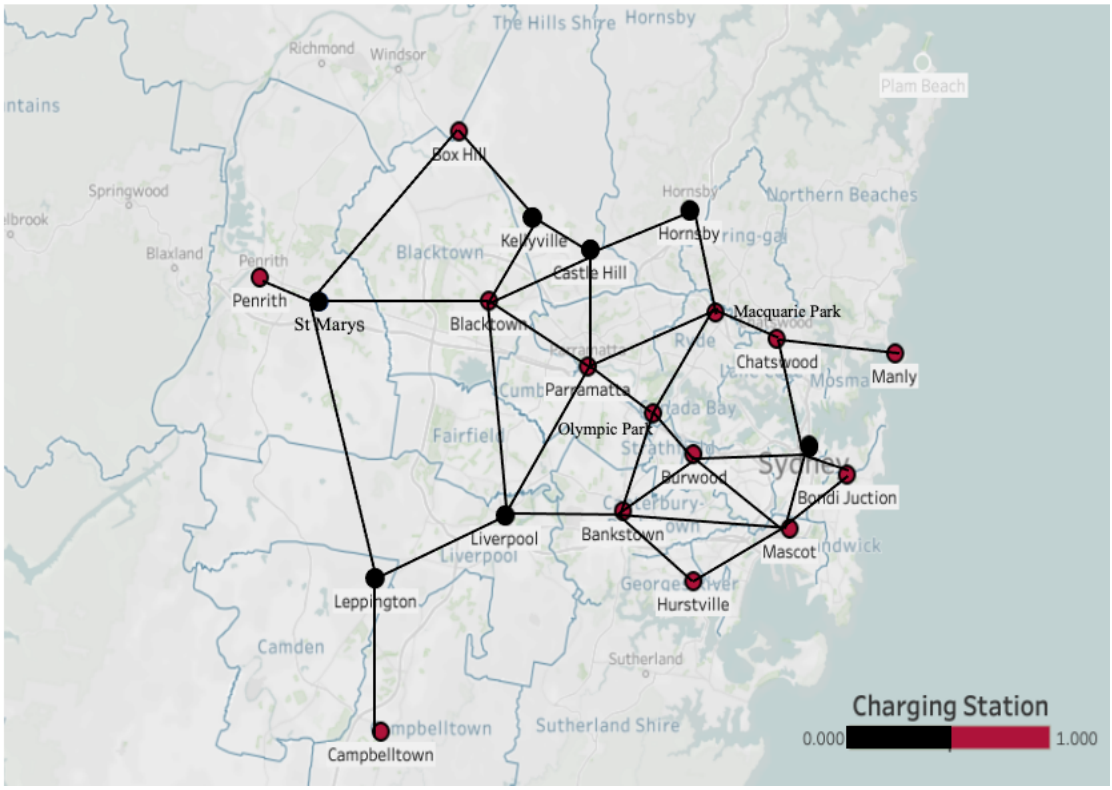
Assignment 2
Electric Vehicle Charging
Submit Deadline: 5pm 28 May 2023

1. Problem description

Electric vehicles (EVs) have become more and more popular due to advantages of less carbon dioxide emissions, lower running costs and better driving experience. In Australia, the growth rate of EVs in recent years has been around 10%. In 2022, the number of EVs sold in Australia was 33,410[1]. However, there are only around 3700 public chargers, at just over 2,100 locations, across Australia[2]. Imagine that you are driving an electric vehicle on a road in Sydney metropolitan. If your vehicle has nearly run out of power and you are going to find the nearest charging station. Or you still have a bit power left but you want to find the cheapest charging station within the distance that the remaining power can travel. Or you plan to travel from one location to another location. You want to find out how many charging stations are available along the shortest path between the two locations.



[1] https://thedriven.io/2023/01/11/australian-ev-sales-data-for-2022-by-model-with-2023-boom-to-follow/
[2] https://www.theguardian.com/environment/2023/mar/05/power-vacuum-how-a-lack-of-charging-stations-is-holding-back-australias-ev-revolution

Figure 1. *Road network of Sydney metropolitan*.

This assignment is to provide a solution to some of the problems related to EV charging.

