CSE222 Homework7 Report

Akif Safa Angi

200104004079

1-) Introduction

This assignment requires you to implement a balanced tree data structure, specifically an AVL tree, to manage stock data. The goal is to efficiently store, retrieve, and manipulate stock information while maintaining a balanced structure for optimal performance.

2-) Implementation Details

AVLTree.java

The `AVLTree` class is designed to implement an AVL tree, a self-balancing binary search tree.

1. Node Class

- Represents the nodes of the AVL tree.

- Each node contains a `Stock` object, references to left and right child nodes, and a height attribute.

2. Insertion

- The `insert` method allows adding new `Stock` objects to the tree.

- Ensures the tree remains balanced by checking and maintaining the balance factor after each insertion.

3. Search

- The `search` method allows for searching a stock by its symbol in the AVL tree.

- The method traverses the tree based on the comparison of the symbol with the current node's symbol, returning the `Stock` object if found or `null` if not found.

4. Delete

- The `delete` method allows for removing a stock by its symbol from the AVL tree.

- The method first finds the node to be deleted, then performs the necessary restructuring to maintain the AVL tree properties. It handles cases where the node has child or not.

- Ensures the tree remains balanced after deletion by performing rotations if needed.

5. Rotation Methods

- Includes `rightRotate` and `leftRotate` methods to perform rotations necessary to maintain balance.

6. Balance

- The `getBalance` method calculates the balance factor of a node to determine if rotations are needed.

7. Traversal Methods

- Implements in-order, pre-order, and post-order traversals to visit and display the nodes of the tree.

8. printTree Method

- Prints the tree in the tree format

GUIVisualization.java

The `GUIVisualization` class is designed to create a graphical user interface for visualizing the performance of various operations on the AVL tree.

1. Attributes

- `dataPointsX`: List to store x-axis data points.

- `addTimes`, `searchTimes`, `updateTimes`, `removeTimes`: Lists to store the times taken for respective operations.

2. Constructor

- Initializes the attributes and sets up the JFrame with a title, size, close operation, and center position.

3. addDataPoint Method

- Adds a new data point to the lists and triggers a repaint to update the visualization.

4. paint Method

- Overrides the `paint` method to draw the graph.

- Includes axis lines and handles both line and scatter plot types.

5. Helper Methods

- `getMaxYValue`: Finds the maximum y-value to scale the graph accordingly.

InputFileGenerator.java

The `InputFileGenerator` class is designed to generate a file with random stock operations.

1. Main Method

- Accepts command-line arguments for the output file name and the number of operations to generate.

- Calls the `generateInputFile` method to create the file.

2. generateInputFile

- Generates random stock operations (`ADD`, `REMOVE`, `SEARCH`, `UPDATE`).

- Writes these operations to the specified output file.

- Utilizes `Random` class to randomize the stock symbols, prices, and quantities.

Main.java

1. Main Method
   - Accepts a command-line argument for the input file name.
   - Reads the input file and processes each command using the `StockDataManager`.
   - Initializes the GUI for visualizing performance analysis.
2. processCommand Method
   - Parses and executes commands from the input file (`ADD`, `REMOVE`, `SEARCH`, `UPDATE`).
3. performPerformanceAnalysis Method
   - Measures the performance of various operations (add, search, update, remove) on the AVL tree.
   - Calculates average times for these operations and updates the GUI with the results.

Stock.java

The `Stock` class represents a stock with attributes such as symbol, price, volume, and market cap.

1. Attributes
   - `symbol`: The symbol of the stock.
   - `price`: The price of the stock.
   - `volume`: The volume of the stock.
   - `marketCap`: The market cap of the stock.
2. Getters and Setters
   - Provides methods to get and set the values of the stock attributes.
3. toString Method
   - Returns a string representation of the stock object.

StockDataManager.java

The `StockDataManager` class manages stocks using an AVL tree.

1. Attributes
   - `avlTree`: An instance of `AVLTree` to store and manage stocks.
2. Constructor

- Initializes the `avlTree` object.

3. addOrUpdateStock Method
   - Adds a new stock or updates an existing stock's information in the AVL tree.
4. removeStock Method
   - Removes a stock from the AVL tree based on the symbol.
5. searchStock Method
   - Searches for a stock in the AVL tree by its symbol and returns the stock object.
6. updateStock Method
   - Updates the details of an existing stock in the AVL tree.
7. printInOrder, printPreOrder, printPostOrder Methods
   - Prints the stocks in the AVL tree using in-order, pre-order, and post-order traversals respectively.

3-) How to run

To run program:

Firstly write to terminal "**make clean**" to make sure there are just *.java and input files exist.

Write "**make**" to compile

Write "**make run**" to run program

To create a JavaDoc write "**make doc**" to terminal

To delete JavaDoc files write "**make cleandoc**" to terminal

To create a random input file write "**make rand <output_file> <num_operations>**" to terminal

4-) The Parts I Have Completed

I haven't completed performance analysis graph correctly.

