Dynamic Vehicle Registration Management System with Efficient Expiration Tracking and License Plate Search

## Name: Brendan Yeong

## Student ID: 005025797

## Course: MSCS532

## Project Phase 1 Deliverable 1

1. Application context: Vehicle Registration System

A vehicle registration system that is responsible for keeping track of vehicle registrations, owner details (first name, last name), and associated information such as vehicle details, registration statuses (active, expired, pending, suspended, etc.), and expiration dates(dd/mm/yyyy). This system must be able to handle the following dynamic changes:

* vehicle registration records (including adding, updating, retrieve and removing vehicles).
* Owner details (each car is associated with an owner who can be queried by name or ID).
* Registration status(active, expired, pending, suspended, etc.) and expiration dates for registration that will expire .

The system should be able to handle searches, modifications, and queries (e.g., finding cars by owner, searching for expired registrations, etc.) efficiently.

2. Design the data structure

For this dynamic vehicle registration system, I have decided these data structures that could be a good fit for managing the types of data to be stored and handle dynamic changes. Let’s define a few data structure needed for these features, vehicle registration records, expiration dates queries, finding vehicle by owner, vehicle plate queries.

* Hash Table (Dictionary) for vehicle details:
  + We need to store each vehicle uniquely with the number plate and its associating details such as registration details (make, model, color, year, classification, vin number), and details such as owner name, registration date, and expiration date. A hash table allows average time complexity for it inserts, search, updates, and remove operations. This is important for when we are going to be managing thousands to millions of registrations. Using the license plate number as a unique key, the Hash Table offers fast access to vehicle details and it is space-efficient
* Heap (Priority Queue) for expiration dates:
  + We can use a priority queue to keep track of registration expiration dates to quickly identify the next registration that will expire. With priority queue the run-time for insertion and deletion are and accessing to the top element is , making it easy to lookup the nearest expiration date with the license plate efficiently. Using heap in this case is very efficient because we can quickly access and update the expiring registrations without needing to iterate over all entries.
* Balance BST (AVL) for owner-based queries:
  + We need a feature to that allows searches to find all the vehicle associated by a specific owner (Driver’s License number). An AVL, binary search tree with a balance condition ensures the operation of insertion, deletion, and search run-time is . This means that it is efficient for owner-base queries. Another advantage in doing this implementation is for each time a new owner is entered, it ensures that the height is balanced and this guarantees that the tree will remain logarithmic in height making it consistently efficient with all the operations.
* Trie Data Structure for Plate prefix queries:
  + Last feature that I would like to include is a plate prefix lookup feature that can return a list of plates from the partial lookup. For this feature we can use Trie data structure as it allows a autocomplete feature. A Trie uses a time complexity of where is the length of the prefix. The only drawback is it uses a lot of memory but it allows users to search up vehicle license plate numbers partially.

3. Implement the Data Structures in Python:

Here are the code snippets for all the implementations. For more details head to the GitHub link, https://github.com/CookiesBot3/MSCS532\_Project

Hash Table (Dictionary) for vehicle details:

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

Heap (Priority Queue) for expiration dates:

A screen shot of a computer program

Description automatically generated

Balance BST (AVL) for owner-based queries:

A computer screen shot of a code

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A computer screen shot of a program code

Description automatically generated

Trie Data Structure for Plate prefix queries:

A screenshot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated

Challenges and limitations

One of the limitations, I see in the long term is managing data synchronizations. As the system grows, managing data consistency and synchronization with multiple users accessing and updating data could lead to some complications and that could be a challenge. A layer of concurrency control or transaction handling is definitely needed.

Although Trie data structure is one of the main data structure for search results it does consume a lot of memory, especially when they are many unique license plates. Optimizations such as making it into a compressed tries can potentially mitigate this issue but it adds a layer of complexity to the implementations.

Another is the AVL tree. Even though it is very efficient in doing it’s basic operations it is very hard to implement and maintain overtime when new data or requirement is added. For a future large-scale application, a more vigorous testing is needed to prevent performance bottlenecks or potential errors occurs in the system.

Reference

Goodrich, M. T., & Tamassia, R. (2014). *Data Structures and algorithms in Java Michael T. Goodrich; Roberto Tamassia*. John Wiley.

Knuth, D. E. (1998). *The art of computer programming. volume 3*. Addison-Wesley.

Weiss, M. A. (2012). *Data structures and algorithm analysis in Java*. Addison-Wesley.