COMP5008 DATA STRUCTURES AND ALGORITHMS

Assignment Report

NOVEMBER 3, 2024

Student ID: 21678145
Surname: Niu
Given name: Ben

Table of Contents

1. 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.

The objectives are:

- Implement an ADT Graph to represent the road network.
- Implement an ADT HashTable to store and manage the autonomous vehicles.
- Implement a Vehicle class to encapsulate various attributes.
- Implement the vehicle recommendation using two sorting algorithms – Heapsort and Quicksort to sort a list of vehicles based on different attributes.
- Implement an interactive menu for users to call the functions.
- Implement test cases to test the basic functionalities of each class and method.

2. Class and Method Descriptions

2.1 Class DSAGraph

DSAGraph is used to define a graph data abstract type.

Methods:

- addVertex(label): Add a new vertex to the graph.
- deleteVertex(label): Delete a vertex from the graph.
- addEdge(label1, label2, value): Add an new edge to the graph.
- deleteEdge(label1, label2): Delete an existing edge from the graph.
- hasVertex(label): Check if a vertex exists in the graph.
- getVertexCount(): Get the count of vertices in the graph.
- getEdgeCount(): Get the count of edges in the graph.
- getVertex(label): Get a vertex from the graph using the vertex label.
- getVertexNode(label): Get a linked list node of a vertex from the graph using the vertex label.
- getAdjacent(label): Get a linked list of adjacent vertices of a vertex using the vertex label.
- isAdjacent(label1, label2): Use to check if two vertices are adjacent.
- getEdge(label1, label2): Get an edge from the graph using the vertex labels.
- getDistance(label1, label2): Get the distance between two vertices.
- getPath(label1, label2): Get the path between two vertices.

- isPath(label1, label2): Check if there is a path between two vertices using the vertex label.
- displayAsList(): Display the graph as a list.
- displayEdges(): Display all the edges of the graph.
- displayAsMatrix(): Display the graph as a matrix.
- clearVisited(): Clear the visited flags for all the vertices.
- breadthFirstSearch(): Perform a breadth first research on the graph.
- depthFirstSearch(): Perform a depth first research on the graph.
- sort(): Sort the vertices list and edges list.

2.1.1 Inner class _DSAGraphVertex

_DSAGraphVertex is used to define a vertex in a graph.

Methods:

- setVisited(): Set the visited flag.
- clearVisited(): Clear the visited flag.

2.1.2 Inner class _DSAGraphEdge

_DSAGraphEdge is used to define an edge in a graph. It will store the labels and value.

2.2 Class DSAHashTable

DSAHashTable is used to define a hash table data abstract type.

Methods:

- initHashArray(size): Init the hash array with a given size.
- search(key): Search a value by the given key.
- insert(vehicle): Insert a vehicle into the hash table.
- delete(key): Delete a vehicle by the given key.
- getLoadFactor(): Get the load factor of the hash table.
- export(): Export all vehicles in the hash table to an array.
- _resize(): Resize the array in the hash table by the load factor.
- _hash(key): Hash the key to an index in the hash table.
- _stepHash(key): Step hash method to avoid duplicated index.
- _nextPrime(size): Get the next prime number by the given value.
- hasKey(key): Check if the hash table contains the given key.
- _checkKey(idx, key): Check if the key is in the hash table by the given index.
- _findKey(key): Find the index in the hash table from the key.

- _checkEmpty(idx, key): Check if the index is useful in the hash table.
- _findEmpty(key): Find an empty index in the hash table by the input key.

2.2.1 Inner class _DSAHashEntry

_DSAHashEntry is used to define a hash entry in a hash table.

2.3 Class DSALinkedList

DSALinkedList is used to define a linked list data abstract type.

Methods:

- insertFirst(value): Insert a new value at the first position of the linked list.
- insertLast(value): Insert a new value at the last position of the linked list.
- removeFirst(): Remove the first node from the linked list.
- removeLast(): Remove the last node from the linked list.
- peekFirst(): Return the value of the first node in the linked list.
- peekLast(): Return the value of the last node in the linked list.
- find(valueToFind): Find the node with the input value in the linked list.
- remove(valueToFind): Remove the node with the input value from the linked list.
- isEmpty(): Check if the linked list is empty.

2.3.1 Inner class _DSALinkedNode

_DSALinkedNode is used to define a node used in a linked list.

2.4 Class DSAQueue

DSAQueue is used to define a queue data abstract type.

Methods:

- enqueue(value): Enqueue a new value into the queue.
- dequeue(): Dequeue a value from the queue.
- peek(): Peek the first value in the queue.
- isEmpty(): Check if the queue is empty.
- getCount(): Get the item count of the queue.

