Advanced Data Structures

COP 5536 Fall 2023 Programming Project

GatorLibrary Management System

Name: Satya Aakash Chowdary Obellaneni
UF ID: 26077180
UF email: s.obellaneni@ufl.edu

Under the Guidance of
Dr. Sartaj Sahni

Project Objectives:
The GatorLibrary Management System's goal is to create and implement an efficient software solution for the fictitious GatorLibrary. The system uses Red-Black tree data structures for book management and Binary Min-heaps for book reservations to streamline the management of books, patrons, and borrowing activities. The following are the primary objectives:
Implementation of the Red-Black Tree: Create a Red-Black tree structure to represent books, with properties such as BookID, BookName, AuthorName, AvailabilityStatus, BorrowedBy, and a ReservationHeap handled as a Binary Min-heap.

Priority-Queue method: Within the ReservationHeap, implement a priority-queue method to manage book reservations and waitlists, prioritizing clients based on their allocated priority numbers and reservation timestamps.
Printing book details, inserting new books, borrowing books, returning books, deleting books, finding the closest book to a given ID, and analyzing color flips in the Red-Black tree during tree operations are all examples of efficient book operations.

Handling Input and Output: Read input data from a specified text file, perform the desired procedures, and generate output to a corresponding text file. Ensure that results are correctly formatted and reported, paying special attention to Reservation Heap printing and file name rules.

Usability and robustness: Make the system user-friendly and robust, addressing edge circumstances and assuring proper operation execution. In circumstances where a book cannot be found or an action is invalid, provide relevant error messages.
Performance and scalability: Optimize the system for optimal performance, particularly when dealing with enormous datasets. Assure that the Red-Black tree and Binary Min-heaps can scale in order to handle the library's expanding collection and patron base.

Technologies: Java, Visual Studio Code

Execution Steps:

- Extract the contents of Obellaneni_SatyaAakash.zip and navigate into the resulting folder.
- Open Command Prompt or PowerShell in the current folder and execute the following command: **java gatorLibrary filename**
- The output file will be generated in the same folder with the name **filename_output_file.txt.**

Function Prototypes and code structures:

The entire code is divided into 7 code files :

1) **EmptyRBNode.java: The** EmptyRBNode class is in charge of generating a nil sentinel node for usage in the Red-Black tree implementation. This sentinel node is a RedBlackNode instance that has been given the ID -1. The class is primarily used to manage the particular node in the Red-Black tree**.**

```
public class EmptyRBNode {

  static RedBlackNode nil = new RedBlackNode(-1);

}
```

Time Complexity: O(1)

- The file consists of a single static initialization, which is constant time.

GatorLibrary.java

The GatorLibrary Management System's principal entry point is the gatorLibrary class. It takes input from a text file, executes various library operations with the RedBlackTree class, and outputs the results. The main function, a parsing method, and a method for producing and publishing output to a text file are all part of the class structure.

This class takes an organized approach to library management, including error handling and integration with the RedBlackTree class. The main method sets up the necessary resources, the parsing method reads the input rows and calls the appropriate RedBlackTree methods, and the output generation method generates a text file with the output data.

```
import java.io.*;

public class gatorLibrary {

  // Class constants for delimiters

  private static final String COMMA = ",";

  private static final String OPEN_PARENTHESIS = "(";

  private static final String CLOSED_PARENTHESIS = ")";


  // Main method for executing the GatorLibrary Management System

  public static void main(String[] args) {

    // Implementation details: Reads input from a specified text file, initializes RedBlackTree instance, and processes library operations.

  }


  // Method to parse input rows and execute corresponding library operations

  private static void parse(RedBlackTree rbTree, String row, String fileName) throws IOException {

    // Implementation details: Extracts operation details from input rows, invokes corresponding RedBlackTree methods, and handles invalid operations.

  }


  // Method to create and write output to a text file

  private static void createOutput(String fileName) throws IOException {
```

```
        // Implementation details: Writes the output to a text file with a specific naming convention.
    }
}
```

Time Complexity:

The main method has a loop that reads lines from a file, and for each line, it calls the parse method. The time complexity of reading each line is O(N), where N is the length of the line.

