

Diameter of Binary Tree – Complete Documentation


Table of Contents

1. Problem Statement
2. Intuition
3. Key Observations
4. Approach
5. Edge Cases
6. Complexity Analysis
 - Time Complexity
 - Space Complexity
7. Alternative Approaches
8. Test Cases
9. Final Thoughts

1. Problem Statement

Given the root of a binary tree, return the length of the diameter of the tree.


- The diameter is defined as the number of edges on the longest path between any two nodes in the tree.
- The path may or may not pass through the root.

 Example 1:

Input: root = [1,2,3,4,5]

Output: 3

Explanation: Longest path is [4,2,1,3] or [5,2,1,3]

 Example 2:

Input: root = [1,2]

Output: 1

2. 💡 Intuition

To find the diameter:

- We need to know the longest path between any two nodes.
- This longest path can be formed by combining the left and right subtree heights at any node.
- So, we need to calculate the height of each node and track the maximum sum of left and right heights across all nodes.

3. 🔑 Key Observations

- The diameter may or may not pass through the root.
- At any node, the longest path going through it = height of left subtree + height of right subtree.
- So, the maximum of all such (left + right) sums across all nodes gives the diameter.

4. □ Approach

- Use a recursive DFS (Depth-First Search) to calculate the height of each node.
- At every node:
 - Recursively get the left and right heights.
 - Update the global max_diameter as $\max(\text{max_diameter}, \text{left} + \text{right})$
- Return the final max_diameter.

5. ⚠ Edge Cases

- Tree with only one node \rightarrow Diameter = 0
- Tree with two nodes \rightarrow Diameter = 1
- Left or right skewed tree (like a linked list) \rightarrow Diameter = number of nodes - 1

6. Complexity Analysis

Time Complexity:

- $O(N)$ — Every node is visited once, where N is the number of nodes.

Space Complexity:

- $O(H)$ — Due to recursion stack, where H is the height of the tree. In the worst case (skewed tree), $H = N$.

7. Alternative Approaches

- Iterative DFS with stack: Can simulate the traversal using stack, but becomes complex for height tracking.
- Two-pass traversal: First find the farthest node from root, then find farthest node from that — but requires extra traversal.
- Using BFS: Possible with additional tracking of depth and parent pointers, but not space efficient.

DFS recursion is optimal here in both performance and readability.

8. Test Cases

Test Case 1:

Input: $[1, 2, 3, 4, 5]$

Output: 3

Explanation: Path $[4, 2, 1, 3]$

Test Case 2:

Input: $[1]$

Output: 0

Explanation: Only one node \rightarrow no edges

Test Case 3:

Input: [1,2]

Output: 1

Explanation: Path [2,1]

Test Case 4 (Skewed Tree):

Input: [1,null,2,null,3,null,4]

Output: 3

9. 🚀 Final Thoughts

- This is a classic tree problem that uses post-order traversal to solve.
- The key is understanding how height relates to the diameter.
- This solution is both efficient and elegant using recursion.
- Great practice for problems involving global tracking during recursion.