CSE 222 Data Structures and Algorithms Homework #7 Report

Deniz Altınışık

210104004064

Stock Data Management and Performance Analysis

Introduction

This project implements a stock data management system using an AVL tree to efficiently store stock information. The system supports various operations such as adding, removing, searching, and updating stock data. Additionally, the project includes performance measurement for these operations and visualizes the performance metrics using a graphical user interface (GUI) built with Java Swing. The performance of the add operation, in particular, is expected to exhibit O(log n) behavior due to the properties of the AVL tree.

Components and Classes

1. TestDataGenerator

This class generates test data for the stock data management system, creating a file with a specified number of initial nodes and subsequent operations (ADD, REMOVE, SEARCH, and UPDATE).

Key Methods:

- `generateTestFile(String filename, int initialNodes, int addOps, int removeOps, int searchOps, int updateOps)`: Generates a test file with the specified number of initial nodes and operations.
- `generateRandomSymbol(Random random)`: Generates a random stock symbol.

2. StockDataManager

This class manages the stock data using an AVL tree and provides methods to add, remove, search, and update stock data.

Key Methods:

- `addStock(String symbol, double price, long volume, long marketCap)`: Adds a new stock to the AVL tree.
- `removeStock(String symbol)`: Removes a stock from the AVL tree.
- `searchStock(String symbol)`: Searches for a stock in the AVL tree.
- `updateStock(String symbol, String newSymbol, double newPrice, long newVolume, long newMarketCap)`: Updates an existing stock in the AVL tree.

3. Stock

This class represents a stock with attributes such as symbol, price, volume, and market capitalization.

Key Methods:

- Getter and setter methods for symbol, price, volume, and market capitalization.

4. PerformanceData

This class collects and stores performance data for different operations, maintaining a map of operation types to lists of performance data points.

Key Methods:

- `addData(String operation, int nodes, double time)`: Adds a new performance data point.
- `getData(String operation)`: Retrieves performance data for a specific operation.
- `clearData()`: Clears all stored performance data.

5. PerformanceChart

This class displays the performance data in a graphical format using Java Swing, drawing a scatter plot with different colored dots for each operation type.

Key Methods:

- `displayChart()`: Launches the Swing GUI to display the performance chart.
- `drawGraph(Graphics g)`: Draws the axes and calls `plotData` to plot the data points.
- `plotData(Graphics2D g2d, List<PerformanceData> data, int width, int height, Color color)`: Plots the data points on the graph.

Explanation of the Graph:

- **Axes**: The x-axis represents the number of nodes in the AVL tree, while the y-axis represents the time taken (in microseconds) for each operation.
- **Operations**: Each type of operation (ADD, REMOVE, SEARCH, UPDATE) is represented by dots of different colors.

- **Logarithmic Behavior**: The add operation's performance is expected to show a logarithmic trend (O(log n)), reflecting the balancing properties of the AVL tree.

6. AVLTree

This class implements an AVL tree, a self-balancing binary search tree, ensuring the tree remains balanced after every insertion and deletion.

Key Methods:

- `insert(Stock stock)`: Inserts a new stock into the AVL tree.
- `delete(String symbol)`: Deletes a stock from the AVL tree.
- `search(String symbol)`: Searches for a stock in the AVL tree.
- `rightRotate(Node y)`, `leftRotate(Node x)`: Performs right and left rotations to maintain the balance of the tree.
- `getBalance(Node N)`: Calculates the balance factor of a node.

7. Main

This class contains the `main` method which generates test data, measures performance, and displays the performance chart.

Key Methods:

- `main(String[] args)`: Generates test data and measures performance.
- `measurePerformance(String filename, int initialNodes)`: Measures the performance of the operations by executing them and recording the time taken.

Makefile

The `Makefile` compiles and runs the project, specifying the target files and their dependencies, and includes commands to clean up the compiled files.

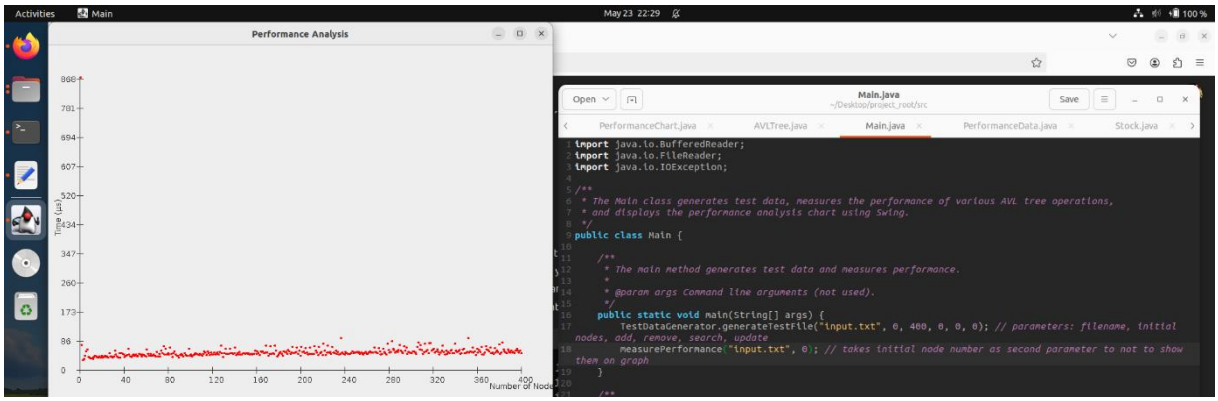
Key Targets:

- `default`: Compiles and runs the project.
- `clean`: Removes the compiled class files.