The parse method contains conditional statements, each taking constant time.

The overall time complexity of the main method is O(N), where N is the total size of the input file.

MinHeap.java

The MinHeap class represents a Binary Min-Heap data structure that is used in the GatorLibrary Management System to manage reservations. It includes methods for inserting, extracting, and manipulating reservation nodes depending on priority. The class offers heapifyUp and heapifyDown operations for preserving the heap property, as well as utility functions for accessing and displaying the heap**.**

```
import java.util.ArrayList;
import java.util.List;


public class MinHeap {
    // Class variables
    private final int capacity;
    private int size;
    public ReservationNode[] heap;

    // Constructor to initialize the MinHeap
    public MinHeap(int capacity) {
        // Implementation details: Initializes the heap with the specified capacity and sets the size to zero.
    }

    // Method to check if the MinHeap is empty
    public boolean isEmpty() {
        // Implementation details: Returns true if the size is zero, indicating an empty heap.
    }

    // Method to get the current size of the MinHeap
    public int size() {
        // Implementation details: Returns the current size of the heap.
    }
```

```
// ... (Other utility methods for index calculations and accessing elements)

// Method to retrieve the top element (minimum) without removing it
public ReservationNode peek() {
    // Implementation details: Returns the root element of the heap (minimum) without removing it.
}

// Method to extract and remove the top element (minimum) from the MinHeap
public ReservationNode poll() {
    // Implementation details: Removes and returns the root element of the heap (minimum).
}

// Method to insert a new ReservationNode into the MinHeap
public void insertNode(ReservationNode reservation) {
    // Implementation details: Inserts a new reservation into the heap and maintains the heap property.
}

// Method to restore the heap property after insertion
public void heapifyUp() {
    // Implementation details: Restores the heap property by moving the newly inserted element up the heap.
}

// Method to restore the heap property after extraction
public void heapifyDown() {
    // Implementation details: Restores the heap property by moving the element down the heap.
}

// Method to swap two elements in the heap
public void swap(int x, int y) {
    // Implementation details: Swaps two elements in the heap at specified indices.
}

// Method to print the current state of the MinHeap
public void printHeap() {
    // Implementation details: Prints the elements of the heap for debugging purposes.
}

// toString method for converting the heap to a string representation
@Override
```

```
    public String toString() {
        // Implementation details: Converts the heap to a comma-separated string representation.
    }
}
```

The MinHeap class incorporates the critical functionality required for properly managing reservations in the GatorLibrary system. It follows the principles of a Binary Min-Heap, keeping the smallest element at the root and allowing for insertion, extraction, and property restoration.

Time Complexity:

Insertion (insertNode method): O(log N), where N is the number of elements in the heap.

Deletion (poll method): O(log N), where N is the number of elements in the heap.

Heapify Up (heapifyUp method): O(log N), where N is the number of elements in the heap.

Heapify Down (heapifyDown method): O(log N), where N is the number of elements in the heap.

Overall, the time complexity of operations is logarithmic in the number of elements.

Nodecolor.java

The NodeColor enumeration defines two color states, RED and BLACK, which are used in the GatorLibrary Management System's Red-Black tree structure.

```
public enum NodeColor {
    RED, BLACK;
}
```

- The NodeColor enum represents the two potential colors associated with nodes in a Red-Black tree in a straightforward manner. It improves the readability and clarity of the code by expressing the color of each tree node in a meaningful and succinct manner.

  Time Complexity: O(1)
- This file only contains an enumeration, and there are no operations performed. The time complexity is constant.

