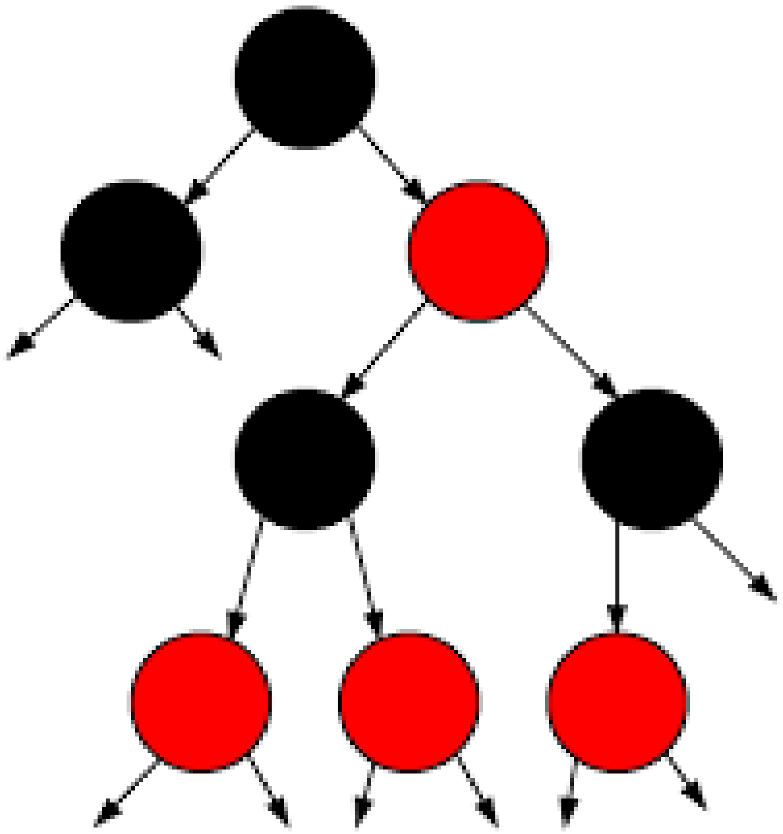


Gator Library Management System

COP 5536 Fall 2023

Sushanth Reddy Kotha
UFID: 67664160
su.kotha@ufl.edu

Overview

A C++ program that simulates a library management system using a Red-Black Tree. The system allows users to insert books, borrow and return books, delete books, and perform various queries.

Function Prototypes:

void InsertBook(const Book& newBook);

This function in the RedBlackTree class is responsible for inserting a new book into the RBT. It takes a Book object as an argument, which contains details about the book to be inserted, such as its ID, name, author, and availability status. The function creates a new node in the tree for this book, ensuring that the properties of the red-black tree (such as the balance and color rules) are maintained after the insertion.

void PrintBooks(int bookID1, int bookID2) const;

This const function in the RedBlackTree class is designed to print the details of all books whose IDs fall within the specified range, from bookID1 to bookID2. It traverses the red-black tree and prints details of each book that lies within this range.

void PrintBook(int bookID) const;

The void PrintBook(int bookID) const function in the RedBlackTree class is used to print the details of a specific book identified by its bookID. It searches the red-black tree for a node (book) with the given ID and, if found, prints its details.

void BorrowBook(int patronID, int bookID, int patronPriority);

This function in the RedBlackTree class is designed to manage the borrowing of a book by a patron. It takes three parameters: the ID of the patron borrowing the book (patronID), the ID of the book to be borrowed (bookID), and the priority of the patron (patronPriority). This function searches for the book in the red-black tree, and if the book is available, it updates the book's status to show that it is borrowed by the specified patron. If the book is not available, it adds the patron to the reservation list for the book, taking into account their priority.

void ReturnBook(int patronID, int bookID);

Checks if the specified book is borrowed by the given patron and processes its return, updating the tree and handling reservations.

void DeleteBook(int bookID);

Locates and removes the book with the given ID from the tree, ensuring the tree's balance and properties are maintained post-deletion.

void FindClosestBook(int targetID) const;

Searches the tree to find the book with the closest ID to the given target ID, then prints its details.

int ColorFlipCount() const;