2.5 Class DSAStack

- DSAStack is used to define a stack data abstract type.
- push(value): Push a new value into the stack.

- pop(): Pop a value from the stack.
- top(): Peek the top value in the stack.
- isEmpty(): Check if the stack is empty.
- getCount(): Get the item count of the stack.

2.6 Class DSAVehicle

Vehicle class is used to store the information of a vehicle.

Methods:

- setLocation(loc): Set the location of the vehicle.
- setDestination(loc): Set the destination of the vehicle.
- setDistanceToDest(distance): Set the distance to the destination of the vehicle.
- setBatteryLevel(level): Set the battery level of the vehicle.
- getLoc(): Get the location of the vehicle.
- getDest(): Get the destination of the vehicle.
- getDistanceToDest(): Get the distance to the destination of the vehicle.
- getBatteryLevel(): Get the battery level of the vehicle.

2.7 File DSAsorts

Methods:

- quickSort(A): Sort the array using quick sort algorithm
- quickSortRecurseLeft(A, leftIdx, rightIdx): Implement quick sort algorithm using left-most element as pivot.
- quickSortRecurseRan(A, leftIdx, rightIdx): Implement quick sort algorithm using random element as pivot.
- quickSortRecurseMid(A, leftIdx, rightIdx): Implement quick sort algorithm using middle element as pivot.
- quickSortRecurseMedian3(A, leftIdx, rightIdx): Implement quick sort algorithm using median 3 element as pivot.
- doPartitioning(A, leftIdx, rightIdx, pivotIdx): Find all values that are smaller than the pivot and transfer them to the left-hand-side of the array.
- trickleDown(A, curIdx, numItems): Trickle down the item to the correct position in the heap sort.
- heapify(A, numItems): Heapify the array.
- heapSort(A): Sort the array using heap sort algorithm.
- findNearestVehicle(vehicles): Find the nearest vehicle from the list of vehicles.
- findVehicleWithHighestBattery(vehicles): Find the vehicle with the highest battery from the list

of vehicles.

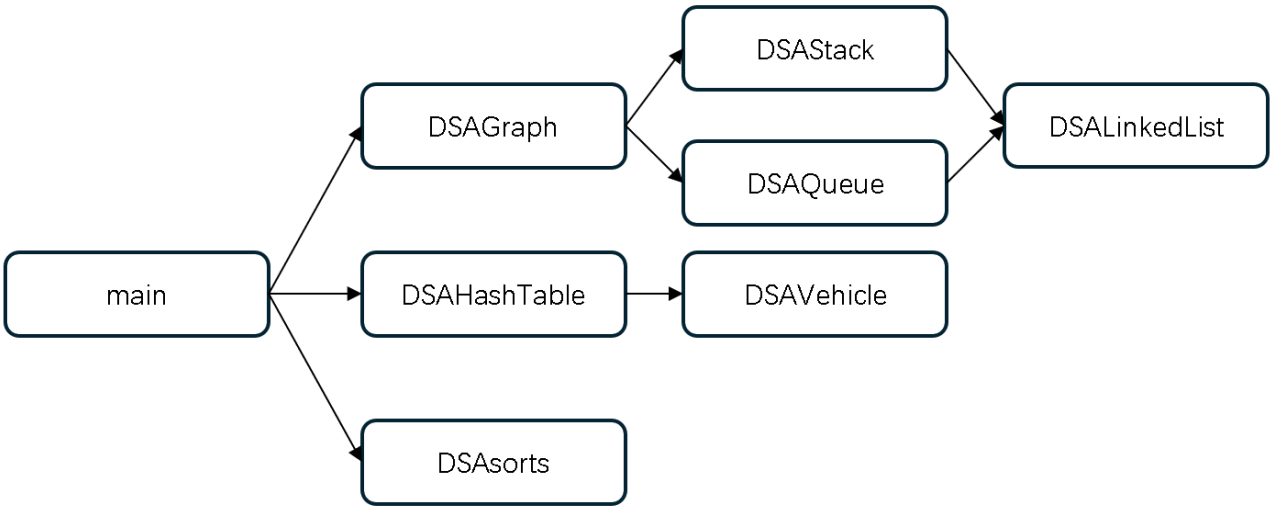
2.8 File main

Methods:

- usage(): Print the usage of the interactive menu.
- interactive(): Interactive menu for the DSA assignment.

