12	–1
19			
20	2	6	16

Memory representation of binary trees: (a) without threading, (b) with one-way, and (c) two-way threading

Traversing a Threaded Binary Tree

- For every node, visit the left sub-tree first, provided if one exists and has not been visited earlier.
- Then the node (root) itself is followed by visiting its right sub-tree (if one exists). In case there is no right sub-tree, check for the threaded link and make the threaded node the current node.



```
Step 1: Check if the current node has a left child that has not been visited. If a left child exists
        that has not been visited, go to Step 2, else go to Step 3.
Step 2: Add the left child in the list of visited nodes. Make it as the current node and then go
        to Step 6.
Step 3: If the current node has a right child, go to Step 4 else go to Step 5.
Step 4: Make that right child as current node and go to Step 6.
Step 5: Print the node and if there is a threaded node make it the current node.
Step 6: If all the nodes have visited then END else go to Step 1.
```

Algorithm for in-order traversal of a threaded binary tree

SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

AVL TREES

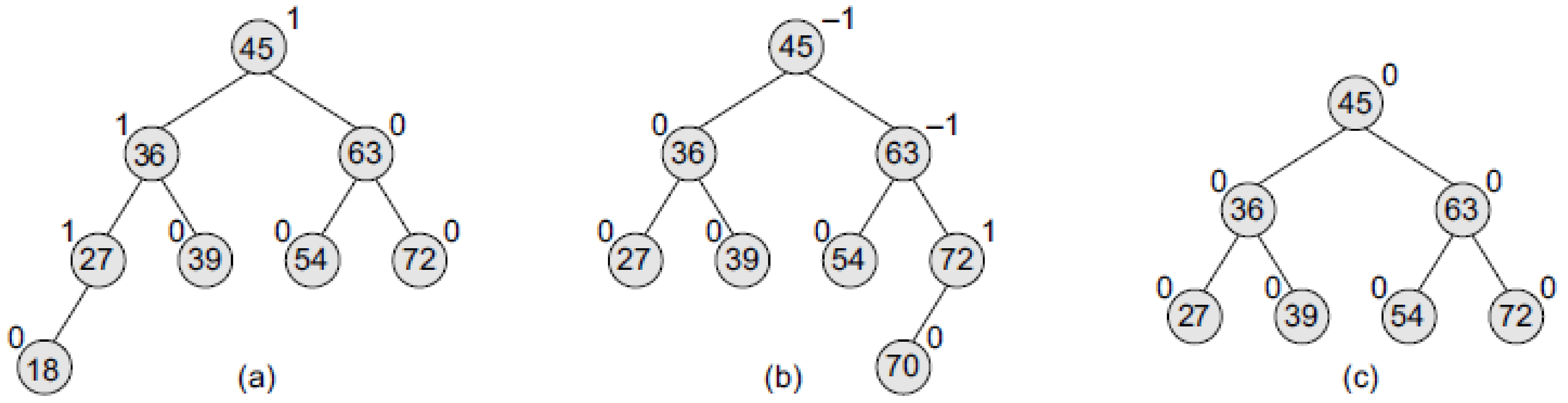
- AVL tree is a self-balancing binary search tree invented by G.M. Adelson-Velsky and E.M. Landis in 1962. The tree is named AVL in honor of its inventors.
- In an AVL tree, the heights of the two sub-trees of a node may differ by at most one. Due to this property, the AVL tree is also known as a height-balanced tree.
- The structure of an AVL tree is the same as that of a binary search tree but with a little difference, it stores an additional variable called the Balance Factor. Thus, every node has a balance factor associated with it.
- The balance factor of a node is calculated by subtracting the height of its right sub-tree from the height of its left sub-tree.
- A binary search tree in which every node has a balance factor of –1, 0, or 1 is said to be height balanced. A node with any other balance factor is considered to be unbalanced and requires rebalancing of the tree.

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

AVL Trees…

- If the balance factor of a node is 1, then it means that the left sub-tree of the tree is one level higher than that of the right sub-tree. Such a tree is therefore called as a *left-heavy tree*.
- If the balance factor of a node is 0, then it means that the height of the left sub-tree (longest path in the left sub-tree) is equal to the height of the right sub-tree.
- If the balance factor of a node is –1, then it means that the left sub-tree of the tree is one level lower than that of the right sub-tree. Such a tree is therefore called as a *right-heavy tree*.

