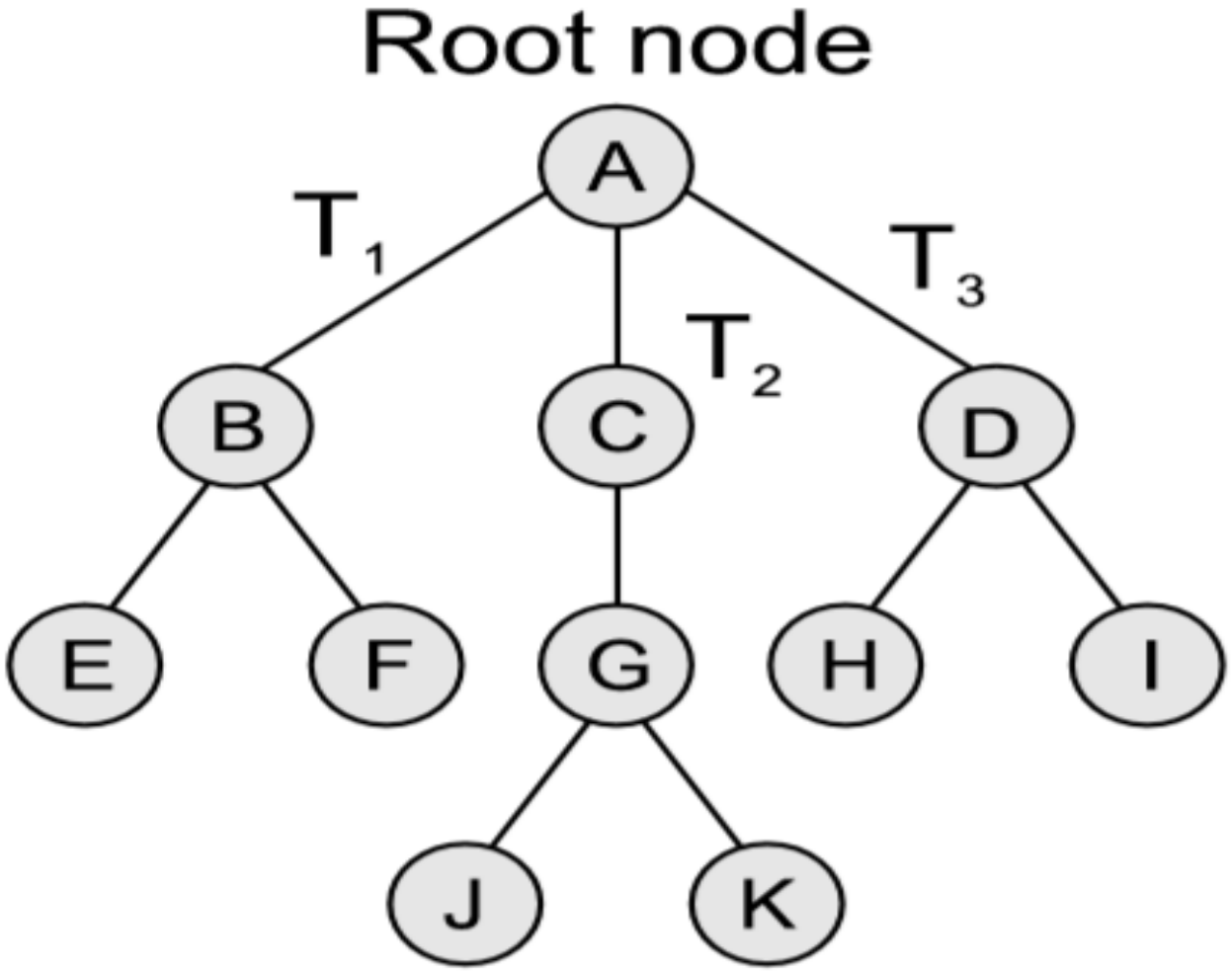
Indian Institute of Information Technology Kottayam

Binary Tree and Traversal

BY

Arun Cyril Jose

Trees



Trees

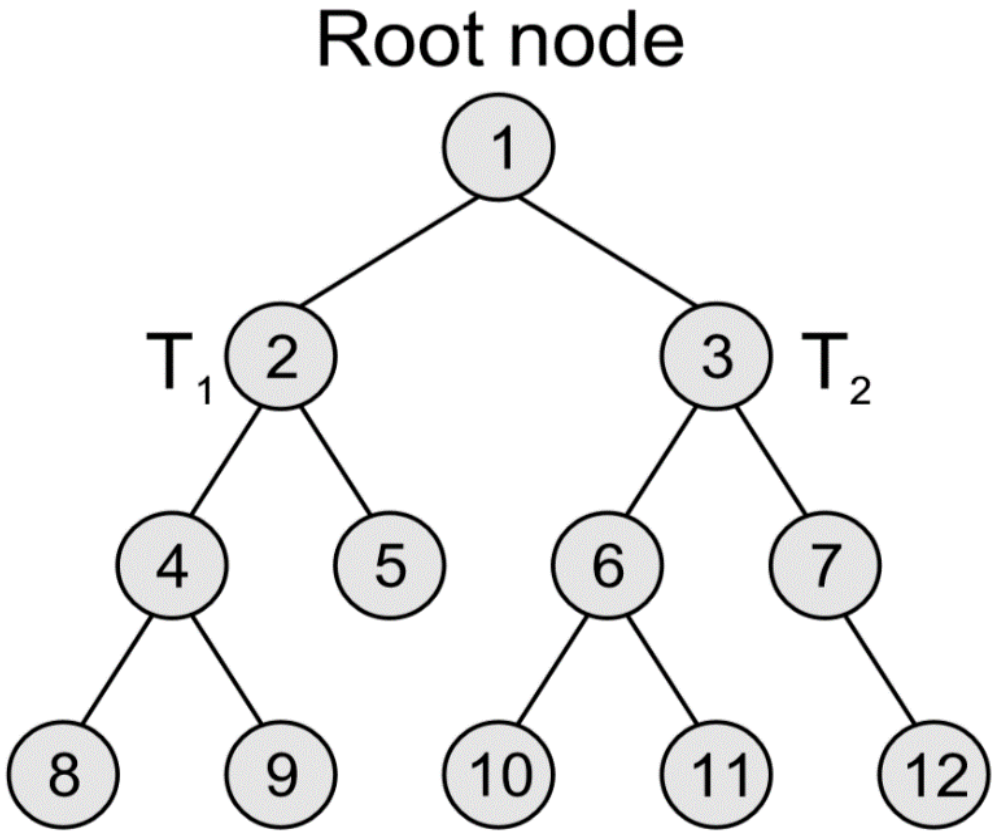
- **Connected acyclic undirected graph.**
- A Graph in which any two vertices are connected by exactly one path.

Trees

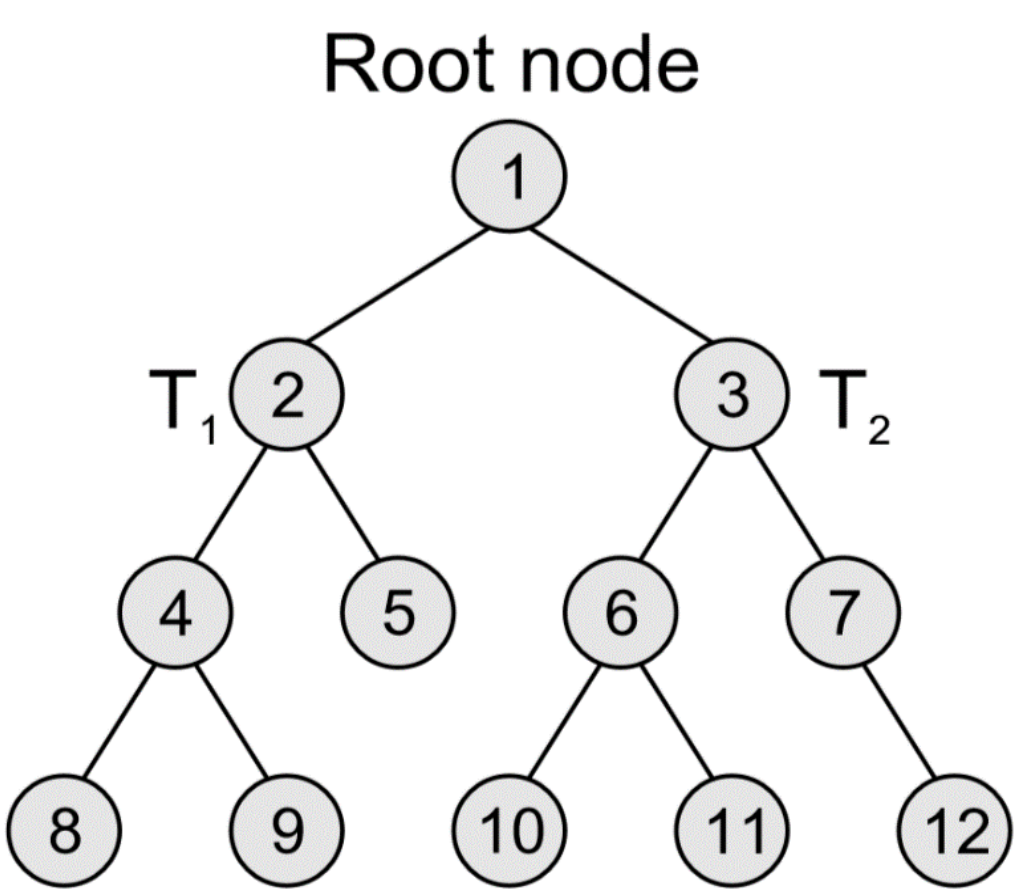
- **Root:** Root node R is the topmost node in the tree.
- **Leaf node:** A node that has no children is called the leaf node or the terminal node.
- **Sub-trees:** If the root node R is not NULL, then the trees T1, T2, and T3 are called the sub-trees of R.
- **Path:** A sequence of consecutive edges is called a path.
- **Ancestor node:** An ancestor of a node is any predecessor node on the path from root to that node.

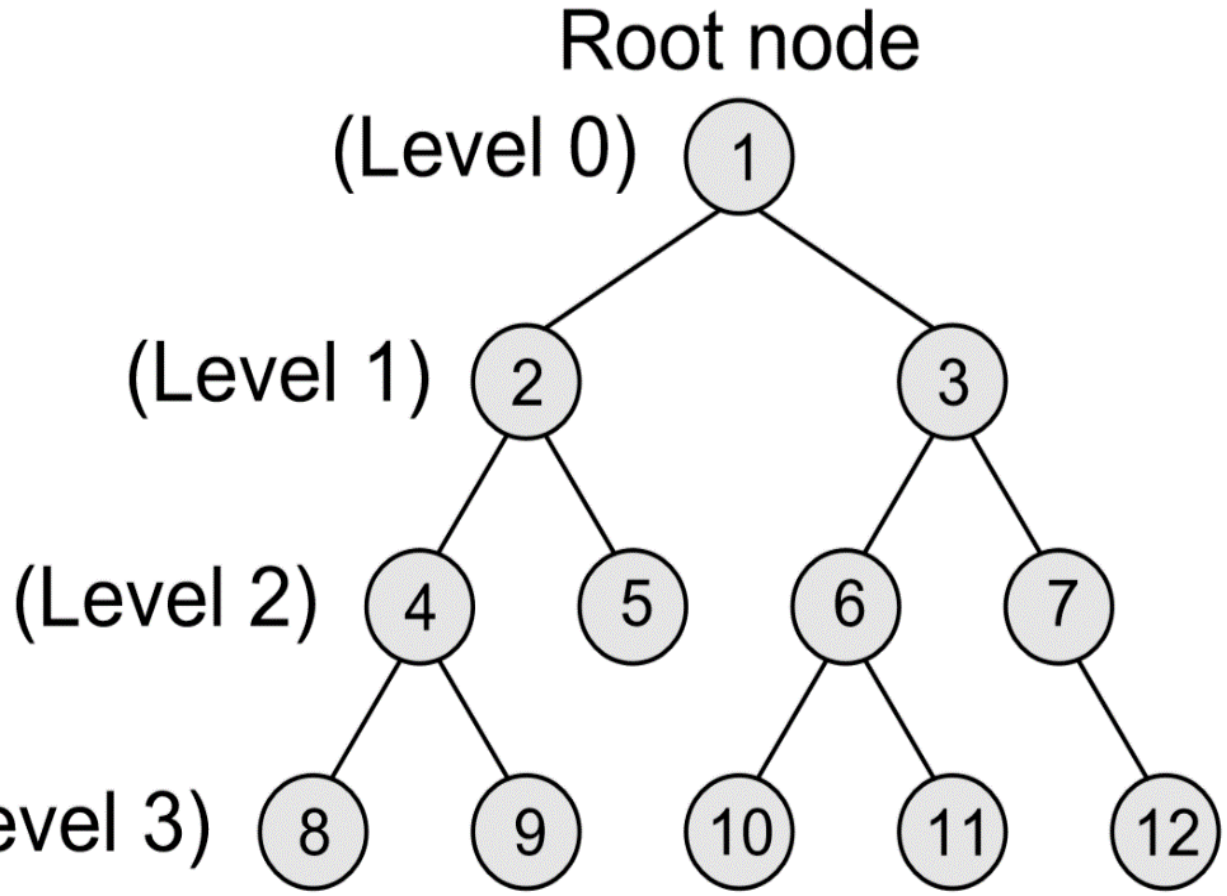
Binary Tree

- DS defined as a collection of elements called nodes.
- Topmost element is called the **root node**, and each node has 0, 1, or at the most 2 children.



