COP5536 Advanced Data Structures

Project Report

March 31, 2025

Name: Anay Sinhal
UFID: 68789243
UF Email: sinhal.anay@ufl.edu

Project Overview

This project implements a Flying Broomstick Management System for the Office of Transportation, Ministry of Magic. The system manages license plates for flying broomsticks using a Red-Black Tree as the underlying data structure.

Key Features

- Registration of customized and random license plates
- Removal of license plates from the system
- Lookup operations for existing plates
- Finding lexicographically previous and next plates
- Range searches for plates between specified bounds
- Revenue calculations for standard and customized plates

Technical Implementation

Data Structure

The system uses a Red-Black Tree implemented from scratch (without built-in libraries) to efficiently manage license plate data with $O(\log n)$ time complexity for most operations.
A Red-Black Tree is a self-balancing binary search tree with the following properties:

1. Every node is either red or black
2. The root is black
3. Every leaf (NIL) is black
4. If a node is red, then both its children are black
5. All simple paths from a node to descendant leaves contain the same number of black nodes

Project Structure

Files

- `plateMgmt.java`: Main class implementing the license plate management system
- `RBTree.java`: Red-Black Tree implementation for storing and managing license plates
- `Makefile`: For compiling the project and creating the executable
- `test.txt`: Sample test cases for verifying functionality

Class Diagram

```
+-------------------------+         +------------------------+
|       plateMgmt         |         |        RBTree          |
+-------------------------+         +------------------------+
| -licenseTree: RBTree    |<------->| -root: Node            |
| -standardPlateCount: int|         | -RED/BLACK: boolean    |
| -customPlateCount: int  |         +------------------------+
| -outputWriter: Writer   |         | +insert(key): boolean  |
+-------------------------+         | +delete(key): boolean  |
| +addLicence(plateNum)   |         | +search(key): boolean  |
| +addLicence()           |         | +predecessor(key): str |
| +dropLicence(plateNum)  |         | +successor(key): str   |
| +lookupLicence(plateNum)|         | +range(lo, hi): str[]  |
| +lookupPrev(plateNum)   |         +------------------------+
| +lookupNext(plateNum)   |                     ^
| +lookupRange(lo, hi)    |                     |
| +revenue()              |                     |
+-------------------------+                     |
                                      +------------------+
                                      |       Node       |
                                      +------------------+
                                      | -key: String     |
                                      | -color: boolean  |
                                      | -left: Node      |
                                      | -right: Node     |
                                      | -parent: Node    |
                                      +------------------+
```

Function Prototypes and Explanations

plateMgmt Class

Main Operations

```
// Register new customized license plate
public void addLicence(String plateNum)

// Generate and register a random license plate
public void addRandomLicence()
```

```
// Remove license plate from the system
public void dropLicence(String plateNum)

// Check if license plate exists
public void lookupLicence(String plateNum)

// Find lexicographically previous plate
public void lookupPrev(String plateNum)

// Find lexicographically next plate
public void lookupNext(String plateNum)

// Find all plates in a given range
public void lookupRange(String lo, String hi)

// Calculate and report annual revenue
public void revenue()
```

Utility Functions

```
// Initialize output writer
public void initOutput(String outputFile)

// Close output writer
public void closeOutput()

// Process a command from input
public void processCommand(String command)

// Entry point for the program
public static void main(String[] args)
```

RBTree Class

Public Interface

```
public boolean insert(String key)
public boolean delete(String key)
public boolean search(String key)
public String predecessor(String key)
public String successor(String key)
public String[] range(String lo, String hi)
```

Tree Operations

```
private void fixAfterInsertion(Node node)
private void fixAfterDeletion(Node x)
private void rotateLeft(Node x)
private void rotateRight(Node x)
```

Implementation Details

License Plate Format: 4 characters, each can be a digit (0–9) or a capital letter (A–Z).
Fee Structure:

- Standard plates: 4 Galleons annually
- Customized plates: 7 Galleons annually

File I/O: Input commands are read from a file. Output is written to `<inputFilename> output.txt`.
Red-Black Tree Mechanics:

- Insertion: BST insert + RB tree fixes
- Deletion: BST delete + RB tree fixes
- Rotations: Left and right
- Recoloring: To maintain properties

Algorithm Analysis

Time Complexity:

- Search, Insert, Delete: $O(\log n)$
- Range search: $O(\log n + k)$ where k is number of results

Space Complexity: $O(n)$

Conclusion

The Flying Broomstick Management System efficiently manages license plates using a Red-Black Tree. It satisfies the assignment's requirements, providing robust support for various operations, including custom/random plate registration, lookups, and revenue computation.

References

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.
2. Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley.
3. Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). *Data Structures and Algorithms in Java* (6th ed.). Wiley.