CSE222 / BİL505
Data Structures and Algorithms
Homework #7 – Report

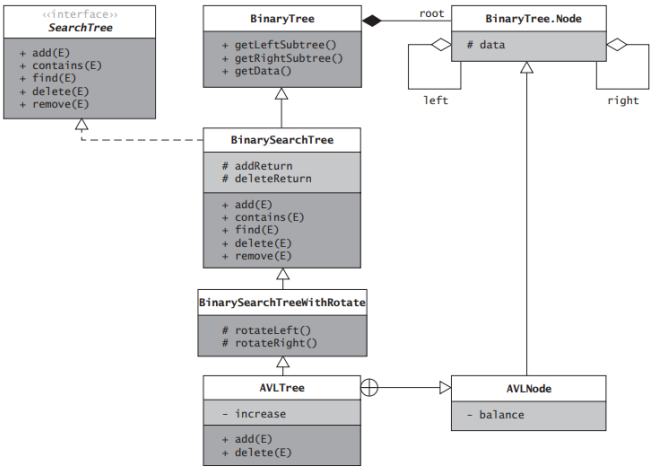
Muhammet Talha Memişoğlu

- ## INTRODUCTION

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

- ## METHODOLOGY

I have implemented an AVL Tree to manage a custom Stock class. The AVL Tree is designed to efficiently store, retrieve, and manipulate stock information while ensuring the tree remains balanced for optimal performance. Below is the UML class diagram for my AVL Tree implementation.



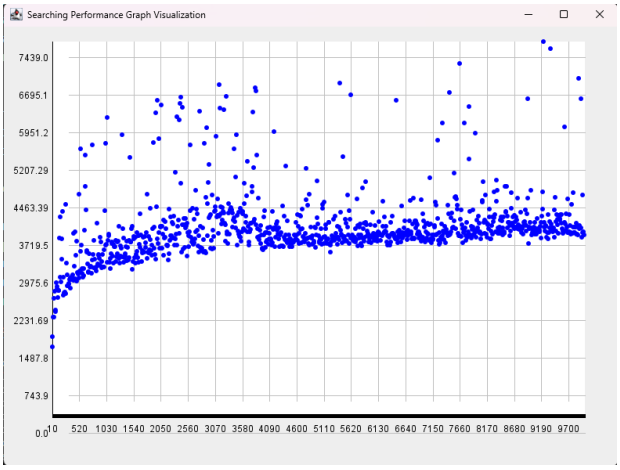
Firstly, I implemented the **BinarySearchTreeWithRotate** class, which extends the **BinarySearchTree** class. In this class, I implemented the **rotateLeft** and **rotateRight** methods. Subsequently, I created the **AVLTree** class, which extends the **BinarySearchTreeWithRotate** class. In the **AVLTree** class, I defined the **add** and **delete** methods, which use rebalance methods and manage node heights. Additionally, I decided to implement traversal methods in the **AVLTree** class.

I have also implemented a **Stock** class and StockDataManager class. The **StockDataManager** class builds an AVL Tree to store instances of the **Stock** class

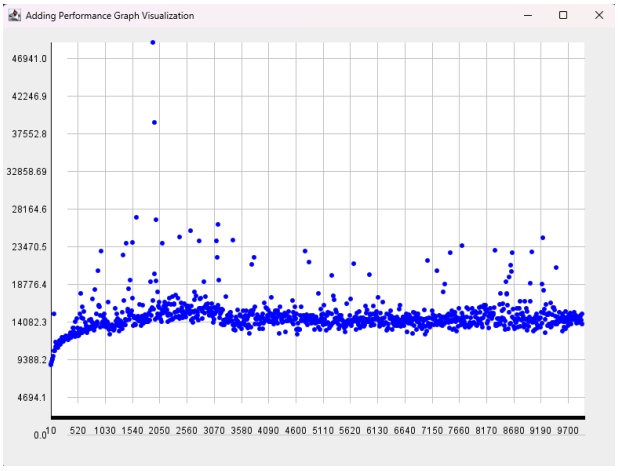
- **RESULTS**

To test the adding, searching, and removing operations, I have implemented the **RandomTreeGenerator** class, which creates random node commands in a file based on a node size provided by the user. The **Main** class reads the node commands from this file and uses the **StockDataManager** class to build the AVL Trees. And I have also implemented **PerformanceAnalysis** class, which adds, searches, and removes one node in the AVL Tree multiple times as specified by the input. This class returns the average time cost for these operations. Finally, using the **GUIVisualization** class, I transferred the cost time of specific operations to graphs for visualization.

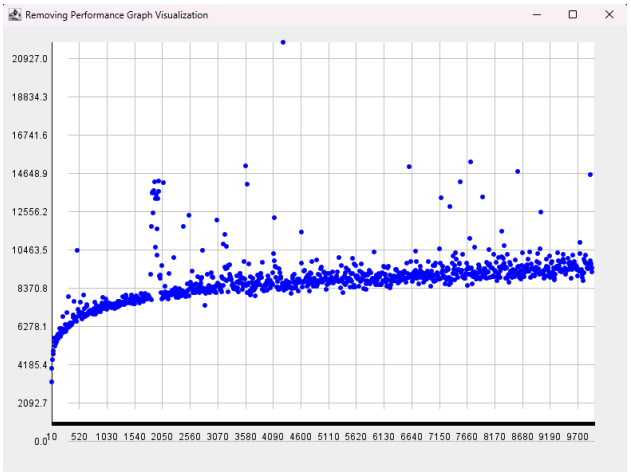
The graph below demonstrates the logarithmic function log(*n*) approximately. Due to the limitations of my test methods and Java's precision in measuring the exact cost time of operations, some measurements were incorrect. However, the overall trend still roughly resembles a logarithmic function. This indicates that my AVL tree is balanced, as expected.



*The search time cost**(ns)** is plotted against the size of the tree nodes*



*The add time cost**(ns)** is plotted against the size of the tree nodes*



The remove time cost(ns) is plotted against the size of the tree nodes