SWIFTEX COURIER ENGINE

Project Documentation

Session 2023 - 2027

Submitted by:

AFEEF BIN MASOOD 2024-CD-CS-650

Submitted to:

Sir Ali Raza

Course:

CSC-200L Data Structures (DSA)

Department of Computer Science

University of Engineering and Technology

New Campus

1. Project Overview:

SwiftEx Courier Engine is a menu-driven, console-based courier management system implemented fully in C++ using custom data structures. The system simulates the complete lifecycle of a parcel from registration and pickup to routing, delivery, tracking, and exception handling while demonstrating core Data Structures & Algorithms (DSA) concepts. The project strictly avoids STL containers (queues, stacks, maps, etc.) and instead implements linked lists, stacks, queues, heaps, graphs, and hash tables manually, fulfilling typical Data Structures course requirements.

2. Objectives:

The main objectives of this project are:

- To design a complete courier management system using C++
- To implement all major data structures manually (no STL)
- To apply appropriate algorithms for sorting, searching, and routing
- To provide a fully interactive CLI-based system
- To simulate the complete lifecycle of a parcel
- To demonstrate undo functionality and exception handling

3. Scope of the Project:

The system supports:

- Parcel registration with priority and weight categorization
- Multi-stage parcel processing (Pickup → Sorting → Warehouse → Transit)
- Priority-based sorting using a Min-Heap
- Rider assignment with capacity constraints
- Route calculation with shortest path and alternative routes
- Dynamic road blocking and restoration
- Parcel tracking using a hash table
- Undoing recent actions
- Delivery lifecycle simulation

Limitations:

- Console-based interface only
- Fixed number of cities and riders
- No persistent storage (data resets on exit)

4. System Architecture:

4.1 Architectural Overview:

The system follows a modular layered architecture. A central controller class (Courier System) coordinates between different modules such as queues, heap, routing engine, tracking engine, and undo stack.

Description Of Diagram:

1. **Input Processing Module:**

   - Responsible for the Main Menu loop.
   - Handles data sanitation (e.g., ensuring cin doesn't crash if a user enters text instead of a number).

2. **Courier Engine:**

   - The central hub (Courier System class) that connects all other modules. It ensures that a parcel cannot be routed before it is sorted.

