ADVANCED DATA STRUCTURES PROJECT REPORT

Name: Vijay Abhinav Telukunta

UFID: 86262606

UF Email: vtelukunta@ufl.edu

CONTENTS

- Project Overview
- RedBlackTreeNode class
- RedBlackTree class
- MinHeapNode class
- MinHeap class
- Main class (gatorLibrary)

PROJECT OVERVIEW

- The project encompasses a Library Management System designed to efficiently manage book operations and reservations.
- The system is implemented in Java and comprises of five key classes: RedBlackTreeNode, RedBlackTree, MinHeapNode, MinHeap, and gatorLibrary.
- The RedBlackTreeNode class represents a node in a Red-Black Tree. Each node encapsulates information about a book, including its ID, name, author, availability status, borrower information, reservation heap, color of the node and tree-related pointers (left, right, and parent pointers)
- RedBlackTree class handles various operations associated with RedBlackTree nodes such as insert, delete, findClosest etc.
- MinHeapNode class represents nodes within the Min Heap for reservations based on patron priority and timestamps. Each node encapsulates information about patronId, patronPriority and timestamp.
- MinHeap class handles various operations associated with MinHeap nodes such as insert, removeMin, heapify etc.
- Finally, the main class (gatorLibrary) serves as the entry point for the Library Management System and takes the input file, invokes appropriate functions, and writes to output file.

REDBLACKTREENODE CLASS

Class Fields:

- **int bookId:** The unique identifier of the book.
- **String bookName:** The name of the book.
- **String authorName:** The name of the book's author.
- **String availabilityStatus:** The availability status of the book.
- **int borrowedBy:** The ID of the user who borrowed the book.
- **MinHeap reservationHeap:** A min-heap representing reservations for the book.
- **RedBlackTreeNode left:** Pointer to the left child node.
- **RedBlackTreeNode right:** Pointer to the right child node.
- **RedBlackTreeNode parent:** Pointer to the parent node.
- **String color:** The color of the node in the Red-Black Tree.

Class Methods:

public RedBlackTreeNode(int bookId, String bookName, String authorName, String availabilityStatus)

- Initializes a new instance of the RedBlackTreeNode class with the specified book details.

REDBLACKTREE CLASS

Class Fields:

- **String RED:** Represents red in the Red-Black Tree.
- **String BLACK:** Represents black in the Red-Black Tree**.**
- **int colorFlips:** Counts the number of color flips in the Red-Black Tree.
- **Map<Integer, String> oldColorMap:** A map that stores the previous color of nodes based on their IDs.
- **RedBlackTreeNode root:** Reference to the root node of the Red-Black Tree.

Class Methods:

public RedBlackTree()

- The constructor initializes the class field root, setting it to null initially.

public RedBlackTreeNode search(int bookId)

- Searches for a node with the specified book ID in the Red-Black Tree.
- If the node is found in the tree, it returns that redblacktree node otherwise returns null.

public List<RedBlackTreeNode> rangeSearch(int lowBookId, int highBookId)

- Performs a range search on the Red-Black Tree, retrieving nodes with book IDs within the specified range (lowBookId to highBookId).
- It invokes rangeSearchHelper function to help with collecting nodes which are in the specified range and returns list of those redblacktree nodes.

public void rangeSearchHelper(List<RedBlackTreeNode>arr, RedBlackTreeNode node, int lowBookId, int highBookId)

- Assists the range search by recursively traversing the Red-Black Tree and adding nodes within the specified range to the provided list.

public List<RedBlackTreeNode> findClosest(int targetBookId)

- Finds nodes in the Red-Black Tree whose book IDs have the minimum difference from the given target book ID.

