Advanced Data Structures
COP 5536

Programming Project Report

By -
Riddhish Harshal Thakare
21533553
riddhishthakare@ufl.edu

Under Guidance of -
Dr. Sartaj Sahni

Project Description:

GatorLibrary is a fictional library that needs a software system to efficiently manage its books, patrons, and borrowing operations.
The system should utilize a Red-Black tree data structure to ensure efficient management of the books. Implement a priority-queue mechanism using Binary Min-heaps as a data structure for managing book reservations in case a book is not currently available to be borrowed. Each book will have its own min-heap to keep track of book reservations made by the patrons.

Each node in the Red-Black tree will represent a book and will have the following structure:
BookId // Integer ID
BookName //Name of the
AuthorName //Name of the
AvailabilityStatus //To indicate whether it is currently
BorrowedBy //ID of the Patron who borrowed the
ReservationHeap: Implement Binary Min-heap for managing book reservations and waitlist for the book ordered by the patron's priority which is an integer

The system should support the following operations:
1. PrintBook(bookID): Print information about a specific book identified by its unique bookID (e.g., title, author, availability status).

2. PrintBooks(bookID1, bookID2): Print information about all books with bookIDs in the range [bookID1, bookID2].
3. InsertBook(bookID, bookName, authorName, availabilityStatus, borrowedBy, reservationHeap): Add a new book to the library. BookID should be unique, and availability indicates whether the book is available for borrowing.
4. BorrowBook(patronID, bookID, patronPriority): Allow a patron to borrow a book that is available and update the status of the book. If a book is currently unavailable, create a reservation node in the heap as per the patron's priority (patronPriority).
5. ReturnBook(patronID, bookID): Allow a patron to return a borrowed book. Update the book's status and assign the book to the patron with highest priority in the Reservation Heap. *(if there's a reservation)*.
6. DeleteBook(bookID): Delete the book from the library and notify the patrons in the reservation list that the book is no longer available to borrow.
7. FindClosestBook(targetID) : Find the book with an ID closest to the given ID (checking on both sides of the ID). Print all the details about the book. In case of ties, print both the books ordered by bookIDs.
8. ColorFlipCount(): GatorLibrary's Red-Black tree structure requires an analytics tool to monitor and analyze the frequency of color flips in the Red-Black tree. Track the occurrence of color changes in the Red-Black tree nodes during tree operations, such as insertion, deletion, and rotations.

Implementation:

Technologies Used:

- Python
- Visual Studio Code

Steps to run:

- Unzip Riddhish_Thakare.zip
- Run - python3 gatorLibrary.py <input_filename>
- Find the output generated by output_input_filename.txt

Code Structure:

There are 3 files in the folder:

gatorLibrary.py - Main file containing the overall project logic and execution flow.
redblacktree.py - Implementation of a Red-Black Tree data structure.
binaryminheap.py - Implementation of a Binary Min Heap data structure.

gatorLibrary.py invokes both redblacktree.py and binaryminheap.py.
redblacktree.py invokes binaryminheap.py.

Now let's go through each one of them:

1. gatorLibrary.py

Class GatorLibrary:

The GatorLibrary class has several functions that provide various operations on a library system. Here is a brief description of each function:

__init__(self): The constructor initializes the GatorLibrary class with a Red-Black Tree (books_rb_tree) and a Min Heap (reservation_heap) to manage book information and reservations.

readInput(self, file_path): This function reads the content of a file specified by file_path and returns the lines as a list. It handles file not found errors and other exceptions.

PrintBook(self, book_id): Prints the details of a book with the given book_id, including title, author, availability status, reservations, and borrower information if applicable.

PrintBooks(self, book_id1, book_id2): Prints details of all books within the specified range of book_id1 to book_id2.

_print_books_range_helper(self, node, book_id1, book_id2): A helper function for PrintBooks that recursively prints books within the specified range.

InsertBook(self, book_id, book_name, author_name, availability_status=True, borrowed_by=None, reservation_heap=None): Inserts a new book into the library with the provided details. Checks for duplicate book IDs before insertion.

BorrowBook(self, patron_id, book_id, patron_priority): Allows a patron with patron_id to borrow a book with book_id. Handles book availability and manages reservations.

ReturnBook(self, patron_id, book_id): Allows a patron with patron_id to return a borrowed book with book_id. Updates book status and assigns the book to the next patron in the reservation list if applicable.

DeleteBook(self, book_id): Deletes a book with the given book_id from the library. Notifies patrons with reservations about the book's unavailability.

FindClosestBook(self, target_id): Finds the book with the closest book_id to the given target_id and prints its details. Handles cases where multiple books have the same closest distance.

