زياد اسلام حسني 22010778

يوسف احمد عبدالغني 22011369
عبدالرحمن خالد 22010877
يوسف عوض الشربيني 22011390
عبدالرحمن السيد سعد 22010869

Implementation of Self-Balanced BSTs (AVL & Red-Black Trees)

1. Implementation Details

1.1 AVL Tree

Structure

Each node stores:

`value` (generic type `T`).

`left` and `right` child pointers.

`height` (to track subtree height for balancing).

Key Operations

Insertion (`insert(T value)`)

Recursively finds the correct position.

Updates heights and checks balance factor (`balance = right.height - left.height`).

Performs rotations if unbalanced:

- **Left-Left (LL)**: Single right rotation.
- **Right-Right (RR)**: Single left rotation.
- **Left-Right (LR)**: Left rotation on child, then right rotation on parent.
- **Right-Left (RL)**: Right rotation on child, then left rotation on parent.

Deletion (`delete(T value)`)

Recursively removes the node.

Replaces with predecessor (rightmost in left subtree) if needed.

Rebalances the tree using the same rotation logic as insertion.

Search (`search(T value)`)

Standard BST search (`O(log n)`).

Balancing (`balanceTree(Node node)`)

Checks balance factor and applies rotations to restore AVL property.

AVL Tree Pseudocode

class AVLNode:

    value: T

    left: AVLNode

    right: AVLNode

    height: int

2.2 Red-Black Tree

Structure

Each node stores:

`value` (generic type `T`).

`left`, `right`, and `parent` pointers.

`color` (`RED` or `BLACK`).

Key Operations

Insertion (`insert(T value)`)

Inserts as in BST, coloring new nodes **RED**.

Fixes violations:

Case 1: Uncle is **RED** → Recolor parent, uncle, and grandparent.

Case 2: Uncle is **BLACK** (triangle) → Rotate parent.

Case 3: Uncle is **BLACK** (line) → Rotate grandparent and recolor.

Deletion (`delete(T value)`)

Removes the node and replaces it with its child.

Fixes violations using `fixAfterDeletion()`:

Handles cases where sibling is **RED** or **BLACK**.

Performs rotations and recoloring to maintain properties.

Search (`search(T value)`)

Standard BST search (`O(log n)`).

Red-Black Tree Pseudocode

```
class RBNode:
    value: T
    left: RBNode
    right: RBNode
    parent: RBNode
    color: enum {RED, BLACK}
```

3. Time Complexity Analysis

Operation	AVL Tree	Red-Black Tree
Insert	O(log n)	O(log n)
Delete	O(log n)	O(log n)
Search	O(log n)	O(log n)
Size	O(1)	O(1)
Height	O(log n)*	O(log n)*

*Height is **O(n)** if not cached, but typically **O(log n)** due to balancing.

Comparison in Balance Condition:

AVL Tree:

Strictly balanced.

For every node, the **height difference** (balance factor) between its left and right subtrees must be **-1, 0, or +1**.

This stricter balance ensures **faster lookups** (more balanced = shorter height).

Red-Black Tree:

Loosely balanced.

Each node has a color (red or black), and the tree must follow these rules:

Root is always black.

No two red nodes can be adjacent (no red-red parent-child).

Every path from a node to a descendant null node must contain the **same number of black nodes**.

Because of looser balancing, it allows for **fewer rotations** during insertions

and deletions.

2. Worst-Case Height:

Let **n** be the number of nodes.

AVL Tree:

Worst-case height is **O(log n)**, but tighter than Red-Black.

Specifically, maximum height ≈ **1.44 $\log_2$(n)**.

Red-Black Tree:

Worst-case height is also **O(log n)**, but higher than AVL.

Maximum height ≈ **2 $\log_2$(n)** (due to more flexible balancing).

Summary:

Feature	AVL Tree	Red-Black Tree
Balance Condition	Balance factor ∈ {-1,0,1}	Black height rule & coloring
Balance Strictness	More strict	Less strict
Worst-case Height	~1.44 $\log_2(n)$	~2 $\log_2(n)$
Lookup Performance	Better	Slightly worse
Insert/Delete Speed	Slower (more rotations)	Faster (fewer rotations)

Conclusion:

- Use **AVL** when you need **faster search** (read-heavy).
- Use **Red-Black** when you need **faster insert/delete** (write-heavy).

