AVL Tree Implementation

Key Methods and Their Functionality

1. **Insert method:** Inserts a new stock into the AVL tree.
    a. Inserts a new node with the given stock.
    b. Updates the height and calculate balance factor.
    c. Balances tree if necessary by performing rotations .

2. **Delete method:** Deletes a stock with the given symbol from the AVL tree.
    a. Deletes a stock with the given symbol.
    b. Updates the height and calculate balance factor.
    c. Balances tree if necessary by performing rotations .

3. **Search method:** Searches for a stock with the given symbol in the AVL tree.
    a. Searches a stock with the given symbol.
    b. Returns the node containing the stock, if it exists.
    c. Traverse the tree by comparison the symbols.

4. **Balancing methods:** Performs a right rotation, left rotation, left-right rotatin or right-left rotation on the given node.
    a. **Right rotation:**
        i. Performs a right rotation on the subtree rooted at the given node.
        ii. Updates the height of current node and new root.
    b. Left rotation:
        i. Performs a left rotation on the subtree rooted at the given node.
        ii. Updates the height of current node and new root.
    c. Left-right rotation:
        i. Performs a left-right rotation on the subtree rooted at the given node.
    d. Right-left rotation:
        i. Performs a right-left rotation on the subtree rooted at the given node.

Other Methods and Their Functionality

1. **Get minimum node method:** Helper method to find the node with the minimum value in the subtree.

2. **Update height method:** The height of node is updated based on the heights of its children.

3. **Calculate balance factor method:** The balance factor is calculated with differences between height of the left and height of the right subtrees.

4. **Traversal methods:**
    a. **In order traversal method:** Performs an in-order traversal of the AVL tree and prints the stock data.
    b. **Pre order traversal method:** Performs an pre-order traversal of the AVL tree and prints the stock data.
    c. **Post order traversal method:** Performs an post-order traversal of the AVL tree and prints the stock data.

Input Format and Commands Processing

Input File Format

The input file contains command like fallowing:

- ADD (symbol, price, volume, marketCap)
- REMOVE (symbol, price, volume, marketCap)
- SEARCH (symbol, price, volume, marketCap)
- UPDATE (Symbol, newSymbol, newPrice, newVolume, newMarketCap)

Command Processing

Processes each command from the input file and performs corresponding stock management operations.

Generating Commands

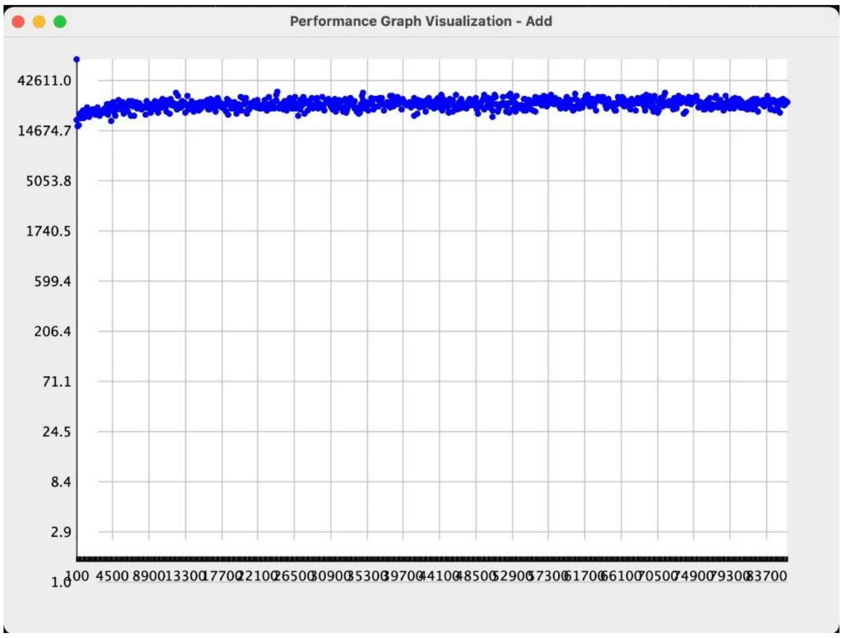
The Command class generates a file with a specified number of random commands. Each command is related to stock operations such as ADD, REMOVE, SEARCH, and UPDATE. The commands are written to a specified file