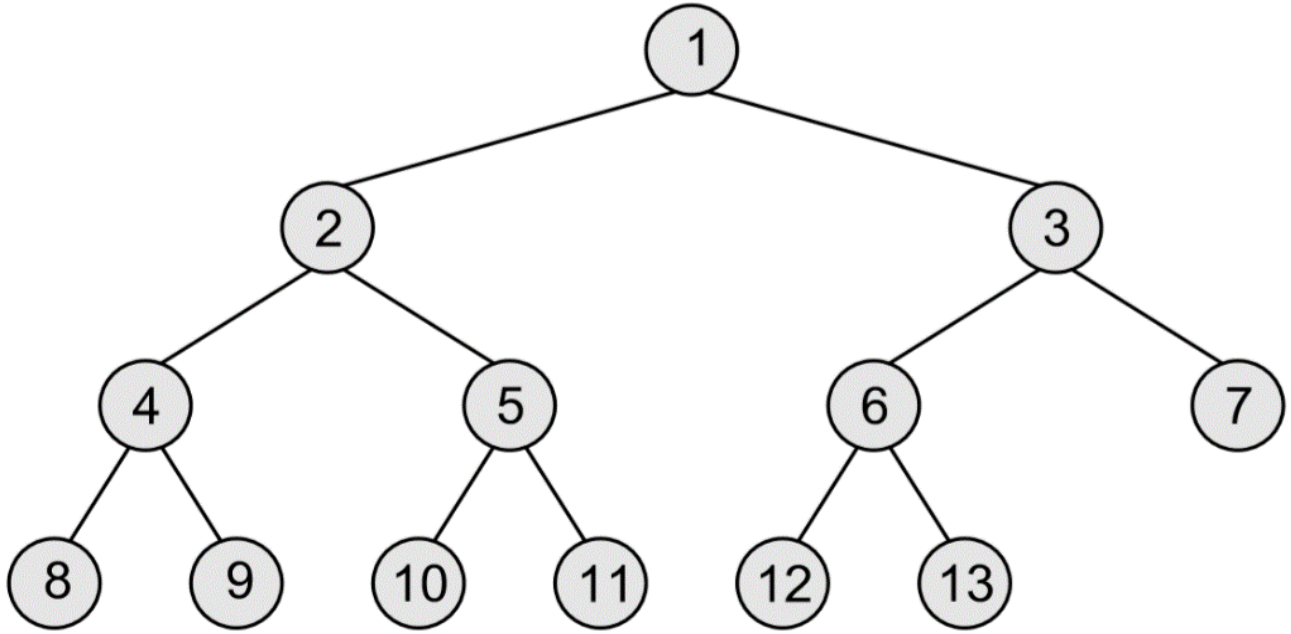
Binary Trees





Complete Binary Tree

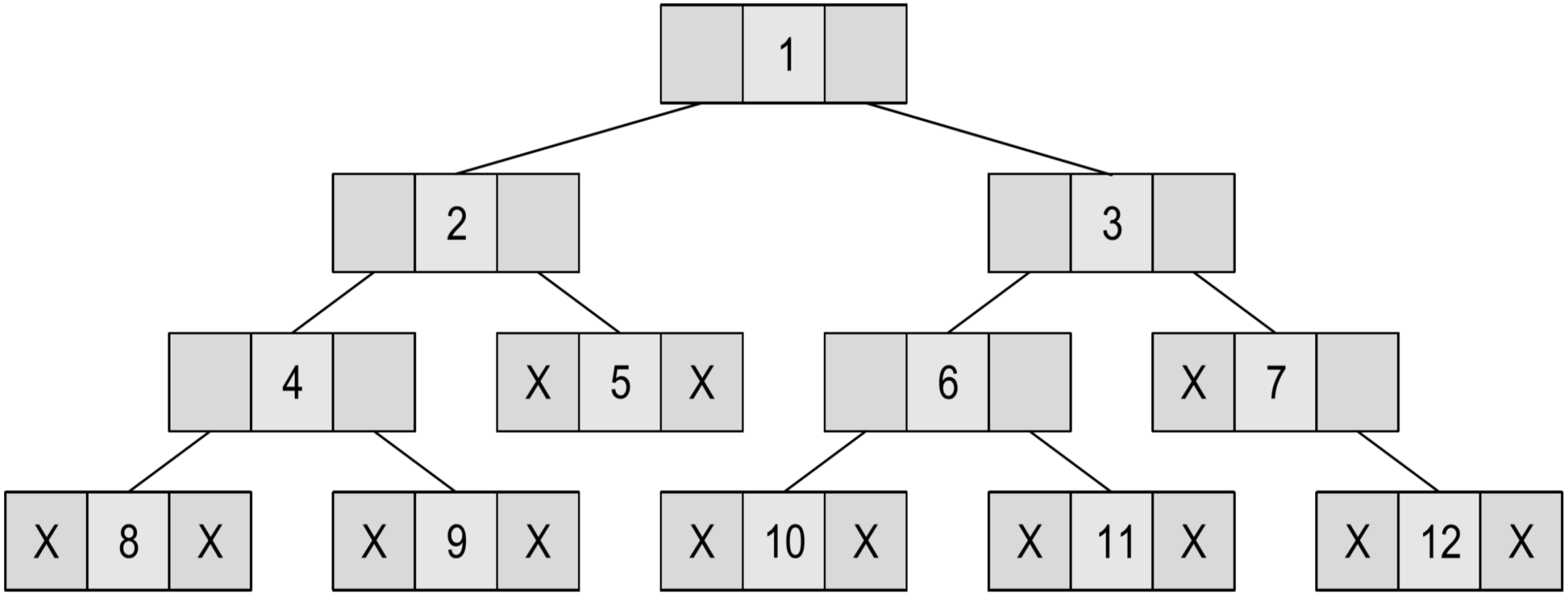
- Binary Tree that satisfies these two properties:
- Every level, except possibly the last, is completely filled.
- All nodes appear as far left as possible.
- In a complete binary tree T_n, there are exactly n nodes and level r of T can have at most 2^r nodes.



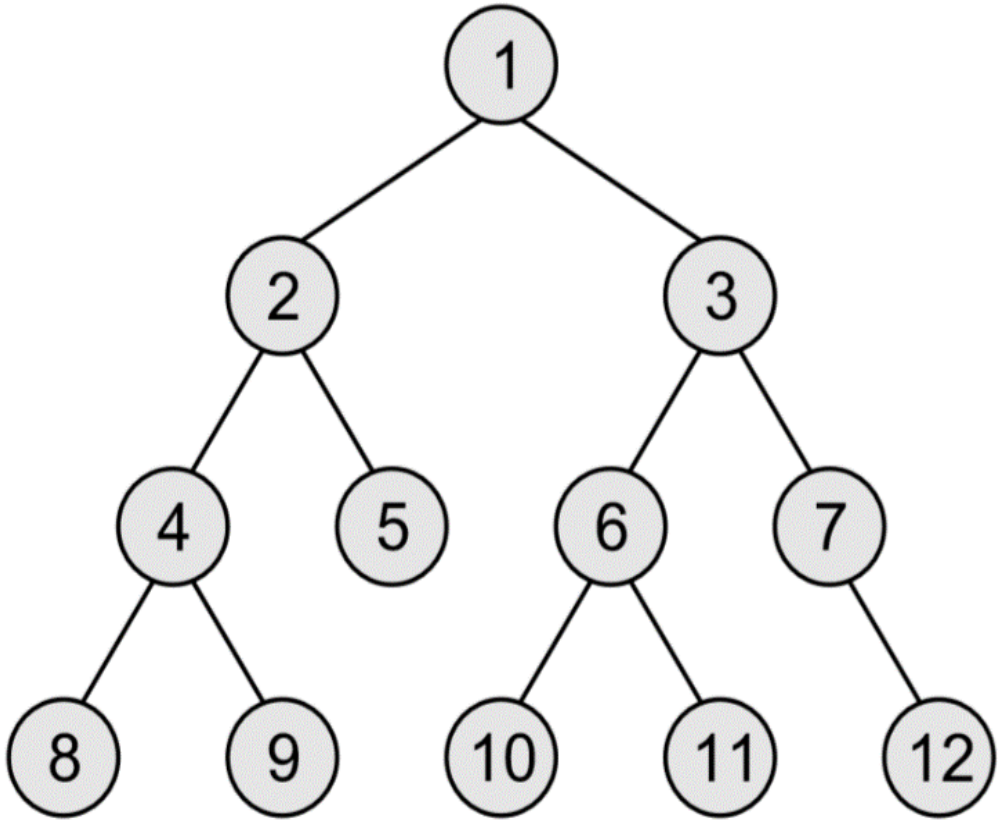
Binary Tree Representation

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

Binary Tree Representation



Binary Tree Representation



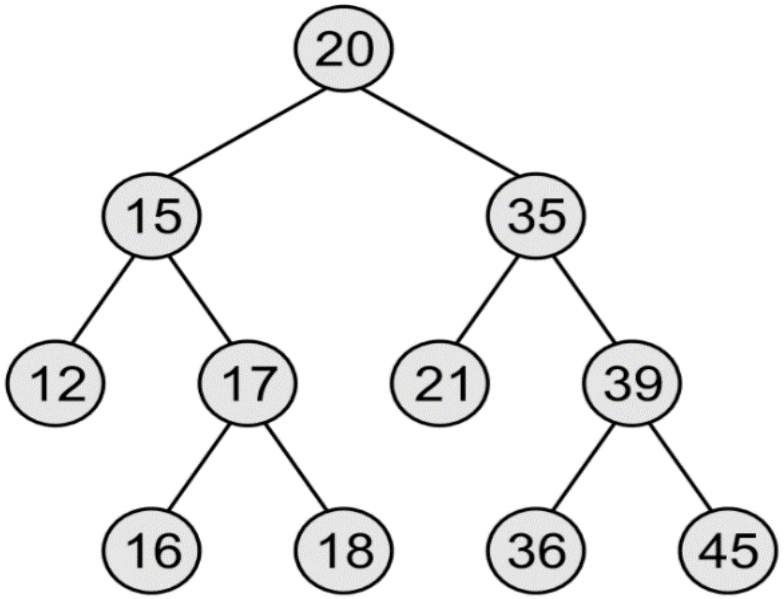
ROOT 3

AVAIL 15

	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

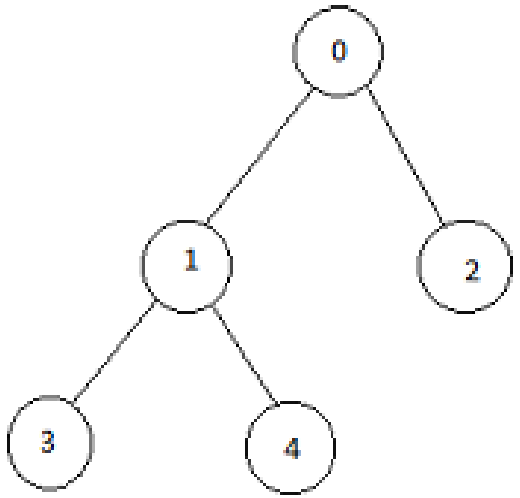
Binary Tree Representation

- 1D array.
- Root is stored in the first location
  - Here it is TREE[1]
- The children of a node stored in location K will be stored in locations (2 × K) and (2 × K+1)
- Maximum size of the array TREE is given as $(2^h–1)$, where h is the height of the tree.

