Alper Tavşanoğlu-210104004142-HW-7-Report

The primary goal of this project is to efficiently manage stock data using a balanced tree data structure, specifically an AVL tree. This ensures optimal performance in operations such as insertion, deletion, and searching by maintaining the balanced nature of the tree.

AVL Tree Implementation

The AVLTree class maintains the balance of the tree by ensuring the height difference between the left and right subtrees of any node does not exceed one. This is achieved through rotations left, right, left-right, and right-left based on the balance factor of each node.

Key Methods

insert (Node node, Stock stock): Inserts a new stock or updates an existing one based on the symbol.

remove (Node node, String symbol): Removes a stock based on its symbol.

search (Node node, String symbol): Searches for a stock by its symbol.

balance (Node node): Balances the tree by performing appropriate rotations.

Performance Optimization

To ensure optimal performance, the AVL tree operations are designed to run in logarithmic time complexity, O(logn), where n n is the number of nodes in the tree. This is crucial for maintaining quick response times as the size of the data grows.

The system reads commands from an input file, which includes operations like ADD, REMOVE, SEARCH, and UPDATE. Each command specifies the operation to be performed and the necessary stock attributes.

Testing was conducted by generating input files with varying sizes and operation mixes using a custom script. The system's response to these operations was then measured in terms of execution time and correctness.

Performance tests were conducted by measuring the execution time for each operation across different tree sizes. These tests helped verify the logarithmic complexity of the operations and ensure the efficiency of the AVL tree implementation.

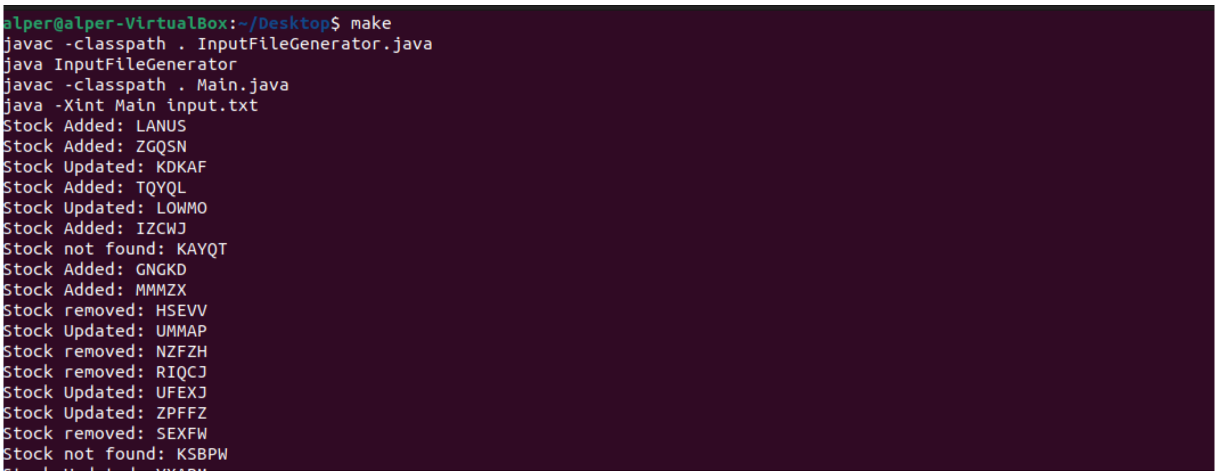
Performance graphs were plotted showing the relationship between the number of nodes and the execution time for each operation. These graphs confirmed that the operations maintained a logarithmic time complexity as expected.

A simple Java Swing interface was implemented to visually represent the performance analysis graphs. This GUI helps in easily interpreting the performance data and understanding the efficiency of different operations.

