COMP2034 Coursework 2 Report

Garbage Management System

Prepared for

Dr. Doreen Sim Ying Ying

University of Nottingham Malaysia

22 April 2024

By

Lua Chong En

20417309

Table of Contents

1. Introduction

This coursework focuses on creating an Automated Garbage Management System. In Malaysia, the management of garbage collection presents significant issues that harm the environment. Some of these challenges regarding garbage collection include: excessive bacterial growth, foul smell, and increase of littering. This is mainly due to either overdue or early garbage collection.

Therefore if garbage is collected too late, challenges of bacterial growth and foul smell can arise. Whereas if garbage is collected too early, challenges of excessive resource utilisation and unoptimized garbage collection trips can arise. To address these issues efficiently and in an organised manner, an intricate Garbage Management System is required.

This report aims to create an automated solution by implementing an efficient Garbage Management System using C++ programming. The system aims to optimise garbage collection by using 4 different algorithms: Unoptimized Route - Fixed Shortest Path, Greedy Route - Dijkstra Algorithm, Optimised Route - Floyd Warshall Algorithm, and Travelling Salesman Route - Greedy Algorithm.

The following sections detail the new proposed Garbage Management System. Also included in this report is the design and the detailed implementation of the system. It highlights the solutions in controlling and optimising the environmental and health hazards. Aiming to ultimately reduce resource consumption whilst also increasing overall garbage management efficiency.

2. Project Scope

The Automated Garbage Management System is an extensive solution built to solve the existing Garbage Management practices in Malaysia. By implementing four different shortest path algorithms with the aim to enhance route optimisation and cost calculation along the shortest path, thereby, displaying improvements on efficiency and cost-effectiveness of the Garbage Management System.

Improved optimization ensures that garbage is collected in a timely manner, preventing prevalent issues previously outlined such as bacterial growth, foul smells, and littering. Therefore the Automated Garbage Management System represents a significant improvement in waste management. It offers a better and more sustainable solution to the existing Garbage Management Collection System in Malaysia.

3. Project Aim

The primary aim of the project is to implement a new and improved Garbage Management System. Looking to revolutionise the method in which garbage collection is performed in Malaysia. Leveraging four different shortest path algorithms to route optimisation and cost calculation, hence leading to enhanced waste management practices. The system also aims to minimise the environmental impact by reducing the resource utilisation whilst also improving overall efficiency.

Resource utilisation is affected by several factors: fuel consumption, inefficient trips or possibly inefficient routes. Heavy fuel consumption leads to high carbon emissions which is bad for the environment, it links closely to inefficient trips as some garbage collections may not be required at the point in time and yet it is still performed. Furthermore, inefficient routes signify that the shortest path is suboptimal in terms of the time taken, distance, and also the resource utilisation. Inefficient routes may involve backtracking or overlapping over the same locations.

Another aim of the project is to boost the overall cleanliness in Malaysia. By thoroughly optimising the Garbage Management System, it can reduce any challenges faced by the community which may involve littering or overflowing rubbish bins.

To summarise, the proposed Garbage Management System aims to assist and contribute to a more sustainable environment in Malaysia.

4. Libraries used

“<iostream>”: This library is used for the input and output operations in C++. It provides the functionality for reading from and writing to the standard input/output streams.

“<string>”: This library is used for strings in C++. It contains functions for string concatenation, comparison, and substring operations.

“<vector>”: This library is used for a dynamic array data structure in C++. The vector can grow and shrink in size. Therefore making it easy to store information dynamically.

“<stdio.h>” and “<stdlib.h>”: This is a C standard library and the functions are used for input/ output operations and memory allocation/deallocation. It assists with the compatibility of the code.

“<ctime>”: This library is used for using date and time in C++. Retrieving the current date and time, using it as a seed for the random waste level generator.

“<cmath>”: This library is used for maths related functions in C++.

“<map>”: The library is used to store key-value pairs in C++. The distance map used in this coursework uses a map to define the distance matrix.

“<limits>”: This library uses constants that represent the limits of various data types in C++. “INF” uses the limits in C++.

“<algorithm>”: This library provides a collection of functions for working with containers in C++.

“<tuple>”: This library is used to store a fixed-size collection of elements in C++.

“<climits>”: This library uses constants for limits of integral types, like “INT MAX”.

“<fstream>”: This library is used for file input/ output operations in C++. File input and output are done using this library. File outputs using fstream are used in this coursework widely.

5. Assumptions

All four algorithms perform the same function of finding the shortest path from the Waste Station to the target location (according to the conditions applied to each algorithm). The output of each algorithm is in the same format, therefore providing consistent output in terms of total distance, cost, time, fuel consumption, waste collected, and driver wage.

However, each algorithm operates differently in finding this shortest path. For Non-Optimised Route, it uses a fixed path algorithm to find the shortest path to every target location. For Greedy Route, it uses the Dijkstra Algorithm to find the shortest path to every target location. For Optimised Route, the Floyd Warshall Algorithm is used to find the shortest path to every target location. Finally for TSP Route, the Travelling Salesman Problem is used to find the shortest path to every target location. Therefore, every algorithm is different and operates differently to find the shortest path.

The conditions applied to each algorithm balances out each individual strengths and weaknesses, allowing them to operate at their optimal efficiency. Therefore, for Non-Optimised Route, the waste level at the location must be greater than or equal to forty, whereas, for Greedy Route, the waste level at the location must be greater than thirty. Greedy Route is generally more dynamic and optimised than Non-Optimised Route which uses a fixed-path algorithm, hence the waste levels are altered to allow each algorithm to operate fairly in the program.

Furthermore, the users using my program are Garbage Collection Companies. Therefore there are many garbage trucks and multiple routes, not limited to 1 garbage truck nor 1 single route.

Users have the ability to regenerate waste levels, updating them to test which algorithm performs best under the waste levels. New waste levels may possibly make algorithms visit new locations and hence alter and modify the overall efficiency of each algorithm.

All algorithms operate on the given graph and distance map. The graph is customised and unique. The distance map is used to define the distances from a location to every

other location. The value used frequently “INF” defines that there exists no direct path from the location to the other location.

Users can constantly reuse and repeat algorithms in the program. There is no set limit on how many times the algorithms can be used or how many times the program can be re-run and restarted.

Furthermore, the cumulative total percentage of waste is calculated. Then the waste in kilograms (kg) is calculated as well for ease of visualisation. The maximum total waste at a location is capped at 500 kg.

Finally, note that there are a total of 8 locations. Waste Station which is the source, and locations 1 - 7. Therefore a cumulative total of 8 locations.

6. Literature Review

a. Non-Optimised Route

A fixed shortest path was chosen as the primary routing algorithm. The main reason was due to its simplicity and straightforward calculation when calculating shortest path and all other metrics. Therefore, it aims to provide a straightforward method without considering any dynamics within the distance matrix/ graph. It calculates the shortest path to every valid location starting from Waste Station by following a pre-defined shortest path. The pre-defined shortest path does not adjust or change according to the routing to every valid location.

b. Greedy Route

The Dijkstra algorithm was chosen as the routing algorithm for Greedy Route. The Dijkstra algorithm is also known as a single-source shortest path algorithm, meaning that it starts from the Waste Station and routes by finding the shortest path to every valid location. It is known to be efficient whilst also having good performance in finding the optimal path and solution.

Dijkstra operates by picking the unvisited vertex with the lowest distance. Afterwards, it calculates the distance through the unvisited vertex to each unvisited neighbour and updates the neighbours distance only if the distance is smaller.

c. Optimised Route

For Optimised Route, Floyd Warshall algorithm was chosen. Although other similar algorithms like Prims Algorithm, Kruskal Algorithm, and Bellman Ford Algorithm were considered. However, Floyd Warshall stood out as the primary algorithm to be implemented. The Floyd Warshall Algorithm considers all possible combinations from Waste Station to all valid locations and it allows for an optimised and comprehensive approach that does not sacrifice efficiency and performance of the algorithm.

Floyd Warshall operates by iteratively updating a matrix that consists of shortest path distances until the matrix finally converges towards a final state.

d. TSP Route

For the TSP route, the Travelling Salesman Problem algorithm with Greedy was used. TSP is designed to solve the problem of finding the shortest path from the Waste Station to every valid location in the distance matrix and returning to the starting point. However, in this coursework, do note that the TSP algorithm is slightly modified so that it returns the shortest distance path from Waste Station to every valid location without returning to Waste Station.

TSP Greedy operates by sorting the edges in the graph, then selecting the shortest edge and adding it into the list.

7. Distance Map

All locations are stored in a distance matrix using 2 maps. There are a total of 8 locations: Waste Station / Location 0 / Source, Location 1, Location 2, Location 3, Location 4, Location 5, Location 6, and Location 7.

The first map is map<string, int> wasteLevels. This map stores the waste levels of each location. The key is the location name, and the value is the randomly generated waste level, from 0 to 100.

The second map is map<string, map<string, int>> distanceMatrix. This map stores the distances between locations. It's a nested map where the outer key is the starting location name, the inner key is the destination location name, and the value is the distance between them.

Within the garbageLocations constructor, the constructor initialises the distanceMatrix and wasteLevels maps. It sets the distances between locations based on a predefined matrix and sets the waste levels of each location randomly between 0 and 100. The distanceMatrix is initialised with distances between locations, using "INF" for locations that are not directly connected, distance for locations that point to themselves is set to 0 by default. The wasteLevels are initialised randomly using rand() % 101 and a seed using the current time, which generates a random number between 0 and 100 inclusive.

Refer below the location names and the distance matrix.

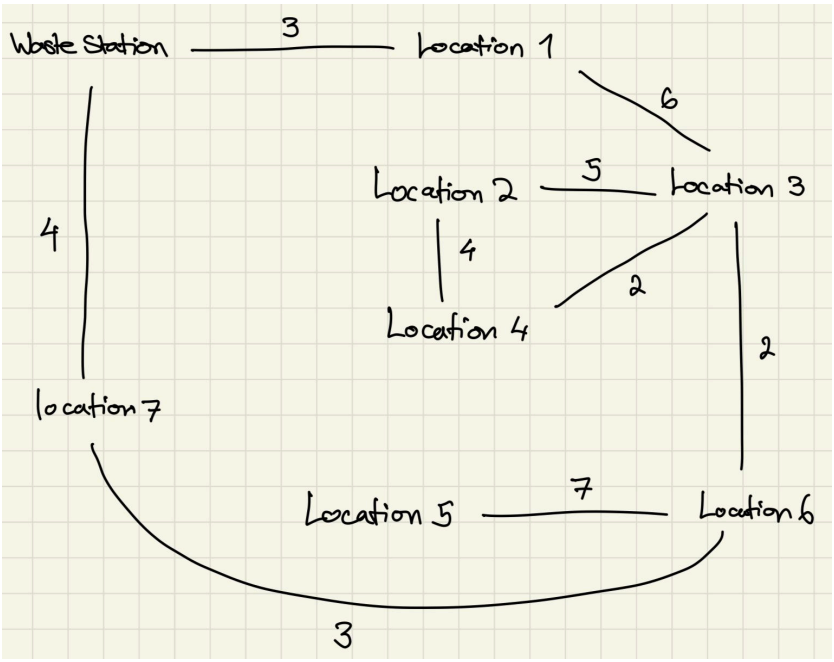
```
locations.push_back("Waste Station");
locations.push_back("Location 1");
locations.push_back("Location 2");
locations.push_back("Location 3");
locations.push_back("Location 4");
locations.push_back("Location 5");
locations.push_back("Location 6");
locations.push_back("Location 7");

distanceMatrix["Waste Station"]["Waste Station"] = 0;
distanceMatrix["Waste Station"]["Location 1"] = 3;
```

```
distanceMatrix["Waste Station"]["Location 2"] = INF;
distanceMatrix["Waste Station"]["Location 3"] = INF;
distanceMatrix["Waste Station"]["Location 4"] = INF;
distanceMatrix["Waste Station"]["Location 5"] = INF;
distanceMatrix["Waste Station"]["Location 6"] = INF;
distanceMatrix["Waste Station"]["Location 7"] = 4;

distanceMatrix["Location 1"]["Waste Station"] = 3;
distanceMatrix["Location 1"]["Location 1"] = 0;
distanceMatrix["Location 1"]["Location 2"] = INF;
distanceMatrix["Location 1"]["Location 3"] = 6;
distanceMatrix["Location 1"]["Location 4"] = INF;
distanceMatrix["Location 1"]["Location 5"] = INF,
distanceMatrix["Location 1"]["Location 6"] = INF;
distanceMatrix["Location 1"]["Location 7"] = INF;

distanceMatrix["Location 2"]["Waste Station"] = INF;
distanceMatrix["Location 2"]["Location 1"] = INF;
distanceMatrix["Location 2"]["Location 2"] = 0;
distanceMatrix["Location 2"]["Location 3"] = 5;
distanceMatrix["Location 2"]["Location 4"] = 4;
distanceMatrix["Location 2"]["Location 5"] = INF;
distanceMatrix["Location 2"]["Location 6"] = INF;
distanceMatrix["Location 2"]["Location 7"] = INF;

distanceMatrix["Location 3"]["Waste Station"] = INF;
distanceMatrix["Location 3"]["Location 1"] = 6;
distanceMatrix["Location 3"]["Location 2"] = 5;
distanceMatrix["Location 3"]["Location 3"] = 0;
distanceMatrix["Location 3"]["Location 4"] = 2;
distanceMatrix["Location 3"]["Location 5"] = INF;
distanceMatrix["Location 3"]["Location 6"] = 2;
distanceMatrix["Location 3"]["Location 7"] = INF;
```

```
distanceMatrix["Location 4"]["Waste Station"] = INF;
distanceMatrix["Location 4"]["Location 1"] = INF;
distanceMatrix["Location 4"]["Location 2"] = 4;
distanceMatrix["Location 4"]["Location 3"] = 2;
distanceMatrix["Location 4"]["Location 4"] = 0;
distanceMatrix["Location 4"]["Location 5"] = INF;
distanceMatrix["Location 4"]["Location 6"] = INF;
distanceMatrix["Location 4"]["Location 7"] = INF;

distanceMatrix["Location 5"]["Waste Station"] = INF;
distanceMatrix["Location 5"]["Location 1"] = INF;
distanceMatrix["Location 5"]["Location 2"] = INF;
distanceMatrix["Location 5"]["Location 3"] = INF;
distanceMatrix["Location 5"]["Location 4"] = INF;
distanceMatrix["Location 5"]["Location 5"] = 0;
distanceMatrix["Location 5"]["Location 6"] = 7;
distanceMatrix["Location 5"]["Location 7"] = INF;

distanceMatrix["Location 6"]["Waste Station"] = INF;
distanceMatrix["Location 6"]["Location 1"] = INF;
distanceMatrix["Location 6"]["Location 2"] = INF;
distanceMatrix["Location 6"]["Location 3"] = 2;
distanceMatrix["Location 6"]["Location 4"] = INF;
distanceMatrix["Location 6"]["Location 5"] = 7;
distanceMatrix["Location 6"]["Location 6"] = 0;
distanceMatrix["Location 6"]["Location 7"] = 3;

distanceMatrix["Location 7"]["Waste Station"] = 4;
distanceMatrix["Location 7"]["Location 1"] = INF;
distanceMatrix["Location 7"]["Location 2"] = INF;
distanceMatrix["Location 7"]["Location 3"] = INF;
distanceMatrix["Location 7"]["Location 4"] = INF;
distanceMatrix["Location 7"]["Location 5"] = INF;
distanceMatrix["Location 7"]["Location 6"] = 3;
```

```
distanceMatrix["Location 7"]["Location 7"] = 0;
```

Refer to the image below a basic diagram of the distance matrix in a visual form/ graph.



*Note: The waste level at “Waste Station” is always set to 0.

*Note: The distance between the location and itself is set to 0 by default.

8. Algorithms Involved

a. Algorithm 1: Non-Optimised Route

The Non-Optimised Route operates using a fixed shortest distance path. The fixed shortest distance path is a manually predefined path. It provides the functionality to calculate and print the shortest path from the “Waste Station” to each location in the distance map. The 3 conditions that must be satisfied are 1) Waste at location must be >=40% to be visited and collected 2) Distance from the Waste Station to location must be <=30 km 3) Time taken must be within 12 hours, or else the location will not be visited.