2.9 Diagram

The file structure is shown in the figure below:



3. Test Cases

3.1 testGraph

Testing graph operations include: addVertex, addEdge, getVertexCount, getEdgeCount, displayAsList, displayAsMatrix, isPath, deleteEdge, deleteVertex, sort, depthFirstSearch, breadFirstSearch.

```
ben_niu@LAPTOP-E6OI78J8:~/DSA/FinalAssignment$ python3 -m unittest testGraph.py

Display as list, UNDIRECT, vertex count=0, edge count=0

Display as matrix, UNDIRECT, vertex count=0 edge count=0


Display as list, UNDIRECT, vertex count=2, edge count=0
  V: A [ ]
  V: B [ ]

Display as matrix, UNDIRECT, vertex count=2 edge count=0
  A B
A 0 0
B 0 0
Get Path: from A to B
Display as list, UNDIRECT, vertex count=2, edge count=1
  V: A [ B ]
  V: B [ A ]

Display as matrix, UNDIRECT, vertex count=2 edge count=1
  A B
A 0 1
B 1 0
Get Path: from A to B { (A,B) }

Display as list, UNDIRECT, vertex count=3, edge count=2
  V: A [ B C ]
  V: B [ A ]
  V: C [ A ]

Display as matrix, UNDIRECT, vertex count=3 edge count=2
  A B C
A 0 1 1
B 1 0 0
C 1 0 0
Get Path: from A to C { (A,C) }
Get Path: from C to B { (C,A) (A,B) }
```

```
Display as list, UNDIRECT, vertex count=3, edge count=3
  V: A [ B C ]
  V: B [ A C ]
  V: C [ A B ]

Display as matrix, UNDIRECT, vertex count=3 edge count=3
  A B C
A 0 1 1
B 1 0 1
C 1 1 0
Get Path: from C to B { (C,B) }

Display as list, UNDIRECT, vertex count=3, edge count=2
  V: A [ B C ]
  V: B [ A ]
  V: C [ A ]

Display as matrix, UNDIRECT, vertex count=3 edge count=2
  A B C
A 0 1 1
B 1 0 0
C 1 0 0
Get Path: from C to B { (C,A) (A,B) }

Display as list, UNDIRECT, vertex count=2, edge count=1
  V: A [ B ]
  V: B [ A ]

Display as matrix, UNDIRECT, vertex count=2 edge count=1
  A B
A 0 1
B 1 0
DFS: { (A,B) }
BFS: { (A,B) }
.
----------------------------------------------------------------------
Ran 1 test in 0.003s

OK
```

3.2 testHashTable

Testing HashTable operations include: insert, search, delete.

```
ben_niu@LAPTOP-E6OI78J8:~/DSA/FinalAssignment$ python3 -m unittest testHashTable.py
Count: 0 Size: 5 LF: 0.0 Low: 0.2 High: 0.6
Count: 2 Size: 5 LF: 0.4 Low: 0.2 High: 0.6
 HashEntry: key=A value=Vehicle ID: A Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=B value=Vehicle ID: B Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
Count: 3 Size: 5 LF: 0.6 Low: 0.2 High: 0.6
 HashEntry: key=A value=Vehicle ID: A Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=B value=Vehicle ID: B Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=C value=Vehicle ID: C Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
Count: 4 Size: 11 LF: 0.36 Low: 0.2 High: 0.6
 HashEntry: key=B value=Vehicle ID: B Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=C value=Vehicle ID: C Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=D value=Vehicle ID: D Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=A value=Vehicle ID: A Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
Count: 5 Size: 11 LF: 0.45 Low: 0.2 High: 0.6
 HashEntry: key=B value=Vehicle ID: B Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=C value=Vehicle ID: C Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=D value=Vehicle ID: D Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=E value=Vehicle ID: E Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=A value=Vehicle ID: A Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
Count: 6 Size: 11 LF: 0.55 Low: 0.2 High: 0.6
 HashEntry: key=B value=Vehicle ID: B Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=C value=Vehicle ID: C Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=D value=Vehicle ID: D Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=E value=Vehicle ID: E Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=F value=Vehicle ID: F Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=A value=Vehicle ID: A Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
Count: 7 Size: 23 LF: 0.3 Low: 0.2 High: 0.6
 HashEntry: key=E value=Vehicle ID: E Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=F value=Vehicle ID: F Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=G value=Vehicle ID: G Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=A value=Vehicle ID: A Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=B value=Vehicle ID: B Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=C value=Vehicle ID: C Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=D value=Vehicle ID: D Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
```

```
Count: 6 Size: 23 LF: 0.26 Low: 0.2 High: 0.6
 HashEntry: key=E value=Vehicle ID: E Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=F value=Vehicle ID: F Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=G value=Vehicle ID: G Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=B value=Vehicle ID: B Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=C value=Vehicle ID: C Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=D value=Vehicle ID: D Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
Count: 5 Size: 23 LF: 0.22 Low: 0.2 High: 0.6
 HashEntry: key=E value=Vehicle ID: E Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=F value=Vehicle ID: F Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=G value=Vehicle ID: G Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=C value=Vehicle ID: C Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=D value=Vehicle ID: D Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
Count: 4 Size: 11 LF: 0.36 Low: 0.2 High: 0.6
 HashEntry: key=D value=Vehicle ID: D Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=E value=Vehicle ID: E Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=F value=Vehicle ID: F Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=G value=Vehicle ID: G Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
Count: 3 Size: 11 LF: 0.27 Low: 0.2 High: 0.6
 HashEntry: key=E value=Vehicle ID: E Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=F value=Vehicle ID: F Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=G value=Vehicle ID: G Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
Count: 2 Size: 5 LF: 0.4 Low: 0.2 High: 0.6
 HashEntry: key=F value=Vehicle ID: F Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
 HashEntry: key=G value=Vehicle ID: G Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
Count: 1 Size: 5 LF: 0.2 Low: 0.2 High: 0.6
 HashEntry: key=G value=Vehicle ID: G Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: -1 state=1
Count: 0 Size: 3 LF: 0.0 Low: 0.2 High: 0.6
.
----------------------------------------------------------------------
Ran 1 test in 0.008s

OK
```

