COP 5536 Advanced Data Structures
Programming Project

GatorLibrary Management System

Name : Aashritha Reddy Donapati
UFID : 7242-2083
Email : a.donapati@ufl.edu

Under the Guidance of Dr. Sartaj Sahni

Table of Contents

Objective:

The objective of the GatorLibrary project is to develop a software system that efficiently manages books, patrons, and borrowing operations, aiming to enhance operational efficiency and provide a user-friendly platform for both library staff and patrons.

Each node in the Red-Black tree will represent a book and will have the following structure:

1. BookId // Integer ID
2. BookName //Name of the book
3. AuthorName //Name of the Author
4. AvailabilityStatus //To indicate whether it is currently borrowed
5. BorrowedBy //ID of the Patron who borrowed the book
6. ReservationHeap: Implement Binary Min-heap for managing book reservations and waitlists for the book ordered by the patron's priority which is an integer. Ties need be broken by considering the timestamp at which the reservation was made (first come first serve basis). Every node of the Min-heap should contain (patronID, priorityNumber, timeOfReservation)

Operations overview:

1. PrintBook(bookID): Print information about a specific book identified by its unique bookID (e.g., title, author, availability status). Note*: If not found, Print "Book not found in the Library"

2. PrintBooks(bookID1, bookID2): Print information about all books with bookIDs in the range [bookID1, bookID2].

3. InsertBook(bookID, bookName, authorName, availabilityStatus, borrowedBy, reservationHeap): Add a new book to the library. BookID should be unique, and availability indicates whether the book is available for borrowing. Note*: There is only one copy of a book i.e. all books are unique.

4. BorrowBook(patronID, bookID, patronPriority): Allow a patron to borrow a book that is available and update the status of the book. If a book is currently unavailable, create a reservation node in the heap as per the patron's priority (patronPriority). *Note: It is assumed that a book will not be borrowed by same patron twice.

5. ReturnBook(patronID, bookID): Allow a patron to return a borrowed book. Update the book's status and assign the book to the patron with highest priority in the Reservation Heap. (if there's a reservation).

6. DeleteBook(bookID): Delete the book from the library and notify the patrons in the reservation list that the book is no longer available to borrow.

7. FindClosestBook(targetID) : Find the book with an ID closest to the given ID (checking on both sides of the ID). Print all the details about the book. In case of ties, print both the books ordered by bookIDs.

8. ColorFlipCount(): GatorLibrary's Red-Black tree structure requires an analytics tool to monitor and analyze the frequency of color flips in the Red-Black tree. Track the occurrence of color changes in the Red-Black tree nodes during tree operations, such as insertion, deletion, and rotations. Note*: Only color flips should be counted i.e. when black changes to red and vice versa. (Only count the color flip if the color of the node has changed from previous state (before the operation) to current state)

Technology used?

1. Visual Studio Code
2. Java Development Kit (JDK)
3. Supported OS: Windows, MacOS

How to run the program?

1. Unzip Donapati_Aashritha..zip and open the folder.
2. Open a terminal and go to the path of the folder.
3. Run the following commands
    a. For compilation, run the ‘javac GatorLibrary.java’ command in the terminal. It will create class files for each class.
    b. For execution, run the following command in the terminal:
    Java GatorLibrary
4. Find the output in output_file.txt generated by the code

Structure of the code:

The entire code is divided into 5 files:

1. GatorLibrary.java
2. MinHeap.java
3. RedBlackTree.java
4. Patron.java
5. Book.java

GatorLibrary.java:

1. main method:
    a. This is the entry point of the program.
    b. It initializes a Red-Black Tree (redBlackTree) and reads input from a file named "input.txt".
    c. It processes each line of the input file and performs operations based on the input commands.
    d. The results are written to an output file named "output_file.txt".
2. InsertBook command:
    a. Parses input parameters to extract book information (bookId, bookName, authorName, availabilityStatus).
    b. Calls the insertBook method of the Red-Black Tree to insert a new book into the tree.
3. PrintBook command:
    a. Handles two cases:
        i. If only one bookId is provided, it searches for the book and writes its information to the output file.
        ii. If two bookIds are provided, it calls printBooks to get a list of books within the specified range and writes them to the output file.
4. BorrowBook command:
    a. Parses input parameters to extract patron information (patronId, bookId, patronPriority) and records the current time.
    b. Calls the borrowBook method of the Red-Black Tree to handle the book borrowing.
5. ReturnBook command:
    a. Parses input parameters to extract patron and book information.
    b. Calls the returnBook method of the Red-Black Tree to handle returning a borrowed book.
6. FindClosestBook command:
    a. Parses input parameters to extract the target bookId.

