

Emergency Vehicle Dispatching System



Design and Analysis of Algorithms

Final Project Submission report

Team Members

Sridevi Mallipudi (16251191)

Pragathi Thammaneni (16230695)

Arun Kumar Anthati (16232818)

Emergency Vehicle Dispatching System

Introduction:

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks.

Dijkstra's original variant found the shortest path between two nodes but a more common variant fix a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.

- ✓ It can be either directed or undirected.
- ✓ It contains all the weighted edges.
- ✓ It is a connected graph

Pseudo Code:

Initialize the cost of each node to ∞

Initialize the cost of the source to 0

While there are unknown nodes left in the graph

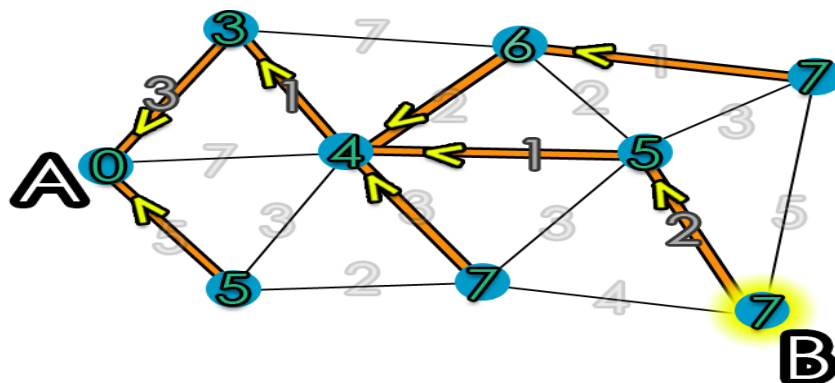
 Select the unknown node with the lowest cost: n

 Mark n as known

 For each node a which is adjacent to n

a 's cost = $\min(a$'s old cost, n 's cost + cost of (n, a))

For example, consider source node as A and destination node as B the shortest path from A to B can be calculated as shown below:



Motivation:

The main aim of the project is to design an emergency vehicle dispatching system using an algorithm. The algorithm should find the shortest distance between the nodes and allocate vehicles by processing the requests one by one. Given that there are three different types of emergency vehicles:

1. Ambulance
2. Fire Truck
3. Police car

Also assume that every request only needs one emergency vehicle.

Graph Construction:

We considered zip codes as nodes and distance between the zip codes as weights. And implemented the solution in python with json data.

Sample data:

Request data

```
"requests":  
[ {  
    "id": 1,  
    "vehicle_type": 2,  
    "zipcode": 64167,  
    "vehicle_id": null,  
    "distance": 0  
  },  
  ]
```

Vehicles data

```
"vehicles":  
[ {  
    "id": 1,  
    "type": 1,  
    "zipcode": 64149  
  },  
  ]
```

Distance between nodes

```
"distances":  
[ {  
    "zipcode1": 64149,  
    "zipcode2": 64150,  
    "distance": 4  
},
```

Attributes considered:

Zip codes as nodes

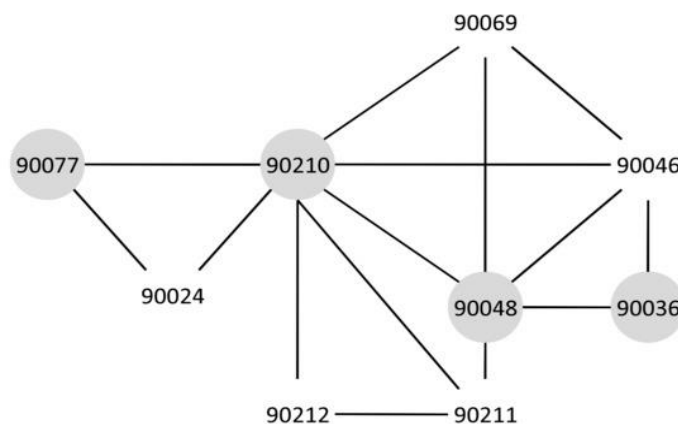
Distances as the edges weight with connected vertices

Vehicle type: among the above three types

For instance find the distance between u and v using Dijkstra's Algorithm?