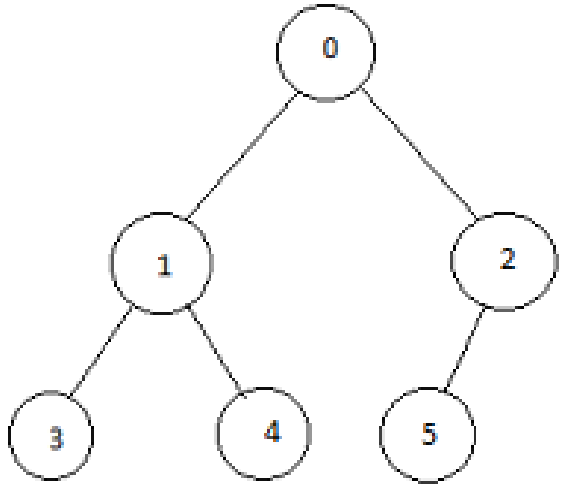


1	20
2	15
3	35
4	12
5	17
6	21
7	39
8	
9	
10	16
11	18
12	
13	
14	36
15	45

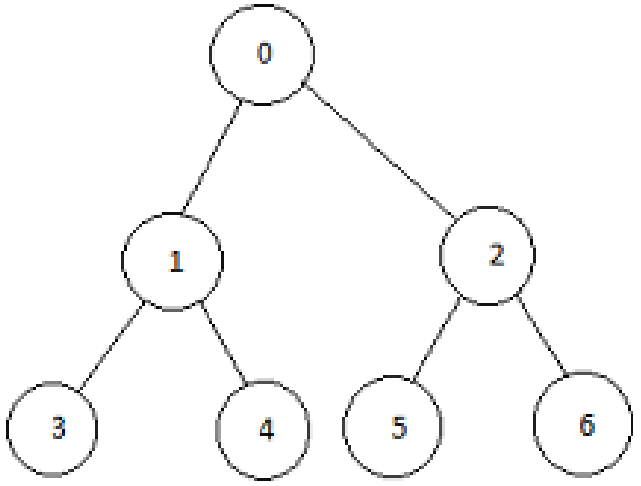
Binary Tree



Full
Binary Tree



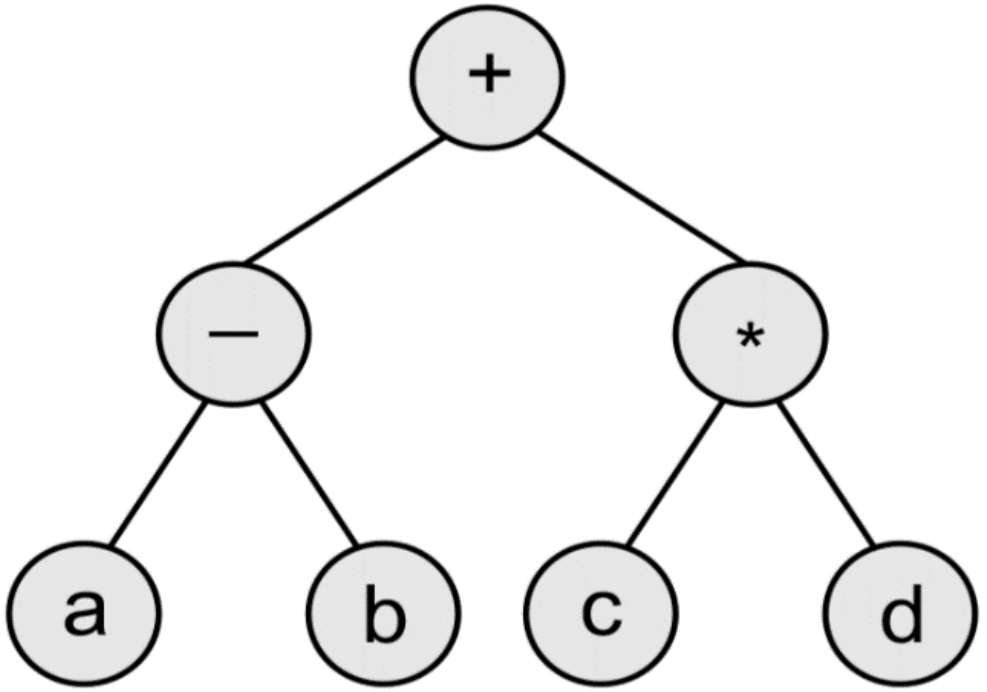
Complete
Binary Tree



Perfect
Binary Tree

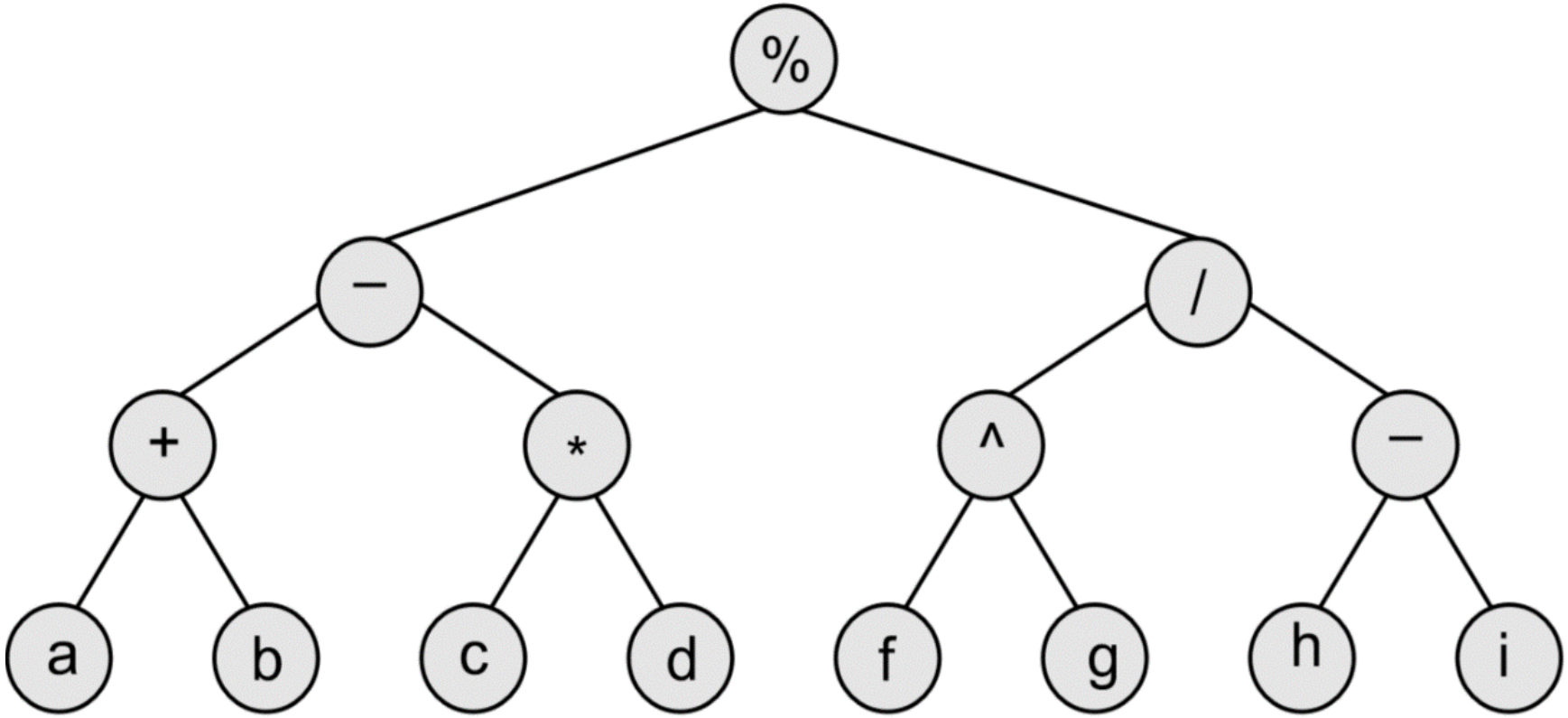
Expression Tree

- Binary Tree can be used to store algebraic expression.
- (a – b) + (c * d)



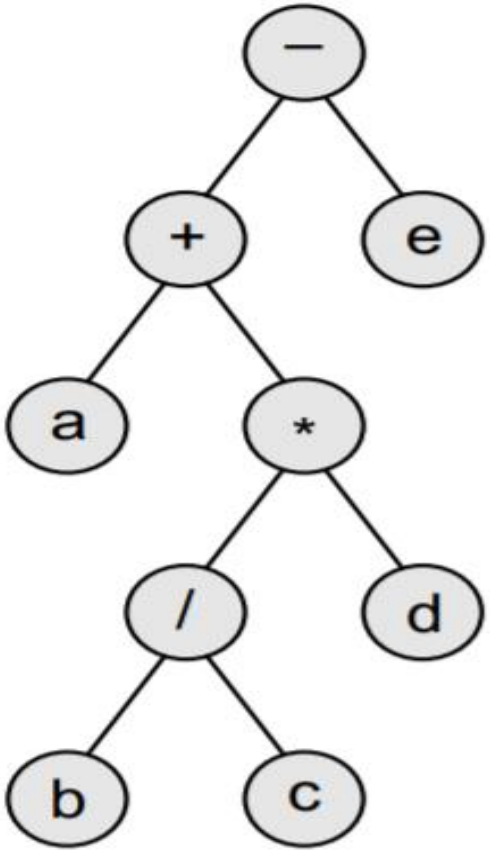
Expression Tree

- ((a + b) – (c * d)) % ((f ^ g) / (h – i))

