



Design & Analysis of Algorithms

CS5592 SPRING2018

PROJECT REPORT

Emergency Vehicle Dispatch System

Instructor: M. Amin Kuhail, PhD.

Email: kuhailm@umkc.edu

Department of Computer Science & Electrical Engineering

University of Missouri, Kansas City

TEAM MEMBERS:

MAHESH JAMALLAMUDI (16234558)

GitHub URL:

https://github.com/mahesh9595/DA_Final_project

PROJECT DESCRIPTION

Project 2 – Emergency Vehicle Dispatching System

- We are trying to design an emergency vehicle dispatching system. Let's assume that there are three different types of emergency vehicles: 1 (Ambulance), 2 (Fire Truck), 3 (Police car).
- Let's also assume that every request only needs one emergency vehicle.
- Here's an example of the table of EmergencyVehicles:

ID	Type	ZipCode
1	1	64151
2	1	64151
3	1	64151
4	2	64151
5	3	64151
6	3	64151
7	3	64149
...

(e.g. from the table above, we can see there are 3 ambulances, 1 fire truck, and 1 police car at 64151.

- Here's an example of the table Request

ID	VehicleType	ZipCode	VehicleID
1	1	64151	?
2	2	64149	?
3	1	51234	?
...	?

(e.g. from the table above, we can see there is a request for an ambulance at zip code 64151. The question mark means we still haven't assigned a vehicle for the request.

- Figure 3 shows the table distance, which consists of information telling us how far apart are neighboring zip codes.

ZipCode1	ZipCode2	Distance
64150	64151	4
64151	64152	2
64152	64153	3
...

(e.g. from the table above, we can see that the zip code 64150 is 4 miles away from 64151).

Note: the distance between zip codes that are not neighboring is not readily available.

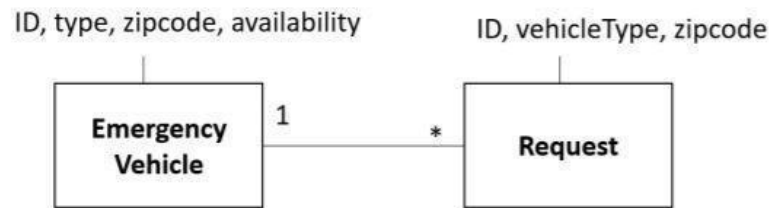


Figure 2

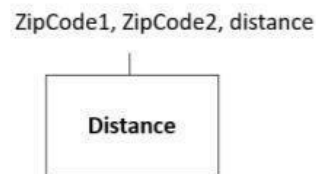


Figure 3

Technical Requirements

- Implement an algorithm that processes requests one by one. For each request, the algorithm should try to find the closest available emergency vehicle
- Implement the algorithm in a language you are comfortable with.
- Make up the data for the project. It should be similar to the examples I gave. You can generate the files at run time, or have it stored in files, and you read it at run time.
- The algorithm should produce the vehicle matching, e.g.:

ID	VehicleType	ZipCode	VehicleID	Distance
1	1	64151	1	2
2	2	64149	15	1
3	1	51234	20	3
...

PROJECT APPROACH

Language Used: JAVA

Software: ECLIPSE JAVA OXYGEN

Operating System: Windows

Briefing:

- Main objective of the project is to dispatch an Emergency Vehicle to the required area.
- From the given data we can say that Vehicle type -1 is for Ambulance, 2 is for Fire Truck, and 3 for Police Car.
- So, the data of Vehicles like how many vehicles and what vehicles are present in respective areas are manually inserted using JTables.
- We also make some distance assumptions between the zip codes we have selected, and these are also manually inserted in tables using JTables.
- If there is no vehicle which is requested in respective area, then one has to dispatch from another location which is nearer to requested location.
- For this purpose, we use Dijkstra's algorithm to find the minimum distance.
- After calculating the minimum distance, a vehicle is dispatched from the near location to the requested location based on the availability.

