Intelligent Courier Logistics Engine

Session 2024 - 2028

Submitted By:

Muhammad Ayan Sajid	**2024 CS 661**
Shehroz Shafaqat	**2024 CS 690**
Shareen Asim	**2024 CS 693**
Fiza Shahid Khan	**2024 CS 680**

Submitted To:

Mr. Ali Raza

Department of Computer Science, UET Lahore, New Campus

Table of Contents

Table of Figures

1 Introduction

The goal of this project is to design and implement an **Intelligent Parcel Sorting, Routing, and Tracking System** for a courier logistics company. The system uses appropriate data structures and algorithms to manage parcel sorting, route selection, tracking, and courier operations efficiently. Through this project, real-world logistics problems are addressed while demonstrating problem-solving skills, structured program design, and ethical computing practices using C++.

2 Project Overview

The Intelligent Courier Logistics Engine is a C++–based simulation system developed to model real-world courier logistics operations. The system automates parcel sorting, route computation, tracking, and courier workflow management using fundamental Data Structures and Algorithms. It is designed as a menu-driven CLI application that processes parcel and map data from input files and performs intelligent decision-making to ensure efficient delivery operations.

3 UML Diagram

3.1 Class Diagram

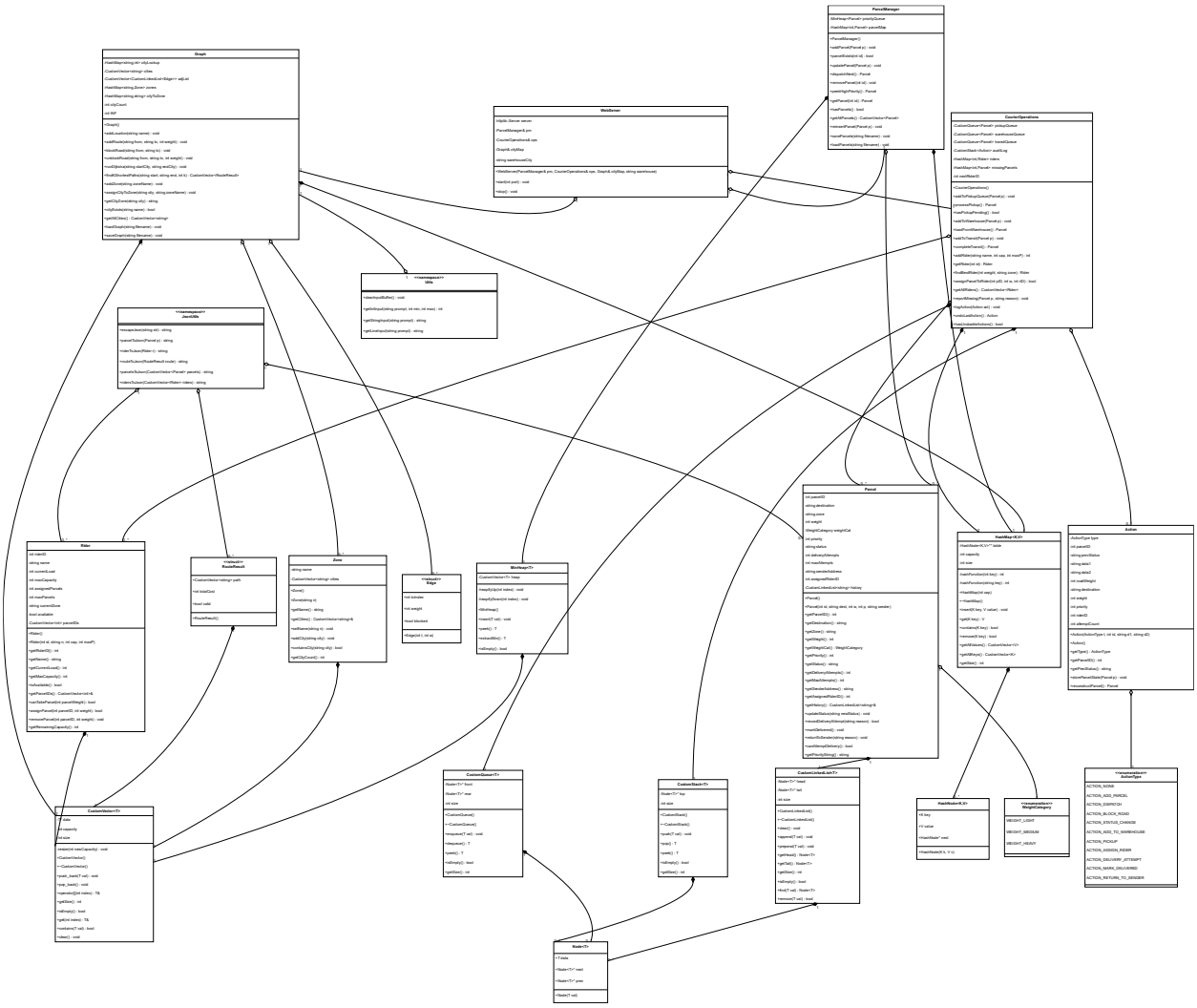

Figure 1 Class Diagram

4 Flowcharts

4.1 Main Application Flowchart

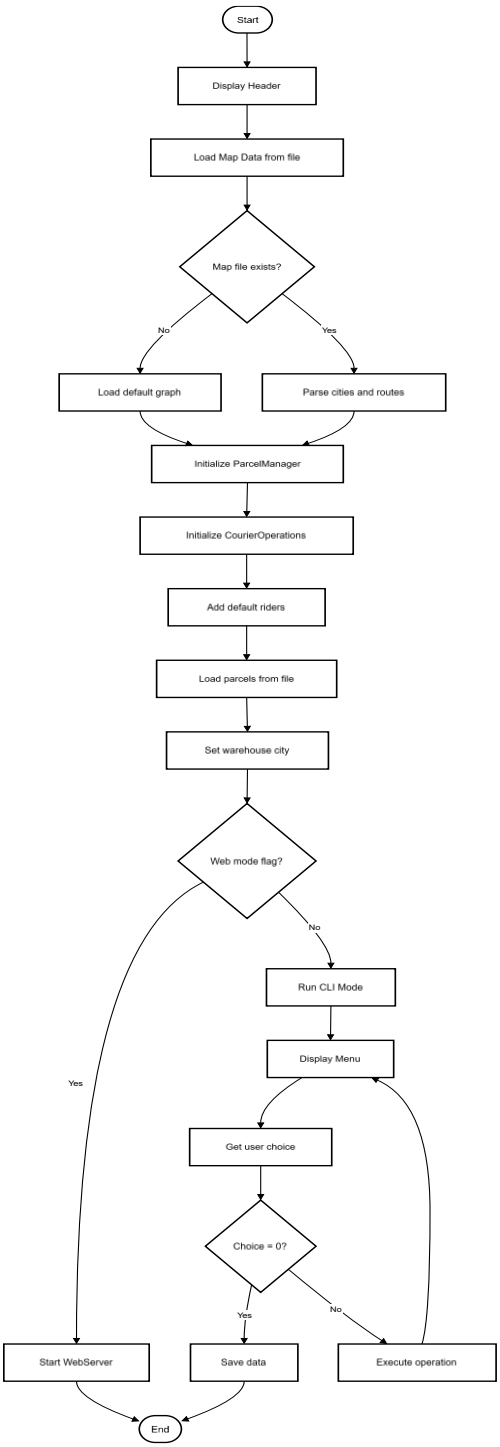

Figure 2 Main Application Flowchart

4.2 Add Parcel Flowchart

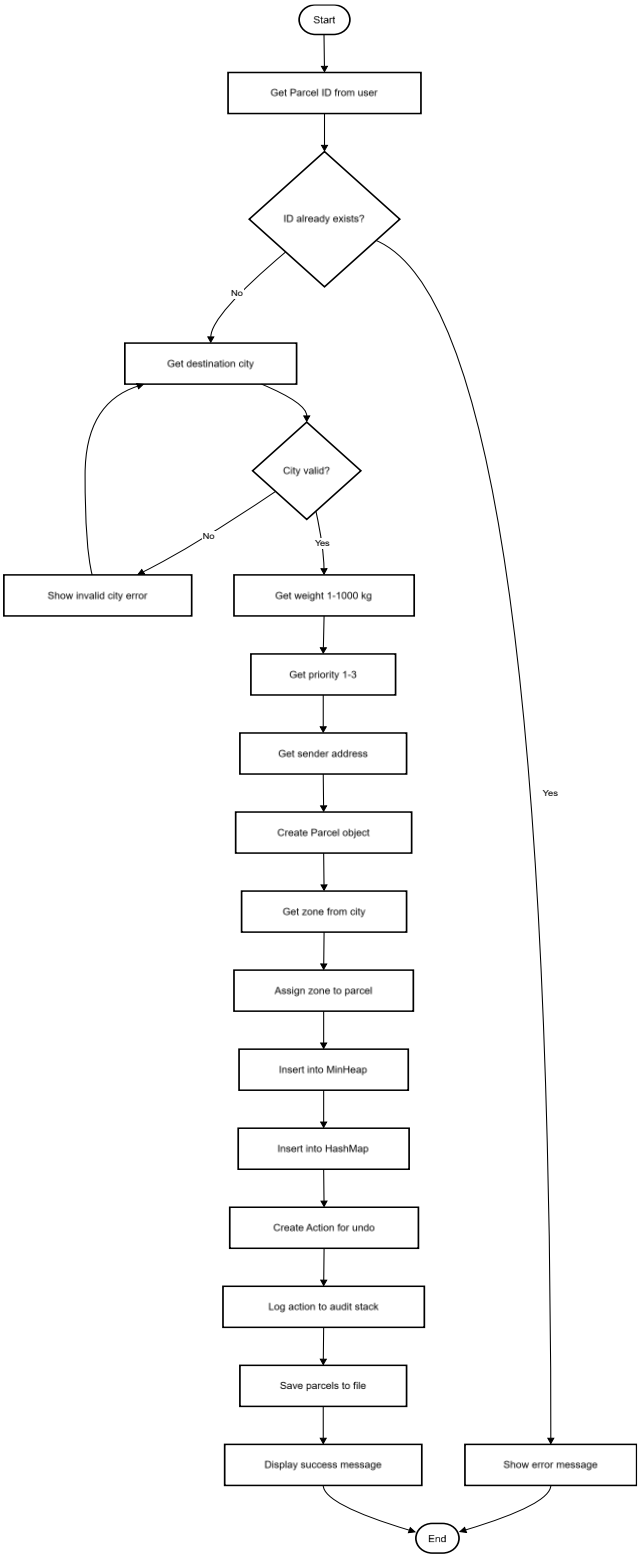

Figure 3 Add Parcel Flowchart

