



**DESIGN AND ANALYSIS OF ALGORITHMS (CS 5592)**

**PROJECT REPORT**

**“EMERGENCY VEHICLE DISPATCHING SYSTEM”**

**Team Members**

**Under the guidance Of-**

**Mohammed Amin Kuhail**

**Harmeet Singh (16233090)**

**Nitin Bhandari (16234365)**

**Rajarshi Tarafdar (16232740)**

**Ritika Chowdri (16230840)**

**Sri Sai Anusha Gandu (16230560)**

## **Problem Statement**

Here the problem statement states that we have to create an “**Emergency Vehicle Dispatching System**” which can be used to keep the track of emergency vehicle at each zip code and will be helpful in case of emergency dispatching. Emergency vehicles include basically three types of vehicles which are Type1-Ambulance, Type 2-Fire Truck, Type 3-Police. This system can be very helpful and used as a real time application system.

## **Approach to the above Problem Statement**

As per the problem statement the system which we should design should be real time and can be implemented in real time. So our approach to solve this problem statement was decided after many brain storming sessions so that it could really be implemented efficiently.

Here we tried to implement a system with each zip codes having an availability of all types of vehicle with the initial count of 3 at each system. So let us say if we request for a particular type of vehicle from a specific zip code. The system will check the availability of that vehicle at that zip code by checking the no of count and the fetching the vehicle. If the vehicle is not present at that particular node then the system will fetch the nearest neighbor and check the availability of that vehicle at those zip codes.

For this system we used an approach of graphs to solve this. Every zip code was considered as a node and distance between each node can be fetched from tables in the database. We have used the SQL Queries to fetch data from our SQL database. This is the brief description of approach we used to solve the above problem statement. The implementation in details has been discussed in implementation section of this report.

## **Assumptions**

- Every node is assumed to have a maximum of 3 neighbours
- When a zip code is selected and if it doesn't have the resources when requested, it goes to the neighbour closest in distance
- The selection of the neighbour is done based on shortest path distance

