**Complexity Analysis**

**Introduction:**

Demonstrate your understanding of the problem.

* Program is designed to solve problems involving an electric vehicle that wants to travel from location to location.
* It explores the topic of graph theory, which means dealing with an adjacency list, and a weighted matrix.

**Data Structures:**

List all the major data structures that have been used in the whole program, including the ones in the base code, the additional code provided, and the code you implemented.

* vectors
* queues / priority queues
* hash maps

**Algorithms:**

Choose three algorithms you have implemented for your tasks in your program to explain your idea of implementation using pseudocode or simplified source code (list the major operations only).



* takes two arguments:
  + (int) index of the origin
  + (int) amount to charge
* uses a slightly modified shortest path find function to generate a vector that holds the destination’s index and the path as a pair
* iterates through the container to find suitable paths to a location with a charging station
* a path is deemed suitable if:
  + the path’s destination is not the path’s origin
  + the path’s destination has a charging station
  + the aforementioned charging station is suitable for when the energy requirement is above 25kWh (because then you can’t use the free charging stations)
* once the path is deemed suitable, calculate the total cost of the trip and store it
* repeat this entire process again, comparing every suitable path’s total cost until you find the path with the lowest total cost
* then return both with the path’s destination’s index and it’s total cost
* note: that the total cost of the trip includes the return trip



* takes three arguments:
  + (int) index of the origin
  + (int) amount to charge
  + (int) index of the destination
* uses a slightly modified shortest path find function to generate a vector that holds the destination’s index and the path as a pair
* iterates through the container to find suitable paths to a location with a charging station
* a path is deemed suitable if:
  + the path’s destination has a charging station
  + the aforementioned charging station is suitable for when the energy requirement is above 25kWh (because then you can’t use the free charging stations)
* repeat this again to find suitable paths from the previous path’s destination location to the destination taken by the argument
* this means you now have both the path to the place with the charging station then a path from there to the end destination
* once you have the complete path, calculate the total cost of the trip and store it, alongside other important information
* repeat this entire process again, comparing every suitable path’s total cost until you find the path with the lowest total cost (and storing other important information)
* then return the path



* similar to above, except under the right conditions, it will consider a second charging station too
* the right condition being if the amount to charge is over the free charging station limit, then consider multiple FREE charging stations
* this means that instead of finding a path from A to B to C, you consider an additional path
* this means you would consider A to B to C to D, wherein B and C are charging stations
* once you have the complete path, then calculate the total cost of the trip, and store it, alongside other important information
* repeat this entire process again, comparing every suitable path’s total cost until you find the path with the lowest total cost (and storing other important information)
* then return the path

**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.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Functions** | | | | **O(n)** | **Overall O(n)** |
| **location\_with\_charging\_station\_lowest\_total\_cost\_find();** | | | | | **O(n2)** |
| **contains:** | * shortest\_path\_find(); | | | O(n2) |  |
|  | * for loop | | | O(n) |  |
| **best\_path\_one();** | | | | | **O(n3)** |
| **contains:** | * shortest\_path\_find(); | | | O(n2) |  |
|  | * for loop | | | O(n3) |  |
|  | **contains:** | * shortest\_path\_find(); | | O(n2) |  |
|  |  | * for loop | | O(n) |  |
| **best\_path\_two();** | | | | | **O(n4)** |
| **contains:** | * shortest\_path\_find(); | | | O(n2) |  |
|  | * for loop | | | O(n4) |  |
|  | **contains:** | * shortest\_path\_find(); | | O(n2) |  |
|  |  | * for loop | | O(n3) |  |
|  |  | **contains:** | * shortest\_path\_find(); | O(n2) |  |
|  |  |  | * for loop | O(n) |  |

**Conclusion:**

Summarise what you did.

Using weighted graphs and their algorithms, and the Dijkstra’s algorithm, we can perform complex searches to find optimal paths to locations. With optimal being contextual.