Program to Find Maximum Depth or Height of a Binary Tree

Mr. Vikas Kumar
Assistant Professor
Industry Embedded Program

Lesson Plan

Subject/Course	Competitive Coding
Lesson Title	Program to Find Maximum Depth or Height of a Binary Tree
Lesson Objectives	
Understand the concept of depth and height in a binary tree.	
Learn how to calculate the maximum depth using recursion.	
Explore how height relates to levels in a binary tree.	
Analyze time and space complexity of the height algorithm.	

Problem Statement:

Write a program for implementing a MINSTACK which should support operations like push, pop, overflow, underflow, display.

1. Construct a stack of N-capacity
2. Push elements
3. Pop elements
4. Top element
5. Retrieve the min element from the stack

Concept

What is Depth (Height) of a Tree?

- The **depth** or **height** of a binary tree is the **number of nodes** along the **longest path** from the root node down to the farthest leaf node.
- Example:
    - Depth of a single node tree = 1
    - Depth of an empty tree = 0

- **Key Idea (Recursive Definition):**

maxDepth(node) = 1 + max(maxDepth(left subtree), maxDepth(right subtree))

Algorithm/Logic

1. If root is null → return 0.
2. Recursively find height of left subtree.
3. Recursively find height of right subtree.
4. Return 1 + maximum of left and right heights.
5. Base condition handles empty trees automatically.

Algorithm/Logic

Example Steps:

1. Move recursively to leftmost and rightmost nodes.

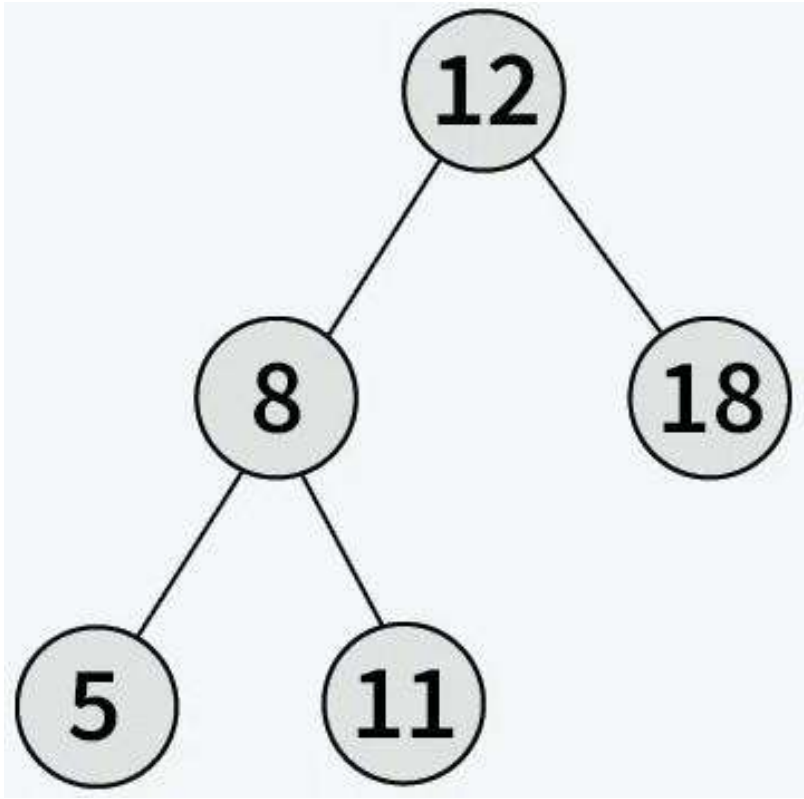
2. Compute heights bottom-up.

3. Add 1 for each parent level on return.

4. Final result gives total height of the tree.

Visualization

Example Tree:



Depth Calculation:

Depth(5) = 1

Depth(11) = 1

Depth(8) = 1 + max(1, 1) = 2

Depth(12) = 1 + max(1, 2) = 3

Code Implementation

```
class Node {
    int data;
    Node left, right;

    Node(int val) {
        data = val;
        left = right = null;
    }
}
class BinaryTree {
    Node root;

    int height(Node node) {
        if (node == null)
            return 0;

        int leftHeight = height(node.left);
        int rightHeight = height(node.right);

        return 1 + Math.max(leftHeight, rightHeight);
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        BinaryTree tree = new BinaryTree();

        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);

        int result = tree.height(tree.root);
        System.out.println("Height of the tree: " + result);
    }
}
```

Output

```
Height of the tree: 3
```

Time & Space Complexity

Time Complexity:

O(n) – each node is visited once.

Space Complexity:

O(h) – due to recursion stack (h = height of tree).

Summary

1. Height is the longest path from root to leaf.
2. Recursive approach computes height efficiently.
3. Time Complexity: O(n), Space: O(h).
4. Used in balance checking, diameter, and level-order traversal.

Practice Questions:

1. Maximum Depth of Binary Tree — LeetCode #104

🔗https://leetcode.com/problems/maximum-depth-of-binary-tree/

Concept: Find height using recursive traversal.

Why Practice: Core concept for many tree problems.

Practice Questions:

2. Minimum Depth of Binary Tree — LeetCode #111

🔗 https://leetcode.com/problems/minimum-depth-of-binary-tree/

Concept: Find shortest path from root to a leaf.

Why Practice: Complements maximum depth understanding.

Practice Questions:

3. Diameter of Binary Tree — LeetCode #543

🔗https://leetcode.com/problems/diameter-of-binary-tree/

Concept: Compute the longest path between any two nodes in the tree using depth logic.

Why Practice: Extends the max depth concept — shows how tree depth contributes to diameter calculation.

Thanks