    b. Calls the findClosestBook method of the Red-Black Tree to find the closest books and writes the results to the output file.
7. DeleteBook command:
    a. Parses input parameters to extract the bookId.
    b. Calls the deleteBook method of the Red-Black Tree to delete the specified book.
8. ColorFlipCount command:
    a. Calls the colorFlipCount method of the Red-Black Tree and writes the result to the output file. This method appears to print directly to the console.
9. Quit command:
    a. Writes a termination message to the output file and exits the program.

MinHeap.java:

1. Constructor - MinHeap()
    a. Initializes an empty ArrayList to serve as the underlying data structure for the min-heap.
2. toString()
    a. Creates and returns a string representation of the elements in the min-heap. It iterates through the minHeap list and appends each Patron item followed by a comma to the result string.
3. isEmpty()
    a. Checks if the min-heap is empty by calling the isEmpty() method of the underlying ArrayList.
4. getAllElements()
    a. Returns a reference to the entire list of Patron elements in the min-heap.
5. insert(Patron patron)
    a. Adds a new Patron to the min-heap and performs the necessary heap operations to maintain the min-heap property. It compares the newly inserted Patron with its parent and swaps them if needed until the heap property is restored.
6. getMin()
    a. Retrieves the minimum (root) element from the min-heap without removing it. Returns null if the heap is empty.
7. removeMin()
    a. Removes and returns the minimum (root) element from the min-heap. It replaces the root with the last element, and then adjusts the heap to maintain the min-heap property by comparing the new root with its children and swapping if necessary.
8. delete(Patron patron)
    a. Removes a specific Patron from the min-heap. It first finds the index of the target Patron, replaces it with the last element, and then adjusts the heap to

maintain the min-heap property. The method ensures that both the parent-child relationships and the ordering of elements are preserved.

9. swap(int i, int j)
    a. Swaps two elements in the min-heap given their indices i and j. This is a helper method used in the insert(), removeMin(), and delete() methods to facilitate swapping elements during heap operations.

RedBlackTree.java:

1. Constructor - public RedBlackTree()

    a. Initializes the Red-Black Tree by creating a sentinel node (book) representing null.
    b. Initializes the root of the tree, colorFlip counter, and a MinHeap for managing reservations.
2. Search Tree Helper - private Book searchTreeHelper(Book node, int key)
    a. Recursive helper method to search the tree for a given key.
    b. Returns the node with the specified key or the sentinel node if not found.
3. Get Book - public Book getBook(): Returns the sentinel node representing null.
4. Delete Fix - private void deleteFix(Book x): Balances the Red-Black Tree after deletion to maintain its properties.
5. RB Transplant - private void rbTransplant(Book u, Book v): Replaces one subtree with another during deletion.
6. Find Minimum - private Book findMinimum(Book node): Finds the minimum node in a given subtree.
7. Delete Node Helper - private void deleteNodeHelper(Book node, int key): Deletes a node with a given key and fixes the Red-Black Tree afterward.
8. Fix Insert - private void fixInsert(Book k): Fixes the Red-Black Tree after insertion to maintain its properties.
9. Left Rotate - private void leftRotate(Book x): Performs a left rotation in the Red-Black Tree.
10. Right Rotate - private void rightRotate(Book x): Performs a right rotation in the Red-Black Tree.
11. Closest Node - private Book closestNode(Book node, int bookId, int[] minDiff, Book[] minDiffBook): Finds the closest node to a given key in the tree.
12. Find Closest Book - public List<String> findClosestBook(int targetId): Finds the closest book to a target ID and returns a list of tied books.
13. Find Closest Node - private Book findClosestNode(Book node, int targetId, Book currentClosest): Finds the closest node to a given key.