It iterates over each location, then checks the first condition. Checking if the waste at the location is greater than or equal to 40%. After checking the first condition, the location will pass through the second condition, checking if the distance from the waste station to the location is less than or equal to 30 km. If both conditions are satisfied then the location visited is valid. Therefore the location will be visited and garbage will be collected. The final condition will check the time it takes to visit all locations, if the time taken is under 12 hours to visit all valid locations, the visited location will remain. However, if the time taken is over 12 hours, then the location will not be visited and then the location will be flagged.

It also calculates the total cost, time taken, fuel consumption, and driver wage for all visited locations. The total cost is calculated by multiplying the total distance by 2.5. The time taken is calculated by multiplying the total distance by 0.2. Finally the fuel consumption is calculated by multiplying the total distance by 2.

Furthermore, the cumulative total percentage of waste is calculated. Then the waste in kilograms (kg) is calculated as well for ease of visualisation.

Overall, the NonOptimizedRoute class is a straightforward approach to calculating and displaying the shortest path from the Waste Station to every location without optimising the routes.

b. Algorithm 2: Greedy Route

The greedy route operates by using Dijkstra Algorithm. The Dijkstra Algorithm generates a Shortest Path Tree (SPT) with a given source as the root, which in this project's case is the Waste Station. There are 2 sets, one set that contains the vertices included in the SPT and the other set that contains vertices not yet included in the SPT. And at every step of the algorithm, Dijkstra Algorithm finds a vertex that is in the other set and has a minimum distance from the source.

The time complexity of Dijkstra Algorithm is $O(V^2)$ and space complexity of $O(V)$.

There are 3 conditions that are to be satisfied for locations to be visited in the Greedy Route Algorithm. 1) Waste at the location must be greater than or equal to 30% 2) There is no distance restriction from “Waste Station” to the location unlike other algorithms 3) Time taken must be within 8 hours, or else location will not be visited.

The “buildGraph” method initialises the distance matrix based on the provided locations and the associated distances. Then the “printPath” method recursively prints the path from the Waste Station to all valid locations. The “printSolution” method finds the shortest path from the Waste Station to each location. The “minDistance” method finds the minimum distance from the current location to the unvisited location following with the implementation of Dijkstra Algorithm.

It also calculates the total cost, time taken, fuel consumption, and driver wage for all visited locations. The total cost is calculated by multiplying the total distance by 2.5. The time taken is calculated by multiplying the total distance by 0.2. Finally the fuel consumption is calculated by multiplying the total distance by 2.

Furthermore, the cumulative total percentage of waste is calculated. Then the waste in kilograms (kg) is calculated as well for ease of visualisation.

Overall the Greedy Route uses the Dijkstra Algorithm to provide a reasonable solution in finding the shortest path for waste collection optimisation.

c. Algorithm 3: Optimised Route

The optimised route operates by using Floyd Warshall Algorithm. The Floyd Warshall Algorithm finds the shortest path from the “Waste Station” to each location. Unlike Dijkstra Algorithm which is a single source shortest path algorithm, only starting from “Waste Station”, the Floyd Warshall Algorithm is more dynamic and flexible; it works for both directed and undirected graphs. It however does not work with graphs that have negative cycles. The Floyd Warshall approach checks every possible path by visiting every possible location

in order to successfully calculate the shortest path between every pair of locations.

The algorithm operates on a graph that represents all the distances between locations. The “buildGraph” method helps to initialise the distance matrix based on all garbage locations from the garbageLocations class. The “printShortestPaths” method prints the shortest path from the “Waste Station” to every valid location. Furthermore, the Floyd Warshall algorithm implements the Floyd Warshall algorithm to find the shortest path to every valid location starting from the “Waste Station”. It starts by initialising the shortest distance matrix and predecessor matrix based on the graph created by “buildGraph”, then iteratively updates the shortest distance matrix and predecessor matrix through an iterative loop. The “printShortestPaths” will then finally print the results.

The time complexity of Floyd Warshall Algorithm is $O(V^3)$ and space complexity of $O(V^2)$.

There are 3 conditions that are to be satisfied for locations to be visited in the Optimised Route Algorithm. 1) Waste at the location must be greater than or equal 50% 2) The distance from “Waste Station” to every valid location must be within 11km 3) Time taken must be within 8 hours, or else location will not be visited.

It also calculates the total cost, time taken, fuel consumption, and driver wage for all visited locations. The total cost is calculated by multiplying the total distance by 2.5. The time taken is calculated by multiplying the total distance by 0.2. Finally the fuel consumption is calculated by multiplying the total distance by 2.

Furthermore, the cumulative total percentage of waste is calculated. Then the waste in kilograms (kg) is calculated as well for ease of visualisation.

To conclude, the Optimised Route algorithm provides an efficient alternative to the other algorithms already outlined. The Floyd Warshall algorithm follows a

dynamic programming approach and checks every possible path through every valid location to calculate the shortest path.

d. Algorithm 4: TSP Route

The TSP Route operates by using the Travelling Salesman Problem (TSP) with Greedy to find the shortest path from the “Waste Station” to every valid location. TSP works by finding the shortest path by visiting each valid location once only and returning to “Waste Station” which is the source vertex.

The “buildGraph” method initialises the distance matrix based on the distance matrix. The “minDistance” finds the vertex based on the minimum distance value based on the set of vertices that is not yet included in the shortest path tree. Following, the “printPath” method recursively prints the path from the “Waste Station” to every valid location. Next, the “printSolution” method calculates and prints the shortest path from the “Waste Station” to every valid location.

The time complexity of Floyd Warshall Algorithm is $O(n^2 * 2^n)$ and space complexity of $O(n * 2^n)$.

There are 3 conditions that are to be satisfied for locations to be visited in the Optimised Route Algorithm. 1) Waste at the location must be greater than or equal to 60% 2) The distance from “Waste Station” to every valid location must be within 12 km 3) Time taken must be within 8 hours, or else location will not be visited.

It also calculates the total cost, time taken, fuel consumption, and driver wage for all visited locations. The total cost is calculated by multiplying the total distance by 2.5. The time taken is calculated by multiplying the total distance by 0.2. Finally the fuel consumption is calculated by multiplying the total distance by 2.

Furthermore, the cumulative total percentage of waste is calculated. Then the waste in kilograms (kg) is calculated as well for ease of visualisation.

Overall, the TSP Route algorithm is an efficient solution to find the optimal route for visiting all valid locations by considering all possible paths that visit each valid location.

9. Restrictions on Algorithms

With all 4 algorithm implementations, there are specific conditions set. Briefly explained earlier on the 3 conditions that are applied on every algorithm. Here the restrictions will be clearly explained.

a. Restrictions on Non-Optimised Route

For Non-Optimised Route, the first condition is that the waste at location must be greater or equal to 40% to be visited and collected. Therefore, the randomly generated waste level at the location must be greater or equal to 40% for the algorithm to visit and collect the waste. The number used is 40 because it is a sufficient threshold of waste to be collected, proportional to the effectiveness of the Non-Optimised Route Algorithm. The second condition is that the distance from Waste Station to location must be less than or equal to 30 km, this is to ensure that only efficient routes are chosen and taken. Therefore the distance from the Waste Station to the valid location must have a distance less than or equal to 30 km to be visited and for the waste to be collected. Finally there is a time limit, where the time taken to visit all valid locations must be less than or equal to 12 hours. The time limit of 12 hours was chosen because Non-Optimised Route Algorithm will naturally take longer to visit locations when compared to other shortest path algorithms, and therefore to compensate for the performance, more time is given.

b. Restrictions on Greedy Route

Moving on, for the Greedy Route, the first condition is that the waste at location must be greater or equal to 30% to be visited and collected. Compared to the Non-Optimised Route Algorithm, the Greedy Route is given 10% less. This is purposely intended because the Greedy Route uses the Dijkstra Algorithm and

therefore is considered more optimised and dynamic compared to the Non-Optimised Route Algorithm which uses a fixed shortest path. Moving on, there is no distance restriction for the location to be visited because the algorithm is designed to prioritise visiting locations based on their proximity to the Waste Station. Furthermore, aiming to find the shortest path from the Waste Station to each valid location. Lastly the time limit applied on the Greedy Route Algorithm is 8 hours. This is 4 hours less when compared to the Non-Optimised Route Algorithm because the Greedy Route uses Dijkstra Algorithm which is considered to be more dynamic and adaptive.

c. Restrictions on Optimised Route

Next, for the Optimised Route, the first condition is that the waste at the location must be greater or equal to 50% to be visited and collected. Similarly to the Greedy Route Algorithm, the distance given is % more compared to the Non-Optimized Route Algorithm. This is intended because the Optimised Route uses the Floyd Warshall Algorithm, which is naturally more dynamic and adaptable with the given distance matrix compared to Non-Optimised Algorithm. The second condition is that the distance from the Waste Station to location must be less than or equal to 11 km. By limiting the distance to 11 km, the algorithm can efficiently route through the distance matrix in manageable time and also within efficient use of resources. Finally the time limit threshold set is similar to the Greedy Route where it is 8 hours. Although 4 hours less when compared to Non-Optimised Route, it is proportional to the performance of the Floyd Warshall algorithm present in the optimised route.

d. Restrictions on TSP Route

For the TSP Route, the first condition is that the waste at the location must be greater or equal to 60% to be visited and collected. In contrast to the Greedy and Optimised routes, this number is significantly higher. This is intended to allow for the TSP route to focus on visiting locations that have higher waste levels, allowing the algorithm to perform its visitations at all valid locations effectively. Moreover, it is also cost-effective and resource effective. This reflects the

strategy applied to the TSP route, optimising the waste collection in a different manner. The distance from the waste station to the location must be less than or equal to 12 km. Therefore similarly compared to the Greedy and Optimised routes, the number is chosen strategically to allow the TSP algorithm to operate effectively on the distance matrix and all valid locations. Lastly the time limit set is the same as the Greedy and Optimised Routes. The number is proportional to the performance of the Travelling Salesman Problem Algorithm.

10. Program Flow

1) User starts the program.
    a) The waste levels are displayed.
    b) The options are displayed.
2) User selects an algorithm or other function.
    a) The waste levels are displayed.
    b) The output of the algorithm is displayed.
3) User chooses to either perform another algorithm or function or exit.
4) If the user chooses an algorithm then it will go back to (2), if the user chooses another function then to (5) or (6).
5) If the user selects option 5 which is to display the map and distance matrix.
    a) 2 different distance matrices in 2 different forms appear for better visualisations.
6) If the user selects option 6 which is to regenerate the waste levels.
7) User exits.

11. Design structure of the program and algorithms

The program is split into multiple header files and classes. Each header file is included in the main.cpp file and the constructor is created in every file and called for every class in main.cpp. Below is the outline of every header file and their respective class:

a. GarbageLocations.h
b. distanceMap.h
c. NonOptimisedRoute.h

d. OptimisedRoute.h
e. GreedyRoute.h
f. TSPRoute.h
g. Locations.h
h. wasteLevel.h
i. main.cpp

Therefore there are a total of nine files. Where eight of the files are header files and the remaining is the main.cpp file.

12. Consistency in output

All four algorithms present the same formatted output. Each algorithm displays the shortest path from the WasteStation to every location. Also displaying the total distance, cost, time, fuel consumption, driver wage, and waste collected.

Refer to the images below on the consistency of the output for all four algorithms:

```
Non-Optimized Route - Fixed Shortest Path Algorithm

------------------------------------------------------

Details of Non-Optimized Route
1. Waste at location must be >=40% to be visited and collected
2. Distance from Waste Station to location must be <= 30km
3. Time taken must be within 12 hours, or else location will not be visited

Current Waste Levels:
Waste at Location 1: 49%
Waste at Location 2: 16%
Waste at Location 3: 51%
Waste at Location 4: 6%
Waste at Location 5: 76%
Waste at Location 6: 32%
Waste at Location 7: 83%

Waste in kg:
Waste in KG at Location 1: 245kg
Waste in KG at Location 2: 80kg
Waste in KG at Location 3: 255kg
Waste in KG at Location 4: 30kg
Waste in KG at Location 5: 380kg
Waste in KG at Location 6: 160kg
Waste in KG at Location 7: 415kg

Shortest Path from Waste Station to each location:
Waste Station to Location 1: 3 km       Path : Waste Station -> Location 1
Waste Station to Location 3: 9 km       Path : Waste Station -> Location 1 -> Location 3
Waste Station to Location 5: 29 km      Path : Waste Station -> Location 1 -> Location 3 -> Location 2 -> Location 4 -> Location 3 -> Location 6 -> Location 5
Waste Station to Location 7: Distance exceeds 30 km, Location not visited

Driver Wage for this trip: 164 MYR

Total Distance for Visited Locations: 41 km
Total Cost: 102.5 MYR
Total Time: 8.2 hours
Total Fuel Consumption: 4.1 Liters
Cumulative total percentage of waste collected from all valid locations: 176%
Total Waste Collected: 880kg

Do you want to run another algorithm or perform other functions? (y/n):
```

```
Greedy Route - Dijkstra Algorithm

-------------------------------------

Details of Greedy Route
1. Waste at location must be >=30% to be visited and collected
2. There is NO distance restriction for the location to be visited
3. Time taken must be within 8 hours, or else location will not be visited

Current Waste Levels:
Waste at Location 1: 49%
Waste at Location 2: 16%
Waste at Location 3: 51%
Waste at Location 4: 6%
Waste at Location 5: 76%
Waste at Location 6: 32%
Waste at Location 7: 83%

Waste in kg:
Waste in KG at Location 1: 245kg
Waste in KG at Location 2: 80kg
Waste in KG at Location 3: 255kg
Waste in KG at Location 4: 30kg
Waste in KG at Location 5: 380kg
Waste in KG at Location 6: 160kg
Waste in KG at Location 7: 415kg

Shortest Path from Waste Station to each location:
Waste Station to Location 1: 3 km       Path: Waste Station -> Location 1
Waste Station to Location 3: 9 km       Path: Waste Station -> Location 1 -> Location 3
Waste Station to Location 5: 14 km      Path: Waste Station -> Location 7 -> Location 6 -> Location 5
Waste Station to Location 6: 7 km       Path: Waste Station -> Location 7 -> Location 6
Waste Station to Location 7: 4 km       Path: Waste Station -> Location 7

Driver Wage for this trip: 148 MYR

Total Distance for Visited Locations: 37 km
Total Cost: 92.5 MYR
Total Time: 7.4 hours
Total Fuel Consumption: 25.9 Liters
Cumulative total percentage of waste collected from all valid locations: 291%
Total Waste Collected: 1455kg

Do you want to run another algorithm or perform other functions? (y/n):
```

```
Optimized Route - Floyd Warshall Algorithm

-------------------------------------------

Details of Optimized Route
1. Waste at location must be >=50% to be visited and collected
2. Distance from Waste STation to location must be <= 11km
3. Time taken must be within 8 hours, or else location will not be visited

Current Waste Levels:
Waste at Location 1: 49%
Waste at Location 2: 16%
Waste at Location 3: 51%
Waste at Location 4: 6%
Waste at Location 5: 76%
Waste at Location 6: 32%
Waste at Location 7: 83%

Waste in kg:
Waste in KG at Location 1: 245kg
Waste in KG at Location 2: 80kg
Waste in KG at Location 3: 255kg
Waste in KG at Location 4: 30kg
Waste in KG at Location 5: 380kg
Waste in KG at Location 6: 160kg
Waste in KG at Location 7: 415kg

Shortest Path from Waste Station to each location:
Waste Station to Location 3: 9 km       Path: Waste Station -> Location 1 -> Location 3
Waste Station to Location 5: Distance exceeds 11 km
Waste Station to Location 7: 4 km       Path: Waste Station -> Location 7

Driver Wage for this trip: 52 MYR

Total Distance for Visited Locations: 13 km
Total Cost: 32.5 MYR
Total Time: 2.6 hours
Total Fuel Consumption: 9.1 Liters
Cumulative total percentage of waste collected from all valid locations: 134%
Total Waste Collected: 670kg

Do you want to run another algorithm or perform other functions? (y/n):
```

```
TSP Route - Travelling Salesman Problem Algorithm

-----------------------------------------------------

Details of TSP Route
1. Waste at location must be >=60% to be visited and collected
2. Distance from Waste Station to location must be <= 12km
3. Time taken must be within 8 hours, or else location will not be visited

Current Waste Levels:
Waste at Location 1: 49%
Waste at Location 2: 16%
Waste at Location 3: 51%
Waste at Location 4: 6%
Waste at Location 5: 76%
Waste at Location 6: 32%
Waste at Location 7: 83%

Waste in kg:
Waste in KG at Location 1: 245kg
Waste in KG at Location 2: 80kg
Waste in KG at Location 3: 255kg
Waste in KG at Location 4: 30kg
Waste in KG at Location 5: 380kg
Waste in KG at Location 6: 160kg
Waste in KG at Location 7: 415kg

Shortest Path from Waste Station to each location:
Waste Station to Location 5: Distance exceeds 12 km
Waste Station to Location 7: 4 km       Path: Waste Station -> Location 7

Driver Wage for this trip: 16 MYR

Total Distance for Visited Locations: 4 km
Total Cost: 10 MYR
Total Time: 0.8 hours
Total Fuel Consumption: 2.8 Liters
Cumulative total percentage of waste collected from all valid locations: 83%
Total Waste Collected: 415kg

Do you want to run another algorithm or perform other functions? (y/n):
```

