COP 5536 Fall 2023

Project Report

Name: Manoj Virinchi Chitta
UFID: 75988972
Email: mchitta@ufl.edu

Folder Contents and Program Structure:

- GatorLibrary.java
- MinHeap.java
- ReservationNode.java
- RedBlackTree.java
- Makefile
- projectReport.pdf

ReservationjNode.java

- This class represents the information about thje noes that are being used in Binary min heap. It has three significant variables:
  1**. patronID**- An integer variable that stores the patronId of the person who makes a book reservation.
  2. **priorityNumber** - An integer indicating the importance of a patron's reservation.
  3. **timeOfReservation** - A long containing the timestamp of when the reservation was made.

- This class also contains getters which are used to get the variables that are being used.
- The **toString()** method has been overwritten to provide a string representation of the HeapNode i.e patronID,priorityNumber,timeOfReservation.
- **compareTo()** method has also been overwritten to meet the requirements.

```java
import java.util.*;

public class ReservationNode implements Comparable<ReservationNode> {
    private int patronId;
    private int priorityNumber;
    private Date timeOfReservation;

    public ReservationNode(int patronId, int priorityNumber, Date timeOfReservation) {
        this.patronId = patronId;
        this.priorityNumber = priorityNumber;
        this.timeOfReservation = timeOfReservation;
    }

    public ReservationNode() {

    }

    public int getPatronId() {
        return patronId;
    }

    public int getPriorityNumber() {
        return priorityNumber;
    }

    public Date getTimeOfReservation() {
        return timeOfReservation;
    }

    @Override
    public String toString() {
        return "(" + patronId + ", " + priorityNumber + ", " + timeOfReservation + ")";
    }

    @Override
    public int compareTo(ReservationNode o) {
        if (this.priorityNumber != o.priorityNumber)
            return this.priorityNumber - o.priorityNumber;
        else
            return this.timeOfReservation.compareTo(o.timeOfReservation);
    }

}
```

MinHeap.java

- **MinHeap.java** implements the binary min heap data structure.
- The constructor initializes the MinHeap with a specified capacity. It creates an array of ReservationNode objects to represent the heap.
- It has various methods
    - **public boolean isEmpty():** This checks if the heap is empty or not. If it is empty it will return true else false.
    - **Public int size() :** It will return the size of the min Heap.
    - **Int getLeftChildIdx(int x) :** It will return the left child of the parent node.
    - **Int getRightChildIdx(int parentIndex) :** It will return the right child of the parent at the given index.

- **Int getParentIdx(int child):** Returns the parent of the given child index.
- **ReservationNode leftChild(int parent):** Returns the left child of the given node in the heap array .
- **ReservationNode rightChild(int parent):** Returns the right child of the given node using the heap array.
- **ReservationNode parent(int child) :** Return the parent node of the given child index.
- **ReservationNode peek():** Returns the root or the minimum element in the heap.
- **ReservationNode poll():** Removes and returns the root of the min heap.
- **void insertNode(ReservationNode reservation)** : Insert a new reservation node into the heap.
- **Void heapifyUp ()and void heapifyDown()** : Restores the heapify property by moving a newly added node up or down to bring it to its correct position .
- **Void swap(int index1,int index2)**: swapping two elements in the heap.
- **Void printHeap()** : it will print the min heap.
- **toString()**: this method is overwritten to meet the “print” requirements.