The order I followed in the project is

1. CREATING TABLES:

As mentioned above I created a table for vehicle data which consists of vehicle data like vehicle ID, type and respective zip codes and another table for distances between the zip codes.

Both the table data is based on the assumption of distances, no. of vehicles, etc....

The screenshot shows the Eclipse IDE with the 'Vehicle.java' file open. The code defines a class 'Vehicle' that extends 'JFrame'. It includes a table declaration with three columns: 'vehicle ID', 'Type', and 'zipcode'. The 'vehicle_details' array contains 21 rows of data. The 'main' method creates a new 'Vehicle' object and calls 'setVisible(true)'. A preview window of the table is shown on the right.

```
1 import javax.swing.JFrame;
2
3
4
5 public class Vehicle extends JFrame {
6
7     JTable jt;
8     //table declarations
9     String [] column_headers= {"vehicle ID", "Type", "zipcode"};
10    String [][] vehicle_details = {{ "1", "1", "64151"}, {"2", "1", "64151"}, {"3", "1", "64151"}, {"4", "2", "64151"}, {"5", "3", "64151"}, {"6", "3", "64151"}, {"7", "1", "64149"}, {"8", "2", "64149"}, {"9", "1", "64110"}, {"10", "1", "64110"}, {"11", "2", "64110"}, {"12", "3", "64110"}, {"13", "1", "64109"}, {"14", "2", "64109"}, {"15", "2", "64109"}, {"16", "2", "64109"}, {"17", "3", "64109"}, {"18", "3", "64109"}, {"19", "1", "66109"}, {"20", "2", "66109"}, {"21", "3", "66109"};
22
23    public Vehicle()
24    {
25        //Visualization of table
26        jt = new JTable(vehicle_details, column_headers);
27        jt.setBounds(100,200,300,400);
28        JScrollPane js = new JScrollPane(jt);
29        this.add(js);
30        this.setSize(400,500);
31        this.setVisible(true);
32    }
33
34    public static void main(String[] args) {
35        // TODO Auto-generated method stub
36        new Vehicle();
37    }
38 }
```

vehicle ID	Type	zipcode
1	1	64151
2	1	64151
3	1	64151
4	2	64151
5	3	64151
6	3	64151
7	1	64149
8	2	64149
9	1	64110
10	1	64110
11	2	64110
12	3	64110
13	1	64109
14	2	64109
15	2	64109
16	2	64109
17	3	64109
18	3	64109
19	1	66109
20	2	66109
21	3	66109

Example:

In zip code 64151 we have 3 Ambulances, 1 Fire Truck, and 2 Police cars.

Table 2 is created with the same logic as table one implementing JTable. In this table, we have distances between two zip codes.

Example:

Distance between zipcode1 (64151) and zipcode2 (64149) is 10.

The screenshot shows the Eclipse IDE interface. The Package Explorer on the left lists the project structure: Emergency Vehicle Dispatch, JRE System Library [JavaSE-10], and src (default package). The src package contains files: Availability.java, DijkstraAlgorithmSet.java, Distance.java, Main.java, Request.java, TestTableSortFilter.java, and Vehicle.java. The main editor displays the code for Distance.java, which includes a JTable with the following data:

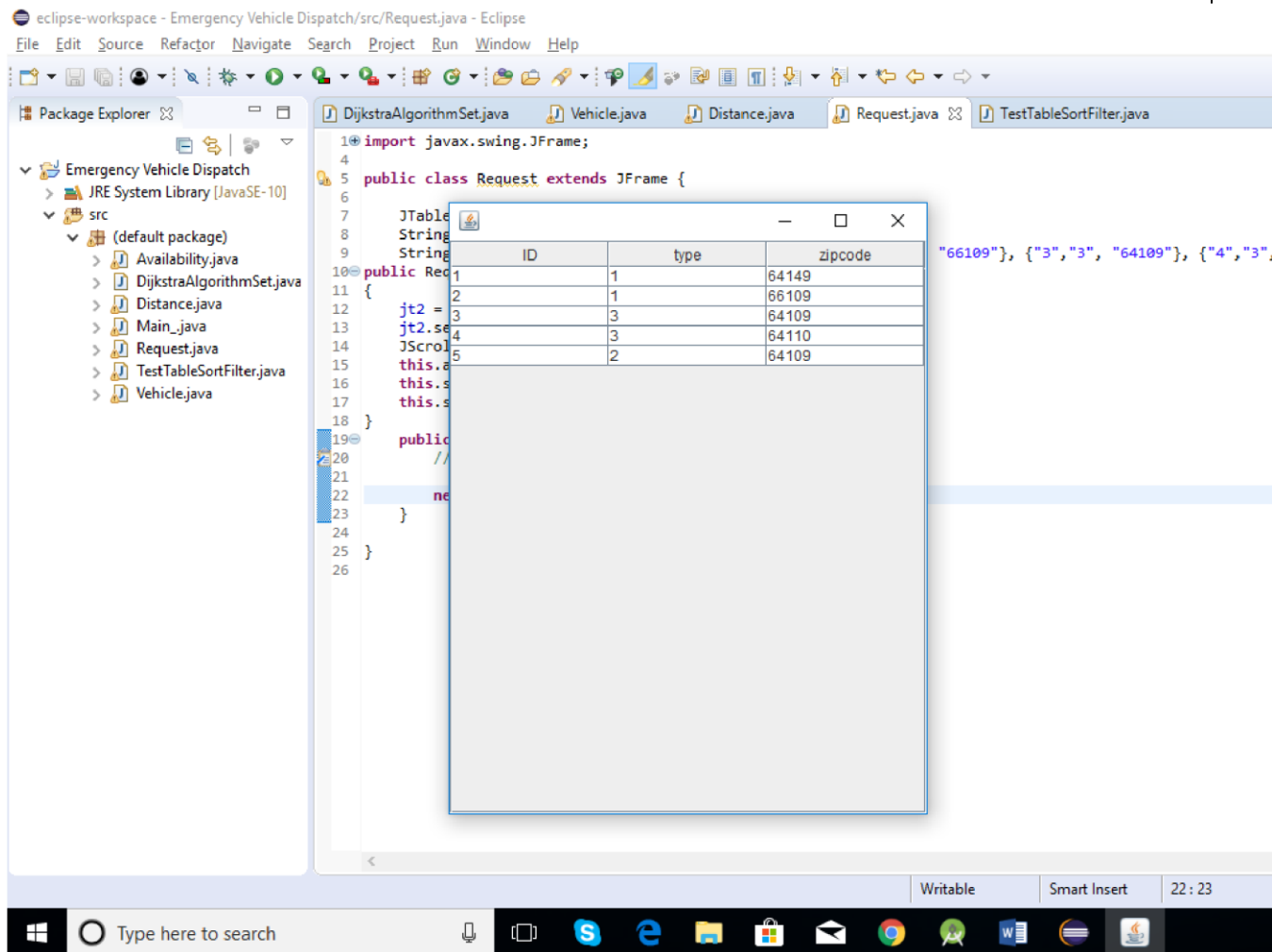
zipcode-1	zipcode-2	distance
64151	64149	10
64151	66109	120
64149	64109	20
64149	64110	50
64110	64109	5
64109	66109	100
64109	64151	30
66109	64149	90

The JTable window is titled 'Distance.java' and has a scroll bar on the right. The code in the background shows the initialization of the table and the implementation of the Dijkstra algorithm.

2. Request table:

We have another table for the requests in which we can say that a vehicle is needed for a zip code.