RedBlackNode.java

The RedBlackNode class represents a node in the Red-Black tree used in the GatorLibrary Management System to manage books. It contains information about books such as BookID, title, author, availability, borrower data, color, and a MinHeap for reservations.

```
public class RedBlackNode {
    // Instance variables
    int bookId;
```

```
    String bookName;
    String authorName;
    boolean isAvailable;
    int borrowedBy;
    RedBlackNode left, right, parent;
    NodeColor color;
    MinHeap minHeap;

    // Constructors
    public RedBlackNode() {
        // Default constructor
    }

    public RedBlackNode(int bookId, String bookName, String authorName, boolean isAvailable) {
        // Parameterized constructor for creating a new RedBlackNode with book details and MinHeap for reservations
    }

    public RedBlackNode(int bookId) {
        // Constructor for creating a new RedBlackNode with only BookID and default values
    }

    // toString method for creating a string representation of the RedBlackNode
    @Override
    public String toString() {
        // Converts the RedBlackNode to a formatted string for display
    }
}
```

Time Complexity: O(1)

- The class defines the structure of a Red-Black Tree node. The methods in this class primarily involve assignments and comparisons, each of which is constant time.

The RedBlackNode class contains the essential features of a library book, such as the BookID, title, author, availability, borrower details, color, and a MinHeap for managing reservations. The constructors enable the generation of nodes with varying levels of complexity, while the toString method provides a formatted string representation for display in the GatorLibrary Management System.

RedBlackTree.java

The RedBlackTree class controls a Red-Black tree structure used in the GatorLibrary Management System to arrange and conduct operations on books. This class has methods for inserting, deleting, borrowing, returning, searching, and performing many utility actions.

```
class RedBlackTree {

    // Class variables
    public static Map<Integer, NodeColor> hm1 = new HashMap<>();
    public static Map<Integer, NodeColor> hm2 = new HashMap<>();
    private final RedBlackNode nil = EmptyRBNode.nil;
    public int flipCount;
    private RedBlackNode root;
    public static StringBuilder resultString = new StringBuilder();

    // Constructor
    public RedBlackTree() {
        // Initializes the Red-Black tree with a nil node and sets the initial flipCount to zero
    }

    // Public method to insert a book into the Red-Black tree
    public void insertBook(int bookId, String bookName, String authorName, String isAvailable) {
        // Inserts a new book into the Red-Black tree and updates color flip count
    }

    // Public method to get and print the color flip count
    public void getColorFlipCount() {
        // Appends the color flip count to the resultString
    }

    // Public method to print books within a given range of book IDs
    public void printBooks(int bookId1, int bookId2) {
        // Prints the details of books within the specified book ID range
    }

    // Public method to find and print the closest book to a target book ID
    public void findClosestBook(int targetId) {
        // Finds and prints the closest book to the target book ID
    }
```

```
    // Public method to borrow a book by a patron
    public void borrowBook(int patronId, int bookId, int patronPriority) {
        // Borrows a book or reserves it based on availability and patron reservations
    }

    // Public method to check if a book is already reserved by a patron
    public boolean alreadyReservedByPatron(int patronId, RedBlackNode book) {
        // Checks if a book is already reserved by a specific patron
    }

    // Public method to return a book by a patron
    public void returnBook(int patronId, int bookId) {
        // Returns a book, updates its availability, and processes reservations if any
    }

    // Public method to quit the program
    public void quit() {
        // Appends a termination message to the resultString and sets the root to null
    }

    // Private method to perform an inorder traversal and populate a list of books within a specified range
    private void inorder(RedBlackNode book, int lower, int upper, List<RedBlackNode> listOfBooks, boolean flag) {
        // Traverses the Red-Black tree in inorder and populates a list of books based on specified criteria
    }

    // Other private utility methods for Red-Black tree operations (insertion, deletion, rotations, etc.)
    // ...

}
```

Time Complexity:

Insertion (insertBook method): O(log N), where N is the number of nodes in the Red-Black Tree.

Deletion (deleteBook method): O(log N), where N is the number of nodes in the Red-Black Tree.

Search (printBook method): O(log N), where N is the number of nodes in the Red-Black Tree.

Overall, the time complexity of Red-Black Tree operations is logarithmic in the number of nodes.