13. CPP features used

All CPP features outlined in the coursework sheet have been used in this coursework. Refer below for all the features used in this coursework.

a. C++ Libraries
b. Inheritance
c. Polymorphism - Method overloading and Method overriding
d. File I/O - Using fstream and output stream
e. Exception Handling - Conditional statements
f. Encapsulation
g. Abstraction
h. Data Structures - Map and Vectors
i. Control and Conditional Statements
j. Random Number Generator - Custom seed using current time

14. Extraordinary Features

This coursework has implemented a variety of different extraordinary features. Refer below for the extraordinary features implemented.

a. Consistent outputs
    i. All algorithms find the shortest path from the Waste Station to the target location
    ii. All algorithms is able to calculate the same information: Total distance, total cost, total time, total fuel consumption, cumulative total percentage of waste collected from all valid locations, and total waste collected
b. All algorithms provide the same result format
    i. The output in console for every algorithm is in the same format, aids to avoid any misunderstanding for the user
c. Implementing all 4 algorithms: Non-Optimised, Greedy, Optimised, and TSP
d. All algorithms calculate the shortest distance from the source to the target location
e. The algorithms can all show the path accurately

f. If the restrictions are met, then the algorithms displays the restrictions and adapts accordingly
    i. Restriction on distance, it will show “Distance exceed x km, location not visited”
    ii. Restriction on time, it will show “**Total time exceeds x hours, refer to the locations below that can be visited within x hours**”
g. If the total time restriction is met, the algorithms will display the locations that can be visited within the specified timeframe
h. Show the graph and distance for visualisation. The graph is in ASCII and the distance matrix is in a table.
i. Used a fixed path algorithm for Non-Optimised
j. Used Dijkstra Algorithm for Greedy
k. Used Floyd Warshall Algorithm for Optimised
l. Used Travelling Salesman Problem for TSP
m. All algorithms can calculate the same information
n. Regenerate waste levels, update the waste levels to the current waste levels, therefore providing a more updated outcome
o. Calculate the waste level in kilograms (kg), using the randomly generated waste level. Converts the percentage into kilograms format for ease of visualisation.
p. Display the total waste collected in kilograms (kg)
q. Calculating the total cost based on my assumptions
r. Calculating the total distance based on my customised graph and assumptions
s. Calculating the total time based on my assumptions
t. Calculating the total fuel consumption based on my assumptions
u. Calculating the cumulative total percentage of waste collected from all locations based on my assumptions
v. Calculating the total waste collected based on my assumptions
w. Display map and graph, show the distance matrix and the graph to aid with understanding of how the different algorithms operate
x. TSP is a modified TSP method whereby it stops at the target location instead of traversing through all locations
y. Restrictions on the conditions applied
    i. Distance exceeded
    ii. Time exceeded

z. Error handling and ease of use

aa. All outputs in the console for every algorithm chosen are also printed into an external text file, allowing the data to be used outside of the program. The graph and map (option 5) is also printed into an external text file

bb. Used “INF” values for the graph and distance matrix to indicate no direct path

cc. Error Handling to handle incorrect input

15. Explanation of code

*Note: An output stream is used extensively throughout the code to print all information not only to the console but to a separate output text file as well.

a. GarbageLocations.h

The GarbageLocations class contains 2 attributes: wasteLevels and distanceMatrix. The wasteLevels is a map that stores the waste level for each location, it is closely related to the wasteLevel.h class where a random waste level is generated for each location. The constructor initialises the distanceMatrix with pre-defined distances to every location. INF represents no direct path. The regenerateWasteLevel method updates the wasteLevels by generating new random values using a customised seeding method.

b. distanceMap.h

The distanceMap contains 2 public methods: printMap and printMapInformation. The printMap method prints a graphical representation of the map with labelled locations and distances. Whereas the printMapInformation method prints the distance matrix of the map, displaying the distances of each location between every other location, thereby giving the user a better representation of the map.

c. NonOptimisedRoute.h

The NonOptimisedRoute contains 2 public methods: noSolution (short for non-optimised solution) and calculateShortestPath. noSolution calculates the

waste collection route for each valid location based on a fixed predefined shortest path.

The shortest path is as outlined below:

```
shortestPathLocations.push_back("Waste Station");
shortestPathLocations.push_back("Location 1");
shortestPathLocations.push_back("Location 3");
shortestPathLocations.push_back("Location 2");
shortestPathLocations.push_back("Location 4");
shortestPathLocations.push_back("Location 3");
shortestPathLocations.push_back("Location 6");
shortestPathLocations.push_back("Location 5");
shortestPathLocations.push_back("Location 6");
shortestPathLocations.push_back("Location 7");
```

The calculateShortestPath method returns a predefined shortest path.

d. GreedyRoute.h

The GreedyRoute contains many public methods: createGraph, createPath, daSolution, minimumDistance, and dijkstraAlgorithm. The createGraph method creates a graph (2D array) using the predefined distance matrix from the GarbageLocations class. The createPath method recursively creates the path from the waste station to the target location using an array. The daSolution (short for dijkstra algorithm solution) is the implementation for the algorithm, and also calculates the related information such as: total distance, cost, time, fuel consumption, and waste collected.

e. OptimisedRoute.h

The OptimisedRoute contains a few public methods: createGraph, fwSolution, and floydWarshall methods. The createGraph method creates a graph (2D array) using the predefined distance matrix from the GarbageLocations class. The

fwSolution (short for floyd warshall solution) implements the floyd warshall algorithm and also calculates the related information such as: total distance, cost, time, fuel consumption, and waste collected. The floydWarshall method initialises and constructs the shortest distance and matrices for the algorithm.

f. TSPRoute.h

The TSPRoute contains public methods: createGraph, minimumDistance, tspPath, tspSolution, and travellingSalesmanProblem method. The createGraph method creates a graph (2D array) using the predefined distance matrix from the GarbageLocations class. The minimumDistance method finds the vertex with the minimum distance value from the Waste Station (Source). tspPath constructs the path for the TSP solution through recursion, calling the method inside the method. tspSolution calculates the related information such as: total distance, cost, time, fuel consumption, and waste collected. The travellingSalesmanProblem method initialises the shortest distance array and finds the shortest path.

g. Locations.h

The Locations.h file contains one public method. The public method is initializeLocations which pushes back each location into a currentLocations vector. It is called within the main.cpp file.

h. wasteLevel.h

Using method overloading, present are two methods with the same method name but different parameters. Both are used to display the wasteLevels. This class relates closely to GarbageLocations.

i. main.cpp

All header files are called in this main.cpp file. As shown below:

```
#include "GarbageLocations.h"
#include "NonOptimizedRoute.h"
#include "GreedyRoute.h"
#include "OptimizedRoute.h"
#include "TSPRoute.h"
#include "Locations.h"
#include "wasteLevel.h"
#include "distanceMap.h"
```

Then the instances of each class are called. As shown below:

```
NonOptimizedRoute nonOptimizedRoute;
GreedyRoute greedyRoute;
OptimizedRoute optimizedRoute;
TSPRoute tspRoute;
Locations location;
distanceMap distancemap;
wasteLevel wastelevel;
GarbageLocations garbageLocations;
```

16. Screenshots of code

a. Start/ Homepage

```
------------------------------------------------------------

Welcome to the Waste Management System

Enter your choice of which algorithm you want to run:
1. Non-Optimized Route - Fixed Shortest Path Algorithm
2. Greedy Route - Djikstra Algorithm
3. Optimized Route - Floyd Warshall Algorithm
4. TSP Route - Travelling Salesman Problem Algorithm
5. Print Map - Graph, Distance Matrix
6. Regenerate Waste Levels
7. Exit


Current Waste Levels:
Waste at Location 1: 49%
Waste at Location 2: 16%
Waste at Location 3: 51%
Waste at Location 4: 6%
Waste at Location 5: 76%
Waste at Location 6: 32%
Waste at Location 7: 83%

Waste in kg:
Waste in KG at Location 1: 245kg
Waste in KG at Location 2: 80kg
Waste in KG at Location 3: 255kg
Waste in KG at Location 4: 30kg
Waste in KG at Location 5: 380kg
Waste in KG at Location 6: 160kg
Waste in KG at Location 7: 415kg

Enter your choice (1 - 7): 1
```

b. Algorithm 1: Non-Optimized Route

```
Non-Optimized Route - Fixed Shortest Path Algorithm

------------------------------------------------------

Details of Non-Optimized Route
1. Waste at location must be >=40% to be visited and collected
2. Distance from Waste Station to location must be <= 30km
3. Time taken must be within 12 hours, or else location will not be visited

Current Waste Levels:
Waste at Location 1: 49%
Waste at Location 2: 16%
Waste at Location 3: 51%
Waste at Location 4: 6%
Waste at Location 5: 76%
Waste at Location 6: 32%
Waste at Location 7: 83%

Waste in kg:
Waste in KG at Location 1: 245kg
Waste in KG at Location 2: 80kg
Waste in KG at Location 3: 255kg
Waste in KG at Location 4: 30kg
Waste in KG at Location 5: 380kg
Waste in KG at Location 6: 160kg
Waste in KG at Location 7: 415kg

Shortest Path from Waste Station to each location:
Waste Station to Location 1: 3 km       Path : Waste Station -> Location 1
Waste Station to Location 3: 9 km       Path : Waste Station -> Location 1 -> Location 3
Waste Station to Location 5: 29 km      Path : Waste Station -> Location 1 -> Location 3 -> Location 2 -> Location 4 -> Location 3 -> Location 6 -> Location 5
Waste Station to Location 7: Distance exceeds 30 km, Location not visited

Driver Wage for this trip: 164 MYR

Total Distance for Visited Locations: 41 km
Total Cost: 102.5 MYR
Total Time: 8.2 hours
Total Fuel Consumption: 4.1 Liters
Cumulative total percentage of waste collected from all valid locations: 176%
Total Waste Collected: 880kg

Do you want to run another algorithm or perform other functions? (y/n):
```

c. Algorithm 2: Greedy Algorithm

```
Greedy Route - Dijkstra Algorithm

--------------------------------

Details of Greedy Route
1. Waste at location must be >=30% to be visited and collected
2. There is NO distance restriction for the location to be visited
3. Time taken must be within 8 hours, or else location will not be visited

Current Waste Levels:
Waste at Location 1: 49%
Waste at Location 2: 16%
Waste at Location 3: 51%
Waste at Location 4: 6%
Waste at Location 5: 76%
Waste at Location 6: 32%
Waste at Location 7: 83%

Waste in kg:
Waste in KG at Location 1: 245kg
Waste in KG at Location 2: 80kg
Waste in KG at Location 3: 255kg
Waste in KG at Location 4: 30kg
Waste in KG at Location 5: 380kg
Waste in KG at Location 6: 160kg
Waste in KG at Location 7: 415kg

Shortest Path from Waste Station to each location:
Waste Station to Location 1: 3 km       Path: Waste Station -> Location 1
Waste Station to Location 3: 9 km       Path: Waste Station -> Location 1 -> Location 3
Waste Station to Location 5: 14 km      Path: Waste Station -> Location 7 -> Location 6 -> Location 5
Waste Station to Location 6: 7 km       Path: Waste Station -> Location 7 -> Location 6
Waste Station to Location 7: 4 km       Path: Waste Station -> Location 7

Driver Wage for this trip: 148 MYR

Total Distance for Visited Locations: 37 km
Total Cost: 92.5 MYR
Total Time: 7.4 hours
Total Fuel Consumption: 25.9 Liters
Cumulative total percentage of waste collected from all valid locations: 291%
Total Waste Collected: 1455kg

Do you want to run another algorithm or perform other functions? (y/n):
```

d. Algorithm 3: Optimised Route

```
Optimized Route - Floyd Warshall Algorithm

-------------------------------------------

Details of Optimized Route
1. Waste at location must be >=50% to be visited and collected
2. Distance from Waste STation to location must be <= 11km
3. Time taken must be within 8 hours, or else location will not be visited

Current Waste Levels:
Waste at Location 1: 49%
Waste at Location 2: 16%
Waste at Location 3: 51%
Waste at Location 4: 6%
Waste at Location 5: 76%
Waste at Location 6: 32%
Waste at Location 7: 83%

Waste in kg:
Waste in KG at Location 1: 245kg
Waste in KG at Location 2: 80kg
Waste in KG at Location 3: 255kg
Waste in KG at Location 4: 30kg
Waste in KG at Location 5: 380kg
Waste in KG at Location 6: 160kg
Waste in KG at Location 7: 415kg

Shortest Path from Waste Station to each location:
Waste Station to Location 3: 9 km       Path: Waste Station -> Location 1 -> Location 3
Waste Station to Location 5: Distance exceeds 11 km
Waste Station to Location 7: 4 km       Path: Waste Station -> Location 7

Driver Wage for this trip: 52 MYR

Total Distance for Visited Locations: 13 km
Total Cost: 32.5 MYR
Total Time: 2.6 hours
Total Fuel Consumption: 9.1 Liters
Cumulative total percentage of waste collected from all valid locations: 134%
Total Waste Collected: 670kg

Do you want to run another algorithm or perform other functions? (y/n):
```

e. Algorithm 4: TSP Route

```
TSP Route - Travelling Salesman Problem Algorithm

_________________________________________________

Details of TSP Route
1. Waste at location must be >=60% to be visited and collected
2. Distance from Waste Station to location must be <= 12km
3. Time taken must be within 8 hours, or else location will not be visited

Current Waste Levels:
Waste at Location 1: 49%
Waste at Location 2: 16%
Waste at Location 3: 51%
Waste at Location 4: 6%
Waste at Location 5: 76%
Waste at Location 6: 32%
Waste at Location 7: 83%

Waste in kg:
Waste in KG at Location 1: 245kg
Waste in KG at Location 2: 80kg
Waste in KG at Location 3: 255kg
Waste in KG at Location 4: 30kg
Waste in KG at Location 5: 380kg
Waste in KG at Location 6: 160kg
Waste in KG at Location 7: 415kg

Shortest Path from Waste Station to each location:
Waste Station to Location 5: Distance exceeds 12 km
Waste Station to Location 7: 4 km       Path: Waste Station -> Location 7

Driver Wage for this trip: 16 MYR

Total Distance for Visited Locations: 4 km
Total Cost: 10 MYR
Total Time: 0.8 hours
Total Fuel Consumption: 2.8 Liters
Cumulative total percentage of waste collected from all valid locations: 83%
Total Waste Collected: 415kg

Do you want to run another algorithm or perform other functions? (y/n):
```

f. Print Map - Graph, Distance Matrix

```
Graph:
                        3
   +---> WasteStation <----> Location1
   |                          ^
   |                          |
   |                          | 6
   |                          |
   |                          |
   |                    5     v
  4|        Location2 <----> Location3
   |          ^          ^   ^
   |          |          |   |
   |        4 |    +-------+   | 2
   |          |    |    2      |
   |          |    |           |
   |          v    v           v
   |        Location4        Location6 <----> Location5
   |                          ^         7
   v                          |
Location7 <-----------------------+
                    3
Distance Matrix:

                Waste Station     Location 1     Location 2     Location 3     Location 4     Location 5     Location 6     Location 7
  Waste Station             0              3            INF            INF            INF            INF            INF              4
     Location 1             3              0            INF              6            INF            INF            INF            INF
     Location 2           INF            INF              0              5              4            INF            INF            INF
     Location 3           INF              6              5              0              2            INF              2            INF
     Location 4           INF            INF              4              2              0            INF            INF            INF
     Location 5           INF            INF            INF            INF            INF              0              7            INF
     Location 6           INF            INF            INF              2            INF              7              0              3
     Location 7             4            INF            INF            INF            INF            INF              3              0

Do you want to run another algorithm or perform other functions? (y/n):
```