2025-05-22 10:16:28 - Starting test: testLeftRightRotation_AVL
2025-05-22 10:16:28 - Test testLeftRightRotation_AVL PASSED (9 ms)

2025-05-22 10:16:28 - Starting test: testDeleteFromSingleElementTree_RB
2025-05-22 10:16:28 - Test testDeleteFromSingleElementTree_RB PASSED (1 ms)

2025-05-22 10:16:28 - Starting test: testInsertDuplicateElement_RB
2025-05-22 10:16:28 - Test testInsertDuplicateElement_RB PASSED (1 ms)

2025-05-22 10:16:28 - Starting test: testDeleteNodeWithTwoChildren_AVL
2025-05-22 10:16:28 - Test testDeleteNodeWithTwoChildren_AVL PASSED (1 ms)

2025-05-22 10:16:28 - Starting test: testLeftRightRotation_RB
2025-05-22 10:16:28 - Test testLeftRightRotation_RB PASSED (1 ms)

2025-05-22 10:16:28 - Starting test: testRightLeftRotation_AVL
2025-05-22 10:16:28 - Test testRightLeftRotation_AVL PASSED (0 ms)

2025-05-22 10:16:28 - Starting test: testTreeHeightAfterDeletions_RB
2025-05-22 10:16:28 - Test testTreeHeightAfterDeletions_RB PASSED (0 ms)

2025-05-22 10:16:28 - Starting test: testMultipleInsertions_AVL
2025-05-22 10:16:28 - Test testMultipleInsertions_AVL PASSED (0 ms)

2025-05-22 10:16:28 - Starting test: testLeftRotation_AVL
2025-05-22 10:16:28 - Test testLeftRotation_AVL PASSED (0 ms)

2025-05-22 10:16:28 - Starting test: testSearchEmptyTree_AVL
2025-05-22 10:16:28 - Test testSearchEmptyTree_AVL PASSED (1 ms)

2025-05-22 10:16:28 - Starting test: testDeleteNodeWithOneChild_RB
2025-05-22 10:16:28 - Test testDeleteNodeWithOneChild_RB PASSED (0 ms)

2025-05-22 10:16:28 - Starting test: testMultipleInsertions_RB
2025-05-22 10:16:28 - Test testMultipleInsertions_RB PASSED (0 ms)

2025-05-22 10:16:28 - Starting test: testDeleteLeafNode_RB
2025-05-22 10:16:28 - Test testDeleteLeafNode_RB PASSED (1 ms)

2025-05-22 10:16:28 - Starting test: testRightRotation_RB
2025-05-22 10:16:28 - Test testRightRotation_RB PASSED (0 ms)

2025-05-22 10:16:28 - Starting test: testComplexInsertDeleteSequence_RB
2025-05-22 10:16:28 - Test testComplexInsertDeleteSequence_RB PASSED (2 ms)

2025-05-22 10:16:28 - Starting test: testEmptyTreeProperties_RB
2025-05-22 10:16:28 - Test testEmptyTreeProperties_RB PASSED (0 ms)

2025-05-22 10:16:28 - Starting test: testDeleteNodeWithOneChild_AVL
2025-05-22 10:16:28 - Test testDeleteNodeWithOneChild_AVL PASSED (0 ms)

2025-05-22 10:16:28 - Starting test: testDeleteLeafNode_AVL
2025-05-22 10:16:28 - Test testDeleteLeafNode_AVL PASSED (1 ms)

2025-05-22 10:16:28 - Starting test: testSearchEmptyTree_RB
2025-05-22 10:16:28 - Test testSearchEmptyTree_RB PASSED (0 ms)

2025-05-22 10:16:28 - Starting test: testInsertDuplicateElement_AVL
2025-05-22 10:16:28 - Test testInsertDuplicateElement_AVL PASSED (0 ms)

2025-05-22 10:16:28 - Starting test: testEmptyTreeProperties_AVL
2025-05-22 10:16:28 - Test testEmptyTreeProperties_AVL PASSED (0 ms)

2025-05-22 10:16:28 - Starting test: testDeleteNodeWithTwoChildren_RB
2025-05-22 10:16:28 - Test testDeleteNodeWithTwoChildren_RB PASSED (0 ms)

2025-05-22 10:16:28 - Starting test: testInsertSingleElement_AVL
2025-05-22 10:16:28 - Test testInsertSingleElement_AVL PASSED (0 ms)

2025-05-22 10:16:28 - Starting test: testRightRotation_AVL
2025-05-22 10:16:28 - Test testRightRotation_AVL PASSED (0 ms)

2025-05-22 10:16:28 - Starting test: testLeftRotation_RB
2025-05-22 10:16:28 - Test testLeftRotation_RB PASSED (0 ms)

2025-05-22 10:16:28 - Starting test: testComplexInsertDeleteSequence_AVL
2025-05-22 10:16:28 - Test testComplexInsertDeleteSequence_AVL PASSED (0 ms)

2025-05-22 10:16:28 - Starting test: testRightLeftRotation_RB
2025-05-22 10:16:28 - Test testRightLeftRotation_RB PASSED (0 ms)

2025-05-22 10:16:28 - Starting test: testDeleteFromSingleElementTree_AVL
2025-05-22 10:16:28 - Test testDeleteFromSingleElementTree_AVL PASSED (0 ms)

2025-05-22 10:16:28 - Starting test: testTreeHeightAfterDeletions_AVL
2025-05-22 10:16:28 - Test testTreeHeightAfterDeletions_AVL PASSED (1 ms)

2025-05-22 10:16:28 - Starting test: testInsertSingleElement_RB
2025-05-22 10:16:28 - Test testInsertSingleElement_RB PASSED (0 ms)