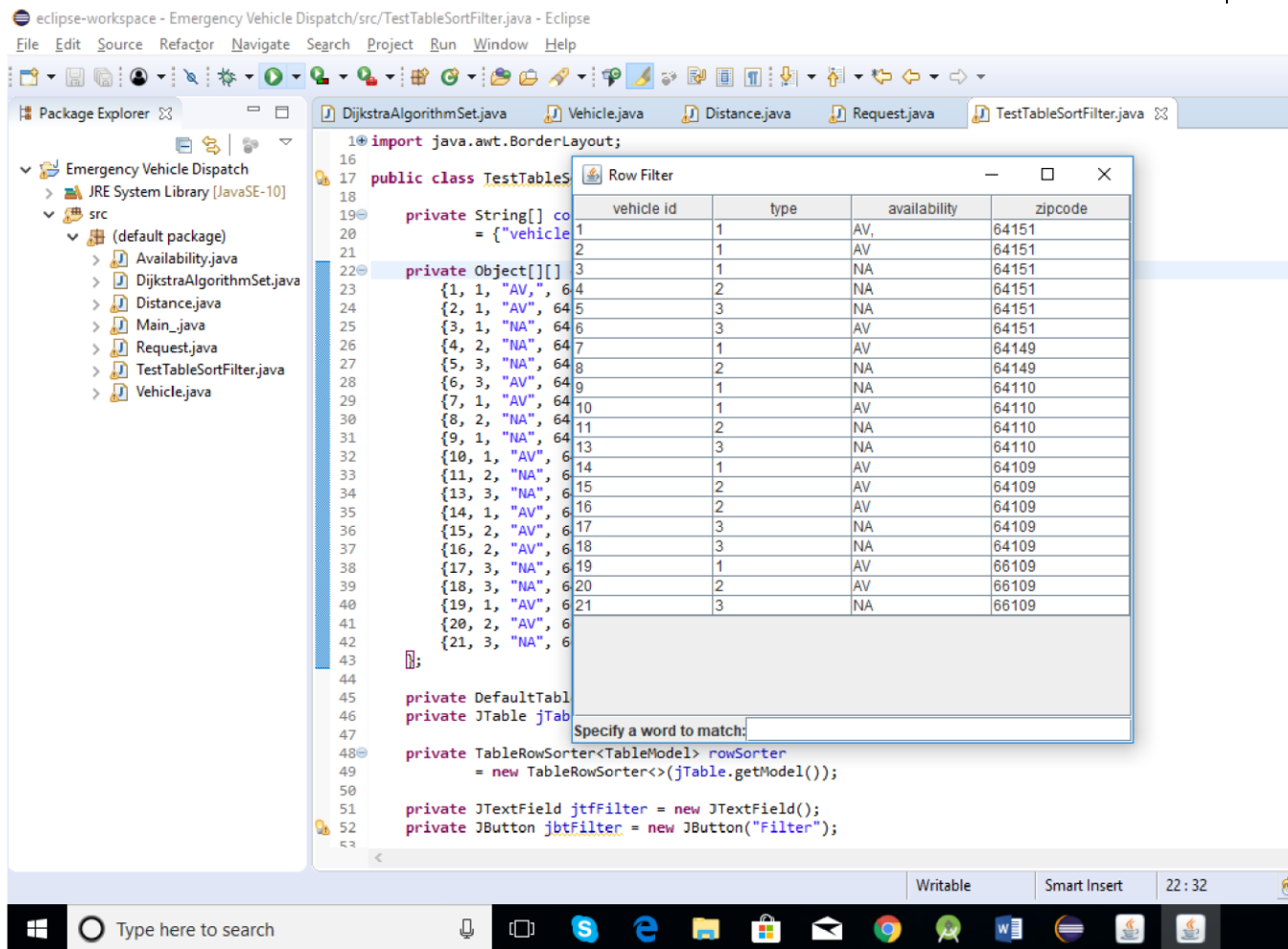
For example, there is requirement of type-1 vehicle at 64149 areas.



3. Availability check:

To dispatch an emergency vehicle to a location first one has to check for it's availability. So, I created a table in which we can check for the available vehicles using a search bar.