AVL Trees…



(a) Left-heavy AVL tree, (b) right-heavy tree, (c) balanced tree

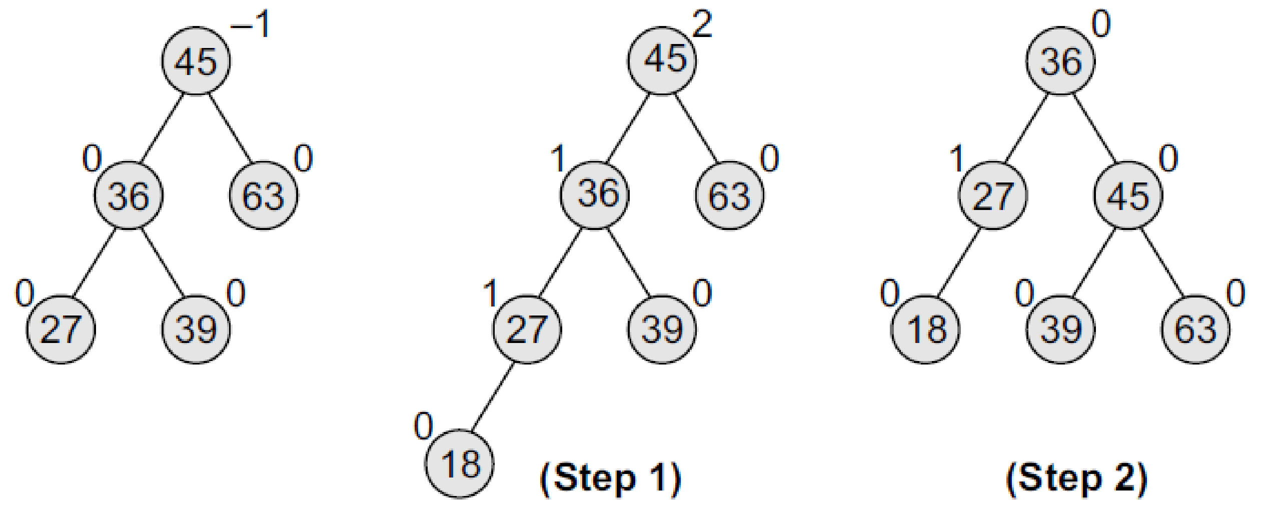
Operations on AVL Trees

Searching for a Node in an AVL Tree

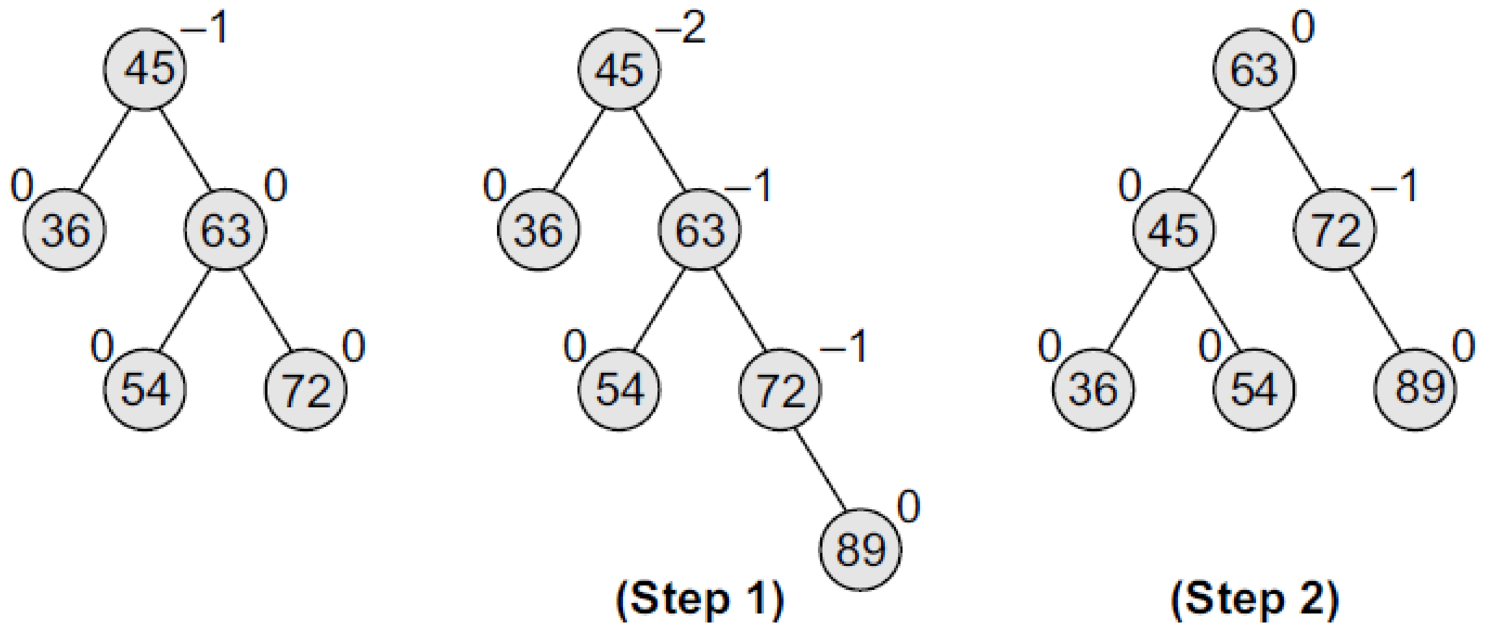
- Searching in an AVL tree is performed exactly the same way as it is performed in a binary search tree.

AVL Trees…

- Insertion:
- **Left Rotation (LL Rotation)**:
  - Occurs when the right subtree of a node becomes taller by 2 or more levels than the left subtree.
  - This rotation is used to balance the tree by promoting the right child to the position of its parent.
  - The left child of the right child becomes the new right child of the original parent.
  - Consider a given AVL tree and insert node 18 into a tree



- **Right Rotation (RR Rotation)**:
    - Occurs when the left subtree of a node becomes taller by 2 or more levels than the right subtree.
    - This rotation is used to balance the tree by promoting the left child to the position of its parent.
    - The right child of the left child becomes the new left child of the original parent.
- Consider the AVL tree given and insert 89 into it.



(Step 1)

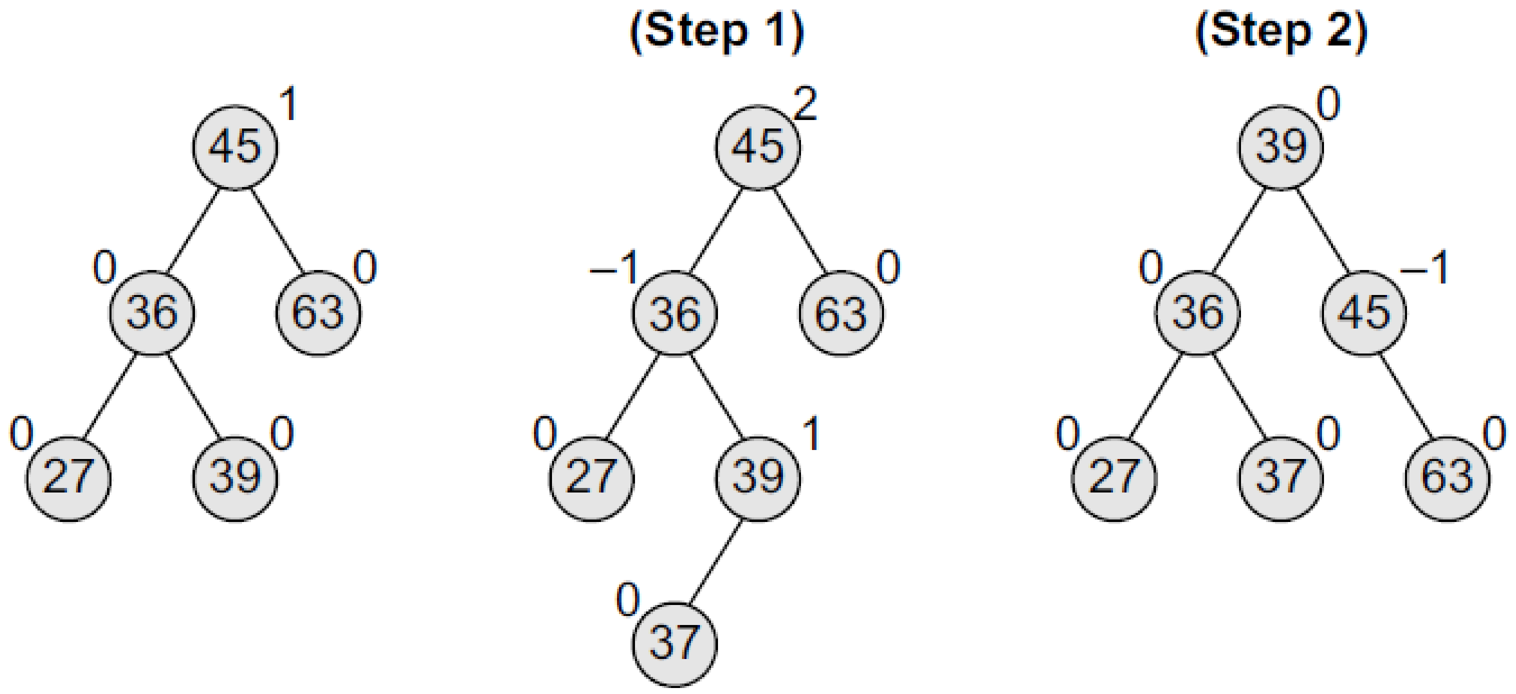
(Step 2)