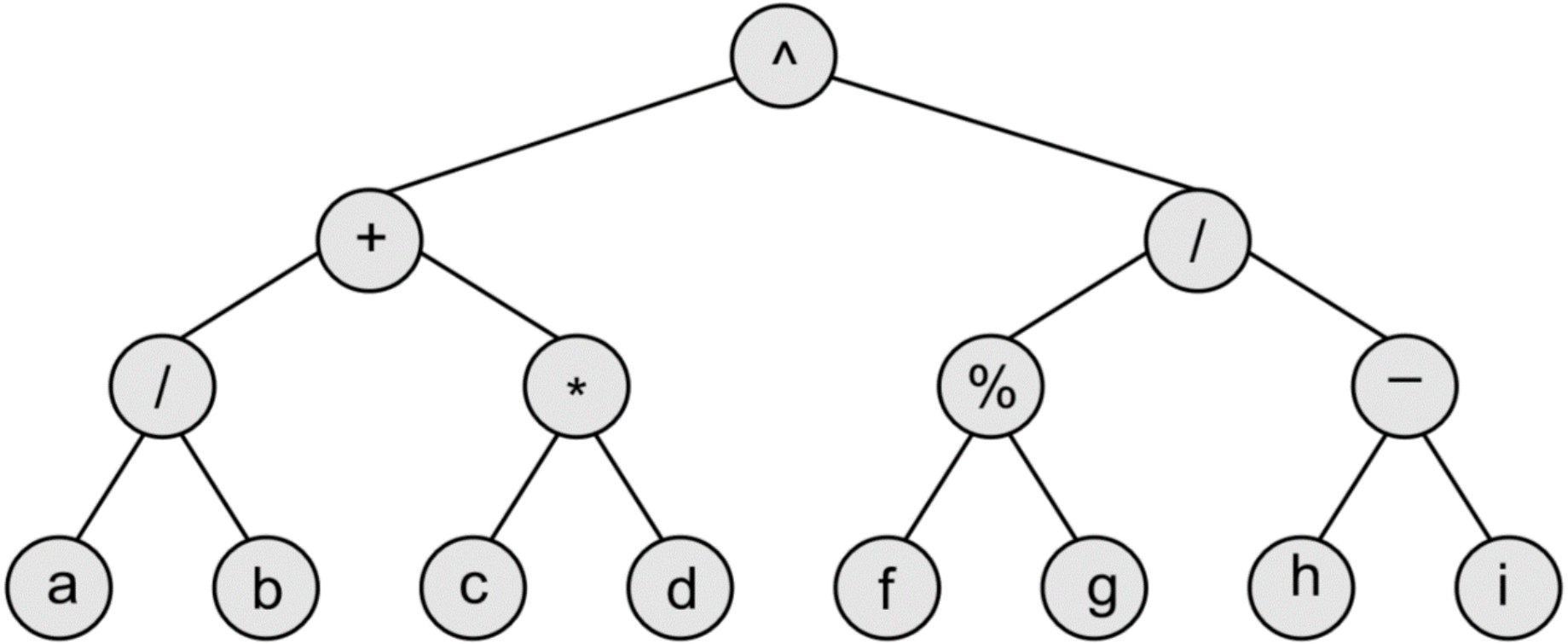


Expression Tree

- a + b / c * d – e



Expression Tree



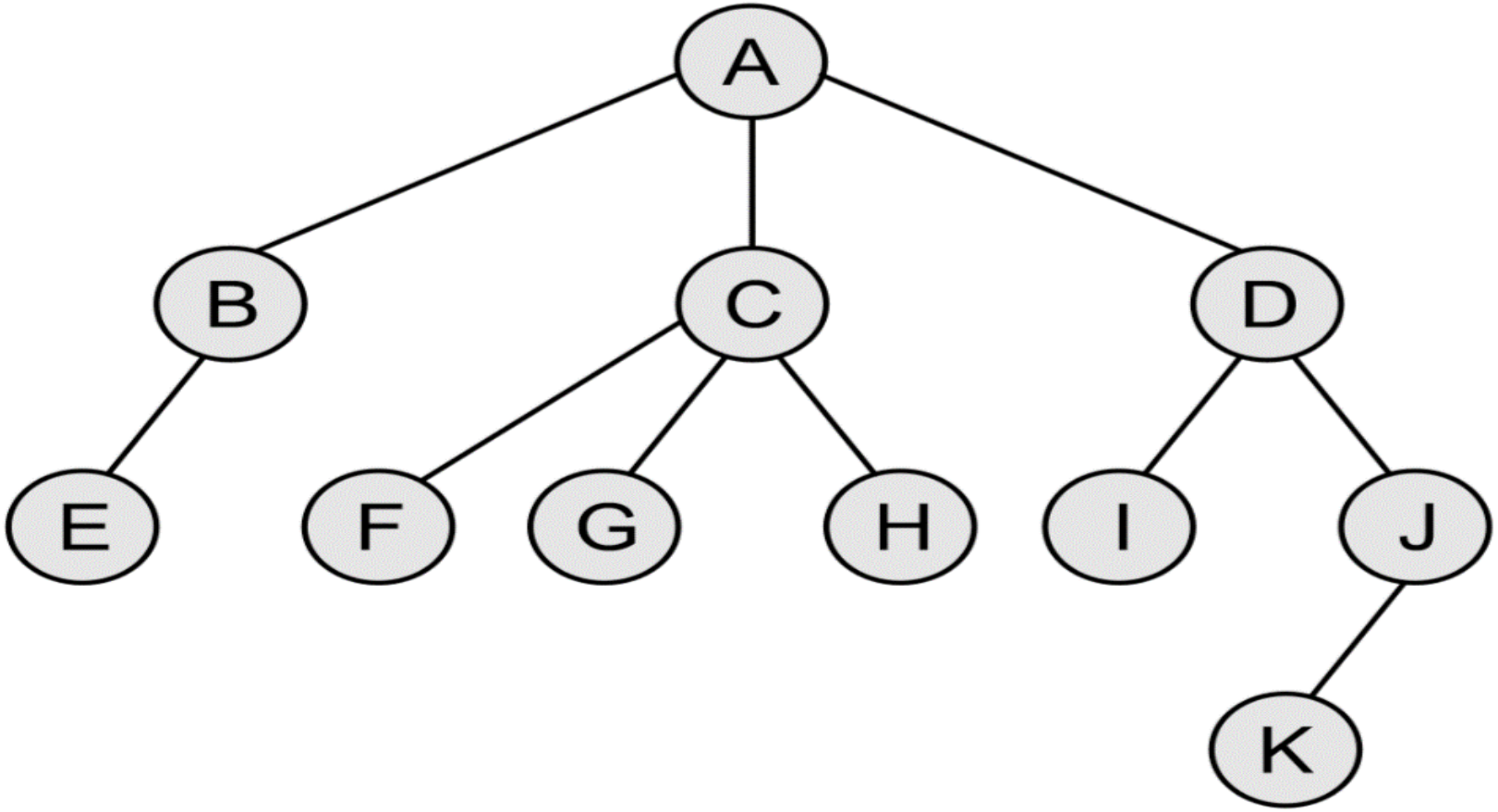
Creating Binary Tree from General Tree

Rule 1: Root of the binary tree = Root of the general tree

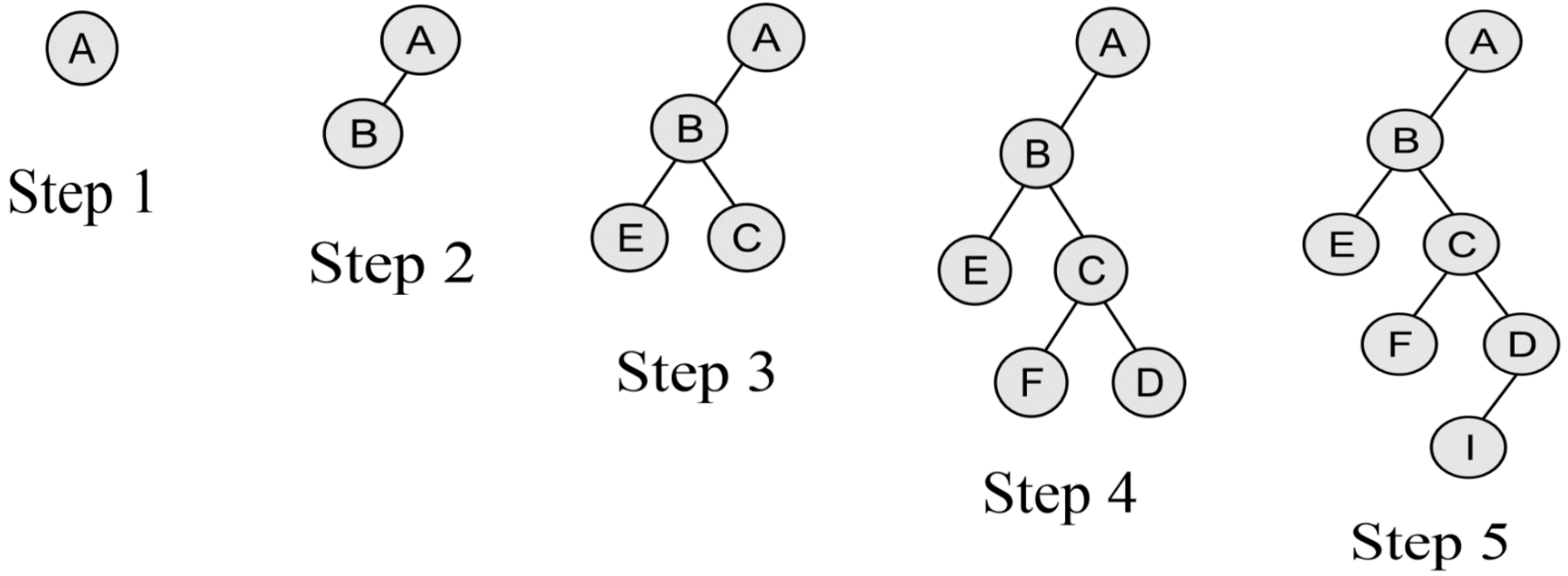
Rule 2: Left child of a node = Leftmost child of the node
in the binary tree in the general tree

Rule 3: Right child of a node
in the binary tree = Right sibling of the node in the general tree

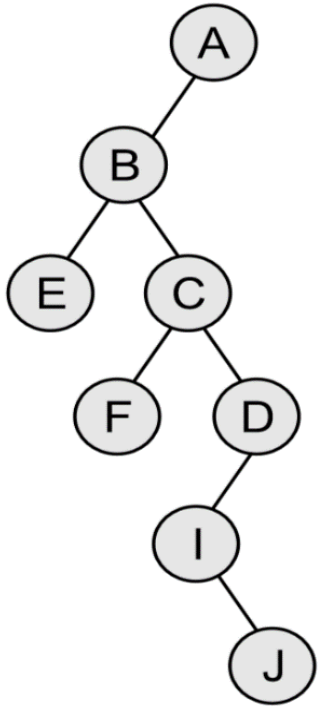
Creating Binary Tree from General Tree



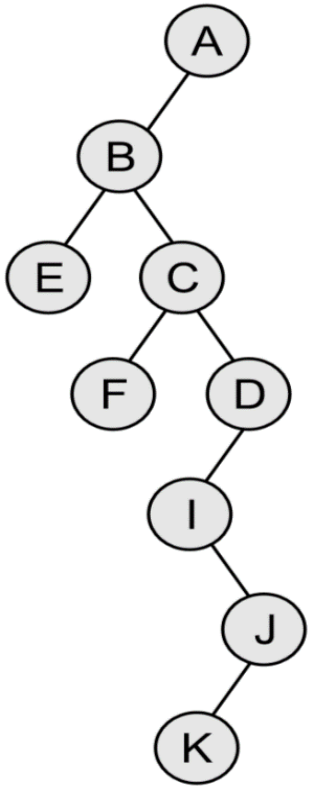
Creating Binary Tree from General Tree



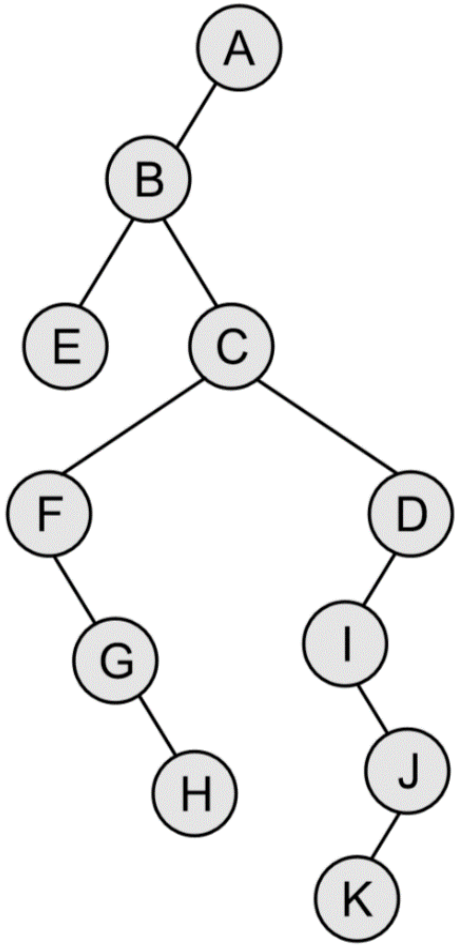
Creating Binary Tree from General Tree



Step 6



Step 7



Step 8

Binary Tree Traversal

- Process of visiting each node in the tree exactly once in a systematic way.
- Nonlinear data structure can be traversed in many different ways.
- Algorithms differ in the order in which the nodes are visited.

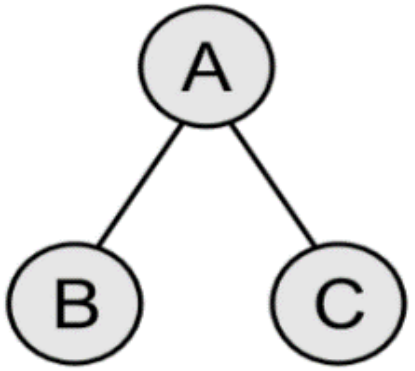
Binary Tree Traversal: Pre-order Traversal

- Following operations are **performed recursively** at each node:
  1. Visiting the root node.
  2. Traversing the left sub-tree.
  3. Traversing the right sub-tree.

```
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
```

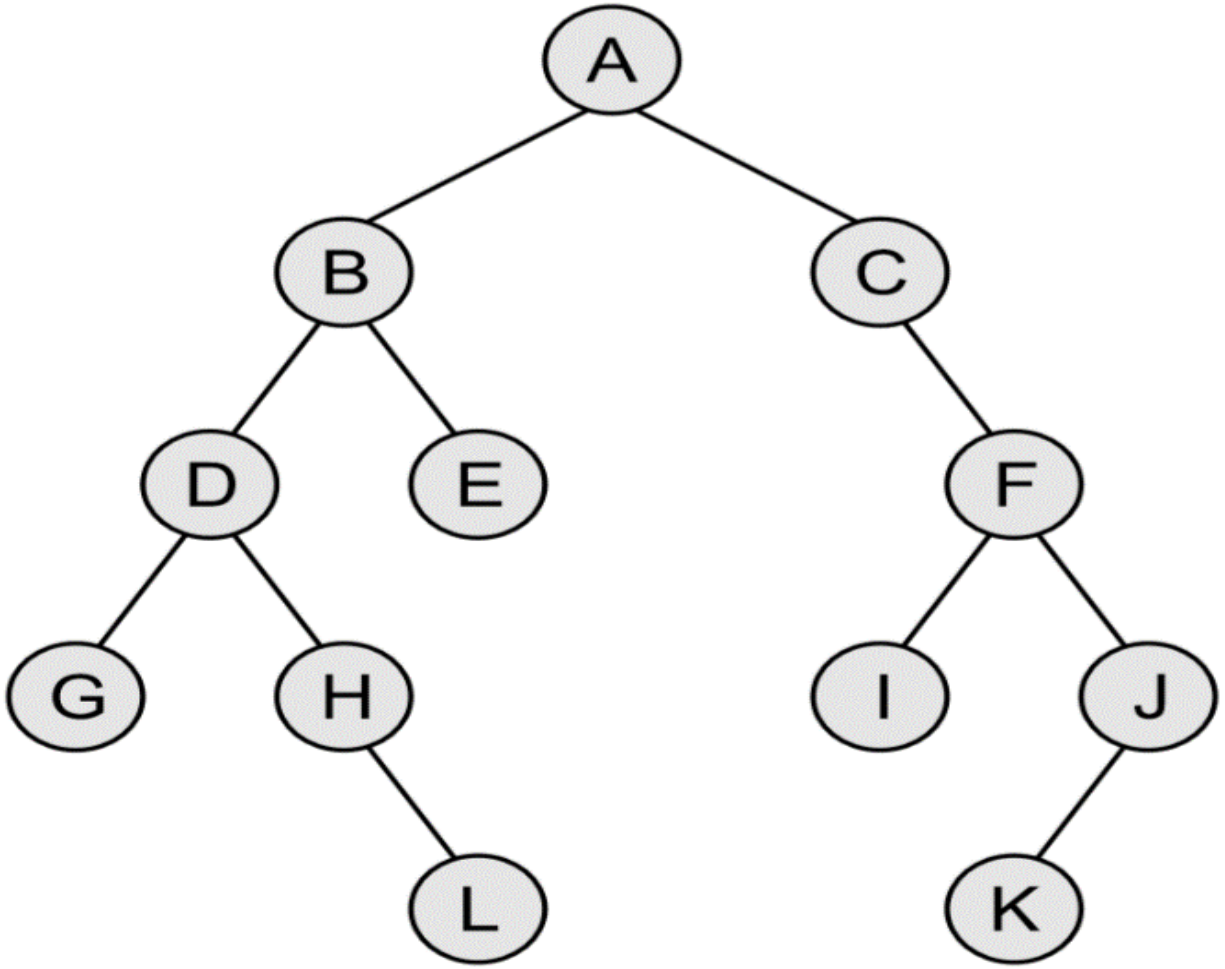
Binary Tree Traversal: Pre-order Traversal

- Pre-order Traversal: A, B, C

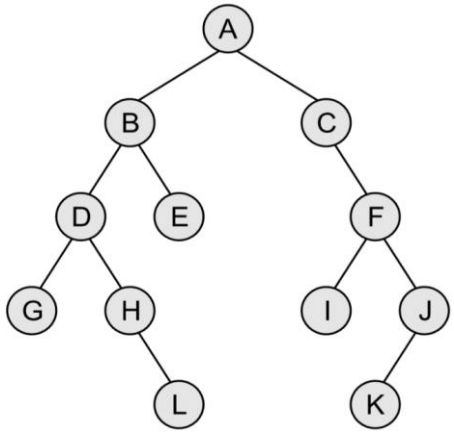


- Depth-first traversal.
- Root node is accessed prior to any other nodes in the left and right sub-trees.
- NLR traversal algorithm (Node-Left-Right).