Quit(self): Terminates the program when called. Prints a termination message.

ColorFlipCount(self): Prints and returns the count of color flips in the Red-Black Tree during its operations.

determine_actual_close(self, target_id, book_count): Helper method to find the actual closest books to a target book ID (target_id) from a list of book IDs (book_count).

notify_patrons(self, book_id): Notifies patrons about a book's unavailability, canceling reservations if applicable, based on Red-Black Tree search results.

Main:

The code block enclosed by if __name__ == "__main__": initializes a GatorLibrary object named library and then reads a file path from the command line arguments. It uses the readInput method of the library object to read the content of the specified file. The output is redirected to a file named '{name}_output_file.txt', where 'name' is derived from the input file path. The script then iterates through each line of the file, prepends 'library.' to each line, and executes the resulting command using the exec function. This effectively performs a sequence of operations on the library object based on the commands specified in the input file, and the results are written to the output file. The exec function is used to dynamically execute the commands provided in the input file.

2. redblacktree.py

Class Node:

The Node class represents a node in a red-black tree and is primarily used for organizing and maintaining the structure of the tree. The __init__ function initializes a node with a reference to a Book object, sets its parent, left child, right child, and color (1 for red, 0 for black). This class is integral for the implementation of the red-black tree, ensuring properties like balanced height and efficient search operations.

Class Book:

The Book class represents a book in a library system. The __init__ function initializes a book with essential attributes such as book_id, book_name, author_name, availability_status (defaulted to True), a list of borrowed_by users (defaulted to an empty list), and a reservation_heap (defaulted to a MinHeap if not provided). The __str__ function provides a string representation of the book, including its ID, title, author, and availability status. This class encapsulates the information and functionalities related to individual books, including their reservation status and borrower information.

Class RedBlackTree:

__init__(self): Initializes the red-black tree with a sentinel node (TNULL) representing null. The root is set to TNULL, and the color flip count is initialized to zero.

search_tree_helper(self, node, key): Recursively searches the tree for a node with the given key. Returns the node if found; otherwise, returns TNULL.

fix_delete(self, x): Restores the red-black properties of the tree after a node deletion. It performs rotations and color adjustments as needed.

fix_insert(self, k): Maintains the red-black properties after a node insertion. It performs rotations and color adjustments to ensure the tree remains balanced.

left_rotate(self, x): Performs a left rotation around the given node x to maintain the red-black properties.

right_rotate(self, x): Performs a right rotation around the given node x to maintain the red-black properties.

insert(self, key, book): Inserts a new node with the given key and book into the red-black tree and calls fix_insert to balance the tree.

get_min_value_node(self, node): Returns the node with the minimum key value in the subtree rooted at the given node.

get_max_value_node(self, node): Returns the node with the maximum key value in the subtree rooted at the given node.

delete_node(self, key): Deletes the node with the specified key from the red-black tree and calls fix_delete to maintain balance.

delete_node_helper(self, root, key): Helper function for node deletion, identifying the node to be deleted and handling the various cases.

transplant(self, u, v): Replaces subtree rooted at node u with the subtree rooted at node v.

find_closest_books(self, target_id): Finds the closest lower and higher nodes to the given target_id and returns their book information.

find_closest_lower(self, node, target_id): Finds the node with the largest key less than the target_id.

find_closest_higher(self, node, target_id): Finds the node with the smallest key greater than the target_id.

3. binaryminheap.py

Minheapnode:

The MinHeapNode class represents a node in a minimum heap data structure, commonly used in priority queue implementations. The __init__ method initializes a MinHeapNode object with two attributes: priority and value. The priority attribute denotes the priority of the node within the heap, which is essential for maintaining the heap ordering property. Nodes with lower priority values will have higher precedence in the heap. The value attribute holds the actual data associated with the node.

MinHeap:

The MinHeap class is designed to implement a minimum binary heap, a data structure where the value of each node is less than or equal to the values of its children. Here are the functionalities of each function within the class:

__init__(self): This is the constructor method that initializes the heap as an empty list.

insert(self, priority, value): This function inserts a new node with the given priority and value into the heap. It creates a MinHeapNode object, appends it to the heap, and then ensures the heap property is maintained by calling _heapify_up to move the new node to its correct position.

extract_min(self): This function removes and returns the node with the minimum priority (the root of the heap). If the heap is empty, it returns None. It maintains the heap property by replacing the root with the last node and then calling _heapify_down to move the node to its correct position.

is_empty(self): This simple function returns True if the heap is empty and False otherwise.

_heapify_up(self, index): This method is used to restore the heap property by moving a node up the heap until its priority is greater than or equal to its parent's priority.