AVL Trees…

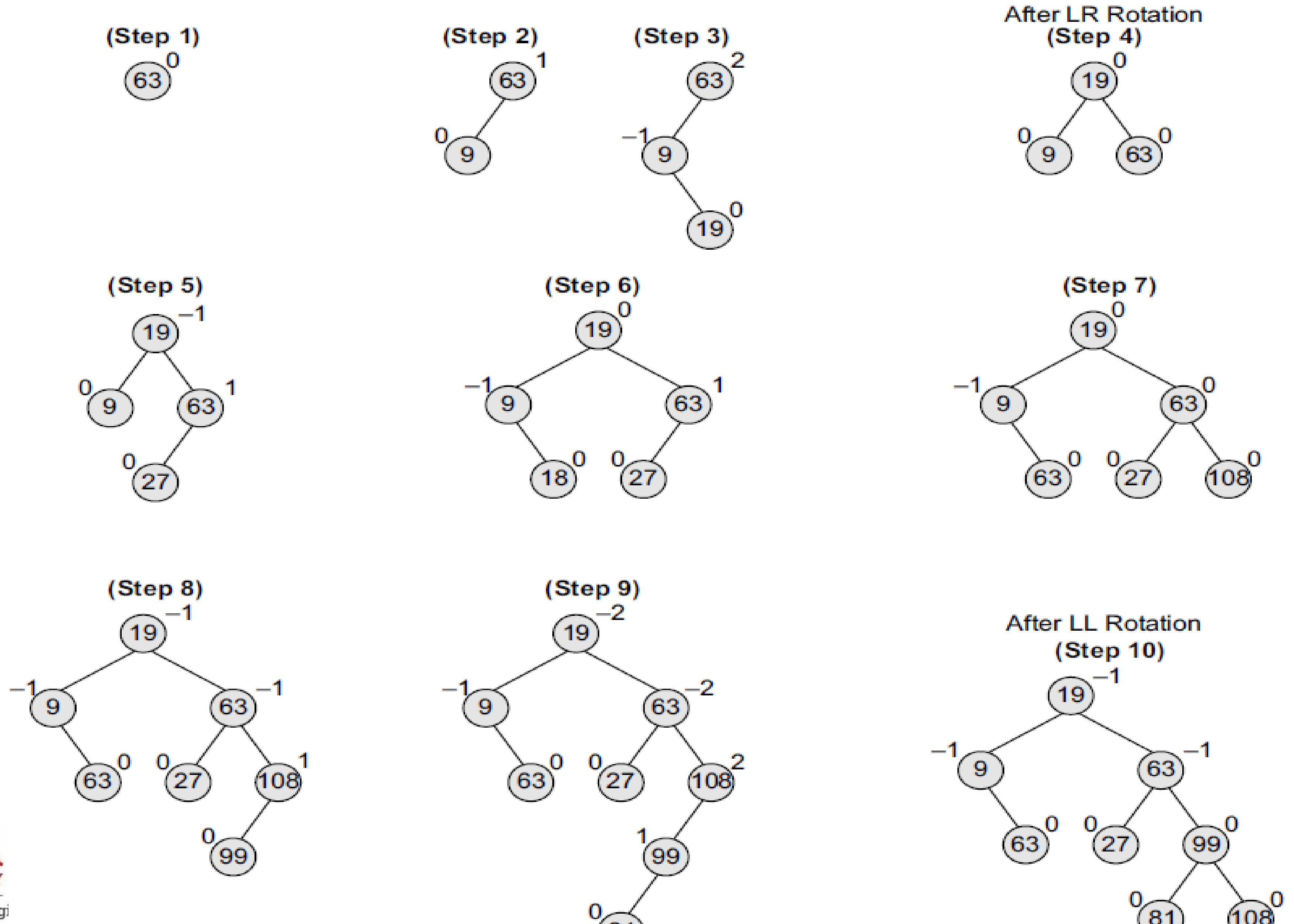
- **Left-Right Rotation (LR Rotation)**:
  - Occurs when a node's left child is unbalanced with a higher right subtree, and it requires both a left and a right rotation to balance.
  - The left child is rotated right (RR Rotation), and then the parent is rotated left (LL Rotation)
- **Right-Left Rotation (RL Rotation)**:
  - Occurs when a node's right child is unbalanced with a higher left subtree, and it requires both a right and a left rotation to balance.
  - The right child is rotated left (LL Rotation), and then the parent is rotated right (RR Rotation).

AVL Trees…

- Consider the AVL tree given and insert 37 into it.



Example: Construct an AVL tree by inserting the following elements in the given order. 63, 9, 19, 27, 18, 108, 99, 81.