g. Regenerate Waste Levels

```
Regenerating waste levels


------------------------------------------------------------


Welcome to the Waste Management System

Enter your choice of which algorithm you want to run:
1. Non-Optimized Route - Fixed Shortest Path Algorithm
2. Greedy Route - Djikstra Algorithm
3. Optimized Route - Floyd Warshall Algorithm
4. TSP Route - Travelling Salesman Problem Algorithm
5. Print Map - Graph, Distance Matrix
6. Regenerate Waste Levels
7. Exit


Current Waste Levels:
Waste at Location 1: 72%
Waste at Location 2: 56%
Waste at Location 3: 97%
Waste at Location 4: 51%
Waste at Location 5: 56%
Waste at Location 6: 79%
Waste at Location 7: 17%

Waste in kg:
Waste in KG at Location 1: 360kg
Waste in KG at Location 2: 280kg
Waste in KG at Location 3: 485kg
Waste in KG at Location 4: 255kg
Waste in KG at Location 5: 280kg
Waste in KG at Location 6: 395kg
Waste in KG at Location 7: 85kg

Enter your choice (1 - 7):
```

h. Exit Program

```
Welcome to the Waste Management System

Enter your choice of which algorithm you want to run:
1. Non-Optimized Route - Fixed Shortest Path Algorithm
2. Greedy Route - Djikstra Algorithm
3. Optimized Route - Floyd Warshall Algorithm
4. TSP Route - Travelling Salesman Problem Algorithm
5. Print Map - Graph, Distance Matrix
6. Regenerate Waste Levels
7. Exit


Current Waste Levels:
Waste at Location 1: 72%
Waste at Location 2: 56%
Waste at Location 3: 97%
Waste at Location 4: 51%
Waste at Location 5: 56%
Waste at Location 6: 79%
Waste at Location 7: 17%

Waste in kg:
Waste in KG at Location 1: 360kg
Waste in KG at Location 2: 280kg
Waste in KG at Location 3: 485kg
Waste in KG at Location 4: 255kg
Waste in KG at Location 5: 280kg
Waste in KG at Location 6: 395kg
Waste in KG at Location 7: 85kg

Enter your choice (1 - 7): 7

Exiting the program
```

i. Error Handling - Entering a number outside of (1-7)

```
Enter your choice (1 - 7): 9
Invalid choice. Please enter a valid choice.
```

j. Choosing to run another function

```
Do you want to run another algorithm or perform other functions? (y/n): y

------------------------------------------------------------


Welcome to the Waste Management System

Enter your choice of which algorithm you want to run:
1. Non-Optimized Route - Fixed Shortest Path Algorithm
2. Greedy Route - Djikstra Algorithm
3. Optimized Route - Floyd Warshall Algorithm
4. TSP Route - Travelling Salesman Problem Algorithm
5. Print Map - Graph, Distance Matrix
6. Regenerate Waste Levels
7. Exit
```

k. Choosing to NOT run another function

```
Do you want to run another algorithm or perform other functions? (y/n): n

Exiting the program
```

l. Error Handling - Entering a letter that is not “y” or “n”

```
Do you want to run another algorithm or perform other functions? (y/n): o
Invalid choice. Enter again.
Do you want to run another algorithm or perform other functions? (y/n):
```

17. Conclusion

To conclude, the Automated Garbage Management System has been successfully implemented with the use of four different algorithms. The four different algorithms for waste collection route optimisation: Non-Optimised Route, Greedy Route, Optimised Route, and TSP Route. Each algorithm is different, the approach to tackling the map and finding shortest path to the valid locations are different for each algorithm.

Starting with Non-Optimized Route Algorithm, it is the simplest algorithm approach towards tackling the map and finding the shortest path from Waste Station to every valid location. It utilises a simple pre-defined shortest path route to calculate the path. It is not dynamic and does not consider all possible routes.

The next approach used was the Greedy Route Algorithm, it specifically uses Dijkstra Algorithm. The Dijkstra Algorithm finds the shortest path between the Waste Station and the valid locations that satisfies the 3 conditions.

The third approach used is the Optimised Route that explicitly uses the Floyd Warshall Algorithm. Compared to the Non-Optimised Route Algorithm and Greedy Route Algorithm, it is more dynamic and adaptable with the map. Therefore, providing more optimised results with the restrictions imposed.

The final approach used is the TSP Route Algorithm. It uses the Travelling Salesman Problem with slight modifications. Instead of routing back to the source after reaching the target location, it stops at the target location and returns the shortest path. TSP Roue Algorithm is known to be optimised in a different way where it selects the more efficient paths.

To summarise, every algorithm implemented has their own respective strengths and weaknesses. Throughout this coursework, implementing the algorithms has allowed for new insights into the performance of every algorithm and how they compare against each other.

18. Feedback Questions

a. Notable Difficulties

The first difficulty I faced was understanding how to accurately define and efficiently store the distance matrix for the map. I had trouble initialising the “INF” values to represent infinity for paths that don’t exist. However, through research and learning how to use INF correctly, I was able to include INF into my distance matrix effectively and calculate distances normally.

The second difficulty I faced was learning how to apply the algorithms correctly with the predefined distance matrix. I faced many instances where I would get overflowed values, due to the presence of INF. Through trial and error and using conditional statements correctly within each algorithm, I was able to overcome the issue.

Third, was learning to effectively use header files and classes. Improving encapsulation and calling every header file and constructor into the main.cpp file. Allowing for cleaner and more efficient code.

b. Things Learned

The first thing I learned was the importance of selecting the correct algorithms. Every algorithm is vastly different in their own ways. For example, when selecting an algorithm for Optimised Route, there were many different choices but through sufficient literature review and decision making, it was narrowed down to the Floyd Warshall Algorithm.

Furthermore, I was able to really learn how to implement all 4 algorithms effectively and in a custom manner to fit with my own predefined distance matrix.

Overall, I was able to gain a deeper understanding of each algorithm implemented in the Automated Garbage Management System in this coursework. Thank you Dr. Doreen for all the assistance throughout in providing insights into specific algorithms and how it should be implemented.

19. Appendix

GarbageLocations.h:

```
#include <iostream> //Input and Output operations
#include <string> //String operations
#include <vector> //Dynamic array structure operations
#include <stdio.h> //Standard Input/Output operations
#include <stdlib.h> //Standard Library operations
```

```
#include <ctime> //To get date and time
#include <cmath> //Math related functions
#include <map> //Store map opearations for key value pairs
#include <limits> //Uses constants that represent the limits of data types
#include <algorithm> //Provides a collection of functions for working with containers
#include <tuple> //Allows the use of fixed-size collection of elements
#include <climits> //So INF and INT MAX can be used

using namespace std; //Using standard namespace

#define INF INT_MAX //Set INF to a max int value

#ifndef GARBAGE_LOCATIONS_H //Header file guard
#define GARBAGE_LOCATIONS_H //Header file guard

class GarbageLocations {  //GarbageLocations class

public: //Public specifier
  map<string, int> wasteLevels; //Defining a map key value pair for the wastelevels
  map<string, map<string, int> > distanceMatrix; //Defining a map key value pair for the
distance matrix

  GarbageLocations() { //Constructor
    vector<string> locations; //Defining the vector locations
    locations.push_back("Waste Station"); //Push the locations into the vector
    locations.push_back("Location 1"); //Push the locations into the vector
    locations.push_back("Location 2"); //Push the locations into the vector
    locations.push_back("Location 3"); //Push the locations into the vector
    locations.push_back("Location 4"); //Push the locations into the vector
    locations.push_back("Location 5"); //Push the locations into the vector
    locations.push_back("Location 6"); //Push the locations into the vector
    locations.push_back("Location 7"); //Push the locations into the vector

    distanceMatrix["Waste Station"]["Waste Station"] = 0; //Location pointing to itself is 0
    distanceMatrix["Waste Station"]["Location 1"] = 3; //The distance between the waste
station and location 1 is 3
    distanceMatrix["Waste Station"]["Location 2"] = INF; //There is no direct path
between the waste station and location 2
    distanceMatrix["Waste Station"]["Location 3"] = INF; //There is no direct path
between the waste station and location 3
    distanceMatrix["Waste Station"]["Location 4"] = INF; //There is no direct path
between the waste station and location 4
    distanceMatrix["Waste Station"]["Location 5"] = INF; //There is no direct path
between the waste station and location 5
```

```
    distanceMatrix["Waste Station"]["Location 6"] = INF; //There is no direct path between the waste station and location 6
    distanceMatrix["Waste Station"]["Location 7"] = 4; //The distance between the waste station and location 7 is 4

    distanceMatrix["Location 1"]["Waste Station"] = 3; //The distance between location 1 and the waste station is 3
    distanceMatrix["Location 1"]["Location 1"] = 0; //Location pointing to itself is 0
    distanceMatrix["Location 1"]["Location 2"] = INF; //There is no direct path between location 1 and location 2
    distanceMatrix["Location 1"]["Location 3"] = 6; //The distance between location 1 and location 3 is 6
    distanceMatrix["Location 1"]["Location 4"] = INF; //There is no direct path between location 1 and location 4
    distanceMatrix["Location 1"]["Location 5"] = INF, //There is no direct path between location 1 and location 5
    distanceMatrix["Location 1"]["Location 6"] = INF; //There is no direct path between location 1 and location 6
    distanceMatrix["Location 1"]["Location 7"] = INF; //There is no direct path between location 1 and location 7

    distanceMatrix["Location 2"]["Waste Station"] = INF; //There is no direct path between location 2 and the waste station
    distanceMatrix["Location 2"]["Location 1"] = INF; //There is no direct path between location 2 and location 1
    distanceMatrix["Location 2"]["Location 2"] = 0; //Location pointing to itself is 0
    distanceMatrix["Location 2"]["Location 3"] = 5; //The distance between location 2 and location 3 is 5
    distanceMatrix["Location 2"]["Location 4"] = 4; //The distance between location 2 and location 4 is 4
    distanceMatrix["Location 2"]["Location 5"] = INF; //There is no direct path between location 2 and location 5
    distanceMatrix["Location 2"]["Location 6"] = INF; //There is no direct path between location 2 and location 6
    distanceMatrix["Location 2"]["Location 7"] = INF; //There is no direct path between location 2 and location 7

    distanceMatrix["Location 3"]["Waste Station"] = INF; //There is no direct path between location 3 and the waste station
    distanceMatrix["Location 3"]["Location 1"] = 6; //The distance between location 3 and location 1 is 6
    distanceMatrix["Location 3"]["Location 2"] = 5; //The distance between location 3 and location 2 is 5
    distanceMatrix["Location 3"]["Location 3"] = 0; //Location pointing to itself is 0
```

```
    distanceMatrix["Location 3"]["Location 4"] = 2; //The distance between location 3 and location 4 is 2
    distanceMatrix["Location 3"]["Location 5"] = INF; //There is no direct path between location 3 and location 5
    distanceMatrix["Location 3"]["Location 6"] = 2; //The distance between location 3 and location 6 is 2
    distanceMatrix["Location 3"]["Location 7"] = INF; //There is no direct path between location 3 and location 7

    distanceMatrix["Location 4"]["Waste Station"] = INF; //There is no direct path between location 4 and the waste station
    distanceMatrix["Location 4"]["Location 1"] = INF; //There is no direct path between location 4 and location 1
    distanceMatrix["Location 4"]["Location 2"] = 4; //The distance between location 4 and location 2 is 4
    distanceMatrix["Location 4"]["Location 3"] = 2; //The distance between location 4 and location 3 is 2
    distanceMatrix["Location 4"]["Location 4"] = 0; //Location pointing to itself is 0
    distanceMatrix["Location 4"]["Location 5"] = INF; //There is no direct path between location 4 and location 5
    distanceMatrix["Location 4"]["Location 6"] = INF; //There is no direct path between location 4 and location 6
    distanceMatrix["Location 4"]["Location 7"] = INF; //There is no direct path between location 4 and location 7

    distanceMatrix["Location 5"]["Waste Station"] = INF; //There is no direct path between location 5 and the waste station
    distanceMatrix["Location 5"]["Location 1"] = INF; //There is no direct path between location 5 and location 1
    distanceMatrix["Location 5"]["Location 2"] = INF; //There is no direct path between location 5 and location 2
    distanceMatrix["Location 5"]["Location 3"] = INF; //There is no direct path between location 5 and location 3
    distanceMatrix["Location 5"]["Location 4"] = INF; //There is no direct path between location 5 and location 4
    distanceMatrix["Location 5"]["Location 5"] = 0; //Location pointing to itself is 0
    distanceMatrix["Location 5"]["Location 6"] = 7; //The distance between location 5 and location 6 is 7
    distanceMatrix["Location 5"]["Location 7"] = INF; //There is no direct path between location 5 and location 7

    distanceMatrix["Location 6"]["Waste Station"] = INF; //There is no direct path between location 6 and the waste station
```

```
    distanceMatrix["Location 6"]["Location 1"] = INF; //There is no direct path between
location 6 and location 1
    distanceMatrix["Location 6"]["Location 2"] = INF; //There is no direct path between
location 6 and location 2
    distanceMatrix["Location 6"]["Location 3"] = 2; //The distance between location 6 and
location 3 is 2
    distanceMatrix["Location 6"]["Location 4"] = INF; //There is no direct path between
location 6 and location 4
    distanceMatrix["Location 6"]["Location 5"] = 7; //The distance between location 6 and
location 5 is 7
    distanceMatrix["Location 6"]["Location 6"] = 0; //Location pointing to itself is 0
    distanceMatrix["Location 6"]["Location 7"] = 3; //The distance between location 6 and
location 7 is 3

    distanceMatrix["Location 7"]["Waste Station"] = 4; //The distance between location 7
and the waste station is 4
    distanceMatrix["Location 7"]["Location 1"] = INF; //There is no direct path between
location 7 and location 1
    distanceMatrix["Location 7"]["Location 2"] = INF; //There is no direct path between
location 7 and location 2
    distanceMatrix["Location 7"]["Location 3"] = INF; //There is no direct path between
location 7 and location 3
    distanceMatrix["Location 7"]["Location 4"] = INF; //There is no direct path between
location 7 and location 4
    distanceMatrix["Location 7"]["Location 5"] = INF; //There is no direct path between
location 7 and location 5
    distanceMatrix["Location 7"]["Location 6"] = 3; //The distance between location 7 and
location 6 is 3
    distanceMatrix["Location 7"]["Location 7"] = 0; //Location pointing to itself is 0

    srand(time(0)); //seeding the random number generator

    wasteLevels[locations[0]] = 0; //Waste at source is 0
    for (int index = 1; index < locations.size(); index++) {
      int wasteLevel = rand() % 101; //Assigining a random value to every location
except the source
      wasteLevels[locations[index]] = wasteLevel;
    }
  }

  void regenerateWasteLevels() { //Regenerating waste levels
    srand(time(0));
    wasteLevels.clear(); //Clearing the waste levels
    vector<string> locations;
```

```
        locations.push_back("Waste Station");
        locations.push_back("Location 1");
        locations.push_back("Location 2");
        locations.push_back("Location 3");
        locations.push_back("Location 4");
        locations.push_back("Location 5");
        locations.push_back("Location 6");
        locations.push_back("Location 7");

        wasteLevels["Waste Station"] = 0;
        for (int index = 1; index < locations.size(); index++) {
            int wasteLevel = rand() % 101;
            wasteLevels[locations[index]] = wasteLevel;
        }
    }
};

#endif
```