Performance Analysis

Performance Analysis Graphs

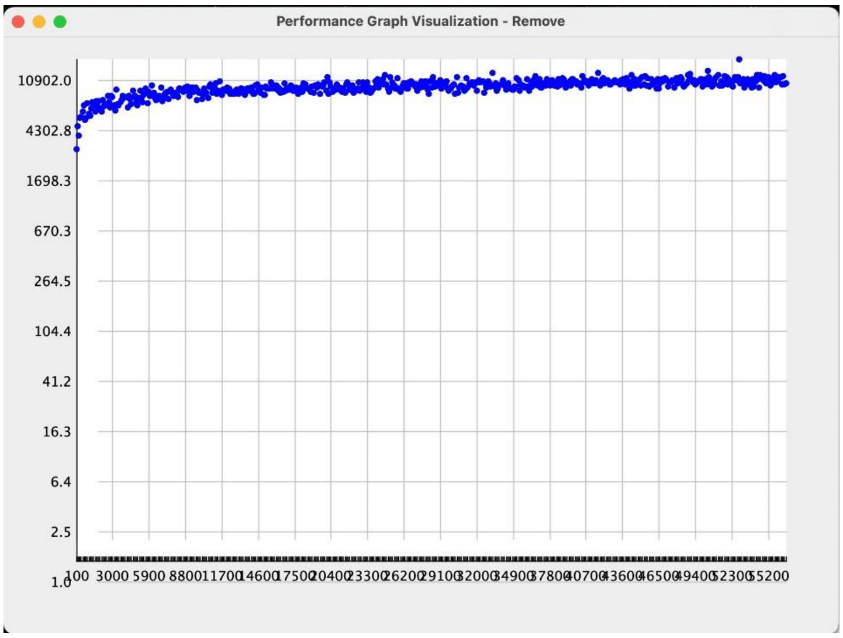
The performance of add, remove and search operations was measured and plotted.

1. **ADD operation:**
    a. **Graph:**



    b. **Time Complexity:** O(log n) due to the AVL Tree.
    c. **Explanation:** Insertion in an AVL tree involves first finding the correct position for the new node using standard BST insertion. After placing the node, we update the heights of the affected nodes and check their balance factors. If an imbalance is detected we perform the necessary rotations to restore balance and ensure the tree remains balanced.

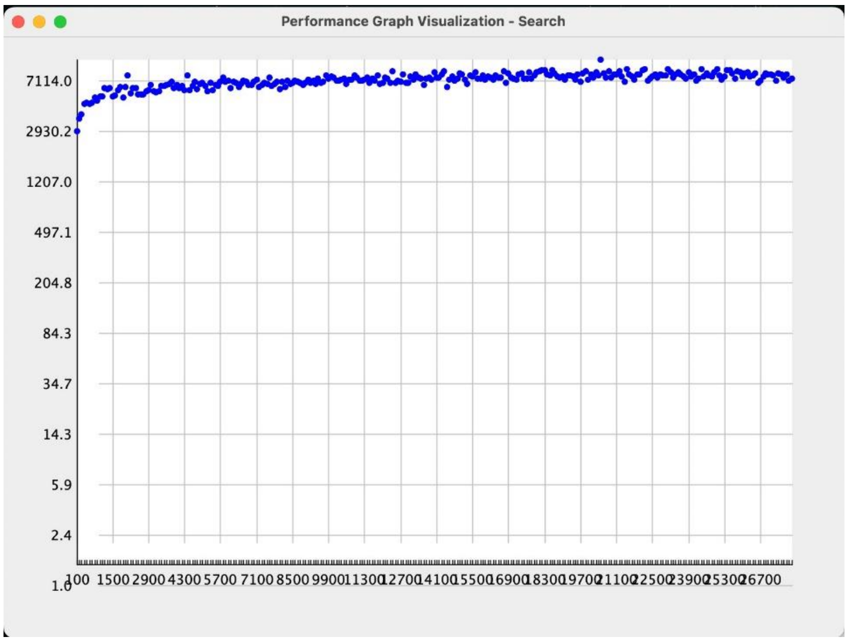
2. **REMOVE operation:**
    a. **Graph:**



    b. **Time Complexity: O(log n) due to the AVL Tree.**

c. **Explanation:** Removal in an AVL tree involves first finding and removing the target node using standard BST deletion. After the node is removed, we update the heights of the affected nodes and check their balance factors. If an imbalance is detected, we perform the necessary rotations to restore balance and ensure the tree remains balanced.

3. **SEARCH operation:**
    a. **Graph:**



    b. **Time Complexity: O(log n) due to th AVL Tree.**
    c. **Explanation:** Search in an AVL tree involves traversing the tree from the root to the target node, following the binary search tree property. At each node, we compare the target value with the current node's value and move left if the target is smaller or right if the target is larger. This process continues until the target node is found or a leaf is reached.

Challanges Faced

1. **Plotting Graph**
   Challange: Measuring the performance of tree operations and plotting the results accurately required data collection and visualization.

   Solution: Collect detailed performance data during tests and get the average of processing times.

2. **Implementing Correct Rotations**
   Challange: Implementing right, left, left-right, and right-left rotations correctly to rebalance the tree without introducing errors.

   Solution: Separate the rotation logic into distinct methods for right rotation, left rotation, left-right rotation, and right-left rotation. Ensure each method correctly rearranges the nodes and updates their heights.

3. **Handling Large Input Files Efficiently**
   Challange: Processing a large number of commands from an input file efficiently, especially when the commands involve multiple tree operations.

   Solution: Buffered reading was used for handling large files and ensuring that tree operations (add, remove, search) were optimized for performance. The processCommand method was used to handle each command and measure the execution time.