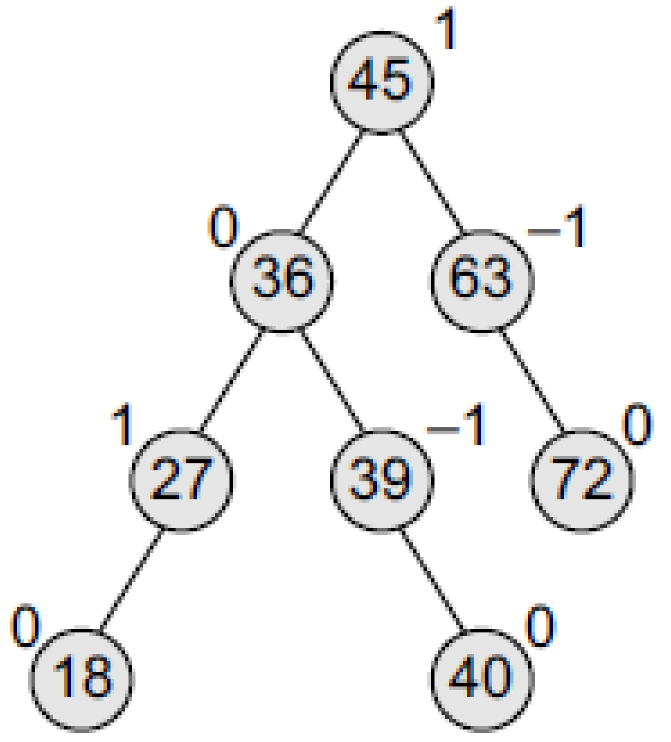


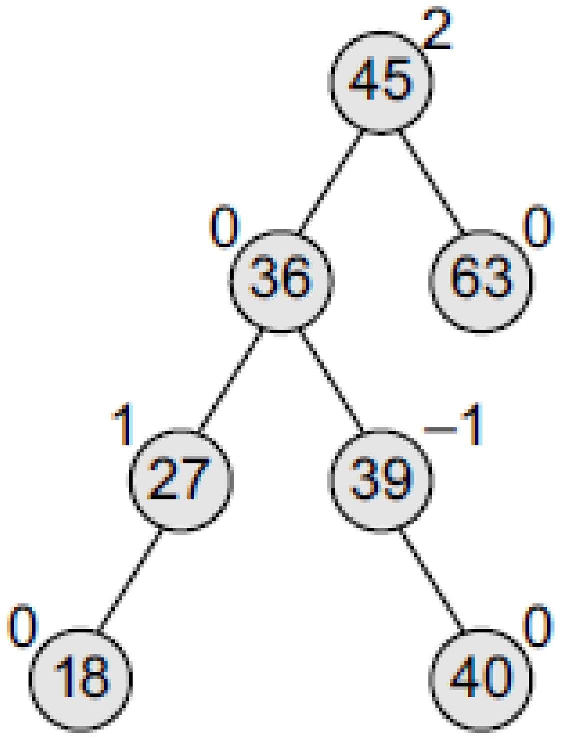
SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engi
Somaiya TRUST

Deleting a node from AVL tree

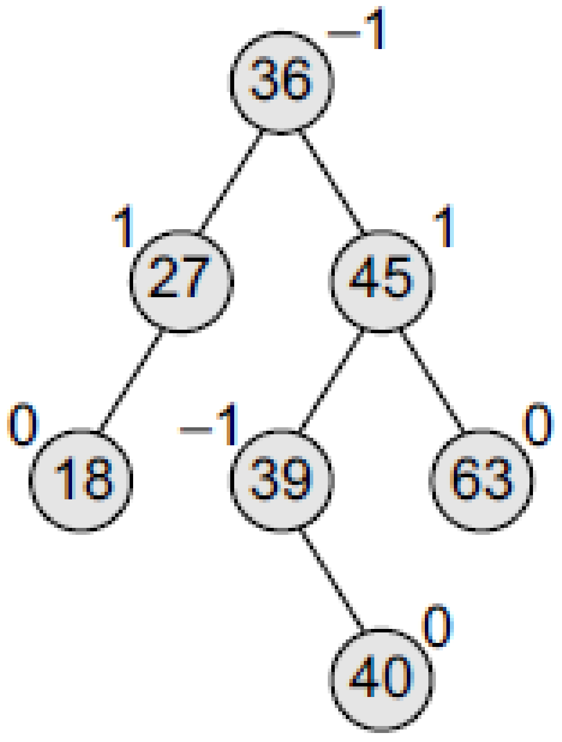
- When deleting a node from an AVL tree, you need to perform tree rotations to maintain the balance factor of the tree and ensure that it remains an AVL tree.
- The process typically involves two steps: deleting the node and then rebalancing the tree as needed. Here are the steps for deleting a node from an AVL tree:
    - **Node Deletion**:
    - a. If the node to be deleted has no children (a leaf node), simply remove it.
    - b. If the node to be deleted has one child, replace the node with its child.
    - c. If the node to be deleted has two children, find the in-order successor (the smallest node in its right subtree), copy its data to the node to be deleted, and then delete the in-order successor.

Consider the AVL tree given and delete 72 from it.(R0)



