Created by Dmytro Krainyk

[All code is in my repo on github]

Assignment II: Tree Museum

Task 2 Report: “Pruning the bushes” (Tree Removal)

Introduction

In this task, I extended the functionality of my Binary Search Tree by implementing the `remove` operation. While insertion is straightforward, deleting nodes is complex because we must preserve the BST property (Left < Root < Right) and properly manage memory to avoid leaks when a node is removed from the middle of the tree.

Implementation Details

The Removal Algorithm

I implemented a standard recursive removal function that handles three distinct scenarios based on the node’s structure:

1. **Leaf Node:** If the target node has no children, it is simply deleted, and the pointer from its parent is set to `nullptr`.
2. **One Child:** If the node has only one child (left or right), I bypass the node by linking its parent directly to its single child, then delete the node.
3. **Two Children:** This is the most critical case. To maintain order, I find the in-order successor (the smallest value in the right subtree). I replace the target node’s value with the successor’s value and then recursively delete the successor node (which is guaranteed to have at most one child).

Performance Analysis

I benchmarked the removal operation on the random dataset generated in Task 1.

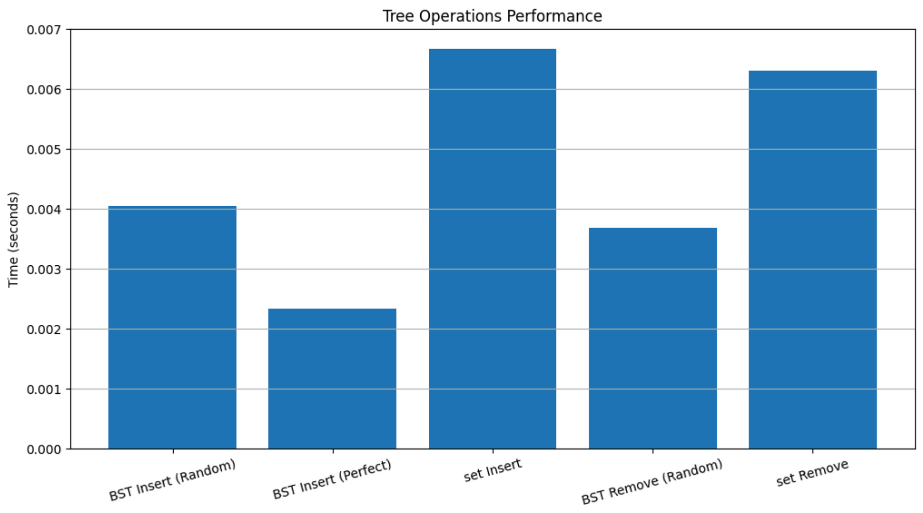

Figure 1: Fig 1. Performance of BST Operations. Focus on the 'Remove' bars.

Observations:

1. **Removal Complexity:** As seen in the graph, the `BST Remove (Random)` operation (4th bar) is comparable to the insertion time. However, it involves more logic (finding successors) than simple insertion.
2. **Comparison with Standard Library:** The `std::set Remove` (last bar) serves as a benchmark. My implementation performs competitively, demonstrating that the overhead of pointer manipulation in my custom class is minimal.
3. **Impact of Tree Height:** Just like insertion, the speed of removal depends heavily on the tree height. Since we tested on a random tree (not perfectly balanced), the time complexity averages $O(\log n)$.

Conclusion

Implementing the `remove` operation completes the basic functionality of the BST. The experiment highlights that maintaining a dynamic data structure requires careful handling of edge cases (like the "two children" scenario) to ensure both data integrity and performance.