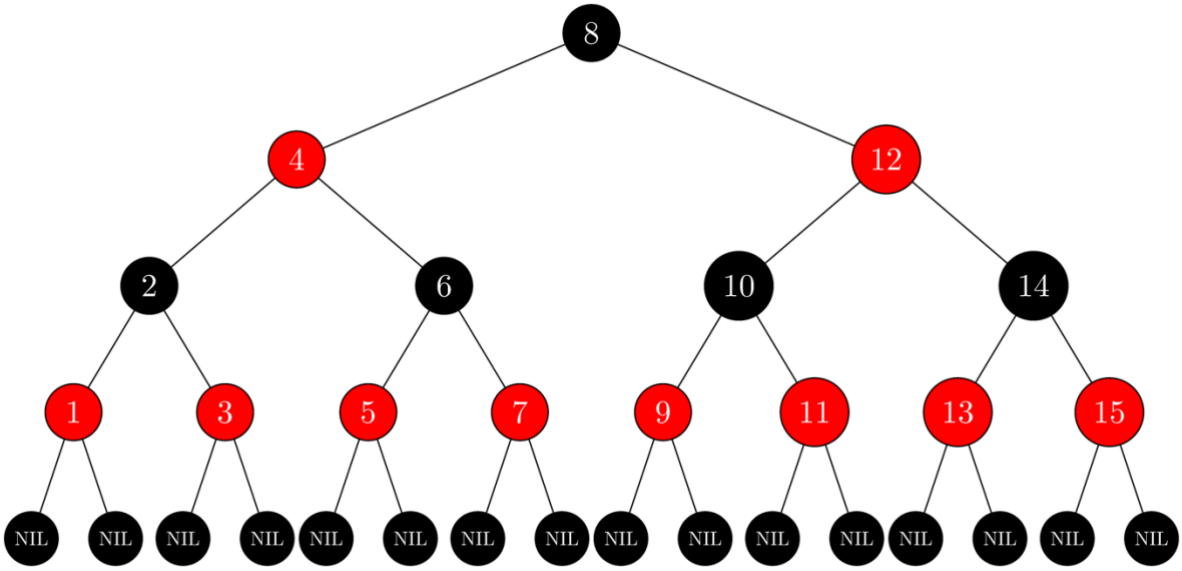
Advanced Data Structures

COP5536, Fall 2023

Programming Project Report

RED BLACK TREES & MIN-HEAP



NAME: HARSHIT LOHAAN
UFID: 7615-8695
EMAIL: H.LOHAAN@GMAIL.COM

Under The Guidance Of
DR. SARTAJ SAHNI

Project Objective:

A library manage system called GatorLibarary is implemented to efficiently manage its patrons, borrowing operations and books overall. This implementation utilizes Red Black data structure for effective management. A priority-queue is also implemented using binary min-heaps for maintaining book reservations. Each book has its respective priority queue to keep track of the reservations.

Function Prototypes and code structure:

Following is the Book class structure:

- `BookId`: An integer representing the ID of the book.
- `BookName`: A string representing the name of the book.
- `AuthorName`: A string representing the name of the author of the book.
- `AvailabilityStatus`: A boolean indicating the availability status of the book. True means the book is available, and False means it is borrowed.
- `BorrowedBy`: A string representing the ID of the patron who borrowed the book.
- `ReservationHeap`: A list representing the reservations made for the book. Each reservation is a tuple containing the patron ID, priority number, and time of reservation.

The Book class also has the following functions:

- `__init__(self, BookId=0, BookName="", AuthorName="", AvailabilityStatus=True, BorrowedBy=None)`: The constructor method that initializes a Book object with the given attributes.
- `__str__(self)`: Returns a string representation of the Book object.
- get_reservation_list(self): Returns a list of patron IDs who have reserved the book.
- `add_reservation(self, patron_id, priority_number, time_of_reservation)`: Adds a reservation for the book with the given patron ID, priority number, and time of reservation.
- `_heapify_up(self)`: Performs the heapify up operation to maintain the heap property of the reservation heap.
- `_compare_reservations(self, index1, index2)`: Compares two reservations based on priority number and time of reservation.
- `_swap(self, index1, index2)`: Swaps two reservations in the reservation heap.

These functions are used to initialize a Book object, retrieve information about the book, add reservations, and maintain the reservation heap.

Following are some of the important data members and all member functions for the implementation of Red-black tree.

Data Members:

- `root`: This is a reference to the root node of a Red-Black Tree data structure.