Returns the cumulative count of color flips (red to black or vice versa) that occurred during tree modifications.

void RunLibrarySystem(const string& inputFile, const string& outputFile);

Processes commands from the input file (like adding, borrowing, returning books) and outputs results to the specified file.

void InsertNode(RBTreeNode* newNode);

Inserts a new node in the appropriate location in the tree and rebalances it if necessary to maintain tree properties.

void FixInsertViolation(RBTreeNode* node);

Adjusts the tree after inserting a node to fix any red-black property violations through rotations and color changes.

void RotateLeft(RBTreeNode* node);, void RotateRight(RBTreeNode* node);

Perform left and right rotations on a given node to maintain or restore the tree's balance

void BorrowBookHelper(RBTreeNode* node, int patronID, int bookID, int patronPriority);

Aids in borrowing a book by updating its status or adding the patron to the reservation list based on availability.

void ReturnBookHelper(RBTreeNode* node, int patronID, int bookID);

Facilitates the return of a book, making it available again and managing the reservation queue.

void PrintBookHelper(const RBTreeNode* node, int bookID) const;

void PrintBooksHelper(const RBTreeNode* node, int bookID1, int bookID2) const;

Assist in printing book details, either for a single book or for all books within a specified range of IDs.

void FindClosestBookHelper(const RBTreeNode* node, int targetID) const;

Helps locate the book closest to a given ID within the tree.

void DeleteBookHelper(RBTreeNode* node, int bookID);

Supports the deletion process by locating and removing the specified book.

void DeleteNode(RBTreeNode* node);

Manages the actual removal of a node from the tree and rebalances the tree as needed.

RBTreeNode* GetSuccessor(RBTreeNode* node);

Identifies the successor of a given node, typically used in deletion scenarios.

void FixDeleteViolation(RBTreeNode* node);

Resolves any violations of red-black properties after a node's deletion.

void ReplaceNode(RBTreeNode* node, RBTreeNode* child);

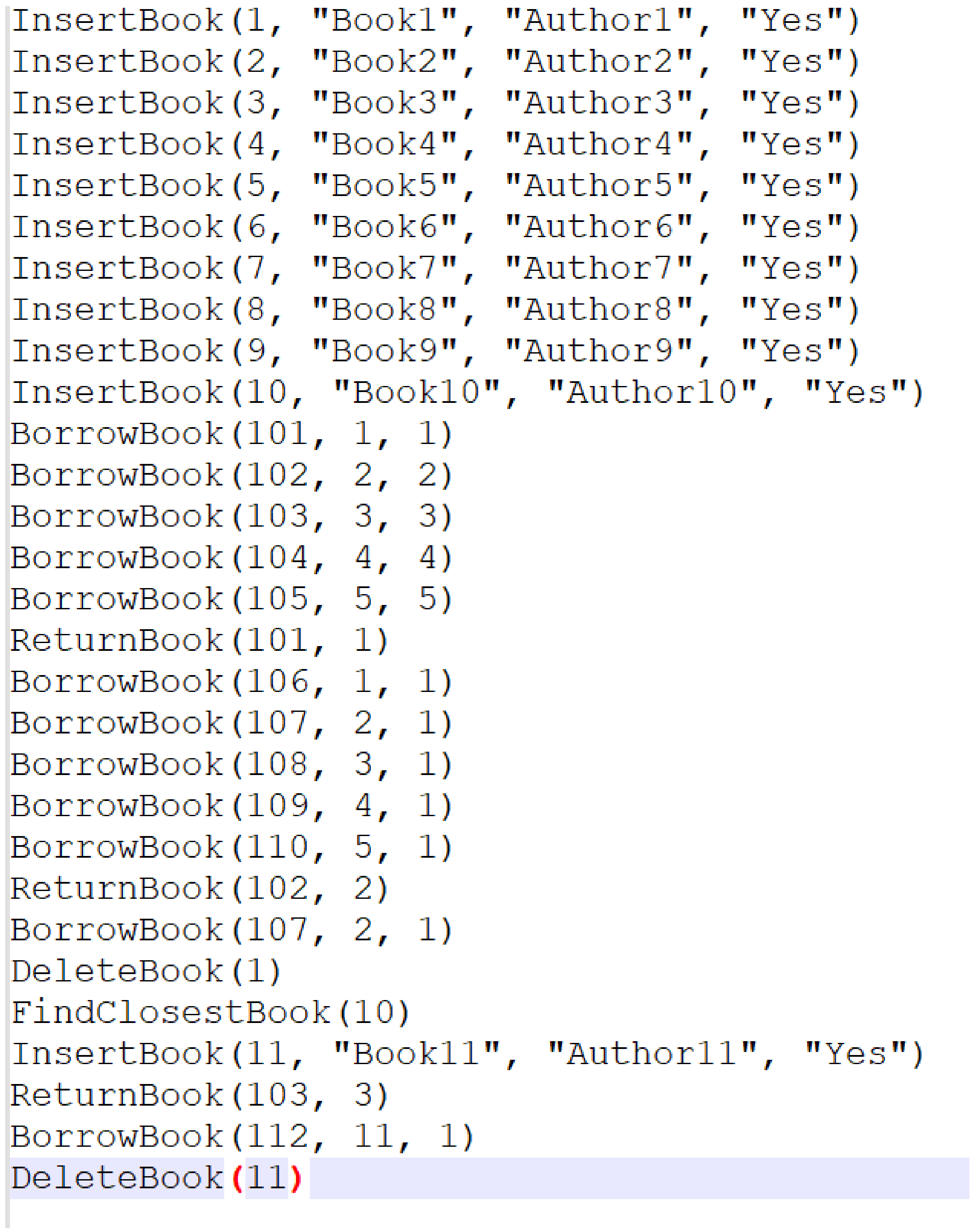
Replaces one node with another in the tree, used in deletion processes.

