Advanced Data Structures

Project Report

Name: Guna Teja Athota

UFID: 36434173
UF email account: gunateja.athota@ufl.edu

Project Overview:

GatorLibrary, a fictitious library, requires a complex software solution to manage its vast book collection, users, and borrowing operations. The proposed system makes use of a Red-Black tree data structure for efficient book management and a Binary Min-heap to handle book reservations when a book is not readily accessible for borrowing. The project's goal is to offer a comprehensive solution that fits GatorLibrary's particular demands.

System Architecture:

The development of a Red-Black tree to effectively handle books is the system's base. Each node in the tree represents a different book and includes properties like BookId, BookName, AuthorName, AvailabilityStatus, BorrowedBy, and a ReservationHeap. The ReservationHeap, which is implemented as a Binary Min-heap, organizes book reservations and waitlists in the order specified by the patron priority, ensuring an organized and prioritized queue for borrowing.

Node Structure:

The following properties define each Red-Black tree node representing a book:

BookId: A unique integer ID assigned to each book.
BookName: The title of the book.
AuthorName: The author's name.
Availabilitystatus of a book indicates whether or not it is currently borrowed.
BorrowedBy: The patron's ID who borrowed the book.
ReservationHeap: Manages book reservations and waitlists in a binary min-heap. Each heap node has (patronID, priorityNumber, timeOfReservation).

Code Structure:

The following Java code implements a Red-Black Tree (RBT) data structure with added features for book management in a library system. The following is an explanation of the major components as well as the theory underlying Red-Black Trees:

1. **The Red-Black Tree**:
   A self-balancing binary search tree with an additional bit for expressing the color (either red or black) at each node.
   Each node has one of two colors: red or black.
   Black roots and leaves (null nodes).
   There can be no red children (no consecutive red nodes on any route).
   Every path from a node to its descendant leaves has the same amount of black nodes, which ensures that the height is balanced.

2. **Book Class:**

A node in the Red-Black Tree that stores information on a book in the library.
Each book node has a distinct bookId and may be connected with a variety of variables such as bookName, authorName, availabilityStatus, borrowedBy, and a ReservationHeap for reservation management.

3. **ReservationHeap Class**:
   Represents a min-heap of reservations associated with a book.
   To handle reservations based on priority numbers and reservation times, a binary heap (implemented as an inner class BinaryHeap) is used.

4. **BinaryHeap Class**:
   Implements a binary heap structure for priority queue management.
   Offer (insertion), peek (see the highest-priority element), and poll (delete and return the highest-priority element) are all supported operations.

5. **Red Black Tree Operations**:
   Insertion('inserBook')
   Deletion(' deleteBook')
   Searching('printBook')
   Travesal('inorderTraversal', 'inorder')

6. **Library Management Operations:**
   borrowBook
   returnBook
   Additional features include publishing books within a certain range, locating the nearest book to a target, and displaying color flip counts.

7. **GatorLibrary Class Methods**
   parseLine(RedBlackTree redBlackTree, String line): Parses a line from the input file and performs operations on the Red-Black Tree accordingly.

8. **File Reading**
   Reads input from a specified file (**input_file3.txt**)

9. **Red black Tree Operations:**

10. 'make' command compiles the source code and produces an executable file

Operations:

PrintBook(bookID):

Prints precise information on a book identifiable by its unique BookID.
If the book cannot be discovered, it is printed "Book not found in the Library."

PrintBooks(bookID1, bookID2):

Prints information about all books with bookIDs between [bookID1, bookID2].

InsertBook(bookID, bookName, authorName, availabilityStatus, borrowedBy, reservationHeap):

Adds a new book to the library with a unique BookID and includes information like BookName, AuthorName, and availability status.
Each book is one-of-a-kind, with just one copy available.

BorrowBook(patronID, bookID, patronPriority):

If a book is available, a patron may borrow it.
If a book is currently unavailable, it generates a heap reserve node depending on the patron's priority.

ReturnBook(bookID, patronID):

Allows a patron to return a borrowed book and update the status of the book.
If there is a reservation, assigns the book to the patron with the greatest priority in the ReservationHeap.

DeleteBook(bookID):

The book is removed from the library.
Notifies reservation list patrons that the book is no longer available for borrowing.

FindClosestBook(targetID):

Checks both sides for the book with the closest ID to the specified targetID.
All information about the book is printed.
In the event of a tie, publish both books in the order specified by bookIDs.

ColorFlipCount():

A monitoring and analysis tool for the frequency of color flips in the Red-Black tree structure.
Color changes in Red-Black tree nodes are tracked throughout operations such as insertion, deletion, and rotation.
Only occasions where the color changes from black to red or vice versa are counted.

Time Compexity:

Insertion Operation (‘insertBook’ method):

The average insertion operation in a Red-Black Tree takes O(log n) time, where n is the number of items in the tree. However, due to the balancing procedures, it may take O(log n) time in the worst case.

Deletion Operation (‘deleteBook’ method):

The deletion operation in a Red-Black Tree, like insertion, requires O(log n) time in the average and worst case.

Search Operation (‘printBook’ method):

In the average situation, searching for a certain book in the Red-Black Tree takes O(log n) time, where n is the number of items in the tree.

Inorder Traversal (‘inorderTraversal’ method):

The temporal complexity of the Red-Black Tree inorder traversal is O(n), where n is the number of of elements in the tree.

Borrow and Return Operations (methods ‘borrowBook’ and ‘returnBook’):

Both processes entail searching for a certain book, which has an average time complexity of O(log n).

Termination Operation(‘quit’ method):

The termination procedure contains no sophisticated algorithms and has a minor time complexity when compared to other operations.

Overall, the Red-Black Tree operations dominate the time complexity of the code, and the most popular operations like insertion, deletion, and search have an average time complexity of O(log n). Keep in mind that these complexity are predicated on the Red-Black Tree being balanced during normal usage.

INPUT and OUTPUT

The input is the Input_file2.txt

The output:

Conclusion:

The GatorLibrary Management System is a powerful and comprehensive system designed to suit the complicated requirements of a fictitious library. The use of Red-Black trees in conjunction with Binary Min-heaps provides effective book management and reservation processing. The project follows the rules for programming language flexibility, input/output requirements, and submission. The exhaustive study delves into the system's architecture, algorithms, and implementation decisions, resulting in a complete knowledge of the GatorLibrary Management System.