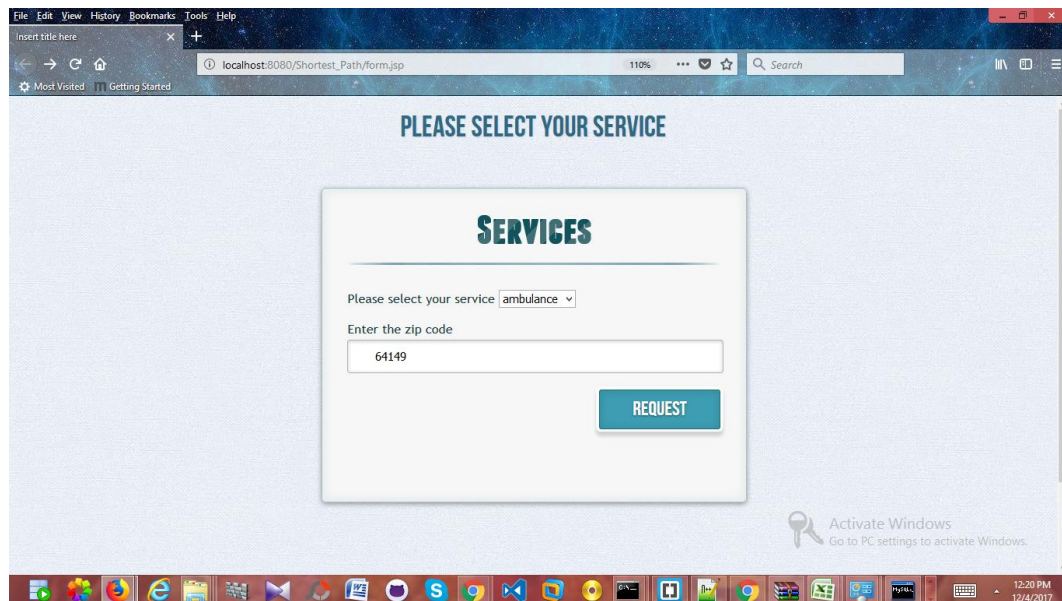
## **Implementation**

### **Languages and Platforms**

- Java
- SQL
- JSP
- CSS
- Eclipse IDE

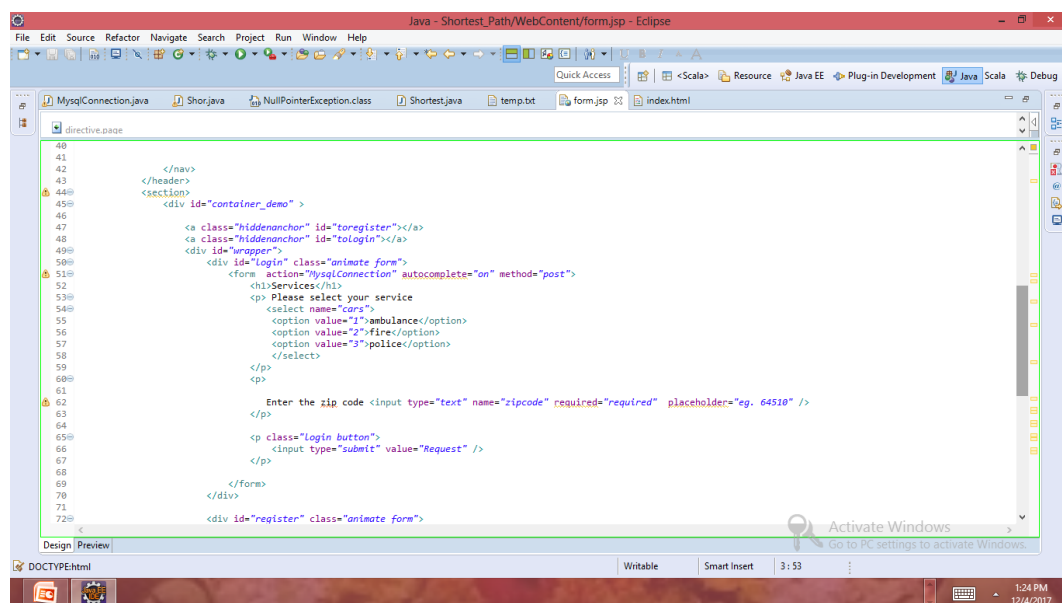
The number of available resources at a given zip code is stored in the database.

The code is run on the server which displays a web page where the selection for the service is done. Once the service is selected and the request is made for a zip code, if resources are available at the given zip code, the number of available resources reduces by one in the database. If the given zip code has no available resources, it checks for the nearest neighboring zip codes which holds the resources. This is done using SQL.



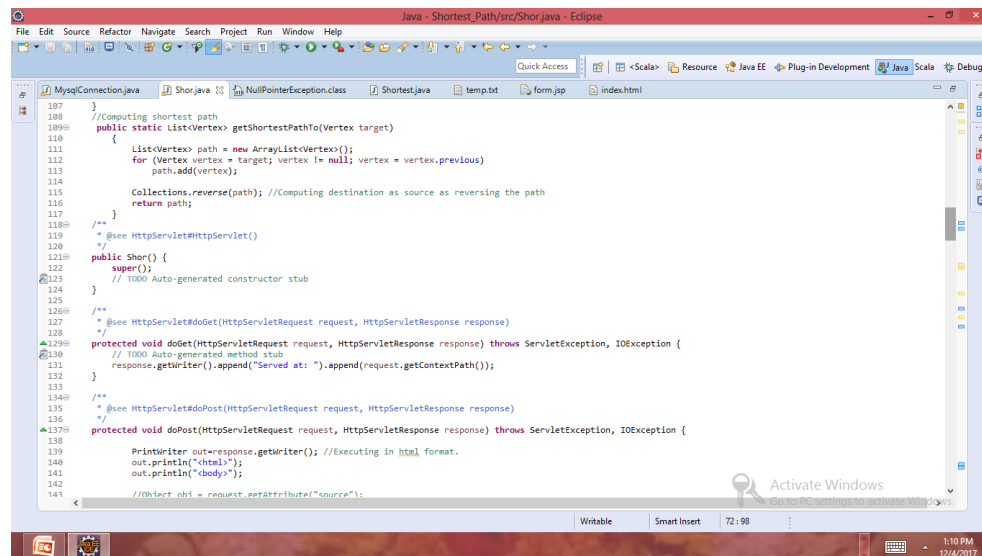
The code contains two main functions

- Compute / Min. Distance
- Return Shortest Distance



## Compute / Min. Distance

If a zip code contains 2 neighbours, the code runs compute function. It computes the length of the paths to the available neighbours simply compares them and returns the neighbour with the shortest path. If a zip code has 3 neighbours, the code runs min. distance and returns the zip code with the minimum distance. This function contains one recursive function.