Program Structure:

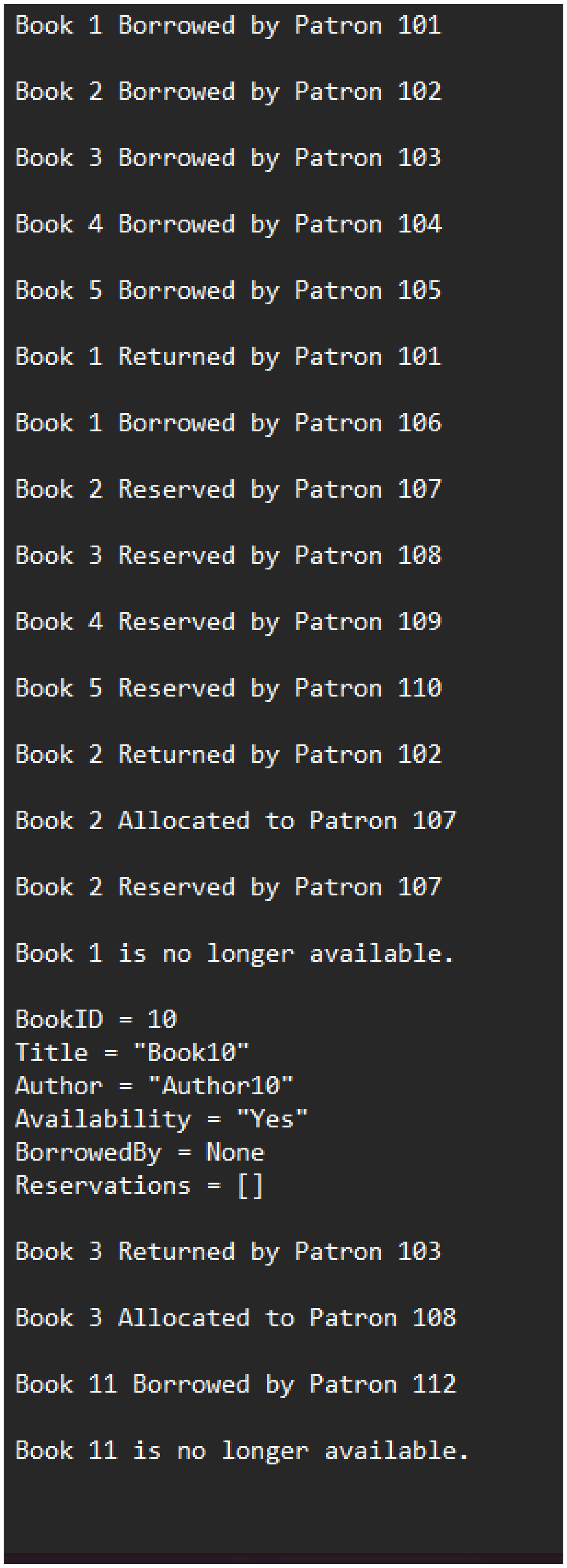
```
#include <bits/stdc++.h>
using namespace std;

// Enumeration for book color
enum class Color { RED, BLACK };

// Output file stream
ofstream outputFile;

// Class representing a Reservation
class Reservation { /* ... */ };

// Class representing a Book
class Book { /* ... */ };

// Class representing a Red-Black Tree Node
class RBTreeNode { /* ... */ };

// Class representing a Red-Black Tree
class RedBlackTree { /* ... */ };
```

```
// Function prototypes
void RunLibrarySystem(const string& inputFile, const string& outputFile);
// ... Other function prototypes ...

int main() {
    // Example usage of the library management system
    RunLibrarySystem("input.txt", "output.txt");
    return 0;
}

// Function implementations
// ... Implementation of RunLibrarySystem and other functions ...
```