4.3 Dispatch Parcel Flowchart

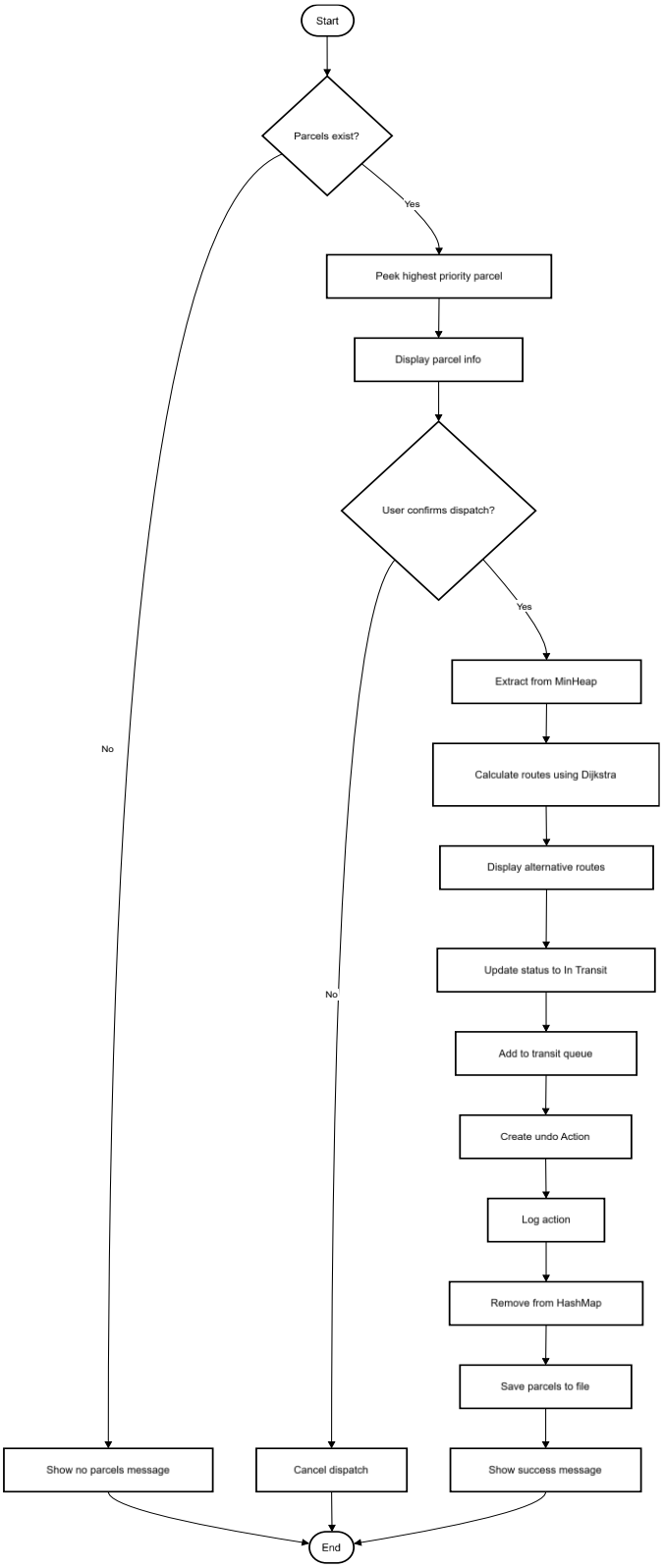

Figure 4 Dispatch Parcel Flowchart

4.4 Dijkstra's Shortest Path Algorithm Flowchart

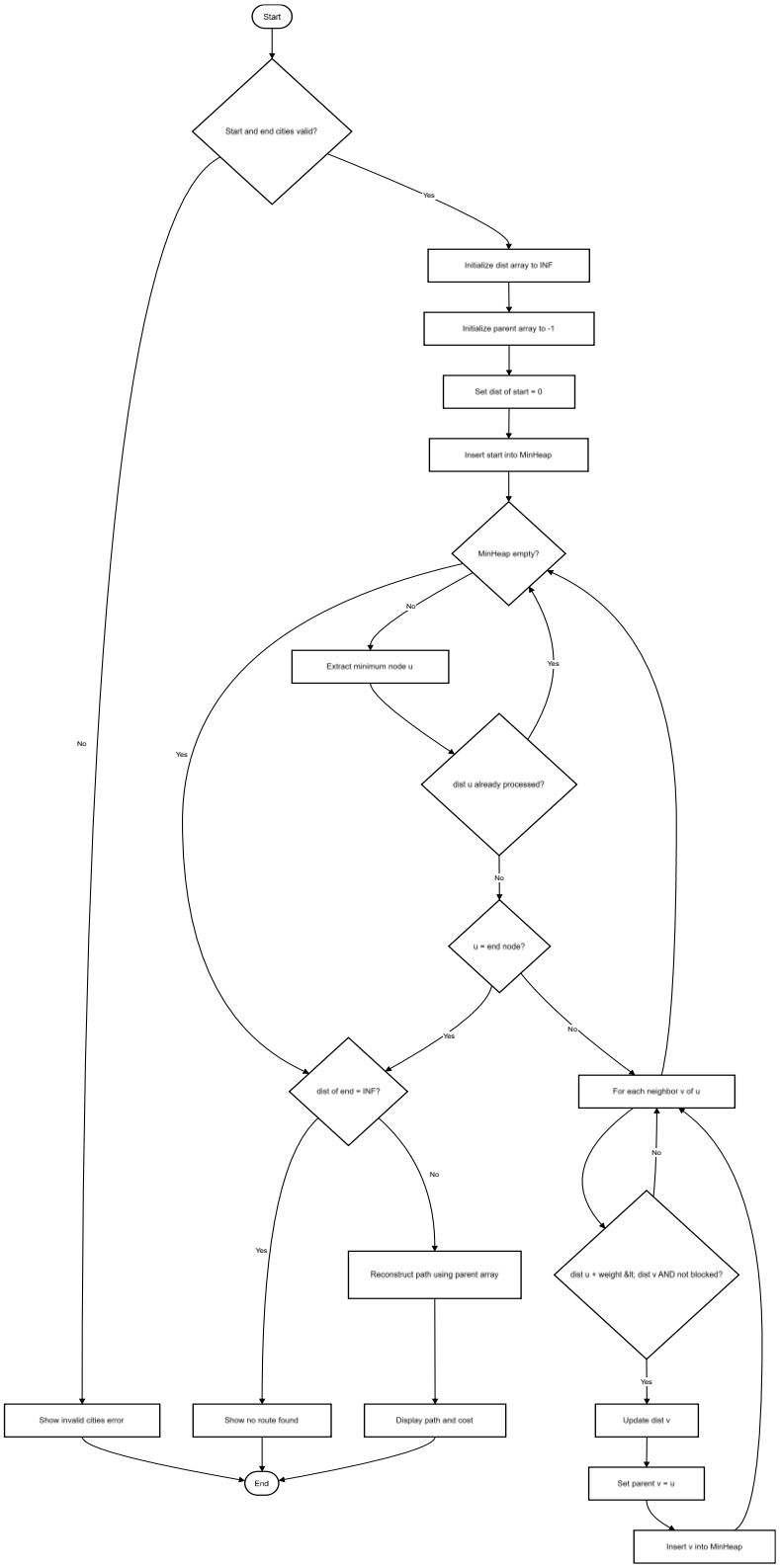

Figure 5 Dijkstra's Shortest Path Algorithm Flowchart

4.5 K-Shortest Paths Algorithm Flowchart

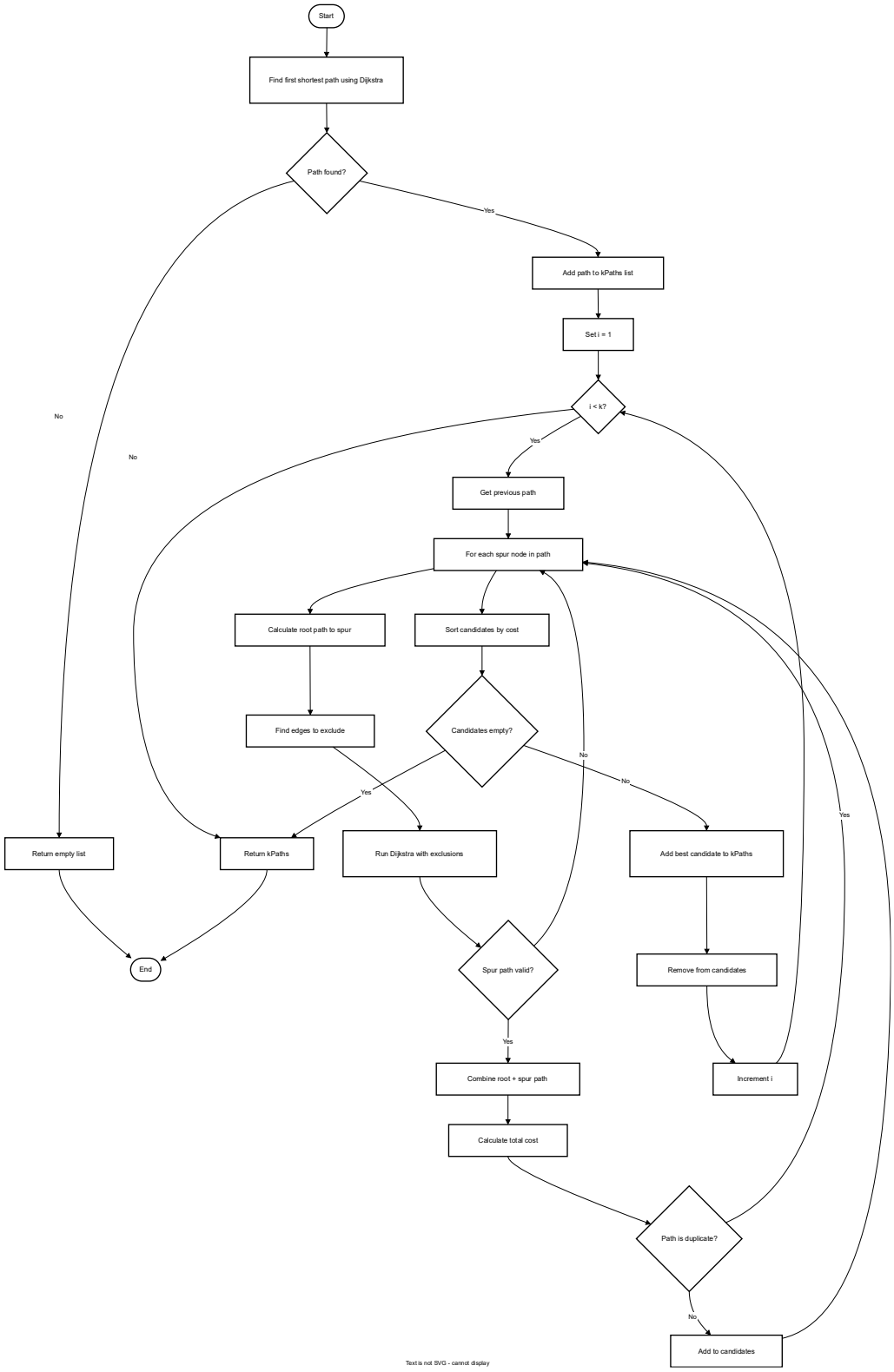

