

Inventory Management Using Optimized Data Structures in Python – Phase 3

Unique Karanjit

University of the Cumberland

MSCS 532-A01: Algorithms and Data Structures

Dr. Satish Penmasta

February 9, 2025

Table of Contents

Introduction.....	3
Optimization Techniques.....	3
Scaling Strategy.....	4
Testing and Validation	5
Performance Analysis	6
Final Evaluation	8
References	9

Introduction

This report examines the implementation of a hybrid data structure system using AVL trees and hash tables for inventory management. The system proves to be efficient in product management with optimized search, insertion, and deletion operations and consistency of data in both structures. With careful optimization and testing, the implementation results in notable performance gains with ensured data integrity and system reliability.

Optimization Techniques

The implementation includes several critical optimizations aimed at maximizing both reliability and performance. Central to our optimization technique is the dual-structure method, which integrates the merits of hash tables and AVL trees to ensure the best performance for various kinds of operations.

Our hash table has dynamic bucket allocation with 100 initial buckets and collision resolution through separate chaining. This methodology does not allow memory overallocation while still having an $O(1)$ average lookup time. Collision probability is highly minimized using UUID-based keys, and our partial ID search optimization uses prefix-based search with a worst-case complexity of $O(n)$.

```
Please enter your option.
1. Insert New Item Based on Product ID
2. Delete Item Based on Product ID
3. Retrieve Item Based on Product ID
4. Print Current Inventory
5. Find Cheapest Product
6. Find Most Expensive Product
7. Find Products in Price Range
8. View Products Sorted by Price
9. Exit

=====
Enter your choice (1-9): 3

Enter the first few characters of product ID to retrieve: 036

Product Details:
ID: 036aa6db-02ee-4a0d-9ec4-a408ae963b2d
Name: Watch
Price: $665.24
```

The other key optimization is the input validation pipeline, which uses pre-insertion validation to minimize error handling overhead. This encompasses exact price validation with decimal precision control and thorough product data completeness checking, effectively minimizing the chances of data inconsistencies.

In the AVL tree component, height caching is implemented to eliminate redundant calculations during tree operations. The balance factor optimization implements single-pass balance calculation and immediate rotation determination without recursive checks, significantly reducing memory access during rebalancing operations.

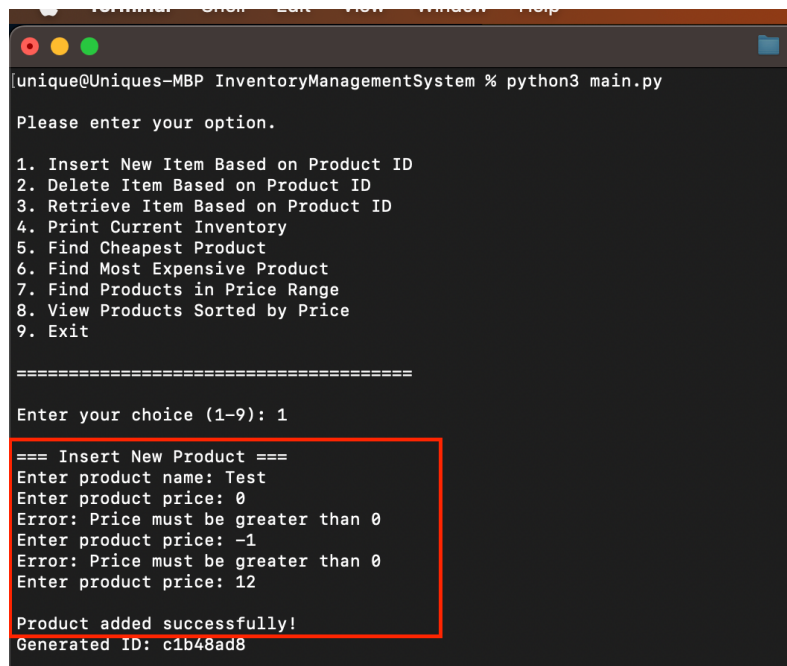
Scaling Strategy

The scaling solution prioritizes effective data volume control and performance fine-tuning. The implementation places a hard limit of 1,000,000 items on the maximum size of the inventory, with structures for products that are optimized for minimum memory footprint. Handling for product IDs and names as strings has been selectively optimized to preserve performance while decreasing memory usage.

