Name: Sadaf Tabassum Shaik

Contents:

Overview:

This report provides an overview of the GatorTicketMaster project, a ticket reservation system implemented in Python. The project utilizes red black tree, min heap and max heap data structures to manage seat reservations, waitlists, and user priorities efficiently.

Program Structure:

The GatorTicketMaster system has the following classes:

- Node: Represents a node in Red Black tree.
- RedBlackTree: Implements a Red Black tree for efficient storage and retrieval of reservations.
- MinHeap: The MinHeap class is an implementation of a binary min heap data structure.
- MaxHeap: The MaxHeap class implements a binary max heap data structure
- GatorTicketMaster: The main class that integrates all components and provides the core functionality

Function Prototypes:

Node Class

- __init__(self, key, value, color="RED"): Initializes a node with a key, value, and color.

RedBlackTree Class

- __init__(self): Initializes an empty Red-Black Tree.
- insert(self, key, value): Inserts a new key-value pair into the tree.
- delete(self, key): Removes a node with the given key from the tree.

- search(self, key): Searches for a value associated with the given key.
- inorder_traversal(self): Returns an inorder traversal of the tree.

MinHeap Class

- __init__(self): Initializes an empty min heap.
- insert(self, item): Adds a new item to the heap.
- extract_min(self): Removes and returns the minimum element.
- _sift_up(self, i): Moves an element up to maintain the heap property.
- _sift_down(self, i): Moves an element down to maintain the heap property.

MaxHeap Class

- __init__(self): Initializes an empty max heap.
- insert(self, item): Adds a new item to the heap.
- extract_max(self): Removes and returns the maximum element.
- remove(self, value, key_index=0): Removes a specific element from the heap.
- _sift_up(self, i) and _sift_down(self, i): Maintain the heap property.
- _compare(self, a, b): Compares elements based on priority and timestamp.
-

GatorTicketMaster Class

- __init__(self): Initializes the ticket master system.
- initialize(self, seat_count): Sets up the initial number of available seats.
- available(self): Returns the count of available seats and waitlist length.
- reserve(self, user_id, user_priority): Reserves a seat for a user or adds them to the waitlist.
- cancel(self, seat_id, user_id): Cancels a user's reservation and updates the system.
- exit_waitlist(self, user_id): Removes a user from the waitlist.
- update_priority(self, user_id, user_priority): Updates a user's priority in the waitlist.
- add_seats(self, count): Adds new seats and assigns them to waitlisted users if possible.
- print_reservations(self): Returns a list of all current reservations.
- release_seats(self, user_id1, user_id2): Releases seats for a range of user IDs and reassigns them.

How to run?

To run the GatorTicketMaster program, follow these steps:

- Ensure you have Python installed on your system.
- Save the provided code in a file named gatorTicketMaster.py.
- Prepare an input file (e.g., input.txt) with the commands you want to execute.
- Open a terminal or command prompt.
- Navigate to the directory containing gatorTicketMaster.py and your input file.
- Run the program using the following command:

  python3 gatorTicketMaster.py input.txt
- Replace input.txt with the name of actual input file.
- The program will process the commands from the input file and generate an output file named input_output_file.txt

How to create and run executable file?

- To create an executable file, in the terminal go to the directory where the makefile is located and run 'make' command in the terminal.
- To run use the following command:

  make run INPUT_FILE=input.txt
- Replace input.txt with the name of actual input file.