14. Find Ties - private void findTies(Book node, int targetId, int distance, List<Book> closestBooks): Finds tied books at the same distance from the target ID.
15. Insert Book - public void insertBook(int bookId, String bookName, String authorName, String availabilityStatus): Inserts a book into the Red-Black Tree and maintains the tree properties.
16. Delete Book - public String deleteBook(int bookId): Deletes a book from the Red-Black Tree and returns a message indicating the status.
17. Search Book - public Book searchBook(int bookId): Searches for a book with the given ID in the Red-Black Tree.
18. Print Book - public String printBook(int bookId): Prints information about a book with the specified ID.
19. Print Books - public List<String> printBooks(int bookId1, int bookId2): Prints information about a range of books.
20. Borrow Book - public String borrowBook(int patronId, int bookId, int patronPriority, LocalDateTime timeOfReservation): Handles the borrowing of a book by a patron.
21. Return Book - public String returnBook(int patronId, int bookId): Handles the return of a book by a patron.
22. Color Flip Count - public String colorFlipCount(): Returns the count of color flip operations.
23. In-Order Helper - public int inOrderHelper(Book node): Helper method for in-order traversal to count transformations.
24. Quit - public void quit(): Terminates the program and prints a termination message.

Patron.java:

1. getPatronId(): This method is a getter method that returns the value of the patronId attribute.
2. setPatronId(int patronId): This is a setter method that sets the value of the patronId attribute.
3. getPatronPriority(): This method is a getter method that returns the value of the patronPriority attribute.
4. setPatronPriority(int patronPriority): This is a setter method that sets the value of the patronPriority attribute.
5. getTimeOfReservation(): This method is a getter method that returns the value of the timeOfReservation attribute.
6. setTimeOfReservation(LocalDateTime timeOfReservation): This is a setter method that sets the value of the timeOfReservation attribute.
7. Patron(int patronId, int patronPriority, LocalDateTime timeOfReservation): This is the constructor of the Patron class. It initializes the object with the provided values for patronId, patronPriority, and timeOfReservation.

8. compare(Patron other): This method compares the current Patron object with another Patron object (other). It first compares the patronPriority values. If they are not equal, it returns false if the current object's priority is greater; otherwise, it compares the timeOfReservation values. If the current object's reservation time is earlier, it returns true, indicating that it has higher priority.
9. toString(): This method returns a string representation of the Patron object. It includes the patronId, patronPriority, and timeOfReservation values.
10. isGreaterThan(Patron other): This method checks if the current Patron object has a higher priority than another Patron object (other). It returns true if the current object's priority is greater.’

Book.java:

1. Constructor: Initializes the fields of the Book class, including bookId, bookName, authorName, availabilityStatus, borrowedBy, reservationHeap, parent, left, right, color, transformations, and flipCount.
2. Getter and Setter Methods: Provide access to private fields, allowing external classes to retrieve and modify the state of a Book object.
3. toString Method: Generates a string representation of the Book object, including details like bookId, bookName, authorName, availabilityStatus, borrowedBy, and a list of reservations.
4. insertIntoReservation Method: Inserts a reservation into the reservation heap. If the patron already has a reservation, the method updates the reservation priority. It also tracks and prints information about the reservation.
5. borrowBook Method: Handles the process of borrowing a book by a patron. If the book is available, it updates the borrowedBy and availabilityStatus fields. If not, it inserts the patron into the reservation heap.
6. returnBook Method: Handles the process of returning a book by a patron. If the book is borrowed by the specified patron, it updates the borrowedBy and availabilityStatus fields. If there are reservations, it assigns the book to the next patron in the reservation heap.
7. deleteBook Method: Deletes a book and cancels reservations if there are any. It generates a message indicating that the book is no longer available and lists the patrons whose reservations have been canceled. If there are no reservations, it simply states that the book is no longer available.

