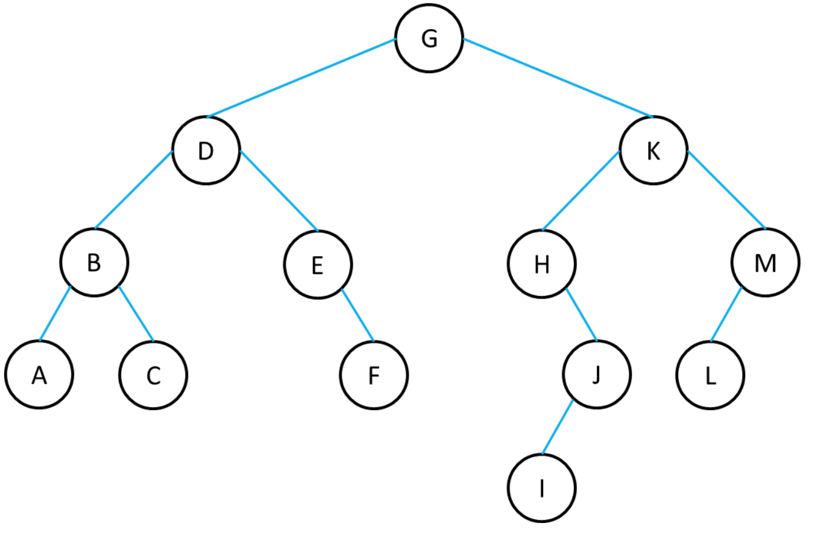
Trees and Binary Trees

Introduction

- Trees are one of the fundamental data structures in computer science.
- Unlike Array and Linked List, which are linear data structures, tree is hierarchical (or non-linear) data structure.
  - File system, organization chart, network topology, etc.



Introduction

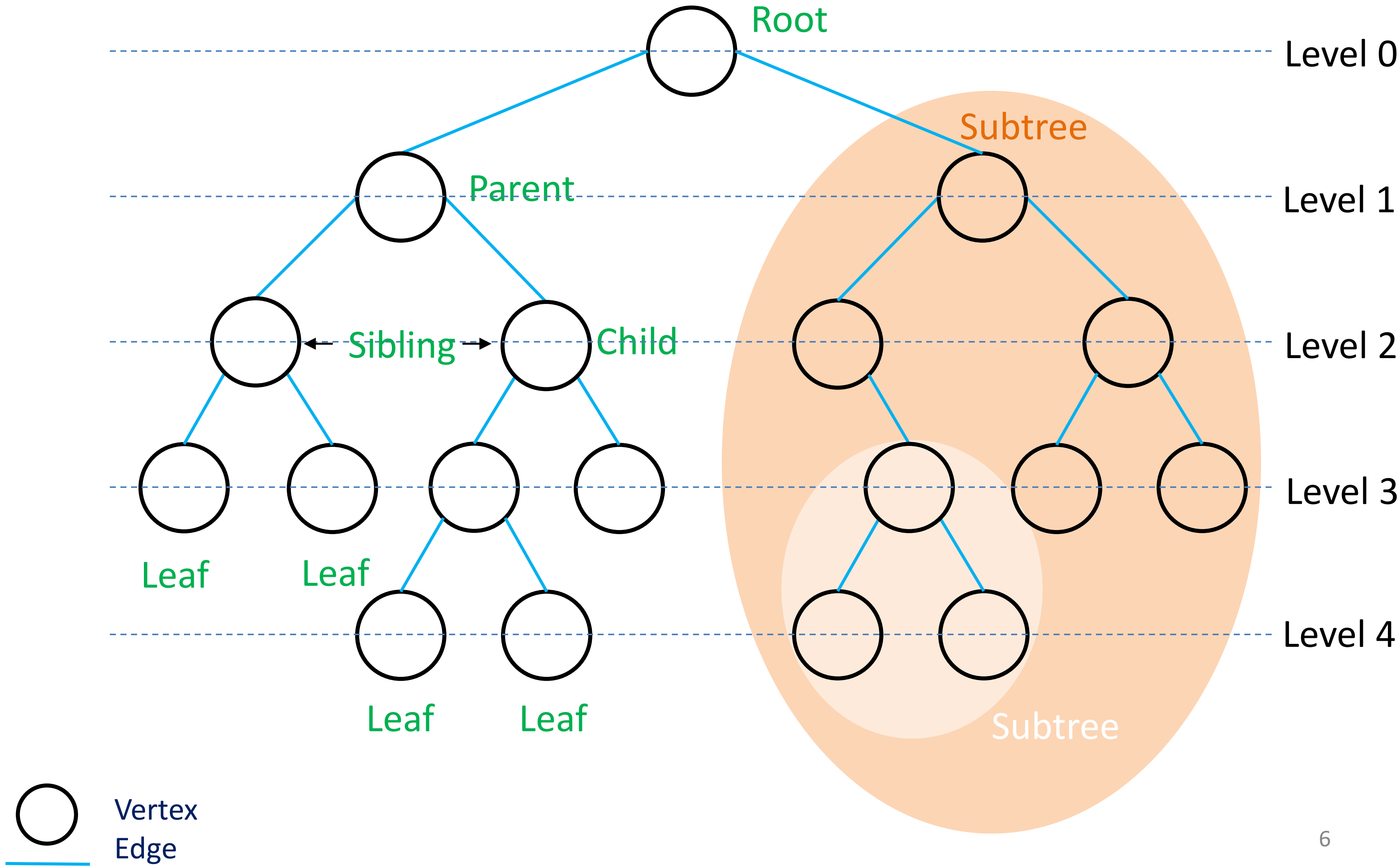
- Advantages:
  - Searching efficiency
  - Sorting efficiency
  - Dynamic data
  - Efficient insertion and deletion
  - Easy to implement

Outline

- Terminology of Trees
- Basic Properties of Trees
- Classification of Binary Trees
- Implementation of Tree Algorithms
  - Number of nodes in a tree
  - Height of a tree
  - Traversal of a binary tree

Terminology of Trees

Terminology: Basic Definitions



Terminology

- Trees consist of vertices and edges.
- Vertex: An object that carries associated information. (node)
- Edge: A connection between two vertices. A link from one node to another.

Terminology

- Child/Parent
  - If either the right or left link of A is a link to B, then B is a child of A and A is a parent of B.
- Sibling
  - Nodes that have the same parent.
- Root
  - A node that has no parent. There is only one root in a tree.
- Leaf
  - A node with no children
  - Also called external node or terminal node
- Non-Leaf
  - A node with at least one child
  - Also called internal node or non-terminal node
- Subtree
  - Any given node, with all of its descendants (children, grandchildren, great grandchildren etc.).

Terminology

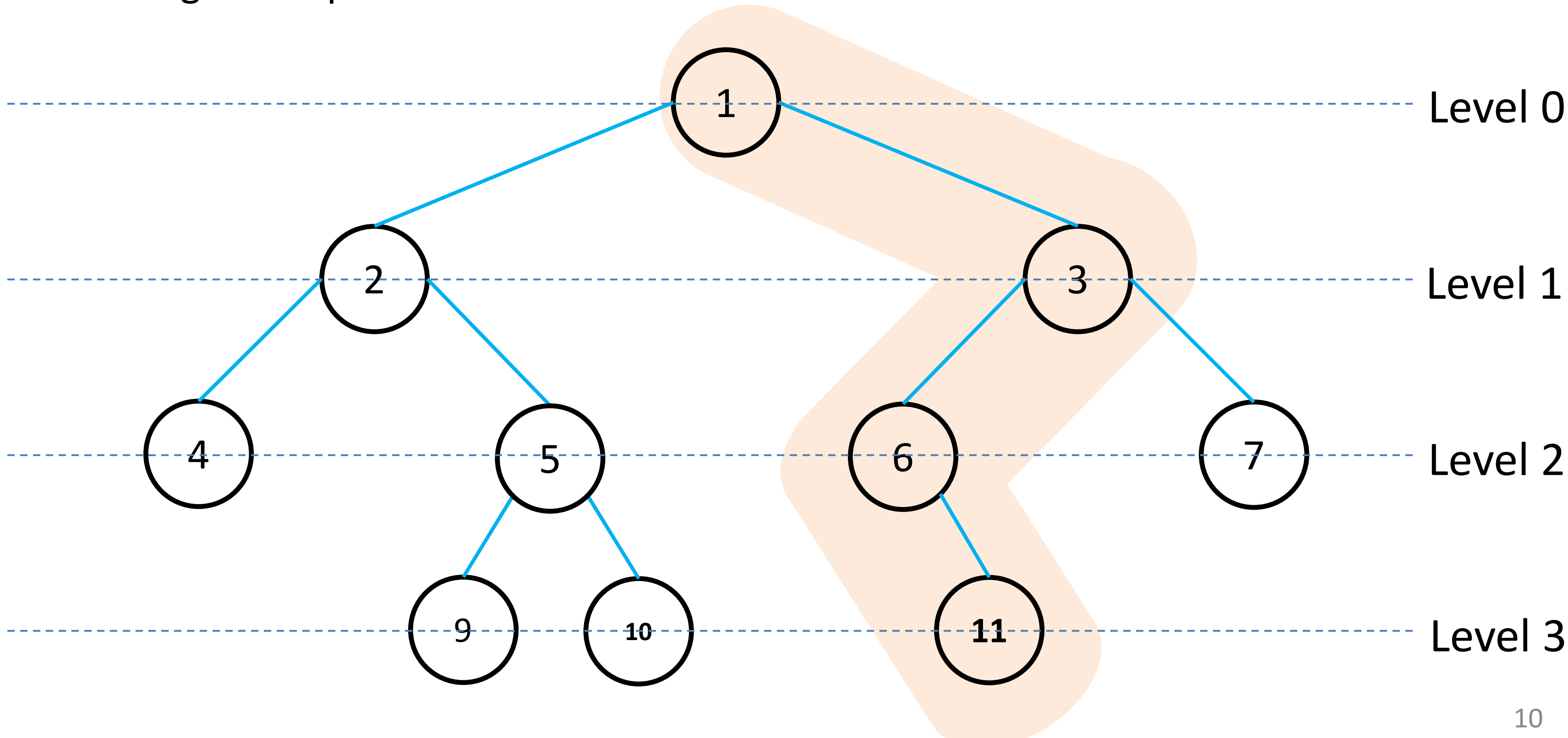
- The level of a given node in a tree is defined recursively as:

$$\text{Level\_of\_a\_node} = \text{Level\_of\_its\_parent} + 1$$

  - The root node is said to be at level 0.
  - The root node's children are at level 1.
  - The children of the node at level 1 will be level 2, and so on.

