Assignment No. 4

Department : IT Academic year: 2025-26 Sem: 5 Sub: DAA LAB

Aim:

To design and implement an intelligent traffic management system that finds the shortest path for an ambulance from a source location to the nearest hospital in a dynamic urban environment.

Objective:

- To model the city's road network as a weighted graph.
- To implement Dijkstra's algorithm to calculate the shortest travel time from an ambulance's starting point to all available hospitals.
- To ensure the system can handle real-time updates to road travel times (edge weights) due to changing traffic conditions.
- To identify the optimal route to the nearest hospital and suggest it for navigation.

Problem Statement:

Develop a system to determine the fastest route for an ambulance through a city's road network. The network is represented as a graph where intersections are nodes and roads are weighted edges, with weights corresponding to real-time travel minutes. The system must use Dijkstra's algorithm to find the shortest path from the ambulance's current location (Source) to various hospital locations (Destinations). It must also be capable of dynamically adjusting the recommended path in response to live changes in traffic congestion, thereby minimizing emergency response times.

Outcomes:

1. Apply graph data structures to model and solve a real-world logistics and routing problem. 2. Successfully implement Dijkstra's algorithm to find the single-source shortest path in a weighted graph.
3. Develop a solution that can adapt to dynamic data changes, a key requirement for real-time systems.

Theory:

1. Graph Representation of a City Road Network - A Digital Map

A city road network is represented as a graph in digital maps.

- Nodes (vertices) represent intersections, landmarks, or important places in the city.
- Edges (links) represent roads that connect these points.
  Each edge has a weight, usually showing distance or travel time between two locations.

This graph model helps in solving important problems like:

- Finding the shortest route between two places.
- Calculating fastest travel times.
- Managing traffic flow efficiently.

2. Dijkstra's Algorithm - The Ultimate Route Planner

Dijkstra's Algorithm helps find the shortest path between two points in a road network (graph).

- Each node is an intersection or place.
- Each edge is a road with a weight (distance or time).

How it works:

1. Start from the source node and set its distance to 0.
2. Visit nearby nodes, updating their distances if a shorter path is found.
3. Repeat by choosing the unvisited node with the smallest distance.
4. Continue until all nodes are visited or the destination is reached.

Result: The fastest or shortest route for navigation apps.

3. Handling Dynamic Edge Weights - Adapting to Traffic

In a city road network, edge weights (distance or time) can change due to traffic, accidents, or road work. To adapt, algorithms like Dijkstra's are modified to handle dynamic edge weights by updating the weights in real-time.

How it works:

- Traffic data is collected from sensors or GPS.
- Edge weights are updated frequently to reflect current conditions (e.g., longer travel time in heavy traffic).
- The algorithm recalculates the shortest path using the updated weights.

This ensures that route planners always suggest the fastest, most efficient route based on current traffic conditions.

4. Algorithm for Finding the Shortest Path - Putting It All Together

Here is the entire process from start to finish in simple steps:

1. **Build the Digital Map:** The system first constructs the graph—defining all the intersections (nodes) and roads (edges), and assigning the latest travel times as weights.
2. **Run the Route Planner:** It executes Dijkstra's algorithm, with the ambulance's current location as the starting point.
3. **Find the Nearest Hospital:** After the algorithm is done, the system has a list of the shortest travel times to *every* node. It simply looks at the times for all the hospital nodes and picks the one with the lowest number.
4. **Generate Directions:** To create a navigable route, the algorithm works backward from the chosen hospital, using the path information it saved during the process, to build the step-by-step list of roads the ambulance driver needs to take.
5. **Watch and Repeat:** The system never stops listening for traffic updates. If a major change occurs, it instantly goes back to Step 2 to recalculate the route, ensuring constant optimization.

Questions:

Q1. What is the role of the Priority Queue and why is it essential for the algorithm's performance in a large city?

Ans:A Priority Queue stores nodes, always keeping the node with the smallest distance at the top.

- During the algorithm, the next node to visit is always the one with the shortest known distance so far.

Why It's Essential for Large Cities:

- Large cities have many intersections (nodes) and roads (edges).
- Without a priority queue, searching for the next closest node would take a lot of time (O(n) per step).
- With a priority queue (using a min-heap), each operation (insert, extract-min) takes O(log n) time.

The priority queue makes Dijkstra's algorithm fast and efficient in large city networks by quickly selecting the next node with the smallest distance, improving overall performance.

Q2. Why is Dijkstra's algorithm more suitable for this problem than an uninformed search algorithm like Breadth-First Search (BFS)?

Ans: Dijkstra's Algorithm uses edge weights (like distance or travel time) to always find the shortest or fastest path.

- Breadth-First Search (BFS) treats all edges as equal (unweighted), so it finds the shortest path by number of steps, not distance or time.

Example:

- In a city, one road may be longer but faster (highway), while another is short but slow (traffic jam).
- BFS doesn't consider this and may give a wrong, inefficient route.
- Dijkstra's considers actual distances or times, giving an optimal solution.

Conclusion: Dijkstra's algorithm, with the efficient use of a Priority Queue, provides an optimal and scalable solution for shortest path problems in large cities. Its consideration of weighted edges makes it superior to uninformed search methods like BFS, ensuring practical and reliable route planning in real-world transportation systems.