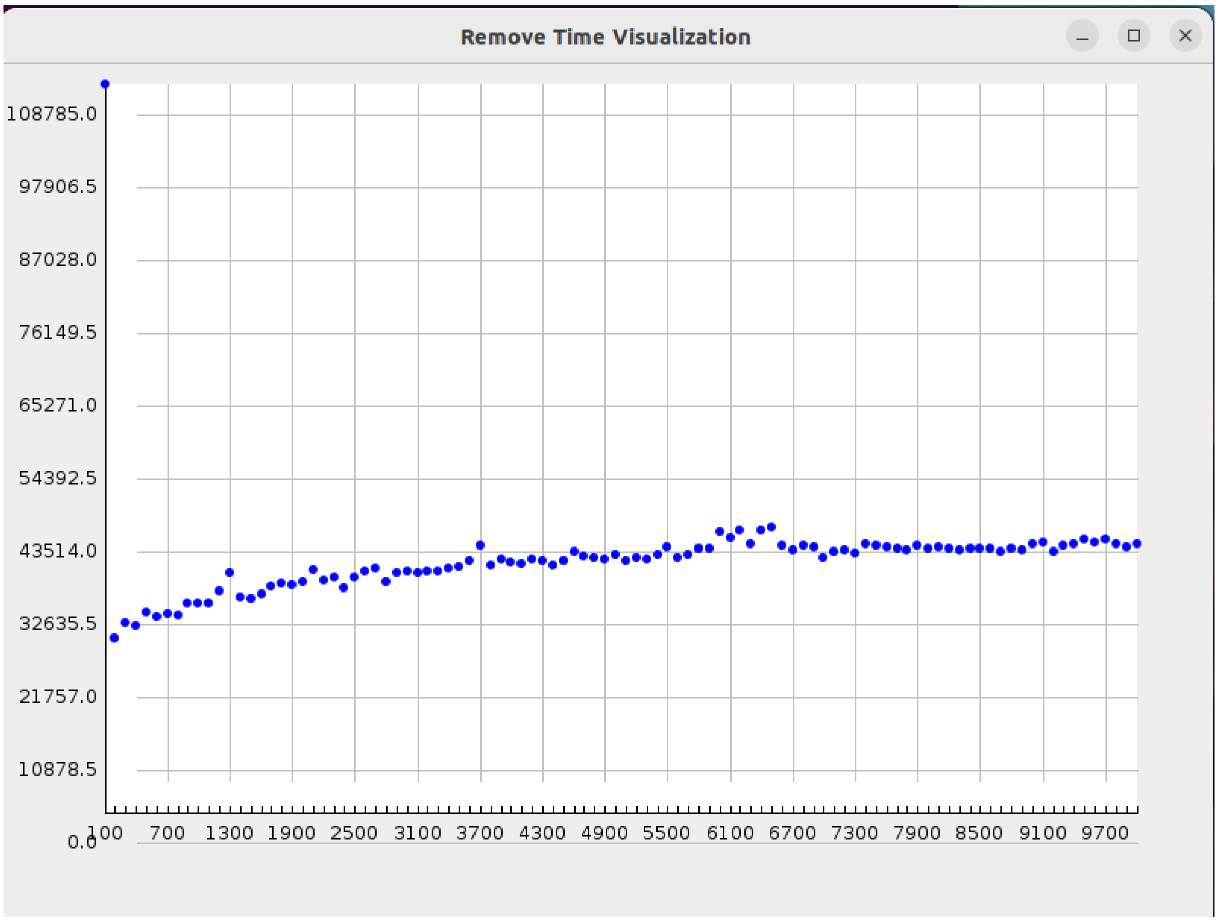
During implementation, challenges such as managing duplicate symbols during insertion and ensuring efficient memory usage during rotations were encountered. These were addressed by refining the tree balancing logic and optimizing the node insertion algorithm.