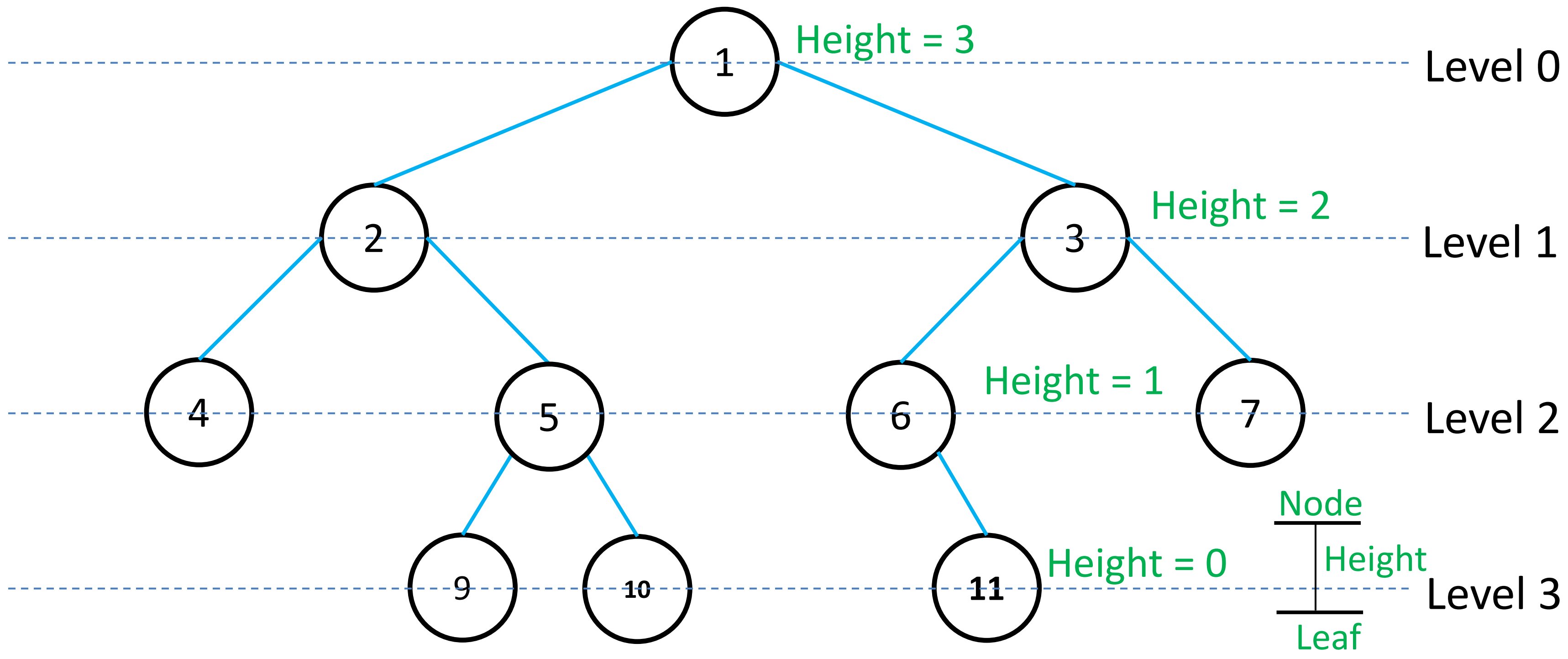
Terminology: Path and Length

- Path: A sequence of nodes where each adjacent pair is connected by an edge.
  - The path from node 1 to node 11 is represented by the sequence of nodes 1, 3, 6, 11.
- Length of a path: The number of edges on the path
  - The length of this path is 3 (3 edges).
- The length of a path of k nodes is $k - 1$.



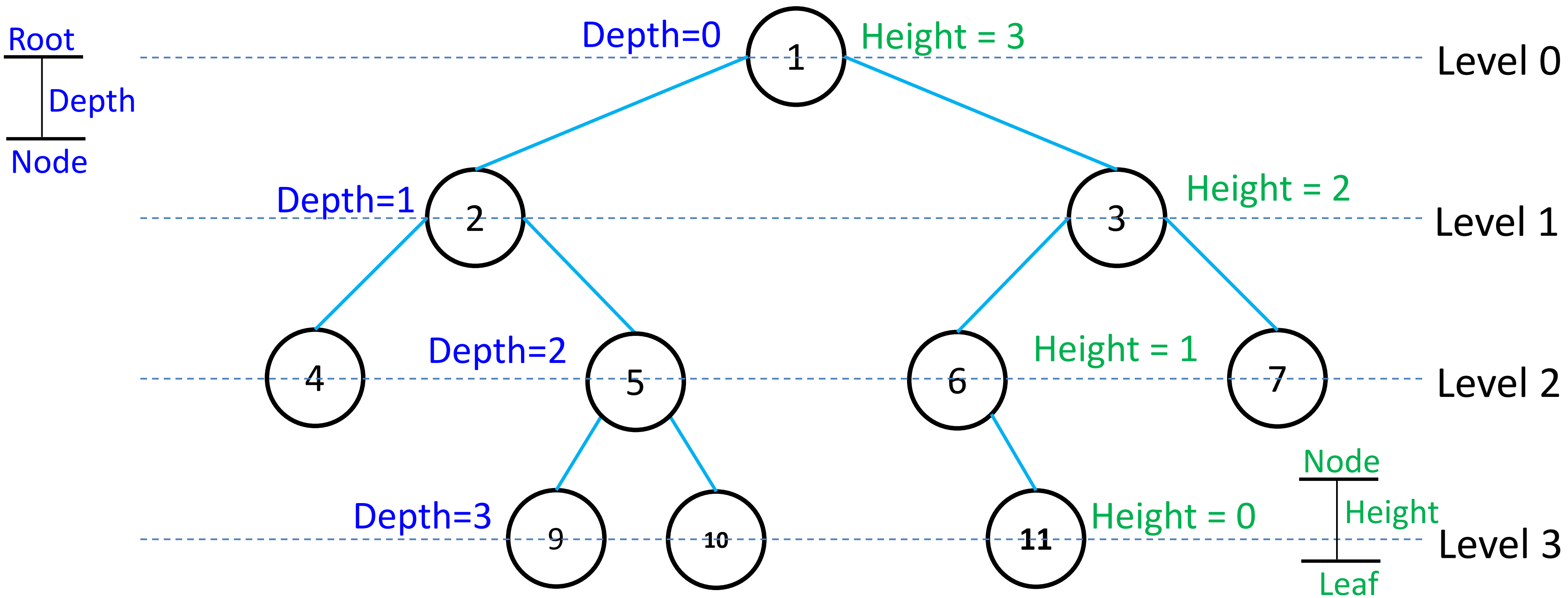
Terminology: Height

- Height of a node
  - The length of the longest path from the node to a leaf.
  - i.e., the number of edges from the node to the deepest leaf
- Height of a tree
  - the height of the root node



Terminology: Depth

- Depth of a node
  - The length of the path from the root to a node
  - i.e., The number of edges in the path from the root to the node
  - = level of a node in a tree

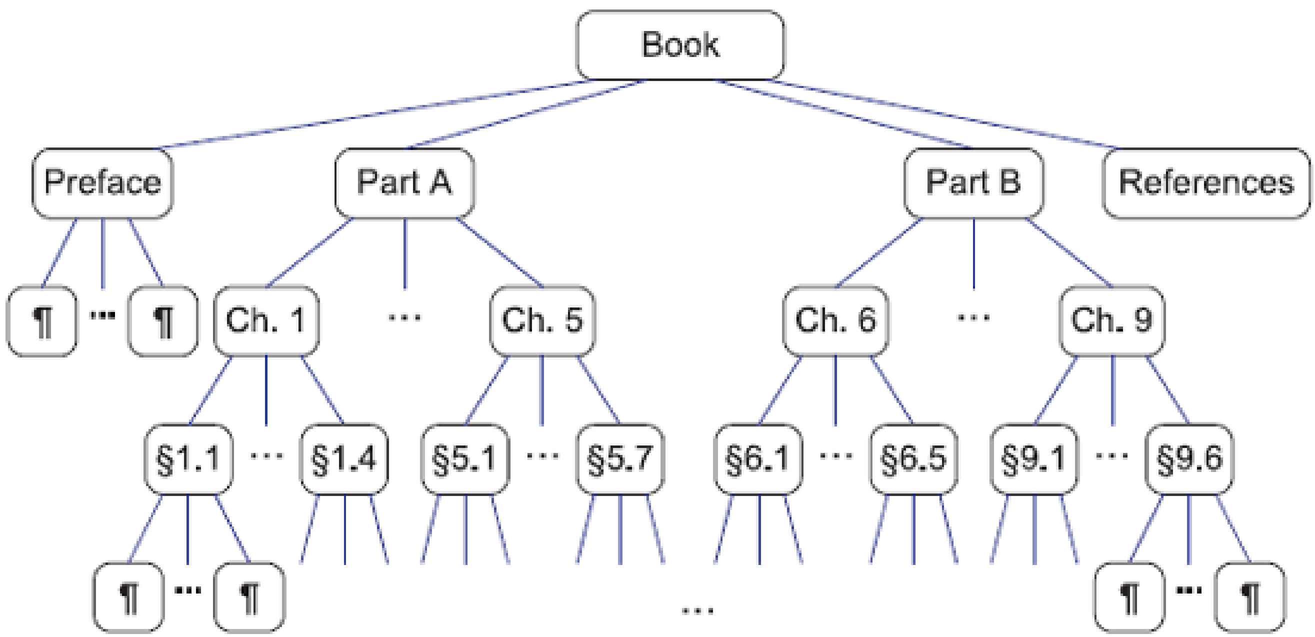


Terminology: Height of a Tree

- Height of the tree
  - Height of the root node
  - Maximum depth of its nodes
  - Maximum levels of its nodes
- Special case:
  - A tree consisting of 1 node (the root) has a height of 0.
  - An empty tree (a tree with no nodes) has a height of -1.

