

Emergency Vehicle Dispatching System

Implemented using python

Anna Johnson | Kyle Son | Computer Science 404 Project 1

Githublink: <https://github.com/Sonjongkook/Group4-Emergency-Vehicle-Dispatching-System-Project->

# Assumptions

1. Requests are processed by order of their ID. Once a vehicle is assigned to a request, it becomes unavailable for use in later requests.
2. When building the graph, edges are assumed to be undirected.
3. The number of vehicles available is greater than the number of vehicles requested.
4. If the distance is the same for multiple vehicles, we make the selection based on ID number in ascending order (if there are two vehicles equidistance from a request zip code, we would send vehicle with ID 1 over vehicle with ID 2).

# Project Workflow

1. Make json data according to Project guidelines
2. Process the information from the Json file
3. Find shortest path from request zip code to zip code with available vehicle of the correct type using Dijkstra’s Shortest Path Algorithm
4. Dispatch most suitable vehicle to the request and set it to unavailable

Step 1: Creating a Json file

As described in the project guidelines, we have created data for three tables: vehicles, requests, and distances. This is shown in the Json file.

The first section in the Json file is for vehicles. Each vehicle listed has an ID, type, and zip code. The vehicles are in ascending order by their IDs. Vehicles types are in accordance with the project description – (1) Ambulance, (2) Firetruck, and (3) Police Car.

The second section is for requests. Each request has an ID, type of vehicles, and zip code. The requests are also ordered by ascending ID.

The last section, distances, contains zip code 1, zip code 2, and the distance between the two. This data will be used to construct the graph.

We can manually input all of this information or use a Json generator in the source code.

Step 2: Process information from Json file

Iterate through all of the vehicles and set their availability to “available”.

We also need the construct the graph using the Graph class and the “distances” section of the Json file. Vertices will be formed from the “zip code 1” and “zip code 2” data, and edges will be weighted with the distances between them.

Step 3: Finding shortest path using Dijkstra’s Shortest Path Algorithm

In order to implement an efficient algorithm, we have used Dijkstra's algorithm, which provides the minimum shortest path without leaving any vertex behind.

For each request, reset the graph so all vertices are initialized back to their starting state.

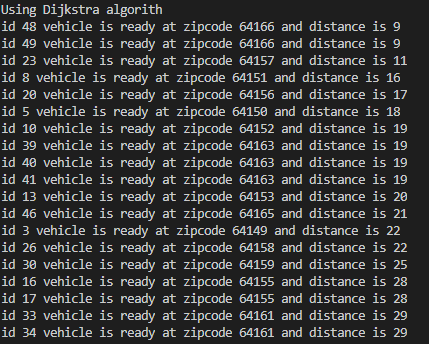
We create a list to hold available vehicles and populate it by iterating over each vehicle and adding it only if the type matches the request type and if the vehicle is set to available.

We then run Dijkstra’s algorithm with the request node as the starting point. The algorithm will find the distance from the starting node to each vertex in the graph.

Step 4: Dispatch most suitable vehicle to the request

We order the list of available vehicles by their distances found in the Dijkstra function.

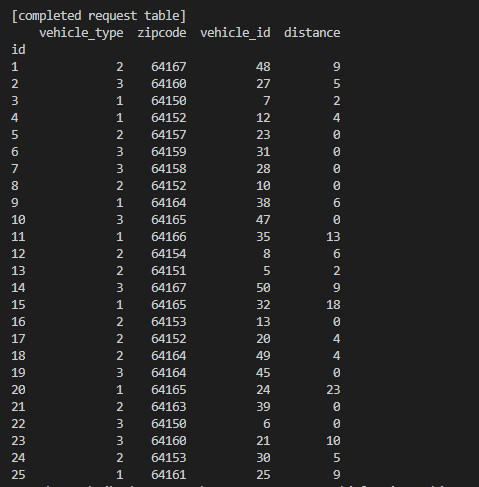
We can then print this list to the console:



Lastly, we will assign the vehicle at the list index ‘0’ to the request, set this vehicle’s availability to ‘False’, and print the solution to the console:

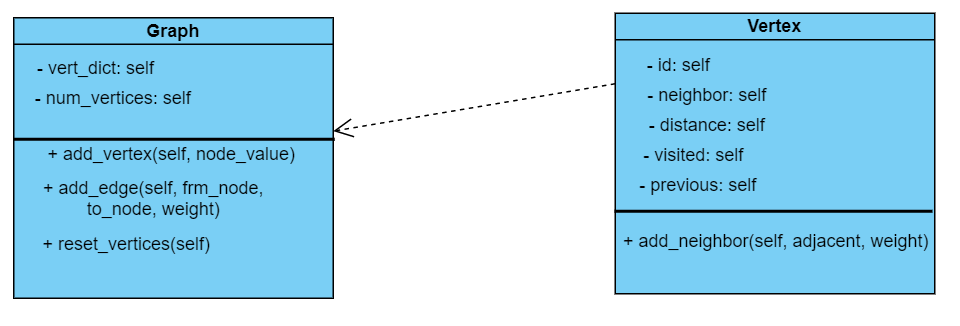


This process will repeat for all requests until none are remaining, at which point the program will print “all requests have been processed” to the console and completed request table is printed on console to clarify the result of request then our code is quit.



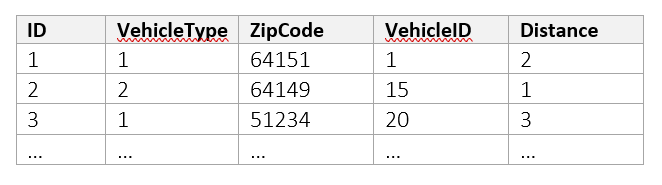
# Diagrams

Classes:

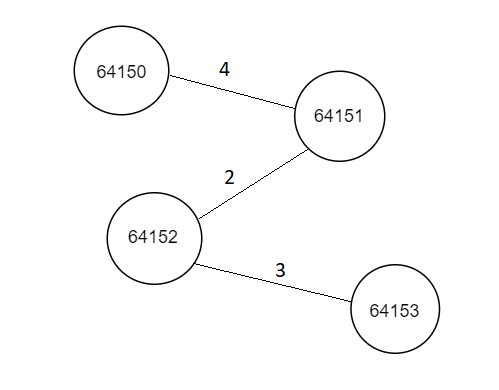


Graph Example:

Taking this table from the project description:



The graph we would create in “main” using the “Graph” class is:



If a request was entered for zip code 64150 and there were two available vehicles at 64152 and 64153, the dijkstra function would find a distance of 6 for the vehicle at zip code 64152, and a distance of 9 for the vehicle at zip code 64153. The program would then dispatch the vehicle at 64152.

# Big-O of Significant Functions

The heapsort function used to sort the list of available vehicles has a time complexity of O(n).

The time complexity of dijkstra for ‘v’ vertices, is O(n\*v2), because there are two nested while-loops, and “heapify” is also used in this function as well.

# References

<https://stackoverflow.com/questions/23110383/how-to-dynamically-build-a-json-object-with-python>

<https://www.youtube.com/watch?v=XB4MIexjvY0> (Dijkstra algorithm)

<https://docs.python.org/3/library/heapq.html> (heapq library)

<https://www.bogotobogo.com/python/python_Dijkstras_Shortest_Path_Algorithm.php> (implement Dijkstra)

<https://stackoverflow.com/questions/53554199/heapq-push-typeerror-not-supported-between-instances> (to implement heapify successfully)

<https://www.datacamp.com/community/tutorials/json-data-python> (how to deal with Json in python)