distanceMap.h:

```
#include <iostream> //Input and Output operations
#include <string> //String operations
#include <vector> //Dynamic array structure operations
#include <stdio.h> //Standard Input/Output operations
#include <stdlib.h> //Standard Library operations
#include <ctime> //To get date and time
#include <cmath> //Math related functions
#include <map> //Store map opearations for key value pairs
#include <limits> //Uses constants that represent the limits of data types
#include <algorithm> //Provides a collection of functions for working with containers
#include <tuple> //Allows the use of fixed-size collection of elements
#include <climits> //So INF and INT MAX can be used

using namespace std; //Use standard namespace

#ifndef MAP_H //Header file guard
#define MAP_H //Header file guard

class distanceMap { //Class for the distance map
public: //Public specifier
    void printMap(ofstream &outFile) { //Print map to console and to output file

        outFile << "                    3                    \n";
```

```
    outFile << "   +---> WasteStation <----> Location1                \n";
    outFile << "   |                    ^                    \n";
    outFile << "   |                    |                  \n";
    outFile << "   |                    | 6                 \n";
    outFile << "   |                    |                  \n";
    outFile << "   |                    |                  \n";
    outFile << "   |                5       v                  \n";
    outFile << "  4|       Location2 <----> Location3               \n";
    outFile << "   |          ^          ^   ^                  \n";
    outFile << "   |          |          |   |                  \n";
    outFile << "   |         4 |    +-------+   | 2                \n";
    outFile << "   |          |    |    2     |                 \n";
    outFile << "   |          |    |          |                  \n";
    outFile << "   |          v    v          v                    \n";
    outFile << "   |       Location4        Location6 <----> Location5\n";
    outFile << "   |                    ^         7           \n";
    outFile << "   v                    |                  \n";
    outFile << "Location7 <----------------------+                  \n";
    outFile << "                3                            \n";

    cout << "                    3                          \n";
    cout << "   +---> WasteStation <----> Location1                \n";
    cout << "   |                    ^                    \n";
    cout << "   |                    |                  \n";
    cout << "   |                    | 6                 \n";
    cout << "   |                    |                  \n";
    cout << "   |                    |                  \n";
    cout << "   |                5       v                  \n";
    cout << "  4|       Location2 <----> Location3               \n";
    cout << "   |          ^          ^   ^                  \n";
    cout << "   |          |          |   |                  \n";
    cout << "   |         4 |    +-------+   | 2                \n";
    cout << "   |          |    |    2     |                 \n";
    cout << "   |          |    |          |                  \n";
    cout << "   |          v    v          v                    \n";
    cout << "   |       Location4        Location6 <----> Location5\n";
    cout << "   |                    ^         7           \n";
    cout << "   v                    |                  \n";
    cout << "Location7 <----------------------+                  \n";
    cout << "                3                            \n";
  }

  void printMapInformation(ofstream &outFile) { //Print map information to console and to
output file
```

```
    vector<string> locations; //Vector to store the locations
    locations.push_back("Waste Station");
    locations.push_back("Location 1");
    locations.push_back("Location 2");
    locations.push_back("Location 3");
    locations.push_back("Location 4");
    locations.push_back("Location 5");
    locations.push_back("Location 6");
    locations.push_back("Location 7");

    vector<vector<int> > distanceMatrix(8, vector<int>(8)); //Vector to store the distance
matrix

    distanceMatrix[0][0] = 0;
    distanceMatrix[0][1] = 3;
    distanceMatrix[0][2] = INF;
    distanceMatrix[0][3] = INF;
    distanceMatrix[0][4] = INF;
    distanceMatrix[0][5] = INF;
    distanceMatrix[0][6] = INF;
    distanceMatrix[0][7] = 4;

    distanceMatrix[1][0] = 3;
    distanceMatrix[1][1] = 0;
    distanceMatrix[1][2] = INF;
    distanceMatrix[1][3] = 6;
    distanceMatrix[1][4] = INF;
    distanceMatrix[1][5] = INF;
    distanceMatrix[1][6] = INF;
    distanceMatrix[1][7] = INF;

    distanceMatrix[2][0] = INF;
    distanceMatrix[2][1] = INF;
    distanceMatrix[2][2] = 0;
    distanceMatrix[2][3] = 5;
    distanceMatrix[2][4] = 4;
    distanceMatrix[2][5] = INF;
    distanceMatrix[2][6] = INF;
    distanceMatrix[2][7] = INF;

    distanceMatrix[3][0] = INF;
    distanceMatrix[3][1] = 6;
    distanceMatrix[3][2] = 5;
```

```
distanceMatrix[3][3] = 0;
distanceMatrix[3][4] = 2;
distanceMatrix[3][5] = INF;
distanceMatrix[3][6] = 2;
distanceMatrix[3][7] = INF;

distanceMatrix[4][0] = INF;
distanceMatrix[4][1] = INF;
distanceMatrix[4][2] = 4;
distanceMatrix[4][3] = 2;
distanceMatrix[4][4] = 0;
distanceMatrix[4][5] = INF;
distanceMatrix[4][6] = INF;
distanceMatrix[4][7] = INF;

distanceMatrix[5][0] = INF;
distanceMatrix[5][1] = INF;
distanceMatrix[5][2] = INF;
distanceMatrix[5][3] = INF;
distanceMatrix[5][4] = INF;
distanceMatrix[5][5] = 0;
distanceMatrix[5][6] = 7;
distanceMatrix[5][7] = INF;

distanceMatrix[6][0] = INF;
distanceMatrix[6][1] = INF;
distanceMatrix[6][2] = INF;
distanceMatrix[6][3] = 2;
distanceMatrix[6][4] = INF;
distanceMatrix[6][5] = 7;
distanceMatrix[6][6] = 0;
distanceMatrix[6][7] = 3;

distanceMatrix[7][0] = 4;
distanceMatrix[7][1] = INF;
distanceMatrix[7][2] = INF;
distanceMatrix[7][3] = INF;
distanceMatrix[7][4] = INF;
distanceMatrix[7][5] = INF;
distanceMatrix[7][6] = 3;
distanceMatrix[7][7] = 0;

outFile << "Distance Matrix: \n\n";
cout << "Distance Matrix: \n\n";
```

```
        outFile << setw(15) << " "; //Print the distance matrix and include formatting using
setw
        cout << setw(15) << " "; //Print the distance matrix and include formatting using setw
        for (vector<string>::const_iterator iterator = locations.begin(); iterator !=
locations.end(); iterator++) { //Print the distance matrix and include formatting using setw
            outFile << setw(15) << *iterator;
            cout << setw(15) << *iterator;
        }
        outFile << endl;
        cout << endl;

        for (int index=0; index < locations.size(); index++) { //Print the distance matrix and
include formatting using setw
            outFile << setw(15) << locations[index];
            cout << setw(15) << locations[index];
            for (int index2 = 0; index2 < locations.size(); index2++) {
                if (distanceMatrix[index][index2] != INF) { //If there is a direct path because it is
!= INF then print the distance and display
                    outFile << setw(15) << distanceMatrix[index][index2];
                    cout << setw(15) << distanceMatrix[index][index2];
                } else {
                    outFile << setw(15) << "INF"; //Print INF values to indicate no direct path
                    cout << setw(15) << "INF"; //Print INF values to indicate no direct path
                }
            }
            outFile << endl;
            cout << endl;
        }
    }
};

#endif
```

NonOptimizedRoute.h:

```
#include <iostream> //Input and Output operations
#include <string> //String operations
#include <vector> //Dynamic array structure operations
#include <stdio.h> //Standard Input/Output operations
#include <stdlib.h> //Standard Library operations
```

```
#include <ctime> //To get date and time
#include <cmath> //Math related functions
#include <map> //Store map opearations for key value pairs
#include <limits> //Uses constants that represent the limits of data types
#include <algorithm> //Provides a collection of functions for working with containers
#include <tuple> //Allows the use of fixed-size collection of elements
#include <climits> //So INF and INT MAX can be used
#include "GarbageLocations.h" //Include the GarbageLocations header file
#include "wasteLevel.h" //Include the wasteLevel header file

#ifndef NON_OPTIMIZED_ROUTE_H //Header file guard
#define NON_OPTIMIZED_ROUTE_H //Header file guard

using namespace std; //Use standard namespace

class NonOptimizedRoute { //Class for the NonOptimizedRoute

public: //public specifier
  void noSolution(GarbageLocations& garbageLocations, const vector<string>&
currentLocations, const vector<string>& shortestPathLocations, ofstream& outFileNOR)
{ //Non-Optimized solution for the route
    cout << "Shortest Path from Waste Station to each location:" << endl;
    outFileNOR << "Shortest Path from Waste Station to each location:" << endl;

    int totalDistance = 0; //Initiailizing the totalDistance variable to 0
    int totalWasteCollected = 0; //Initiailizing the totalWasteCollected variable to 0
    float totalCost, totalTime, totalFuelConsumption, driverWage = 0; //Initiailizing the
float totalCost, totalTime, totalFuelConsumption and driverWage variables to 0
    int wasteInKG = 0; //Initiailizing the wasteInKG variable to 0

    vector<string> locationsWithin12Hours; //Creating a vector of strings called
locationsWithin12Hours

    for (int index = 1; index < currentLocations.size(); index++) { //For loop to iterate
through the currentLocations vector, this is all the locations in the distance matrix
      if (garbageLocations.wasteLevels[currentLocations[index]] >= 40) { //Check if the
waste level is greater than or equal to 40
        int pathDistanceFromWasteStation = 0; //Initiailizing the
pathDistanceFromWasteStation variable to 0
        cout << "Waste Station to " << currentLocations[index] << ": ";
        outFileNOR << "Waste Station to " << currentLocations[index] << ": ";
        for (int index2 = 1; index2 < shortestPathLocations.size(); index2++) { //Loop
the shortestPathLocations vector
          if (shortestPathLocations[index2] == currentLocations[index]) {
```

```
                for (int index3 = 1; index3 <= index2; index3++) {
                    int findIndex = index3 - 1;
                    pathDistanceFromWasteStation +=
garbageLocations.distanceMatrix[shortestPathLocations[findIndex]][shortestPathLocatio
ns[index3]];
                }
                if (pathDistanceFromWasteStation <= 30) { //Check if the distance from
the waste station to target location is less than or equal to 30
                    totalDistance += pathDistanceFromWasteStation;
                    totalWasteCollected +=
garbageLocations.wasteLevels[currentLocations[index]];
                    cout << pathDistanceFromWasteStation << " km \tPath : ";
                    outFileNOR << pathDistanceFromWasteStation << " km \tPath: ";
                    for (int index4 = 0; index4 <= index2; index4++) {
                        cout << shortestPathLocations[index4];
                        outFileNOR << shortestPathLocations[index4];
                        if (index4 != index2) {
                            cout << " -> ";
                            outFileNOR << " -> ";
                        }
                    }
                } else { //If not then display the output below
                    cout << "Distance exceeds 30 km, Location not visited";
                    outFileNOR << "Distance exceeds 30 km, Location not visited";
                }
                break;
            }
        }

        wasteInKG = (totalWasteCollected * 500) / 100; //Convert the waste collected
from percentage to KG

        cout << endl;
        outFileNOR << endl;
        totalCost = totalDistance * 2.5;
        totalTime = totalDistance * 0.2;
        totalFuelConsumption = 2 * totalDistance;

        if (totalTime <= 12) {
            locationsWithin12Hours.push_back(currentLocations[index]);
        } else {
            cout << "\n**Total time exceeds 12 hours, refer to the locations below that
can be visited within 12 hours**" << endl;
```

```
                outFileNOR << "\n**Total time exceeds 12 hours, refer to the locations below
that can be visited within 12 hours**" << endl;
                break;
            }
        }
    }

    if (totalTime > 12) { //There is a time limit of 12 hours, if exceeded then display the
output below
        cout << "\nLocations that can be visited within 12 hours:" << endl;
        outFileNOR << "\nLocations that can be visited within 12 hours:" << endl;
        for (int i = 0; i < locationsWithin12Hours.size(); i++) {
            cout << locationsWithin12Hours[i] << endl;
            outFileNOR << locationsWithin12Hours[i] << endl;
        }
    }

    if (totalTime > 12) {
        driverWage = 20 * 12;  // Driver wage for visited locations
        cout << "\nDriver Wage for this trip: " << driverWage << " MYR" << endl;
        outFileNOR << "\nDriver Wage for this trip: " << driverWage << " MYR" << endl;
    } else {
        driverWage = 20 * totalTime;  // Driver wage for visited locations
        cout << "\nDriver Wage for this trip: " << driverWage << " MYR" << endl;
        outFileNOR << "\nDriver Wage for this trip: " << driverWage << " MYR" << endl;
    }

    //Display the total distance, cost, time, fuel consumption and waste collected - all
related information
    cout << "\nTotal Distance for Visited Locations: " << totalDistance << " km" << endl;
    outFileNOR << "\nTotal Distance for Visited Locations: " << totalDistance << " km"
<< endl;
    cout << "Total Cost: " << totalCost << " MYR" << endl;
    outFileNOR << "Total Cost: " << totalCost << " MYR" << endl;
    cout << "Total Time: " << totalTime << " hours" << endl;
    outFileNOR << "Total Time: " << totalTime << " hours" << endl;
    cout << "Total Fuel Consumption: " << totalFuelConsumption << " Liters" << endl;
    outFileNOR << "Total Fuel Consumption: " << totalFuelConsumption << " Liters" <<
endl;
    cout << "Cumulative total percentage of waste collected from all valid locations: " <<
totalWasteCollected << "%" << endl;
    outFileNOR << "Cumulative total percentage of waste collected from all valid
locations: " << totalWasteCollected << "%" << endl;
    cout << "Total Waste Collected: " << wasteInKG << "kg" << endl;
```

```
        outFileNOR << "Total Waste Collected: " << wasteInKG << "kg" << endl;
    }

    vector<string> calculateShortestPath(const vector<string>& currentLocations) {
//Method that pre-defines the shortest path using the graph and distance matrix
        vector<string> shortestPathLocations;
        shortestPathLocations.push_back("Waste Station");
        shortestPathLocations.push_back("Location 1");
        shortestPathLocations.push_back("Location 3");
        shortestPathLocations.push_back("Location 2");
        shortestPathLocations.push_back("Location 4");
        shortestPathLocations.push_back("Location 3");
        shortestPathLocations.push_back("Location 6");
        shortestPathLocations.push_back("Location 5");
        shortestPathLocations.push_back("Location 6");
        shortestPathLocations.push_back("Location 7");

        return shortestPathLocations;
    }
};

#endif
```