5-) Input and output tests from program

```
a@DESKTOP-5F6OAM7: /mnt,
a@DESKTOP-5F6OAM7:/mnt/c/Users/akif_/Desktop/Data/200104004079_Akif_Safa_Angi_HW7/src$ make run input.txt
java -Xint Main input.txt
Stock [symbol=XYZ, price=25.5, volume=1000000, marketCap=5000000]
Stock [symbol=DEF, price=15.75, volume=750000, marketCap=3750000]
Stock [symbol=JKL, price=22.5, volume=1500000, marketCap=7500000]
AVL Tree:
JKL
   left->DEF
      left->null
      right->null
   right->XYZ
      left->MNO
         left->null
         right->null
      right->null
In Order Traversal:
Stock [symbol=DEF, price=15.75, volume=750000, marketCap=3750000]
Stock [symbol=JKL, price=22.5, volume=1500000, marketCap=7500000]
Stock [symbol=MNO, price=12.8, volume=600000, marketCap=3000000]
Stock [symbol=XYZ, price=25.5, volume=1000000, marketCap=5000000]
Pre Order Traversal:
Stock [symbol=JKL, price=22.5, volume=1500000, marketCap=7500000]
Stock [symbol=DEF, price=15.75, volume=750000, marketCap=3750000]
Stock [symbol=XYZ, price=25.5, volume=1000000, marketCap=5000000]
Stock [symbol=MNO, price=12.8, volume=600000, marketCap=3000000]
Post Order Traversal:
Stock [symbol=DEF, price=15.75, volume=750000, marketCap=3750000]
Stock [symbol=MNO, price=12.8, volume=600000, marketCap=3000000]
Stock [symbol=XYZ, price=25.5, volume=1000000, marketCap=5000000]
Stock [symbol=JKL, price=22.5, volume=1500000, marketCap=7500000]
```

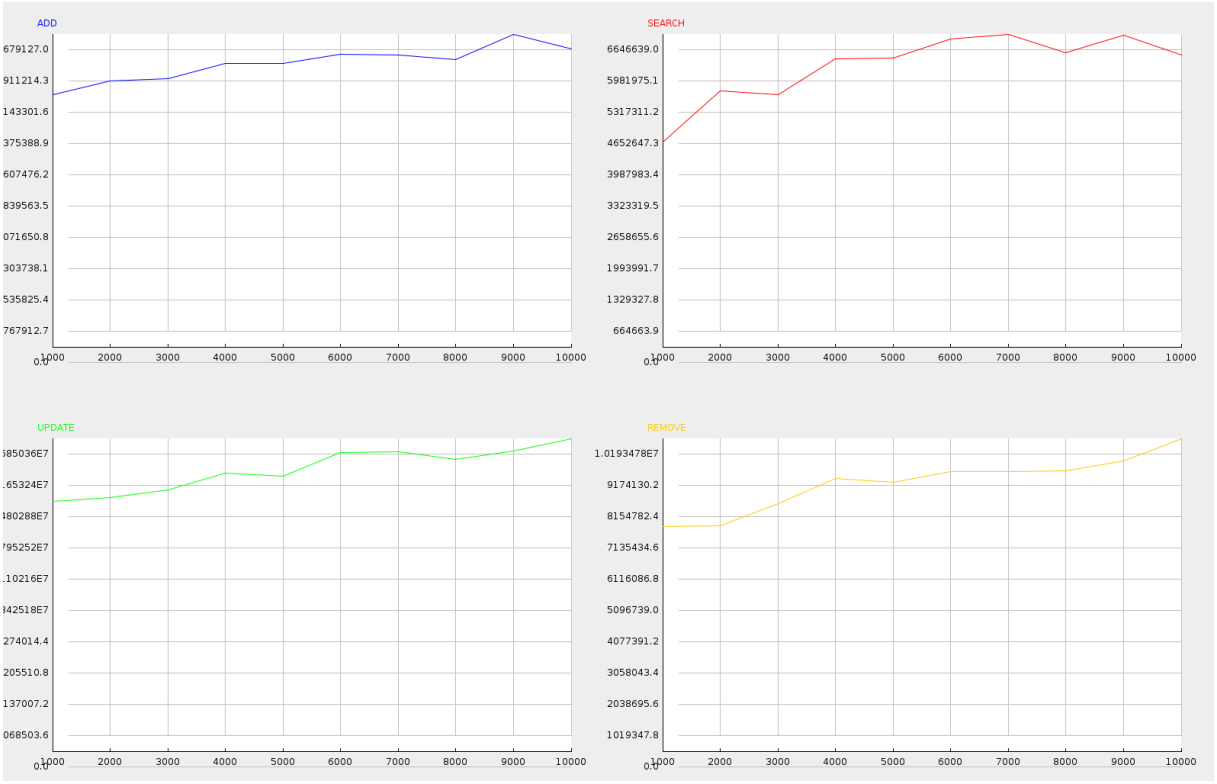
C:\Users\akif_\Desktop\Data\200104004079_Akif_Safa_Angi_HW7\src\input.txt - Notepad++

Dosya Düzen Ara Görünüm Kodlama Diller Ayarlar Araçlar Makrolar Çalıştır Eklentiler Pencereler ?

