CSE 222 HW7 Report

AVL Tree Structure

The **AVLTree** is implemented with an inner class **Node**, which represents the nodes in the tree. Each node stores a **Stock** object, references to its left and right children, and its **height**. The height of a node is used to maintain the balance of the tree. Every time a node is inserted or deleted; the tree will balance itself if necessary, and for balanced trees insertion, deletion and search operations will have a time complexity of O(log n). AVL Trees are balanced trees so time complexities for these operations will be O(log n).

Insertion

The insert method adds a new **Stock** object to the **AVLTree**. If a node with the same stock symbol already exists, its data is **updated**. The tree is then rebalanced to maintain the AVL property.

> **Comparison and Placement**: The new stock is compared with the current node's stock symbol to determine its position (left or right subtree).
> **Height Update**: After insertion, the height of the current node is updated.
> **Balancing**: The balance factor of the node is calculated. If the balance factor is outside the range [-1, 1], rotations are performed to restore balance.

Deletion

The delete method removes a Stock object from the AVL Tree based on its symbol. The deletion process follows 2 steps: first is deletion, and the second is rebalancing the tree.

> **Deletion**: The node to be deleted is found. If it has one or no children, it is removed directly. If it has two children, the in-order successor (smallest in the right subtree) is found, and its data replaces the current node's data. The successor node is then deleted.
> **Height Update and Balancing**: After deletion, the height of the current node is updated, and the node is rebalanced if necessary.

Search Algorithm

Standard binary search tree (BST) search process is used. It traverses the tree starting from the root and compares the target symbol with the current node's stock symbol to decide whether to move left or right.

Right Rotation

A right rotation is performed on a node y when the left subtree of y is taller by more than one level. Here's how right rotation works:

Identify Nodes: Let x be the left child of y, and T2 be the right child of x.
Perform Rotation:

- x becomes the new root of the subtree.
- y becomes the right child of x.
- T2 becomes the left child of y.

Update Heights: Update the heights of y and x.

Left Rotation

A left rotation is performed on a node x when the right subtree of x is taller by more than one level. Here's how left rotation works:
Identify Nodes: Let y be the right child of x, and T1 be the left child of y.
Perform Rotation:

- y becomes the new root of the subtree.
- x becomes the left child of y.
- T1 becomes the right child of x.

Update Heights: Update the heights of x and y.

Balancing

Determine the Type of Imbalance:

- **Left-Left (LL) Case**: If balance > 1 and the symbol of the newly inserted or deleted node is less than the symbol of the left child of node. This means the imbalance is in the left subtree of the left child.
- **Right-Right (RR) Case**: If balance < -1 and the symbol of the newly inserted or deleted node is greater than the symbol of the right child of node. This means the imbalance is in the right subtree of the right child.
- **Left-Right (LR) Case**: If balance > 1 and the symbol of the newly inserted or deleted node is greater than the symbol of the left child of node. This means the imbalance is in the right subtree of the left child.
- **Right-Left (RL) Case**: If balance < -1 and the symbol of the newly inserted or deleted node is less than the symbol of the right child of node. This means the imbalance is in the left subtree of the right child.

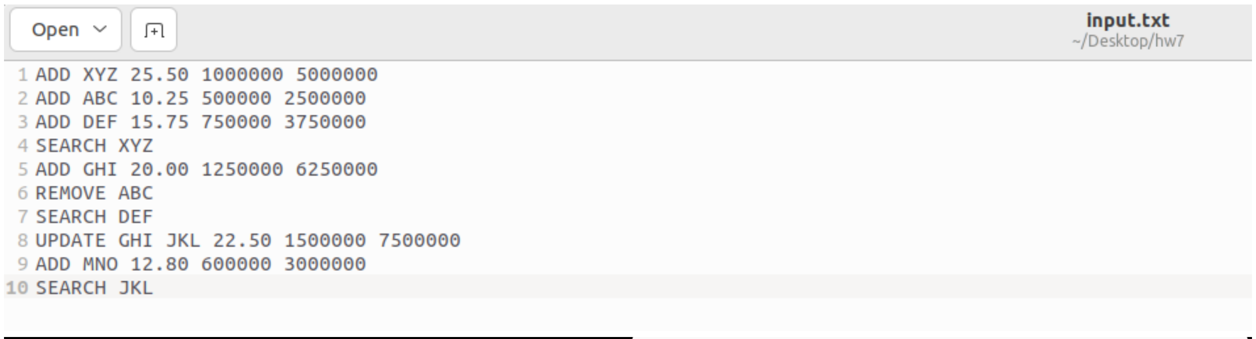
Perform the Appropriate Rotation:

- **Left-Left (LL) Case**: Perform a right rotation on node.
- **Right-Right (RR) Case**: Perform a left rotation on node.
- **Left-Right (LR) Case**: Perform a left rotation on the left child of node, then perform a right rotation on node.
- **Right-Left (RL) Case**: Perform a right rotation on the right child of node, then perform a left rotation on node.

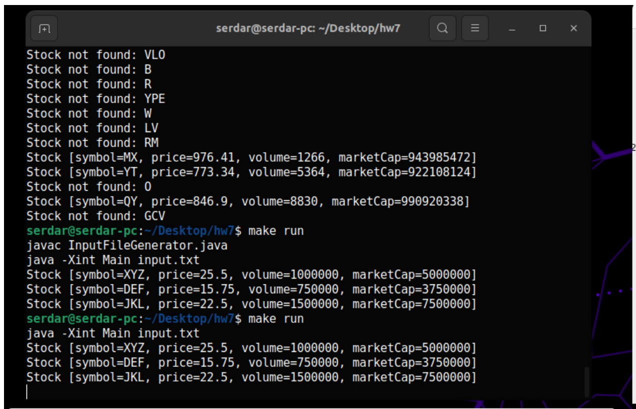
Note: I also added a function to print the AVLTree but it’s not called anywhere.

TESTS:

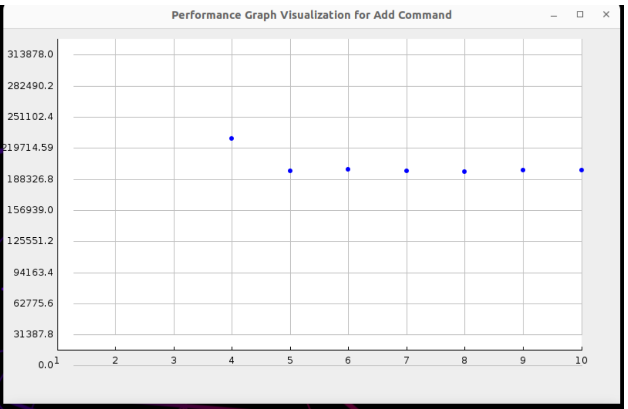
Test with small command number (10)