- It returns a list of redblacktree nodes whose book IDs have the minimum difference from the target book ID.

public void inorder(List<RedBlackTreeNode> arr, RedBlackTreeNode root)

- Performs an inorder traversal of the Red-Black Tree and populates the provided list with the nodes in ascending order of book IDs.
- This inorder traversal is invoked in find closest to iterate over the nodes to find closest nodes with given target bookId.

public boolean borrowBook(int patronId, int bookId, int patronPriority)

- Handles the borrowing process for a book.
- If the book is available, it is borrowed; otherwise, a reservation heap is created, and the patron is inserted into the heap.

public int returnBook(int patronId, int bookId)

- Handles the process of returning a book.
- It changes the availability status to "Yes" and, if the reservation heap is not empty, allocates the returned book to the top patron in the reservation heap.
- It returns the ID of the patron to whom the returned book is allocated.
- Returns -1 if the reservation heap is empty.

public int colorFlipCount()

- Returns the count of color flips that have occurred in the Red-Black Tree.

public void updateColorFlips()

- Updates the color flip count after every insert and delete operation.
- It compares the colors of nodes before and after the operation and increments the count if a color change is detected.

public void insert(int bookId, String bookName, String authorName, String availabilityStatus)

- Inserts a new node with the specified book details into the Red-Black Tree and maintains the Red-Black Tree properties.

public RedBlackTreeNode findMaximum(RedBlackTreeNode node)

- Finds the node with the maximum book ID in the Red-Black Tree, starting from the given node.

public void fixRBTPropertiesAfterInsert(RedBlackTreeNode node)

- Fixes the Red-Black Tree properties after the insertion of a new node.
- If the parent is null, indicating that the current node is the root, it colors the root black and returns.
- If the parent's color is black, no violation has occurred, and the function returns.
- If the parent's color is red, the function proceeds to handle the cases where the grandparent and uncle exist.
- If the uncle is red, it performs a color flip operation on the parent, grandparent, and uncle.
- If the uncle is black or null, it proceeds to handle rotations based on the positions of the parent, grandparent, and the current node.
- The function continues with rotations and color adjustments until the Red-Black Tree properties are restored.

public void uncleRed(RedBlackTreeNode parent, RedBlackTreeNode grandparent)

- Handles the case where the uncle of a newly inserted node is red. Recolors the grandparent, uncle, and parent nodes, and then recursively calls the fixRBTPropertiesAfterInsert function on the grandparent.

public void grandParentLeftChildParent(RedBlackTreeNode node, RedBlackTreeNode parent, RedBlackTreeNode grandparent)

- Handles the case where the newly inserted node can be any child of its parent, and the parent is a left child of its grandparent.

public void grandParentRightChildParent(RedBlackTreeNode node, RedBlackTreeNode parent, RedBlackTreeNode grandparent)

- Handles the case where the newly inserted node can be any child of its parent, and the parent is a right child of its grandparent.

public RedBlackTreeNode getRBTUncle(RedBlackTreeNode parent)

- Gets the uncle of a given node in the Red-Black Tree. The uncle is the grandparent's other child.

public String delete(int bookId)

- Deletes a node with the specified book ID from the Red-Black Tree and maintains the Red-Black Tree properties.
- It returns a string representation of the patrons (patron IDs) associated with the deleted node to notify them.
- Returns -1 if the node is not found.
- If the node has one or zero children, the deleteZeroOrOneChildNode function is called to handle the deletion and return the node that moved up in the tree. The color of the deleted node is stored.
- If the node has two children, the in-order predecessor (maximum node in the left subtree) is found. The data is copied from the predecessor, and the predecessor node is then deleted using the same logic as for a node with one or zero children.
- If the color of the deleted node is black, the fixRedBlackPropertiesAfterDelete function is called to maintain the Red-Black Tree properties.
- If the moved-up node is a temporary NIL node, it is removed.

public void clone(RedBlackTreeNode node1, RedBlackTreeNode node2)

- Copies all the contents of one node to another. Typically used when the in-order successor needs to be deleted during the deletion operation.

public RedBlackTreeNode deleteZeroOrOneChildNode(RedBlackTreeNode node)

- Handles the deletion of a Red-Black Tree node with either zero or one child.
- It returns the replacement node that moved up in the tree.

public void fixRedBlackPropertiesAfterDelete(RedBlackTreeNode node)

- The fixRedBlackPropertiesAfterDelete function in the Red-Black Tree class plays a crucial role in maintaining the Red-Black Tree properties after the deletion of a node.
- It handles various cases, including those with red or black siblings and at least one red child.
- The function ensures that the Red-Black Tree remains balanced and adheres to the rules of a Red-Black Tree even after deletion operations.

public void lookafterRedChild(RedBlackTreeNode node, RedBlackTreeNode sibling)

- Handles the case where the sibling of a node being deleted is red. Recolors and rotates the nodes to maintain the Red-Black Tree properties.