As shown in **Figure** 1, we selected 21 locations in the road network of Sydney metropolitan. A location with a red dot means that the location has a charging station[3]. A location with a black dot means that no charging station is available there. The file *Appendix.xlsx* contains the information of the distance between adjacent locations and the information about the locations, including their names, charging station availability, and charging price[4]. Assume that each free charging station has a limitation of charging up to 25kWh per vehicle.

There are a few factors that affect an electric vehicle to choose where to charge - origin (starting/current location), destination, remaining power (travel range), travel cost, amount of power to charge, and charging cost. All these factors vary with different vehicles in difference situations. For simplicity, we assume that the cost for all EVs is $0.10/km (including power consumption and running costs).

For an easy start, a package *Assignment2BaseCode.zip* is provided, which contains the following files:

- *Location.h* – a class specifying the information of each location.
- *WeighghtedGraph.h* – a class that implements the basic components of a weighted graph data structure.
- *EVCharging.h* – a class that encapsulates the information of road network (weighted graph) and charging infrastructure (locations).
- *EVChargingApp.cpp* – the driver of the program.
- *Locations.txt* – data file contains the information of locations (vertex data).
- *Weights.txt* – data file contains the distance information of adjacent locations (adjacency matrix with weights).

The code should be runnable on any platform. It is designed as a base for you to further work on to complete the tasks listed in the following section.

2. Programming tasks (85 marks)

The assignment consists of two parts – programming and complexity analysis. You are required to complete the programming tasks based on the base code as mentioned above. You may change any part of the base code if needed. You may use any programs provided by the subject, including the code for lectures, practicals and tutorials, without referencing. You may also use the algorithms from the Internet but you must reference the source clearly. You are highly recommended to use C++ to implement your solution. Other general-purpose languages, such Java or Python, can be used. However, you must literally translate the base code to the language you use.

[3] For simplicity, we do not consider waiting time of a charging station. We simply assume that any EV can charge at any location with a charging station.
[4] Download the file from vUWS under Assignment 2 folder.

Task 1 (10 marks): Download *Assignment2BaseCode.zip*. Read the code and fully understand it. Run the code so that you can see the list of locations and adjacency matrix with weights.

Task 2 (10 marks): Add three additional locations to the road network by adding related data into the data files. Make your own selection of the locations[5]. You may use google map or other maps to find the adjacent locations and the distance to their adjacent locations. You may make up the data for availability of charging station and charging price. Manually add the related data into the files - *Location.txt* and *Weight.txt*. Make sure the program still running properly with the extended files[6].

Task 3 (10 marks): Add a function to the `EVCharging` class to list all the locations with charging stations in ascending order of charging price. You may choose your own sorting method. **Hint:** *You may transfer all the data in the map (`locations`) into a vector or a priority queue so that you can use STL data structures and algorithms to sort the locations based on charging price.*

Task 4 (10 marks): Extend your program so that it can take an input of a location and output the list of adjacent locations with charging stations. If none of the adjacent locations has a charging station, output NONE. **Hint:** *The `list<int>` pointer `graph` in `WeightedGraphType` class stores adjacency list (call the getAdjancencyList to get the data).*

Task 5 (10 marks): Extend your program so that it can take an input of the current location and find the adjacent location with the lowest charging cost[7]. Assume that the travel cost of an EVs is $0.10/km. Charging amount ranges from 10kWh to 50kWh (can be randomly generated). For the travel cost, you need also to consider the cost of return (back to the original location). For instance, if the distance to an adjacent location is 15km, the charging price at that location is $0.45/kWh, and the vehicle needs to charge 40kWh, the total cost of charging and travel is:

$$15km * 2 * \$0.10/km + 40kWh * \$0.45/kWh = \$21$$

Hint: *You only need to consider the charging stations adjacent to the given location*.

Task 6 (10 marks): Extend your program so that it can take an input of the current location and output the nearest location with a charging station (excluding the current charging station). Note that it is not necessarily adjacent to the current location. **Hint:** *Use `shortestPath` function to find the shortest distances to all other locations from this location.*

Task 7 (10 marks): Extend your program so that for any given current location, it can find another location with the lowest total cost of traveling and charging (also excluding the current charging station). Assume that charging amount randomly ranges from 10kWh to 50kWh. The same as Task 5, the travel cost of an EVs is $0.10/km. *You also need to consider the two-way travel*. Assume that one vehicle charges only once at one location[9].

---

[5] Try not to be the same as others.