GreedyRoute.h:

```
#include <iostream> //Input and Output operations
#include <string> //String operations
#include <vector> //Dynamic array structure operations
#include <stdio.h> //Standard Input/Output operations
#include <stdlib.h> //Standard Library operations
#include <ctime> //To get date and time
#include <cmath> //Math related functions
#include <map> //Store map opearations for key value pairs
#include <limits> //Uses constants that represent the limits of data types
#include <algorithm> //Provides a collection of functions for working with containers
#include <tuple> //Allows the use of fixed-size collection of elements
#include <climits> //So INF and INT MAX can be used
#include "GarbageLocations.h" //Include the GarbageLocations header file

#ifndef GREEDY_ROUTE_H //Header file guard
#define GREEDY_ROUTE_H //Header file guard

#define INF INT_MAX //Set INF to the maximum integer value
```

```
#define V 8 //Set V to 8 which is the number of vertices, because there are 8 locations then define the number of vertices as well for the algorithm to work

using namespace std; //Use standard namespace

class GreedyRoute { //Class for the Greedy Route algorithm

public: //Public specifier
  void createGraph(const GarbageLocations& garbageLocations, int garbageLocationGraph[][V], const vector<string>& currentLocations) { //Create the graph for the Greedy Route algorithm
      for (int index = 0; index < V; index++) { //Iterate over the number of vertices/ locations
          for (int index2 = 0; index2 < V; index2++) { //Then iterate over the number of vertices/ locations again, this is to set the distance between the locations
              garbageLocationGraph[index][index2] = garbageLocations.distanceMatrix.at(currentLocations[index]).at(currentLocations[index2]); //Distance between the locations is initialised here
          }
      }
  }

  void createPath(int parentLocation[], int index, vector<string> currentLocations, ofstream& outFileGR) { //Create the path for the algorithm, initialise the path using the locations
      if (parentLocation[index] == -1) { //If the parent location is equal to -1, then it is the source/ waste station
          outFileGR << "Waste Station";
          cout << "Waste Station";
      }

      if (parentLocation[index] != 0) { //If the parent location is not equal to 0, then it is not the source/ waste station, opposite to the first condition
          createPath(parentLocation, parentLocation[index], currentLocations, outFileGR);
      }

      if (currentLocations[parentLocation[index]] == "Waste Station") { //So if the parent lcoation is source/ waste station, then start by printing the path from "Waste Station -> "
          outFileGR << "Waste Station -> " << currentLocations[index];
          cout << "Waste Station -> " << currentLocations[index];
      } else if (parentLocation[index] != 0) { //If parent location is NOT the source/ waste station, then print the path from " -> ", this way it prints the path consistently
          outFileGR << " -> " << currentLocations[index];
          cout << " -> " << currentLocations[index];
```

```
    } else { //
        outFileGR << currentLocations[index]; //Print the current location
        cout << currentLocations[index]; //Print the current location
    }
  }

  void daSolution(float distance[], int vertex, int parentLocation[], vector<string>
currentLocations, const GarbageLocations& garbageLocations, ofstream& outFileGR) {
//Dijkstra Algorithm solution method
    float totalCost, totalTime, totalFuelConsumption, driverWage, totalDistance = 0;
//Initialise the float variables for the total cost, total time, total fuel consumption, driver
wage and total distance to 0
    int totalWasteCollected = 0; //Initialise the integer total waste collected to 0
    int wasteInKG = 0; //Initialise the integer waste in KG to 0

    outFileGR << "Shortest Path from Waste Station to each location:" << endl;
    cout << "Shortest Path from Waste Station to each location:" << endl;

    vector<string> locationsWithin8Hours;  //Create the vector to store locations that can
be visited under 8 hours

    for (int index = 1; index < V; index++) { //Iterate over the number of vertices/
locations
        if (distance[index] < INF &&
garbageLocations.wasteLevels.at(currentLocations[index]) >= 30) { //If the distance is
less than the defined INf and the waste level is greater than or equal to 30 then perform
the following operations

            totalWasteCollected +=
garbageLocations.wasteLevels.at(currentLocations[index]);
            totalDistance += distance[index];

            outFileGR << "Waste Station to " << currentLocations[index] << ": " <<
distance[index] << " km\t";
            cout << "Waste Station to " << currentLocations[index] << ": " <<
distance[index] << " km\t";
            outFileGR << "Path: ";
            cout << "Path: ";

            createPath(parentLocation, index, currentLocations, outFileGR);

            outFileGR << endl;
            cout << endl;
```

```
            totalTime = totalDistance * 0.2;
            if (totalTime <= 8) { //Check if the total time is less than or equal to 8 hours
                locationsWithin8Hours.push_back(currentLocations[index]);
            } else {
                outFileGR << "\n**Total time exceeds 8 hours, refer to the locations below
that can be visited within 8 hours**" << endl;
                cout << "\n**Total time exceeds 8 hours, refer to the locations below that can
be visited within 8 hours**" << endl;
                break;
            }

            wasteInKG = (totalWasteCollected * 500)/100;
        }
    }

    if (totalTime > 8) { //If the total time is greater than 8 hours, then perform the
following operations
        outFileGR << "\nLocations that can be visited within 8 hours:" << endl;
        cout << "\nLocations that can be visited within 8 hours:" << endl;

        for (int i = 0; i < locationsWithin8Hours.size(); i++) {
            outFileGR << locationsWithin8Hours[i] << endl;
            cout << locationsWithin8Hours[i] << endl;
        }
    }

    if(totalTime > 8) {
        driverWage = 20 * 8;
        outFileGR << "\nDriver Wage for this trip: " << driverWage << " MYR" << endl;
        cout << "\nDriver Wage for this trip: " << driverWage << " MYR" << endl;
    } else {
        driverWage = 20 * totalTime;
        outFileGR << "\nDriver Wage for this trip: " << driverWage << " MYR" << endl;
        cout << "\nDriver Wage for this trip: " << driverWage << " MYR" << endl;
    }

    totalCost = totalDistance * 2.5;
    totalFuelConsumption = 2 * totalDistance;

    outFileGR << "\nTotal Distance for Visited Locations: " << totalDistance << " km" <<
endl;
    cout << "\nTotal Distance for Visited Locations: " << totalDistance << " km" << endl;
    outFileGR << "Total Cost: " << totalCost << " MYR" << endl;
    cout << "Total Cost: " << totalCost << " MYR" << endl;
```

```
    outFileGR << "Total Time: " << totalTime << " hours" << endl;
    cout << "Total Time: " << totalTime << " hours" << endl;
    outFileGR << "Total Fuel Consumption: " << totalFuelConsumption << " Liters" <<
endl;
    cout << "Total Fuel Consumption: " << totalFuelConsumption << " Liters" << endl;
    cout << "Cumulative total percentage of waste collected from all valid locations: " <<
totalWasteCollected << "%" << endl;
    outFileGR << "Cumulative total percentage of waste collected from all valid
locations: " << totalWasteCollected << "%" << endl;
    outFileGR << "Total Waste Collected: " << wasteInKG << "kg" << endl;
    cout << "Total Waste Collected: " << wasteInKG << "kg" << endl;
  }

  int minimumDistance(float distance[], bool shortestPathTree[]) { //Method to find the
minimum distance, this method is used to find the minimum distance from the source to
the destination so that the shortest path can be found through the Dijkstra Algorithm
    float minimumValue = INF; //First assign the minimumValue to the defined INF
    int minimumIndex; //Initialise the minimumIndex to 0

    for (int vertex = 0; vertex < V; vertex++) { //Iterate over the number of vertices which
is the number of locations present
      if (distance[vertex] <= minimumValue && shortestPathTree[vertex] == false){ //If
the distance at the location is less than INF and the shortest path tree at the location is
false then perform the following operations
        minimumValue = distance[vertex];
        minimumIndex = vertex;
      }
    }

    return minimumIndex;
  }

  void dijkstraAlgorithm(int garbageLocationGraph[V][V], int source, const
vector<string>& currentLocations, const GarbageLocations& garbageLocations,
ofstream& outFileGR) { //This is the Dijkstra Algorithm method which contains the
algorithm logic
    float distance[V]; //First initialise the distance array with the number of locations
    bool shortestPathTree[V]; //Then initialise the shortestPathTree array with the
number of locations
    int parentLocation[V]; //Then initialise the parentLocation array with the number of
locations this is to store the parent location of the current location

    for (int index = 0; index < V; index++) { //Iterate over the number of locations
      parentLocation[source] = -1;
```

```
            distance[index] = INF;
            shortestPathTree[index] = false;
        }

        distance[source] = 0; //Assign the source distance as 0

        for (int count = 0; count < V - 1; count++) { //Iterate over the number of locations - 1
            int closestVertex = minimumDistance(distance, shortestPathTree);
            shortestPathTree[closestVertex] = true;

            for (int vertex = 0; vertex < V; vertex++) { //Iterate over the number of locations
                int temporaryVertex = distance[closestVertex] +
garbageLocationGraph[closestVertex][vertex];
                if (garbageLocationGraph[closestVertex][vertex] && temporaryVertex <
distance[vertex] && !shortestPathTree[vertex]) {
                    parentLocation[vertex] = closestVertex;
                    distance[vertex] = distance[closestVertex] +
garbageLocationGraph[closestVertex][vertex];
                }
            }
        }

        daSolution(distance, V, parentLocation, currentLocations, garbageLocations,
outFileGR); //Call the daSolution method to find the solution
    }
};

#endif
```

OptimizedRoute.h:

```
#include <iostream> //Input and Output operations
#include <string> //String operations
#include <vector> //Dynamic array structure operations
#include <stdio.h> //Standard Input/Output operations
#include <stdlib.h> //Standard Library operations
#include <ctime> //To get date and time
#include <cmath> //Math related functions
#include <map> //Store map opearations for key value pairs
#include <limits> //Uses constants that represent the limits of data types
#include <algorithm> //Provides a collection of functions for working with containers
#include <tuple> //Allows the use of fixed-size collection of elements
```

```
#include <climits> //So INF and INT MAX can be used
#include "GarbageLocations.h" //Include the GarbageLocations header file

#ifndef OPTIMIZEDROUTE_H //Header file guard
#define OPTIMIZEDROUTE_H //Header file guard

#define INF INT_MAX //Set INF as the maximum integer value
#define V 8 //Set V to 8 which is the number of vertices, because there are 8 locations then define the number of vertices as well for the algorithm to work

using namespace std; //Use standard namespace

class OptimizedRoute { //Class for the optimized route

public: //Public specifier
  void createGraph(const GarbageLocations& garbageLocations, int garbageLocationGraph[][V], const vector<string>& currentLocations) { //Create the graph for the garbage locations
      for (int index = 0; index < V; index++) { //Iterate over the vertices which is the number of locations, similar to greedy algorithm
          for (int index2 = 0; index2 < V; index2++) { //Iterate again over the number of locations
              garbageLocationGraph[index][index2] = garbageLocations.distanceMatrix.at(currentLocations[index]).at(currentLocations[index2]); //Then set the distances between the locations
          }
      }
  }

  void fwSolution(const GarbageLocations& garbageLocations, int shortestDistanceMatrix[][V], int predecessorMatrix[][V], const vector<string>& currentLocations, ofstream& outFileOR) { //Floyd-Warshall algorithm to find the shortest path
      outFileOR << "Shortest Path from Waste Station to each location:" << endl;
      cout << "Shortest Path from Waste Station to each location:" << endl;
      int totalWasteCollected = 0;
      int wasteInKG = 0;
      int source = 0;
      float totalShortestDistanceMatrix = 0;
      float totalCost, totalTime, totalFuelConsumption, driverWage = 0;
      vector<string> locationsWithin8Hours;
      for (int index = 1; index < V; index++) { //Iterate over the number of locations/ vertices
```

```
        if (garbageLocations.wasteLevels.at(currentLocations[index]) >= 50) { //Then
check if the waste levels are greater than or equal to 50
            if (shortestDistanceMatrix[source][index] <= 11) { //Then check if the shortest
distance is less than or equal to 11 km
                totalWasteCollected +=
garbageLocations.wasteLevels.at(currentLocations[index]);
                totalShortestDistanceMatrix += shortestDistanceMatrix[0][index];
                outFileOR << "Waste Station to " << currentLocations[index] << ": " <<
shortestDistanceMatrix[source][index] << " km\tPath: ";
                cout << "Waste Station to " << currentLocations[index] << ": " <<
shortestDistanceMatrix[source][index] << " km\tPath: ";

                int index2 = index;
                vector<string> shortestPathLocations;
                shortestPathLocations.push_back(currentLocations[index2]);
                while (predecessorMatrix[source][index2] != -1) {

shortestPathLocations.push_back(currentLocations[predecessorMatrix[source][index2]]);
                    index2 = predecessorMatrix[source][index2];
                }

                for (int p = shortestPathLocations.size() - 1; p >= 0; p--) {
                    outFileOR << shortestPathLocations[p];
                    cout << shortestPathLocations[p];
                    if (p != 0) {
                        outFileOR << " -> ";
                        cout << " -> ";
                    }
                }

                totalTime = totalShortestDistanceMatrix * 0.2;
                wasteInKG = (totalWasteCollected * 500)/100;

                if (totalTime <= 8) { //Check if the locations visited is less than or equal to 8
hours which is the time restriction
                    locationsWithin8Hours.push_back(currentLocations[index]);
                } else {
                    outFileOR << "\n**Total time exceeds 8 hours, refer to the locations below
that can be visited within 8 hours**" << endl;
                    cout << "\n**Total time exceeds 8 hours, refer to the locations below that
can be visited within 8 hours**" << endl;
                    break;
                }
```

```
                outFileOR << endl;
                cout << endl;
            } else {
                outFileOR << "Waste Station to " << currentLocations[index] << ": Distance
exceeds 11 km" << endl;
                cout << "Waste Station to " << currentLocations[index] << ": Distance
exceeds 11 km" << endl;
            }
        }
    }

    if (totalTime > 8) { //If the total time exceeds 8 hours
        outFileOR << "\nLocations that can be visited within 8 hours:" << endl;
        cout << "\nLocations that can be visited within 8 hours:" << endl;
        for (int i = 0; i < locationsWithin8Hours.size(); i++) {
            outFileOR << locationsWithin8Hours[i] << endl;
            cout << locationsWithin8Hours[i] << endl;
        }
    }

    if(totalTime > 8) { //If the total time exceeds 8 hours then calculate the driver wage
        driverWage = 20 * 8;
        outFileOR << "\nDriver Wage for this trip: " << driverWage << " MYR" << endl;
//Print the wage
        cout << "\nDriver Wage for this trip: " << driverWage << " MYR" << endl;
    } else {
        driverWage = 20 * totalTime;
        outFileOR << "\nDriver Wage for this trip: " << driverWage << " MYR" << endl;
//Print the wage
        cout << "\nDriver Wage for this trip: " << driverWage << " MYR" << endl;
    }

    totalCost = totalShortestDistanceMatrix * 2.5;
    totalFuelConsumption = 2 * totalShortestDistanceMatrix;

    outFileOR << "\nTotal Distance for Visited Locations: " <<
totalShortestDistanceMatrix << " km" << endl;
    cout << "\nTotal Distance for Visited Locations: " << totalShortestDistanceMatrix << "
km" << endl;
    outFileOR << "Total Cost: " << totalCost << " MYR" << endl;
    cout << "Total Cost: " << totalCost << " MYR" << endl;
    outFileOR << "Total Time: " << totalTime << " hours" << endl;
    cout << "Total Time: " << totalTime << " hours" << endl;
```

```
    outFileOR << "Total Fuel Consumption: " << totalFuelConsumption << " Liters" <<
endl;
    cout << "Total Fuel Consumption: " << totalFuelConsumption << " Liters" << endl;
    cout << "Cumulative total percentage of waste collected from all valid locations: " <<
totalWasteCollected << "%" << endl;
    outFileOR << "Cumulative total percentage of waste collected from all valid
locations: " << totalWasteCollected << "%" << endl;
    outFileOR << "Total Waste Collected: " << wasteInKG << "kg" << endl;
    cout << "Total Waste Collected: " << wasteInKG << "kg" << endl;
  }

  void floydWarshall(const vector<string>& currentLocations, const GarbageLocations&
garbageLocations, ofstream& outFileOR) { //Floyd Warshall Algorithm, this contains the
logic for the floyd warshall algorithm
    int garbageLocationGraph[V][V];
    createGraph(garbageLocations, garbageLocationGraph, currentLocations);

    int shortestDistanceMatrix[V][V]; //Shortest distance matrix, this is used to store the
distances between locations that are visited
    int predecessorMatrix[V][V]; //Predecessor matrix, this is used to store the
predecessor of each location which is used by this algorithm

    for (int index = 0; index < V; index++) { //Iterate over the locations and store the
distances and predecessors
        for (int index2 = 0; index2 < V; index2++) { //Iterate over the locationsand store
the distances and predecessors
            shortestDistanceMatrix[index][index2] = garbageLocationGraph[index][index2];
            if (index != index2 && garbageLocationGraph[index][index2] != INF) {
                predecessorMatrix[index][index2] = index;
            } else {
                predecessorMatrix[index][index2] = -1;
            }
        }
    }

    for (int index = 0; index < V; index++) { //Iterate over the locations
        for (int index2 = 0; index2 < V; index2++) { //Iterate over the locations again
            for (int index3 = 0; index3 < V; index3++) {
                if (shortestDistanceMatrix[index][index3] != INF &&
shortestDistanceMatrix[index2][index] + shortestDistanceMatrix[index][index3] <
shortestDistanceMatrix[index2][index3] && shortestDistanceMatrix[index2][index] != INF)
{
                    predecessorMatrix[index2][index3] = predecessorMatrix[index][index3];
```

```
                    shortestDistanceMatrix[index2][index3] =
shortestDistanceMatrix[index2][index] + shortestDistanceMatrix[index][index3];
                }
            }
        }
    }

    fwSolution(garbageLocations, shortestDistanceMatrix, predecessorMatrix,
currentLocations, outFileOR); //Call the method to print the solution
  }
};

#endif
```