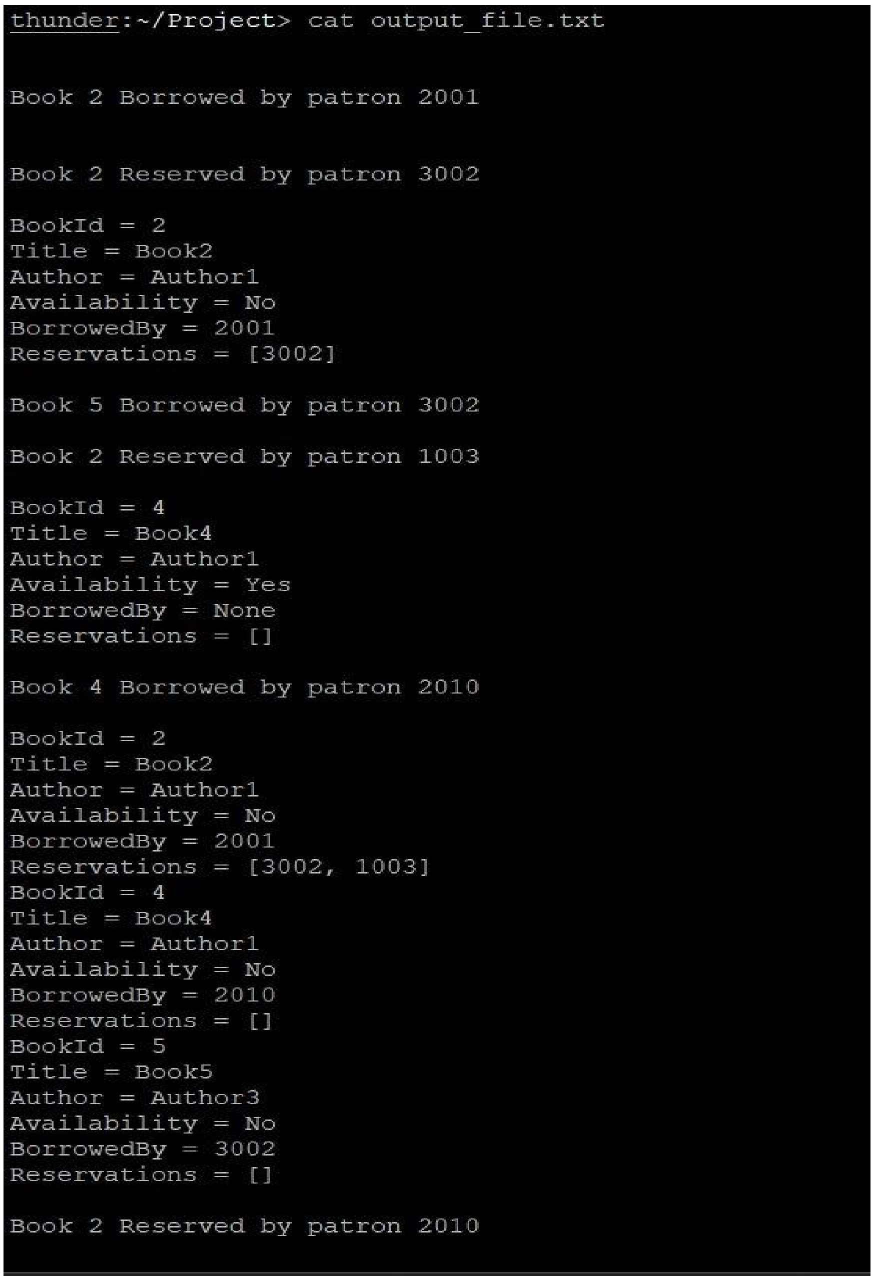
Execution Results:

For the input:

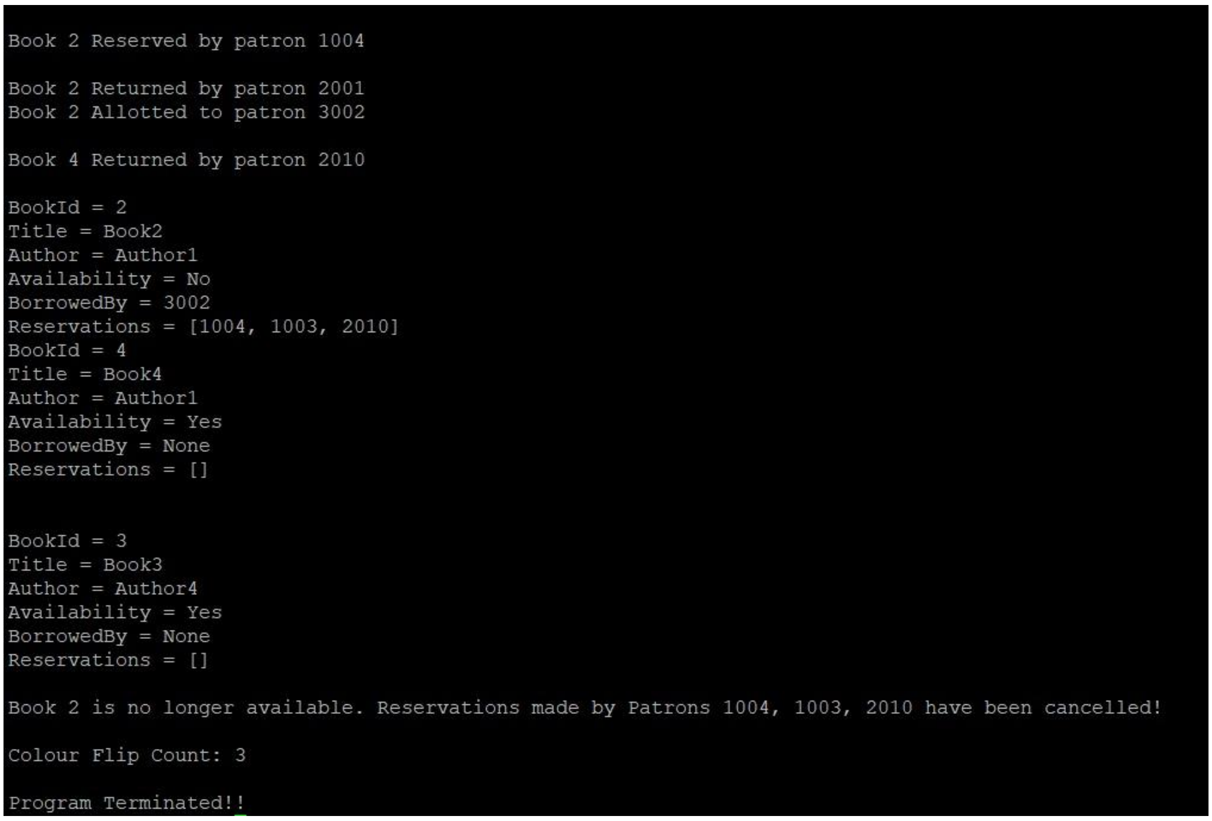
```
InsertBook(4, "Book4", "Author1", "Yes")
InsertBook(2, "Book2", "Author1", "Yes")
BorrowBook(2001, 2, 3)
InsertBook(5, "Book5", "Author3", "Yes")
BorrowBook(3002, 2, 1)
PrintBook(2)
BorrowBook(3002, 5, 1)
BorrowBook(1003, 2, 4)
PrintBook(4)
BorrowBook(2010, 4, 2)
PrintBooks(2, 5)
BorrowBook(2010, 2, 2)
BorrowBook(1004, 2, 4)
ReturnBook(2001, 2)
ReturnBook(2010, 4)
FindClosestBook(3)
InsertBook(3, "Book3", "Author4", "Yes")
FindClosestBook(3)
DeleteBook(2)
ColorFlipCount()
Quit()
PrintBook(4)
BorrowBook(2)
ReturnBook(1003, 2)
```

This is added in the input.txt file

The output is:



```
thunder:~/Project> cat output_file.txt


Book 2 Borrowed by patron 2001


Book 2 Reserved by patron 3002

BookId = 2
Title = Book2
Author = Author1
Availability = No
BorrowedBy = 2001
Reservations = [3002]

Book 5 Borrowed by patron 3002

Book 2 Reserved by patron 1003

BookId = 4
Title = Book4
Author = Author1
Availability = Yes
BorrowedBy = None
Reservations = []

Book 4 Borrowed by patron 2010

BookId = 2
Title = Book2
Author = Author1
Availability = No
BorrowedBy = 2001
Reservations = [3002, 1003]
BookId = 4
Title = Book4
Author = Author1
Availability = No
BorrowedBy = 2010
Reservations = []
BookId = 5
Title = Book5
Author = Author3
Availability = No
BorrowedBy = 3002
Reservations = []

Book 2 Reserved by patron 2010
```



```
Book 2 Reserved by patron 1004

Book 2 Returned by patron 2001
Book 2 Allotted to patron 3002

Book 4 Returned by patron 2010

BookId = 2
Title = Book2
Author = Author1
Availability = No
BorrowedBy = 3002
Reservations = [1004, 1003, 2010]
BookId = 4
Title = Book4
Author = Author1
Availability = Yes
BorrowedBy = None
Reservations = []


BookId = 3
Title = Book3
Author = Author4
Availability = Yes
BorrowedBy = None
Reservations = []

Book 2 is no longer available. Reservations made by Patrons 1004, 1003, 2010 have been cancelled!

Colour Flip Count: 3

Program Terminated!!
```

This can be found in the file output_file.txt