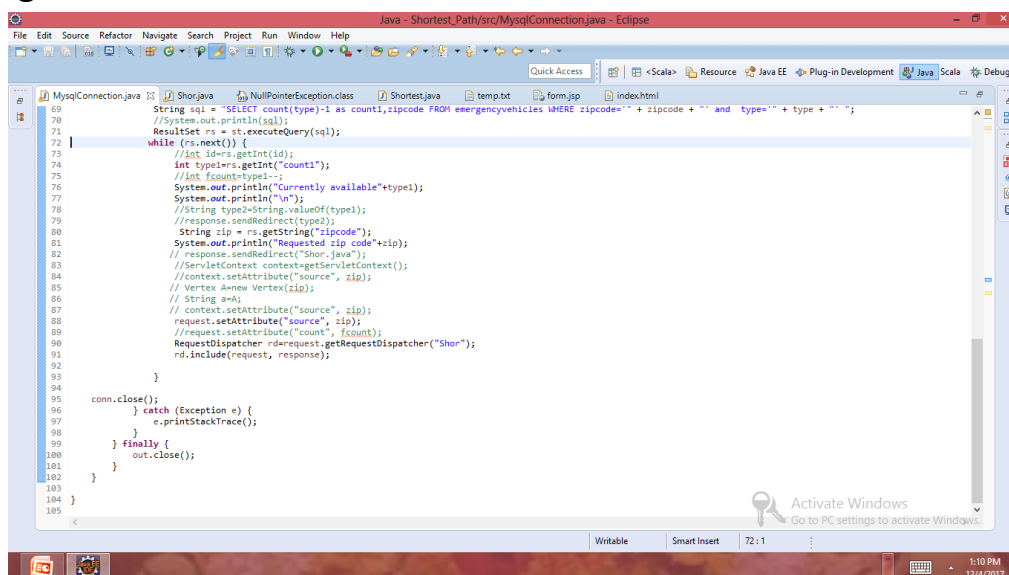


```
107 }
108 //Computing shortest path
109 public static List<Vertex> getShortestPathTo(Vertex target)
110 {
111     List<Vertex> path = new ArrayList<Vertex>();
112     for (Vertex vertex = target; vertex != null; vertex = vertex.previous)
113         path.add(vertex);
114     Collections.reverse(path); //Computing destination as source as reversing the path
115     return path;
116 }
117
118 /**
119  * @see HttpServlet#HttpServlet()
120  */
121 public Shor() {
122     super();
123     // TODO Auto-generated constructor stub
124 }
125
126 /**
127  * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
128  */
129 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
130     // TODO Auto-generated method stub
131     response.getWriter().append("Served at: ").append(request.getContextPath());
132 }
133
134 /**
135  * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
136  */
137 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
138     PrintWriter out=response.getWriter(); //Executing in html format.
139     out.println("<html>");
140     out.println("<body>");
141     //Object obj = request.getAttribute("source");
142 }
```

## Return Shortest Distance

This function returns the path between requested zip code and the shortest length zip code with available resources and prints the path. This function contains one recursive function.

## SQL Integration



```
69 String sql = "SELECT count(type)-1 as count1,zipcode FROM emergencyvehicles WHERE zipcode='"+ zipcode + "' and type='"+ type + "' ";
70 //System.out.println(sql);
71 ResultSet rs = st.executeQuery(sql);
72 while (rs.next()) {
73     //int id=rs.getInt(id);
74     int type=rs.getInt("count1");
75     //int fcount=type-1;
76     System.out.println("Currently available "+type1);
77     System.out.println("\n");
78     //String type=String.valueOf(type1);
79     //response.sendRedirect(type2);
80     String zip = rs.getString("zipcode");
81     System.out.println("Requested zip code "+zip);
82     // response.sendRedirect("Shor.java");
83     //ServletContext context=getServletContext();
84     //context.setAttribute("source", zip);
85     // Vertex a=new Vertex(zip);
86     // String a=a;
87     // context.setAttribute("source", zip);
88     request.setAttribute("source", zip);
89     //request.setAttribute("count", fcount);
90     RequestDispatcher rd=request.getRequestDispatcher("Shor");
91     rd.include(request, response);
92 }
93
94 conn.close();
95 } catch (Exception e) {
96     e.printStackTrace();
97 }
98 } finally {
99     out.close();
100 }
101 }
102 }
103 }
104 }
105 }
```

```

mysql> clear
mysql> show tables;
+-----+
| Tables_in_vsystem |
+-----+
| distance            |
| emergencyvehicles    |
| request             |
| vcount              |
+-----+
4 rows in set (0.02 sec)

mysql> select * from distance;
+-----+-----+-----+
| zipcode1 | zipcode2 | distance |
+-----+-----+-----+
| 64150    | 64151    | 4         |
| 64151    | 64152    | 2         |
| 64152    | 64153    | 3         |
| 64149    | 64150    | 3         |
| 64151    | 64149    | 2         |
+-----+-----+-----+
5 rows in set (0.00 sec)

mysql> select * from emergencyvehicles;
+----+-----+-----+
| id | type | zipcode |
+----+-----+-----+
| 1  | 1    | 64151   |
| 2  | 1    | 64151   |
| 3  | 1    | 64151   |
| 4  | 2    | 64151   |
| 5  | 1    | 64151   |
| 6  | 3    | 64151   |
| 7  | 3    | 64149   |
| 8  | 1    | 64150   |
| 9  | 1    | 64150   |
| 10 | 1    | 64150   |
| 11 | 1    | 64149   |
| 12 | 1    | 64152   |
| 13 | 1    | 64152   |
| 14 | 1    | 64152   |
+----+-----+-----+
14 rows in set (0.03 sec)

mysql> select * from vcount;
+-----+-----+-----+
| count | type | zipcode |
+-----+-----+-----+
| 3     | ambulance | 64151 |
| 0     | ambulance | 64150 |
| 0     | ambulance | 64152 |
| 0     | ambulance | 64149 |
| 3     | ambulance | 64153 |
+-----+-----+-----+
5 rows in set (0.00 sec)

```

## Complexity

The complexity of the algorithm is given by  $n(\log n)$

This can be justified by the following explanation.

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

As we have 2 functions that will be run with one recursive function each, we have a total of two recursive functions with function of 'n' for each iteration. Here, 'n' is the number of zip codes we consider for the service.

## GitHub Link-

<https://github.com/hsingh08/Design-and-Analysis-of-Algorithm-Project>