TSPRoute.h:

```
#include <iostream> //Input and Output operations
#include <string> //String operations
#include <vector> //Dynamic array structure operations
#include <stdio.h> //Standard Input/Output operations
#include <stdlib.h> //Standard Library operations
#include <ctime> //To get date and time
#include <cmath> //Math related functions
#include <map> //Store map opearations for key value pairs
#include <limits> //Uses constants that represent the limits of data types
#include <algorithm> //Provides a collection of functions for working with containers
#include <tuple> //Allows the use of fixed-size collection of elements
#include <climits> //So INF and INT MAX can be used
#include "GarbageLocations.h" //Include the GarbageLocations header file

#ifndef TSPROUTE_H //Header file guard
#define TSPROUTE_H //Header file guard

#define INF INT_MAX //Set INF to the maximum integer value
#define V 8 //Set V to 8 which is the number of vertices, because there are 8 locations
then define the number of vertices as well for the algorithm to work

using namespace std; //Use standard namespace

class TSPRoute { //Class for the TSPRoute

public: //Public specifier
```

```
  void createGraph(int garbageLocationGraph[][V], const vector<string>&
currentLocations, const GarbageLocations& garbageLocations) { //Create the graph for
the TSP algorithm
      for (int index = 0; index < V; index++) { //Iterate over the number of vertices which is
the number of locations
          for (int index2 = 0; index2 < V; index2++) {
              garbageLocationGraph[index][index2] =
garbageLocations.distanceMatrix.at(currentLocations[index]).at(currentLocations[index2]
);
          }
      }
  }

  int minimumDistance(float distance[], bool shortestPathTree[]) {//Find the minimum
distance
      float minimumValue = INF; //Set the minimum value to the INF which indicates the
maximum integer value
      int minimumIndex; //Declare the minimum index

      for (int v = 0; v < V; v++) //Iterate over the number of vertices
          if (shortestPathTree[v] == false && distance[v] <= minimumValue){ //If the shortest
path tree is false and the distance is less than or equal to the minimum value
              minimumValue = distance[v]; //Then assign the minimum value to the distance
              minimumIndex = v; //And the minimum index to the vertex
          }
      return minimumIndex;
  }

  void tspPath(int parentLocation[], int index, const vector<string>& currentLocations,
ofstream& outFileTSP) { //Find the path for the TSP algorithm
      if (parentLocation[index] == -1) { //If the parent location is -1 it means that the
location is the waste station
          outFileTSP << "Waste Station";
          cout << "Waste Station";
          return;
      }

      tspPath(parentLocation, parentLocation[index], currentLocations, outFileTSP);
//Recursively call the function to find the path, this is so that the path is printed in the
correct order

      if (currentLocations[parentLocation[index]] != currentLocations[index]) { //If the
parent location is not equal to the current location
          outFileTSP << " -> " << currentLocations[index];
```

```
            cout << " -> " << currentLocations[index];
        }
    }

    void tspSolution(float distance[], int vertex, int parent[], const vector<string>&
currentLocations, const GarbageLocations& garbageLocations, ofstream& outFileTSP) {
//Find the solution for the TSP algorithm, this is the logic for the algorithm
        outFileTSP << "Shortest Path from Waste Station to each location:" << endl;
        cout << "Shortest Path from Waste Station to each location:" << endl;

        float totalDistance = 0;
        float totalCost, totalTime, totalFuelConsumption, driverWage = 0;
        int totalWasteCollected = 0;
        int wasteInKG = 0;
        vector<string> locationsWithin8Hours;

        for (int index = 1; index < V; index++) { //Iterate over the number of vertices which is
the number of locations
            if (garbageLocations.wasteLevels.at(currentLocations[index]) >= 60) { //If the
waste level is greater than or equal to 60
                if (distance[index] <= 12) { //If the distance is less than or equal to 12
                    totalDistance += distance[index];
                    totalWasteCollected +=
garbageLocations.wasteLevels.at(currentLocations[index]);
                    outFileTSP << "Waste Station to " << currentLocations[index] << ": " <<
distance[index] << " km\t";
                    cout << "Waste Station to " << currentLocations[index] << ": " <<
distance[index] << " km\t";
                    outFileTSP << "Path: ";
                    cout << "Path: ";

                    tspPath(parent, index, currentLocations, outFileTSP);

                    outFileTSP << endl;
                    cout << endl;
                    totalTime = totalDistance * 0.2;
                    if (totalTime <= 8) {
                        locationsWithin8Hours.push_back(currentLocations[index]);
                    } else {
                        outFileTSP << "**Total time exceeds 8 hours, refer to the locations below
that can be visited within 8 hours**" << endl;
                        cout << "**Total time exceeds 8 hours, refer to the locations below that
can be visited within 8 hours**" << endl;
                        break;
```

```
                }
            } else {
                outFileTSP << "Waste Station to " << currentLocations[index] << ": Distance
exceeds 12 km" << endl;
                cout << "Waste Station to " << currentLocations[index] << ": Distance
exceeds 12 km" << endl;
            }
        }
    }

    totalCost = totalDistance * 2.5;
    totalFuelConsumption = 2 * totalDistance;
    wasteInKG = (totalWasteCollected * 500)/100;

    if (totalTime > 8) { //If the total time exceeds 8 hours
        outFileTSP << "\nTotal time exceeds 8 hours" << endl;
        cout << "\nLocations that can be visited within 8 hours:" << endl;
        for (int i = 0; i < locationsWithin8Hours.size(); i++) {
            outFileTSP << locationsWithin8Hours[i] << endl;
            cout << locationsWithin8Hours[i] << endl;
        }
    }

    if(totalTime > 8) { //If the total time exceeds 8 hours then calculate the drivers wage
accordingly
        driverWage = 20 * 8;
        outFileTSP << "\nDriver Wage for this trip: " << driverWage << " MYR" << endl;
        cout << "\nDriver Wage for this trip: " << driverWage << " MYR" << endl;
    } else {
        driverWage = 20 * totalTime;
        outFileTSP << "\nDriver Wage for this trip: " << driverWage << " MYR" << endl;
        cout << "\nDriver Wage for this trip: " << driverWage << " MYR" << endl;
    }

    outFileTSP << "\nTotal Distance for Visited Locations: " << totalDistance << " km" <<
endl;
    cout << "\nTotal Distance for Visited Locations: " << totalDistance << " km" << endl;
    outFileTSP << "Total Cost: " << totalCost << " MYR" << endl;
    cout << "Total Cost: " << totalCost << " MYR" << endl;
    outFileTSP << "Total Time: " << totalTime << " hours" << endl;
    cout << "Total Time: " << totalTime << " hours" << endl;
    outFileTSP << "Total Fuel Consumption: " << totalFuelConsumption << " Liters" <<
endl;
    cout << "Total Fuel Consumption: " << totalFuelConsumption << " Liters" << endl;
```

```
    outFileTSP << "Cumulative total percentage of waste collected from all valid locations: " << totalWasteCollected << "%" << endl;
    cout << "Cumulative total percentage of waste collected from all valid locations: " << totalWasteCollected << "%" << endl;
    outFileTSP << "Total Waste Collected: " << wasteInKG << "kg" << endl;
    cout << "Total Waste Collected: " << wasteInKG << "kg" << endl;
  }

  void travellingSalesmanProblem(int garbageLocationGraph[][V], const vector<string>& currentLocations, const GarbageLocations& garbageLocations, ofstream& outFileTSP) {
//Implement the travelling salesman problem algorithm
    float dist[V]; //Create an array to store the distance, the size is the number of locations
    bool shortestPathTree[V]; //Create an array to store the shortest path tree, the size is the number of locations
    int parent[V]; //Create an array to store the parent, the size is the number of locations

    for (int index = 0; index < V; index++) { //Iterate over all locations
      parent[0] = -1; //Set the source vertex to -1
      dist[index] = INF; //Then set the distance to INF
      shortestPathTree[index] = false; //Set the shortest path tree to false
    }

    dist[0] = 0; //Set source vertex distance to 0

    for (int index2 = 0; index2 < V - 1; index2++) { //Iterate over the number of vertices - 1 which is the number of locations - 1
      int closestVertex = minimumDistance(dist, shortestPathTree);
      shortestPathTree[closestVertex] = true;

      for (int index3 = 0; index3 < V; index3++) //Iterate over the number of vertices which is the number of locations
        if (garbageLocationGraph[closestVertex][index3] && dist[closestVertex] + garbageLocationGraph[closestVertex][index3] < dist[index3] && !shortestPathTree[index3]) {
          parent[index3] = closestVertex;
          dist[index3] = dist[closestVertex] + garbageLocationGraph[closestVertex][index3];
        }
    }

    tspSolution(dist, V, parent, currentLocations, garbageLocations, outFileTSP); //Call the tspSolution function
```

```
  }
};

#endif
```

Locations.h:

```
#include <iostream> //Input and Output operations
#include <string> //String operations
#include <vector> //Dynamic array structure operations
#include <stdio.h> //Standard Input/Output operations
#include <stdlib.h> //Standard Library operations
#include <ctime> //To get date and time
#include <cmath> //Math related functions
#include <map> //Store map opearations for key value pairs
#include <limits> //Uses constants that represent the limits of data types
#include <algorithm> //Provides a collection of functions for working with containers
#include <tuple> //Allows the use of fixed-size collection of elements
#include <climits> //So INF and INT MAX can be used

#ifndef LOCATIONS_H //Header file guard
#define LOCATIONS_H //Header file guard

using namespace std; //Use standard namespace

class Locations{ //Class for the locations

public: //Public specifier
  void initializeLocations(vector<string>& currentLocations) { //Initialize the locations and
push all the locations to the vector
    currentLocations.push_back("Waste Station");
    currentLocations.push_back("Location 1");
    currentLocations.push_back("Location 2");
    currentLocations.push_back("Location 3");
    currentLocations.push_back("Location 4");
    currentLocations.push_back("Location 5");
    currentLocations.push_back("Location 6");
    currentLocations.push_back("Location 7");
  }
};

#endif
```

wasteLevel.h:

```
#include <iostream> //Input and Output operations
#include <string> //String operations
#include <vector> //Dynamic array structure operations
#include <stdio.h> //Standard Input/Output operations
#include <stdlib.h> //Standard Library operations
#include <ctime> //To get date and time
#include <cmath> //Math related functions
#include <map> //Store map opearations for key value pairs
#include <limits> //Uses constants that represent the limits of data types
#include <algorithm> //Provides a collection of functions for working with containers
#include <tuple> //Allows the use of fixed-size collection of elements
#include <climits> //So INF and INT MAX can be used
#include "GarbageLocations.h" //Include the GarbageLocations header file

#ifndef WASTELEVEL_H //Header file guard
#define WASTELEVEL_H //Header file guard

using namespace std; //Use standard namespace

class wasteLevel { //Class for the waste levels

public: //Public specifier
  void wasteLevels(const GarbageLocations& garbageLocations, ofstream &outFile) {
//Method to display the waste levels
      outFile << "\nCurrent Waste Levels: \n";
      cout << "\nCurrent Waste Levels: \n";

      int wasteInKG = 0; //Initialize the waste in KG variable as integer
      int maxWaste = 500; //Initialize the maximum waste as 500

      for (map<string, map<string, int> >::const_iterator iterator =
garbageLocations.distanceMatrix.begin(); iterator !=
garbageLocations.distanceMatrix.end(); iterator++) { //Iterate through the distance matrix
          const pair<string, map<string, int> >& currentLocation = *iterator; //Initialize the
current location as a key value pair
          if (currentLocation.first != "Waste Station") { //If the current location is NOT the
waste station/ source
              outFile << "Waste at " << currentLocation.first << ": " <<
garbageLocations.wasteLevels.at(currentLocation.first) << "%" << endl;
              cout << "Waste at " << currentLocation.first << ": " <<
garbageLocations.wasteLevels.at(currentLocation.first) << "%" << endl;
```

```
        }
    }

    cout << "\nWaste in kg: \n";
    for (map<string, map<string, int> >::const_iterator iterator =
garbageLocations.distanceMatrix.begin(); iterator !=
garbageLocations.distanceMatrix.end(); iterator++) { //Iterate through the distance matrix
        const pair<string, map<string, int> >& currentLocation = *iterator; //Initialize the
current location as a key value pair
        if (currentLocation.first != "Waste Station") { //If the current location is NOT the
waste station/ source
            wasteInKG = (maxWaste *
garbageLocations.wasteLevels.at(currentLocation.first)) / 100; //Calculate the waste in
KG
            outFile << "Waste in KG at " << currentLocation.first << ": " << wasteInKG <<
"kg" << endl;
            cout << "Waste in KG at " << currentLocation.first << ": " << wasteInKG << "kg"
<< endl;
        }
    }

    outFile << "\n";
    cout << "\n";
  }

  void wasteLevels(const GarbageLocations& garbageLocations) {
    int wasteInKG = 0;
    int maxWaste = 500;

    cout << "\nCurrent Waste Levels: \n";
    for (map<string, map<string, int> >::const_iterator iterator =
garbageLocations.distanceMatrix.begin(); iterator !=
garbageLocations.distanceMatrix.end(); iterator++) { //Iterate through the distance matrix
        const pair<string, map<string, int> >& currentLocation = *iterator;
        if (currentLocation.first != "Waste Station") { //If the current location is NOT the
waste station/ source
            cout << "Waste at " << currentLocation.first << ": " <<
garbageLocations.wasteLevels.at(currentLocation.first) << "%" << endl;
        }
    }

    cout << "\nWaste in kg: \n";
```

```
      for (map<string, map<string, int> >::const_iterator iterator =
garbageLocations.distanceMatrix.begin(); iterator !=
garbageLocations.distanceMatrix.end(); iterator++) { //Iterate through the distance matrix
          const pair<string, map<string, int> >& currentLocation = *iterator;
          if (currentLocation.first != "Waste Station") { //If the current location is NOT the
waste station/ source
              wasteInKG = (maxWaste *
garbageLocations.wasteLevels.at(currentLocation.first)) / 100;
              cout << "Waste in KG at " << currentLocation.first << ": " << wasteInKG << "kg"
<< endl;
          }
      }
      cout << "\n";
  }
};

#endif
```