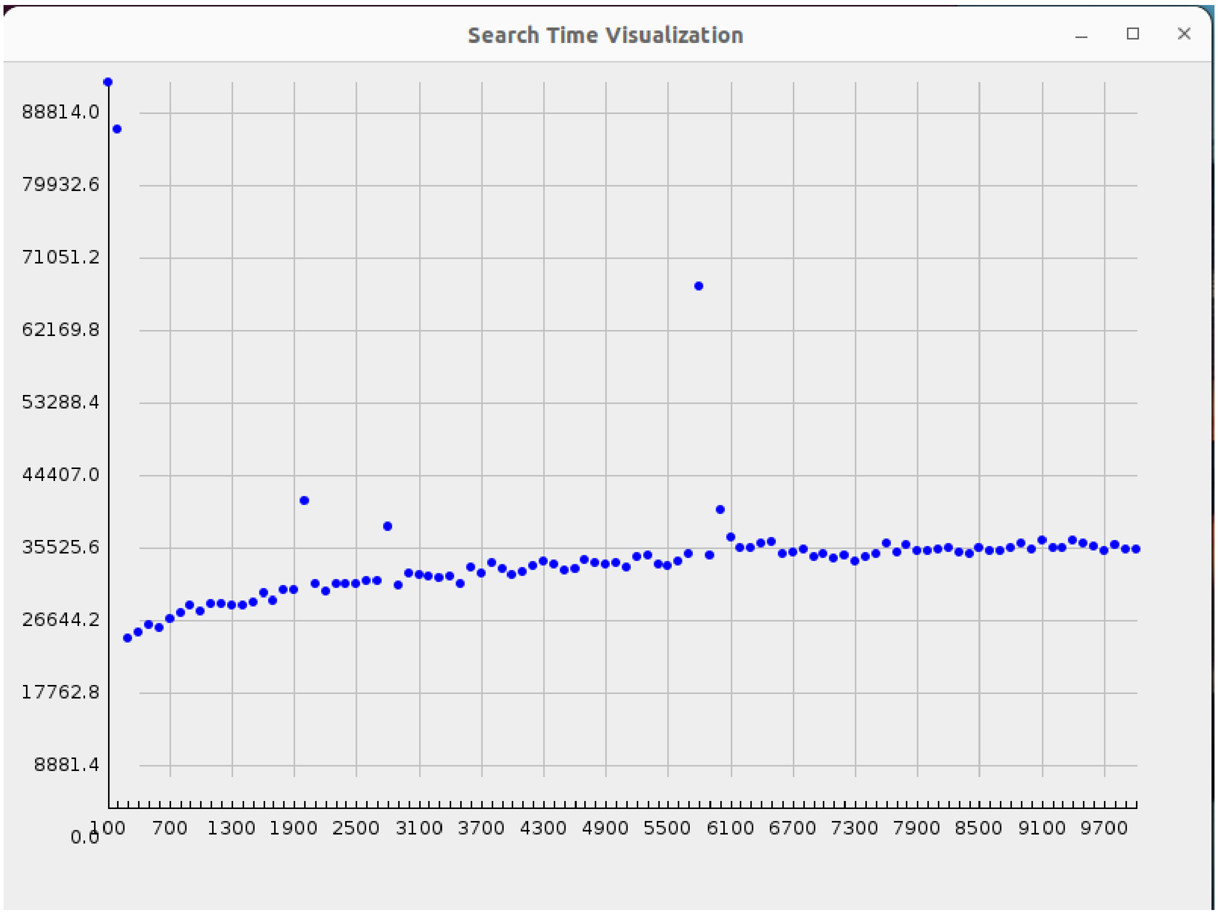
The AVL tree-based stock data management system effectively meets the requirements set out in the assignment. It provides efficient management of stock data with operations that adhere to O(logn) time complexity, demonstrating the capabilities of AVL trees in handling balanced data operations.