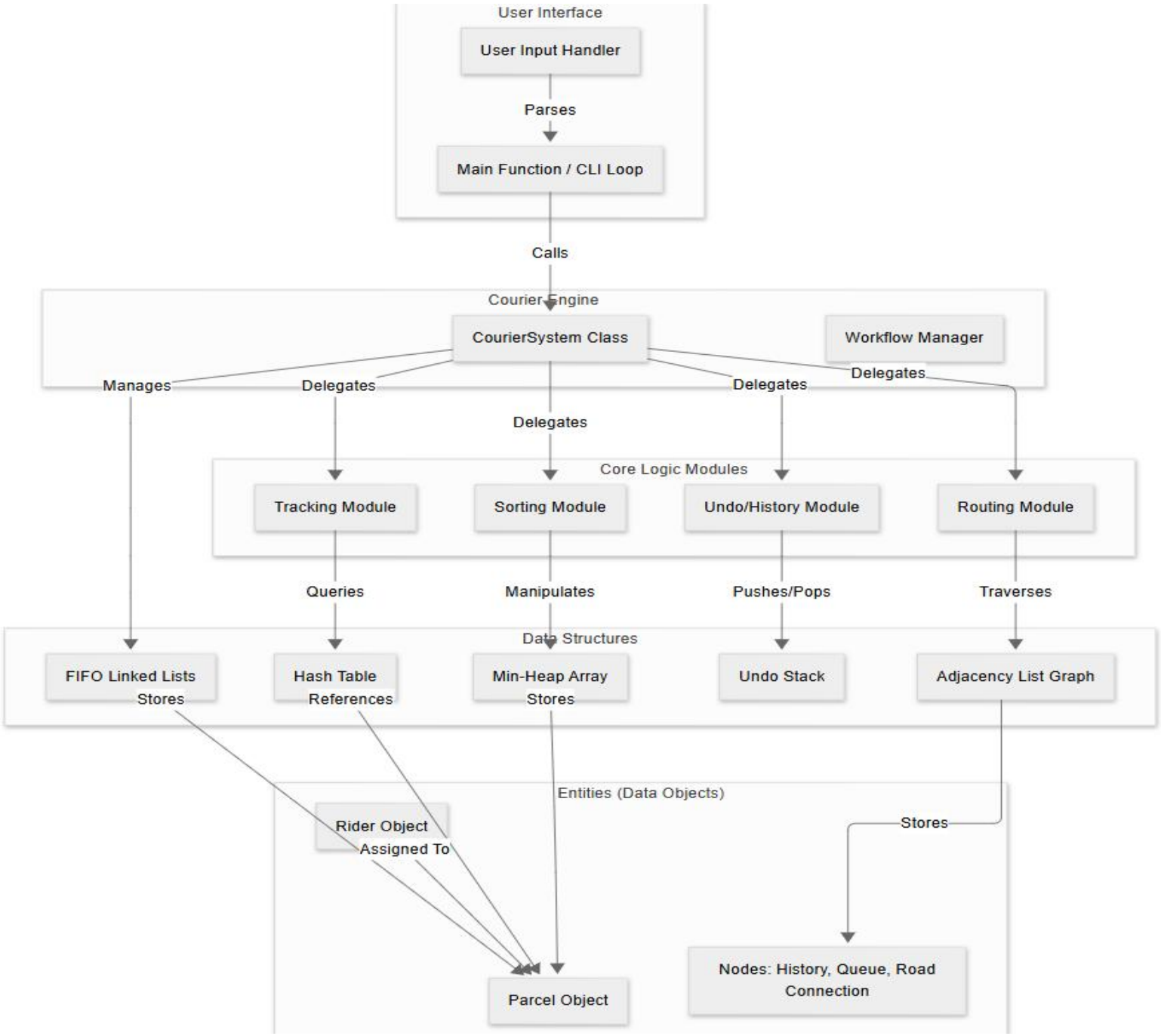
3. **Core Logic Module:**

   - Priority Manager: Uses the Min-Heap to ensure overnight packages are handled first.
   - Route Optimizer: Uses the Graph to calculate costs and distances.
   - Load Balancer: Uses Rider logic to ensure trucks aren't overloaded beyond their capacity (e.g., 200kg limit).

4. **Data Management Module:**

   - Live Tracking: The Hash Table allows for O(1) lookup speed.
   - State Memory: The Stack that stores previous states to enable the "Undo" feature.

4.2 System Architecture Diagram:



5. UML Class Diagram:

The UML Class Diagram breaks the system down into its core data objects, the specific data structures used to manage them, and the central controller that runs the logic.

Brief explanation of the UML class diagram:

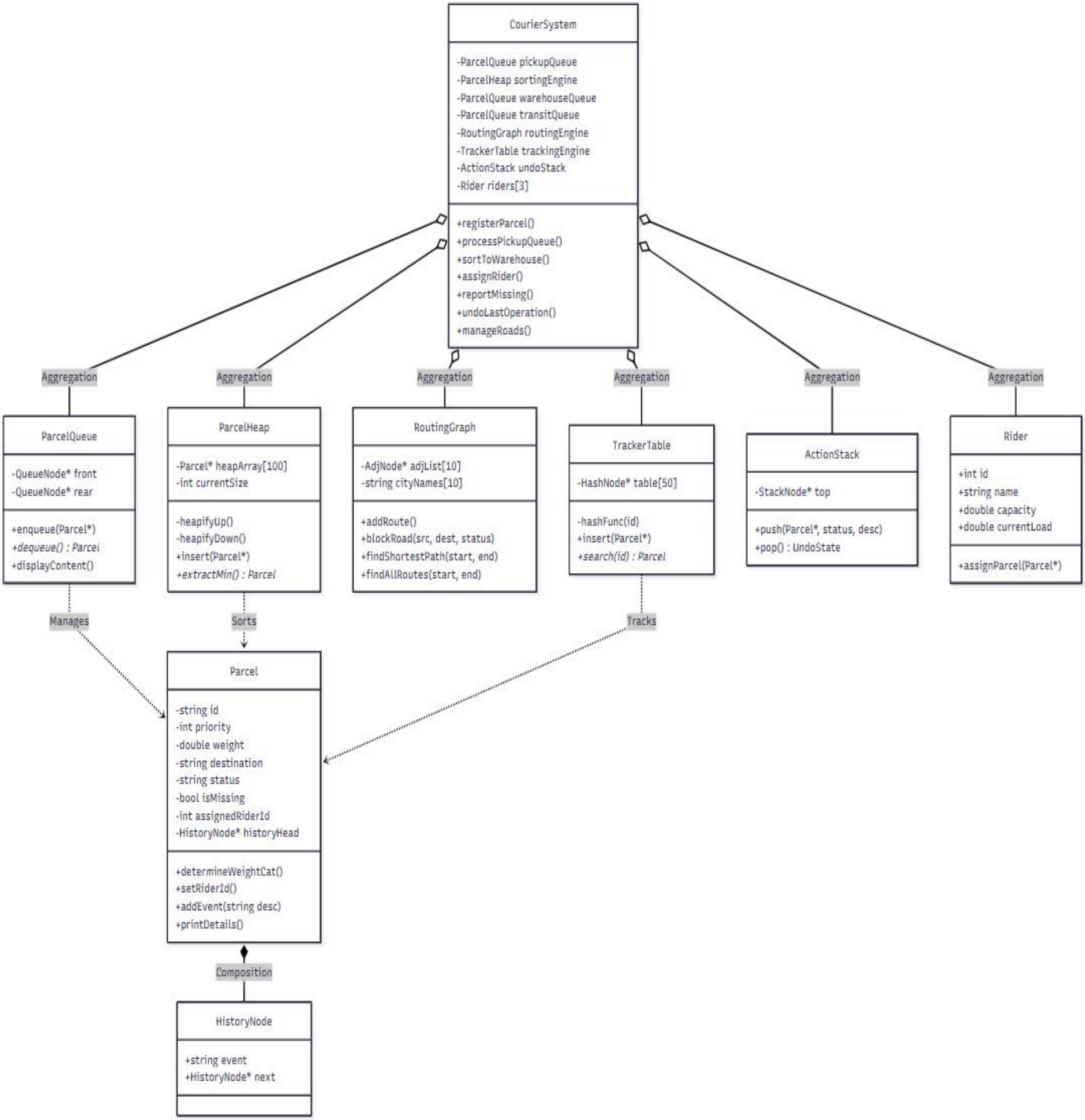
This diagram models a **Courier Logistics System** using multiple data structures.

- **Parcel (core entity):**
  Represents a shipment with ID, priority, weight, destination, status, and assigned rider. It maintains its own history log and provides functions to update events and print details.
- **HistoryNode:**
  A singly linked list node used by Parcel to store audit/history events (e.g., dispatched, delivered). This is a composition, meaning history exists only with a parcel.
- **ParcelQueue:**
  A queue data structure used to manage parcels at different stages (pickup, warehouse, transit).
- **PriorityScheduler:**
  A min-heap that sorts parcels based on priority for efficient scheduling.
- **RoutingGraph:**
  A graph representing cities and roads. It supports route addition, road blocking, shortest path, and all-paths search for parcel routing.
- **TrackerTable:**
  A hash table that enables fast parcel tracking using parcel IDs.
- **undoStack:**
  A stack that stores previous parcel states to support undo operations.
- **Rider:**
  Represents delivery riders with capacity management and parcel assignment.
- **CourierSystem:**
  The main controller that coordinates all components—queues, heap, graph, hash table, undo stack, and riders—to manage parcel registration, sorting, routing, delivery, and undo actions.

Relationships:

- Parcel composes HistoryNode.
- CourierSystem aggregates all major subsystems.
- Queues, heap, and hash table depend on Parcel for management, sorting, and tracking.