Binary Tree Traversal: Pre-order Traversal

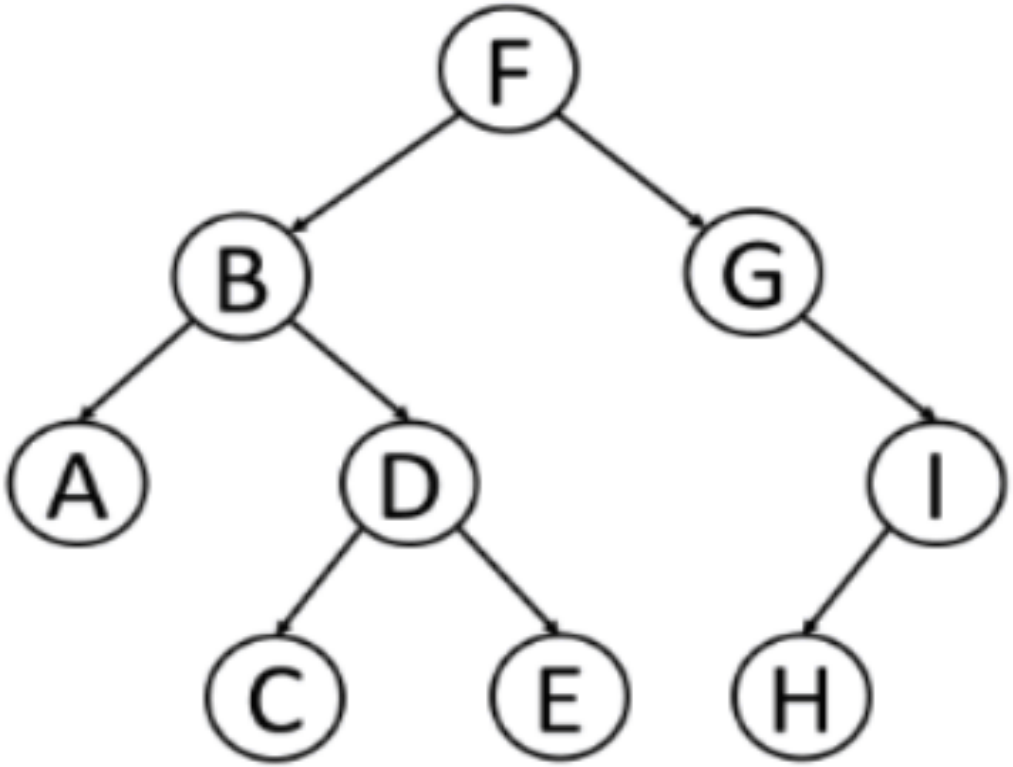


Binary Tree Traversal: Pre-order Traversal



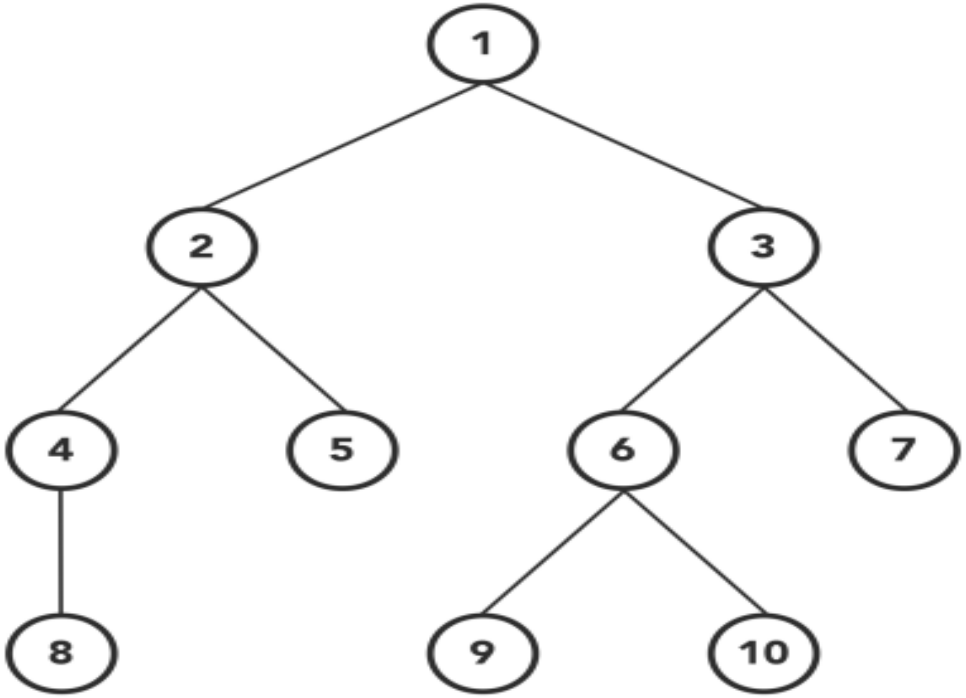
Binary Tree Traversal: Pre-order Traversal

Binary Tree Traversal: Pre-order Traversal

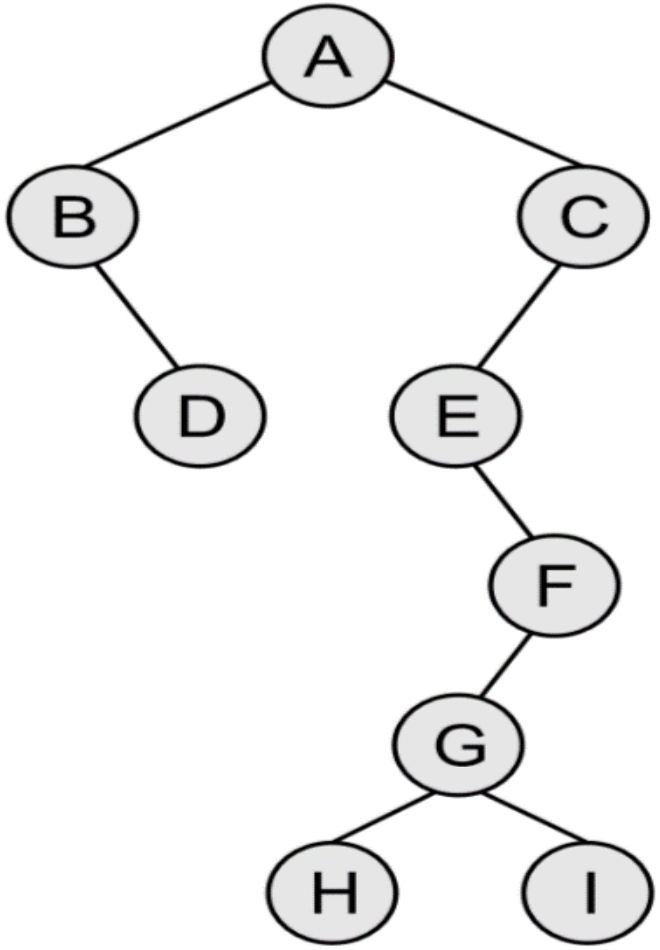


Preorder:

Binary Tree Traversal: Pre-order Traversal

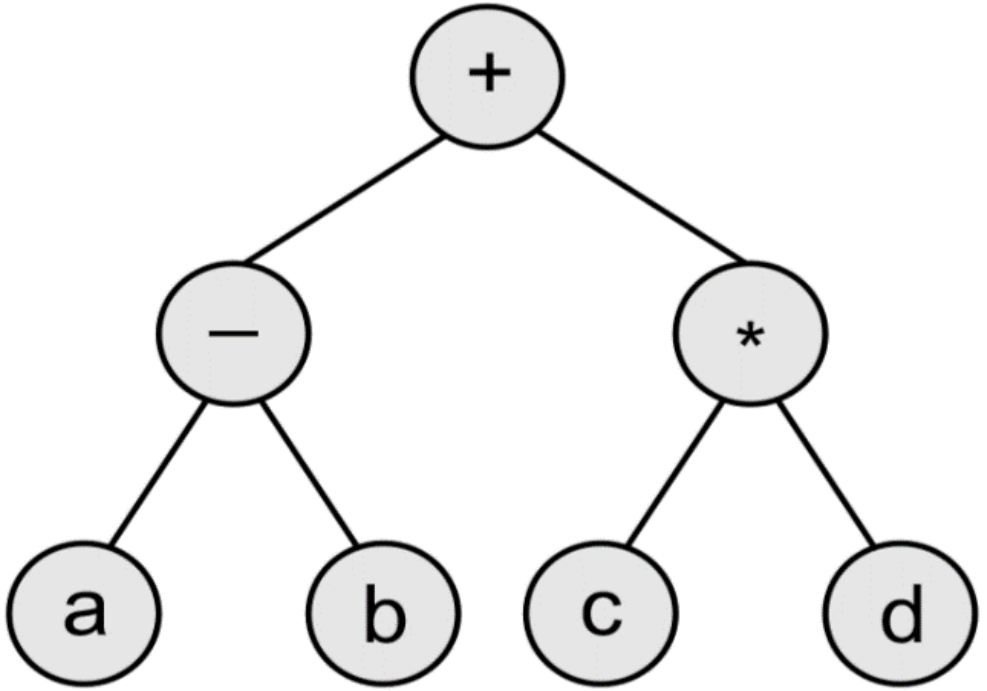


Binary Tree Traversal: Pre-order Traversal



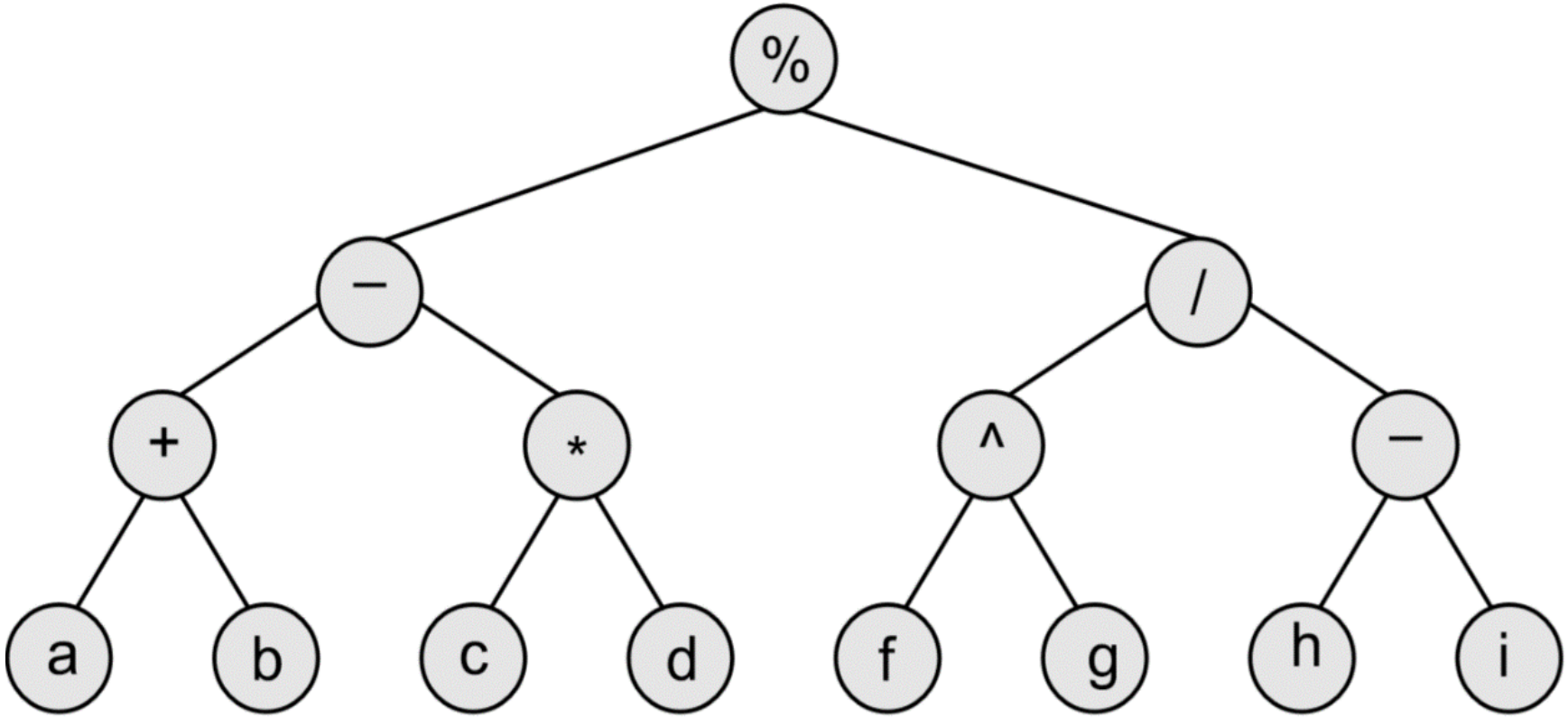
Binary Tree Traversal: Pre-order Traversal

