University of Florida

Department of Computer and Information

Science and Engineering

COP5536 - Advanced Data Structures

Name	UFID	Email ID
Vedant Patil	3948-5964	vedant.patil@ufl.edu

Project Details:

- TreeNode: Represents the data structure which contains all book details to be stored on the red black tree.
- It contains a reservation heap which is a min heap of type UserNode, used to maintain the priority queue for books already allocated to someone else.
- UserNode: Represents the data structure used to store users on the reservation heap. Contains fields of userId, priority and timestamp.
- RedBlack Tree: class with actual implementation of the red black tree with all the functions like insert, delete, print, etc.
- gatorLibrary: Run file, handling the I/O and other aspects of the program.

PrintBook(bookID)	Searches the Red-Black Tree for the given book and prints it if the book is found. [*O(log(n))*]
PrintBook(bookId1, bookId2)	Searches the Red-Black Tree for books in the given range and prints them if any book is found. [*O(log(n)+S)* time]
InsertBook(bookID, bookName, authorName, availabilityStatus, borrowedBy, reservationHeap)	Inserts a book with the given book values in the Red-Black tree. [*O(log(n))* time]
BorrowBook(patronID, bookID, patronPriority)	Allocates the book with bookID to patron with PatronID. If the book is already allocated to some other patron, method adds the patron in the minheap of the book[*O(log(n))* time]
DeleteBook(bookID)	Deletes the book having given bookID. [*O(log(n))* time]
ReturnBook(patronID, bookID)	Allows patron to return the borrowed book and assigns it the next patron on the priority list. [*O(log(n))* time]
FindClosestBook(targetId)	Prints a list of books which are closest to the given bookID. Since we maintain a list of inorderTraversal, we access it in O(N) time.
ColorFlipCount()	Print the cumulative sum of the number of flips each node's color has seen. O(1) since we constantly keep a track of this value using node tracking.

Function to rotate the tree about the node x; Used as helper function while adjusting the tree.

Project Directory:

gatorLibrary.java	*Driver code for the implementation that calls the red-black tree functions as per the input.*
UserNode.java	*Data Structure that stores the user details like userId, priority and timestamp and used as a node in the reservation heap.*
TreeNode.java	*Data Structure to store a red-black tree node with book details and reservation heap.*
RedBlackTree.java	*Builds a red-black tree data structure and includes functions for search, insert, delete, and corresponding red-black tree rotations and transformations.*
PriorityQueue.java	*Builds a priority queue used by the TreeNode*
Makefile	

Time Complexity:

Red-Black Tree Operation	**Time Complexity**	**Space Complexity**
Insert [insert, fixInsert functions]	O(log(n)), where log(n) is the height of the balanced search tree	O(1), no additional space required

Delete [delete using deleteHelper, deleteFixup functions]	O(log(n)), where log(n) is the height of the balanced search tree	O(1), no additional space required
Search [search using searchHelper function]	O(log(n)), where log(n) is the height of the balanced search tree	O(1), no additional space required
leftRotate	O(1) to perform node rotations, there is no recursion	O(1), no additional space required
rightRotate	O(1) to perform node rotations, there is no recursion	O(1), no additional space required
rbTransform	O(1) to perform tree transofrm, there is no recursion	O(1), no additional space required
getRoot	O(1) to just return the root value	O(1), no additional space required

Function prototypes and program structure:

1. **gatorLibrary : Driver class**

Functions:
'public static void main (String args [])' method:

- It takes the input file name as a command line argument, processes the input as per the different operations listed above, and writes the output to the 'input_filename_output_file.txt' file.
- For performing the operations, the main method calls the respective 'PriorityQueue' and 'RedBlackTree' class functions.

- In order to calculate the **ColorFlipCount** the maintains a copy of the tree using the inorderTraversal function before every insert and delete operation and then compares the nodes after the operation to find the ones whose colors are changed. To maintain a copy, the method uses a Hashmap.

2. **UserNode: Class defining the Node Structure of the PriorityQueue**

Constructor:
public UserNode (String userId, int priority, Timestamp timestamp)

- Initializes the priorityQueue node with the book details when a new user requests for the book which is already allocated to someone else.

- When two users share the same priority the conflict is resolved using timestamp.

3. **TreeNode: Class defining the Node Structure of the Red-Black Tree**

Constructor:
public TreeNode(int bookId, String bookName, String authorName, String availability)

- Initializes the red-black tree node with the book details when a new book instance is created for the red-black tree.

- Stores a reservation heap to keep a track of users in the priority queue and in line for the book.