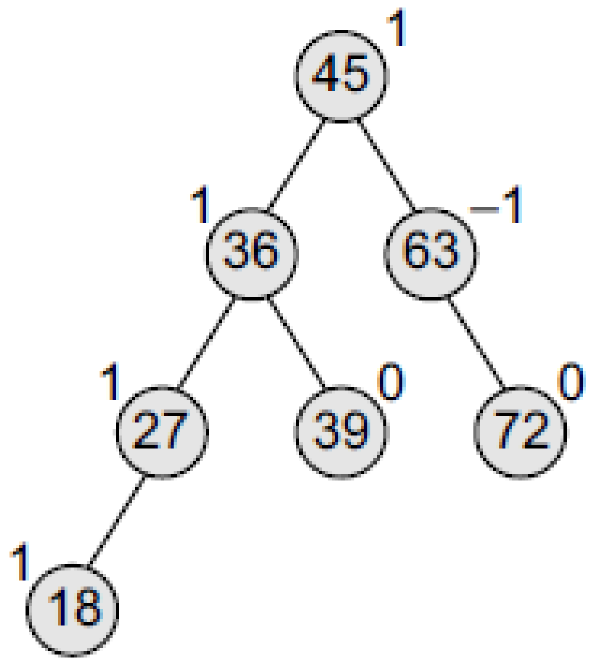


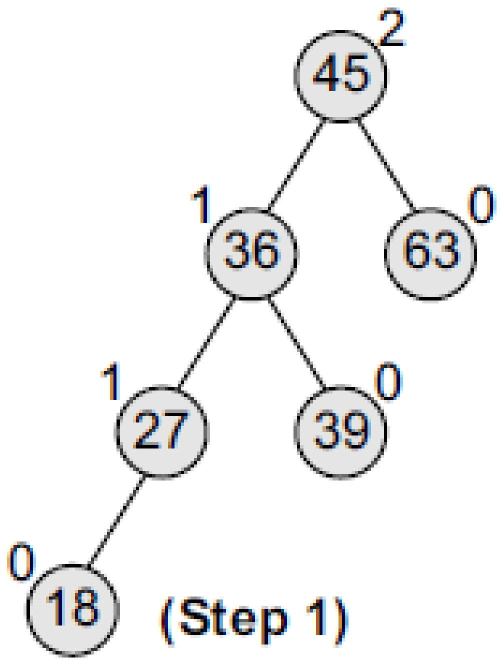
(Step 1)



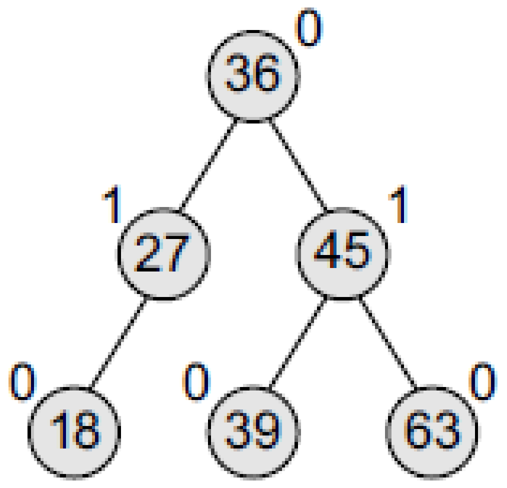
(Step 2)

Consider the AVL tree given and delete 72 from it.(R1)





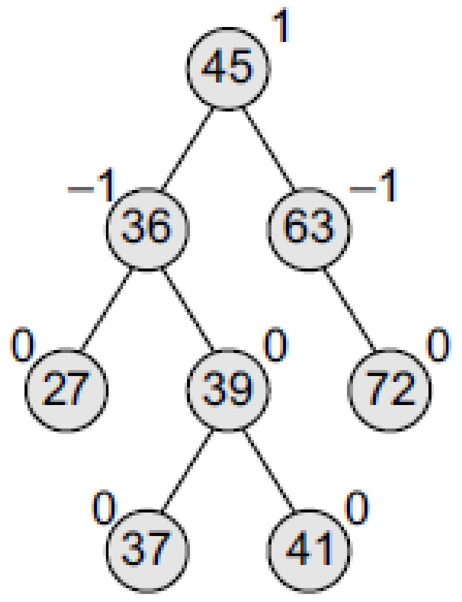
(Step 1)



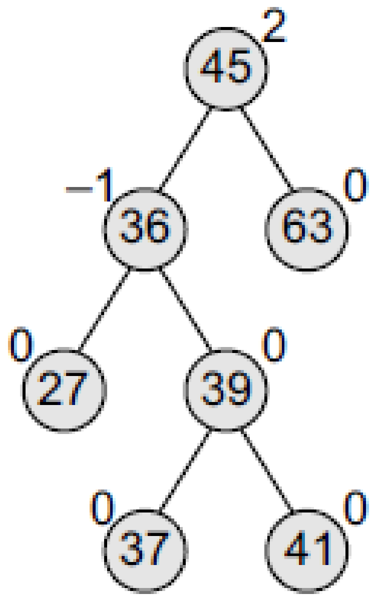
(Step 2)

SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering
Somaiya TRUST

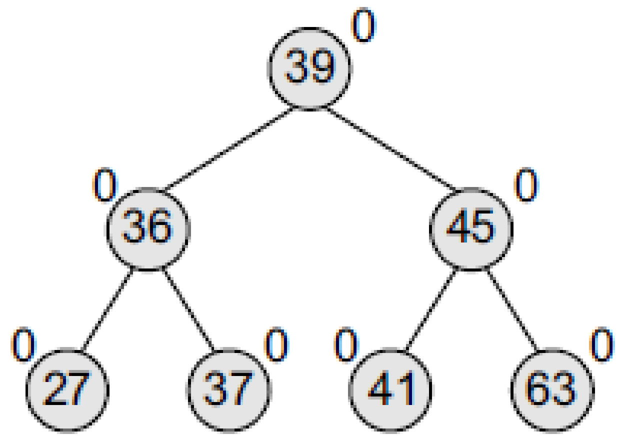
Consider the AVL tree given and delete 72 from it.(R-1)



(Step 1)



(Step 2)



Multi-way Search Trees

- An M-way search tree is a type of tree structure used for organizing and storing data. It's often employed in scenarios where efficient searching and retrieval of data are critical.
- In an M-way search tree, each internal node can have up to M children. The value of M is a parameter that can vary depending on the specific application and implementation.
- M is called as degree of tree