- (a – b) + (c * d)



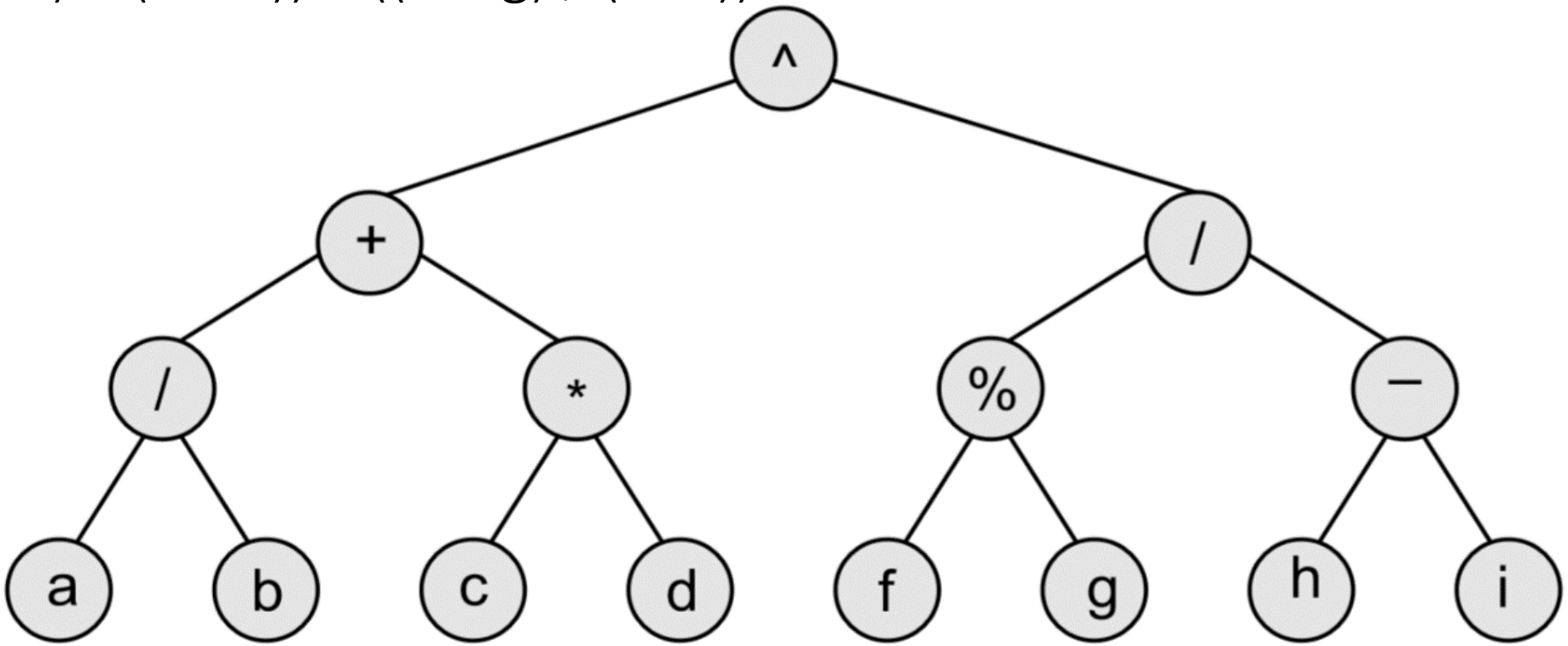
Binary Tree Traversal: Pre-order Traversal

- ((a + b) – (c * d)) % ((f ^ g) / (h – i))



Binary Tree Traversal: Pre-order Traversal

- ((a / b) + (c * d)) ^ ((f % g) / (h - i))



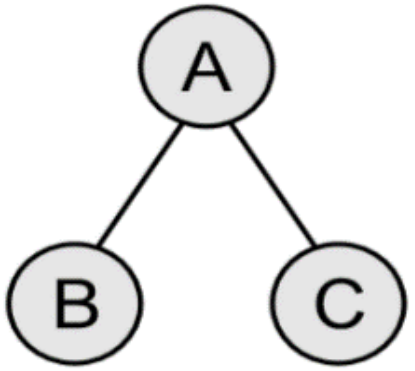
Binary Tree Traversal: In-order Traversal

- Following operations are **performed recursively** at each node:
  1. Traversing the left sub-tree.
  2. Visiting the root node.
  3. Traversing the right sub-tree.

```
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
```

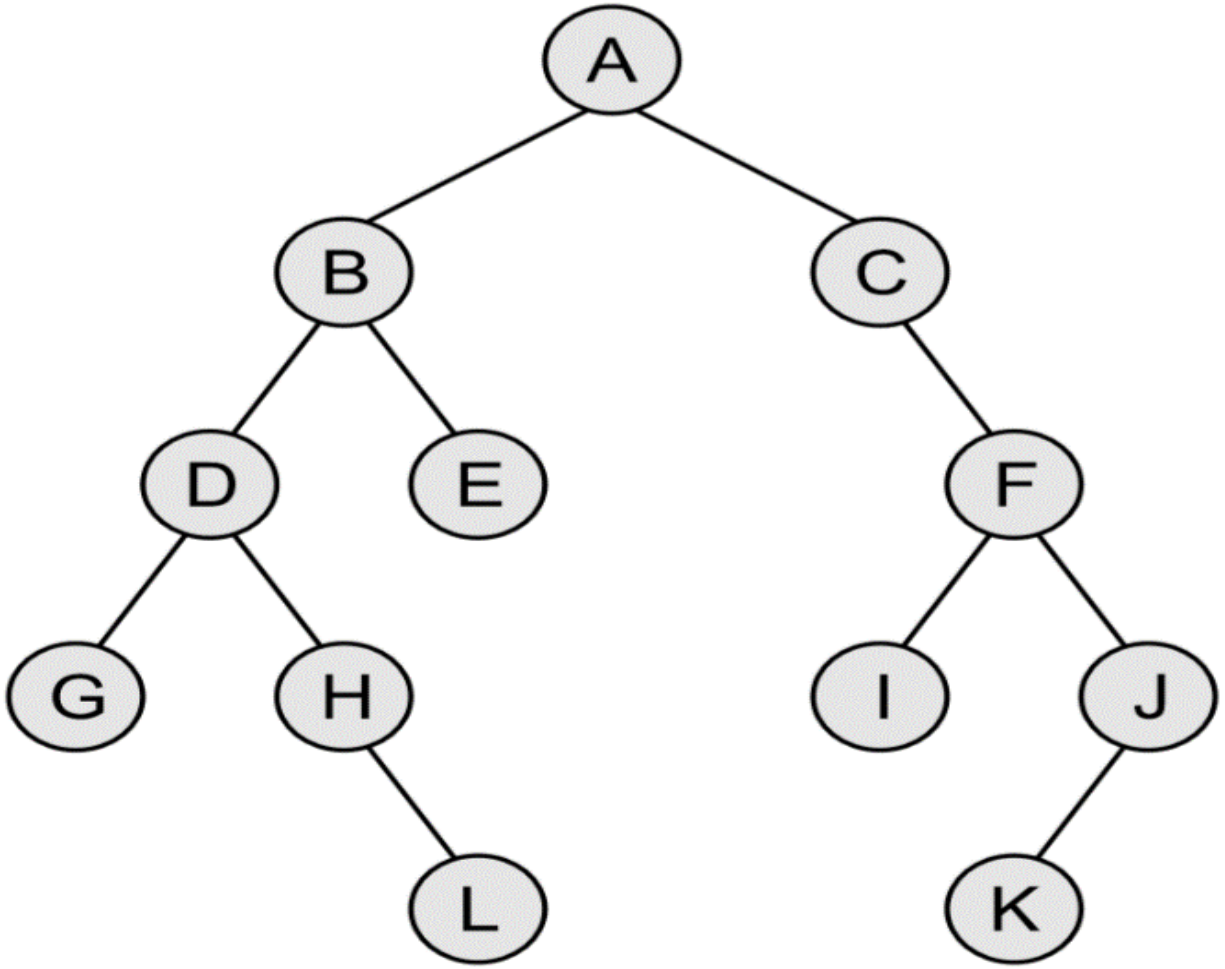
Binary Tree Traversal: In-order Traversal

- In-order Traversal: B, A, C

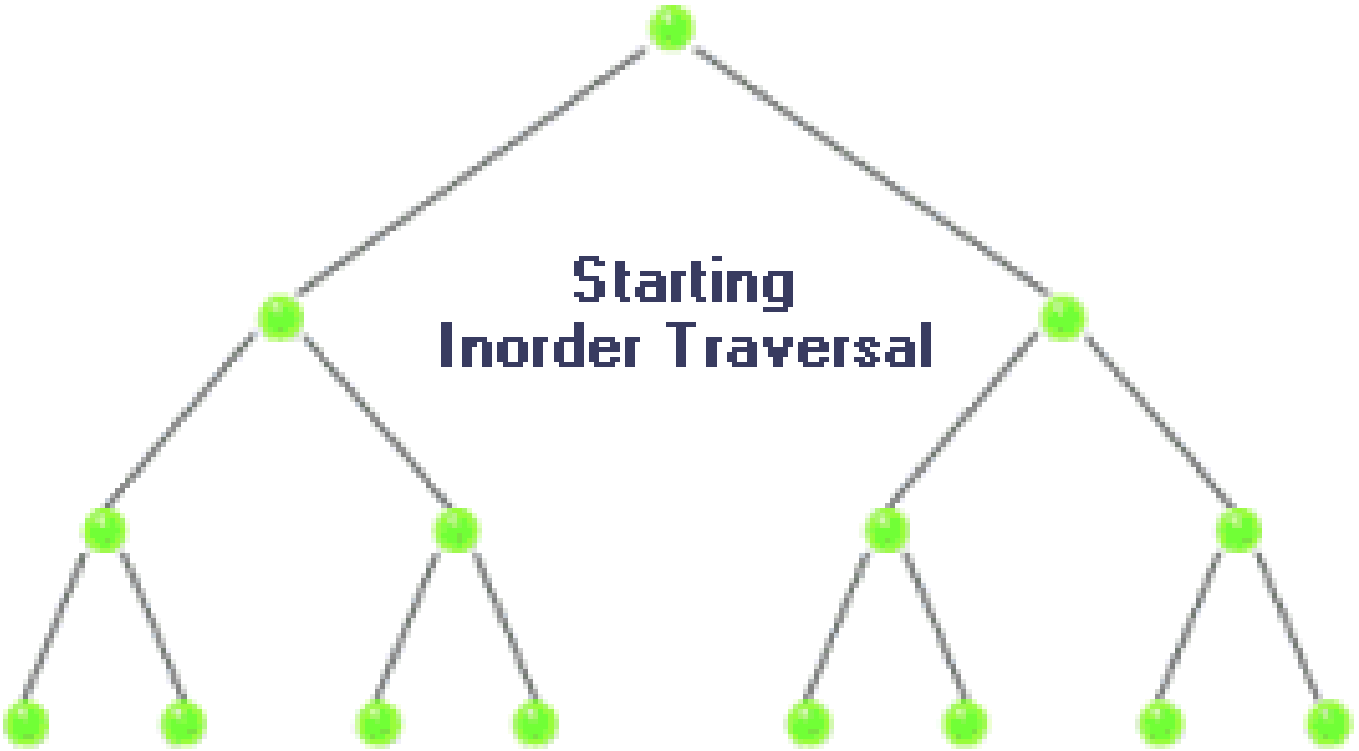


- Symmetric traversal.
- Left sub-tree is always traversed before the root node and the right sub-tree.
- LNR traversal algorithm.