MinHeap.java

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

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

    // Constructor to initialize MinHeap
    public MinHeap(int capacity) {…
    }

    // Check if the MinHeap is empty
    public boolean isEmpty() {…
    }

    // Get the current size of the MinHeap
    public int size() {…
    }

    // Get the index of the left child of a parent node
    public int getLeftChildIdx(int parentIndex) {…
    }

    // Get the index of the right child of a parent node
    public int getRightChildIdx(int parentIndex) {…
    }

    // Get the index of the parent node of a child node
    public int getParentIdx(int childIndex) {…
    }

    // Get the left child of a parent node
    public ReservationNode leftChild(int parentIndex) {…
    }

    // Get the right child of a parent node
    public ReservationNode rightChild(int parentIndex) {…
    }

    // Get the parent node of a child node
    public ReservationNode parent(int childIndex) {…
    }

    // Get the minimum (root) element without removing it from the heap
    public ReservationNode peek() {…
    }

    // Remove and return the minimum (root) element from the heap
    public ReservationNode poll() {…
    }

    // Insert a new reservation node into the heap
    public void insertNode(ReservationNode reservation) {…
    }

    // Restore heap properties from a specific node to the root
    public void heapifyUp() {…
    }

    // Restore heap properties from the root down to a specific node
    public void heapifyDown() {…
    }

    // Swap elements at two indices within the heap array
    public void swap(int index1, int index2) {…
    }

    // Print the elements currently in the heap
    public void printHeap() {…
    }

    // Override toString method to return a string representation of the heap elements
    @Override
    public String toString() {…
    }
}

```

RedBlackTree.java:

- This class represents the red black tree to which various books that are present in the GatorLibrary will be added.
- This class contains a nested class "**RedBlacNode**" . This class represents the nodes that are being used in the red black tree.
    - It has various attributes like : bookId, bookName,authorName,availabilityStatus,borrowedBy,left,right,parent pointers, color, minHeap
- There is also an Enum where the colors i.e Red and Black are declared.
- The following are the various variables and methods that are used in the red black tree implementation.
    - **hm1,hm2,flipCount :** These variables are used to calculate the number of flip counts that happened in total.
    - **public void insertBook(int bookId, String bookName, String authorName, String isAvailable):** This method will insert the book into the red black tree. It uses various helper methods like insert(), fixInsertViolation(), rotateLeft(),rotateRight().
        - **Insert():** This is used to find the position to insert the newNode.
        - **fixInsertViolation() :** This is used to make changes to the redBlackTree by making rotations and color changes. It will use the rotateLeft() and rotateRight() methods to make the rotations.
    - **public void deleteBook(int bookId):** Deletes a book from the Red-Black Tree and handles associated reservations, maintaining tree properties. It uses various methods like fixDeleteViolation,rotateLeft() and rotateRight()
        - **fixDeleteViolation(RedBlackNode x):** This is used to make changes to the red Black Tree after the deletion so that it satisfies the red black tree properties. This method will inturn call the rotateLeft() and rotateRight() methods to make the required rotations at different cases.
    - **public void printBooks(int bookId1, int bookId2):** It will print books that are present between the given two bookId's
    - **public void borrowBook(int patronId, int bookId, int patronPriority):** Handles the borrowing of a book by a patron, updating the book's availability and reservations.
    - **public void returnBook(int patronId, int bookId):** Manages the return of a book by a patron, updating availability and assigning the book to the next patron in the reservation queue if applicable.

RedBlackTree.java

```java
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.util.*;


// declaring red and black using an enum
enum Color {…
}
class RedBlackTree {…
}

// Represents a node in the Red-Black Tree
    public static class RedBlackNode {
    int bookId;
    String bookName;
    String authorName;
    boolean isAvailable;
    int borrowedBy;
    RedBlackTree.RedBlackNode left, right, parent;
    Color color;
    MinHeap minHeap; // For managing reservations associated with this book

    public RedBlackNode() {
    }

    public RedBlackNode(int bookId, String bookName, String authorName, boolean isAvailable) {…
    }

    public RedBlackNode(int bookId) {…
    }

    @Override
    public String toString() {…
    }
}
 // HashMaps to track color changes in nodes during operations…
//insertion into a red black tree
    public void insertBook(int bookId, String bookName, String authorName, String isAvailable) {…
    }

// logic to calculate flip count
    public void colorFlipCount() {…
    }
    public void getColorFlipCount() {…
    }