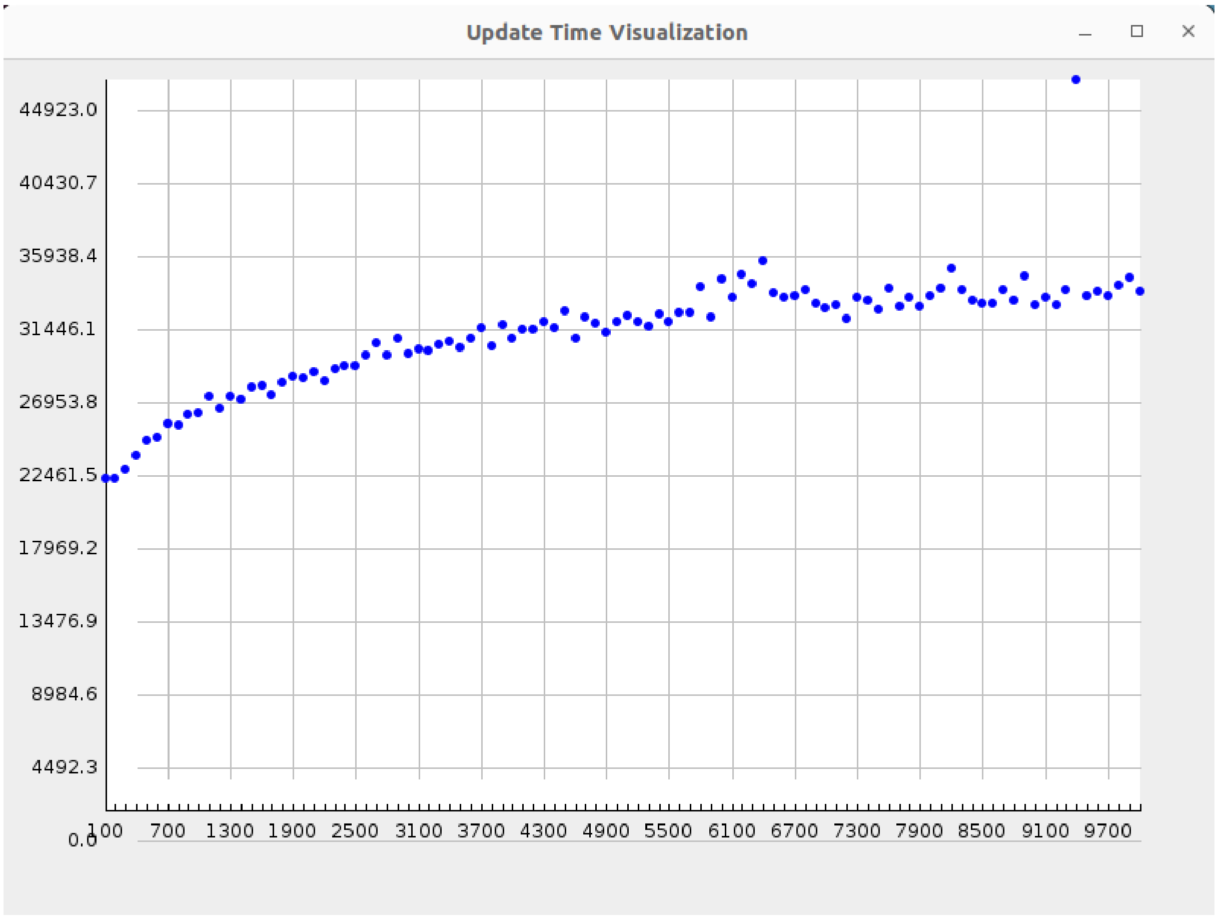
Running The Code with makefile using **make** command. It generated input.txt first and then with this input.txt makfile running main.