main.cpp:

```
#include <iostream> //Input and Output operations
#include <string> //String operations
#include <vector> //Dynamic array structure operations
#include <stdio.h> //Standard Input/Output operations
#include <stdlib.h> //Standard Library operations
#include <ctime> //To get date and time
#include <cmath> //Math related functions
#include <map> //Store map opearations for key value pairs
#include <limits> //Uses constants that represent the limits of data types
#include <algorithm> //Provides a collection of functions for working with containers
#include <tuple> //Allows the use of fixed-size collection of elements
#include <climits> //So INF and INT MAX can be used
#include <fstream> //File operations
#include "GarbageLocations.h" //GarbageLocations header file
#include "NonOptimizedRoute.h" //NonOptimizedRoute header file
#include "GreedyRoute.h" //GreedyRoute header file
#include "OptimizedRoute.h" //OptimizedRoute header file
#include "TSPRoute.h" //TSPRoute header file
#include "Locations.h" //Locations header file
#include "wasteLevel.h" //WasteLevel header file
#include "distanceMap.h" //DistanceMap header file

using namespace std; //Use standard namespace
```

```
int main() { //Main function
  ofstream outFile1("NORoutput.txt"); //Output file for Non-Optimized Route
  ofstream outFile2("GRoutput.txt"); //Output file for Greedy Route
  ofstream outFile3("ORoutput.txt"); //Output file for Optimized Route
  ofstream outFile4("TSPoutput.txt"); //Output file for TSP Route
  ofstream outFile5("MAPoutput.txt"); //Output file for Map

  NonOptimizedRoute nonOptimizedRoute; //Creating object for NonOptimizedRoute, instance of the class
  GreedyRoute greedyRoute; //Creating object for GreedyRoute, instance of the class
  OptimizedRoute optimizedRoute; //Creating object for OptimizedRoute, instance of the class
  TSPRoute tspRoute; //Creating object for TSPRoute, instance of the class
  Locations location; //Creating object for Locations, instance of the class
  distanceMap distancemap; //Creating object for DistanceMap, instance of the class
  wasteLevel wastelevel; //Creating object for WasteLevel, instance of the class
  GarbageLocations garbageLocations; //Creating object for GarbageLocations, instance of the class

  bool loop = true; //Boolean variable loop is true
  bool choice2 = false; //Boolean variable choice2 is false
  bool loopagain = true; //Boolean variable loopagain is true

  while(loop == true){ //While loop is true so the loop will intially run first then depending on the condition it will stop
      choice2 = false; //Choice2 is false
      vector<string> locations; //Vector of strings for locations

      cout << "\n-----------------------------------------------------\n";
      cout << "\nWelcome to the Waste Management System\n";

      cout << "\nEnter your choice of which algorithm you want to run: \n";
      cout << "1. Non-Optimized Route - Fixed Shortest Path Algorithm" << endl;
      cout << "2. Greedy Route - Djikstra Algorithm" << endl;
      cout << "3. Optimized Route - Floyd Warshall Algorithm" << endl;
      cout << "4. TSP Route - Travelling Salesman Problem Algorithm" << endl;
      cout << "5. Print Map - Graph, Distance Matrix" << endl;
      cout << "6. Regenerate Waste Levels" << endl;
      cout << "7. Exit\n" << endl;

      wastelevel.wasteLevels(garbageLocations); //Generate an initial waste level for each location

      int choice;
```

```
    cout << "Enter your choice (1 - 7): ";
    cin >> choice; //User input for choice

    switch (choice) {
       case 1: {
          choice2 = true;
          outFile1 << "\n-----------------------------------------------------\n";
          outFile1 << "\nNon-Optimized Route - Fixed Shortest Path Algorithm\n";
          outFile1 << "\n-----------------------------------------------------\n";

          cout << "\n-----------------------------------------------------\n";
          cout << "\nNon-Optimized Route - Fixed Shortest Path Algorithm\n";
          cout << "\n-----------------------------------------------------\n";

          outFile1 << "\nDetails of Non-Optimized Route" << endl;
          outFile1 << "1. Waste at location must be >=40% to be visited and collected"
<< endl;
          outFile1 << "2. Distance from Waste Station to location must be <= 30km" <<
endl;
          outFile1 << "3. Time taken must be within 12 hours, or else location will not be
visited" << endl;

          cout << "\nDetails of Non-Optimized Route" << endl;
          cout << "1. Waste at location must be >=40% to be visited and collected" <<
endl;
          cout << "2. Distance from Waste Station to location must be <= 30km" << endl;
          cout << "3. Time taken must be within 12 hours, or else location will not be
visited" << endl;

          location.initializeLocations(locations); //Call the initializeLocations method from
the Locations class
          wastelevel.wasteLevels(garbageLocations, outFile1); //Call the wasteLevels
method from the WasteLevel class

          vector<string> shortestPath =
nonOptimizedRoute.calculateShortestPath(locations); //Call the calculateShortestPath
method from the NonOptimizedRoute class
          nonOptimizedRoute.noSolution(garbageLocations, locations, shortestPath,
outFile1); //Call the noSolution method from the NonOptimizedRoute class
          cout << "\n";
          break;
       }
       case 2: {
          choice2 = true;
```

```
            outFile2 << "\n---------------------------------\n";
            outFile2 << "\nGreedy Route - Dijkstra Algorithm\n";
            outFile2 << "\n---------------------------------\n";

            cout << "\n---------------------------------\n";
            cout << "\nGreedy Route - Dijkstra Algorithm\n";
            cout << "\n---------------------------------\n";

            outFile2 << "\nDetails of Greedy Route" << endl;
            outFile2 << "1. Waste at location must be >=30% to be visited and collected"
<< endl;
            outFile2 << "2. There is NO distance restriction for the location to be visited" <<
endl;
            outFile2 << "3. Time taken must be within 8 hours, or else location will not be
visited" << endl;

            cout << "\nDetails of Greedy Route" << endl;
            cout << "1. Waste at location must be >=30% to be visited and collected" <<
endl;
            cout << "2. There is NO distance restriction for the location to be visited" <<
endl;
            cout << "3. Time taken must be within 8 hours, or else location will not be
visited" << endl;

            location.initializeLocations(locations); //Call the initializeLocations method from
the Locations class
            wastelevel.wasteLevels(garbageLocations, outFile2); //Call the wasteLevels
method from the WasteLevel class

            int graph[V][V]; //Graph with V vertices and V edges
            greedyRoute.createGraph(garbageLocations, graph, locations); //Call the
createGraph method from the GreedyRoute class
            greedyRoute.dijkstraAlgorithm(graph, 0, locations, garbageLocations, outFile2);
//Call the dijkstraAlgorithm method from the GreedyRoute class
            cout << "\n";
            break;
        }
        case 3: {
            choice2 = true;
            outFile3 << "\n-----------------------------------------\n";
            outFile3 << "\nOptimized Route - Floyd Warshall Algorithm\n";
            outFile3 << "\n-----------------------------------------\n";

            cout << "\n-----------------------------------------\n";
```

```
            cout << "\nOptimized Route - Floyd Warshall Algorithm\n";
            cout << "\n----------------------------------------\n";

            outFile3 << "\nDetails of Optimized Route" << endl;
            outFile3 << "1. Waste at location must be >=50% to be visited and collected"
<< endl;
            outFile3 << "2. Distance from Waste Station to location must be <= 11km" <<
endl;
            outFile3 << "3. Time taken must be within 8 hours, or else location will not be
visited" << endl;

            cout << "\nDetails of Optimized Route" << endl;
            cout << "1. Waste at location must be >=50% to be visited and collected" <<
endl;
            cout << "2. Distance from Waste STation to location must be <= 11km" << endl;
            cout << "3. Time taken must be within 8 hours, or else location will not be
visited" << endl;

            location.initializeLocations(locations); //Call the initializeLocations method from
the Locations class
            wastelevel.wasteLevels(garbageLocations, outFile3); //Call the wasteLevels
method from the WasteLevel class

            optimizedRoute.floydWarshall(locations, garbageLocations, outFile3); //Call the
floydWarshall method from the OptimizedRoute class
            cout << "\n";
            break;
        }
        case 4: {
            choice2 = true;
            outFile4 << "\n----------------------------------------------------\n";
            outFile4 << "\nTSP Route - Travelling Salesman Problem Algorithm\n";
            outFile4 << "\n----------------------------------------------------\n";

            cout << "\n----------------------------------------------------\n";
            cout << "\nTSP Route - Travelling Salesman Problem Algorithm\n";
            cout << "\n----------------------------------------------------\n";

            outFile4 << "\nDetails of TSP Route" << endl;
            outFile4 << "1. Waste at location must be >=60% to be visited and collected"
<< endl;
            outFile4 << "2. Distance from Waste Station to location must be <= 12km" <<
endl;
```

```
            outFile4 << "3. Time taken must be within 8 hours, or else location will not be
visited" << endl;

            cout << "\nDetails of TSP Route" << endl;
            cout << "1. Waste at location must be >=60% to be visited and collected" <<
endl;
            cout << "2. Distance from Waste Station to location must be <= 12km" << endl;
            cout << "3. Time taken must be within 8 hours, or else location will not be
visited" << endl;

            location.initializeLocations(locations); //Call the initializeLocations method from
the Locations class
            wastelevel.wasteLevels(garbageLocations, outFile4); //Call the wasteLevels
method from the WasteLevel class

            int graph[V][V]; //Graph with V vertices and V edges
            tspRoute.createGraph(graph, locations, garbageLocations); //Call the
createGraph method from the TspRoute class
            tspRoute.travellingSalesmanProblem(graph, locations, garbageLocations,
outFile4); //Call the travellingSalesmanProblem method from the TspRoute class
            cout << "\n";
            break;
        }
        case 5: {
            choice2 = true;
            cout << "\nGraph: \n";
            distancemap.printMap(outFile5); //Call the printMap method from the
DistanceMap class
            distancemap.printMapInformation(outFile5); //Call the printMapInformation
method from the DistanceMap class
            cout << "\n";
            break;
        }
        case 6: {
            choice2 = false;
            cout << "\nRegenerating waste levels\n" << endl;
            garbageLocations.regenerateWasteLevels(); //Call the regenerateWasteLevels
method from the GarbageLocations class
            break;
        }
        case 7: {
            choice2 = false;
            cout << "\nExiting the program\n" << endl;
            return 0;
```

```
            break;
        }
        default: { //If the user enters an invalid choice then clear the input buffer and ask the user to enter a valid choice
            cout << "Invalid choice. Please enter a valid choice.\n";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            break;
        }
    }

    if(choice2 == true) { //If the user has selected an algorithm or function then ask the user if they want to run another algorithm or perform other functions
        loopagain = true;
        while(loopagain == true) {
            cout << "Do you want to run another algorithm or perform other functions? (y/n): ";
            char run;
            cin >> run;

            if (run == 'n' || run == 'N') {
                cout << "\nExiting the program\n" << endl;
                loop = false;
                loopagain = false;
            } else if (run == 'y' || run == 'Y') {
                loop = true;
                loopagain = false;
            } else {
                cout << "Invalid choice. Enter again." << endl;
                loop = false;
                loopagain = true;
                cin.clear();
                cin.ignore(numeric_limits<streamsize>::max(), '\n');
            }
        }
    }
  }

  return 0;
}
```

NORoutput.txt (Sample Output):

Non-Optimized Route - Fixed Shortest Path Algorithm

Details of Non-Optimized Route
1. Waste at location must be >=40% to be visited and collected
2. Distance from Waste Station to location must be <= 30km
3. Time taken must be within 12 hours, or else location will not be visited

Current Waste Levels:
Waste at Location 1: 8%
Waste at Location 2: 84%
Waste at Location 3: 72%
Waste at Location 4: 9%
Waste at Location 5: 51%
Waste at Location 6: 28%
Waste at Location 7: 81%
Waste in KG at Location 1: 40kg
Waste in KG at Location 2: 420kg
Waste in KG at Location 3: 360kg
Waste in KG at Location 4: 45kg
Waste in KG at Location 5: 255kg
Waste in KG at Location 6: 140kg
Waste in KG at Location 7: 405kg

Shortest Path from Waste Station to each location:
Waste Station to Location 2: 14 km Path: Waste Station -> Location 1 -> Location 3 -> Location 2
Waste Station to Location 3: 9 km Path: Waste Station -> Location 1 -> Location 3
Waste Station to Location 5: 29 km Path: Waste Station -> Location 1 -> Location 3 -> Location 2 -> Location 4 -> Location 3 -> Location 6 -> Location 5
Waste Station to Location 7: Distance exceeds 30 km, Location not visited

Driver Wage for this trip: 208 MYR

Total Distance for Visited Locations: 52 km
Total Cost: 130 MYR
Total Time: 10.4 hours
Total Fuel Consumption: 104 Liters
Cumulative total percentage of waste collected from all valid locations: 207%
Total Waste Collected: 1035kg

GRoutput.txt (Sample Output):

Greedy Route - Dijkstra Algorithm

Details of Greedy Route
1. Waste at location must be >=30% to be visited and collected
2. There is NO distance restriction for the location to be visited
3. Time taken must be within 8 hours, or else location will not be visited

Current Waste Levels:
Waste at Location 1: 8%
Waste at Location 2: 84%
Waste at Location 3: 72%
Waste at Location 4: 9%
Waste at Location 5: 51%
Waste at Location 6: 28%
Waste at Location 7: 81%
Waste in KG at Location 1: 40kg
Waste in KG at Location 2: 420kg
Waste in KG at Location 3: 360kg
Waste in KG at Location 4: 45kg
Waste in KG at Location 5: 255kg
Waste in KG at Location 6: 140kg
Waste in KG at Location 7: 405kg

Shortest Path from Waste Station to each location:
Waste Station to Location 2: 14 km Path: Waste Station -> Location 1 -> Location 3 -> Location 2
Waste Station to Location 3: 9 km Path: Waste Station -> Location 1 -> Location 3
Waste Station to Location 5: 14 km Path: Waste Station -> Location 7 -> Location 6 -> Location 5
Waste Station to Location 7: 4 km Path: Waste Station -> Location 7

Total time exceeds 8 hours, refer to the locations below that can be visited within 8 hours

Locations that can be visited within 8 hours:
Location 2
Location 3
Location 5

Driver Wage for this trip: 160 MYR

Total Distance for Visited Locations: 41 km
Total Cost: 102.5 MYR
Total Time: 8.2 hours
Total Fuel Consumption: 82 Liters
Cumulative total percentage of waste collected from all valid locations: 288%
Total Waste Collected: 1035kg

ORoutput.txt (Sample Output):

Optimized Route - Floyd Warshall Algorithm

Details of Optimized Route
1. Waste at location must be >=50% to be visited and collected
2. Distance from Waste Station to location must be <= 11km
3. Time taken must be within 8 hours, or else location will not be visited

Current Waste Levels:
Waste at Location 1: 8%
Waste at Location 2: 84%
Waste at Location 3: 72%
Waste at Location 4: 9%
Waste at Location 5: 51%
Waste at Location 6: 28%
Waste at Location 7: 81%
Waste in KG at Location 1: 40kg
Waste in KG at Location 2: 420kg
Waste in KG at Location 3: 360kg
Waste in KG at Location 4: 45kg
Waste in KG at Location 5: 255kg
Waste in KG at Location 6: 140kg
Waste in KG at Location 7: 405kg

Shortest Path from Waste Station to each location:
Waste Station to Location 2: Distance exceeds 11 km
Waste Station to Location 3: 9 km Path: Waste Station -> Location 1 -> Location 3
Waste Station to Location 5: Distance exceeds 11 km
Waste Station to Location 7: 4 km Path: Waste Station -> Location 7

Driver Wage for this trip: 52 MYR

Total Distance for Visited Locations: 13 km
Total Cost: 32.5 MYR
Total Time: 2.6 hours
Total Fuel Consumption: 26 Liters
Cumulative total percentage of waste collected from all valid locations: 153%
Total Waste Collected: 765kg

TSPoutput.txt (Sample Output):

TSP Route - Travelling Salesman Problem Algorithm

Details of TSP Route
1. Waste at location must be >=60% to be visited and collected
2. Distance from Waste Station to location must be <= 12km
3. Time taken must be within 8 hours, or else location will not be visited

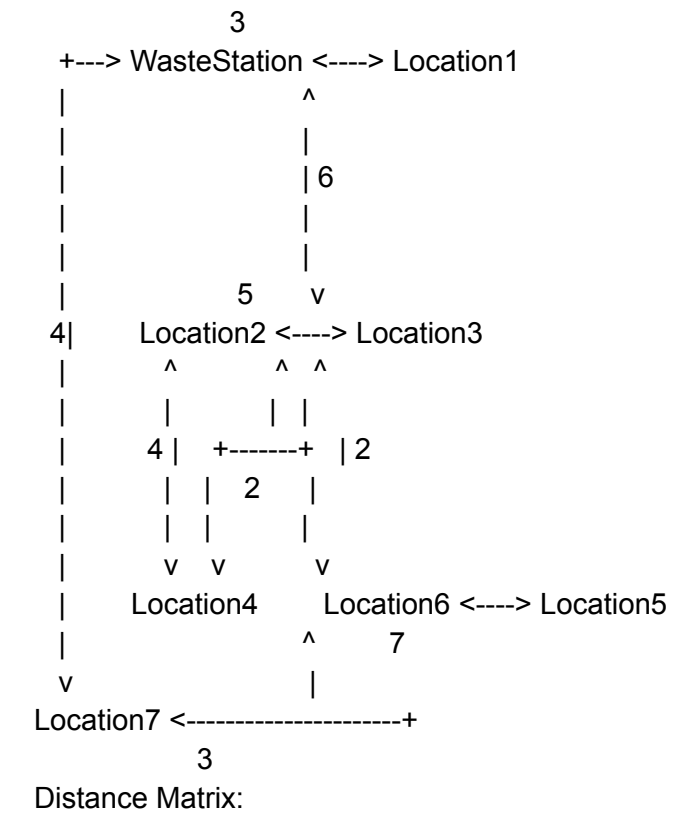
Current Waste Levels:
Waste at Location 1: 8%
Waste at Location 2: 84%
Waste at Location 3: 72%
Waste at Location 4: 9%
Waste at Location 5: 51%
Waste at Location 6: 28%
Waste at Location 7: 81%
Waste in KG at Location 1: 40kg
Waste in KG at Location 2: 420kg
Waste in KG at Location 3: 360kg
Waste in KG at Location 4: 45kg
Waste in KG at Location 5: 255kg
Waste in KG at Location 6: 140kg
Waste in KG at Location 7: 405kg

Shortest Path from Waste Station to each location:
Waste Station to Location 2: Distance exceeds 12 km
Waste Station to Location 3: 9 km Path: Waste Station -> Location 1 -> Location 3
Waste Station to Location 7: 4 km Path: Waste Station -> Location 7

Driver Wage for this trip: 52 MYR

Total Distance for Visited Locations: 13 km
Total Cost: 32.5 MYR
Total Time: 2.6 hours
Total Fuel Consumption: 26 Liters
Cumulative total percentage of waste collected from all valid locations: 153%
Total Waste Collected: 765kg

MAPoutput.txt (Sample Output):



Distance Matrix:

	Waste Station	Location 1	Location 2	Location 3	Location 4	Location 5	Location 6	Location 7
Waste Station	0	3	INF	INF	INF	INF	INF	4
Location 1	3	0	INF	6	INF	INF	INF	INF
Location 2	INF	INF	0	5	4	INF	INF	INF
Location 3	INF	6	5	0	2	INF	2	INF

Location 4	INF	INF	4	2	0	INF	INF	INF
Location 5	INF	INF	INF	INF	INF	0	7	INF
Location 6	INF	INF	INF	2	INF	7	0	3
Location 7	4	INF	INF	INF	INF	INF	3	0