3.3 testLinkedList

Testing LinkedList operations include: removeFirst, removeLast, peekFirst, peekLast, isEmpty, insertFirst, insertLast.

```
ben_niu@LAPTOP-E6OI78J8:~/DSA/FinalAssignment$ python3 -m unittest testLinkedList.py
LinkedList: head=None, tail=None, empty=True
LinkedList: head=LinkedNode: value=1 Not_None? next=False prev=False, tail=LinkedNode: value=1 Not_None? next=False prev=False, empty=False
  LinkedNode: value=1 Not_None? next=False prev=False
LinkedList: head=LinkedNode: value=1 Not_None? next=True prev=False, tail=LinkedNode: value=2 Not_None? next=False prev=True, empty=False
  LinkedNode: value=1 Not_None? next=True prev=False
  LinkedNode: value=2 Not_None? next=False prev=True
LinkedList: head=LinkedNode: value=1 Not_None? next=False prev=False, tail=LinkedNode: value=1 Not_None? next=False prev=False, empty=False
  LinkedNode: value=1 Not_None? next=False prev=False
LinkedList: head=LinkedNode: value=1 Not_None? next=True prev=False, tail=LinkedNode: value=3 Not_None? next=False prev=True, empty=False
  LinkedNode: value=1 Not_None? next=True prev=False
  LinkedNode: value=3 Not_None? next=False prev=True
LinkedList: head=LinkedNode: value=1 Not_None? next=True prev=False, tail=LinkedNode: value=4 Not_None? next=False prev=True, empty=False
  LinkedNode: value=1 Not_None? next=True prev=False
  LinkedNode: value=3 Not_None? next=True prev=True
  LinkedNode: value=4 Not_None? next=False prev=True
LinkedList: head=LinkedNode: value=3 Not_None? next=True prev=False, tail=LinkedNode: value=4 Not_None? next=False prev=True, empty=False
  LinkedNode: value=3 Not_None? next=True prev=False
  LinkedNode: value=4 Not_None? next=False prev=True
LinkedList: head=LinkedNode: value=4 Not_None? next=False prev=False, tail=LinkedNode: value=4 Not_None? next=False prev=False, empty=False
  LinkedNode: value=4 Not_None? next=False prev=False
LinkedList: head=None, tail=None, empty=True
LinkedList: head=LinkedNode: value=1 Not_None? next=False prev=False, tail=LinkedNode: value=1 Not_None? next=False prev=False, empty=False
  LinkedNode: value=1 Not_None? next=False prev=False
LinkedList: head=LinkedNode: value=2 Not_None? next=True prev=False, tail=LinkedNode: value=1 Not_None? next=False prev=True, empty=False
  LinkedNode: value=2 Not_None? next=True prev=False
  LinkedNode: value=1 Not_None? next=False prev=True
LinkedList: head=LinkedNode: value=3 Not_None? next=True prev=False, tail=LinkedNode: value=1 Not_None? next=False prev=True, empty=False
  LinkedNode: value=3 Not_None? next=True prev=False
  LinkedNode: value=2 Not_None? next=True prev=True
  LinkedNode: value=1 Not_None? next=False prev=True
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

3.4 testQueue

Testing LinkedList operations include: isEmpty, getCount, dequeue, enqueue, peek, getCount.