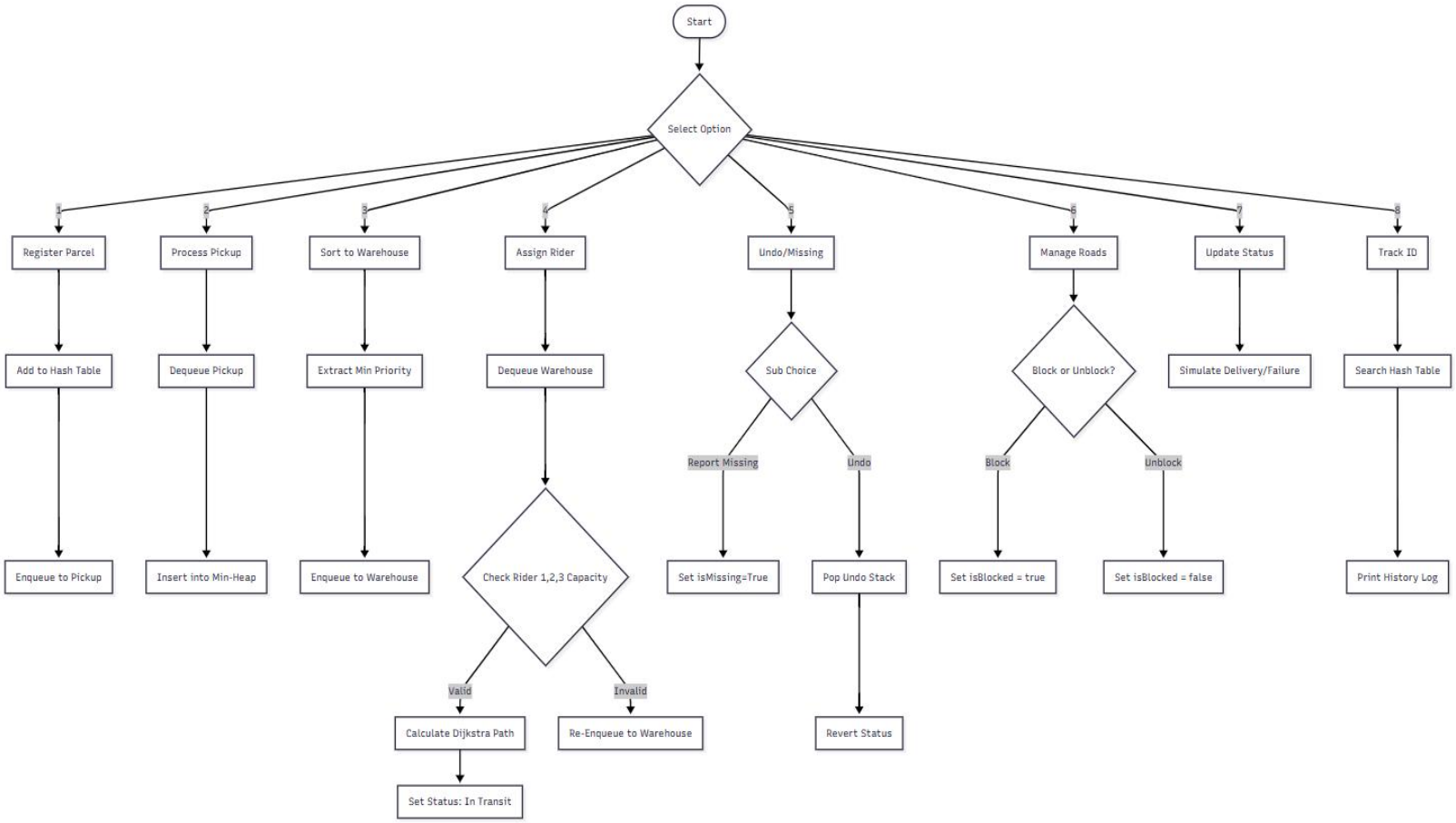
Class Diagram:



6. Flowcharts:

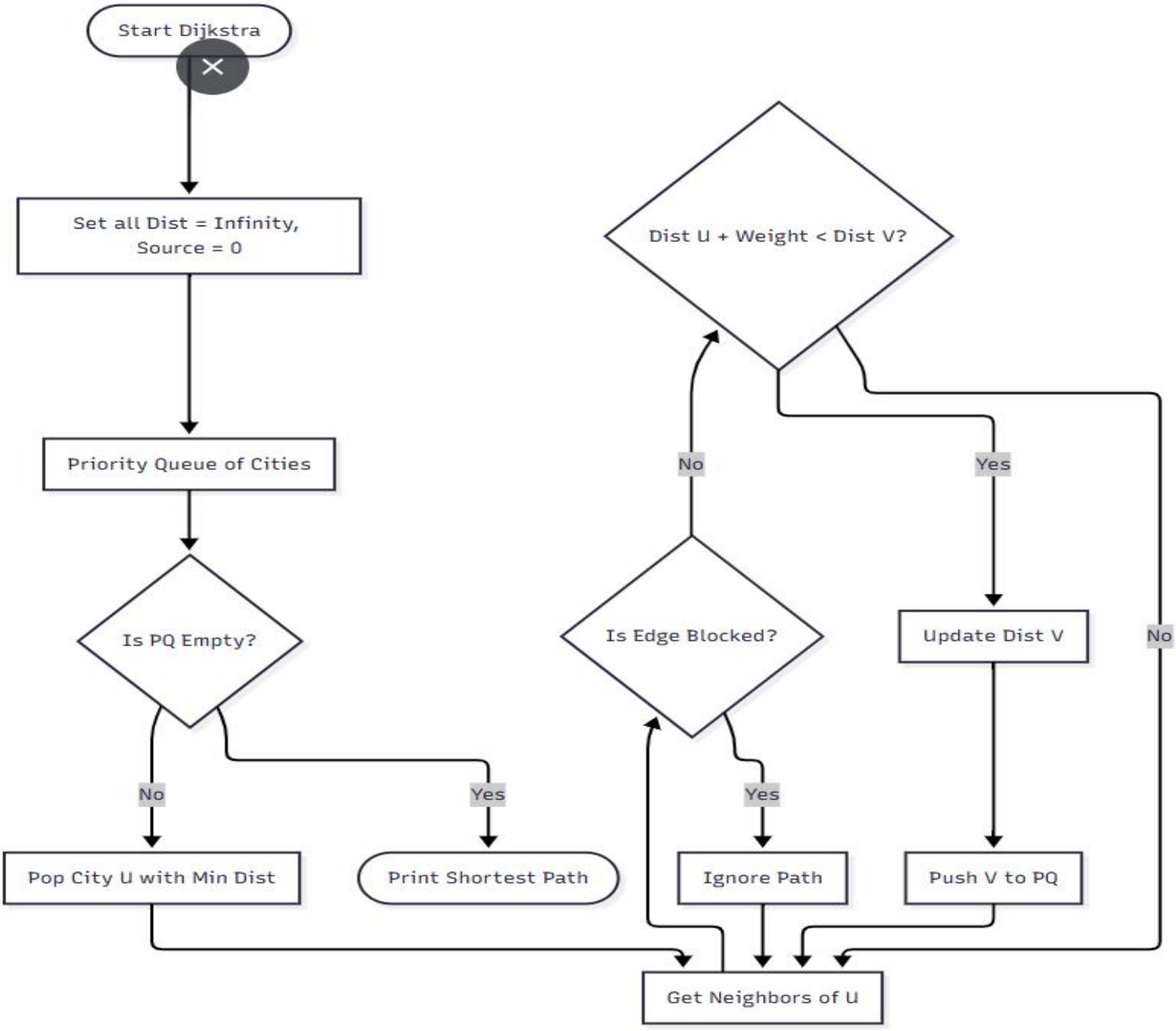
The flowchart represents the menu-driven workflow of a courier logistics system.

- The system starts and displays a menu for user actions.
- Register Parcel: A new parcel is created, stored in the hash table for tracking, and added to the pickup queue.
- Process Pickup: Parcels are removed from the pickup queue and inserted into a min-heap for priority-based scheduling.
- Sort to Warehouse: The highest-priority parcel is extracted from the heap and enqueued into the warehouse queue.
- Assign Rider: A parcel is dequeued from the warehouse, rider capacity is checked, and if valid, the shortest route (Dijkstra) is calculated and the parcel status is set to *In Transit*. If not valid, it is re-queued.
- Undo / Report Missing: Either marks a parcel as missing or undoes the last action using the undo stack.
- Manage Roads: Roads can be blocked or unblocked, affecting routing decisions.
- Update Status: Simulates parcel lifecycle events such as delivery or failure.
- Track Parcel: Searches the hash table by parcel ID and prints its history log.