public void fixAtLeastOneRedChildBlackSibling(RedBlackTreeNode node, RedBlackTreeNode sibling)

- Handles cases during deletion where the sibling of the deleted node has at least one red child and is black.
- Performs recoloring and rotations to maintain Red-Black Tree properties.

public RedBlackTreeNode getRBTNodeSibling(RedBlackTreeNode node)

- Returns the sibling of a given Red-Black Tree node.

public boolean checkNotRed(RedBlackTreeNode node)

- Checks if the given Red-Black Tree node is not red.

public class NilNode extends RedBlackTreeNode

- Represents a special node (NIL node) for temporary purposes in the deletion algorithm. Used when the deleted node is a black leaf.

public void replaceRBTParentsChild(RedBlackTreeNode parent, RedBlackTreeNode oldChild, RedBlackTreeNode newChild)

- Replaces a parent's old child with a new child.
- Updates the parent's child pointer to the new child instead of the old child.
- Updates the parent pointer of the new child.

public void leftRotate(RedBlackTreeNode node)

- Performs a left rotation on the Red-Black Tree.

public void rightRotate(RedBlackTreeNode node)

- Performs a right rotation on the Red-Black Tree.

MINHEAPNODE CLASS

Class Fields:

- **patronId:** Represents the unique identifier of the patron.
- **patronPriority:** Represents the priority assigned to the patron.
- **timestamp:** Represents the timestamp associated with the creation of the MinHeapNode.

Class Methods:

public MinHeapNode(int patronId, int patronPriority, long timestamp)

- Initializes the MinHeapNode with the given patron ID, patron priority, and timestamp.

MINHEAP CLASS

Class Fields:

- **heap:** An array of MinHeapNode instances representing the min heap.
- **size:** The current number of elements in the heap.
- **capacity:** The maximum capacity of the heap.

Class Methods:

void swap(int i, int j)

- Swaps the elements at positions i and j in the heap.

void heapify(int index)

- Restores the min heap property starting from the given index.
- Orders the heap based on patron priority.
- The patrons which have lower patron priority are moved up.
- If the patrons have same priority, ties are broken considering the timestamps.

MinHeap(int capacity)

- Initialize the min heap with the specified capacity**.**

MinHeapNode removeMin()

- Removes the minimum element from the heap and restores the min heap property.

void insert(MinHeapNode newNode)

- Inserts a new node into the heap and restores the min heap property.

String printHeap()

- Returns a string representation of the contents of the heap (patron IDs).

MAIN CLASS (gatorLibrary)

Class Fields:

- **rbTree:** An instance of the RedBlackTree class representing the red-black tree used in the program.
- **outputData:** An ArrayList used to store the output data after each operation.
- **isTerminate:** A boolean flag to check if the termination condition (Quit operation) has been encountered.

Class Methods:

public static void main(String[] args)

- The main function that serves as the entry point of the program.
- It reads the input text file line by line and identifies the operation in each line.
- Maps the operation to appropriate handler functions.
- Finally, it writes the data to output file.

public static void writeToFile(String filename, List<String>list)

- Writes the output data arraylist to a text file.

public static void printBook(String input)

- It invokes the search(bookId) function on redblack tree instance and outputs the data.

public static void printBooks(String input)

- It invokes the rangeSearch(lowBookId, highBookId) function on redblack tree instance and outputs the data.

public static void insertBook(String input)

- It invokes the insert(bookId, bookName, authorName, availabilityStatus) function on redblack tree instance.

public static void borrowBook(String input)

- It invokes the borrowBook(patronId, bookId, patronPriority) function on redblack tree instance.

public static void returnBook(String input)

- It invokes the returnBook(patronId, bookId) function on redblack tree instance and outputs details about allocation of that book to first patron in heap if reservation exists.

public static void deleteBook(String input)

- It invokes the delete(bookId) function on redblack tree instance and outputs the patron list if any reservations exist for it.

public static void findClosestBook(String input)

- It invokes the findClosest(bookId) function on redblack tree instance and outputs it.

public static void colorFlipCount()

- It invokes the colorFlipCount() function on redblack tree instance and outputs it.

public static void quit()

- It terminates the program on quit.
- Sets the isTerminate flag to true.