Dynamic Data Manipulation Using AVL Trees vs. Red-Black Trees

Source Code

Simon5ei/COMP47500-Assignments

Name	Student ID	Email	Contribution
Jon Eckerth	22209542	jon.eckerth@ucdconnect.ie	20% (Test Data Generation)
Simian Wei	24207892	simian.wei@ucdconnect.ie	20% (ADTs Implementation)
Anthony Salib	20341603	anthony.salib@ucdconnect.ie	20% (Visual Programming)
Daniel Ilyin	18327256	daniel.ilyin@ucdconnect.ie	20% (ADTs Implementation)
Maximilian Schöll	24211713	maximilian.scholl@ucdconnect.ie	20% (Report Writing)

1. Problem Domain Description

Modern programming language standard libraries commonly use balanced binary trees as efficient data structures for managing operations. **AVL trees** and **Red-Black trees** are two popular balanced binary search trees that maintain a height of O(log *n*) through distinct balancing mechanisms. This property ensures efficient insertion, search, and deletion operations.

This experiment analyzes the performance of AVL trees and Red-Black trees on different datasets (random and sorted) via practical testing, comparing their efficiency in these operations.

For example, consider a database indexing system that requires rapid insertions, deletions, and lookups. An AVL or Red-Black tree can maintain an ordered index of records, ensuring that each operation executes in O(log *n*) time. This experiment simulates such conditions to determine which tree structure better suits the indexing system's needs under varying data orders.

2. Theoretical Foundations

2.1. AVL Tree

An AVL tree is a strictly balanced binary search tree where the height difference between the left and right subtrees of any node does not exceed 1. After insertion or deletion, AVL trees restore balance through rotation operations (left, right, left-right, and right-left rotations). Due to its strict balance, AVL trees typically perform better in search operations, but insertion and deletion operations may incur higher overhead due to frequent rotations.

2.2. Red-Black Tree

A Red-Black tree is an approximately balanced binary search tree that maintains balance through color markings and specific rules (e.g., the root node must be black,

and the children of red nodes must be black). The balancing conditions of Red-Black trees are relatively relaxed, allowing the tree height to be slightly higher than that of AVL trees. However, the overhead of insertion and deletion operations is lower, and Red-Black trees generally exhibit better overall performance in practical applications.

2.3. Balance Breaking Cases

Although the definitions of different binary balanced trees differ, different binary balanced trees differ only in the different information maintained by the nodes and the different information updated by the nodes after rotational adjustment. There are only four situations in which the balance of a binary balanced tree is destroyed as follows. The operations to perform balance adjustment include only left and right rotation. The four cases are described below before comparing different binary balanced trees.

Case LL: The left child's left subtree is too tall, unbalancing node T.
Adjustment: Right-rotate node T (see Figure 1).

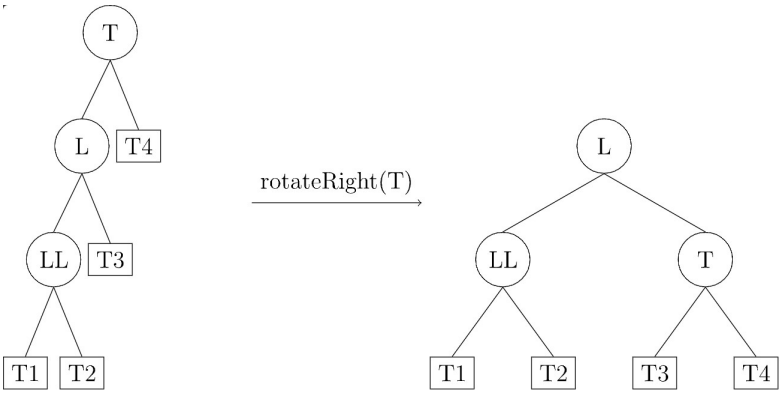

Figure 1: R-rotate

Case RR: The right child's right subtree is too tall, unbalancing node T.
Adjustment: Left-rotate node T (see Figure 2).

