Project Description: Binary Tree Operations

Overview

In this project, you are provided with starter code for a Binary Tree data structure. Your task is to complete the implementation of three key functions that operate on the binary tree:

- **countInternal:** Counts the number of internal (non-leaf) nodes.
- **height:** Determines the height of the tree.
- **isPerfect:** Checks whether the tree is perfect.

The starter code includes the tree representation (nodes with data, left, and right references) and several test cases to validate your implementations. While the provided code is in one specific programming language, your solution should focus on the underlying algorithmic logic so that it can be implemented in any language.

Objectives

- **Understand Tree Structures:** Learn about binary trees and how nodes are connected.
- **Practice Recursion:** Use recursion to traverse and analyze the tree.
- **Algorithm Development:** Develop efficient algorithms for counting nodes, determining tree height, and checking tree perfection.
- **Edge Case Handling:** Consider special cases such as empty trees or trees with only one node.

Functional Requirements

1. **countInternal**
    - **Definition:** An internal node is any node that is not a leaf (i.e., it has at least one child).
    - **Goal:** Write a function that recursively counts and returns the number of internal nodes in the tree.
    - **Edge Case:** If the tree is empty or if a node is a leaf, it should contribute 0 to the count.
2. **height**
    - **Definition:** The height of a tree is defined as the number of nodes on the longest path from the root to a leaf.
    - **Goal:** Write a function that computes and returns the height of the tree.
    - **Edge Case:** An empty tree should have a height of 0.
3. **isPerfect**

- **Definition:** A perfect binary tree is one where every internal node has exactly two children and all leaves are at the same depth.
- **Goal:** Write a function that checks and returns a boolean value indicating whether the tree is perfect.
- **Edge Case:** An empty tree is considered perfect by definition.

Implementation Guidelines

- **Language-Agnostic Approach:** Focus on the algorithm rather than language-specific syntax. Your solution should be adaptable to any programming language.
- **Recursion:** All three functions should be implemented using recursion.
- **Testing:** Use the provided test cases (or design your own) to verify that your functions work for trees of various structures, including:
  - Trees with multiple levels.
  - Trees with one or both subtrees missing.
  - Single-node trees and empty trees.