Solution: -

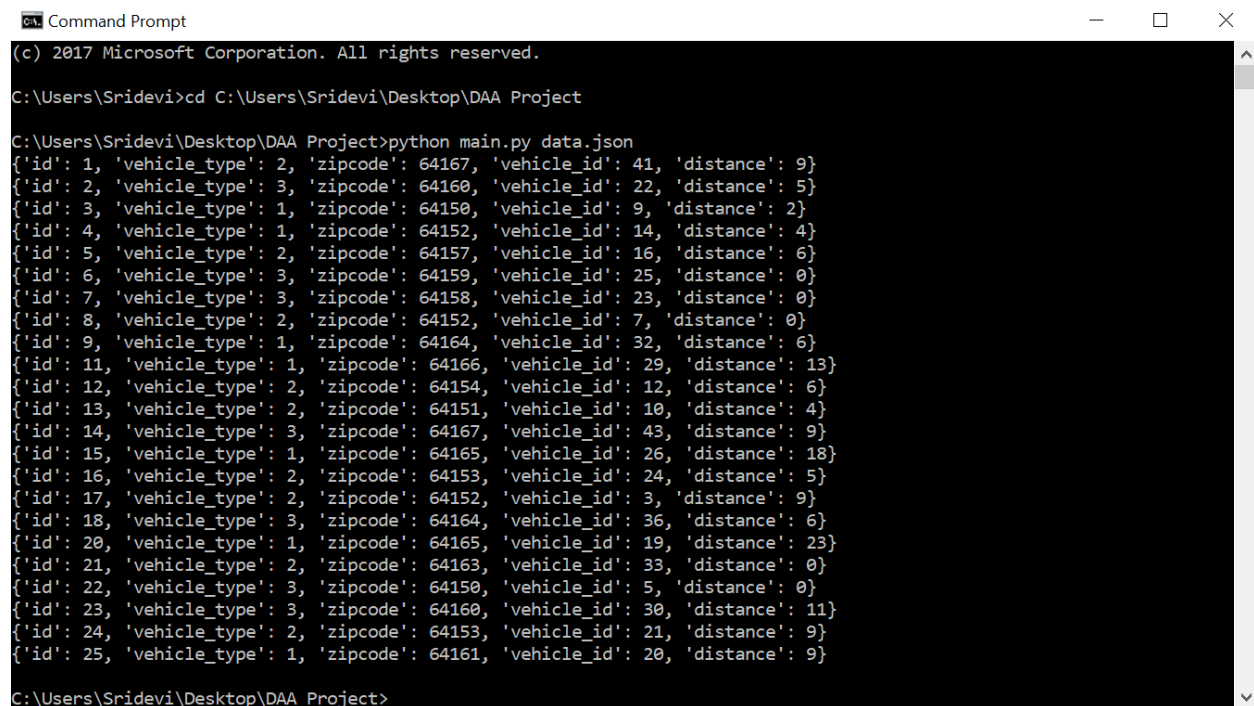
- 1) Create a set sptSet (shortest path tree set) that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty.
- 2) Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.
- 3) While sptSet doesn't include all vertices
 - a) Pick a vertex u which is not there in sptSet and has minimum distance value.
 - b) Include u to sptSet.
 - c) Update distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.



Output:

Algorithms takes each request first come first serve and find the nearest node and check for availability and move to the next nearest node till it finds the node with requested vehicle availability.

Screenshot:



```
Command Prompt
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Sridevi>cd C:\Users\Sridevi\Desktop\DAA Project

C:\Users\Sridevi\Desktop\DAA Project>python main.py data.json
{'id': 1, 'vehicle_type': 2, 'zipcode': 64167, 'vehicle_id': 41, 'distance': 9}
{'id': 2, 'vehicle_type': 3, 'zipcode': 64160, 'vehicle_id': 22, 'distance': 5}
{'id': 3, 'vehicle_type': 1, 'zipcode': 64150, 'vehicle_id': 9, 'distance': 2}
{'id': 4, 'vehicle_type': 1, 'zipcode': 64152, 'vehicle_id': 14, 'distance': 4}
{'id': 5, 'vehicle_type': 2, 'zipcode': 64157, 'vehicle_id': 16, 'distance': 6}
{'id': 6, 'vehicle_type': 3, 'zipcode': 64159, 'vehicle_id': 25, 'distance': 0}
{'id': 7, 'vehicle_type': 3, 'zipcode': 64158, 'vehicle_id': 23, 'distance': 0}
{'id': 8, 'vehicle_type': 2, 'zipcode': 64152, 'vehicle_id': 7, 'distance': 0}
{'id': 9, 'vehicle_type': 1, 'zipcode': 64164, 'vehicle_id': 32, 'distance': 6}
{'id': 11, 'vehicle_type': 1, 'zipcode': 64166, 'vehicle_id': 29, 'distance': 13}
{'id': 12, 'vehicle_type': 2, 'zipcode': 64154, 'vehicle_id': 12, 'distance': 6}
{'id': 13, 'vehicle_type': 2, 'zipcode': 64151, 'vehicle_id': 10, 'distance': 4}
{'id': 14, 'vehicle_type': 3, 'zipcode': 64167, 'vehicle_id': 43, 'distance': 9}
{'id': 15, 'vehicle_type': 1, 'zipcode': 64165, 'vehicle_id': 26, 'distance': 18}
{'id': 16, 'vehicle_type': 2, 'zipcode': 64153, 'vehicle_id': 24, 'distance': 5}
{'id': 17, 'vehicle_type': 2, 'zipcode': 64152, 'vehicle_id': 3, 'distance': 9}
{'id': 18, 'vehicle_type': 3, 'zipcode': 64164, 'vehicle_id': 36, 'distance': 6}
{'id': 20, 'vehicle_type': 1, 'zipcode': 64165, 'vehicle_id': 19, 'distance': 23}
{'id': 21, 'vehicle_type': 2, 'zipcode': 64163, 'vehicle_id': 33, 'distance': 0}
{'id': 22, 'vehicle_type': 3, 'zipcode': 64150, 'vehicle_id': 5, 'distance': 0}
{'id': 23, 'vehicle_type': 3, 'zipcode': 64160, 'vehicle_id': 30, 'distance': 11}
{'id': 24, 'vehicle_type': 2, 'zipcode': 64153, 'vehicle_id': 21, 'distance': 9}
{'id': 25, 'vehicle_type': 1, 'zipcode': 64161, 'vehicle_id': 20, 'distance': 9}

C:\Users\Sridevi\Desktop\DAA Project>
```

Conclusion:

Dijkstra's algorithm finds the shortest path between two nodes. Here we found distance between the nodes from which the request is raised to all other nodes. Found shortest distance to all other nodes. Then we sorted the nodes in increasing order of distance and check node with availability. The first node with the availability (requested vehicle) is the dispatcher node.

Time complexity: assuming number of requests as n and time complexity of dijkstra's being v^2 where v is number of vertices. The time complexity of our algorithm is $O(n \cdot v^2)$

Code:

<https://github.com/PragathiThammaneni/Design-And-Analysis-of-Algorithms/tree/master/Final%20Project/Code>

References:

<http://www.geeksforgeeks.org/greedy-algorithms-set-6-dijkstras-short-est-path-algorithm/>