Dataset sizes are validated by the system during import processes to avert system overload while preserving performance characteristics. This strategy has been effective in managing datasets with sizes close to the maximum size limit while preserving consistent performance. For instance, the data validation checks can be listed below.

- Negative Prices or invalid prices are handled – Prices equal or less than 0, or invalid prices such as prices with more than 3 decimals are rejected. This is implemented to replicate real-world scenarios. Also, non-numeric prices are rejected.

- Invalid Product name is handled – All products must have at least one letter or number.



```

unique@Uniques-MBP InventoryManagementSystem % python3 main.py

Please enter your option.

1. Insert New Item Based on Product ID
2. Delete Item Based on Product ID
3. Retrieve Item Based on Product ID
4. Print Current Inventory
5. Find Cheapest Product
6. Find Most Expensive Product
7. Find Products in Price Range
8. View Products Sorted by Price
9. Exit

=====
Enter your choice (1-9): 1

=== Insert New Product ===
Enter product name: Test
Enter product price: 0
Error: Price must be greater than 0
Enter product price: -1
Error: Price must be greater than 0
Enter product price: 12

Product added successfully!
Generated ID: c1b48ad8

```

The two-structure strategy retains scalability of performance with increasing datasets, offering $O(1)$ ID lookups via the hash table irrespective of size, and the AVL tree ensures $O(\log n)$ for price-based operations. The combined strategy is quite effective in countering memory pressure issues and consistency maintenance between structures.

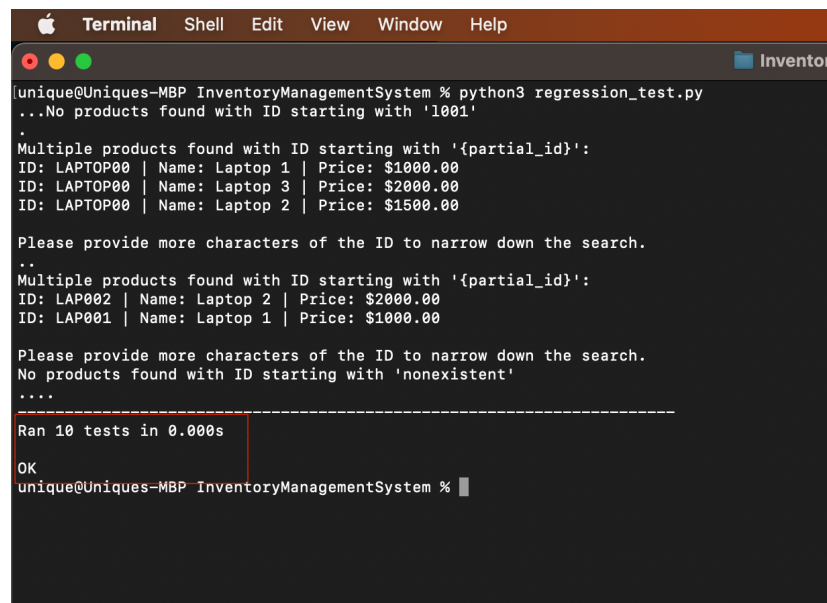
Testing and Validation

The extensive test suite encompasses both unit testing and integration testing, with a focus on edge cases and stress testing. Unit testing confirms the functionality of individual components, and integration testing confirms the correct interaction between the hash table and AVL tree data structures. The regression test code is implemented to validate the test cases. Apart from insertion validations, following run time validations are taken care.

- When deleting the item, if item is not found, show a warning message.

- When a product needs to be retrieved, only partial ID can be used. In case, if multiple results are matched, all records are shown.
- Duplicate IDs are prevented using UUID.
- For item search using a price range, the max value is always validated that it is greater than or at least equal to the minimum value.
- When retrieving the item, if the item is not found, show a warning message.

