Dylan Stirling

Professor Sandifer

CS 499

1 Feb 2025

Algorithms and Data Structures – Enhancement Narrative

-Artifact Overview

The artifact I selected for enhancement is the `ContactService.java` class, a core component of a contact management system. This class is responsible for adding, updating, deleting, searching, and sorting contacts efficiently. The original version of this artifact was created during my coursework in CS 320: Software Testing, Automation, and Quality Assurance, where the focus was primarily on implementing core functionalities and ensuring testability.

In refining this artifact for my ePortfolio, I have made significant improvements to its algorithms and data structures to enhance its efficiency, scalability, and maintainability. The enhancements focus on optimizing sorting, searching, and data storage, making the class more adept at handling large datasets.

-Justification for Inclusion in ePortfolio

I selected this artifact for my ePortfolio because it demonstrates my ability to apply and optimize algorithms and data structures in real-world applications. The key components of the artifact that showcase my skills include sorting optimization, efficient searching, data structure refinements, and scalability enhancements.

Initially, the implementation sorted the contact list every time a sort request was made, which was inefficient. I improved this by using a TreeMap, ensuring that contacts are automatically sorted upon insertion. This change eliminated the need for repeated sorting operations and significantly improved the performance of the system, especially for larger datasets.

The previous implementation also utilized a linear search (O(n)) when retrieving contacts by name. I replaced this with logarithmic-time searches (O(log n)) through the TreeMap's built-in key-based lookup, which greatly enhanced search performance. Additionally, I refined the data structures by replacing an unsorted HashMap with a LinkedHashMap to maintain insertion order while also incorporating a TreeMap for storing pre-sorted contacts. These adjustments reduced redundant computations and improved access times.

Ultimately, the optimized structure allowed for better handling of large datasets, making the ContactService class more efficient and scalable for real-world applications. These improvements align directly with principles of algorithmic efficiency and showcase my ability to analyze and optimize data structures for enhanced performance. By implementing these optimizations, I was able to enhance the functionality of the ContactService class while maintaining its scalability, which is essential for developing robust and efficient software solutions.

-Meeting Course Outcomes and Updates

One of my primary goals in Module One was to enhance my understanding and application of data structures and algorithms to develop efficient software solutions. This artifact demonstrates that goal, as I successfully applied sorting and searching algorithms to optimize performance. Specifically, I replaced inefficient O(n) operations with more efficient O(log n) and O(1) alternatives, which significantly improved the overall performance of the ContactService class. Additionally, I improved my selection of data structures to ensure faster insertions, lookups, and deletions, making the system more efficient and scalable.

At this stage, I have partially met my course outcomes related to algorithmic optimization, but I recognize there are still areas for refinement. While the optimizations made so far have greatly enhanced performance, I see potential for further improvement. Moving forward, I plan to explore more advanced indexing strategies, such as hash-based indexing or trie structures, to enhance search efficiency and optimize system performance. By continuing to refine my understanding and implementation of algorithms and data structures, I aim to build even more efficient software solutions in the future.

-Reflection on the Enhancement Process

Throughout the process of improving this artifact, I gained valuable insights into algorithmic efficiency and data structure selection. One of my key takeaways was the importance of balancing readability with performance. Initially, I considered implementing a more complex indexing structure, but I quickly realized that maintainability is just as crucial as performance. The solution I ultimately chose strikes a balance between efficiency and code clarity, ensuring that the system is both effective and easy for future developers to understand.

Another important lesson I learned was the trade-offs involved in selecting data structures. While a TreeMap provides sorted order, it comes with slightly higher overhead compared to a HashMap. However, because the primary use case involved frequent searches and sorting, the benefits of using a TreeMap far outweighed the costs. This experience deepened my understanding of how different data structures perform under various circumstances.

Additionally, I faced challenges related to testing and debugging the changes made to the internal data structures. Switching from an unsorted HashMap to a sorted TreeMap introduced unexpected behaviors that required thorough testing and debugging to resolve. This experience taught me the importance of extensive testing when modifying core components of an application.

Finally, this enhancement solidified my understanding of how algorithmic choices affect real-world performance. In professional software development, choosing the right data structures is critical, as it can significantly impact an application’s scalability and efficiency. This realization has reinforced the value of optimizing algorithms and selecting the most suitable data structures for the task at hand.

-Conclusion

Enhancing this artifact has strengthened my skills in algorithms and data structures while reinforcing the importance of efficiency, scalability, and maintainability. This process deepened my understanding of sorting, searching, and optimized data storage, which are fundamental to software engineering. Moving forward, I plan to explore even more efficient search techniques and consider implementing multi-threading for concurrent operations. These improvements will further refine the artifact and elevate my proficiency in algorithmic problem-solving.