GatorLibrary Management system

Programming Project

Tues, Nov 21st 2023

Course:
COP 55336 Fall 2023

Report By:
Abdul Kalam Azad Shaik
shaik.abdulkalam@ufl.edu
14960758

Overview

The project's primary goal was to create a robust system capable of handling various library tasks, including the management of books, patrons, and borrowing activities, with efficiency and reliability.

Key features of the system include:

1. Efficient management of book records using a Red-Black Tree data structure.
2. Handling of book reservations and prioritisation through a custom min-heap.
3. Functionalities for borrowing and returning books, along with dynamic availability tracking.
4. An innovative feature to find the closest book by ID, enhancing user experience.

One of the significant technical accomplishments of this project was the implementation of the Red-Black Tree and the custom min-heap, which posed unique algorithmic challenges. These data structures were pivotal in achieving high performance and reliability in book management and reservation tasks.

Developed using Python, the project leveraged some of its default libraries. This report documents the development process, challenges encountered, solutions implemented, and the overall success of the project in meeting its intended objectives.

Execution Steps

This section outlines the necessary steps to set up and run the GatorLibrary Management System:

1. Make sure we have Python 3.8 or later for ease of execution.
2. We don't have any additional dependencies as a requirement.
3. Please extract the zip file that we can download from the canvas.
4. Run the following command: `**python3 gatorLibrary.py <inputFileName.txt>**`
5. It generates the respective output file, named as `**inputFileName_output_file.txt**`

Project structure

The GatorLibrary Management System comprises the following key components:

1. gatorLibrary.py : The main script that interprets and executes library operations based on input commands.
2. red_black_tree.py : Implements the Red-Black Tree for efficient management of books in the library.
3. book.py : Defines the `Book` class, each instance of which represents a book in the library. This class includes methods for reservation handling.
4. min_heap.py : Implements the MinHeap class, which is used in the reservation system to manage and prioritise book reservations.

Classes and Function Prototypes:

- Book Class (book.py):
    - Represents an individual book in the library.
    - Attributes: book_id, book_name, author_name, availability_status, reservation_heap (a min-heap for managing reservations).
    - Key Methods:

        - add_reservation(self, patron_id, priority): Adds a patron's reservation to the book's reservation heap.
        - get_next_reservation(self): Retrieves the next reservation from the heap, prioritising by reservation time and patron priority.

- RedBlackTree Class (red_black_tree.py):
    - Manages the book records using a Red-Black Tree structure.
    - Attributes: root (root of the tree), TNull (Null nodes), color_flip_count (variable to store the all the colour flips performed at various stages)
    - Key Methods:
        - insert_book(self, book_id, book_name, author_name): Adds a new book to the library.
        - delete_book(self, book_id): Removes a book from the library and handles associated reservations.
        - borrow_book(self, patron_id, book_id, patron_priority): Manages borrowing of books and adds reservations if necessary.
        - return_book(self, patron_id, book_id): Handles the return of borrowed books and assigns the book to the next patron in the reservation queue if applicable.
        - find_closest(self, target_id): finds the books closest with book id to the given target_id. In case of ties returns both the nodes.
        - Various printing methods as per the requirements.

- MinHeap Class (min_heap.py):
    - Implements a min-heap structure for handling book reservations.
    - Attributes: a list (or array) of tuple (priority, timestamp, patron_id)
    - Key Methods:
        - insert(self, patron_id, priority): Adds a new reservation to the heap. Timestamp is calculated at the time of insert, inserted accordingly.
        - pop(self): Removes and returns the reservation with the highest priority (lowest numerical value), used when a book becomes available.

- Flow of the project:
    - Whenever an insert operation is performed we create a book accordingly.
    - That also means, we create an empty min heap for every book we created.
    - When a borrow request comes, we check for the availability status and insert elements into the heap of the respective book accordingly.
    - Also, when we delete the book and the entire reservations and other attributes related to the book.
    - We also update the colour flip count of the tree, whenever we perform a insert and delete operations accordingly.
    - The insert and delete methods are carried out in accordance with the methods covered in class or on the reference slides.

- Other system packages used:
    - Firstly, the `re` package, which supports regular expressions, was instrumental in parsing command inputs from text files. Its powerful pattern matching capabilities allowed us to accurately extract commands and their associated arguments, even when these arguments included complex structures, such as strings containing commas. This feature was crucial in interpreting user inputs reliably and flexibly.
    - Secondly, we employed the `time` package, particularly its time.time() function, to generate timestamps. These timestamps played a key role in our reservation system. By using these timestamps, we were able to prioritise reservations not only based on a patron's priority level but also on the order in which reservations were made, thus ensuring fairness and efficiency in book allocations.

Testing and results

The program is tested on various test cases provided in the project description and also the additional test case provided later. I was glad to see that project performing well on all the given test cases by exactly matching the output of the program with the given expected outputs.