QUESTION 1

A **full binary tree** is a tree where:

* Every node has either **0** (leaf node) or **2** children.
* The number of nodes at each level increases exponentially.

The **height** h of the tree is the longest path from the root to a leaf.

In a binary tree:

* **Level 0** (Root): 20 = 1 node
* **Level 1**: 21 = 2 nodes
* **Level 2**: 22 = 4 nodes
* **Level 3**: 23 = 8 nodes
* **Level h**: 2h nodes

Observing the pattern, the number of nodes at level i is: 2*i*

where *i* is the level index (starting from 0 for the root).

To find the total number of nodes in the tree, sum the nodes at all levels from **level 0 to level h**:

N = 20 + 21 + 22 + … + 2h

This is a **geometric series** with:

* First term a = 1 (the root)
* Common ratio r = 2
* Number of terms = h+1 (because we start from level 0)

The sum of a geometric series is given by:

S = a(rn−1)/(r−1)

Substituting a = 1, r = 2, and n = h + 1:

N=2h+1−1/(2−1)

Since 2 – 1 = 1, this simplifies to:

N = 2h+1 − 1

* This formula gives the **maximum** number of nodes in a binary tree of height h.
* It applies only to a **full** binary tree, where each node has **two** children until the last level.

**Example 1: Height h=2**

O (Level 0: 1 node)

/ \

O O (Level 1: 2 nodes)

/ \ / \

O O O O (Level 2: 4 nodes)

Total nodes: 1 + 2 + 4 = 7, which matches:

N = 22 + 1 – 1 = 23 – 1 = 7

**Example 2: Height h=3h = 3h=3**

CopyEdit

O (Level 0: 1 node)

/ \

O O (Level 1: 2 nodes)

/ \ / \

O O O O (Level 2: 4 nodes)

/ \ / \ / \ / \

O O O O O O O O (Level 3: 8 nodes)

Total nodes: 1 + 2 + 4 + 8 = 15, which matches:

N = 23 + 1 − 1 = 24 − 1 = 15

**Final Formula**

N = 2h+1− 1

where:

* h is the height of the binary tree.
* N is the **maximum** number of nodes.

**Question 2**

**(i) Depth First Search (DFS)**

DFS is a tree or graph traversal algorithm that explores as far as possible along each branch before backtracking.

**Example of DFS on a Binary Tree**

Consider the following binary tree:

A

/ \

B C

/ \ \

D E F

**DFS Traversal Types**

There are three common DFS traversal methods:

1. **Preorder (Root → Left → Right)**
   * Start from the root: A
   * Go left: B
   * Go left again: D
   * Backtrack and go right: E
   * Backtrack to root and go right: C
   * Go right again: F
   * **Preorder Output:** A → B → D → E → C → F
2. **Inorder (Left → Root → Right)**
   * Start from the leftmost node: D
   * Go to parent: B
   * Visit right child: E
   * Move to root: A
   * Visit right child: C
   * Visit rightmost child: F
   * **Inorder Output:** D → B → E → A → C → F
3. **Postorder (Left → Right → Root)**
   * Start from left: D
   * Go to right sibling: E
   * Visit parent: B
   * Move to right subtree: F
   * Visit parent: C
   * Finally, visit root: A
   * **Postorder Output:** D → E → B → F → C → A

**(ii) Breadth First Search (BFS) (Level Order Traversal)**

BFS explores all nodes at the present depth level before moving to nodes at the next depth level.

**Comparison of DFS and BFS**:

* **DFS** goes deep along a path before backtracking.
* **BFS** explores all neighbors before moving deeper.

**(iii) AVL Tree**

An **AVL tree** is a **self-balancing binary search tree (BST)**, where the height difference (balance factor) between the left and right subtrees of any node is at most **1**.

**Example: Insertion in an AVL Tree**

Consider inserting the following numbers: **10, 20, 30**

1. **Insert 10** → Tree remains balanced.

10

1. **Insert 20** → Still balanced.

10

\

20

1. **Insert 30** → Now the tree is **unbalanced** (Right-Right Case).

10

\

20

\

30

* + Balance factor of 10 is -2 (height difference is greater than 1).
  + To fix it, **Left Rotation** is applied at 10.

**After Left Rotation**

20

/ \

10 30

Now the AVL tree is **balanced**.

**Properties of AVL Tree**

* **Search Complexity:** O(log n)
* **Insertion Complexity:** O(log n) (because of balancing)
* **Balancing Rotations:** **Left Rotation, Right Rotation, Left-Right Rotation, Right-Left Rotation**

**QUESTION 3**

A **graph** is a collection of **nodes (vertices)** connected by **edges**. A tree is a special type of graph that satisfies the following conditions:

* It is **connected** (every node is reachable from the root).
* It has **no cycles** (there are no loops or repeated paths).
* If there are **N nodes**, there must be exactly **N - 1 edges**.

**Key Difference:** While all **trees** are graphs, not all **graphs** are trees.

**2. Properties of a Tree as a Graph**

A tree is an **acyclic connected graph** with the following key properties:

**i) Rooted Tree**

* A **rooted tree** is a tree where one node is designated as the **root**.
* The root has no parent, and all other nodes are **descendants** of the root.

**Example:**

A (Root)

/ \

B C

/ \ \

D E F

* A is the **root**.
* B and C are **children** of A.
* D, E, and F are **descendants**.

**ii) Parent-Child Relationship**

* In a tree, every node (except the root) has a **parent**.
* Nodes directly connected to a parent are called **children**.

**Example:**

* B is the **child** of A.
* A is the **parent** of B and C.

**iii) Leaf Nodes**

* A **leaf node** is a node that has no children.
* In the above example, D, E, and F are **leaf nodes**.

**iv) Height of a Tree**

* The **height** of a tree is the **longest path from the root to a leaf**.
* If the longest path has **h edges**, the height of the tree is **h**.

**v) Depth of a Node**

* The **depth** of a node is the number of edges from the root to that node.
* Example:
  + A has depth **0**.
  + B has depth **1**.
  + D has depth **2**.

**Types of Trees in Graph Theory**

**i) General Tree**

A tree where each node can have any number of children.

A

/ | \

B C D

/| |\

E F G H

**ii) Binary Tree**

A tree where each node has **at most two** children.

A

/ \

B C

/ \ \

D E F

**iii) Binary Search Tree (BST)**

A **binary tree** where:

* Left child <Parent < Right child.

**Example:**

10

/ \

5 15

/ \ \

2 7 20

**iv) AVL Tree (Self-Balancing Tree)**

* A **BST** that maintains **balance** (height difference between left and right subtrees is at most 1).
* Uses **rotations** to stay balanced.

**v) Spanning Tree**

* A **spanning tree** is a subset of a **graph** that includes all vertices with **N-1 edges** (no cycles).
* Used in **network design (e.g., Minimum Spanning Tree - MST)**.

**4. Tree Traversal as a Graph**

Since a tree is a graph, we can traverse it using graph traversal techniques:

**Depth First Search (DFS)**

* Explores as deep as possible before backtracking.
* Uses **stack (recursion)**.
* Types: **Preorder, Inorder, Postorder**.

**ii) Breadth First Search (BFS)**

* Explores all nodes level by level.
* Uses **queue**.
* Also known as **Level Order Traversal**.

**5. Applications of Trees in Graph Theory**

* **Data Structures:** BST, AVL Trees, Heap.
* **Artificial Intelligence:** Decision Trees.
* **Computer Networks:** Routing Trees.
* **Database Indexing:** B-Trees used in databases.
* **File Systems:** Directory structures.