Project Phase 3 – Optimization, Scaling, and Final Evaluation

Shrisan Kapali

Student Id: 005032249

MSCS 532 – Algorithms and Data Structures

Satish Penmatsa

February 09, 2025

GitHub Link

https://github.com/ShrisanKapali-Cumberlands/MSCS532_Project_Phase_3

Optimization of Data Structures

The Python dynamic inventory management program was optimized using various techniques to achieve faster execution times and optimal memory usage. It was designed to handle large datasets in cases where the application may be scaled. By analyzing the trade-offs of different data structures, the program utilizes dictionaries, tuples, and class-based objects for storing, retrieving, updating, and deleting information.

During the proof-of-concept phase, the time complexity of using dictionaries was examined, with an average time complexity of O (1) for insert, retrieve, and delete operations. As a result, dictionaries were favored over traditional arrays for storing products and categories due to this efficiency. Tuples were chosen over linked lists to maintain the price history of a product. This preference is because tuples are immutable and store data in chronological order upon insertion. Additionally, tuples help ensure data integrity, as the price history should never change once recorded.

The chosen data structures for CRUD operations on categories and products optimize time complexity while ensuring security. However, when considering large data sets, to achieve higher performance, caching was implemented.

Caching is one of the optimization techniques where the data is stored in a faster memory hierarchy, such as RAM (“What is caching and how it works,” n.d.). It increases data retrieval by directly accessing the data stored in a higher memory hierarchy without accessing the lower memory storage level (“What is caching and how it works,” n.d.). The Least Recently Used (LRU) cache was implemented in the Python program. In an LRU cache, the data is organized in

the order they are used, such that the frequently used data are stored in the LRU cache, replacing the least used data when the cache is full (Mitra, 2023). The time complexity to store and retrieve data from a LRU cache is O (1) (Mitra, 2023).

To scale for faster performance, the inventory management program also uses manual memoization to store frequently queried results. It used dictionaries to store the results of the queried search. The next time the same condition is queried, it fetches the result directly from the manual cache without performing the search computation, reducing the processing overhead. The retrieval happens in a constant time of O (1), delivering higher performance. However, as integers were used to store the product and category ID, custom keys had to be implemented to store these values in the manual memoization cache.

The implemented LRU caching and manual memoization utilize hardware resources, resulting in increased memory usage. These caches must be refreshed whenever a new record is added or an existing record is updated to ensure the values remain current. If a new category or product is added to the inventory program, it becomes necessary to clear the existing cache. This leads to the requirement of rebuilding the cache, which can pose challenges in situations where the application needs to be continuously updated.

Testing and Validation

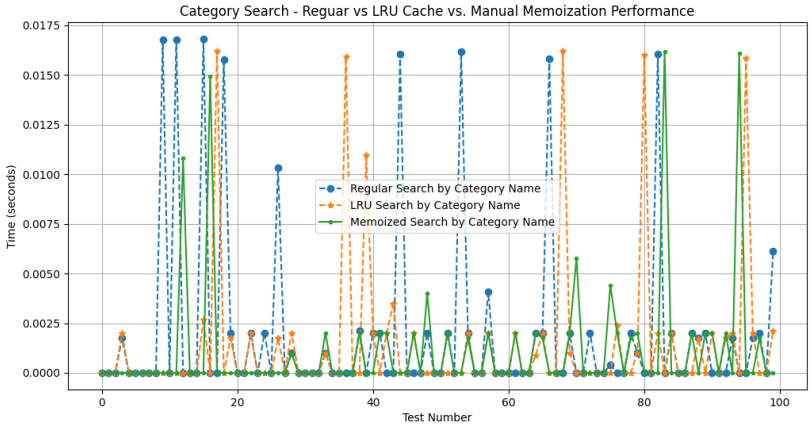
Stress testing was performed to evaluate the performance of data structures used in the inventory management program. Twenty thousand categories and one hundred thousand products were inserted into the inventory. The execution time to insert these new categories was 0.021 seconds, and products were less than 0.4 seconds. This execution time shows that using a dictionary is optimal for inserting new records. Every time a new record was inserted, the manual cache was cleared.

