Assignment
Data Structures and Algorithms
Semester 2, 2024

Curtin University

Title: ADTs for Autonomous Vehicle Management System

Due Date: 27th October 2024 23:59. Additional One week for the students with Curtin Access Plan (CAP).

Introduction

The aim of this assignment is to design and implement an Autonomous Vehicle Management System (AVMS) that utilises various data structures and algorithms. This system will efficiently manage a fleet of autonomous vehicles and support real-time queries about vehicle status and location. You will implement key data structures and algorithms such as graphs, hashing, heap and sorting algorithms (Heapsort, quicksort) to create a user-friendly AVMS.

You are encouraged to re-use the generic ADTs developed in your practicals but must not use built-in Java or Python ADTs. If you have any doubt, please seek clarification.

******READ COMPLETE DOCUMENT BEFORE STARTING*****

Key Components

- ***Graph Representation for Road Network***

  Hints: Implement a Graph class to represent the road network, where nodes represent locations (suburbs/ cities) and edges represent the roads between them. The user should be able to do the following operations:

  - **Adding Vertices and Edges:** Create methods to add vertices (locations) and edges (roads) between them. Each edge should store data like distance or travel time between locations.
  - **Retrieving Neighbors:** Implement a method to retrieve all the neighbouring locations (adjacent vertices) for a given location.
  - **Display Graph**: Create a method to print or display the graph structure, showing all locations and their connections.
  - **Checking for Path Existence:** Implement a method is_path(source, destination) that returns True if a path exists between the two given locations and False if no path exists. Use either BFS or DFS to traverse through the Graph, starting from the source location. If the destination location is found during the traversal, return True. If the traversal ends without reaching the destination, return False. Handle cases where the source or destination doesn't exist in the Graph or the source and destination are the same.

  Marks Distribution
  - Adding Vertices and Edges (2 marks)
  - Retrieving Neighbors (1 marks)
  - Displaying Graph (1 marks)
  - Path Existence Check (3 marks)

- ***VehicleHashTable Class Implementation***

Hints: Create a VehicleHashTable class that uses a **hash table** to store and manage the autonomous vehicles in your system. Define a hash function that calculates the index based on the **vehicle ID**. You will implement essential operations such as **insertion**, **deletion**, **searching**, and **displaying** all vehicles information. The class should support collision resolution techniques like **linear probing** or **chaining** to handle hash collisions. Choose a collision resolution technique and implement it effectively to handle multiple vehicles mapping to the same hash index.

Marks Distribution

- Hashing (2 marks)
- Insertion, Deletion, and Searching (3 marks)
- Collision Resolution Technique (2 marks)
- Displaying all information (2 marks)

- ***Vehicle Class Implementation***

Hints: Define a Vehicle class that encapsulates various attributes of an autonomous vehicle. Each vehicle will have the following attributes and methods:

Vehicle Class Attributes:

- **Vehicle ID:** A unique identifier for each vehicle.
- **Current Location:** Represents the vehicle's current position in the road network (e.g., node A).
- **Destination:** The vehicle's target destination in the road network.
- **Distance_to_Destination**: Calculated distance from the current location to the destination.
- **Battery_Level**: A percentage indicating the remaining battery power.

Vehicle Class Methods:

- **setLocation(location):** Updates the vehicle's current location.
- **setDestination(destination):** Updates the vehicle's destination.
- **setDistanceToDestination(distance):** Updates the vehicle's distance from its destination.
- **setBatteryLevel(level):** Updates the vehicle's remaining battery percentage.
- **getLocation():** Retrieves the current location of the vehicle.
- **getDestination():** Retrieves the destination of the vehicle.
- **getDistanceToDestination():** Retrieves the distance from the vehicle's current location to the destination.
- **getBatteryLevel():** Retrieves the vehicle's battery level.

Marks Distribution

- Correct Attributes and Methods Functionalities (3 marks)

- ***Vehicle Recommendation based on Sorting Algorithms***

Implement two sorting algorithms—Heapsort and Quicksort—to sort a list of vehicles based on different attributes. Your program will use the Vehicle class to perform the sorting and provide recommendations based on the sorted data.

You will use Heapsort to sort vehicles based on their Distance_to_Destination in ascending order. Additionally, you will recommend the vehicle closest to its destination using the method find_nearest_vehicle().

Implement the Quicksort algorithm to sort vehicles based on their **Battery_Level** in descending order. Your program will also recommend the vehicle with the highest battery level using the method find_vehicle_with_highest_battery().

Marks Distribution

- Correct Heapsort Implementation for Distance to Destination (3 marks)
- Correct Quicksort Implementation for Battery Level (3 marks)
- Correct Recommendation Methods (2 marks)

- ***Interactive Menu and Testing***

  Provide an interactive menu for users to call the necessary functions or methods. Test the functionalities of each class and method. Ensure the user interface handles invalid inputs effectively.

Marks Distribution

- Functional Menu with Proper Error Handling (3 marks)

Deliverables and Submission Guidelines

Submit electronically via Blackboard. It is mandatory to attempt the assignment submission to meet the passing criteria of this unit.

You should submit a single file, which should be zipped (.zip) or tarred (.tar.gz). Check that you can decompress it on the lab computers. These are also the computers on which your work will be tested, so make sure that your work runs there. The file must be named DSA_Assignment_<id> where the <id> is replaced by your student id. There should be no spaces in the file name; use underscores as shown. **No group submission is allowed.** You should complete the assignment individually and demonstrate it to your tutor. If you previously worked in a group while submitting practicals, you should self-cite or put comments on your code.

You should submit the following documents and attend the demonstration:

- ✓ **Source Code**: Well-organised and documented code for all components of the AVMS.
- ✓ **Technical Report** (8 marks): Submit a report explaining your implementation strategy and how you structured your classes, methods and algorithms. Include a flowchart or diagram/ UML illustrating the system's architecture, detailing the components' interactions. Discuss any challenges you faced during the implementation and how you addressed them. Provide a brief analysis of the efficiency of your algorithm and potential improvements or alternative approaches.
- ✓ **Test Cases** (2 marks): A document outlining test cases for each component, including the expected outcomes and a summary of testing procedures.
- ✓ **Code Demonstration:** Perform the demonstration live in class or via a virtual class session to your tutor. The demonstration schedule will be confirmed via announcement on Blackboard. **Expected date of demonstration is (28th October – 6th November).** It is mandatory to attend the scheduled demonstration to get the assignment marked. If you have any issue with the schedule date of demonstration, please let your Lecturer know.