I assumed of AV for availability and NA for not available.



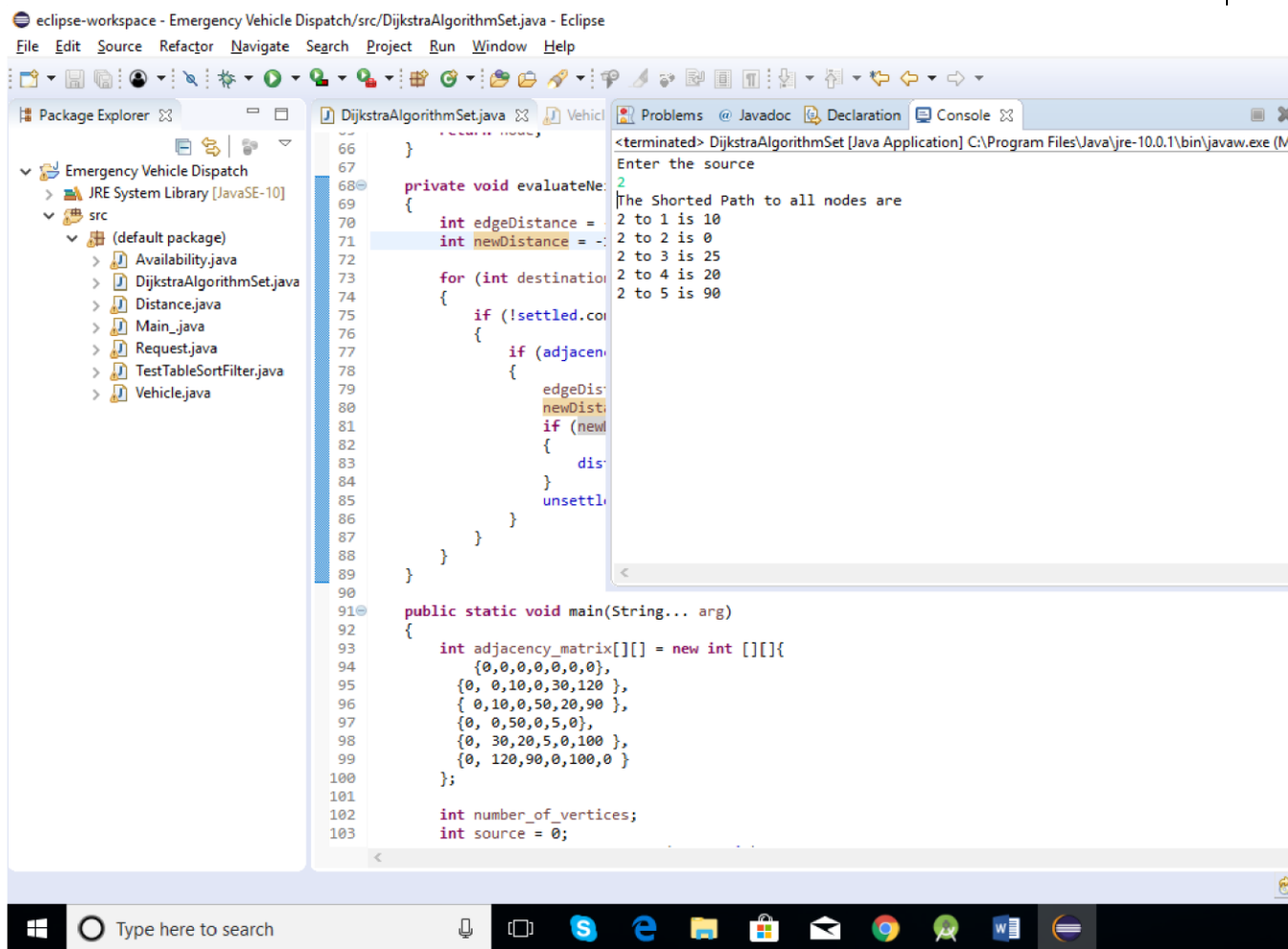
If we search for AV in search bar its will show the list of availability vehicles.