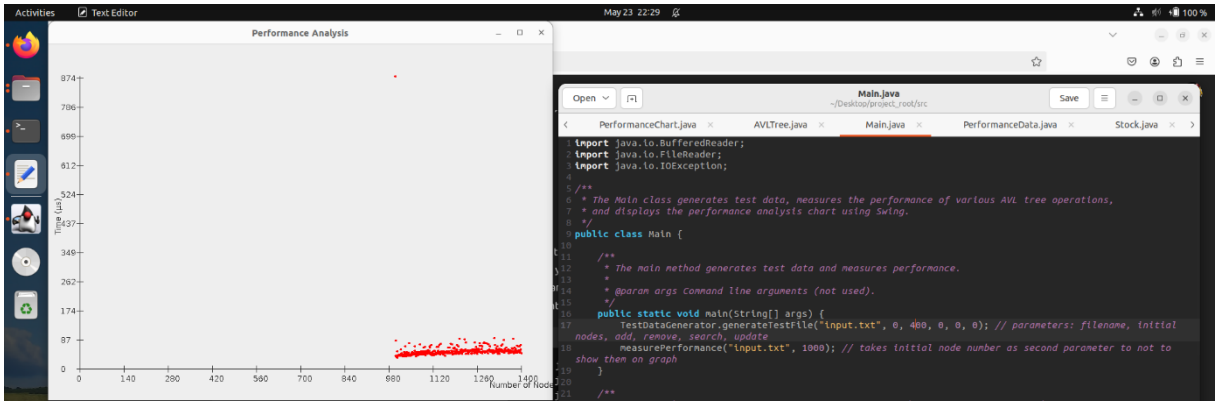
How to Run the Project

1. **Generate Test Data and Measure Performance**:
    - Run the `Main` class, which generates a file named `input.txt` with test data, measures the performance of the stock data operations, and displays the performance chart.
2. **Compile and Run using Makefile**:
    - Use the `Makefile` to compile and run the project by executing the following commands in the terminal:
        - make: Runs program and generates graph.
        - make clean: Removes .class files.
        - make javadoc: Creates javadoc files.
        - make cleandoc: Removes javadoc files.

Screenshots

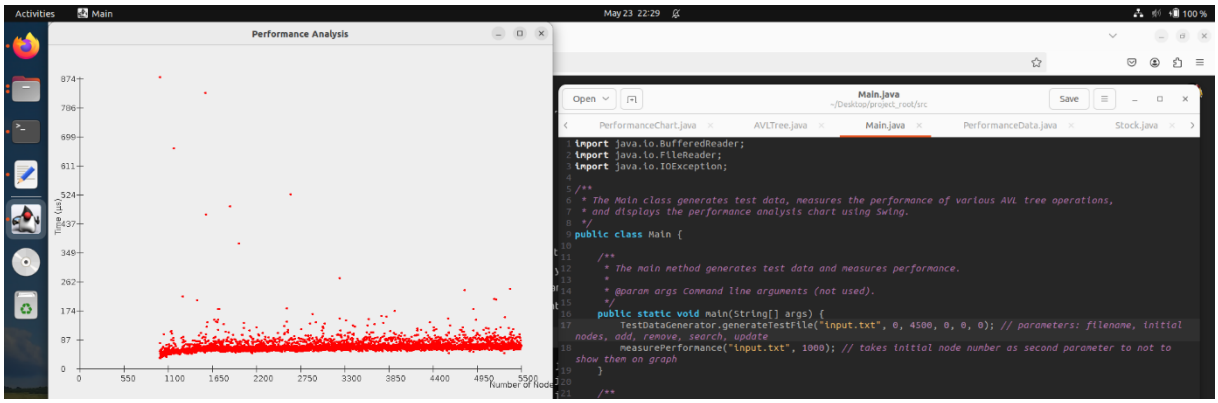


İnitial nodes: 0
add node: 400

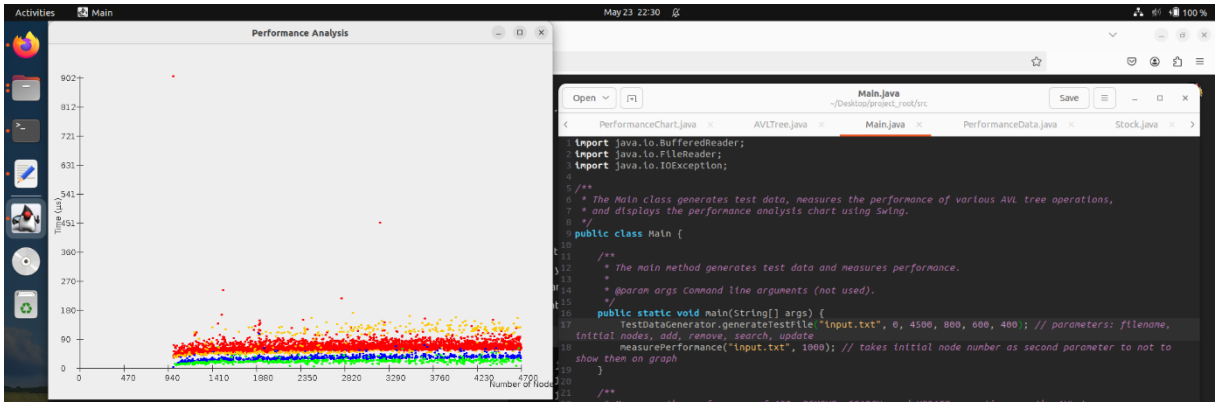


initial nodes: 100

add: 400



initial nodes: 1000
add: 4500



initial nodes: 1000

add(red): 4500
remove(blue): 800
search(green): 600
update(orange): 400