Vandit Shah
UFID: 50341980
shahvandit@ufl.edu
ADVANCE DATA STRUCTURES
COP5536, FALL 23
PROGRAMMING PROJECT REPORT

GATOR LIBRARY MANAGEMENT SYSTEM

USING RED BLACK TREE & MIN HEAP

OBJECTIVE

Primary Objective: Develop and implement a robust, efficient library management system for GatorLibrary, designed to streamline the handling of books, patrons, and borrowing operations.

Key Features and Goals:

Red-Black Tree for Book Management:

- Utilize a Red-Black Tree data structure to manage the library's collection of books.
- Ensure efficient operations for common library tasks, such as adding new books, searching for specific titles or authors, deleting books from the collection, and maintaining an organized inventory.
- Leverage the self-balancing nature of Red-Black Trees to keep the book management operations optimally efficient, particularly for large volumes of books.

Priority Queue for Reservation Management:

- Implement a Binary Min-Heap based priority queue system for each book to manage reservations effectively.
- Prioritize reservations based on specific criteria, such as reservation time or patron priority level, ensuring a fair and systematic allocation of books when they become available.
- Allow patrons to reserve books that are currently on loan and automatically notify them when these books become available.
- Efficient Handling of Borrowing Operations:

- Streamline the process of book checkouts and returns, incorporating real-time updates to book availability status.
- Integrate the reservation system with the borrowing operations, so that returned books can be immediately allocated to the next waiting patron in the priority queue.

- Maintain a history of borrowing activities for each book, assisting in the analysis of book popularity and usage patterns.
- User-Friendly Interface for Patrons and Staff:

- Design an intuitive, easy-to-use interface for both library patrons and staff, facilitating seamless interaction with the system for various operations like book searches, reservation requests, and management tasks.
- Color flip function allows us to track the occurrence of color changes in the Red-Black tree nodes during tree operations, such as insertion, deletion, and rotations

CODE STRUCTURE

Here is an overview of the code structure:

1. ReservationNode

- Purpose: Represents a reservation made by a patron for a book.

- Attributes:

  - patronID: Unique identifier for the patron making the reservation.

  - priorityNumber: Priority of the reservation (lower number indicates higher priority).

  - timeOfReservation: Timestamp of when the reservation was made.

- Functionality: The constructor initializes the node with the patron's ID and priority number and sets the reservation time to the current time.

2. ReservationHeap

- Purpose: Manages a min-heap of ReservationNodes to handle book reservations in order of priority and time.

- Attributes:

  - max: Maximum size of the heap (defaulted to 20).

  - size: Current number of elements in the heap.

  - reservations: Vector of ReservationNodes representing the heap.

- Functionality:

  - Provides methods for heapifying up and down (heapifyUp, heapifyDown) to maintain the heap order based on priority and reservation time.

  - The heap structure ensures that the reservation with the highest priority (and earliest time, in case of a tie) is always at the front.

3. BookNode

- Purpose: Represents a book in the library's collection.

- Attributes:

- bookId: Unique identifier for the book.

- bookName: Title of the book.

- authorName: Name of the author.

- availabilityStatus: Indicates whether the book is available for borrowing.

- borrowedBy: Patron ID of the borrower (if any).

- rhp: ReservationHeap managing reservations for this book.

- color: Color of the node (Red or Black) used in the Red-Black Tree.

- parent, left, right: Pointers to the parent and child nodes in the Red-Black Tree.

- Functionality: Used to store and manage book information and its position in the Red-Black Tree. It also manages reservations through its ReservationHeap.

4. Library

- Purpose: Represents the library, which uses a Red-Black Tree to manage its collection of books.

- Attributes:

- root: Root node of the Red-Black Tree containing BookNodes.

- EXTNODE: External node used in the Red-Black Tree.

- color_flip: Counter for the number of color changes during insertions and deletions.

- Functionality:

- Manages the overall library operations, including adding and deleting books, borrowing and returning books, and searching for books.

- It maintains the Red-Black Tree structure to ensure efficient access and modification of the book collection.

- Provides methods for fixing the tree after insertions and deletions (fixInsert, fixDelete), rotations (leftRotate, rightRotate), and utility functions (borrowBook, returnBook, searchBook, printBook, printBooks, closestBook, searchClosest).

- The color_flip attribute is used to track the number of color changes in the tree, which is an important aspect of maintaining the Red-Black Tree properties.

Overview of functions used in the code:

Library Class

Library()
- Purpose: Constructor for the Library class.
- Functionality: Initializes an empty Red-Black Tree with a sentinel external node (EXTNODE) set to black.

