Performance Analysis

In Phase Three of this project, we will implement Graph data structures to determine the correlation between various products. In addition, we will provide an analysis of this algorithm and the AVL Tree algorithm implemented in phase two.

AVL Tree Analysis and Tests

As discussed in the previous Proof of Concept phase, the AVL Tree was implemented with production insertion, deletion, and search features. In addition, it has already been optimized for handling heavy load sizes as it is self-balancing and was tested in the previous phase on a large dataset.

Analysis

Time Complexity

Insertion:

- Average Case: O(log n)
- Worst Case: O(log n)

Search:

- Average Case: O(log n)
- Worst Case: O(log n)

Deletion:

- Average Case: O(log n)
- Worst Case: O(log n)

Space Efficiency

- Space Complexity: O(n)

The space complexity of the number of nodes is linear, as each node requires a fixed amount of space. Additionally, the space required for maintaining the height information and pointers is constant for each node.

Load Tests Results

Tests were implemented to load test the AVL Tree data structure, and the results were as follows.

Testing with 100 products: Insertion Time: 0 sec Search Time: 0 sec Deletion Time 8: 0 sec	Testing with 500 products: Insertion Time: 0.001 sec Search Time: 0 sec Deletion Time: 0 sec	Testing with 1000 products: Insertion Time: 0.003 sec Search Time: 0 sec Deletion Time: 0 sec
Testing with 5000 products: Insertion Time: 0.025 sec Search Time: 0 sec Deletion Time: 0 sec	Testing with 10000 products: Insertion Time: 0.056 sec Search Time: 0 sec Deletion Time: 0 sec	Testing with 20000 products: Insertion Time: 0.126 sec Search Time: 0 sec Deletion Time: 0 sec

Graph Analysis and Tests

The graph is a non-linear data structure that consists of vertices (nodes) and edges (links). It is useful in fields that benefit from finding relationships between multiple data points. In our implementation, we leverage an undirected graph to find correlations (as edges) between different products (nodes) in case we want to recommend products to customers based on recent or purchase history. The correlation between products is calculated based on attributes such as

product type, product creator, and time posted. The graph is represented using a dictionary where each product is a key, and the value is a list of correlated product IDs.

Analysis

The graph includes multiple key features for processing; these include adding products to the graph, finding correlations between products on various attributes, and calculating correlation scores between products to determine highly correlated products for better recommendations.

Time Complexity

- Insertion: O(1)
- Add Correlation: O(1)
- Correlate by attribute: $O(n^2)$
- Get Correlated Products: O(n log n)

Space Complexity

- Insertion: O(1)
- Add Correlation: O(1)
- Correlate by attribute: O(e)
- Get Correlated Products: O(n)

The Graph performs well with moderate-sized datasets, but its correlation methods' $O(n^2)$ complexity may cause performance issues with enormous datasets, as using nested loops in specific methods can degrade efficiency for larger datasets. To improve scalability, optimizations like utilizing more efficient data structures or parallel processing could be explored.

Load Tests Results

We run load tests on different the implemented graph data structures with different load sizes increasing progressively: 100, 500, 1000, 5000, and 10000. Test results were as follows.

Testing with 100 products: Correlation by Type: 0.001 sec Correlation by Creator: 0.001 sec Correlation by Insert Date: 0.002 sec Get Top 10 Correlated Products Time: 0 sec	Testing with 500 products: Correlation by Type: 0.039 sec Correlation by Creator Time: 0.038 sec Correlation by Insert Date: 0.087 sec Get Top 10 Correlated Products: 0 sec	Testing with 1000 products: Correlation by Type: 0.253 sec Correlation by Creator: 0.265 sec Correlation by Insert Date: 0.594 sec Get Top 10 Correlated Products: 0 sec
Testing with 5000 products: Correlation by Type: 28.95 sec Correlation by Creator: 30.444 sec Correlation by Insert Date: 64.917 sec Get Top 10 Correlated Products: 0.002 sec	Testing with 10000 products: Correlation by Type: 211.580 sec Correlation by Creator: 256.571 sec Correlation by Insert Date: 11888.558 sec Get Top 10 Correlated Products: 0.007 sec	

Conclusion

Various variations of graph algorithms provide different benefits. In our case, we want to provide the maximum performance, but as observed from the test results above, the chosen implementation gets slower as the load size increases. Some improvements would include

implementing caching within the graph and associate Heap data structures implementation to make adding and retrieving products faster.