```
ben_niu@LAPTOP-E6OI78J8:~/DSA/FinalAssignment$ python3 -m unittest testQueue.py
Queue: count=0 queue=LinkedList: head=None, tail=None, empty=True
Queue: count=1 queue=LinkedList: head=LinkedNode: value=1 Not_None? next=False prev=False, tail=LinkedNode: value=1 Not_None? next=False prev=Fal
se, empty=False
  LinkedNode: value=1 Not_None? next=False prev=False
Queue: count=2 queue=LinkedList: head=LinkedNode: value=1 Not_None? next=True prev=False, tail=LinkedNode: value=2 Not_None? next=False prev=True
, empty=False
  LinkedNode: value=1 Not_None? next=True prev=False
  LinkedNode: value=2 Not_None? next=False prev=True
Queue: count=3 queue=LinkedList: head=LinkedNode: value=1 Not_None? next=True prev=False, tail=LinkedNode: value=3 Not_None? next=False prev=True
, empty=False
  LinkedNode: value=1 Not_None? next=True prev=False
  LinkedNode: value=2 Not_None? next=True prev=True
  LinkedNode: value=3 Not_None? next=False prev=True
Queue: count=3 queue=LinkedList: head=LinkedNode: value=2 Not_None? next=True prev=False, tail=LinkedNode: value=4 Not_None? next=False prev=True
, empty=False
  LinkedNode: value=2 Not_None? next=True prev=False
  LinkedNode: value=3 Not_None? next=True prev=True
  LinkedNode: value=4 Not_None? next=False prev=True
Queue: count=4 queue=LinkedList: head=LinkedNode: value=2 Not_None? next=True prev=False, tail=LinkedNode: value=5 Not_None? next=False prev=True
, empty=False
  LinkedNode: value=2 Not_None? next=True prev=False
  LinkedNode: value=3 Not_None? next=True prev=True
  LinkedNode: value=4 Not_None? next=True prev=True
  LinkedNode: value=5 Not_None? next=False prev=True
Queue: count=3 queue=LinkedList: head=LinkedNode: value=3 Not_None? next=True prev=False, tail=LinkedNode: value=5 Not_None? next=False prev=True
, empty=False
  LinkedNode: value=3 Not_None? next=True prev=False
  LinkedNode: value=4 Not_None? next=True prev=True
  LinkedNode: value=5 Not_None? next=False prev=True
Queue: count=2 queue=LinkedList: head=LinkedNode: value=4 Not_None? next=True prev=False, tail=LinkedNode: value=5 Not_None? next=False prev=True
, empty=False
  LinkedNode: value=4 Not_None? next=True prev=False
  LinkedNode: value=5 Not_None? next=False prev=True
Queue: count=1 queue=LinkedList: head=LinkedNode: value=5 Not_None? next=False prev=False, tail=LinkedNode: value=5 Not_None? next=False prev=Fal
se, empty=False
  LinkedNode: value=5 Not_None? next=False prev=False
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

3.5 testStack

Testing Stack operations include: isEmpty, getCount, push, pop, top, getCount.

```
ben_niu@LAPTOP-E6OI78J8:~/DSA/FinalAssignment$ python3 -m unittest testStack.py
Stack: count=1 stack=LinkedList: head=LinkedNode: value=1 Not_None? next=False prev=False
, tail=LinkedNode: value=1 Not_None? next=False prev=False, empty=False
  LinkedNode: value=1 Not_None? next=False prev=False
Stack: count=2 stack=LinkedList: head=LinkedNode: value=2 Not_None? next=True prev=False,
 tail=LinkedNode: value=1 Not_None? next=False prev=True, empty=False
  LinkedNode: value=2 Not_None? next=True prev=False
  LinkedNode: value=1 Not_None? next=False prev=True
Stack: count=3 stack=LinkedList: head=LinkedNode: value=3 Not_None? next=True prev=False,
 tail=LinkedNode: value=1 Not_None? next=False prev=True, empty=False
  LinkedNode: value=3 Not_None? next=True prev=False
  LinkedNode: value=2 Not_None? next=True prev=True
  LinkedNode: value=1 Not_None? next=False prev=True
Stack: count=2 stack=LinkedList: head=LinkedNode: value=2 Not_None? next=True prev=False,
 tail=LinkedNode: value=1 Not_None? next=False prev=True, empty=False
  LinkedNode: value=2 Not_None? next=True prev=False
  LinkedNode: value=1 Not_None? next=False prev=True
Stack: count=2 stack=LinkedList: head=LinkedNode: value=2 Not_None? next=True prev=False,
 tail=LinkedNode: value=1 Not_None? next=False prev=True, empty=False
  LinkedNode: value=2 Not_None? next=True prev=False
  LinkedNode: value=1 Not_None? next=False prev=True
Stack: count=1 stack=LinkedList: head=LinkedNode: value=1 Not_None? next=False prev=False
, tail=LinkedNode: value=1 Not_None? next=False prev=False, empty=False
  LinkedNode: value=1 Not_None? next=False prev=False
Stack: count=0 stack=LinkedList: head=None, tail=None, empty=True
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

3.6 testSort

Testing Sort operations include: testHeapSort, testQuickSort.