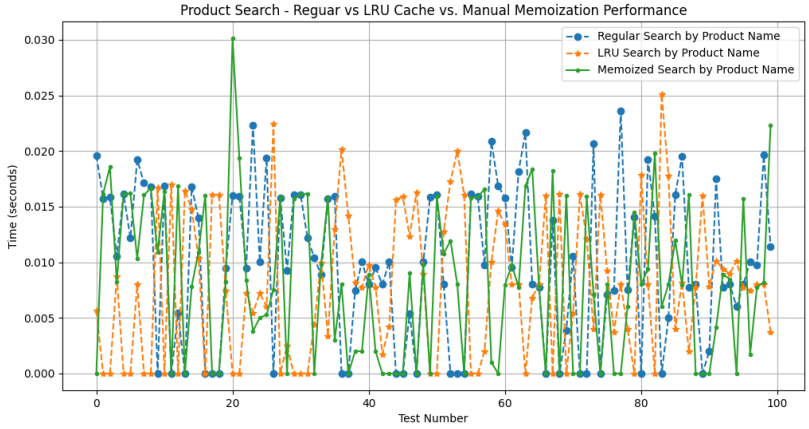
```
PS S:\University of Cumberlands\Spring 2025\MSCS532 - Algorithm and Data Structures\Assignments\MSCS532-Project3> py Project_Phase_3.py
*******************************
*** Initializing Inventory ***
*******************************

*******************************
Adding new category
*******************************
20000 new categories are added
Execution time - 0.02088451385498047 seconds
Current inventory category size 20000
*******************************

*******************************
Adding new products
*******************************
100000 new products are added
Execution time - 0.3034913539886474 seconds
Current inventory products size 100000
*******************************
```

The execution time it took to search the categories and products using regular dictionary search without using any cache, LRU cache, and manual dictionary memoization was evaluated. On average, performing these searches among the vast data, twenty thousand categories, and one hundred thousand products took less than 0.03 seconds. This shows that the dictionary data structure is ideal for retrieving values using the key in a constant time complexity of O (1).





As for the results obtained while plotting the execution time to search for the product and category by name, the search took almost the same time when the cache was implemented and not implemented. The search is fast enough because the dictionary already has an O (1) constant retrieval time. However, the processor must perform extra work to get the results. In our test case, the search name was randomized; however, if the same search parameters are used repeatedly, the cache may slightly outperform the regular dictionary search. In some scenarios, the cache search took more time, as it was to store the results of the search on the cache so that the data could be fetched faster next time.

The test case demonstrates that the data structures employed in developing the inventory management system offer optimal performance and are memory efficient, regardless of the data size. The total time required to retrieve data was under 0.03 seconds, which illustrates the effectiveness of these data structures in managing CRUD operations.

Final evaluation and potential areas for future development

In conclusion, the data structures utilized in developing the inventory management system provide an optimal time complexity of O (1) for basic CRUD operations concerning inventory categories and products. LRU Caching has been implemented to enhance performance. While testing among twenty thousand categories and a hundred thousand products, the time complexity to search the record using factors such as name, price range, and ID was optimal, below 0.3 seconds.

The Python inventory program currently uses only hardware resources for storage and cache. However, practical, real-life applications must be web-based so users can access them without running the program on their local machines. The fundamentals of the inventory

program analyze many data structures and use the best data structures like dictionaries, tuples, and lists to achieve optimal time and space complexity.

The next step to scale this application is to integrate it with database management systems like MySQL and MongoDB and have a user interface such that users can interact with the system. Angular, Vue.js, and React are some of the most commonly used frontend frameworks that can be used to develop the user interface (Mendes & Rodrigues, 2024). In addition to the Python program, REST APIs will need to be implemented so that the frontend application can perform the required functions. Some basic examples of APIs are the POST endpoint, which creates categories and products; the PUT endpoint, which updates them; and the DELETE endpoint, which deletes a category and product.

Redis, Apache Ignite, and Hazelcast are some of the commonly used in-memory caching frameworks that can improve the performance of web-based applications ("Top 10 in-memory caching frameworks for web application development in 2023," 2023). Multithreading can effectively improve performance if functions need to be performed in batches. As multithreading allows the execution of parallel operations, if the operations are independent of one another, it can enhance the performance by a significant factor ("Benefits of multithreading," n.d.).

Finally, to maintain the code standards and check if all the functions are performing well, test cases must be written and updated whenever code changes occur. Regression testing, stress testing, and end-end testing must be done to ensure that the program is functioning well and delivers optimal performance.

References

Benefits of multithreading - javatpoint. www.javatpoint.com. (n.d.). https://www.javatpoint.com/benefits-of-multithreading

Mendes, A., & Rodrigues, O. (2024, November 4). *Top 10 best front end frameworks in 2025*. Imaginary Cloud: Software Development for Digital Acceleration. https://www.imaginarycloud.com/blog/best-frontend-frameworks

Mitra, A. (2023, November 9). *Design an LRU cache*. Medium. https://medium.com/@abhishek.amjeet/design-an-lru-cache-447f49df7bbf

Top 10 in-memory caching frameworks for web application development in 2023. FROMDEV. (2023, October 9). https://www.fromdev.com/2023/05/top-10-in-memory-caching-frameworks-for-web-application-development-in-2023.html

What is caching and how it works | AWS. (n.d.). https://aws.amazon.com/caching/