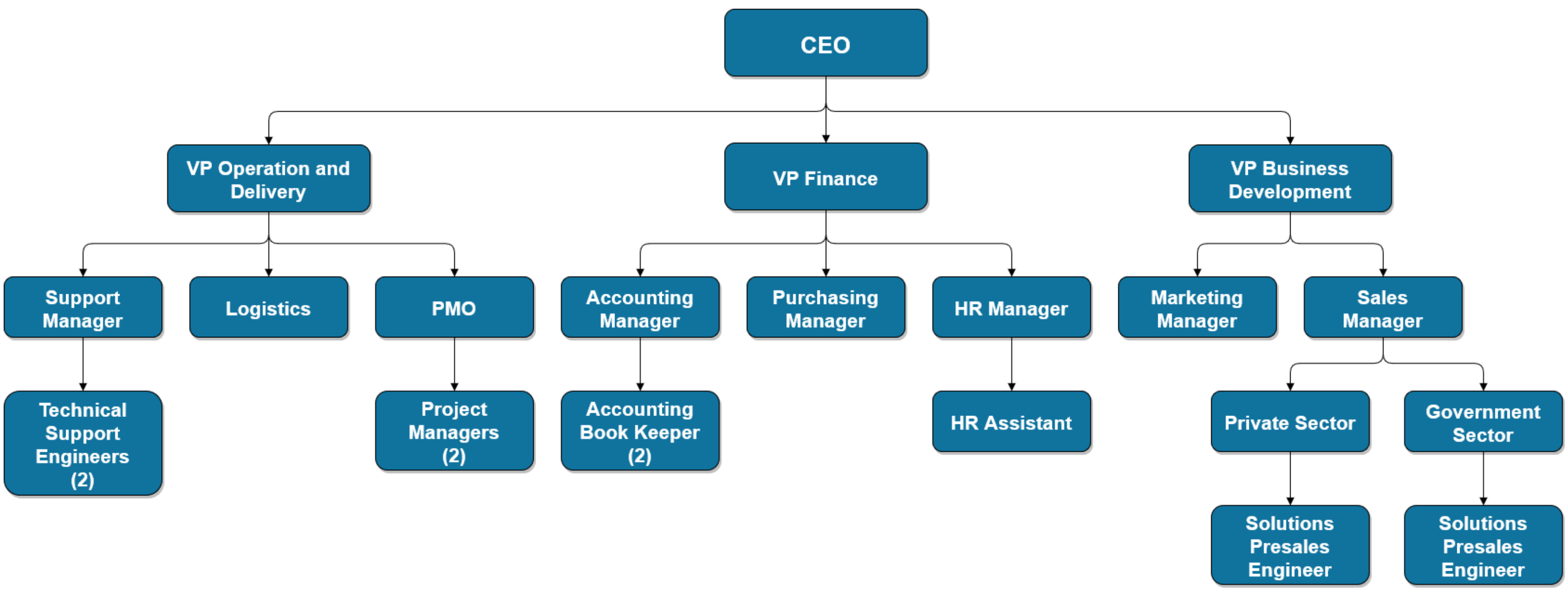
Terminology: Ordered vs Unordered

- Trees can be ordered or unordered.
- In an ordered tree, the children of each node are ordered or have a specific sequence.
  - Example: Organization of a book



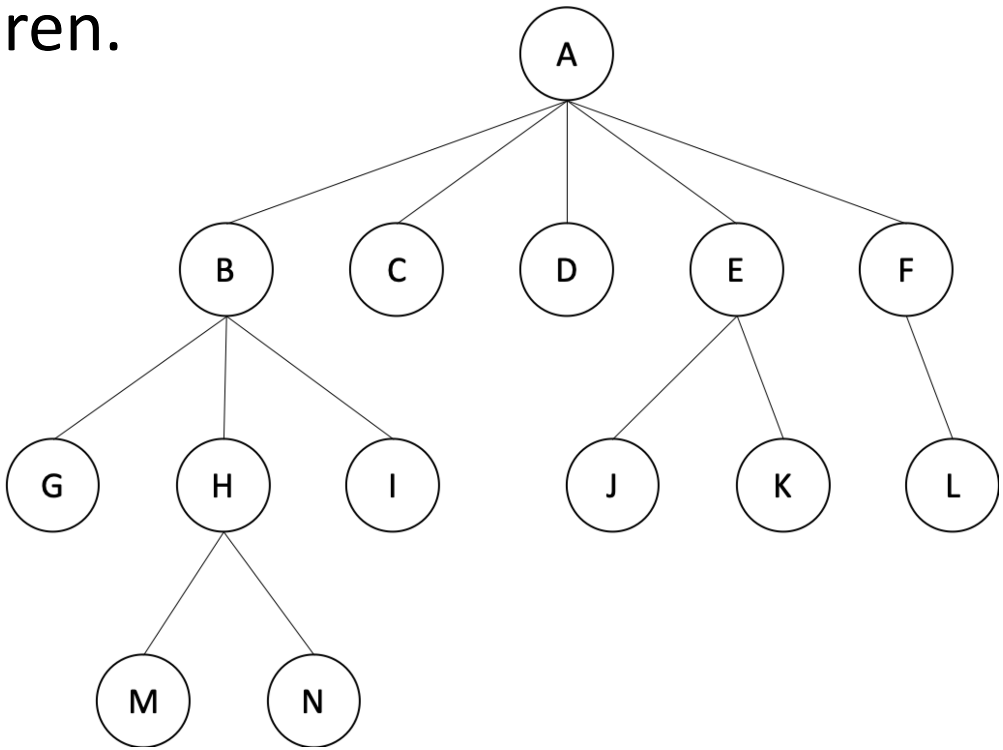
Terminology: Ordered vs Unordered

- In an unordered tree, the children of each node are not ordered or do not have a specific sequence.
  - Example: Organizational chart



Terminology: M-ary tree

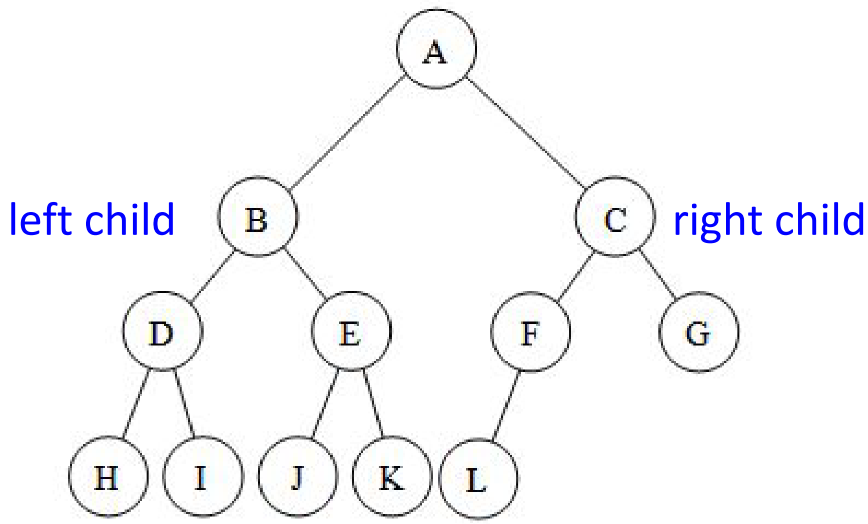
- M-ary tree
  - A tree in which every node has at most M children.



An example of an M-ary tree with $M = 5$

Terminology: Binary Tree

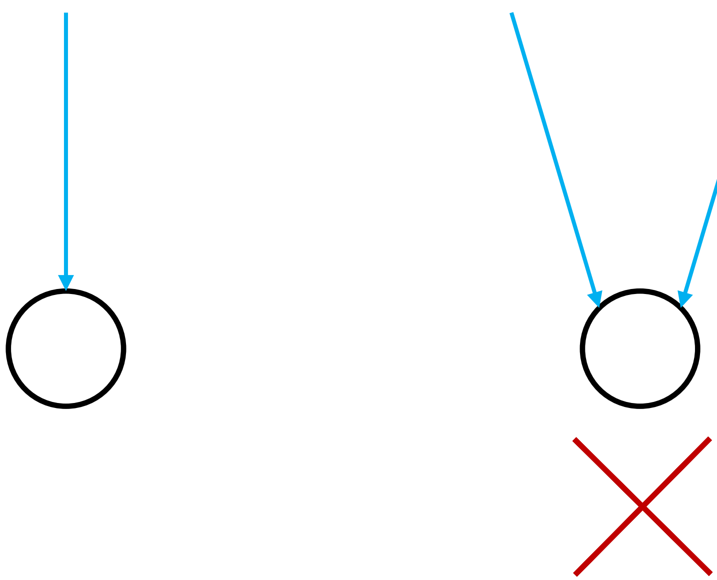
- Binary tree
  - A special case of an M-ary tree with $M = 2$
  - A tree in which every node has at most two children.
  - All internal nodes have at most two children.
  - All external nodes (leaves) have no children.
  - The two children are called the left child and right child.



Basic Properties of Trees

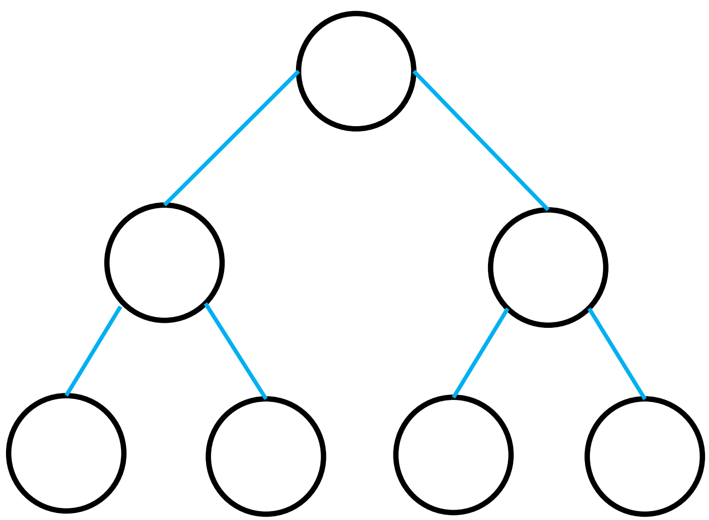
Basic Properties of Trees

- A node has at most one edge leading to it.
  - Each node has exactly one parent, except the root which has no parent.



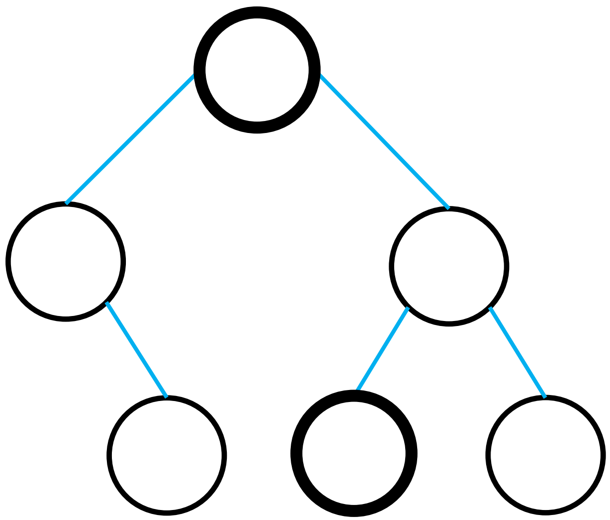
Basic Properties of Trees

- A tree with N nodes has $N - 1$ edges.
  - Every node except the root has an edge to its parent.