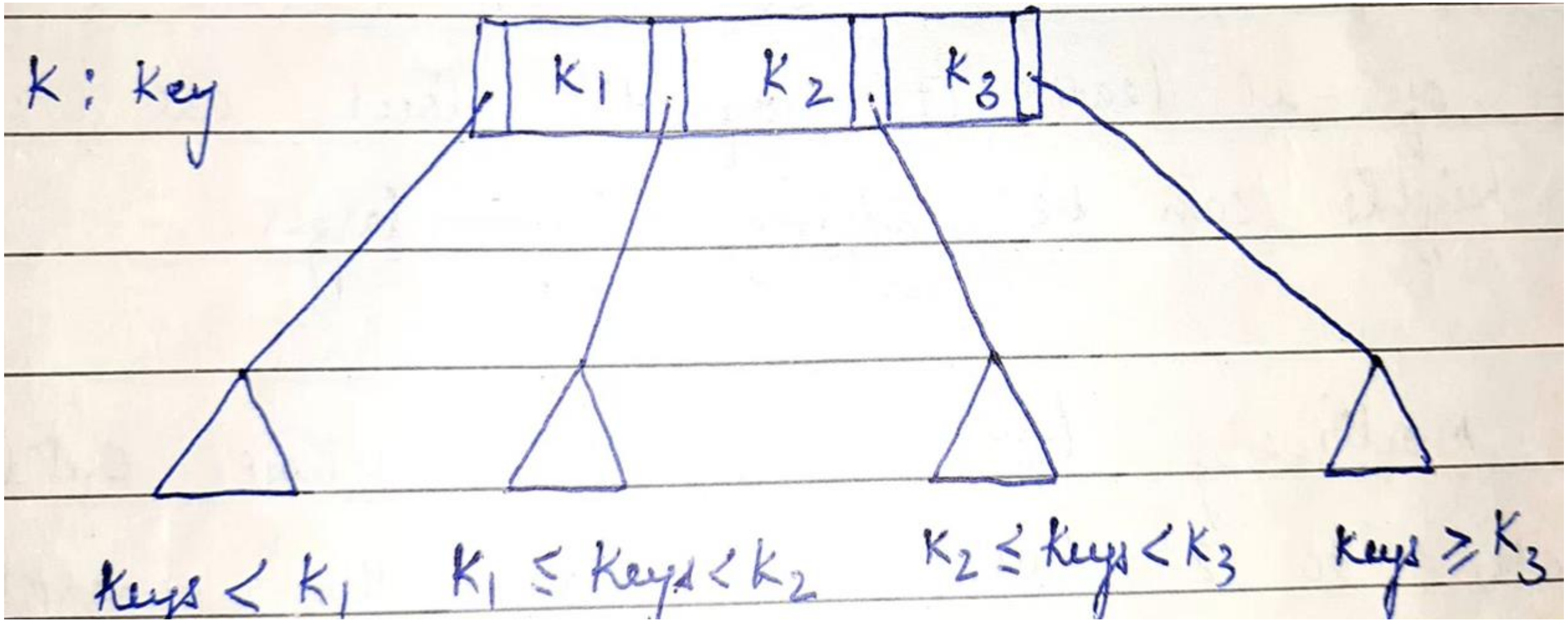
P_0	K_0	P_1	K_1	P_2	K_2	......	P_{n-1}	K_{n-1}	P_n

In the structure shown, $P_0, P_1, P_2, \ldots, P_n$ are pointers to the node's sub-trees and $K_0, K_1, K_2, \ldots, K_{n-1}$ are the key values of the node. All the key values are stored in ascending order. That is, $K_i < K_{i+1}$ for $0 \le i \le n-2$.

SOMAIYA
VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering
Somaiya
TRUST

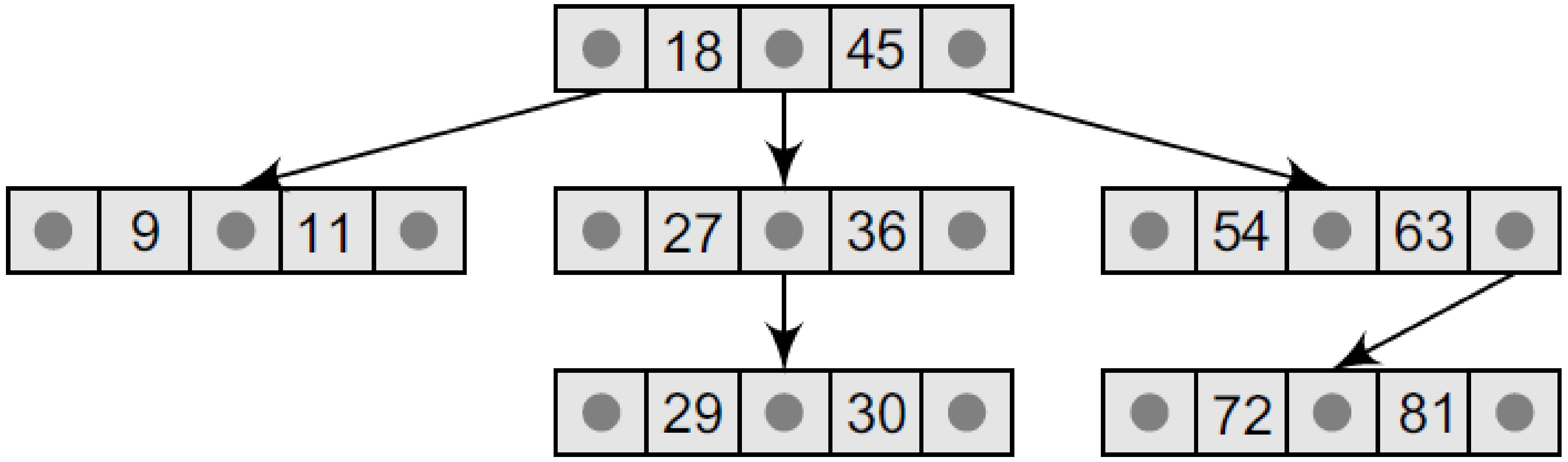
M-way Tree…

M-way tree: M=4 , so number of node=M-1=3



M-way Tree…

- M-way Tree of Order 3: A node can store maximum of 2 key values and can contains pointers to 3 subtrees
- It can be called as (2,3) tree i.e tree with 2 keys in a node and 3 pointers

