Solution: Diameter of Binary Tree

Let's solve the Diameter of Binary Tree problem using the Tree Depth-first Search pattern.

We'll cover the following

Statement

Given a binary tree, you need to compute the length of the tree's diameter. The diameter of a binary tree is the length of the longest path between any two nodes in a tree. This path may or may not pass through the root.

> **Note:** The length of the path between two nodes is represented by the number of edges between them.

Constraints:

- The number of nodes in the tree is in the range $[1, 10^4]$.
- $-100 \leq$ `Node.value` ≤ 100

Solution

So far, you've probably brainstormed some approaches and have an idea of how to solve this problem. Let's explore some of these approaches and figure out which one to follow based on considerations such as time complexity and any implementation constraints.

Naive approach

The naive approach will be to pick one node, find the distance to every other node from the selected node, and keep track of the maximum value. Repeat this process with all the nodes of the tree. After this process, we'll have the maximum possible distance between the two nodes. That will be the diameter of the tree.

The time complexity for the naive approach will be $O(n^2)$. As the tree grows, the time complexity increases.

Optimized approach using depth-first search

We need to calculate the height of the left and right subtrees and return the greater one to calculate the diameter. To calculate the height, we traverse the depth of the left and right subtrees. Therefore, the problem is a natural fit for the tree depth-first search pattern.

As seen in the example below, the diameter of a binary tree is the length of the longest path between any two nodes in a tree. This path may or may not pass through the root. Let's explore both of these conditions.

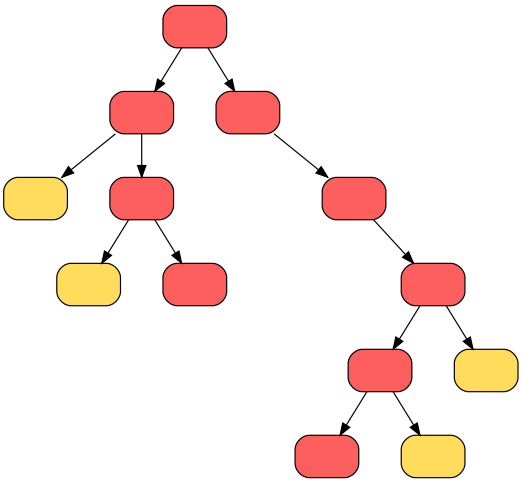
Case 1: The longest path that passes through the root of the tree

We can use the height of the tree when the root node is included in the longest path. The height of a tree is the longest path from the root node to any leaf node in the tree. So, the height of a tree can be considered the longest path starting from the root. We can calculate the longest path of the complete tree as:

height(root -> left) + height(root -> right) + 1(root node itself)

The path starts from the deepest node in the left subtree (adding **height(root -> left)** to the diameter), passes through the root (adding **1** to the diameter), and ends at the deepest node in the right subtree (adding **height(root -> right)** to the diameter).

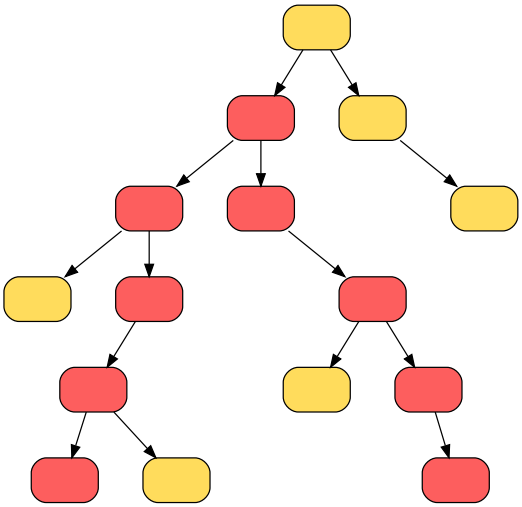
Let's look at an example of this case in the illustration below:



The number of edges in the red part is the diameter of the tree

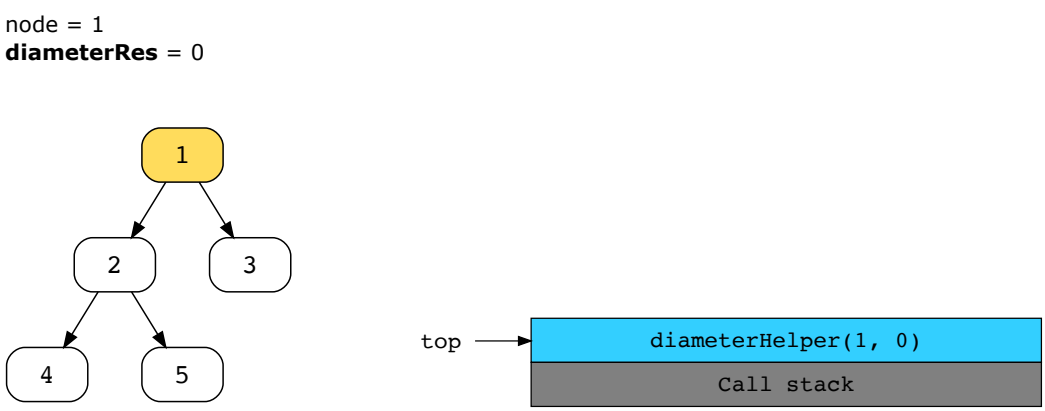
Case 2: The longest path that doesn't pass through the root of the tree

In this case, the longest path can exist in either the left or right subtree. Since the root will not be a part of our path, we can consider the two subtrees to be new individual trees. The longest path for each of these trees needs to be calculated separately, and the longest one will be chosen.



The number of edges in the red part is the diameter of the tree

We'll calculate the diameter of the tree by using the `diameterHelper()` function. We need to calculate the left and right height of each node and update `diameterRes = max(diameterRes, lh + rh)`. We'll calculate this metric for each node and update the `diameterRes`. After traversing the whole tree, `diameterRes` will have the diameter of the tree.



Let's look at the code for this solution below:

> **Note**: The tree is provided as input in the form of a list containing the level order traversal

Java

main.java

BinaryTree.java

TreeNode.java

```
class TreeDiameter {
  public static List<Integer> diameterHelper(TreeNode<Integer> node, int dian
    List<Integer> result = new ArrayList<>();
    if (node == null) {
      result.add(0);
      result.add(diameterRes);
      return result;
    }
    else {
      // Compute the height of each subtree
      List<Integer> lh = diameterHelper(node.left, diameterRes);
      diameterRes = lh.get(1);
      List<Integer> rh = diameterHelper(node.right, diameterRes);
      diameterRes = rh.get(1);
      // update the result with the max of the previous and current diameter
      // value
      diameterRes = Math.max(diameterRes, lh.get(0) + rh.get(0));

      // Use the larger one
      result.add(Math.max(lh.get(0), rh.get(0)) + 1);
      result.add(diameterRes);
      return result;
    }
  }

  public static int diameterOfBinaryTree(TreeNode<Integer> root) {
```

Solution summary

To recap, the solution to this problem can be divided into the following parts:

- Start with the assumption that the diameter is 0.
- Calculate the diameter of the left sub-tree and right sub-tree of the root node using the following recursive process:
  - At a leaf node, the diameter and height with respect to its children is 0 and 1, respectively.
  - For a non-leaf node, calculate the heights as well as the diameters of the left and right sub-trees. If the diameter passes through this node, then the diameter is the sum of the heights of the two sub-trees. Otherwise, it is the greater of the diameters of the two sub-trees.
- Update the diameter as the greater of two values:
  - the sum of the heights of the left and right sub-trees,
  - the greater of the diameters of the two sub-trees.

Time complexity

The time complexity will be $O(n)$ because each of the tree's nodes gets visited once, where n is the number of nodes in the tree.

Space complexity

The space complexity will be $O(n)$ because the recursive stack can grow up to $O(n)$, where n is the number of nodes in the tree.