Advanced Data Structures

Gator Library Management System

Dhanush Paruchuri

UFID:- 3189-3822

UF email :- dhanushparuchuri@ufl.edu

Problem Description:

Designed for the specific needs of the Gator Library, an extensive network of libraries with an extensive book collection, the Gator Library Management System is a Python-based tool. Optimal performance is ensured through the incorporation of advanced data structures into the program, which facilitates efficient and effective management of books within libraries. An overview of the program's architecture and functionality is provided in this report, emphasizing how it addresses a library's operational requirements.

1. Class Node1 (Book):

- Purpose: Represents a book in the library, encapsulating its attributes and status.
- Methods:

1. `__init__(self, book_id, book_name, author_name, availability_status, borrowed_by, reservation_heap)`: Various parameters are used to create a new instance of a book.

- `__repr__(self)`: For debugging and logging purposes, provides a string representation of a Book object.

2. Class MinHeap:

- Purpose: Manages reservations using a minimum heap data structure.
- Methods:

- `__init__(self)`: Initializes an empty heap.
- `insert(self, z)`: Maintains the min-heap property and inserts an element into the heap.
- `extract_min(self)`: The smallest element in the heap is removed and returned.
- `heapify(self, index)`: Maintain the min-heap property with this helper method.
- `bubbleup(self, index)`: The heap can be adjusted upward with this helper method.

3. Class RedBTree:

- Purpose: Implements a Red-Black Tree data structure for efficient book management.

- Methods:

- `__init__(self)`: Initializes an empty Red-Black Tree.

- `min(self, node)`: Finds the node with the minimum value in the subtree.

- `trans(self, u, v)`: A subtree rooted at node v is transplanted into the position of the subtree rooted at node u.

- delete(self, z): It maintains the properties of a node z when it is deleted from the Red-Black Tree.

- `delete_fixup(self, x)`: Adjusts the tree after deletion to maintain Red-Black Tree properties.

- `flip_color(self, node)`: Flips the color of a node and increments the fixup counter.

- `insert(self, node)`: Inserts a node into the Red-Black Tree and fixes the tree to maintain its properties.

- `fix_insert(self, node)`: Fixes the tree after insertion to maintain Red-Black Tree properties.

- `leftrotate(self, x)`: Performs a left rotation around a given node.

- `rightrotate(self, y)`: Performs a right rotation around a given node.

- `print_books_range(self, book_id1, book_id2)`: Prints details of books in a specified range of IDs.

- `_print_books_range(self, node, book_id1, book_id2, booklist)`: Helper method to traverse the tree and collect book details in a specified range.

- `search(self, node, book_id)`: Searches for books by their ID in the tree.

4. Class GatorLibrary:

- Purpose: Provides the main interface for executing commands and managing library operations.
- Methods:

  1. `__init__(self)`: The library is initialized with a Red-Black Tree to store book data and a color flip counter.

  2. `colorflip(self)`: Increments the color flip count.

  3. `filecommands(self, input_filename)`: The specified input file is read and a list of commands is returned.

4.`outputtofile(self, output_filename, output_lines)`: The specified output file is written with the given output lines.

5. `run(self, command)`: Parses and executes a given command string, handling various library operations.

6.`insert_book(self, book_id, book_name, author_name, availability_status)`: Inserts a new book into the library.

7.`print_book(self, book_id)`: Prints details of a single book.

8.`print_books(self, book_id1, book_id2)`: Prints details of books in a specified range.

9.`borrow_book(self, patron_id, book_id, patron_priority)`: Handles borrowing a book.

10. `return_book(self, patron_id, book_id)`: Handles returning a book.

11. `find_closest_book(self, target)`: Finds the closest book by ID.

12.`delete_book(self, book_id)`: Deletes a book from the library.

5. Function main(file_name):

- Purpose: Drives the library management system based on the input from an input file.

6. Function safe_int_convert(value):

- Purpose: Safely converts a string to an integer, handling errors.

An object-oriented programming approach ensures modularity and scalability of the Gator Library Management System. There is a clear separation of concerns between each class, allowing the codebase to be intuitive and adaptable to future enhancements. With the design, the intricacies of library management are effectively addressed, including book operations, patron interactions, and catalog management. Book management is further enhanced by the use of a Red-Black Tree. A robust and adaptable library management system, the Gator Library Management System is well-organized and efficient, leveraging the principles of object-oriented programming.