1. **PriorityQueue: Class implementing the Min-Heap Data Structure**

Constructor:
public MinHeap(int size)

- Initializes a ‘HeapNode’ array with a maximum size of 2000(as given in the project description)

Functions:
private int parent(int i)

- Returns the index of the parent node for the node at index i in the HeapNode array

private int leftChild(int i)

- Returns the index of the left child node for the node at index i in the HeapNode array

private int rightChild(int i)

- Returns the index of the right child node for the node at index i in the HeapNode array

private boolean isLeaf(int i)

- Returns true if the node at index i in the HeapNode array is a leaf node, i.e., its right and left child are null.

private HeapNode insert(HeapNode element)

- Inserts book in the min-heap based on priority comparisons, with the minimum priority book being at the root of the min-heap.

- In case of the same book costs of two books, the book having a smaller trip duration is put at the top of the min-heap.

- Returns the inserted HeapNode to set the corresponding red-black tree pointer in it in the driver class 'gatorTaxi'

public HeapNode remove()

- Remove the min-heap's root and replaces it with the last element in the min-heap.

- Performs minheapify operation(described below) to ensure it is a min-heap after the removal and replacement.

- Returns the root of the min-heap to the driver class to print it out.

public HeapNode deleteKey(int i)

- Deletes a specific book having a certain priority from the min-heap.

- Calls the decreaseKey function(described below) and remove() to delete the bubbled-up book from the min-heap.

- Returns the deleted node copy to the driver class to delete the corresponding red-black tree node.

private void minHeapify(int i)

- Compares and swaps non-leaf node at index i with the minimum priority left or right child node, if the priority of any of the children is lesser.

- In case the node has the same cost as its left/right child, then the trip duration of the two nodes is compared.

- If the trip duration of the child node is lesser than the node at index i then the two nodes are swapped

private void decreaseKey(int i, HeapNode new_val)

- Bubbles up the priority having the low key i to the top of the min-heap by replacing the node with the 'new_val' that has 'Integer.MIN_VALUE' bookCost.

private void swap(int x, int y)

- Swaps the two heap nodes present at indices x and y of the HeapNode array

5. **Red-Black Tree: Class implementing the Red-Black Tree Data Structure**

Constructor:
public RedBlackTree()

- Initializes the RedBlackTree instance with a 'TreeNode' root with TNULL i.e., a TreeNode(0,0,0,null), color=black and null left and right children.

Functions:

public void inOrderTraversalofTree(TreeNode root, HashMap<Integer, Integer> colorTree)

- Performs an inorder traversal on the redblack tree and inserts the bookid-color combination in the hashmap. Used for calculating the colorflipcount of the tree.

private TreeNode searchHelper(TreeNode node, int bookId)

- Checks if the **bookId** matches the current root node, if not, it recursively traverses the left/right subtree depending on if the **bookId** is lesser or greater than the current node value.

- Returns the TreeNode found for the given **bookId** or returns null if no TreeNode is found to the calling search method

private void deleteFixup(TreeNode x)

- Balances the red-black tree after the deletion of a book from the tree considering the various red-black tree constraints.

private void rbTransform(TreeNode node1, TreeNode node2)

- Transform the red-black tree to satisfy the balanced red-black tree constraints.

private void deleteHelper(TreeNode node, int bookId)

- Helper method to delete the **book** for the given **bookId** from the red-black tree.

private void insertFixup(TreeNode k)

- Balances the red-black tree after the insertion of a **book** into the tree taking into account the various red-black tree constraints.

public search(int bookId)

- Calls a searchHelper function to recursively search for the given **bookId** in the tree

- Returns the TreeNode found for the given **bookId** or returns null if no TreeNode is found to the driver class

public TreeNode minimum(TreeNode node)

- Returns the minimum node linked to the passed node i.e., the left node

public void findNodesInRange(TreeNode curr, int bookId1, int bookId2)

- Recursively searches for nodes in the given **bookId** range.

- Writes all the nodes that fall in the range to the output file.

public ArrayList<Integer> printClosest(int bookId)

- calculates the nodes closest to the given bookId in either directions.

public void leftRotate(TreeNode x)

- Performs a left rotation to balance the red-black tree after insertions and deletions.

public void rightRotate(TreeNode x)

- Performs a right rotation to balance the red-black tree after insertions and deletions.

public TreeNode insert(int bookId, String bookName, String authorName, String availability)

- Inserts a new TreeNode into the red-black tree with the given book details.

- Returns the inserted TreeNode to the driver class.

public TreeNode getRoot()

- Returns the root node in the red-black tree to the driver class.

public void deleteNode(int bookId)

- Calls the deleteHelper method to delete the node having the given bookId from the red-black tree.