_heapify_down(self, index): This method restores the heap property by moving a node down the heap until its priority is less than or equal to both of its children's priorities.

get_heap_elements(self): This function returns a list containing the values of all nodes in the heap. It provides a way to inspect the elements of the heap without modifying its structure.

Difference in color flip count values.
My implementation -

To analyze the correctness of your color flip count implementation in a red-black tree, let's go through the key operations in both `fix_insert` and `fix_delete` methods where you are updating the color flip count.

`fix_insert` Method:
1. Case 1 - Uncle is Red:
   - If the uncle of the current node is red, you perform a series of color flips to maintain the properties of a red-black tree.
   - I increment the color flip count after each color flip operation.

2. Case 2 - Uncle is Black:
   - If the uncle is black, you again perform rotations and color flips to balance the tree.
   - I increment the color flip count after each color flip operation.

`fix_delete` Method:
1. Case 1 - Sibling is Red:
   - If the sibling of the current node is red, you perform color flips and rotations to address this case.
   - I increment the color flip count after each color flip operation.

2. Case 2 - Sibling and Nephews are Black:
   - If both the left and right nephews of the sibling are black, you perform a color flip.
   - I increment the color flip count in this case as well.

3. Case 3 - Right Nephew is Black:
   - If the right nephew of the sibling is black, you perform additional rotations and color flips.
   - I increment the color flip count after each color flip operation.

These color flips are crucial for maintaining the balance and properties of the red-black tree. Hence my color flips might differ slightly from the values given in the test case pdf.

For the test case:

```
InsertBook(101, "Introduction to Algorithms", "Thomas H. Cormen", "Yes")
InsertBook(48, "Data Structures and Algorithms", "Sartaj Sahni", "Yes")
PrintBook(48)
InsertBook(132, "Operating System Concepts", "Abraham Silberschatz", "Yes")
InsertBook(25, "Computer Networks", "Andrew S. Tanenbaum", "Yes")
BorrowBook(120, 48, 2)
BorrowBook(132, 101, 1)
InsertBook(73, "Introduction to the Theory of Computation", "Michael Sipser", "Yes")
InsertBook(12, "Artificial Intelligence: A Modern Approach", "Stuart Russell", "Yes")
InsertBook(6, "Database Management Systems", "Raghu Ramakrishnan", "Yes")
BorrowBook(144, 48, 3)
BorrowBook(140, 48, 3)
BorrowBook(142, 48, 2)
BorrowBook(138, 12, 4)
BorrowBook(150, 12, 3)
BorrowBook(162, 12, 1)
ReturnBook(120, 48)
FindClosestBook(9)
DeleteBook(12)
ColorFlipCount()
InsertBook(125, "Computer Organization and Design", "David A. Patterson", "Yes")
InsertBook(180, "Introduction to Software Engineering", "Ian Sommerville", "Yes")
BorrowBook(111, 73, 3)
BorrowBook(52, 73, 1)
InsertBook(115, "Operating Systems: Internals and Design Principles", "William Stallings",
"Yes")
BorrowBook(153, 25, 2)
PrintBooks(10, 150)
InsertBook(210, "Machine Learning: A Probabilistic Perspective", "Kevin P. Murphy", "Yes")
BorrowBook(171, 25, 3)
BorrowBook(2, 132, 2)
FindClosestBook(50)
BorrowBook(18, 101, 2)
InsertBook(80, "Software Engineering: A Practitioner's Approach", "Roger S. Pressman", "Yes")
BorrowBook(210, 210, 1)
BorrowBook(43, 73, 1)
InsertBook(60, "Introduction to Computer Graphics", "David F. Rogers", "Yes")
PrintBook(210)
InsertBook(4, "Design Patterns: Elements of Reusable Object-Oriented Software", "Erich Gamma",
"Yes")
InsertBook(2, "Introduction to the Theory of Computation", "Michael Sipser", "Yes")
BorrowBook(34, 210, 2)
InsertBook(65, "Computer Networks: Principles, Protocols, and Practice", "Olivier
Bonaventure", "Yes")
ColorFlipCount()
DeleteBook(125)
DeleteBook(115)
DeleteBook(210)
ColorFlipCount()
DeleteBook(25)
DeleteBook(80)
ColorFlipCount()
Quit()
```

Colour Flip count values -
Pdf Test Case values

1. Color Flip Count: 8
2. Color Flip Count: 19
3. Color Flip Count: 23

4. Color Flip Count: 27

My values

1. Color Flip Count: 8
2. Color Flip Count: 22
3. Color Flip Count: 24
4. Color Flip Count: 28