Enhancement Two: Algorithms and Data Structures

Caleb Irwin

CS-499: Milestone 4-2

Professor Gene Bryant

March 30, 2025

Enhancement Two: Algorithms and Data Structures

Briefly describe the artifact. What is it? When was it created?

For this enhancement, I selected the ABCU Advising Program from CS300: Data Structures and Algorithms: Analysis and Design. This artifact was the final project of that course and was created in February 2024. The original artifact needed to import a list of courses into a data structure (array, hash table, or binary search tree) and then be able to display a list of courses in alphabetical order or search for a specific course.

Justify the inclusion of the artifact in your ePortfolio. Why did you select this item? What specific components of the artifact showcase your skills and abilities in software development? How was the artifact improved?

I selected this artifact for my enhancement because it offered a clear opportunity to demonstrate and improve my understanding and application of data structures and algorithmic principles. The original artifact utilized a binary search tree (BST) and allowed users to load, display, and search a list of courses. One thing I noticed during the original implementation was that the tree could easily become unbalanced in a real-world scenario. The original BST implementation performed adequately for a balanced dataset but lacked robustness for cases where the data was uneven or heavily skewed—such as when the imported data wasn't organized in a specific way or if users were allowed to remove courses after a dataset was loaded. This could lead to situations where search times approached the worst-case scenario. If the program were working with a larger dataset, it wouldn't exhibit the responsiveness the original artifact was intended to support, which is one of the reasons I had originally chosen to implement a BST over a linked list or hash table.

To improve the original artifact, I opted to implement an AVL tree, which is self-balancing and is designed to ensure efficient search, insertion, and deletion operations in worst-case scenarios. To implement the AVL tree, I added a rebalance() method (*Figure 1*) and a few supporting helper functions (leftRotate(), rightRotate(), height(), and updateHeight()), and then I refactored several existing methods. Additionally, I refactored the program to utilize smart pointers in order to improve memory safety and prevent memory leaks, and I implemented structured exception handling to manage errors rather than just displaying messages using cout. Finally, I enhanced the project's documentation for clarity and generated class-level documentation using Doxygen.

```
unique_ptr<Node> BinarySearchTree::rebalance(unique_ptr<Node> node) {
    if (!node) {
        return node;
    }

    updateHeight(node);
    int balance = height(node->left) - height(node->right);

    // Left Heavy
    if (balance > 1) {
        // Left-Left
        if (height(node->left->left) >= height(node->left->right)) {
            return rightRotate(move(node));
        }
        // Left-Right
        else {
            node->left = leftRotate(move(node->left));
            return rightRotate(move(node));
        }
    }
    // Right Heavy
    else if (balance < -1) {
        // Right-Right
        if (height(node->right->right) >= height(node->right->left)) {
            return leftRotate(move(node));
        }
        // Right-Left
        else {
            node->right = rightRotate(move(node->right));
            return leftRotate(move(node));
        }
    }

    return node;
}
```

Figure 1 – reblance() Method

These improvements showcase my ability to write efficient, maintainable, and scalable code. They also highlight my understanding of algorithmic trade-offs and my ability to improve the performance characteristics of a data structure through thoughtful design and refactoring.

Did you meet the course outcomes you planned to meet with this enhancement in Module One? Do you have any updates to your outcome-coverage plans?

Yes, I met the course outcomes I originally planned to address with this enhancement. Specifically:

[CS499-02] *Design, develop, and deliver professional-quality oral, written, and visual communications that are coherent, technically sound, and appropriately adapted to specific audiences and contexts.*

I've improved the quality of the codebase, in-code comments, and class documentation to better explain the program's behavior and demonstrate this course outcome.

[CS499-03] *Design and evaluate computing solutions that solve a given problem using algorithmic principles and computer science practices and standards appropriate to its solution while managing the trade-offs involved in design choices.*

I applied algorithmic principles and techniques in the design and implementation of this enhancement using an AVL tree. This structure was chosen for its performance in worst-case scenarios and manages the trade-offs between loading and searching the dataset.

[CS499-04] *Demonstrate an ability to use well-founded and innovative techniques, skills, and tools in computing practices for the purpose of implementing computer solutions that deliver value and accomplish industry-specific goals.*

In this enhancement, I utilized well-founded techniques and modern C++ features—such as the use of smart pointers and modular design—that are consistent with industry best practices.

[CS499-05] *Develop a security mindset that anticipates adversarial exploits in software architecture and designs to expose potential vulnerabilities, mitigate design flaws, and ensure privacy and enhanced security of data and resources.*

I demonstrated a security mindset by anticipating potential issues related to data integrity, system stability, and program reliability. I chose to replace the original implementation of raw pointers with smart pointers, which significantly reduced the risk of memory leaks, dangling pointers, and memory management errors. Additionally, I implemented structured exceptions to improve program resilience by catching and gracefully handling errors.

Reflect on the process of enhancing and modifying the artifact. What did you learn as you were creating it and improving it? What challenges did you face?

The enhancement process helped to solidify my understanding of self-balancing trees. While I had previously implemented a basic BST, integrating AVL logic required deeper knowledge of rotation strategies and balance factor calculation. When I created the original artifact, I had no knowledge of smart pointers. I hadn't used pointers in C++ since then, and while implementing

them in this enhancement, I learned a lot about the different kinds of smart pointers and their functionality; while taking advantage of their memory management benefits.

One of the biggest challenges I faced while implementing trees was visualizing the recursive calls to ensure the necessary changes were properly applied to create the desired functionality. When debugging unexpected functionality or output discrepancies, I relied heavily on console outputs and test cases to confirm the correctness of my artifact.