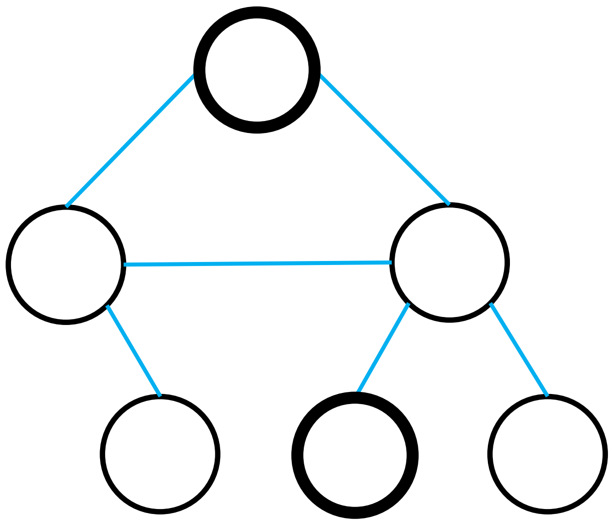


Basic Properties of Trees

- There is exactly one path from the root to each node.
  - Suppose there were 2 paths, then some node along the 2 paths would have 2 parents.



Tree

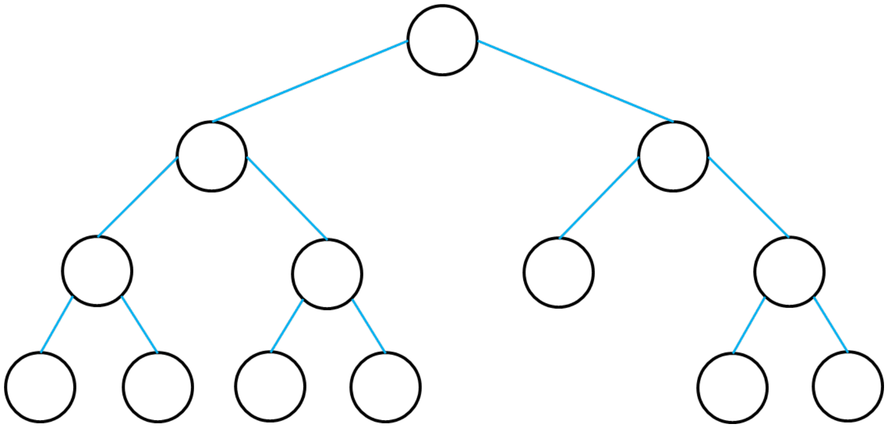


Not a tree

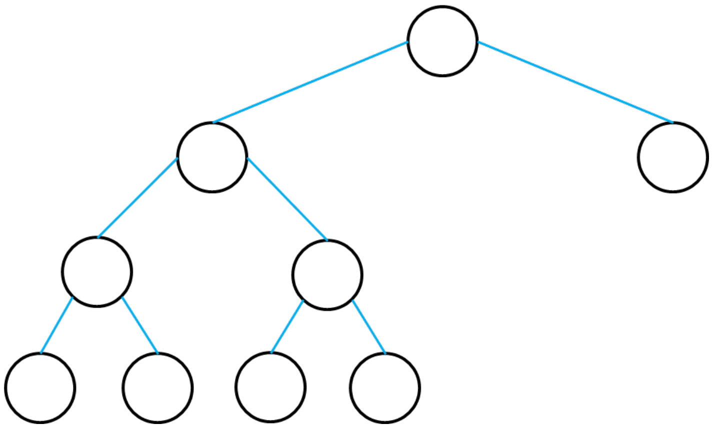
Classification of **Binary** Trees

Balanced Binary Trees

- A **balanced** (or height-balanced) binary tree is a tree in which the height of the left and the right subtree for each node differ by at most 1 (either 0 or 1).



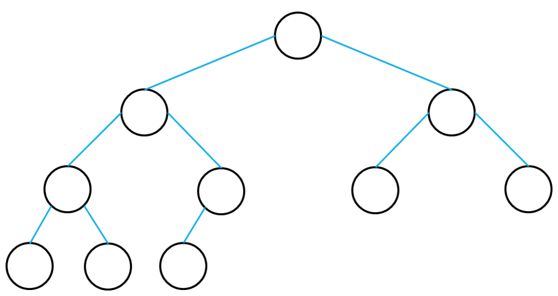
A balanced binary tree



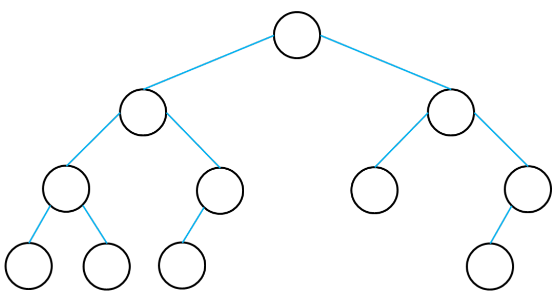
An unbalanced binary tree

Complete Binary Trees

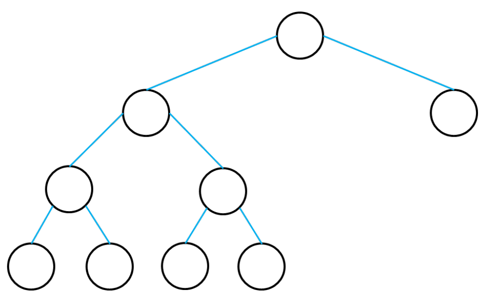
- A **complete** binary tree is a tree where
  - all levels of the tree are filled completely
  - except the deepest level which are filled as far to the left as possible.
- The leaves must be filled from left to right, one level at a time.
- Every complete binary tree is balanced but not the other way around.



A complete binary tree



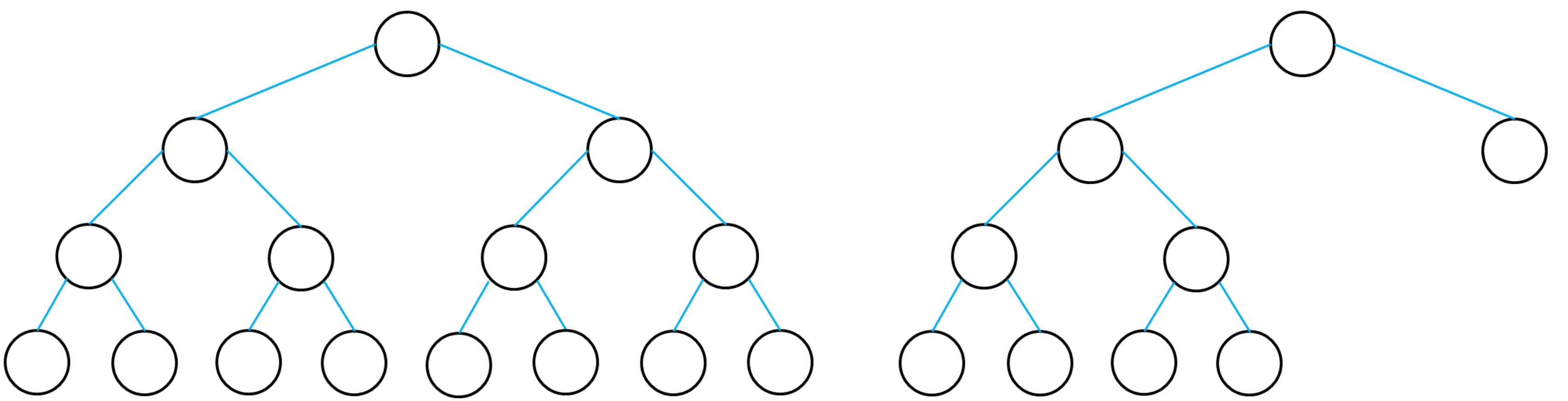
An incomplete binary tree



An incomplete binary tree

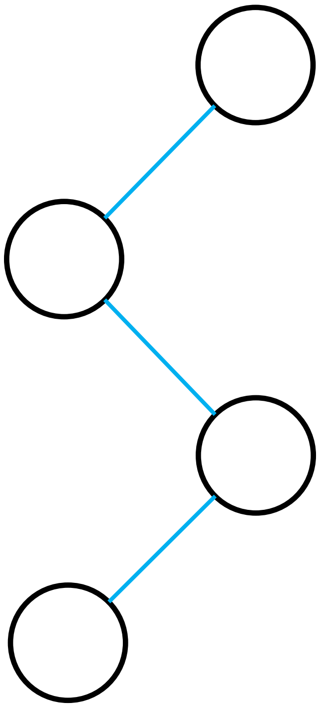
Full Binary Trees

- A **full** binary tree is a tree where every node has either 0 or 2 children.
  - A binary tree where all nodes except leaf nodes have two children.



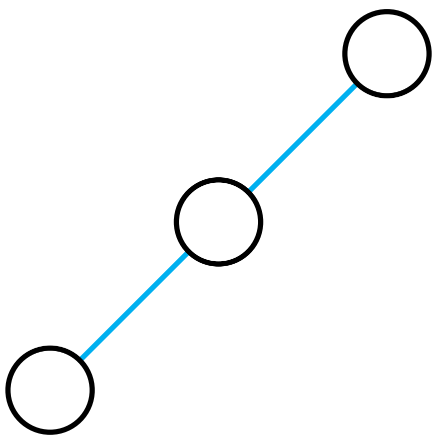
Degenerate Binary Trees

- A **degenerate** binary tree is a tree where every internal node has one child.
- Such a tree is performance-wise the same as a linked list.

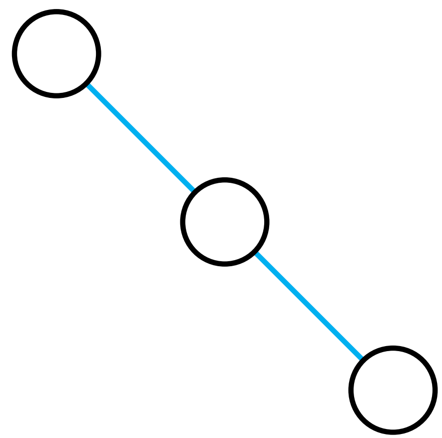


Skewed Binary Tree

- A **skewed** binary tree is a special type of degenerated binary tree where the height of the tree is skewed towards one side.
  - **Left-skewed** binary tree: If all the internal nodes in the degenerate tree have only a left child.
  - **Right-skewed** binary tree: If all the internal nodes in the degenerate tree have only a right child.