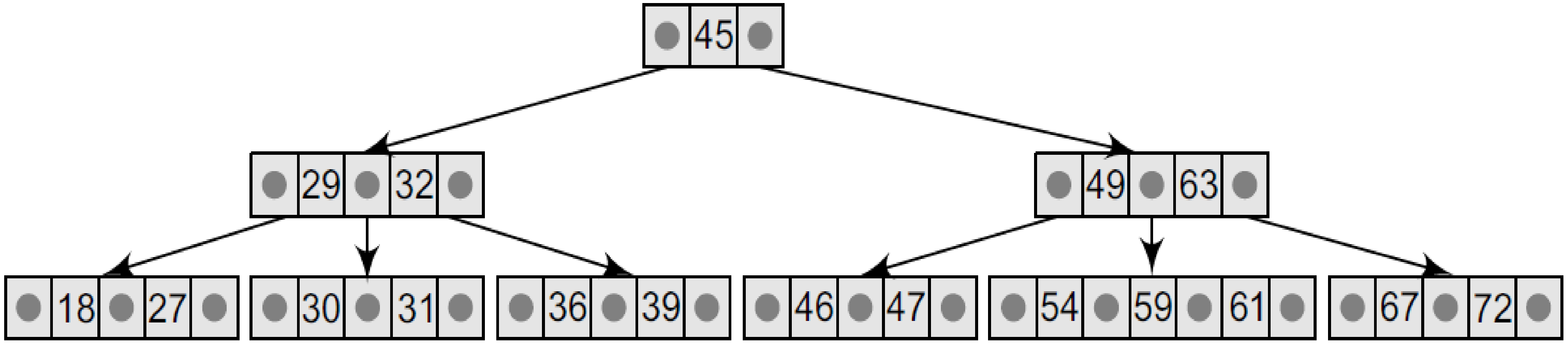


B-Trees

- B-trees are a type of self-balancing search tree data structure. They are commonly used in databases and file systems due to their efficient insertion, deletion, and search operations. B-trees maintain balance by adjusting the structure as nodes are added or removed.
- 1.Every node in the B tree has at most (maximum) `m` children.
- 2. Every node in the B tree except the root node and leaf nodes has at least (minimum) `m/2` children. This condition helps to keep the tree bushy so that the path from the root node to the leaf is very short, even in a tree that stores a lot of data.
- 3. The root node has at least two children if it is not a terminal (leaf) node.
- 4. All leaf nodes are at the same level.

B-Tree…

- An internal node in the B tree can have n number of children, where 0 <= n <= m.
- It is not necessary that every node has the same number of children, but the only restriction is that the node should have at least m/2 children.
- As B tree of order 4 is given in Fig:



Operations on B-Tree

- Searching:

  Searching for an element in a B tree is similar to that in binary search trees.

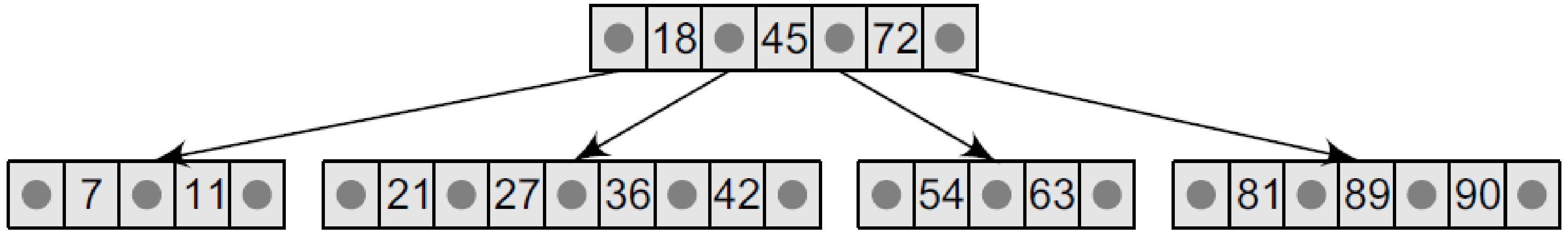
Operations on B-Tree…

- Insertion:
- In a B tree, all insertions are done at the leaf node level. A new value is inserted in the B tree using the algorithm given below.

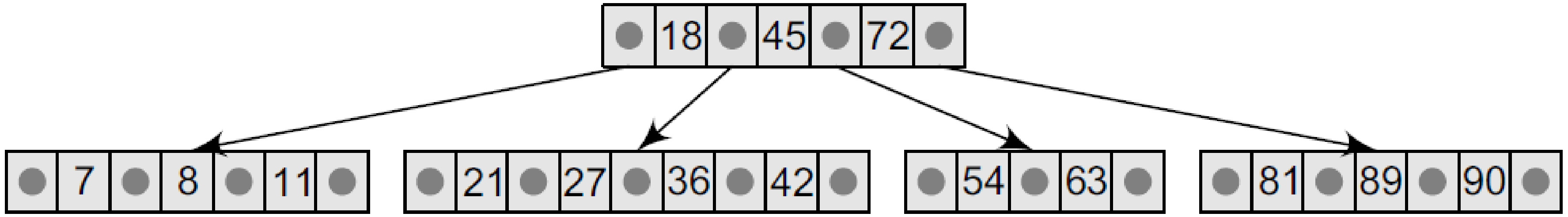
  1. Search the B tree to find the leaf node where the new key value should be inserted.
  2. If the leaf node is not full, that is, it contains less than m–1 key values, then insert the new element in the node keeping the node's elements ordered.
  3. If the leaf node is full, that is, the leaf node already contains m–1 key values, then

  - (a) insert the new value in order into the existing set of keys,
  - (b) split the node at its median into two nodes (note that the split nodes are half full), and
  - (c) push the median element up to its parent's node. If the parent's node is already full, then split the parent node by following the same steps.

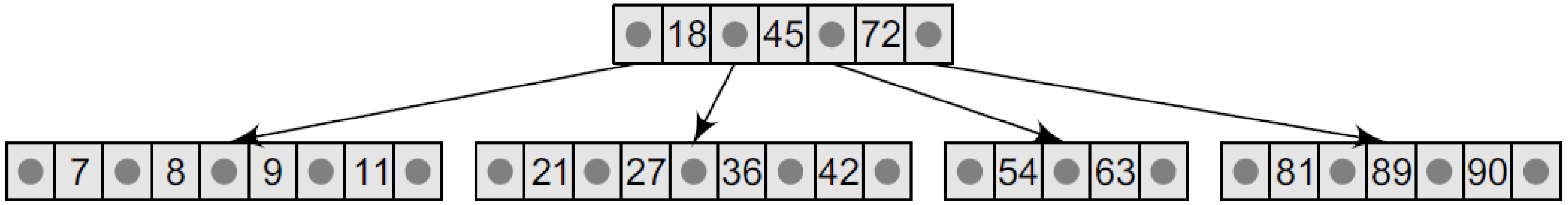
- Look at the B tree of order 5 given below and insert 8, 9, 39, and 4 into it.