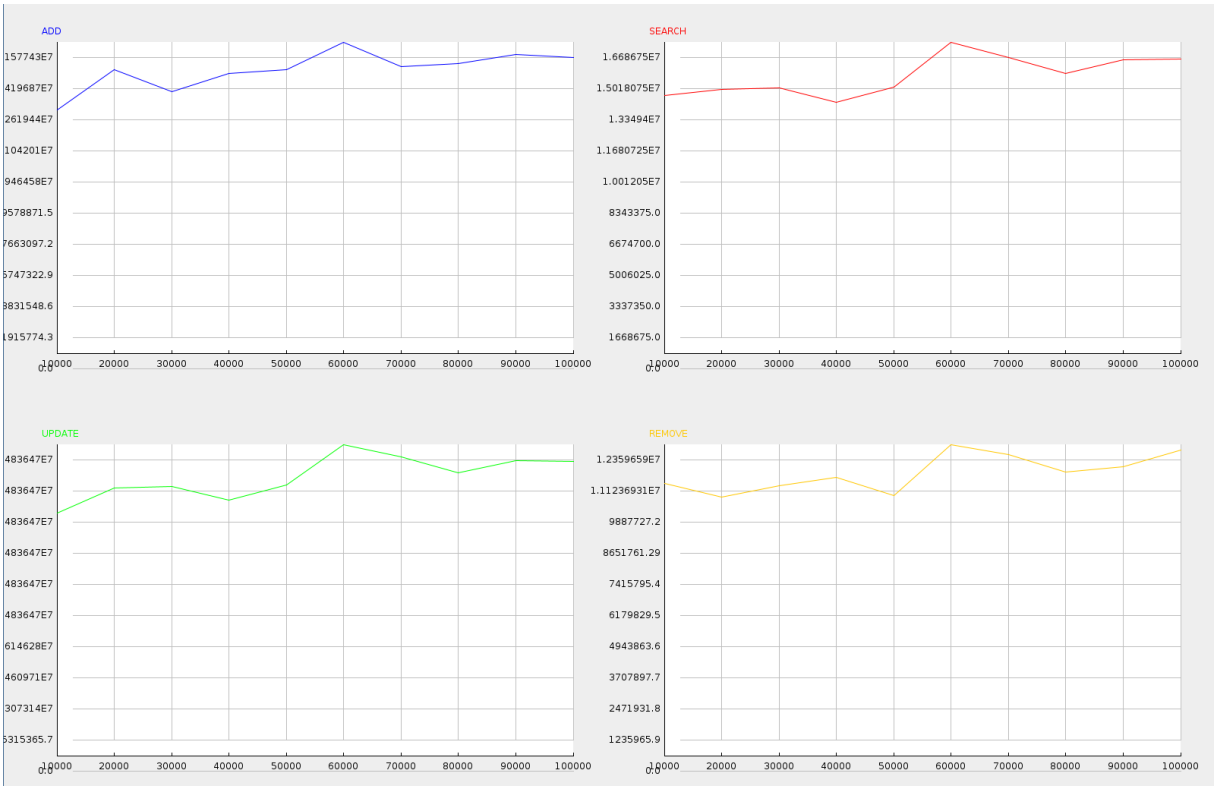
input.txt

```
1  ADD XYZ 25.50 1000000 5000000
2  ADD ABC 10.25 500000 2500000
3  ADD DEF 15.75 750000 3750000
4  SEARCH XYZ
5  ADD GHI 20.00 1250000 6250000
6  REMOVE ABC
7  SEARCH DEF
8  UPDATE GHI JKL 22.50 1500000 7500000
9  ADD MNO 12.80 600000 3000000
10 SEARCH JKL
```

Performance Graph



.


ADD
157743E7
419687E7
261944E7
104201E7
946458E7
9578871.5
7663097.2
5747322.9
3831548.6
1915774.3
0.0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000
SEARCH
1.668675E7
1.5018075E7
1.33494E7
1.1680725E7
1.001205E7
8343375.0
6674700.0
5006025.0
3337350.0
1668675.0
0.0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000
UPDATE
483647E7
483647E7
483647E7
483647E7
483647E7
483647E7
614628E7
460971E7
307314E7
5315365.7
0.0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000
REMOVE
1.2359659E7
1.11236931E7
9887727.2
8651761.29
7415795.4
6179829.5
4943863.6
3707897.7
2471931.8
1235965.9
0.0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000