Figure 6 K-Shortest Path Algorithm Flowchart

4.6 Record Delivery Attempt Flowchart

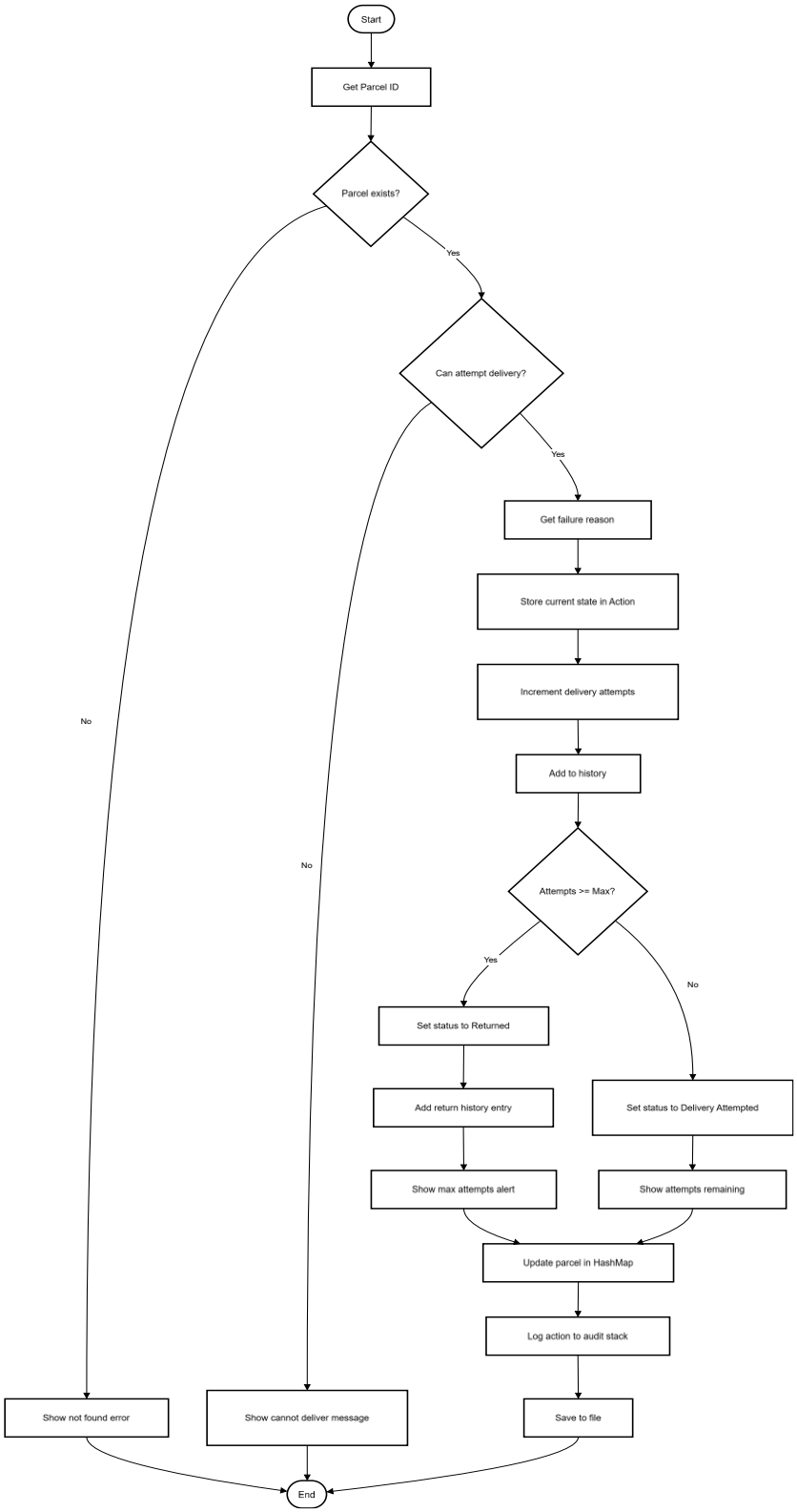

Figure 7 Record Delivery Attempt Flowchart

4.7 Assign Parcel to Rider Flowchart

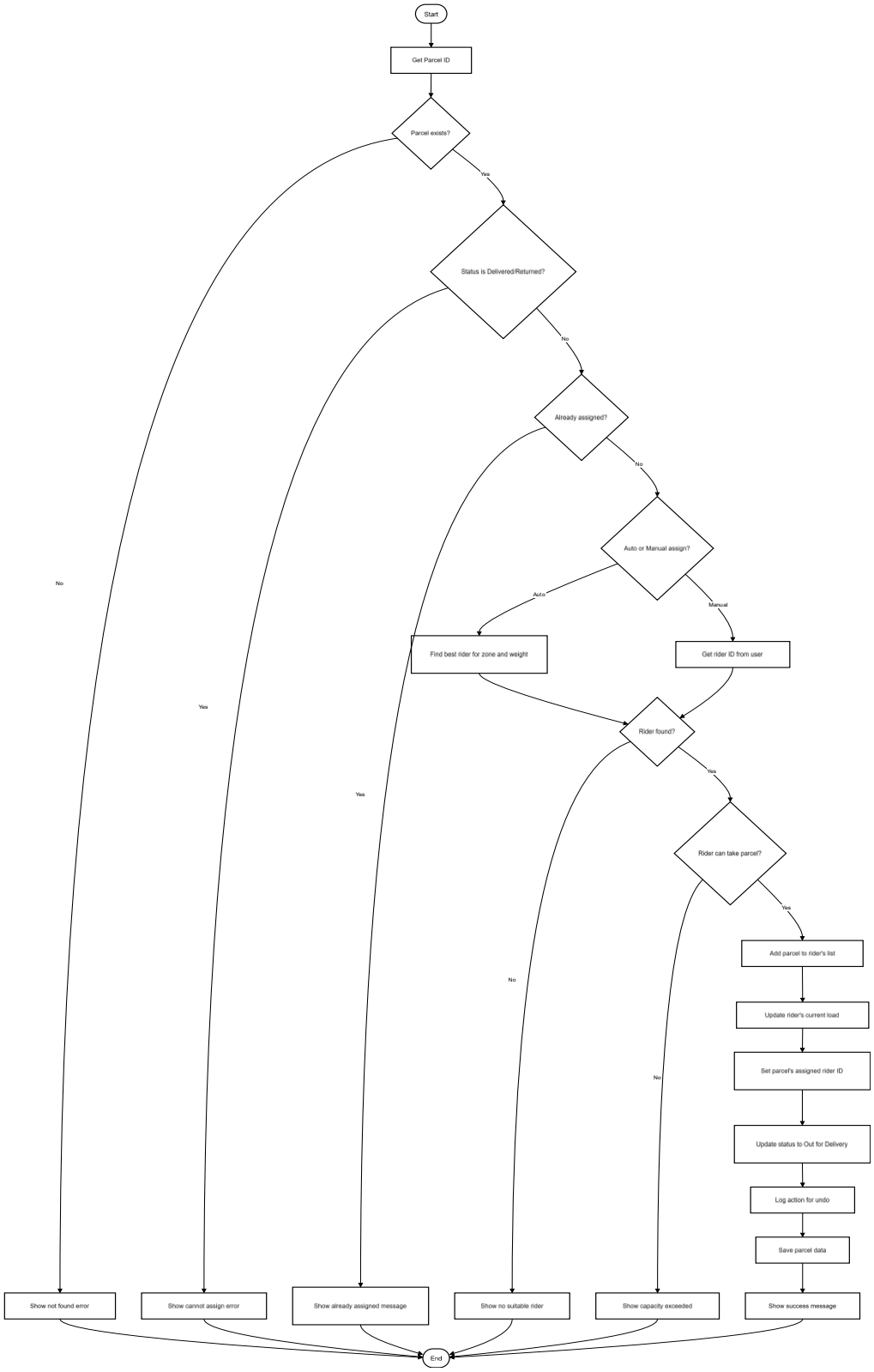

Figure 8 Assign Parcel to Rider Flowchart

4.8 Find Best Rider Flowchart