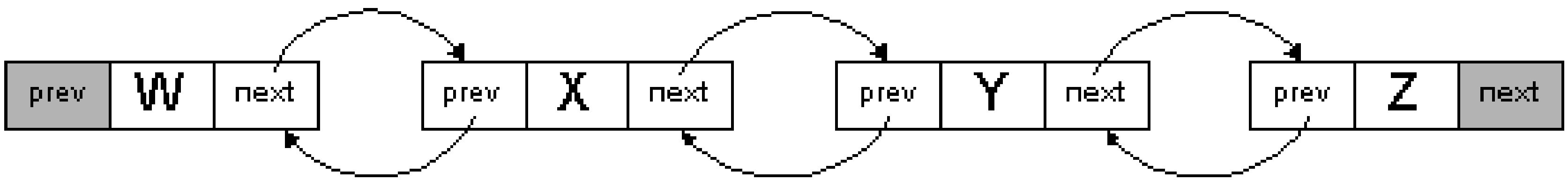
Left-skewed



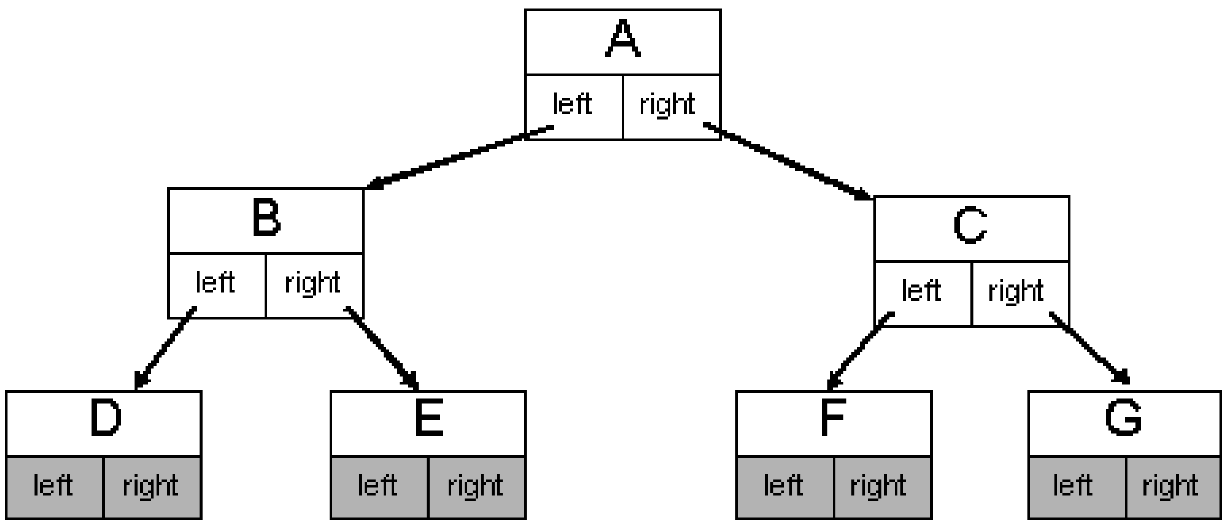
Right-skewed

Implementation of Tree Algorithms

Trees v.s. Linked List



Doubly Linked List



Binary Tree

Trees v.s. Linked List

```
struct ListNode
{
  ListNode *next;
  ListNode *prev;
  int data;
};
```

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

- The two links in a binary tree are not quite the same as the two links in a doubly linked list
  - Trees have left and right link.
  - Lists have previous and next link.
  - Both imply ordering, but a different kind of ordering.

Implementing Tree Algorithms

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

Node *MakeNode(int Data)
{
    Node *node = new Node;
    node->data = Data;
    node->left = 0;
    node->right = 0;
    return node;
}
```

```
void FreeNode(Node *node){
    delete node;
}

typedef Node* Tree;
```

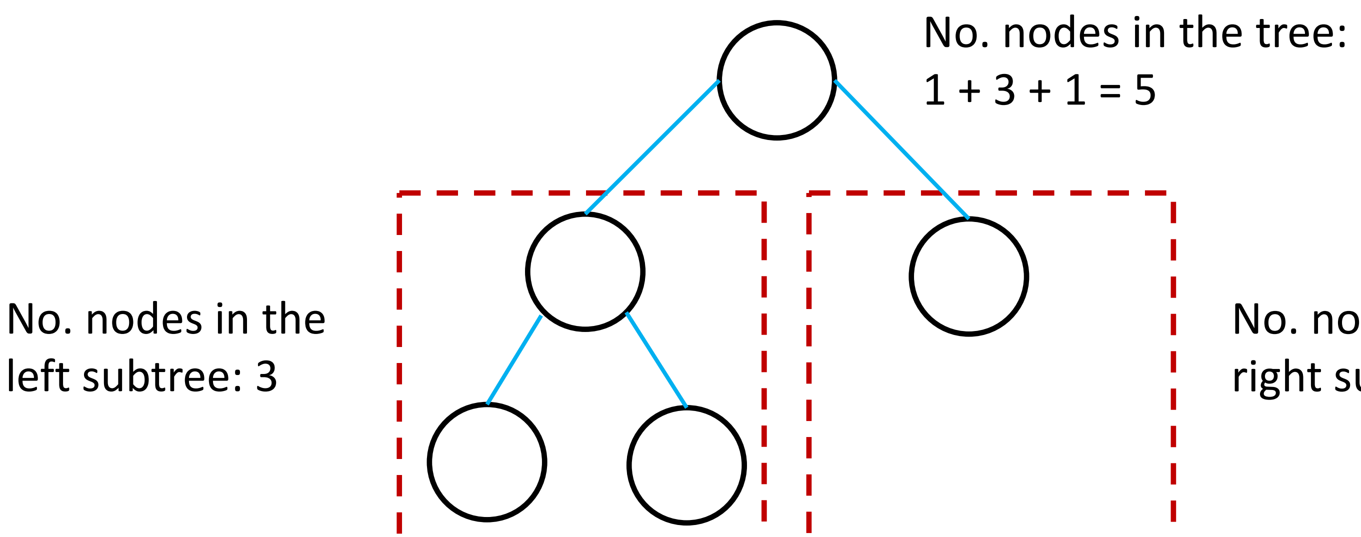
Recursive Algorithms for Binary Trees

- Binary trees are inherently recursive data structures.

- Recursive algorithms are quite appropriate.
  - In some cases, iterative algorithms can be significantly more complicated.

Finding the Number of Nodes in a Tree

- Algorithm for finding the number of nodes in a tree:
  - If the tree is empty: 0
  - If the tree is not empty:
    - 1 + nodes_in_left _subtree + nodes_in_right_subtree

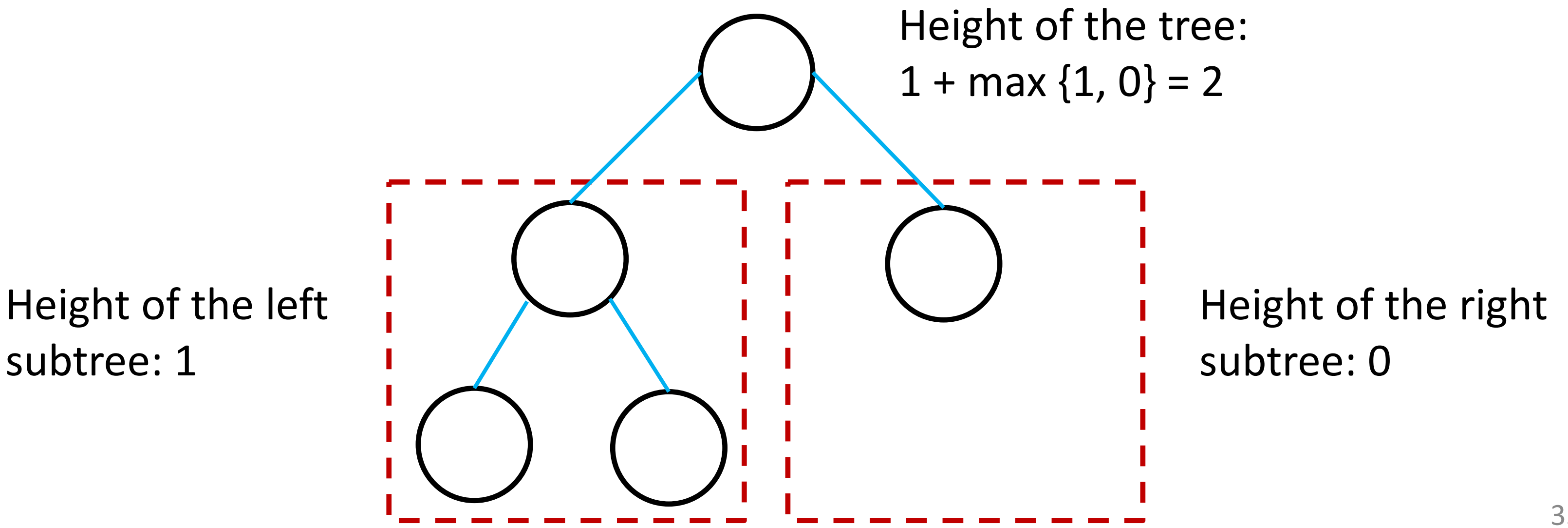


Finding the Number of Nodes in a Tree

```
int NodeCount(Tree tree){
    if (tree == 0)
        return 0;
    else
        return 1 + NodeCount(tree->left) + NodeCount(tree->right);
}
```

Find the Height of a Tree

- Algorithm for finding the height of a tree
  - If the tree is empty: -1
  - If the tree is not empty:
    $1 + \max\{\text{height\_of\_left\_subtree}, \text{height\_of\_right\_subtree}\}$



Find the Height of a Tree

```
int Height(Tree tree){
    if (tree == 0)
        return -1;
    if (Height(tree->left) > Height(tree->right))
        return Height(tree->left) + 1;
    else
        return Height(tree->right) + 1;
}
```

A Better Implementation