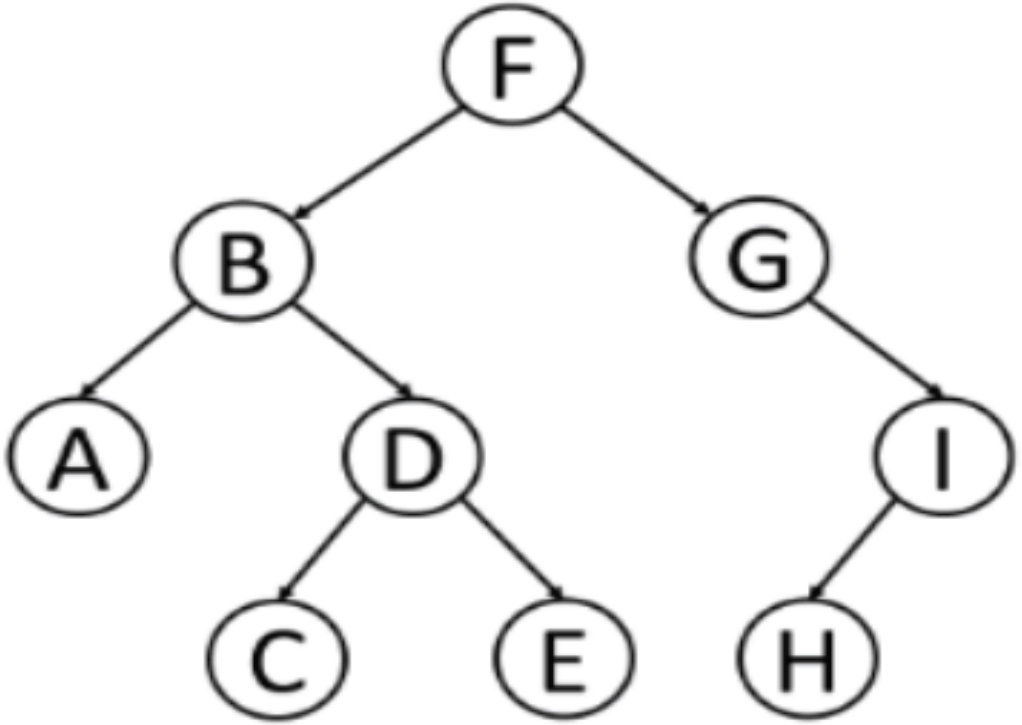
Binary Tree Traversal: In-order Traversal



Binary Tree Traversal: In-order Traversal

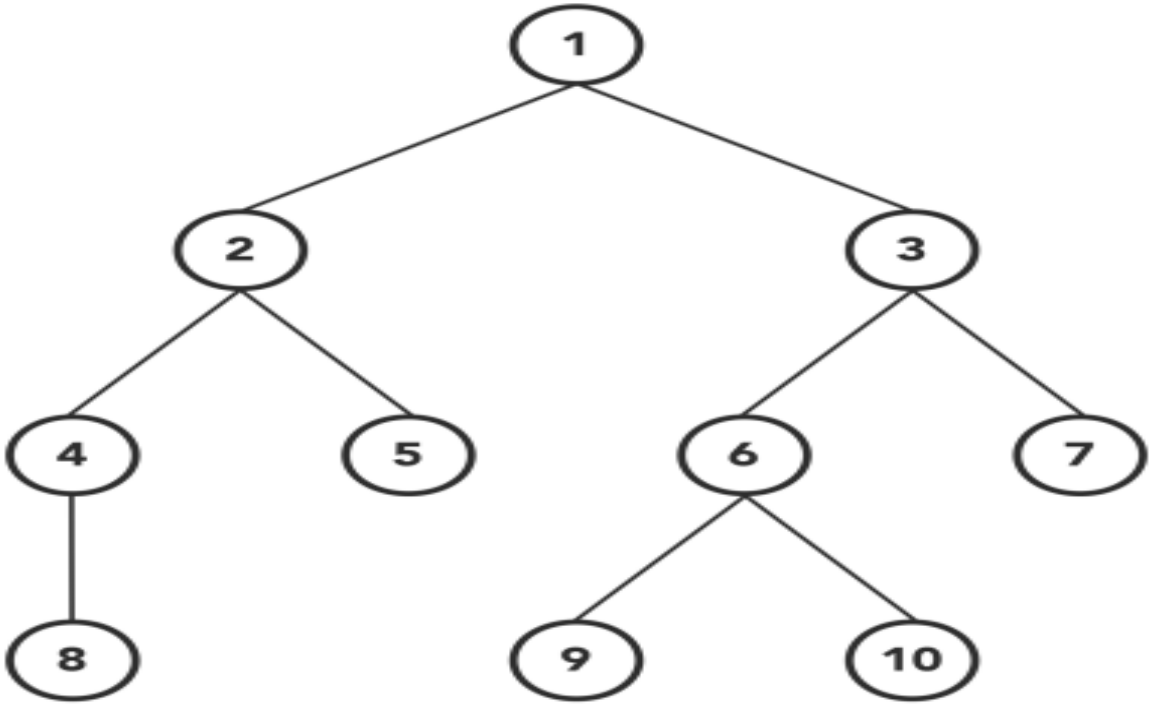


Binary Tree Traversal: In-order Traversal

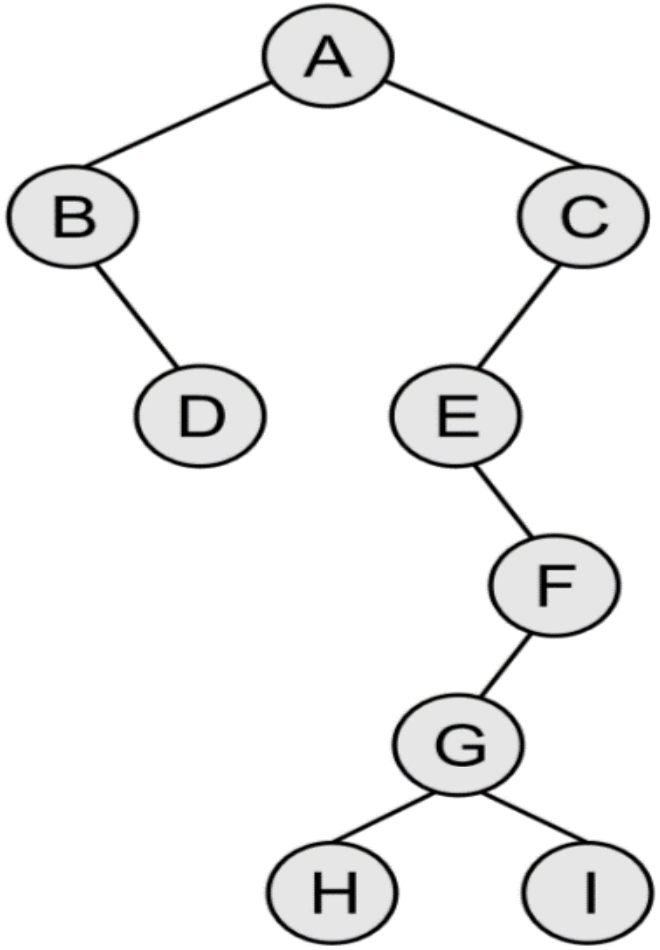


Inorder:

Binary Tree Traversal: In-order Traversal



Binary Tree Traversal: In-order Traversal

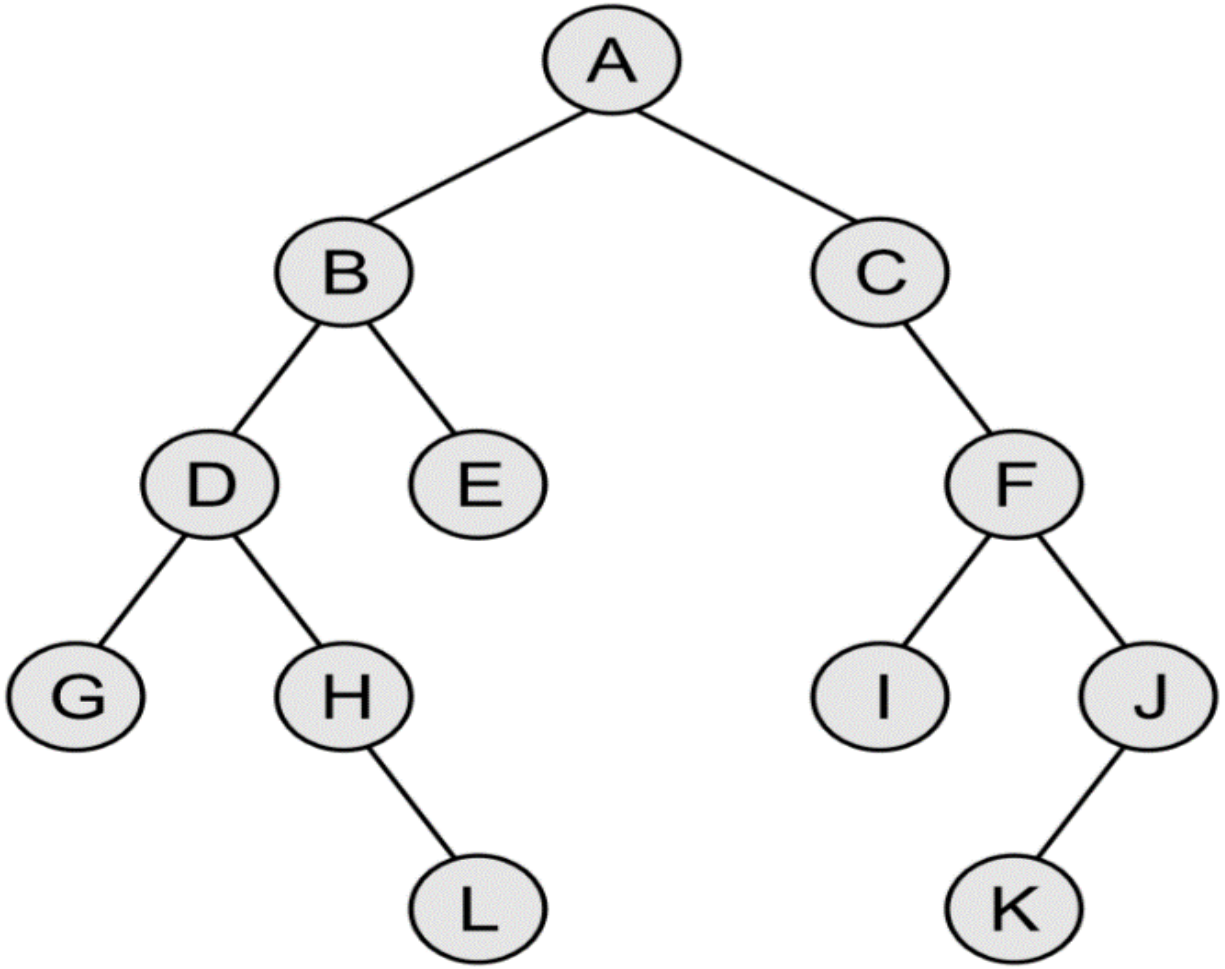


Binary Tree Traversal: Post-order Traversal

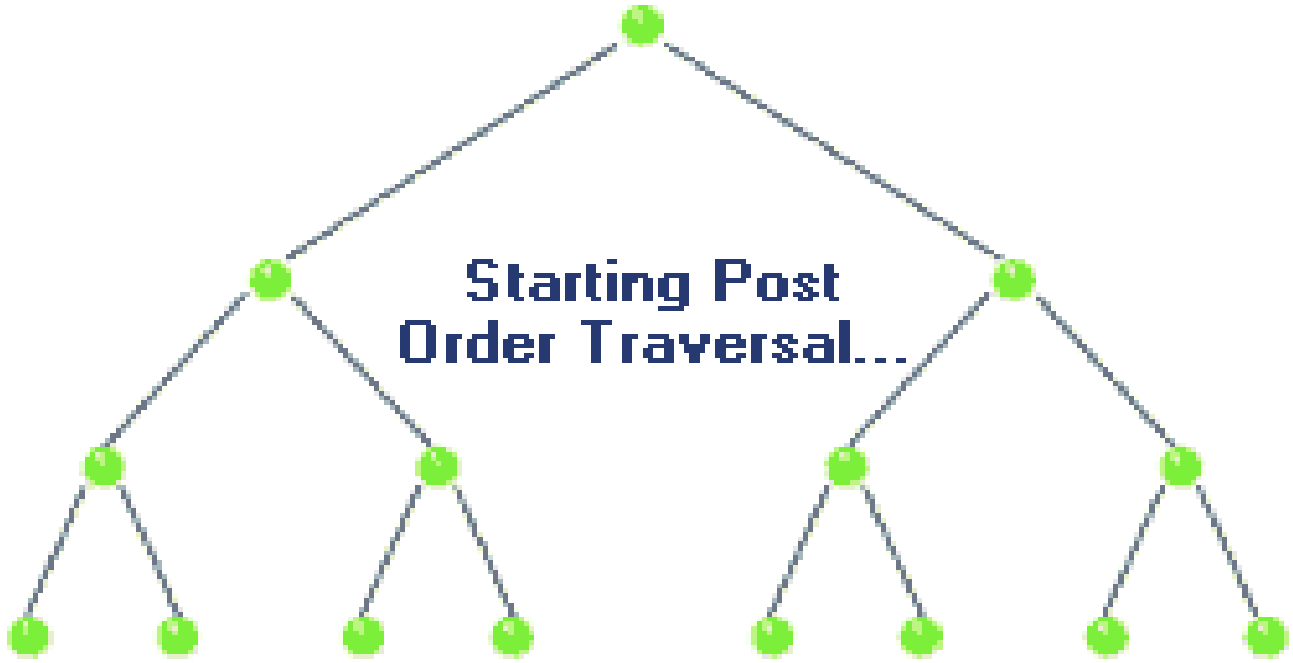
- Following operations are **performed recursively** at each node:
  1. Traversing the left sub-tree.
  2. Traversing the right sub-tree.
  3. Visiting the root node.

```
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
```