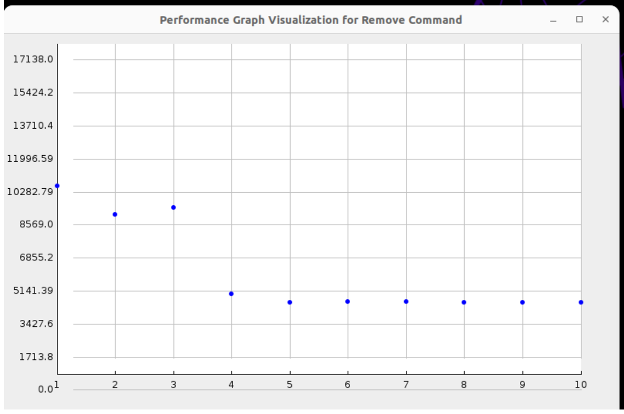


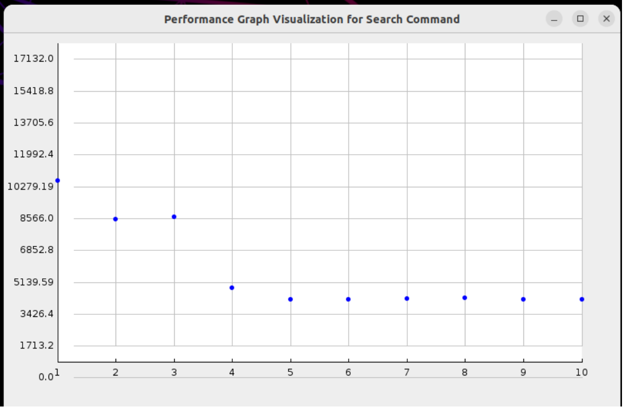
```
ADD XYZ 25.50 1000000 5000000
ADD ABC 10.25 500000 2500000
ADD DEF 15.75 750000 3750000
SEARCH XYZ
ADD GHI 20.00 1250000 6250000
REMOVE ABC
SEARCH DEF
UPDATE GHI JKL 22.50 1500000 7500000
ADD MNO 12.80 600000 3000000
SEARCH JKL
```



```
Stock not found: VLO
Stock not found: B
Stock not found: R
Stock not found: YPE
Stock not found: W
Stock not found: LV
Stock not found: RM
Stock [symbol=MX, price=976.41, volume=1266, marketCap=943985472]
Stock [symbol=YT, price=773.34, volume=5364, marketCap=922108124]
Stock not found: O
Stock [symbol=QY, price=846.9, volume=8830, marketCap=990920338]
Stock not found: GCV
serdar@serdar-pc:~/Desktop/hw7$ make run
javac InputFileGenerator.java
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]
serdar@serdar-pc:~/Desktop/hw7$ make run
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]
```

