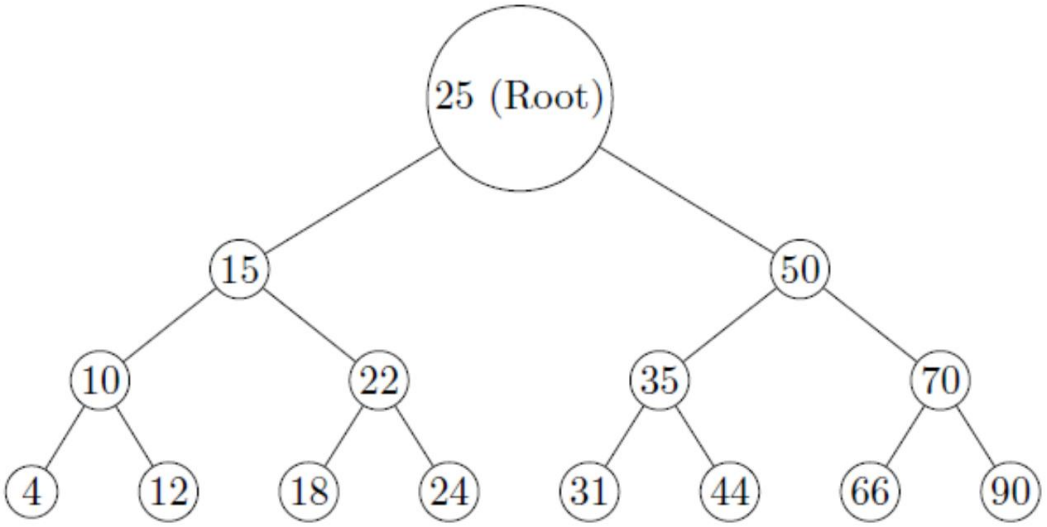
CS401 Lab 9: Binary Search Tree Implementation

Overview

- This lab is to be completed individually
- Focus: Understanding and implementing Binary Search Tree (BST) operations
- Objective: Create a BST and implement various tree traversal and analysis methods

Binary Search Tree Structure

Use the following BST structure as your implementation example:



Requirements

1. Base Node Class

```
public class TreeNode {
    private int data;
    private TreeNode left;
    private TreeNode right;

    public TreeNode(int data) {
        this.data = data;
        left = null;
        right = null;
    }
    // Add getters and setters
}
```

2. BST Implementation

Implement a BST class with the following methods:

1. Tree Creation and Basic Operations:

- *insert(int value)*: Insert a new node
- *delete(int value)*: Remove a node
- *search(int value)*: Find a node

2. Tree Analysis Methods:
    - int *getMaxDepth()*: Find maximum depth of the tree
    - Maximum depth is the number of nodes in the longest path from root to leaf
    - Example: For the given tree, max depth = 4

    - Size Calculation:
        - int *getSizeRecursive()*: Calculate tree size recursively
        - int *getSizeIterative()*: Calculate tree size iteratively

3. Tree Traversal Methods:
    - *void inorderTraversal()*: Left -> Root -> Right
    - *void preorderTraversal()*: Root -> Left -> Right
    - *void postorderTraversal()*: Left -> Right -> Root

Expected Output

Traversal Results

For the example tree, your program should output:

- In-order Traversal:

```
4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 66, 70, 90
```

- Pre-order Traversal:

```
25, 15, 10, 4, 12, 22, 18, 24, 50, 35, 31, 44, 70, 66, 90
```

- Post-order Traversal:

```
4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 66, 90, 70, 50, 25
```

Analysis Results

Your program should also output:

- Maximum depth of the tree
- Size of the tree (both recursive and iterative calculations)

Implementation Requirements

1. Code Organization:
    - Create separate *TreeNode* and BST classes
    - Implement all required methods in the BST class
    - Create a main class for testing

2. Code Documentation:
    - Include detailed comments for each method
    - Explain complex algorithms
    - Document edge cases and error handling

3. Error Handling:
    - Handle empty tree cases
    - Validate input values
    - Manage duplicate values appropriately

Submission Requirements

1. Source Code Files:
    - TreeNode.java
    - BST.java
    - Main.java (test program)

2. Compiled Files:
    - All corresponding .class files
    - Executable JAR file

3. Documentation:
    - PDF file containing:
    - All traversal outputs
    - Tree analysis results
    - Screenshots of program execution
    - README file with:
        - Program description
        - Compilation instructions
        - Execution instructions
        - JAR file execution command

Important Notes

- Test your implementation with the provided example tree
- Verify traversal outputs match the given sequences
- Ensure proper handling of edge cases
- Include appropriate error messages