Input/Output Screenshots:



```
InsertBook(1, "Book1", "Author1", "Yes")
InsertBook(2, "Book2", "Author2", "Yes")
InsertBook(3, "Book3", "Author3", "Yes")
InsertBook(4, "Book4", "Author4", "Yes")
InsertBook(5, "Book5", "Author5", "Yes")
InsertBook(6, "Book6", "Author6", "Yes")
InsertBook(7, "Book7", "Author7", "Yes")
InsertBook(8, "Book8", "Author8", "Yes")
InsertBook(9, "Book9", "Author9", "Yes")
InsertBook(10, "Book10", "Author10", "Yes")
BorrowBook(101, 1, 1)
BorrowBook(102, 2, 2)
BorrowBook(103, 3, 3)
BorrowBook(104, 4, 4)
BorrowBook(105, 5, 5)
ReturnBook(101, 1)
BorrowBook(106, 1, 1)
BorrowBook(107, 2, 1)
BorrowBook(108, 3, 1)
BorrowBook(109, 4, 1)
BorrowBook(110, 5, 1)
ReturnBook(102, 2)
BorrowBook(107, 2, 1)
DeleteBook(1)
FindClosestBook(10)
InsertBook(11, "Book11", "Author11", "Yes")
ReturnBook(103, 3)
BorrowBook(112, 11, 1)
DeleteBook(11)
```



```
Book 1 Borrowed by Patron 101

Book 2 Borrowed by Patron 102

Book 3 Borrowed by Patron 103

Book 4 Borrowed by Patron 104

Book 5 Borrowed by Patron 105

Book 1 Returned by Patron 101

Book 1 Borrowed by Patron 106

Book 2 Reserved by Patron 107

Book 3 Reserved by Patron 108

Book 4 Reserved by Patron 109

Book 5 Reserved by Patron 110

Book 2 Returned by Patron 102

Book 2 Allocated to Patron 107

Book 2 Reserved by Patron 107

Book 1 is no longer available.

BookID = 10
Title = "Book10"
Author = "Author10"
Availability = "Yes"
BorrowedBy = None
Reservations = []

Book 3 Returned by Patron 103

Book 3 Allocated to Patron 108

Book 11 Borrowed by Patron 112

Book 11 is no longer available.
```

How I calculated colorFlipCount?

- During insertions or deletions, if a color change of a node is required to maintain the RBT properties, the colorFlipCount is incremented.
- After Rotations: Sometimes, after performing a rotation (left or right), a color flip might be necessary to maintain the red-black properties. This also contributes to the colorFlipCount.
- Insertion Fix-up: When a new node is inserted, it's initially colored red. If this leads to two consecutive red nodes, the tree performs necessary rotations and color flips. Each color flip increments the colorFlipCount.
- Deletion Fix-up: When a node is deleted, especially a black node, the tree needs to ensure that the black height property is maintained. This often requires color flips, which are again counted.