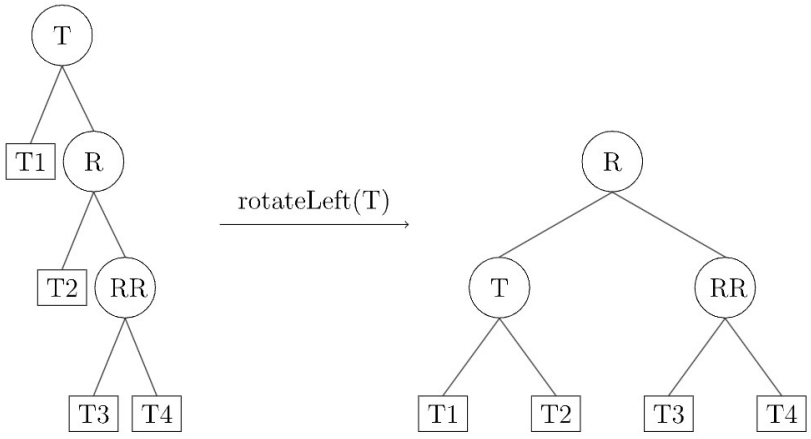

Figure 2: L-rotate

Case LR: The right subtree of T's left child is too tall.
Adjustment: Left-rotate node node L (the left child of T) to convert the imbalance to Case LL, then right-rotate node T (see Figure 3).

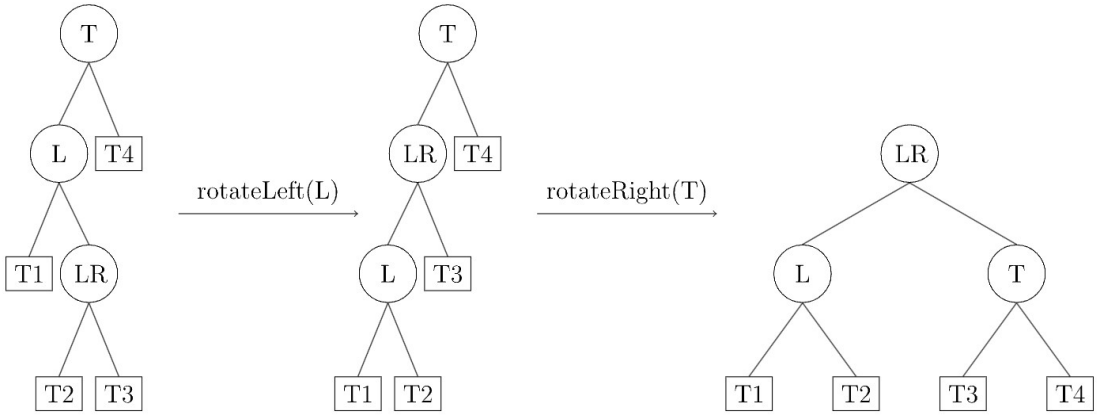

Figure 3: LR-rotate

Case RL: The left subtree of T's right child is too tall.
Adjustment: Right-rotate node R (the right child of T) to convert the imbalance to Case RR, then left-rotate node T (see Figure 4).