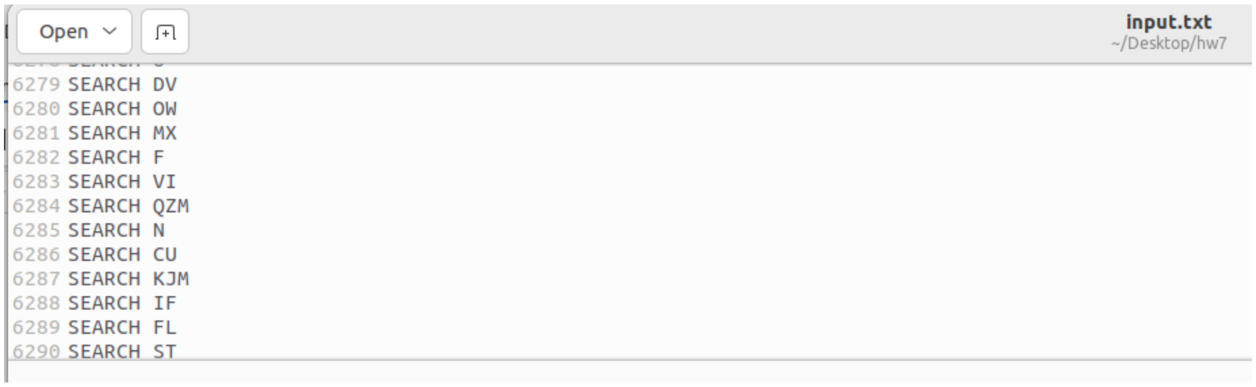




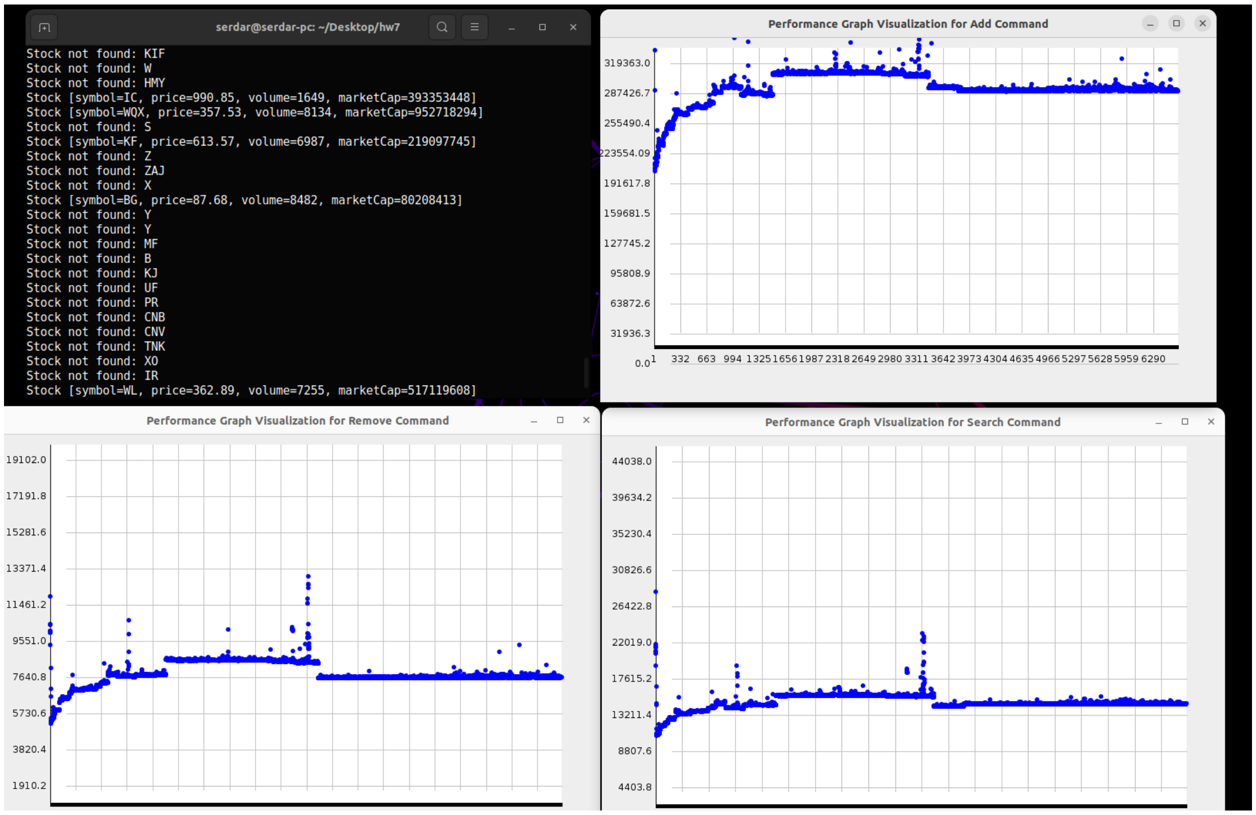


When working with a few commands, we can't see a log n graph, but when there are more, the graph is clearer.

Test with big command number (6290) (randomly generated)



```
SEARCH DV
SEARCH OW
SEARCH MX
SEARCH F
SEARCH VI
SEARCH QZM
SEARCH N
SEARCH CU
SEARCH KJM
SEARCH IF
SEARCH FL
SEARCH ST
```



```
Stock not found: KIF
Stock not found: W
Stock not found: HMY
Stock [symbol=IC, price=990.85, volume=1649, marketCap=393353448]
Stock [symbol=WQX, price=357.53, volume=8134, marketCap=952718294]
Stock not found: S
Stock [symbol=KF, price=613.57, volume=6987, marketCap=219097745]
Stock not found: Z
Stock not found: ZAJ
Stock not found: X
Stock [symbol=BG, price=87.68, volume=8482, marketCap=80208413]
Stock not found: Y
Stock not found: Y
Stock not found: MF
Stock not found: B
Stock not found: KJ
Stock not found: UF
Stock not found: PR
Stock not found: CNB
Stock not found: CNV
Stock not found: TNK
Stock not found: XO
Stock not found: IR
Stock [symbol=WL, price=362.89, volume=7255, marketCap=517119608]
```

How to compile program:

make: compile files

make run: run the main program

make random: create an input.txt file with random commands (you can change command numbers in InputFIleGenerator.java)

make doc: create javadoc

make clean: clean doc file and class files