//helper method
    public void populateHm2() {…
    }

    public void inorderTraversal(RedBlackNode root) {…
    }

    public void transferMap() {…
    }

    public void insert(RedBlackNode book) {…
    }
```

```
// fixing the tree after insertion to satisfy the red-black tree properties
    public void fixInsertViolation(RedBlackNode book) {…
    }

// Rotate Left operation on the red balck tree
    public void rotateLeft(RedBlackNode book) {…
    }

// Rotate right operation on the red black tree
    public void rotateRight(RedBlackNode book) {…
    }

    public RedBlackNode findBook(int bookId) {…
    }

// deletion of the book
    public void deleteBook(int bookId) {…
    }

    private void transplant(RedBlackNode u, RedBlackNode v) {…
    }

    private RedBlackNode treeMinimum(RedBlackNode z) {…
    }

    private RedBlackNode treeMaximum(RedBlackNode z) {…
    }

// fixing the tree after deletion
    public void fixDeleteViolation(RedBlackNode x) {…
    }

// printing books which are between the given two bookID's
    public void printBooks(int bookId1, int bookId2) {…
    }

    public void findClosestBook(int targetId) {…
    }

    public void inorder(RedBlackNode book, int lower, int upper, List<RedBlackNode> listOfBooks, boolean flag) {…
    }

//This method will help you find out if the book is available or if it is borrowed
    public void borrowBook(int patronId, int bookId, int patronPriority) {…
    }

    public boolean alreadyReservedByPatron(int patronId, RedBlackNode book) {…
    }
// method to modify the red black tree when a book is returned
    public void returnBook(int patronId, int bookId) {…
    }

    public void quit() {…
    }

}
```

GatorLibrary.java:

- This class contains the main method, which serves as an entry point to the execution.
- The following are the various methods pressent in the GatorLibrary class:
    - **Public static void main(String args[]):**
        - Reads the input file specified as a command-line argument (args[0]).
        - Creates a BufferedReader to read lines from the input file.
        - Initializes a RedBlackTree object (rbTree) to manage the library operations.
        - Calls the parse() method to parse each line from the input file and performs library operations based on the commands found.
        - After processing all commands, generates an output file using the Output method.
    - **public static void parse(RedBlackTree rbTree, String row, String fileName) throws IOException**:
        - This method parses each line that is present in the input file and perform various operatins on the red black tree.
    - **public static void Output(String fileName) throws IOException:**
        - Creates an outputfile to write output of each operartion that is being performed on the red black tree with the help of a static string builder variable “Output”.

```
import java.io.*;

public class GatorLibrary {
    public static final String COMMA = ",";
    public static final String OPEN_PARENTHESIS = "(";
    public static final String CLOSED_PARENTHESIS = ")";

    public static void main(String[] args) {…
    }

    // Parse each line and perform respective operations based on the GatorLibrary command
public static void parse(RedBlackTree rbTree, String row, String fileName) throws IOException
{…
}
        else if (operation.equals("PrintBook")) {…
}
        else if (operation.equals("PrintBooks")) {…
}
        else if (operation.equals("BorrowBook")) {
    // Borrow a book by a patron, specifying book ID, patron ID, and days to borrow
        rbTree.borrowBook(Integer.parseInt(argArray[0].trim()), Integer.parseInt(argArray[1].trim()),…
}
        else if (operation.equals("ReturnBook")) {…
}
        else if (operation.equals("DeleteBook")) {…
}
        else if (operation.equals("FindClosestBook")) {…
}
        else if (operation.equals("ColorFlipCount")) {…
}
        else if (operation.equals("Quit")) {…
}
        else {
    // If the parsed command is invalid, indicate an invalid operation
    RedBlackTree.resultString.append("Invalid GatorLibrary operation\n");
            }
}

    // Create an output file with the resultString content after all operations are performed
    public static void Output(String fileName) throws IOException {…
    }
}
```