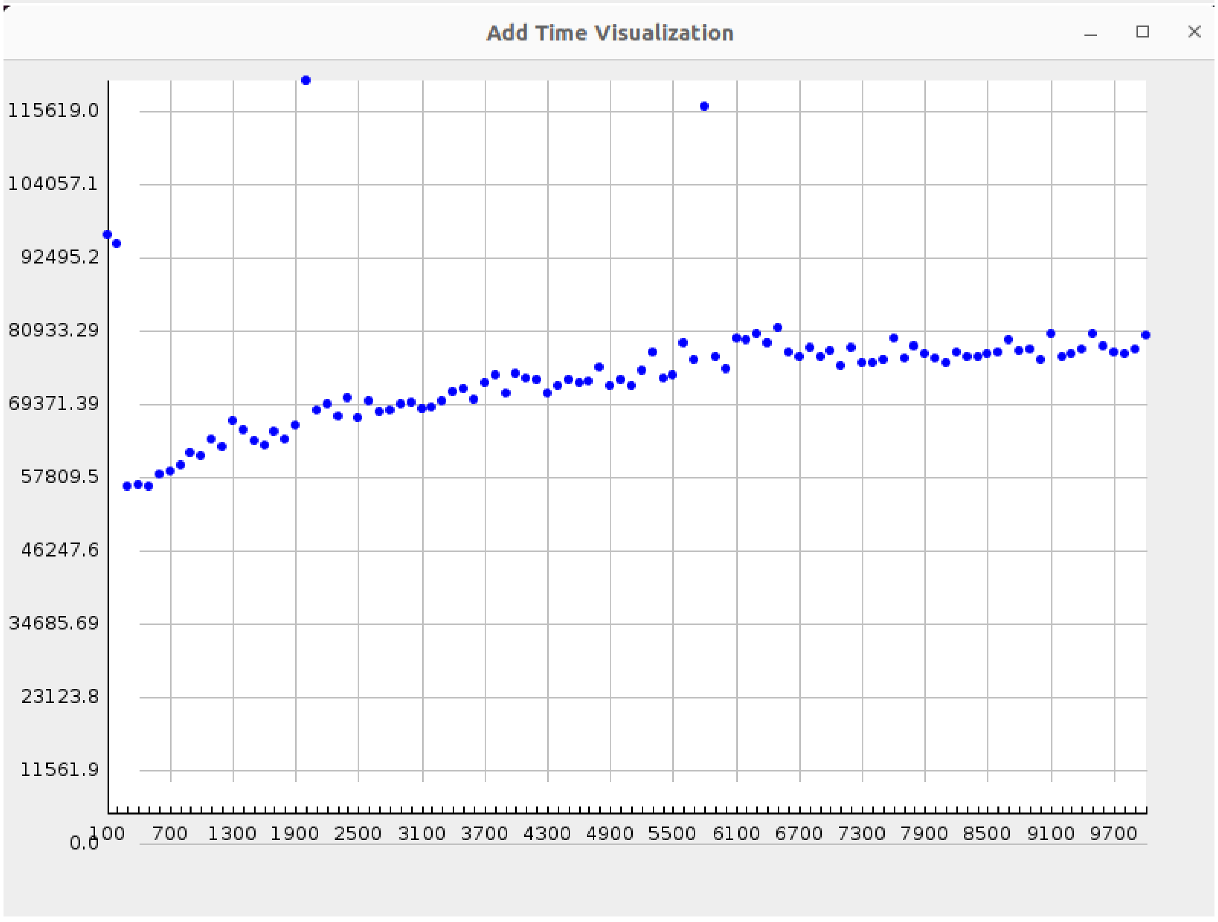


```
alper@alper-VirtualBox:~/Desktop$ make
javac -classpath . InputFileGenerator.java
java InputFileGenerator
javac -classpath . Main.java
java -Xint Main input.txt
Stock Added: LANUS
Stock Added: ZGQSN
Stock Updated: KDKAF
Stock Added: TQYQL
Stock Updated: LOWMO
Stock Added: IZCWJ
Stock not found: KAYQT
Stock Added: GNGKD
Stock Added: MMMZX
Stock removed: HSEVV
Stock Updated: UMMAP
Stock removed: NZFZH
Stock removed: RIQCJ
Stock Updated: UFEXJ
Stock Updated: ZPFFZ
Stock removed: SEXFW
Stock not found: KSBPW
```


Remove Time Visualization
108785.0
97906.5
87028.0
76149.5
65271.0
54392.5
43514.0
32635.5
21757.0
10878.5
0.0
100
700
1300
1900
2500
3100
3700
4300
4900
5500
6100
6700
7300
7900
8500
9100
9700


Search Time Visualization
88814.0
79932.6
71051.2
62169.8
53288.4
44407.0
35525.6
26644.2
17762.8
8881.4
0.0
100
700
1300
1900
2500
3100
3700
4300
4900
5500
6100
6700
7300
7900
8500
9100
9700


Update Time Visualization
44923.0
40430.7
35938.4
31446.1
26953.8
22461.5
17969.2
13476.9
8984.6
4492.3
0.0
100
700
1300
1900
2500
3100
3700
4300
4900
5500
6100
6700
7300
7900
8500
9100
9700


Add Time Visualization
115619.0
104057.1
92495.2
80933.29
69371.39
57809.5
46247.6
34685.69
23123.8
11561.9
0.0
100
700
1300
1900
2500
3100
3700
4300
4900
5500
6100
6700
7300
7900
8500
9100
9700