[6] Please note that the program is very sensitive with the format of the text files. Space and delimiters matter. Be careful with copy-and-paste across platforms which would carry invisible characters.

[7] You may imagine that the current location has no charging station, full or out of service.

[9] With this assumption, any vehicle that want to charge more than 25kWh cannot be charged at free charging stations.

Task 8 (10 marks): Extend your program so that for any given *origin*, *destination*, and amount of power to be charged (ranging between 10kWh and 50kWh[11]), it can find a path with the minimal charging cost and travel cost. Assume again that the travel cost of an EVs is $0.10/km and one vehicle charges only once at one location. Different from Task 7, you only need to consider one way travel from the origin to the destination.

Task 9 (5 marks): Redo Task 8 by allowing one vehicle can charge multiple times in different locations. In this case, free charging stations can be used if an EV needs to charge more than 25kWh power.

Task 10 (10 bonus marks): Extend your program with insert and delete functions in running time (not by changing the input files). You may need to create two insert functions, one for taking a new vertex and the other to take a new edge. Delete functions are the same. Hint: *You may need to change the implementation of graph from 2-dimensional array to a vector of vectors or set a maximum size for the array*.

Task 10 (10 bonus marks): Translate your whole solution from C++ to Java or Python (you must have C++ code to gain the bonus marks)

You can only claim up to 10 bonus marks.

3. Complexity analysis (15 marks)

In this assignment, you are required to submit a formal document no matter how many programming tasks you have implemented. The document should explain the algorithms and data structures you have used in your code (**including the code provided in the base code**). You should also use the document to demonstrate the depth of your understanding of the data structures you have used in the program. Your document should include the following sections:

1. **Introduction:** demonstrate your understanding of the problem. (2 marks)
2. **Data Structures:** List all the major data structures that have been used in the whole program including the once in the base code, the additional code provided and the code you implemented. (3 marks)
3. **Algorithms:** Choose three algorithms you have implemented for **Tasks** 3-9 in your program to explain your idea of implementation using pseudocode or simplified source code (list the major operations only). You should choose the most significant algorithms to demonstrate your skill of programming and the depth of your understanding. (3 marks)
4. **Complexity analysis:** Estimate either worst-case or average complexity of the three algorithms you have chosen above. Assume that the total number of locations is *n* and total number of edges in the graph is *m*. You do not have to give both worst-case and average-case complexity[12]. (6 marks)
5. **Conclusion**: Summarize what you did. (1 marks)

4. Submission

[11] Can either be randomly generated or be given by the user.

[12] If you consider worst-case complexity, you may assume that each vertex has n degree thus you do not need to consider the number of edges.

All source code and documentation should be submitted via vUWS before the deadline for documentation purpose. Your programs (.h, .cpp or .java) can be put in separate files (executable file is not required). The document of your complexity analysis can be submitted in either Word file or PDF file. All these files, including your declaration text file, should be zipped into one file **with your student id as the zipped file name**. Submission that does not follow the format will not be accepted. **Email submission is not acceptable (strict rule).**

All students are required to submit a document (or as a comment in the file where the main function is), containing the following

DECLARATION

/*

I hereby certify that no other part of this submission has been copied from any other sources, including the Internet, books, or other student's work, or generated from generative AI tools, such as ChatGPT except the ones I have listed below:

// List the part of code you acquired from other resources

I hold a copy of this assignment that I can produce if the original is lost or damaged.

*/

You do not have to declare if the code is from practical tasks, example code in the lectures, or anything supplied by the IDE or in the Standard Template Library. Contact your tutor if you are not sure.

5. Demonstration

You are required to demonstrate your program during **your scheduled** practical session in Week 12. Your tutor will check your code and your understanding of the code. **You will receive no marks if you fail the demonstration, especially if you miss the demo time.** Note that it is students' responsibility to get the appropriate compilers or IDEs to run their programs. You are allowed to run your program from your laptop at the specified demo time. **The feedback to your work will be delivered orally during the demonstration.** No further feedback or comments are given afterward.

The demonstrated program should be the same as the one you submitted to vUWS. If the mark of your demonstration will be the final mark of the assignment.

Again, do not send us your work via email. Programming doesn't work in that way. Face-to-face is the most efficient way to check your work.

Detailed marking criteria will be specified in a separate document.

Have fun!