```
int Height(Tree tree){
    if (tree == 0)
        return -1;
    int hl=Height(tree->left);
    int hr=Height(tree->right);
    if (hl > hr)
        return hl + 1;
    else
        return hr + 1;
}
```

Traversal of a Binary Tree

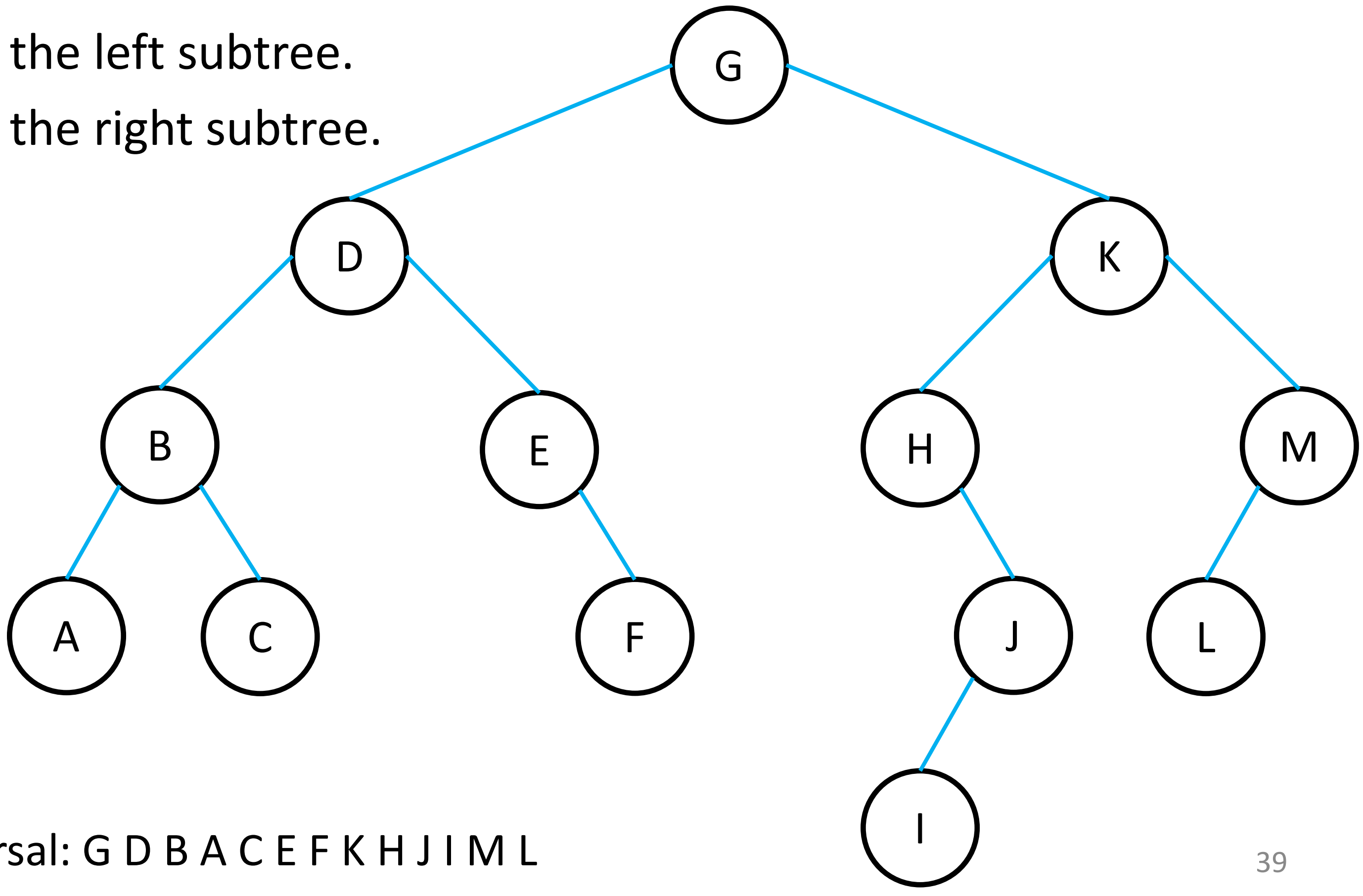
- A traversal of a tree is a systematic way of accessing or visiting all its nodes.
- Tree Traversal algorithms can be classified broadly into two categories:
  - Depth-First Search (DFS) Algorithms
    - Explore as far as possible along each branch before backtracking
  - Breadth-First Search (BFS) Algorithms
    - Visit all the nodes at the current level before moving onto the next level

Traversal of a Binary Tree

- Tree Traversal using Depth-First Search (DFS) can be further classified into three depending on the order in which the node and its left subtree and right subtree are visited.
  - Pre-order traversal
  - In-order traversal
  - Post-order traversal

Pre-order Traversal

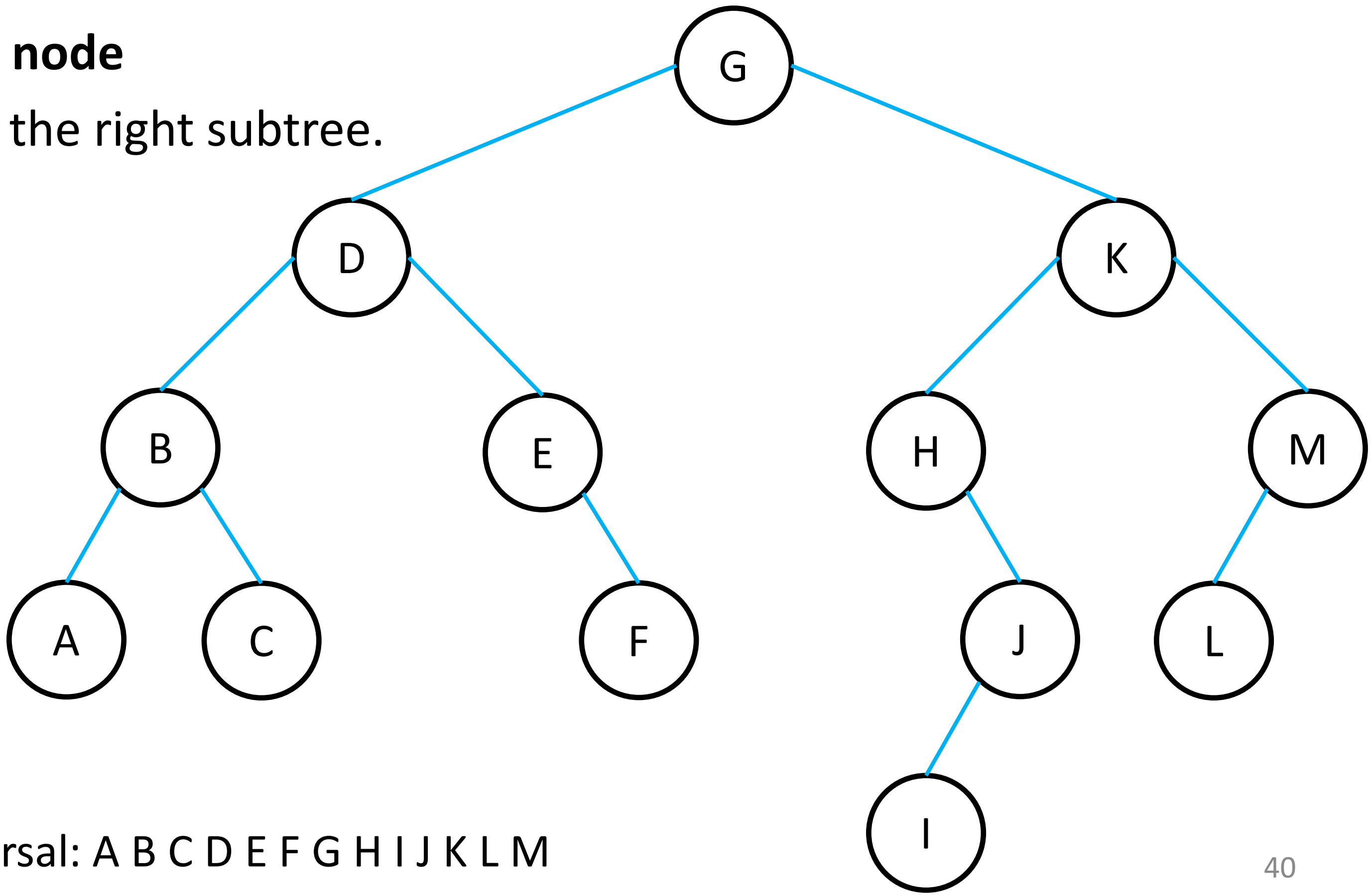
- Pre-order traversal
  1. **Visit the node**
  2. Traverse the left subtree.
  3. Traverse the right subtree.



Pre-order traversal: G D B A C E F K H J I M L

In-order Traversal

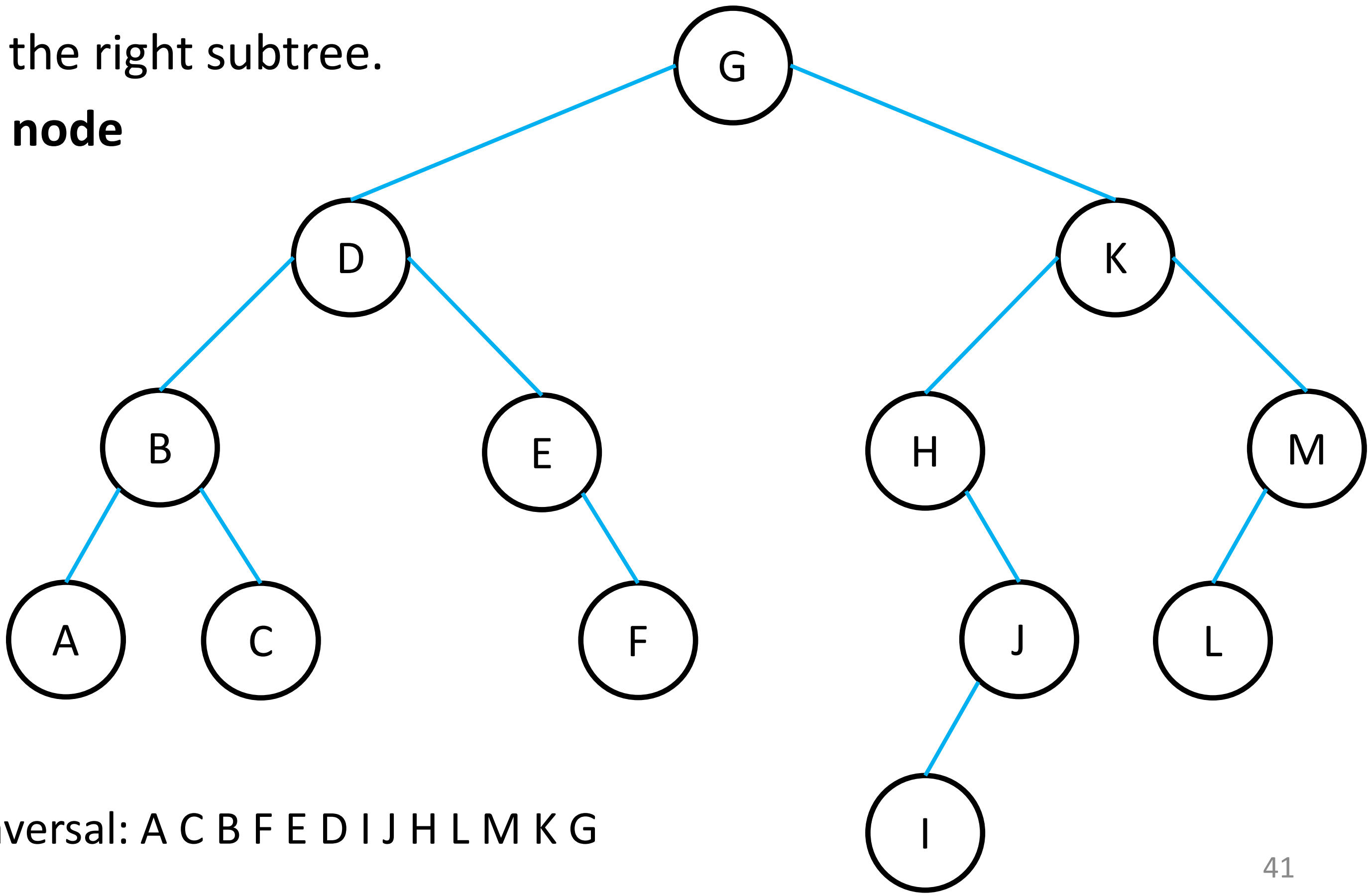
- In-order traversal
  1. Traverse the left subtree.
  2. **Visit the node**
  3. Traverse the right subtree.



In-order traversal: A B C D E F G H I J K L M

Post-order Traversal

- Post-order traversal
  1. Traverse the left subtree.
  2. Traverse the right subtree.
  3. **Visit the node**



Post-order traversal: A C B F E D I J H L M K G

Implement Tree Traversal Algorithms

