**GROUP – 4**

**Transport Problem using Prolog logic**

The prolog code defines a graph representing connections between a warehouse and consumers, along with delivery rules and Dijkstra's algorithm for finding the optimal path to deliver orders. The program also takes user input for the number of orders and details for each order.

**Algorithm:**

**1.Graph Representation:**

The connected predicate defines connections between nodes (e.g., 'Warehouse', consumers) with random weights using random\_weight. random\_weight generates a random weight between 1 and 10.

**2.Dijkstra's Algorithm:**

The dijkstra\_pathpredicate implements Dijkstra's algorithm to find the optimal path from the warehouse to a consumer. It uses the connected predicate to traverse the graph and calculate the cost of each path.

**3.Delivery Rules:**

The deliver\_orders predicate initiates the delivery process, starting from the warehouse and iterating through a list of orders. It uses the dijkstra\_path predicate to find the optimal path for each order. Depending on whether the warehouse is part of the path, the program either delivers with or without a refill.

**4.User Interaction:**

The create\_graph predicate creates a graph based on the defined connections.

The print\_graph predicate prints the edges of the graph.

The get\_orders predicate takes user input for the number of orders and order details.

**5.Example Usage:**

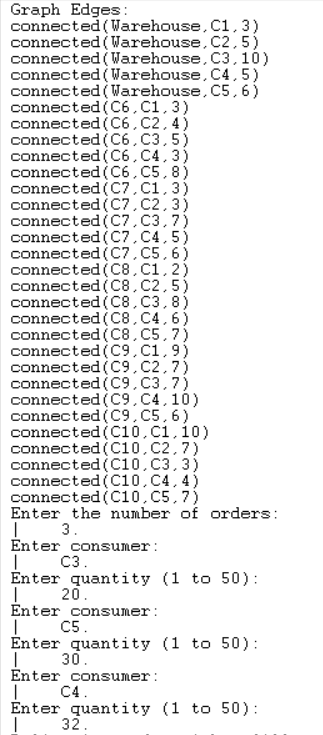
The main/0 predicate initializes the program, creating the graph, printing it, getting user orders, and initiating the delivery process.

**6.Refilling:**

The program simulates refilling at the warehouse when necessary, and it uses the deliver\_without\_refill predicate for both refilling and normal delivery.

The algorithm assumes a truck capacity of 50 units and calculates the cost of each delivery path. The algorithm follows a procedural flow where the optimal path is calculated for each order, and the delivery process is simulated accordingly.

**Sample I/O:**



**Output:**