The RedBlackTree class contains the logic for using a Red-Black tree to manage books in the GatorLibrary. It includes techniques for inserting, deleting, searching, and performing various operations on books while maintaining the Red-Black tree's integrity. In addition, the class keeps track of color flip counts and provides result messages for the GatorLibrary Management System.

ReservationNode.java

The ReservationNode class is a node in the MinHeap that holds information about a patron's reservation, such as the patron ID, priority number, and reservation time. This class implements the Comparable interface to allow for priority and reservation time comparisons.

```
import java.util.Date;

public class ReservationNode implements Comparable<ReservationNode> {

    // Instance variables
    private int patronId;
    private int priorityNumber;
    private Date timeOfReservation;

    // Constructor
    public ReservationNode(int patronId, int priorityNumber, Date timeOfReservation) {
        // Initializes a ReservationNode with the provided patron information
    }

    // Empty constructor
    public ReservationNode() {
        // Default constructor
    }

    // Getter method for patron ID
    public int getPatronId() {
        // Returns the patron ID of the ReservationNode
    }

    // Getter method for priority number
    public int getPriorityNumber() {
```

```
    // Returns the priority number of the ReservationNode
  }

  // Getter method for the time of reservation
  public Date getTimeOfReservation() {
    // Returns the time of reservation of the ReservationNode
  }

  // Override toString method for a formatted string representation of the ReservationNode
  @Override
  public String toString() {
    // Returns a formatted string representation of the ReservationNode
  }

  // Override compareTo method to enable comparisons based on priority and reservation time
  @Override
  public int compareTo(ReservationNode o) {
    // Compares two ReservationNodes based on priority and, in case of ties, reservation time
  }

}
```

Time Complexity: O(1)

- The class represents a reservation node and involves basic assignments and comparisons, each taking constant time.

Outputs for following test cases :

testcase1:

BookID = 48

Title = "Data Structures and Algorithms"

Author = "Sartaj Sahni"

Availability = "Yes"

BorrowedBy = None

Reservations = []

Book 48 Borrowed by Patron 120

Book 101 Borrowed by Patron 132

Book 48 Reserved by Patron 144

Book 48 Reserved by Patron 140

Book 48 Reserved by Patron 142

Book 12 Borrowed by Patron 138

Book 12 Reserved by Patron 150

Book 12 Reserved by Patron 162

Book 48 Returned by Patron 120

Book 48 Allotted to Patron 142

BookID = 6

Title = "Database Management Systems"

Author = "Raghu Ramakrishnan"

Availability = "Yes"

BorrowedBy = None

Reservations = []

BookID = 12

Title = "Artificial Intelligence: A Modern Approach"

Author = "Stuart Russell"

Availability = "No"

BorrowedBy = 138

Reservations = [162,150]

Book 12 is no longer available. Reservations made by Patrons 162,150 have been cancelled!

Color Flip Count : 8

Book 73 Borrowed by Patron 111

Book 73 Reserved by Patron 52

Book 25 Borrowed by Patron 153

BookID = 25

Title = "Computer Networks"

Author = "Andrew S. Tanenbaum"

Availability = "No"

BorrowedBy = 153

Reservations = []

BookID = 48

Title = "Data Structures and Algorithms"

Author = "Sartaj Sahni"

Availability = "No"

BorrowedBy = 142

Reservations = [140,144]

BookID = 73

Title = "Introduction to the Theory of Computation"

Author = "Michael Sipser"

Availability = "No"

BorrowedBy = 111

Reservations = [52]

BookID = 101

Title = "Introduction to Algorithms"

Author = "Thomas H. Cormen"

Availability = "No"

BorrowedBy = 132

Reservations = []

BookID = 115

Title = "Operating Systems: Internals and Design Principles"

Author = "William Stallings"

Availability = "Yes"

BorrowedBy = None

Reservations = []

BookID = 125

Title = "Computer Organization and Design"

Author = "David A. Patterson"

Availability = "Yes"

BorrowedBy = None

Reservations = []

BookID = 132

Title = "Operating System Concepts"