```
void TraversePreOrder(Tree tree){
  if (tree == 0)
    return;
  else{
    VisitNode(tree);
    TraversePreOrder(tree->left);
    TraversePreOrder(tree->right);
  }
}
```

```
void TraversePostOrder(Tree tree){
  if (tree == 0)
    return;
  else{
    TraversePostOrder(tree->left);
    TraversePostOrder(tree->right);
    VisitNode(tree);
  }
}
```

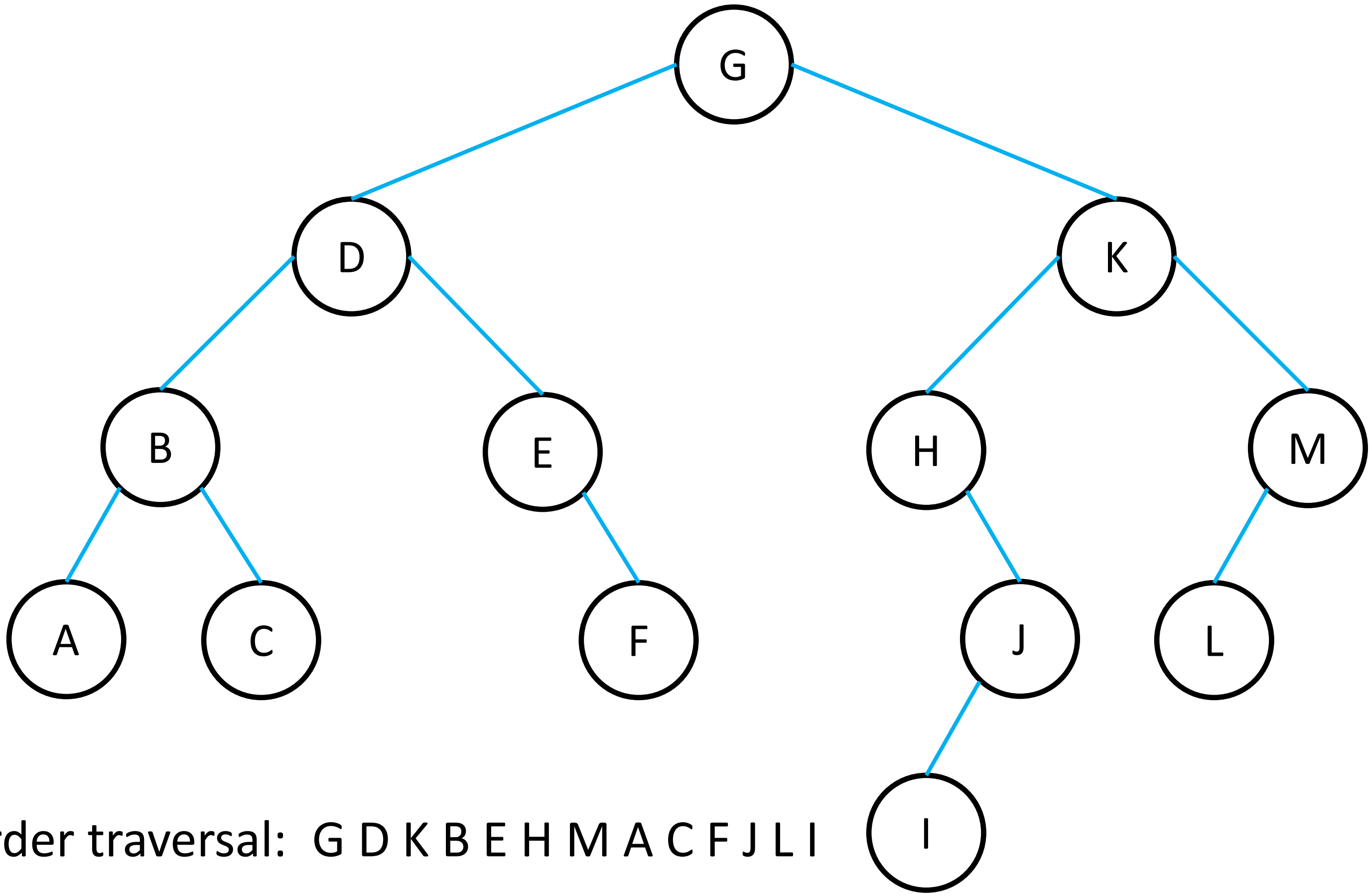
```
void TraverseInOrder(Tree tree){
  if (tree == 0)
    return;
  else{
    TraverseInOrder(tree->left);
    VisitNode(tree);
    TraverseInOrder(tree->right);
  }
}
```

Traversal of a Binary Tree

- Tree traversal algorithm using Breadth-First Search (BFS) is also known as
  - Level-order traversal

Level-Order Traversal

- Traversing all nodes on level 0 from left to right
- Then all nodes on level 1 (left to right);
- Then nodes on level 2 (left to right), etc…



Level-order traversal: G D K B E H M A C F J L I

Level-Order Traversal using a Queue

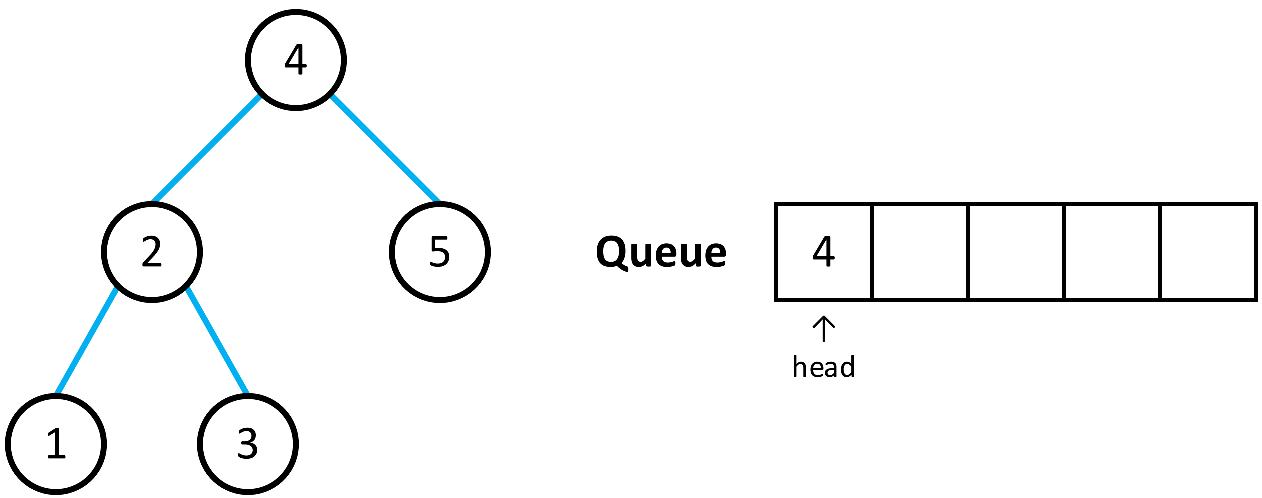
- Use a queue to keep track of the nodes to visit.
- Start with the root node and push it in to the queue.
- When the queue is not empty, repeat
  – Pop the front node;
  – Visit it.
  – Push its left and right children (if any)

Level-Order Traversal using a Queue

```
void TraverseLevelOrderQueue(Tree tree) {
    if (tree == 0) {
        return;
    }

    queue<Tree> q;
    q.push(tree);

    while (!q.empty()) {
        Tree current = q.front();
        q.pop();
        VisitNode(current);