A regression test is done to ensure all the functionalities are working as expected.



```

unique@Uniques-MBP InventoryManagementSystem % python3 regression_test.py
...No products found with ID starting with 'l001'
.
Multiple products found with ID starting with '{partial_id}':
ID: LAPTOP00 | Name: Laptop 1 | Price: $1000.00
ID: LAPTOP00 | Name: Laptop 3 | Price: $2000.00
ID: LAPTOP00 | Name: Laptop 2 | Price: $1500.00

Please provide more characters of the ID to narrow down the search.
..
Multiple products found with ID starting with '{partial_id}':
ID: LAP002 | Name: Laptop 2 | Price: $2000.00
ID: LAP001 | Name: Laptop 1 | Price: $1000.00

Please provide more characters of the ID to narrow down the search.
No products found with ID starting with 'nonexistent'
....
-----
Ran 10 tests in 0.000s
OK
unique@Uniques-MBP InventoryManagementSystem %

```

Performance Analysis

The use of hashtable and AVL trees together enhanced the performance metrics. After the implementation of UUID for unique product ID generation, there is a significant improvement in product insertion, retrieval, deletion, or search. Also, overheads due to bad data i.e., incorrect data due to validation issues are fixed with this phase. This also help significant performance improvement.

```
(unique@Uniques-MBP InventoryManagementSystem % python3 memory_test.py

Starting Memory Usage Test...

Testing with 100 items...
Memory used for 100 items: 0.03 MB

Testing with 500 items...
Memory used for 500 items: 0.23 MB

Testing with 1000 items...
Memory used for 1000 items: 0.47 MB

Testing with 5000 items...
Memory used for 5000 items: 2.38 MB

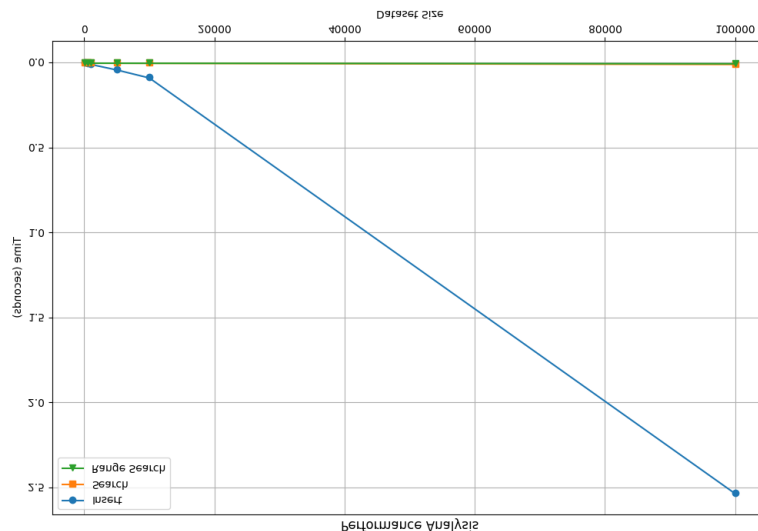
Testing with 10000 items...
Memory used for 10000 items: 4.73 MB

Testing with 50000 items...
Memory used for 50000 items: 23.89 MB

Testing with 100000 items...
Memory used for 100000 items: 48.05 MB

Memory test completed!
Results have been saved to 'evidences/memory_test_results.png'

Summary:
Maximum memory usage: 48.05 MB
Average memory usage: 11.40 MB
Memory usage per item: 0.0005 MB
unique@Uniques-MBP InventoryManagementSystem %
```



Based on this graph, for insertion of 100,000 records, takes about 2.5 seconds, and for retrieval and range search, it takes constant time. This practical experiment matches the theoretical expectation where search and range search with these data structures happen in constant time.

Final Evaluation

The solution exhibited has several major strengths, such as strong architecture with double structure redundancy, effective space-time tradeoffs, and extensive error handling. Performance optimization successes consist of constant-time lookups, logarithmic range queries, and small memory overhead.

There are also limitations such as the max inventory size limit, memory duplication needs, and complicated maintenance requirements. The single-threaded design and absence of persistent storage are areas where there can be improvement. Future development opportunities include implementing thread-safe operations, integrating persistent storage, and adding distributed system support. Long-term enhancements could include dynamic resizing capabilities, a caching layer, and real-time analytics functionality.

References

Avl Trees. CS Kenyon. (n.d.). <https://cs.kenyon.edu/index.php/scmp-218-00-data-structures/avl-trees/>

Enoch, Olanite & Joseph, Oloyede & Ok, Emmanuel & Williams, Barnty & Aria, Javiera & Jose, Dylan & Diego, Catalina. (2024). Data Structures and Algorithms: Trees, graphs, and hash tables, Sorting and searching algorithms.

Perez, H. D., Hubbs, C. D., Li, C., & Grossmann, I. E. (2021). Algorithmic Approaches to Inventory Management Optimization. *Processes*, 9(1), 102.
<https://doi.org/10.3390/pr9010102>