Author = "Abraham Silberschatz"

Availability = "Yes"

BorrowedBy = None

Reservations = []

Book 25 Reserved by Patron 171

Book 132 Borrowed by Patron 2

BookID = 48

Title = "Data Structures and Algorithms"

Author = "Sartaj Sahni"

Availability = "No"

BorrowedBy = 142

Reservations = [140,144]

Book 101 Reserved by Patron 18

Book 210 Borrowed by Patron 210

Book 73 Reserved by Patron 43

BookID = 210

Title = "Machine Learning: A Probabilistic Perspective"

Author = "Kevin P. Murphy"

Availability = "No"

BorrowedBy = 210

Reservations = []

Book 210 Reserved by Patron 34

Color Flip Count : 19

Book 125 is no longer available.

Book 115 is no longer available.

Book 210 is no longer available. Reservations made by Patrons 34 have been cancelled!

Color Flip Count : 23

Book 25 is no longer available. Reservations made by Patrons 171 have been cancelled!

Book 80 is no longer available.

Color Flip Count : 25

Program Terminated!!

Testcase2:

Book 5 Borrowed by Patron 101

Book 3 Borrowed by Patron 101

Book 12 is no longer available.

BookID = 3

Title = "The Great Gatsby"

Author = "Mark Johnson"

Availability = "No"

BorrowedBy = 101

Reservations = []

BookID = 5

Title = "The Secret Garden"

Author = "Jane Smith"

Availability = "No"

BorrowedBy = 101

Reservations = []

Book 3 is no longer available.

Book 5 is no longer available.

Book 5 not found in the library

Book 22 Borrowed by Patron 104

Book 22 Reserved by Patron 171

Book 22 Reserved by Patron 103

Color Flip Count : 6

Book 94 Borrowed by Patron 171

Book 22 Returned by Patron 104

Book 22 Allotted to Patron 171

Book 50 Borrowed by Patron 132

Book 10 Borrowed by Patron 103

Book 72 Borrowed by Patron 109

Book 50 Reserved by Patron 101

Book 94 is no longer available.

Book 22 Reserved by Patron 105

BookID = 10

Title = "The Hobbit"

Author = "J.R.R. Tolkien"

Availability = "No"

BorrowedBy = 103

Reservations = []

BookID = 22

Title = "The Alchemist"

Author = "Paul Coelho"

Availability = "No"

BorrowedBy = 171

Reservations = [105,103]

BookID = 28

Title = "Lord of the Flies"

Author = "William Golding"

Availability = "Yes"

BorrowedBy = None

Reservations = []

BookID = 50

Title = "The Catcher in the Rye"

Author = "Michael Brown"

Availability = "No"

BorrowedBy = 132

Reservations = [101]

BookID = 72

Title = "Brave New World"

Author = "Aldous Huxley"

Availability = "No"

BorrowedBy = 109

Reservations = []

BookID = 28

Title = "Lord of the Flies"

Author = "William Golding"

Availability = "Yes"

BorrowedBy = None

Reservations = []

Book 22 Returned by Patron 171

Book 22 Allotted to Patron 105

Book 28 Borrowed by Patron 171

Book 50 is no longer available. Reservations made by Patrons 101 have been cancelled!

Color Flip Count : 13

Book 22 Reserved by Patron 107

Book 10 Returned by Patron 103

Book 10 Borrowed by Patron 121

Book 10 is no longer available.

Book 22 is no longer available. Reservations made by Patrons 103,107 have been cancelled!

Color Flip Count : 15

Program Terminated!!

Testcase3:

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 2 Returned by Patron 102

Book 3 Returned by Patron 103

Book 4 Returned by Patron 104

Book 5 Returned by Patron 105

Book 6 Borrowed by Patron 101

Book 7 Borrowed by Patron 102

Book 8 Borrowed by Patron 103

Testcase4:

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 Allotted to 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 Allotted to Patron 108

Book 11 Borrowed by Patron 112

Book 11 is no longer available.