- Complexity: O(1), constant time operation.

fixDelete(BookNode* x)
- Purpose: Restores Red-Black Tree properties after a deletion.
- Functionality: Adjusts the tree to maintain balance and color properties following the Red-Black Tree rules.
- Complexity: O(log n), as it might need to traverse the tree height.

deleteBook(int bookID)
- Purpose: Removes a book from the tree.
- Functionality: Searches for and removes the book with the given ID. Adjusts the tree to maintain Red-Black properties using fixDelete.
- Complexity: O(log n), since it involves tree traversal and possibly fixing the tree.

rbTransplant(BookNode* u, BookNode* v)
- Purpose: Transplants nodes in the Red-Black Tree.
- Functionality: Replaces subtree rooted at node u with subtree rooted at node v.
- Complexity: O(1), as it involves only pointer updates.

minimum(BookNode* node)
- Purpose: Finds the node with the minimum key in the subtree.
- Functionality: Traverses the left children to find the minimum key.
- Complexity: O(log n) in the worst case, due to tree height traversal.

insert(BookNode* book)
- Purpose: Inserts a new book into the Red-Black Tree.
- Functionality: Standard binary search tree insertion followed by a call to fixInsert to maintain Red-Black properties.
- Complexity: O(log n), as it involves tree traversal and fixing.

fixInsert(BookNode* k)
- Purpose: Fixes the Red-Black Tree after an insertion.
- Functionality: Adjusts colors and performs rotations to maintain Red-Black properties after a new insertion.
- Complexity: O(log n), as it might need to traverse up to the root.

leftRotate(BookNode* x) and rightRotate(BookNode* x)
- Purpose: Performs left and right rotations in the tree.
- Functionality: Adjusts the tree structure by rotating nodes around x to maintain tree balance.
- Complexity: O(1), as it involves only pointer rearrangements.

borrowBook(int patronID, int bookID, int priorityNumber)
- Purpose: Handles borrowing or reserving a book.
- Functionality: Allows a patron to borrow a book if available, or reserve it if not.
- Complexity: O(log n) for searching the book, O(log m) for inserting into the reservation heap, where m is the number of reservations.

returnBook(int patronID, int bookID)
- Purpose: Handles returning a borrowed book.
- Functionality: Marks a book as returned and updates the borrower if there are reservations.
- Complexity: O(log n) for searching the book, O(log m) for updating the reservation heap.

searchBook(int bookID)
- Purpose: Searches for a book in the library.
- Functionality: Finds a book by its ID using binary search tree traversal.
- Complexity: O(log n), due to tree traversal.

printBook(int bookID) and printBooks(int bookID1, int bookID2)
- Purpose: Prints details of one or two books.
- Functionality: Displays book information, including reservations.
- Complexity: O(log n) for each book search.

searchTreeHelper(BookNode* book, int bookID)
- Purpose: Recursive helper for searching a book.
- Functionality: Performs the actual recursive search in the tree.
- Complexity: O(log n), as it traverses the tree.

closestBook(int n)
- Purpose: Finds books closest to a given ID.
- Functionality: Searches for books with IDs closest to n.
- Complexity: Potentially O(n), as it may traverse many nodes.

searchClosest(BookNode* node, std::vector<int> &result, int key, int diff)
- Purpose: Helper for finding closest books.
- Functionality: Recursively searches for books within a certain ID difference.
- Complexity: Potentially O(n), depending on the tree structure.

get_color_flips()
- Purpose: Debugging utility to track color changes.
- Functionality: Displays the number of color flips that

PROTOTYPE/STRUCTURE:

Class Library

```
class Library {
private:
    BookNode* root;
    int color_flip=0;

public:
    BookNode* EXTNODE;
    Library(){
    ReservationHeap rp;
    EXTNODE= new BookNode(-1, "", "", "No");
    EXTNODE->color=BLACK;
    root = EXTNODE;
    }
```

Class BookNode

```
class BookNode {
    public:
    int bookId;
    std::string bookName;
    std::string authorName;
    std::string availabilityStatus;
    int borrowedBy;
    ReservationHeap rhp;
    Color color;

    BookNode* parent;
    BookNode* left;
    BookNode* right;
```

INSTRUCTIONS TO RUN THE CODE:

- Unzip the file shah_vandit.zip and open the folder
- Run terminal in that directory and run the command ‘make’, it will generate an executable file named gatorlibraryfile_vanditshah
- Then enter command ./gatorlibraryfile_vanditshah
- The output will be displayed for the 1st test case