Emergency Vehicle Dispatching System

A web application that dispatches the nearest emergency vehicle to the desired location based on the type of emergency

Esha Mayuri

16213720

em7gh@mail.umkc.edu

Overview:

Implemented a web application using ASP.net, using which emergency vehicles can be requested. The system would choose closest available vehicle and display the vehicle ID and distance at which it is available. The shortest path calculation is computed using Dijkstra's algorithm.

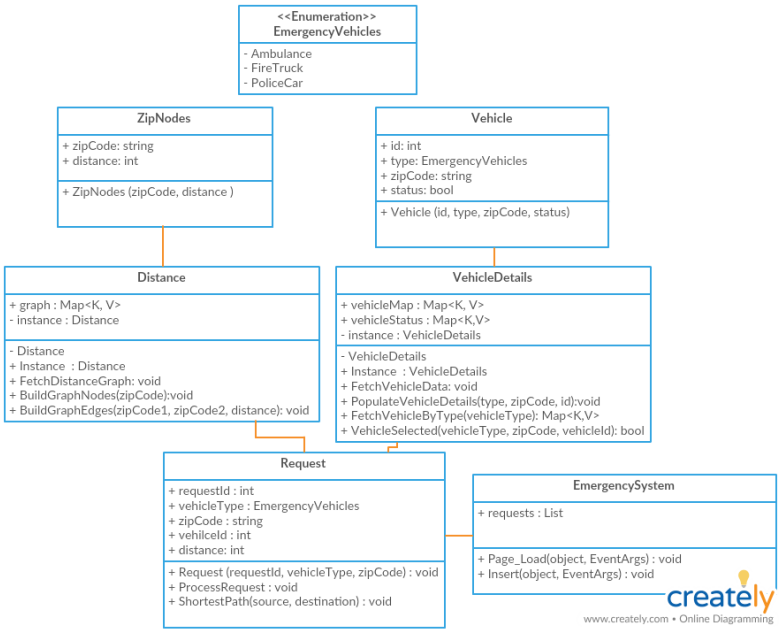
Assumptions:

Following are the assumptions made while trying to calculate the distance:

- All the vehicles travel at same speed irrespective of type of vehicle.
- There is no obstacle in the path of vehicle when it is in motion.
- All the traffic signals in the path will be green while the vehicle is in motion.
- Distance between zip codes is equal to difference between them. The same is computed and stored in data files.
- The distance is zero within a zip code.
- Once the emergency is attended the vehicle will be relocated at same zip code from where it was dispatched.

Illustration Diagram:

Following is the class diagram of the project:



Implementation details:

Data set up:

- The Vehicle information such as vehicle ID, vehicle type and zip code will be provided to project through VehicleDetails.xml file.
- The distance information such as ZipCode1, ZipCode2 and distance will be provided to project through Distance.xml file.

Application flow:

Data Loading:

1. On application startup the vehicle and distance information will be read from the xml files and stored in hash maps data structure for easy access. These two tasks will be executed on two different ***threads*** parallelly to improve performance of the system.

2. Since data instantiation is required once for the lifetime of the application, ***singleton*** concept is used to implement "VehcileDetails.cs" and "Distance.cs" classes. The two singleton instances of class will store the data which can be fetched on demand.

   Time and Space Complexity:

   Complexity of fetching the data from xml file and accessing them on demand.

   Vehicle details:

   Time Complexity:

   O(n) – read elements from xml and store it in hash map

   O(1) – data lookup.

   Space complexity: O(n).

   Note: 'n' is number of elements (i.e. entries) in the VehicleDetails.xml.

   Distance details:

   Time Complexity:

   O(n) – read elements from xml and store it in hash map

   O(1) – data lookup.

   Space complexity: O(n)

   Note: 'n' is number of elements (i.e. entries) in the Distance.xml.

Processing Request:

1. Once the data is loaded, user can access the request page and request emergency vehicle through the user interface provided.
2. User needs to select the vehicle type from the dropdown list provided and enter the zip code of the place where the vehicle is required.
3. On click of 'process request' the application will fetch the details from user interface, add it to a request list and processes the request using the Request class.
4. Request class would accept the request and calculates the shortest path using Dijkstra's algorithm.

   Steps to process request:

   1. If the requested vehicle type is available in same zip code:
      - Checks if the vehicle is available, if so the vehicle details will be fetched.
      - The vehicle is then dispatched, and the details are displayed on the interface.
      - Once the vehicle is dispatched the availability status of the vehicle will be made false.
   2. If the requested vehicle type is not available in the same zip code:
      - Fetch all the zip codes where the requested vehicle type is available and then calculate the shortest path from those zip codes to the zip code where the vehicle is to be dispatched.
      - Fetch the vehicle details with minimum distance and dispatch it.
      - Once the vehicle is dispatched the availability status of the vehicle will be made false.

Additionally, assuming that any type of emergency will be processed in a fixed period of time. On lapse of that period the vehicle will be moved back to the same zip code from where it was dispatched and will be made available for future requests. For testing purposes the time period for attending emergency is taken as 5 minutes.

Dijkstra's Algorithm Implementation:

Data structures used:

Hash sorted set – This is a combination of Hash map and Sorted set (implemented using Red-Black Tree)

It is used to store vertex and distance to it (i.e. edge weight). This will support minimum edge extraction, lookup and adding/updating edge weight when minimum edge weight is encountered.
Time complexity - Add/Insert/Remove: O (log n), Lookup: O (1)
Space complexity: O(n)

Steps to calculate the shortest path:

1. Initialize the distance to source vertex to zero and the same is added to hash map and sorted set.
2. Now we pop out the record with minimum distance from the sorted set and add that record to distance hash map. The popped-out vertex would be considered as the current vertex and distance corresponding to it as current distance.
3. Fetch the adjacent vertices to current vertex and check if it is sorted set
    - If yes, we compute the distance to this vertex from source using current distance and update it in sorted set and hash map if it is less than the value in sorted set.
    - If not, we create a new record of vertex and computed distance.
4. Steps 2 and 3 will be executed until we reach the destination vertex.