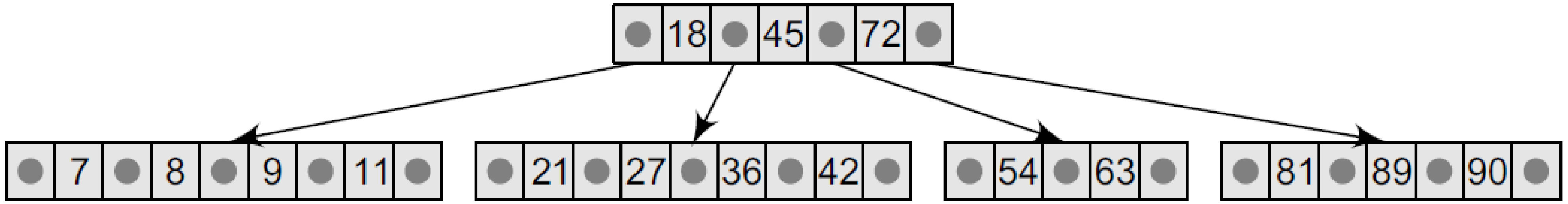
Step 1: Insert 8



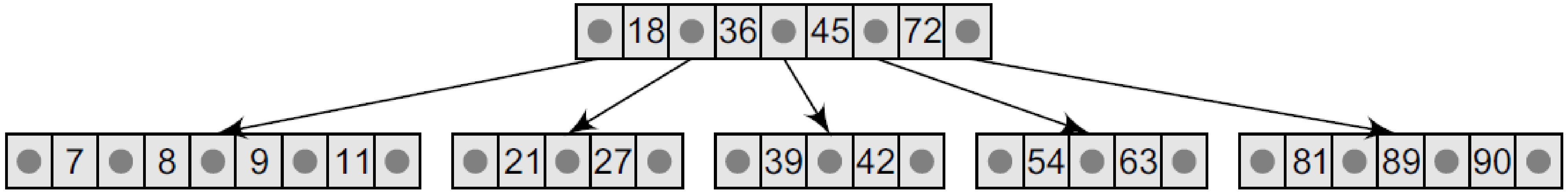
Step 2: Insert 9



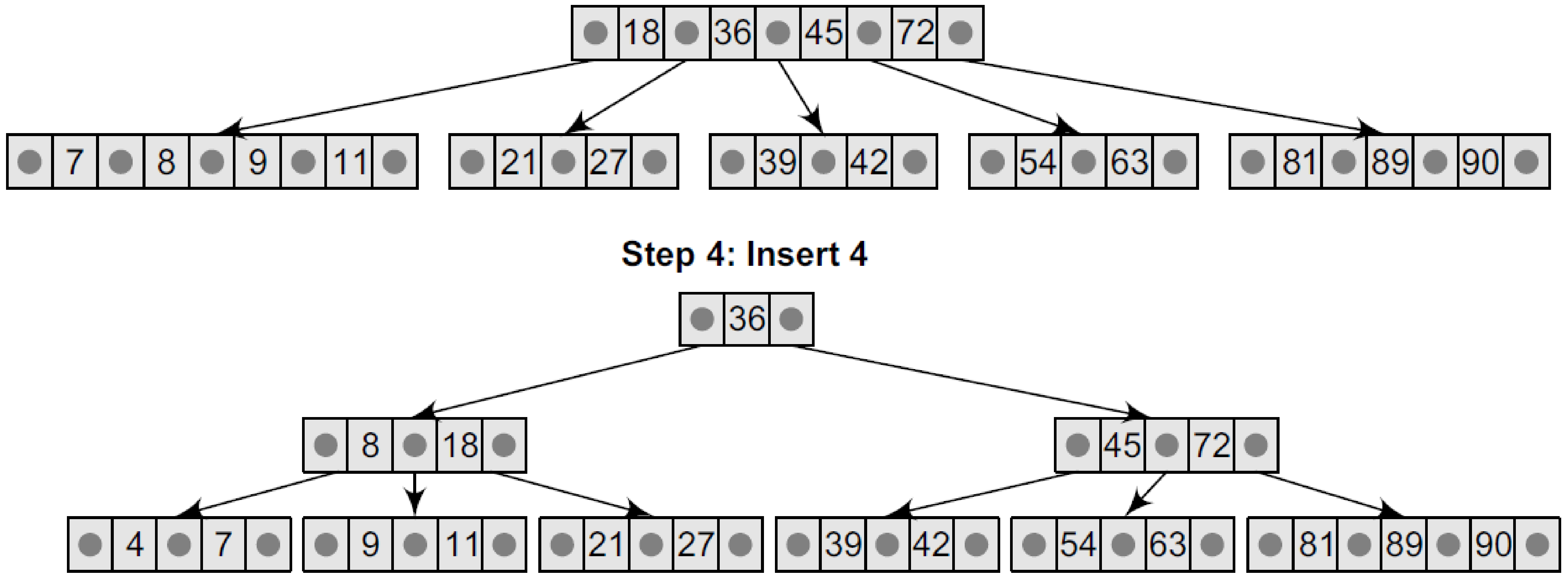
- Till now, we have easily inserted 8 and 9 in the tree because the leaf nodes were not full. But now, the node in which 39 should be inserted is already full as it contains four values. Here we split the nodes to form two separate nodes. But before splitting, arrange the key values in order (including the new value). The ordered set of values is given as 21, 27, 36, 39, and 42. The median value is 36, so push 36 into its parent's node and split the leaf nodes.



Step 3: Insert 39



- Now the node in which 4 should be inserted is already full as it contains four key values. Here we split the nodes to form two separate nodes. But before splitting, we arrange the key values in order (including the new value). The ordered set of values is given as 4, 7, 8, 9, and 11. The median value is 8, so we push 8 into its parent's node and split the leaf nodes. But again, we see that the parent's node is already full, so we split the parent node using the same procedure.



Thank You