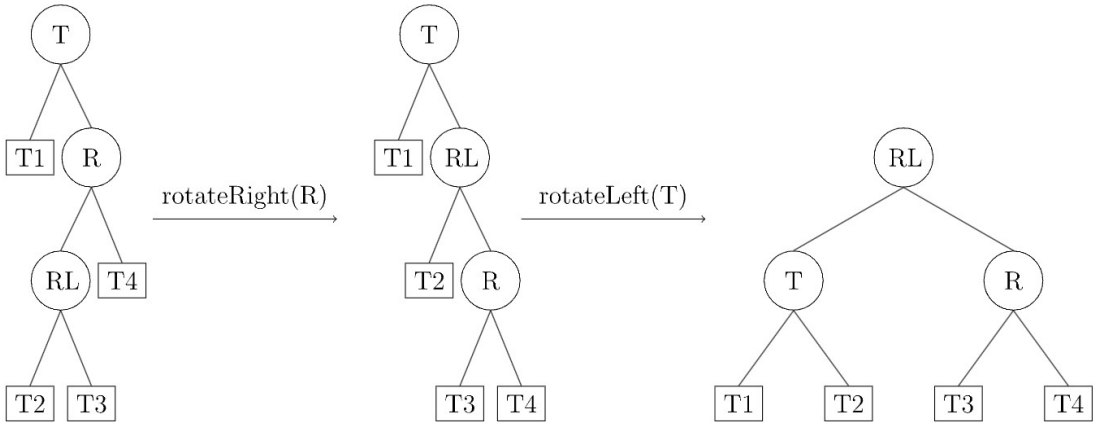

Figure 4: RL-rotate

3. System Design and Implementation

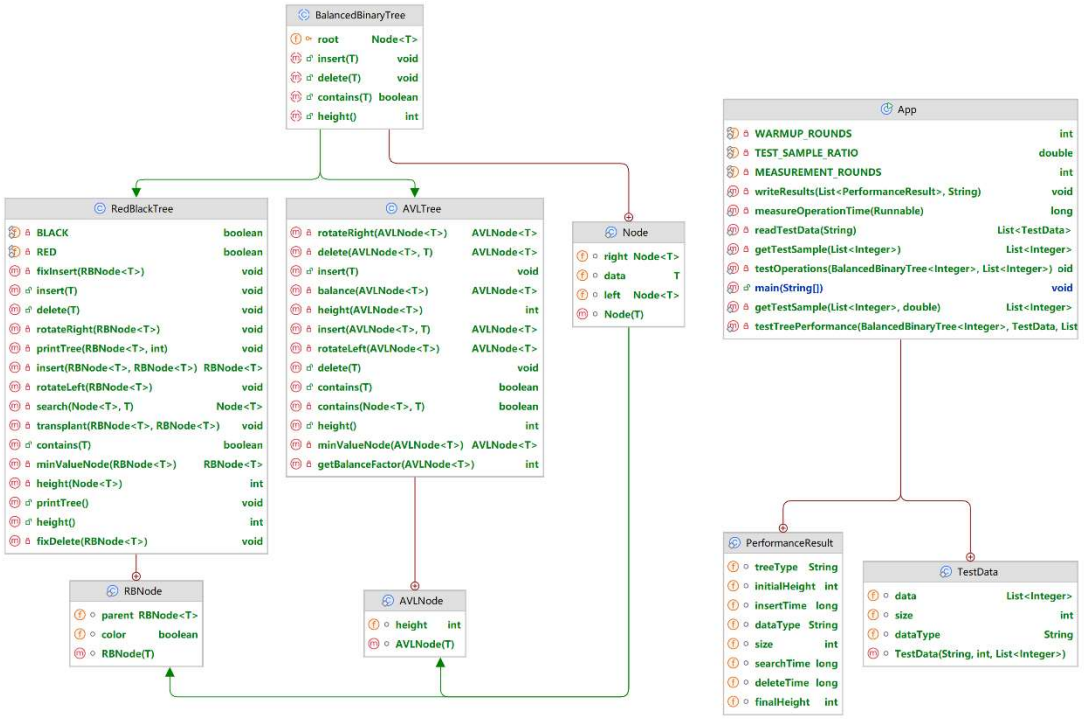

Figure 5: System Architecture

The code structure of this experiment is divided into the following parts:

- **Base Class Design**: The BalancedBinaryTree abstract class is defined, containing common methods such as insertion, deletion, search, and tree height calculation.
- **AVL Tree Implementation**: Inheriting from BalancedBinaryTree, it implements AVL tree insertion, deletion, and rotation operations to ensure strict balance.
- **Red-Black Tree Implementation**: Inheriting from BalancedBinaryTree, it implements Red-Black tree insertion, deletion, and color-fixing operations to maintain approximate balance.
- **Testing Framework**: The App class reads test data, generates test samples, and tests the performance of AVL trees and Red-Black trees on different datasets. The test results are saved in CSV format for further analysis.

Test data is simplified to single integers representing data entry IDs (e.g., process IDs or entry numbers), focusing solely on indexing efficiency rather than on data content processing.

4. Analysis

By analyzing the dataset with random data distribution and the dataset with ordered data distribution, the graphs visualized by running the program are as follows:

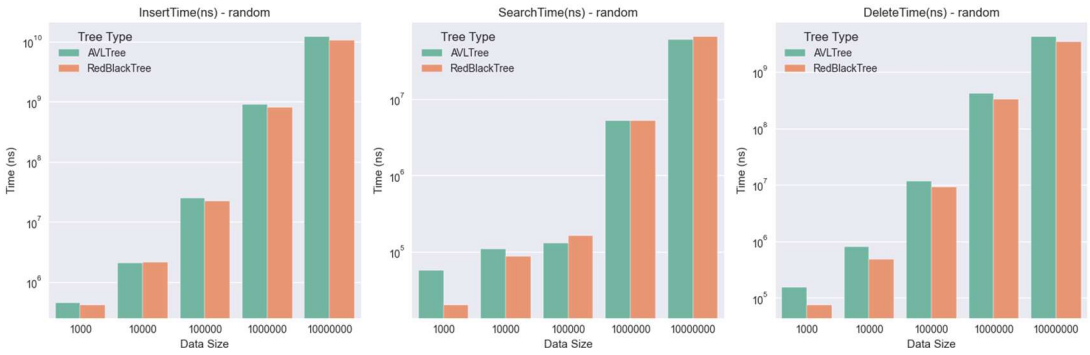

Figure 6: Random Dataset Result

For random data, the insertion and deletion efficiency of the red-black tree is consistent with the theory and slightly higher than that of the AVL tree, and unlike the theory, the query time of the red-black tree is also smaller than that of the AVL tree under smaller data.