```
ben_niu@LAPTOP-E6OI78J8:~/DSA/FinalAssignment$ python3 -m unittest testSort.py
Unsorted vehicles:
Vehicle ID: A Current Location: None Destination Location: None Distance to Destination: 3 Battery Level: -1
Vehicle ID: B Current Location: None Destination Location: None Distance to Destination: 1 Battery Level: -1
Vehicle ID: C Current Location: None Destination Location: None Distance to Destination: 5 Battery Level: -1
Vehicle ID: D Current Location: None Destination Location: None Distance to Destination: 4 Battery Level: -1
Vehicle ID: E Current Location: None Destination Location: None Distance to Destination: 2 Battery Level: -1
Heap sorted vehicles:
Vehicle ID: B Current Location: None Destination Location: None Distance to Destination: 1 Battery Level: -1
Vehicle ID: E Current Location: None Destination Location: None Distance to Destination: 2 Battery Level: -1
Vehicle ID: A Current Location: None Destination Location: None Distance to Destination: 3 Battery Level: -1
Vehicle ID: D Current Location: None Destination Location: None Distance to Destination: 4 Battery Level: -1
Vehicle ID: C Current Location: None Destination Location: None Distance to Destination: 5 Battery Level: -1
.Unsorted vehicles:
Vehicle ID: A Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: 3
Vehicle ID: B Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: 1
Vehicle ID: C Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: 5
Vehicle ID: D Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: 4
Vehicle ID: E Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: 2
Quick sorted vehicles:
Vehicle ID: C Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: 5
Vehicle ID: D Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: 4
Vehicle ID: A Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: 3
Vehicle ID: E Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: 2
Vehicle ID: B Current Location: None Destination Location: None Distance to Destination: -1 Battery Level: 1
.
----------------------------------------------------------------------
Ran 2 tests in 0.001s

OK
```

3.7 testMain

Testing interactive functions include: testGraph, testHashTable, testRecommend.

```
ben_niu@LAPTOP-E6OI78J8:~/DSA/FinalAssignment$ python3 -m unittest testMain.py

================================
Interactive Menu for DSA Assignment
  (gav) Graph: Add Vertex
  (gdv) Graph: Delete Vertex
  (gae) Graph: Add Edge
  (gde) Graph: Delete Edge
  (grn) Graph: Retrieve Neighbors
  (gdg) Graph: Display Graph
  (gcp) Graph: Check Path
  (hi) HashTable: Insert
  (hr) HashTable: Remove
  (hs) HashTable: Search
  (hd) HashTable: Display
  (rnv) Recommend: Nearest Vehicle
  (rhb) Recommend: Highest Battery
  (q) Quit
================================


Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the vertex label:
Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the vertex label:
Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Display as matrix, UNDIRECT, vertex count=2 edge count=0
  A B
A 0 0
B 0 0

Display as list, UNDIRECT, vertex count=2, edge count=0
  V: A [ ]
  V: B [ ]

Display edges, edge count=0

Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the vertex label: Delete node: 'A' success

Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Display as matrix, UNDIRECT, vertex count=1 edge count=0
```

```
   B
B 0

Display as list, UNDIRECT, vertex count=1, edge count=0
  V: B [ ]

Display edges, edge count=0

Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the edge like LABEL1 LABEL2 DISTANCE:
Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the edge like LABEL1 LABEL2 DISTANCE:
Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Display as matrix, UNDIRECT, vertex count=3 edge count=2
  A B C
A 0 1 1
B 1 0 0
C 1 0 0

Display as list, UNDIRECT, vertex count=3, edge count=2
  V: A [ B C ]
  V: B [ A ]
  V: C [ A ]

Display edges, edge count=2
  E: A -> B, 2
  E: B -> A, 2
  E: A -> C, 3
  E: C -> A, 3

Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the edge like LABEL1 LABEL2:
Delete edge: A-C success

Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Display as matrix, UNDIRECT, vertex count=3 edge count=1
  A B C
A 0 1 0
B 1 0 0
C 0 0 0
```

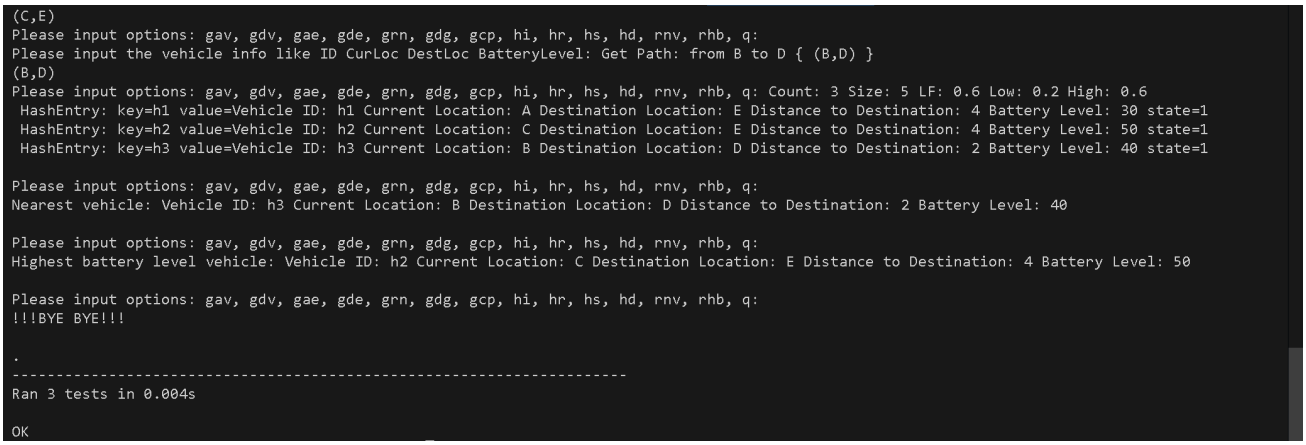
```
Display as list, UNDIRECT, vertex count=3, edge count=1
  V: A [ B ]
  V: B [ A ]
  V: C [ ]

Display edges, edge count=1
  E: A -> B, 2
  E: B -> A, 2

Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the edge like LABEL1 LABEL2 DISTANCE:
Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Display as matrix, UNDIRECT, vertex count=4 edge count=2
  A B C D
A 0 1 0 1
B 1 0 0 0
C 0 0 0 0
D 1 0 0 0

Display as list, UNDIRECT, vertex count=4, edge count=2
  V: A [ B D ]
  V: B [ A ]
  V: C [ ]
  V: D [ A ]

Display edges, edge count=2
  E: A -> B, 2
  E: B -> A, 2
  E: A -> D, 4
  E: D -> A, 4

Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the vertex label:
A neighbors: B, D

Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the vertex label:
D neighbors: A
```

```
Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the edge like LABEL1 LABEL2 DISTANCE:
Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Display as matrix, UNDIRECT, vertex count=4 edge count=3
  A B C D
A 0 1 0 1
B 1 0 1 0
C 0 1 0 0
D 1 0 0 0

Display as list, UNDIRECT, vertex count=4, edge count=3
  V: A [ B D ]
  V: B [ A C ]
  V: C [ B ]
  V: D [ A ]

Display edges, edge count=3
  E: A -> B, 2
  E: B -> A, 2
  E: A -> D, 4
  E: D -> A, 4
  E: B -> C, 3
  E: C -> B, 3

Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the vertex like LABEL1 LABEL2: Get Path: from A to C { (A,B) (B,C) }

There is a path from A to C

Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
!!!BYE BYE!!!
```

```
.
==================================
Interactive Menu for DSA Assignment
  (gav) Graph: Add Vertex
  (gdv) Graph: Delete Vertex
  (gae) Graph: Add Edge
  (gde) Graph: Delete Edge
  (grn) Graph: Retrieve Neighbors
  (gdg) Graph: Display Graph
  (gcp) Graph: Check Path
  (hi) HashTable: Insert
  (hr) HashTable: Remove
  (hs) HashTable: Search
  (hd) HashTable: Display
  (rnv) Recommend: Nearest Vehicle
  (rhb) Recommend: Highest Battery
  (q) Quit
==================================


Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the edge like LABEL1 LABEL2 DISTANCE:
Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the edge like LABEL1 LABEL2 DISTANCE:
Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the edge like LABEL1 LABEL2 DISTANCE:
Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the edge like LABEL1 LABEL2 DISTANCE:
Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the edge like LABEL1 LABEL2 DISTANCE:
Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the edge like LABEL1 LABEL2 DISTANCE:
Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Display as matrix, UNDIRECT, vertex count=5 edge count=6
  A B C D E
A 0 1 1 0 0
B 1 0 0 1 1
C 1 0 0 0 1
D 0 1 0 0 1
E 0 1 1 1 0
```

```
Display as list, UNDIRECT, vertex count=5, edge count=6
  V: A [ B C ]
  V: B [ A D E ]
  V: C [ A E ]
  V: D [ B E ]
  V: E [ B C D ]

Display edges, edge count=6
  E: A -> B, 2
  E: B -> A, 2
  E: A -> C, 3
  E: C -> A, 3
  E: B -> D, 2
  E: D -> B, 2
  E: D -> E, 1
  E: E -> D, 1
  E: C -> E, 4
  E: E -> C, 4
  E: B -> E, 2
  E: E -> B, 2

Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the vehicle info like ID CurLoc DestLoc BatteryLevel: Get Path: from A to E { (A,B) (B,E) }
(A,B) (B,E)
Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q: Count: 1 Size: 5 LF: 0.2 Low: 0.2 High: 0.6
 HashEntry: key=h1 value=Vehicle ID: h1 Current Location: A Destination Location: E Distance to Destination: 4 Battery Level: 30 state=1

Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the vehicle ID:
Find vehicle: Vehicle ID: h1 Current Location: A Destination Location: E Distance to Destination: 4 Battery Level: 30

Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the vehicle ID:
Delete vehicle: h1 success

Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q: Count: 0 Size: 3 LF: 0.0 Low: 0.2 High: 0.6

Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
!!!BYE BYE!!!
```

```
.
==================================
Interactive Menu for DSA Assignment
  (gav) Graph: Add Vertex
  (gdv) Graph: Delete Vertex
  (gae) Graph: Add Edge
  (gde) Graph: Delete Edge
  (grn) Graph: Retrieve Neighbors
  (gdg) Graph: Display Graph
  (gcp) Graph: Check Path
  (hi) HashTable: Insert
  (hr) HashTable: Remove
  (hs) HashTable: Search
  (hd) HashTable: Display
  (rnv) Recommend: Nearest Vehicle
  (rhb) Recommend: Highest Battery
  (q) Quit
==================================


Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the edge like LABEL1 LABEL2 DISTANCE:
Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the edge like LABEL1 LABEL2 DISTANCE:
Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the edge like LABEL1 LABEL2 DISTANCE:
Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the edge like LABEL1 LABEL2 DISTANCE:
Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the edge like LABEL1 LABEL2 DISTANCE:
Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the edge like LABEL1 LABEL2 DISTANCE:
Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the edge like LABEL1 LABEL2 DISTANCE: Input error

Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the vehicle info like ID CurLoc DestLoc BatteryLevel: Get Path: from A to E { (A,B) (B,E) }
(A,B) (B,E)
Please input options: gav, gdv, gae, gde, grn, gdg, gcp, hi, hr, hs, hd, rnv, rhb, q:
Please input the vehicle info like ID CurLoc DestLoc BatteryLevel: Get Path: from C to E { (C,E) }
```



4. Challenges and Solutions

- Checking for Path Existence

  Issue: While checking the path existence, I used DFS to traverse the vertices. I saved the path result in a queue. When I want to dump it for debugging, the queue will be cleared.

  Solution: I use “copy.deepcopy()” to copy a new instance. Then I have 2 copies and 1 for dump.
- Quicksort

  Issue: Quick sort has many implementations: use left-most/random/middle/median-3 element as pivot.

  Solution: Implement all solutions for exercise. Use median-3 element as pivot.

5. Efficiency Analysis and Potential Improvements

- Checking for Path Existence

  Analysis: I used DFS to find the 1st path between A and B. However, it is possible for more than one path to connect A and B, and the paths may have different distances.

  Improvement: Use BFS and DFS to find all possible connection paths, compare their distance, and return the nearest path.

6. Conclusion

In this assignment, I designed and implemented an Autonomous Vehicle Management System (AVMS) that utilises various data structures and algorithms, including Graph to represent the road network, HashTable to store and manage the autonomous vehicles, Heapsort and Quicksort to sort

a vehicle list. This system supports real-time queries about vehicle status and location.

In the system implementation process, I applied much knowledge learned in this course, including LinkedList, Stack, Queue, Graph, HashTable, Heap and advanced sorting.

For each module, a corresponding test case is developed to ensure that the basic functions work well, so that the functions are normal after integration.