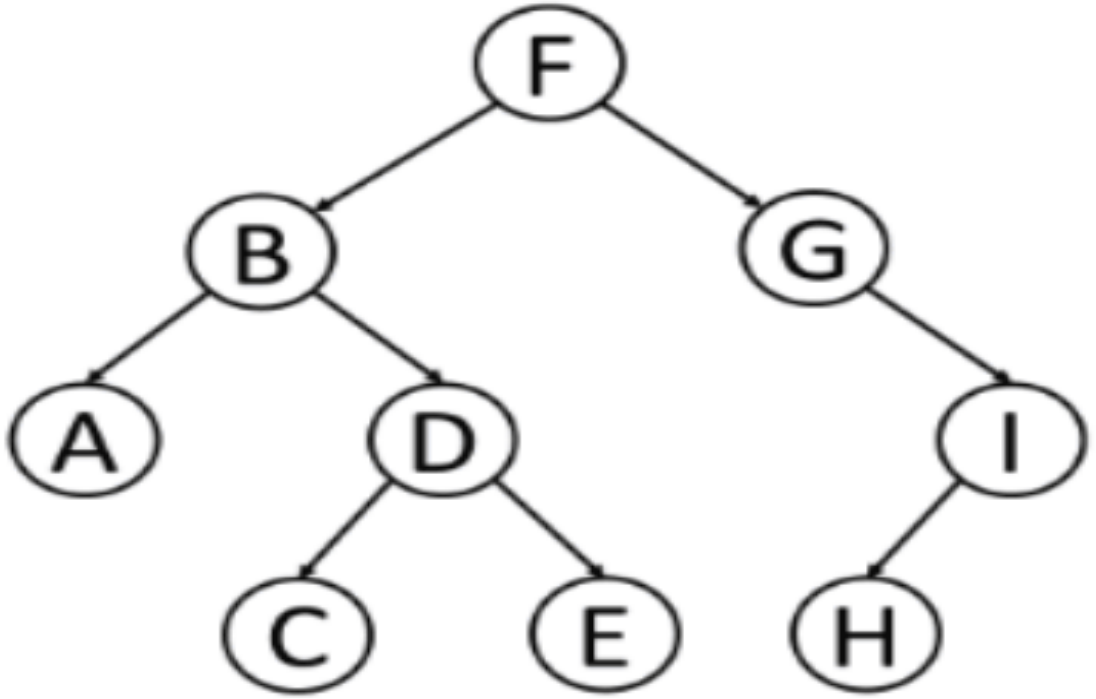
Binary Tree Traversal: Post-order Traversal



Binary Tree Traversal: Post-order Traversal

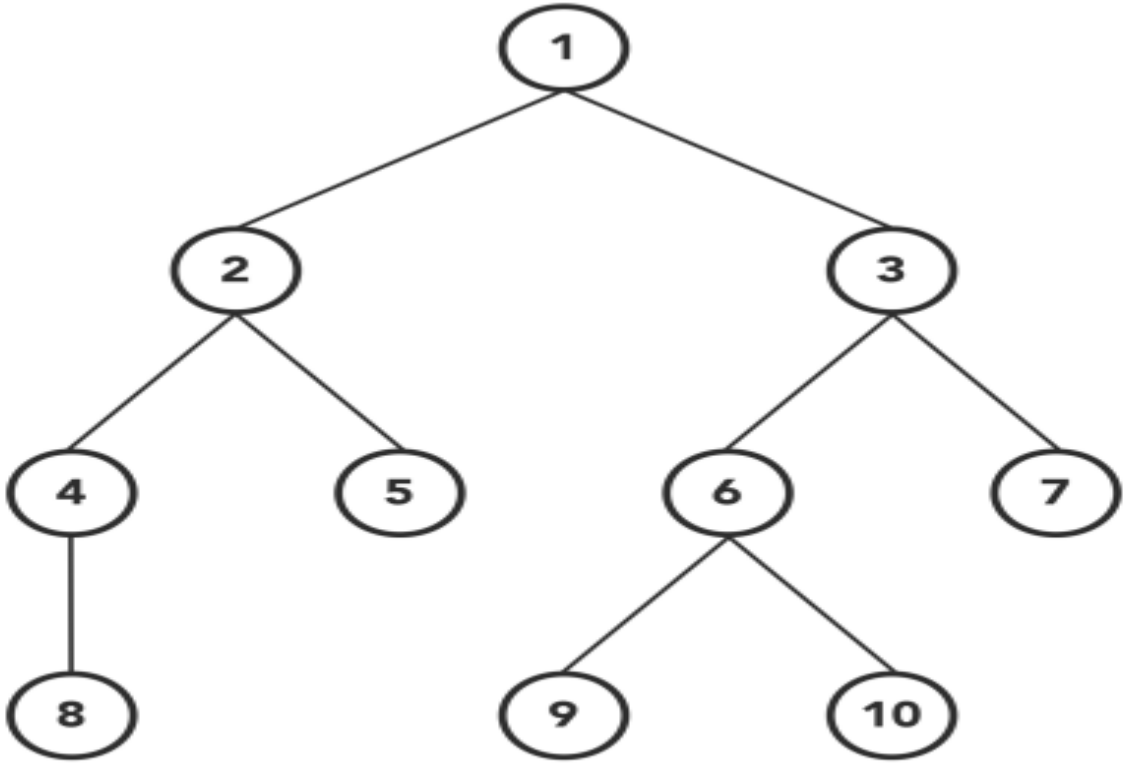


Binary Tree Traversal: Post-order Traversal

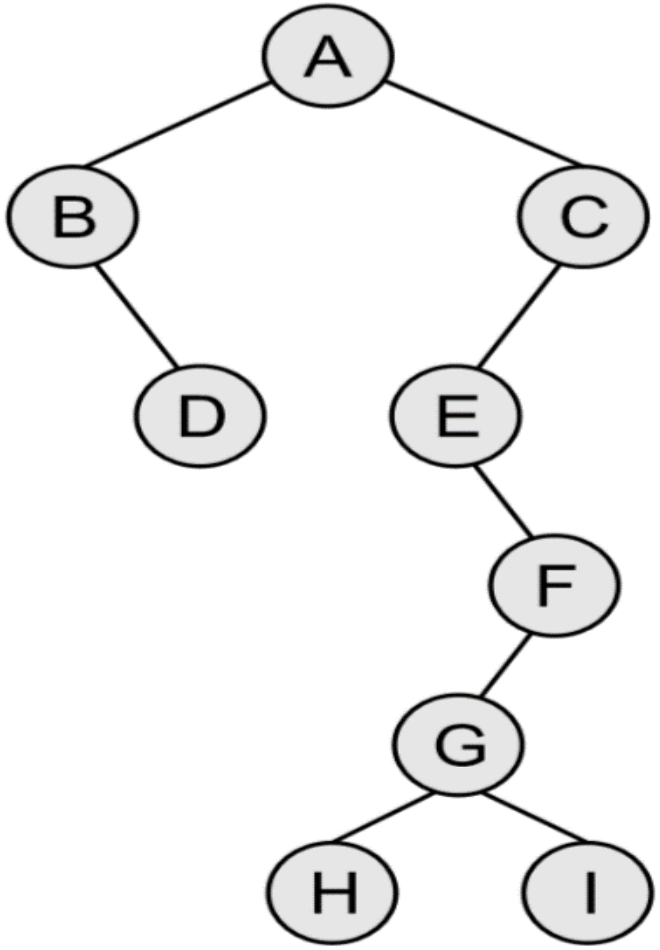


Postorder:

Binary Tree Traversal: Post-order Traversal

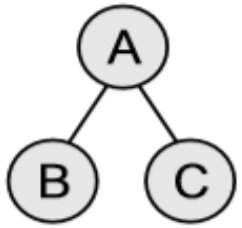


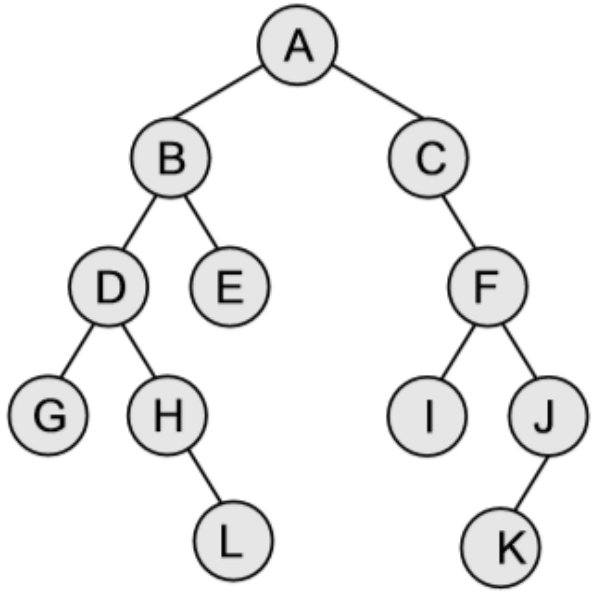
Binary Tree Traversal: Post-order Traversal

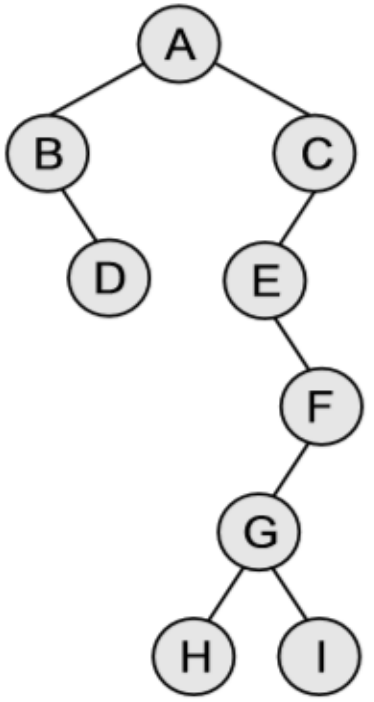


Binary Tree Traversal: Level-order Traversal

- All the nodes at a level are accessed before going to the next level.







- Breadth First Traversal.

Constructing Binary Tree from Traversal Results

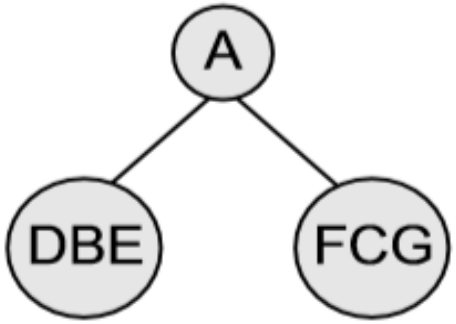
- At least two traversal results.
- One must be the **in-order traversal** and the second can **be either pre-order or post-order traversal**.

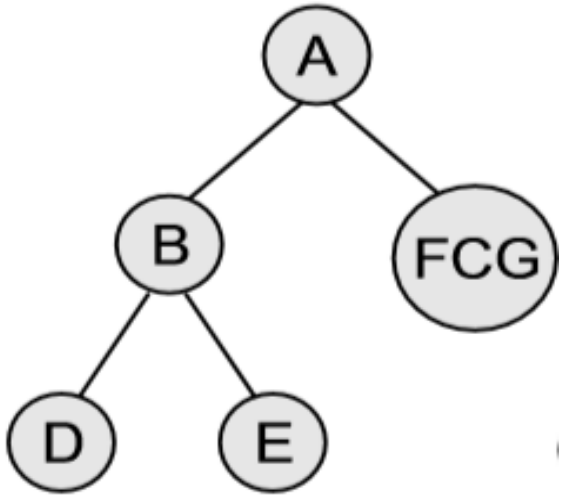
Constructing Binary Tree from Traversal Results

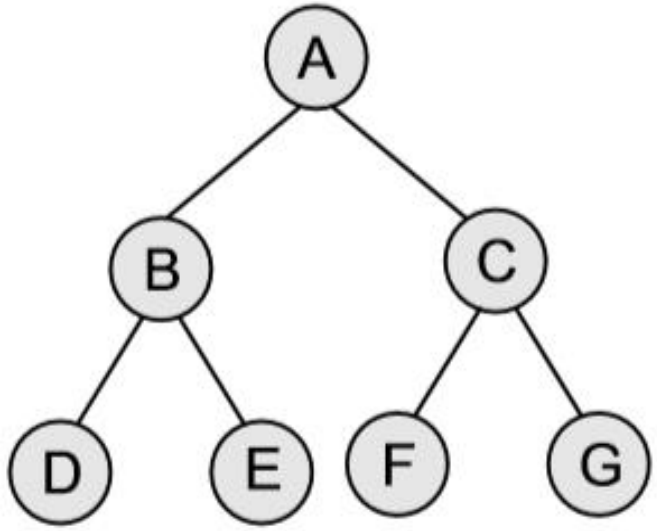
- *Step 1:* Use the pre-order/post-order sequence to determine the root node of the tree.
- *Step 2:* Elements on the left side of the root node in the in-order traversal sequence form the left sub-tree of the root node. Elements on the right side of the root node in the in-order traversal sequence form the right sub-tree of the root node.
- *Step 3:* Recursively select each element from pre-order/post-order traversal sequence and create its left and right sub-trees from the in-order traversal sequence.

Constructing Binary Tree from Traversal Results

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







Constructing 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