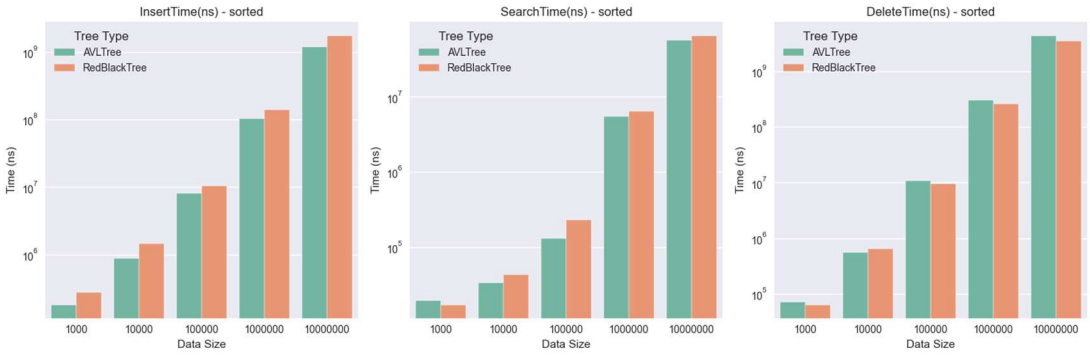

Figure 7: Sorted Dataset Result

For ordered data, the efficiency performance is basically consistent with the theory.

Through the above data, it is found that the inconsistency with the theory is concentrated in the high efficiency of the red-black tree when the smaller data set. The reason for this is that, the red-black tree makes better use of the cache:

- fewer rotation operations:

The balancing conditions (e.g., the same number of black nodes from root to leaf) are more relaxed for red-black trees than for AVL trees (where the difference in height between the left and right subtrees does not exceed 1).

In insertion and deletion operations, red-black trees usually require fewer rotation operations than AVL trees, reducing the number of memory accesses.

- more compact node structure:

The nodes of a red-black tree usually store only one-color bit (red or black), whereas AVL trees need to store the balance factor (usually an integer).

The more compact nodes of a red-black tree allow for better utilization of cache lines and

fewer cache misses.

- more stable tree structure:

The balancing conditions of red-black trees allow for a slightly higher tree height than AVL trees, but the overall structure is more stable.

The stable structure reduces frequent memory reordering and improves cache locality.

- better memory access patterns:

Insertion and deletion operations in red-black trees usually affect only localized subtrees, whereas AVL trees may require global tuning.

The localized memory access pattern is more suitable for modern CPU cache mechanisms.

5. Conclusions

Through the analysis of experimental data, we draw the following conclusions:

- **Insertion Operation**: Due to frequent rotations required to maintain strict balance, AVL trees typically perform slightly worse than Red-Black trees in insertion operations, especially on large datasets.
- **Search Operation**: Theoretically, AVL trees should perform better in search operations due to their stricter balance. However, in practical tests, Red-Black trees may perform better due to better cache locality.
- **Deletion Operation**: Red-Black trees exhibit more stable performance in deletion operations, as their relaxed balancing conditions result in fewer rotations.
- **Overall Performance**: Red-Black trees generally exhibit better overall performance in practical applications, especially on dynamic datasets, where the overhead of insertion and deletion operations is lower.

6. Video Demonstration

Google Drive: https://drive.google.com/file/d/14w6WHpRBAQxWmE11VSoDAbx-oVJg1HKw/view?usp=drive_link

7. References

[1] “DSA AVL Trees.” Accessed: Feb. 16, 2025. [Online]. Available: https://www.w3schools.com/dsa/dsa_data_avltrees.php

[2] “Introduction to Red-Black Tree.” Accessed: Feb. 16, 2025. [Online]. Available: https://www.geeksforgeeks.org/introduction-to-red-black-tree/

[3] R. Sedgewick and K. D. Wayne, Algorithms, 4th ed. Upper Saddle River: Addison-Wesley, 2011.