- `color_flip_count`: This is an integer that keeps track of the number of color flips performed in the Red-Black Tree.
- `currentFunctionId`: This is an identifier that represents the current function being executed.

Member Functions:

- `__init__(self)`: This is the constructor method of the gatorLibrary class. It initializes the root node of the Red-Black Tree and sets the color_flip_count and currentFunctionId to 0.
- `ReturnBook(self, book_id, patron_id)`: This function is used to return a book borrowed by a patron. It takes two parameters, book_id and patron_id, and returns a message indicating the success or failure of the return operation.
- `FindClosestBook(self, target_id)`: This function finds the closest book(s) to a given target book ID in the Red-Black Tree. It takes a target_id parameter and returns a string representation of the closest book(s) to the target book ID.
- `FindClosestBookHelper(self, node, target_id)`: This is a helper function that recursively finds the closest book(s) to a given target book ID in the Red-Black Tree. It takes two parameters, node and target_id, and returns a list of the closest book(s) to the target book ID.
- `InorderPredecessorSuccessor(self, target_id)`: This function finds the inorder predecessor and successor of a given target book ID in the Red-Black Tree. It takes a target_id parameter and returns the inorder predecessor and successor as separate variables.
- `ColorFlipCount(self)`: This function returns the count of color flips performed in the Red-Black Tree.
- `ChangeNodeColor(self, node, new_color)`: This function changes the color of a given node in the Red-Black Tree. It takes two parameters, node and new_color, and updates the color of the node accordingly.
- `Quit(self)`: This function terminates the program and returns a message indicating the termination.
- `ChangeNodeColor(w, color)`: This function is used to change the color of a given node w to the specified color.
- `RightRotate(y)`: This function performs a right rotation on the tree rooted at node y. It restructures the tree in a way that the left child of y becomes the new root, y becomes the right child of the new root, and the right child of the new root becomes the left child of y.
- `LeftRotate(x)`: This function performs a left rotation on the tree rooted at node x. It restructures the tree in a way that the right child of x becomes the new root, x becomes the left child of the new root, and the left child of the new root becomes the right child of x.
- `DeleteFixup(x)`: This function is used to fix the red-black tree properties after a node x has been deleted. It performs various rotations and color changes to maintain the balance and properties of the red-black tree.
- `Transplant(u, v)`: This function replaces the subtree rooted at node u with the subtree rooted at node v. It updates the parent-child relationships accordingly.

- `TreeMaximum(x)`: This function returns the node with the maximum key value in the subtree rooted at node x. It traverses the right child of each node until it reaches the rightmost leaf node.
- `InsertBook(book_id, book_name, author_name, availability_status, borrowed_by, reservation_heap)`: This function is used to insert a new book into the red-black tree. It creates a new Book object with the given details, creates a new RBTreeNode with the Book object, and inserts it into the tree using the Insert function.
- `DeleteBook(book_id)`: This function is used to delete a book from the red-black tree based on its book_id. It searches for the node with the given book_id, deletes it using the Delete function, and returns a message indicating the status of the deletion.
- `SearchBookNode(node, book_id)`: This function searches for a node with the given book_id in the red-black tree. It starts from the root node and traverses the tree based on the comparison of book_id values until it finds a matching node or reaches a leaf node.
- `PrintBook(book_id)`: This function prints the details of a book with the given book_id. It searches for the node with the given book_id using the SearchBookNode function and returns the string representation of the book.
- `PrintBooks(book_id1, book_id2)`: This function prints the details of all the books in the red-black tree whose book_id falls within the range of book_id1 and book_id2. It uses the GetBooksInRange function to retrieve the books and returns a formatted string containing the details of the books.
- `GetBooksInRange(node, book_id1, book_id2)`: This function returns a list of books in the red-black tree whose book_id falls within the range of book_id1 and book_id2. It recursively traverses the tree and adds the books to the list based on the comparison of book_id values.
- `BorrowBook(patron_id, book_id, patron_priority)`: This function allows a patron to borrow a book from the library. It searches for the node with the given book_id using the SearchBookNode function. If the book is available, it updates the availability status and the borrower information. If the book is already borrowed, it adds a reservation for the patron. It returns a message indicating the status of the borrowing operation.

Steps to run:

1. Unzip Harshit_Lohaan.zip and open the folder.
2. Run terminal in that folder and run one of the following commands:
   a. python gatorLibrary.py <input_file.txt>
   b. python3 gatorLibrary.py <input_file.txt>
3. Find the output in `<input_file>_output_file.txt`