6.1 Dijkstra with Road Blockage:

This explains the logic inside RoutingGraph::findShortestPath.



7. Data Structures Used and Justification:

7.1 Linked List

Used in:

- Parcel history logs
- Queue implementation
- Stack implementation

Justification:

- Dynamic memory allocation
- Efficient insertion
- Suitable for unknown data size

7.2 Queue (FIFO)

Used in:

- Pickup Queue
- Warehouse Queue
- Transit Queue

Justification:

- Models real-life parcel flow
- O(1) enqueue and dequeue

7.3 Stack (LIFO)

Used in:

- Undo functionality

Justification:

- Undo operations naturally follow LIFO behavior
- Efficient rollback of recent actions

7.4 Min-Heap

Used in:

- Parcel sorting engine

Sorting Criteria:

- Higher priority (lower numeric value)
- Heavier parcel preference for equal priority

Justification:

- Efficient priority-based scheduling
- O(log n) insertion and deletion

7.5 Graph (Adjacency List)

Used in:

- City routing

Algorithms Implemented:

- Dijkstra's Algorithm (Shortest Path)
- Depth First Search (All Routes)

Justification:

- Accurately models road networks
- Supports dynamic road blocking

7.6 Hash Table

Used in:

- Parcel tracking by ID

Justification:

- O(1) average search time
- Fast and efficient tracking system

8. Algorithms Implemented:

Algorithm	Purpose
Dijkstra	Shortest delivery route
DFS	Alternative route discovery
Heapify	Priority parcel sorting
Hashing	Fast parcel lookup

9. User Interface (CLI):

The system provides a menu-driven interface allowing users to:

- Register parcels
- Process pickup queue
- Sort parcels
- Assign riders and calculate routes
- Report missing parcels / undo actions
- Manage road blockages
- Update delivery status
- Track parcels

10. Demonstration Guide:

To complete the demonstration, follow this script when presenting or running the code:

Scenario: Delivering an Urgent Package from Lahore to Karachi.

Step 1: Registration

- Select Option 1.
- **ID:** PKG-101
- **Priority:** 1 (High/Overnight)
- **Weight:** 5.5
- **Destination:** Karachi
- *Result:* Parcel enters Pickup Queue.

Step 2: Processing

- Select Option 2 (Move from Pickup to Sorting).
- Select Option 3 (Move from Sorter to Warehouse).
- *Observation:* Note how the system auto-calculates "Heavy/Medium" and Priority.

Step 3: Routing & Assignment

- Select Option 4.
- *Result:* System assigns a Rider (likely the Van or Truck depending on weight).
- *Result:* System prints the Optimal Route: Lahore -> Multan -> Karachi.
- *Cost:* 1290 (340 + 950).

Step 4: Road Block Simulation

- Select Option 6 -> 1 (Block Road).

- Block Multan to Karachi.
- Run Option 6 -> 3 (Show Alternatives).
- *Result:* System will show alternate paths or DFS routes avoiding the blocked road.

Step 5: Tracking

- Select Option B.
- Enter PKG-101.
- *Result:* View the full history log (Received -> Sorting -> Warehouse -> Assigned Rider).

Step 6: Undo

- Select Option 5 -> 1 (Mark Missing).
- Select Option 5 -> 2 (Undo).
- *Result:* Status reverts from "MISSING" back to "In Transit".

11. Linked In Video Link:

https://www.linkedin.com/posts/afeef-bin-masood-8b03ab30b_this-project-is-swiftex-courier-engine-implemented-activity-7419387300167438336-SQgI?utm_source=share&utm_medium=member_desktop&rcm=ACoAAE7nDUQB2j40CA5ihdkg_AzTJV5fLTUwbBU

12. Conclusion:

The SwiftEx Courier Engine successfully demonstrates the practical application of Data Structures and Algorithms in a real-world inspired problem. By avoiding STL containers and implementing all data structures manually, the project reinforces core DSA concepts. The system is modular, extensible, and academically robust