These steps will be executed for number of such vehicle types available and then we pick the minimum distance and dispatch that vehicle.

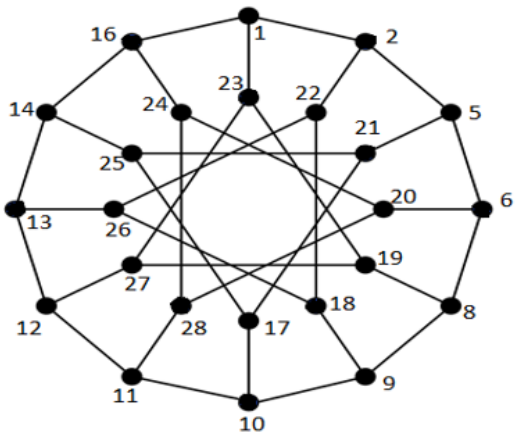
Overall Complexity of Search algorithm:

Time Complexity: O (N * E * log V)
Space Complexity: O(E+V)
where N – Number of vehicles available with the requested vehicle type
E – Number of paths connected to zip code
V – Number of zip codes

Tradeoff:

Currently as most computers have large amount of space and time efficiency is the biggest criteria. So, I have preferred space-time tradeoff to increase the time efficiency of system by using lookup tables. Using which data can be accessed quickly.

Pictorial representation of data used:

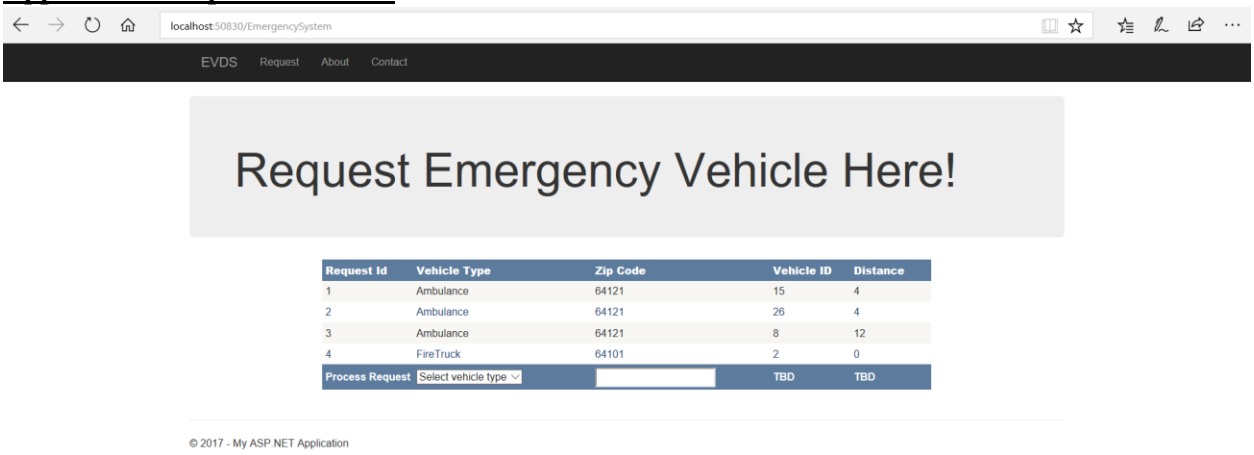


Total nodes(Zip codes) – 24
Total vehicles - 31

Zip code	Vehicle Type	Vehicle Id's
64101	1,2	1,2,31
64102	3	3
64105	2	4
64106	1,3	5,6
64108	2	7
64109	1	8
64110	3	9
64111	3	10
64112	2	11
64113	1	12
64114	3	13
64116	2	14
64117	1	15
64118	2,3	16,17
64119	1,2	18,19
64120	2	20
64121	3	21
64122	2	22
64123	1,2	23,24
64124	1	25
64125	1	26
64126	3	27
64127	3	28
64128	2,3	29,30

This is the data set used in the project where nodes 1, 2, 5 … represents the zip code 64101, 64102, 64105 … respectively. The table displays zip codes, vehicle type and vehicle Id's available at each zip code.
Example:
2 Ambulances are available at 64101 and one Ambulance is at 64106 and so on.
Fire truck is at 64101, 64105 and so on.
Police car is at 64102, 64106 and so on.

Application sample screenshot:



In the above screenshot there are four requests made:

1. Request for Ambulance was made from 64121. As 64121 has no ambulances in its zip code. The distance of the nearest ambulance will be computed and displayed along with vehicle id.
2. Now the request for 2nd ambulance is made from the same zip code, as vehicle id 15 was already dispatched. Now the next nearest vehicle would be fetched and displayed.
3. Similarly request for third Ambulance from 64121 will be processed and different vehicle will be dispatched.
4. Here as the request was made from 64101 for Fire Truck and as this vehicle is available in the same zip code. It will be processed and distance would now be displayed as zero, as requested vehicle at the same zip code.

Possible Enhancements:

Data stored in xml files can be moved to rational database.
Additional tab in the user interface can be added to add/update vehicles and distance data.

Coding Standards:

Followed conventional .Net coding standards
PascalCasing – Namespace, Class names, Method name, Properties, Events
CamelCasing – Parameters

GitHub link:

https://github.com/emayuri/Algorithms

References:

https://msdn.microsoft.com/en-us
http://www.geeksforgeeks.org/
https://stackoverflow.com/