ADS Assignment-1 Report

GatorGlide Delivery Co. Order Management System

Author: Sai Ravi Teja G

UFID: 10504370

UF Email: gangavarapus@ufl.edu

Project Overview

This project implements an efficient order management system for GatorGlide Delivery Co. using AVL-balanced binary search trees. The system supports various operations such as creating orders, cancelling orders, updating delivery times, and retrieving order information.

Implementation Details

The system utilizes two AVL Balanced Binary Search Trees:

1. **Order Priority Tree**: This tree stores the orders based on their priority, which is calculated using a weighted combination of the order value and the current time. It ensures efficient insertion, deletion, and retrieval of orders while maintaining a balanced structure.
2. **ETA Tree**: This tree stores the orders based on their estimated time of arrival (ETA). It is used to deliver orders when the current time reaches their ETA efficiently and to update the ETAs of affected orders.

Function Prototypes and Program Structure

The project is implemented in Python, and the main components are as follows:

Order Class:

```
class OrderNode:
    def __init__(self, id, created_at, order_value, deliveryTime, eta, priority):
        self.id = id
        self.created_at = created_at
        self.order_value = order_value
        self.deliveryTime = deliveryTime
        self.eta = eta
        self.priority = priority
        self.left = None
        self.right = None
        self.parent = None
        self.height = 1
```

AVL Tree Class:

```
class AVLTree:
    def __init__(self):
        self.root = None

    def insert_node(self, current_node, new_node):
        # Logic to insert a new node into the AVL tree
        # Handles rebalancing and rotations as needed

    def remove_node(self, current_node, priority, id):
        # Logic to remove a node from the AVL tree
        # Handles rebalancing and rotations as needed

    def rotate_right(self, node_a, node_b):
```

```
        # Logic for right rotation of AVL tree nodes

    def rotate_left(self, node_a, node_b):
        # Logic for left rotation of AVL tree nodes

    def rebalance(self, current_node):
        # Logic to rebalance the AVL tree after insertions/deletions

    def get_balance_factor(node):
        # Logic to calculate the balance factor of a node

    def find_min(node):
        # Logic to find the node with the minimum priority

    def swap_nodes(node_a, node_b):
        # Logic to swap the values of two nodes
```

Main Program:

The main program consists of several helper functions to handle order operations, such as:

- `calculatePriority(value, time, valueWeight=0.3, timeWeight=0.7)`
- `printOrder(id)`
- `printOrdersInRange(start, end)`
- `getOrderRank(id)`
- `createNewOrder(id, currTime, value, deliveryDuration)`
- `cancelOrderById(id, currTime)`
- `updateDeliveryTime(id, currTime, newDuration)`
- `deliver_orders(current_time)`
- `printAllRemainingOrders()`
- `processCommand(cmd)`

The `main()` function handles the input/output operations and calls the appropriate helper functions based on the user commands.

Usage:

To use the GatorGlide Delivery Co. order management system:

1. Ensure that you have Python installed on your system.
2. Run the Python script with the input file as a command-line argument. For example:

   ```
   python gatorDelivery.py input.txt
   ```

3. The program will
   process the commands from the input file and generate an output file
   with the same name as the input file, appended with
   `_output_file.txt`.
4. Review the generated output file to see the results of the executed commands.

Steps I've Taken to Build

1. **Understood the Problem**: Read through the problem statement and requirements carefully.
2. **Designed Data Structures**: Decided to use two AVL trees. One to store orders based on priority and another to store orders based on ETA.
3. **Implemented AVL Tree**: Implementing the basic AVL tree was the most challenging part. It took me multiple tries to understand how to maintain the balance and handle rotations correctly. Then I made a common class

for both Priority Tree and ETA Tree. However, I used order_id for finding nodes from Priority Tree because there can be duplicates.

4. **Created Order Class**: Defined a simple `OrderNode`derNode class to hold order details like ID, arrival time, value, and duration.
5. **Implemented Order Operations**: Implemented functions to handle order operations like create, cancel, update, and retrieve information. Managing the AVL tree and updating priorities/ETAs after each operation was tricky.
6. **Added Error Handling**: Covered edge cases like canceling non-existent orders or updating already delivered orders.
7. **Tested and Debugged**: Wrote test cases and spent a good amount of time debugging issues with order delivery, priority updates, and AVL tree operations.