Complexity Analysis

Zimraan Anjum, 20778776

# Introduction

The increasing popularity of electric vehicles (EVs) is driven by their environmental benefits, cost savings, and improved driving experience. However, in Australia, the growth of EVs has outpaced the availability of public charging infrastructure. With only 3,700 public chargers across 2,100 locations, EV owners face challenges in finding convenient and cost-effective charging options.

This complexity analysis report aims to address these challenges by providing a solution for EV charging in the Sydney metropolitan area. The analysis focuses on 21 selected locations within the road network, identifying the presence or absence of charging stations. By considering factors such as distance, charging prices, and travel costs, the goal is to optimize charging decisions for EV owners.

Through this analysis, we aim to offer insights and recommendations to enhance EV charging accessibility and efficiency. By considering factors such as starting and destination points, remaining power, travel cost, and charging station availability, EV owners can make informed decisions about where and when to charge their vehicles.

# Data Structures

The data structures that were used within this assignment consists of:

1. Map<int, Location> - This is used to store the locations, where the key is an int index and the value is an instance of the Location class

2. vector<Location>: This vector is used to store the locations with charging stations in the printAscendingPrice() function.

3. WeightedGraphType\*: This is a pointer to an object of the WeightedGraphType class. It represents a weighted graph.

4. list<int>: This list is used to store the adjacency list of each location in the printAdjacentLocations() function.

5. double \*\*weights: This is a 2D array used for storing the weights of edges in the graph. It is dynamically allocated and represents the adjacency matrix.

6. bool \*weightFound: This is a dynamically allocated array used for tracking the vertices for which the smallest weight has been found during the shortest path algorithm.

7. queue<int>: This is a queue implementation from the STL used for implementing breadth-first traversal of the graph.

# Algorithms

The three significant algorithms implemented in the program are as follows:

Algorithm 1: From Task 6, printNearestChargingStation, this algorithms aim is to find the nearest charging Station from the current location. It accomplishes this by using Dijkstra’s algorithm.

Explanation:

1.Prompt the user to enter their current location.

1. Find the index of the current location in the locations map.
2. If the current location is not found, display an error message and return.
3. Use Dijkstra's algorithm to calculate the shortest distances from the current location to all other locations.
4. Find the nearest charging station by iterating over all locations with chargers (excluding the current location) and finding the one with the lowest distance.
5. If a nearest charging station is found, display its name and the corresponding distance. Otherwise, display a message indicating that no nearest charging station was found.

Algorithm 2: From Task 8, findMinCostPath(), this aim of this function is to finds the path with the minimal charging cost and travel cost between two specified cities.  
  
1. Find the index of the origin and destination cities in the list of locations.

2. If the origin city is not found, print an error message and return.

3.If the destination city is not found, print an error message and return.

4.Get the adjacency list for the origin city.

5.Initialize variables for tracking the lowest cost and nearest location index.

6.Iterate through each adjacent location to the origin:

a. Check if it is the destination city with a charger installed.

b. If it is, calculate the distance, charging cost, travel cost, and total cost for this direct connection.

c. If the total cost is lower than the current lowest cost, update the lowest cost and nearest location index.

7.If a nearest location is found:

a. Print the origin, destination, charging amount, and total cost.

b. Set the current location index to the origin.

c. Print the initial path.

d. While the current location index is not the destination:

8.Get the adjacency list for the current location.

9.Initialize variables for tracking the minimum cost and next location index.

10.Iterate through each adjacent location to the current location:

11.Calculate the distance, charging cost, travel cost, and total cost for each connection.

12.If the total cost is lower than the current minimum cost, update the minimum cost and next location index.

13.If a next location is found:

14.Print the next location.

15.Set the current location index to the next location index.

16.Otherwise, break the loop.

17.If no nearest location is found, print an error message indicating that no path with the minimal charging cost and travel cost was found.

Algorithm 3: From Task 3, printAscendingPrice(), this algorithms prints the location with charging stations in asencding order of charging price. It utilizes the sort() function to sort the vector efficiently.

Explanation:

1.Create an empty vector to store locations with charging stations.

2.Iterate over all locations and add those with charging stations to the vector.

3.Sort the vector in ascending order based on the charging price.

4.Print the sorted locations.

This algorithm utilizes the sort() function to sort the vector efficiently.

# 4. N-Notation

Algorithm 1:

* Worst case: O(n + m)

This is because the algorithm has to find the shortest path from current to all location indicating that in a worst-case scenario it’ll have to visit every location.

Algorithm 2:

* Worst case: O(n + m)

The reasoning behind this is because both obtaining the adjacency list of the origin city as well as the calculation of min cost path result in O(m). And we also add in the task of finding the indices of the origin and destination cities, and having to loop through them all would give us O(n). Thus, we end up in O(n+m)

Algorithm 3:

* Worst Case: O (n log n), n is the number of locations.

We come t this result by first populating the vector which is O(n) but then when we get to the sort function the n notation is O( n log n). A

# 5. Conclusion

In conclusion, we can see that there is a use for algorithms when completing this tasks, such as incorporating the Dijkstra algorithm in order to find the shortest path between locations.