4. FINDING SHORTEST PATHS

If there is no available vehicle for the request in respective location, we need to find a location where vehicle is available which is nearer to the request location.

64149 as 2,
64110 as 3,
64109 as 4, 66109
as 5.

So, after we find the shortest path we take the nearest location and then check for availability from availability table, if we have a vehicle then we can



```
eclipse-workspace - Emergency Vehicle Dispatch/src/DijkstraAlgorithmSet.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer
Emergency Vehicle Dispatch
  JRE System Library [JavaSE-10]
  src
    (default package)
      Availability.java
      DijkstraAlgorithmSet.java
      Distance.java
      Main.java
      Request.java
      TestTableSortFilter.java
      Vehicle.java

DijkstraAlgorithmSet.java
66 }
67
68 private void evaluateNearest() {
69     int edgeDistance = -1;
70     int newDistance = -1;
71
72     for (int destination = 1; destination < number_of_vertices; destination++) {
73         if (!settled[destination]) {
74             if (adjacency_matrix[source][destination] < edgeDistance) {
75                 edgeDistance = adjacency_matrix[source][destination];
76                 newDistance = edgeDistance;
77                 if (newDistance < distance[destination]) {
78                     distance[destination] = newDistance;
79                     unsettled[destination] = true;
80                 }
81             }
82         }
83     }
84 }
85
86
87 }
88
89 }
90
91 public static void main(String... arg)
92 {
93     int adjacency_matrix[][] = new int [][]{
94         {0,0,0,0,0,0},
95         {0, 0,10,0,30,120},
96         {0, 10,0,50,20,90},
97         {0, 0,50,0,5,0},
98         {0, 30,20,5,0,100},
99         {0, 120,90,0,100,0}
100     };
101
102     int number_of_vertices = 5;
103     int source = 0;
```

```
<terminated> DijkstraAlgorithmSet [Java Application] C:\Program Files\Java\jre-10.0.1\bin\javaw.exe (M
Enter the source
2
The Shorted Path to all nodes are
2 to 1 is 10
2 to 2 is 0
2 to 3 is 25
2 to 4 is 20
2 to 5 is 90
```

dispatch that vehicle else we find another nearest location and again check for availability and will dispatch the vehicle.

eclipse-workspace - Emergency Vehicle Dispatch/src/TestTableSortFilter.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- Emergency Vehicle Dispatch
 - JRE System Library [JavaSE-10]
 - src
 - (default package)
 - Availability.java
 - DijkstraAlgorithmSet.java
 - Distance.java
 - Main.java
 - Request.java
 - TestTableSortFilter.java
 - Vehicle.java

DijkstraAlgorithmSet.java Vehicle.java Distance.java Request.java TestTableSortFilter.java

```
1 import java.awt.BorderLayout;
16
17 public class TestTableSortFilter {
18
19     private String[] columnNames = {"vehicle id", "type", "availability", "zipcode"};
20
21     private Object[][] data = {
22         {1, 1, "AV", 64151},
23         {2, 1, "AV", 64151},
24         {3, 1, "NA", 64151},
25         {4, 2, "NA", 64151},
26         {5, 3, "NA", 64151},
27         {6, 3, "AV", 64151},
28         {7, 1, "AV", 64151},
29         {8, 2, "NA", 64151},
30         {9, 1, "NA", 64151},
31         {10, 1, "AV", 64151},
32         {11, 2, "NA", 64151},
33         {12, 3, "NA", 64151},
34         {13, 3, "NA", 64151},
35         {14, 1, "AV", 64151},
36         {15, 2, "AV", 64151},
37         {16, 2, "AV", 64151},
38         {17, 3, "NA", 64151},
39         {18, 3, "NA", 64151},
40         {19, 1, "AV", 64151},
41         {20, 2, "AV", 64151},
42         {21, 3, "NA", 64151}
43     };
44
45     private DefaultTableModel dtm;
46     private JTable jTable;
47
48     private TableRowSorter<TableModel> rowSorter
49         = new TableRowSorter<>(jTable.getModel());
50
51     private JTextField jtfFilter = new JTextField();
52     private JButton jbtFilter = new JButton("Filter");
53 }
```