        // Enqueue the left and right children (if any)
        if (current->left) {
            q.push(current->left);
        }
        if (current->right) {
            q.push(current->right);
        }
    }
}
```

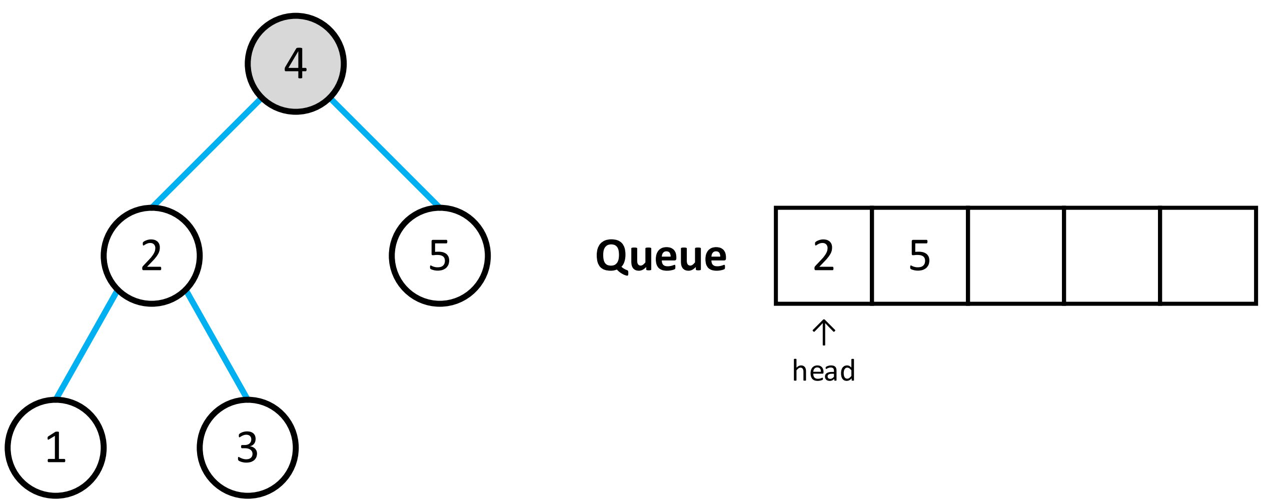
Example: Level-Order Traversal using a Queue

- Step 1: Push the root node (node 4) into the queue.



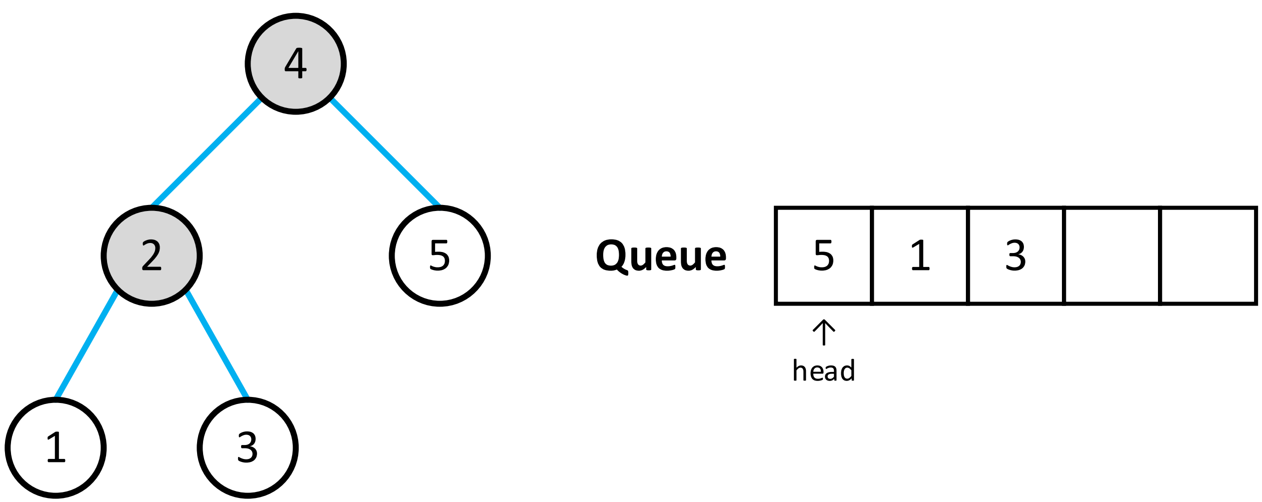
Example:
Level-Order Traversal using a Queue

- Step 2:
  - Pop the front node (node 4) from the queue.
  - Visit the node.
  - Push its children (nodes 2 and 5) to the queue.



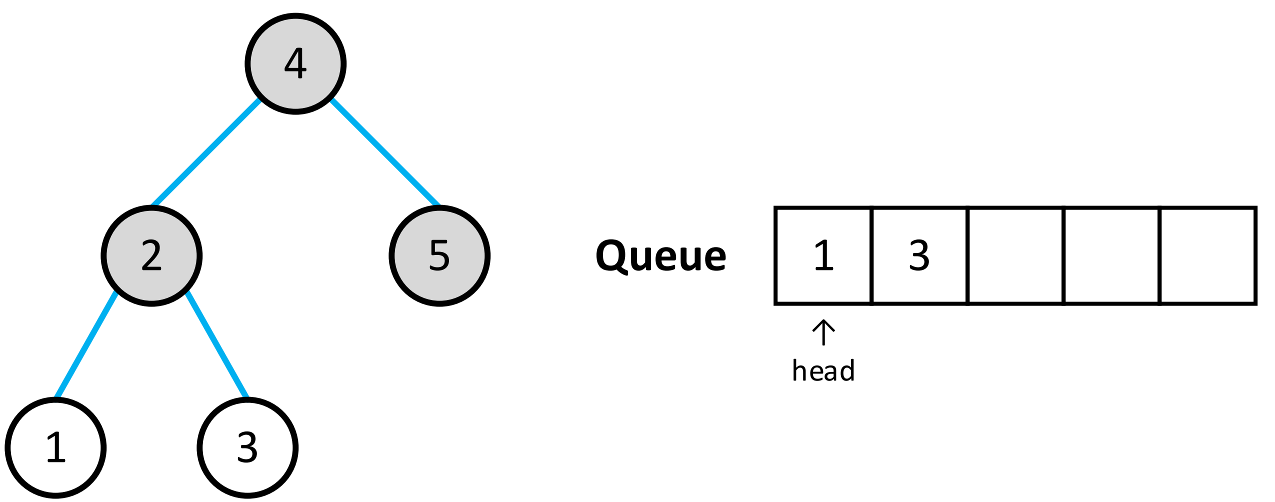
Example: Level-Order Traversal using a Queue

- Step 2:
  - Pop the front node (node 2) from the queue.
  - Visit the node.
  - Push its children (nodes 1 and 3) to the queue.



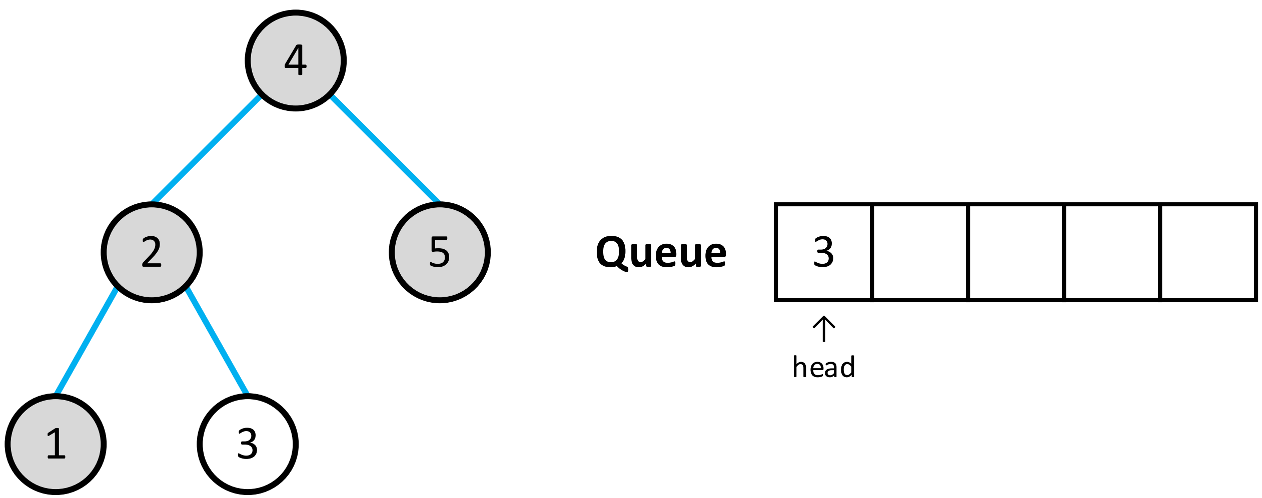
Example:
Level-Order Traversal using a Queue

- Step 3:
  - Pop the front node (node 5) from the queue.
  - Visit the node.
  - Push its children (none) to the queue.



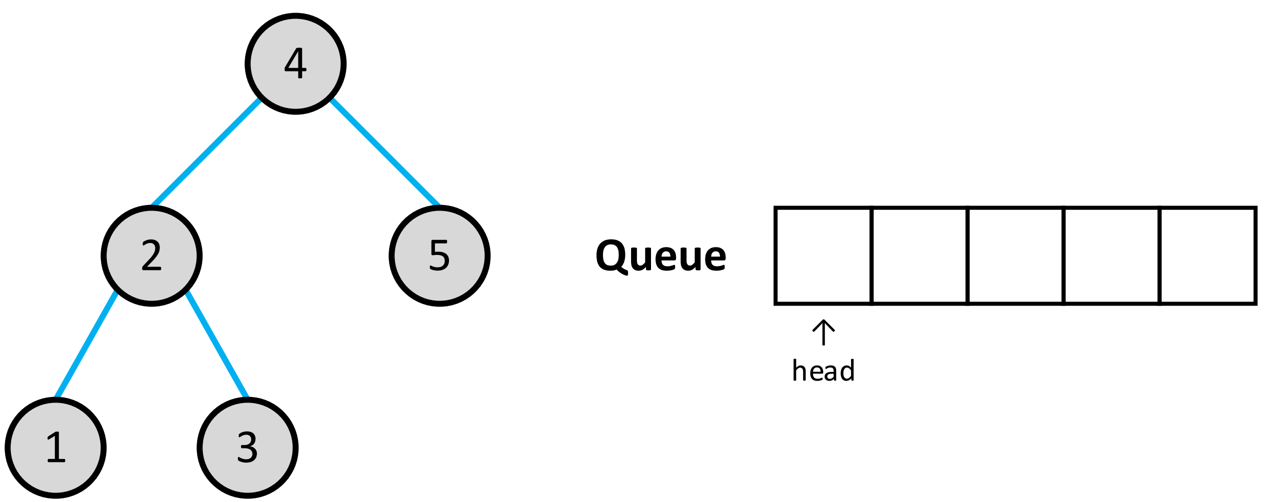
Example:
Level-Order Traversal using a Queue

- Step 4:
  - Pop the front node (node 1) from the queue.
  - Visit the node.
  - Push its children (none) to the queue.



Example:
Level-Order Traversal using a Queue

- Step 5:
  - Pop the front node (node 3) from the queue.
  - Visit the node.
  - Push its children (none) to the queue.



Summary

- Terminology of Trees
- Basic Properties of Trees
- Classification of Binary Trees
- Implementation of Tree Algorithms
  - Number of nodes in a tree
  - Height of a tree
  - Traversal of a binary tree