Row Filter

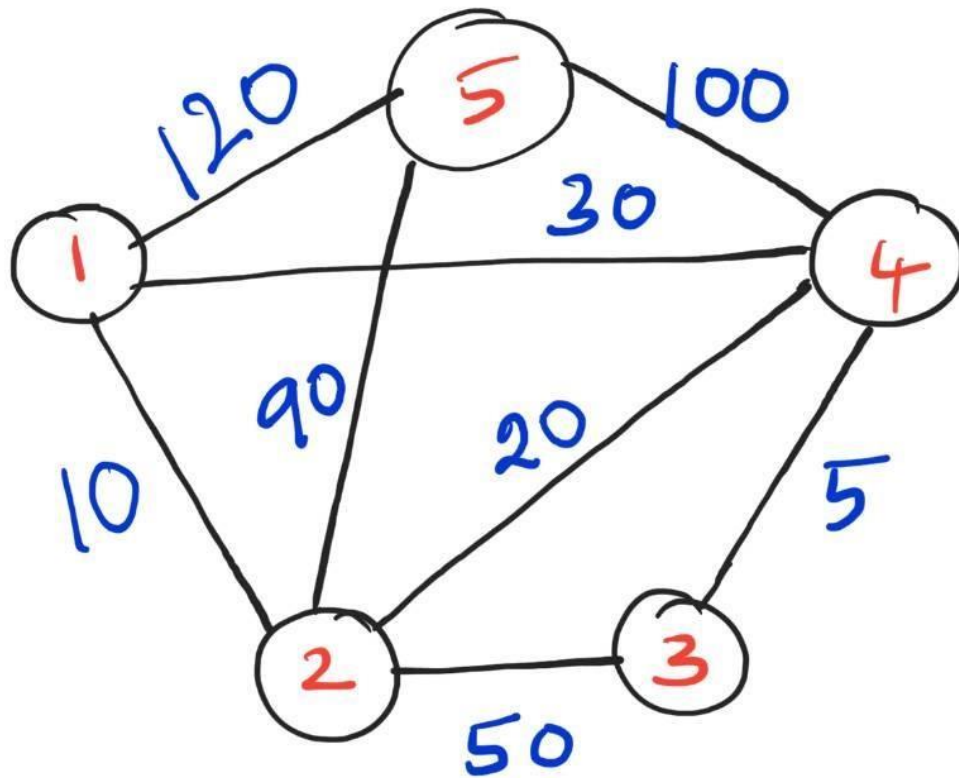
vehicle id	type	availability	zipcode
1	1	AV	64151
2	1	AV	64151
3	1	NA	64151
4	2	NA	64151
5	3	NA	64151
6	3	AV	64151

Specify a word to match: 64151

Writable Smart Insert 22:32

Type here to search

Model Diagram:



Last modified: 10:02 PM

We have used DIJKSTRA's Algorithm which gives the efficient time complexity of $O(E \log V)$ where E – No. of Edges

V – No. of Vertices

REFERENCES

<https://www.youtube.com/watch?v=EbL1pj3tOgQ> <https://www.sanfoundry.com/java-program-implement-dijkstras-algorithm-using-queue/>

<https://stackoverflow.com/questions/22066387/how-to-search-an-element-in-a-jtable-java>

<https://stackoverflow.com/questions/1107911/how-can-i-filter-rows-in-a-jtable>

<https://stackoverflow.com/questions/3179136/jtable-how-to-refresh-table-model-afterinsert-delete-or-update-the-data>