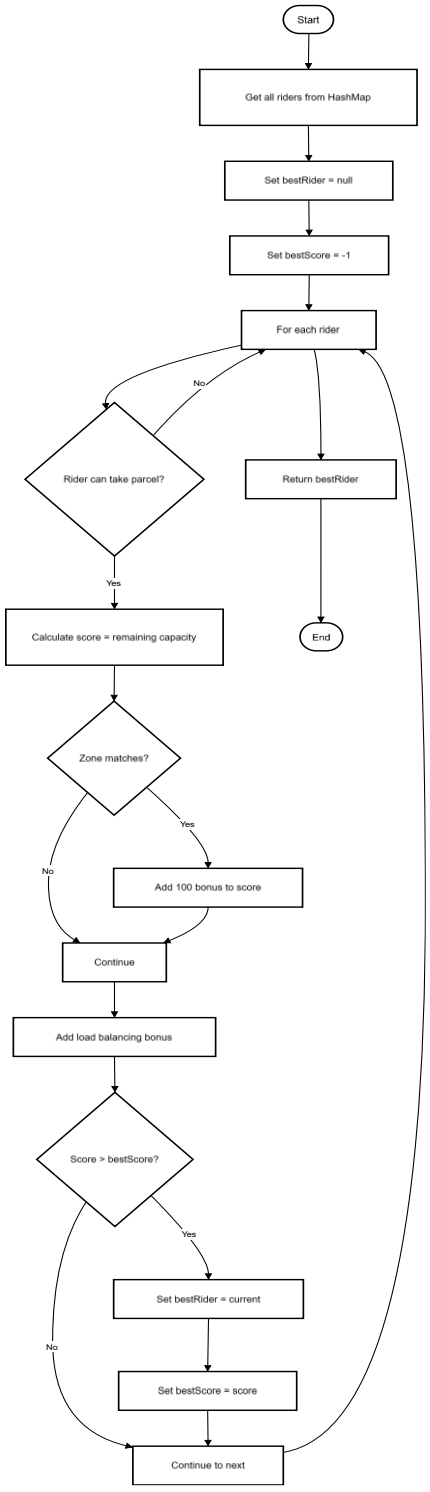

Figure 9 Find Best Rider Algorithm

4.9 Undo Operation Flowchart

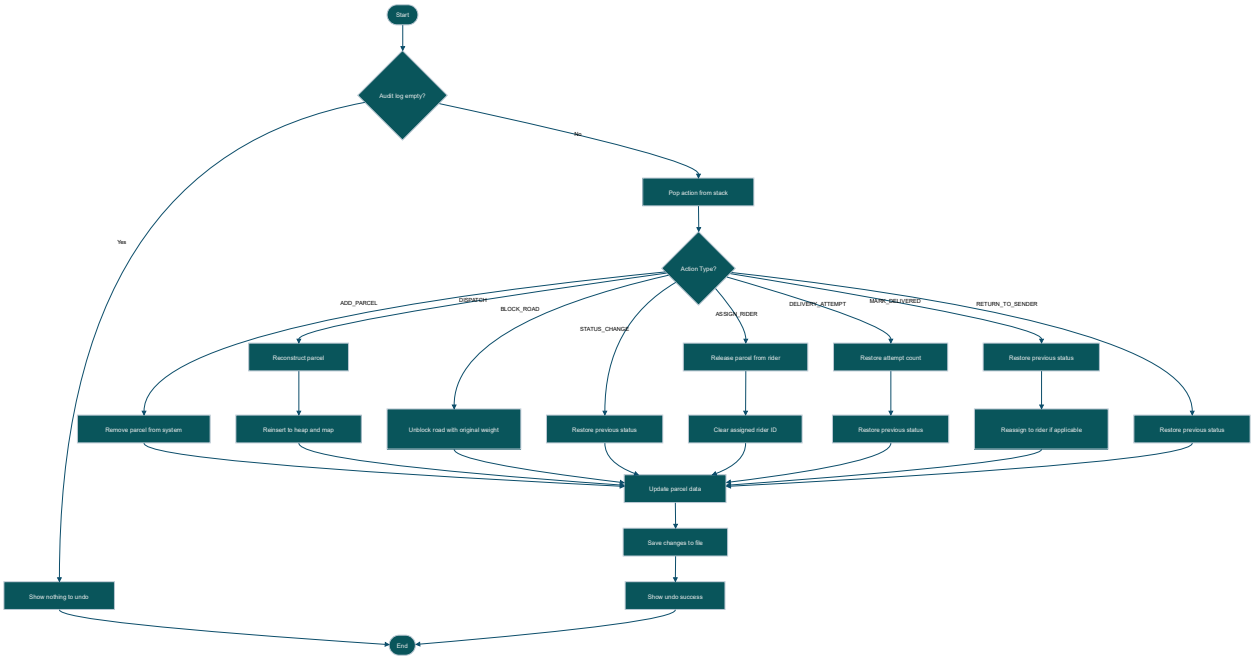

Figure 10 Undo Operation Flowchart

4.10 Block/Unblock Road Flowchart

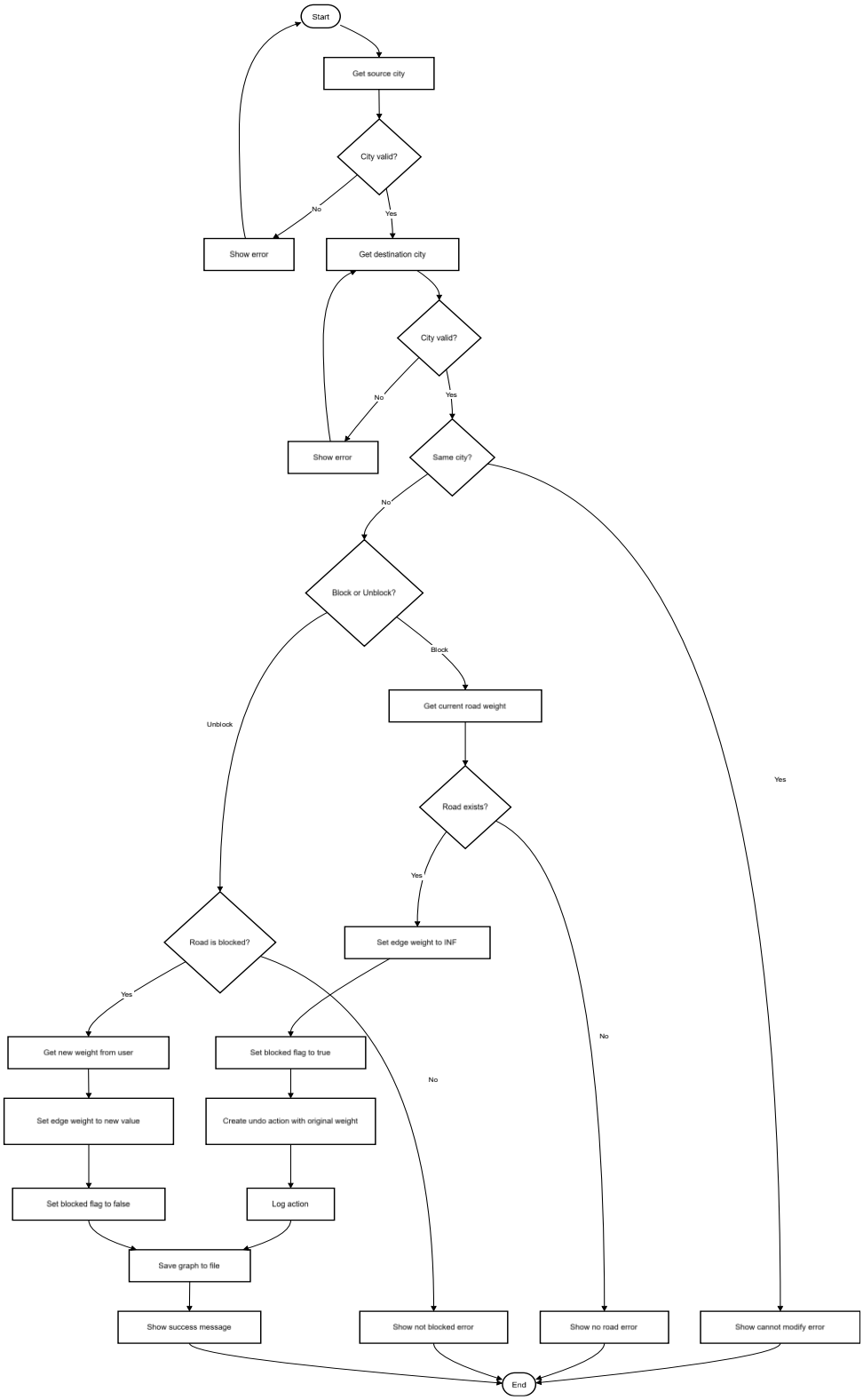

Figure 11 Block/Unblock Road Flowchart

4.11 Parcel Lifecycle Flowchart

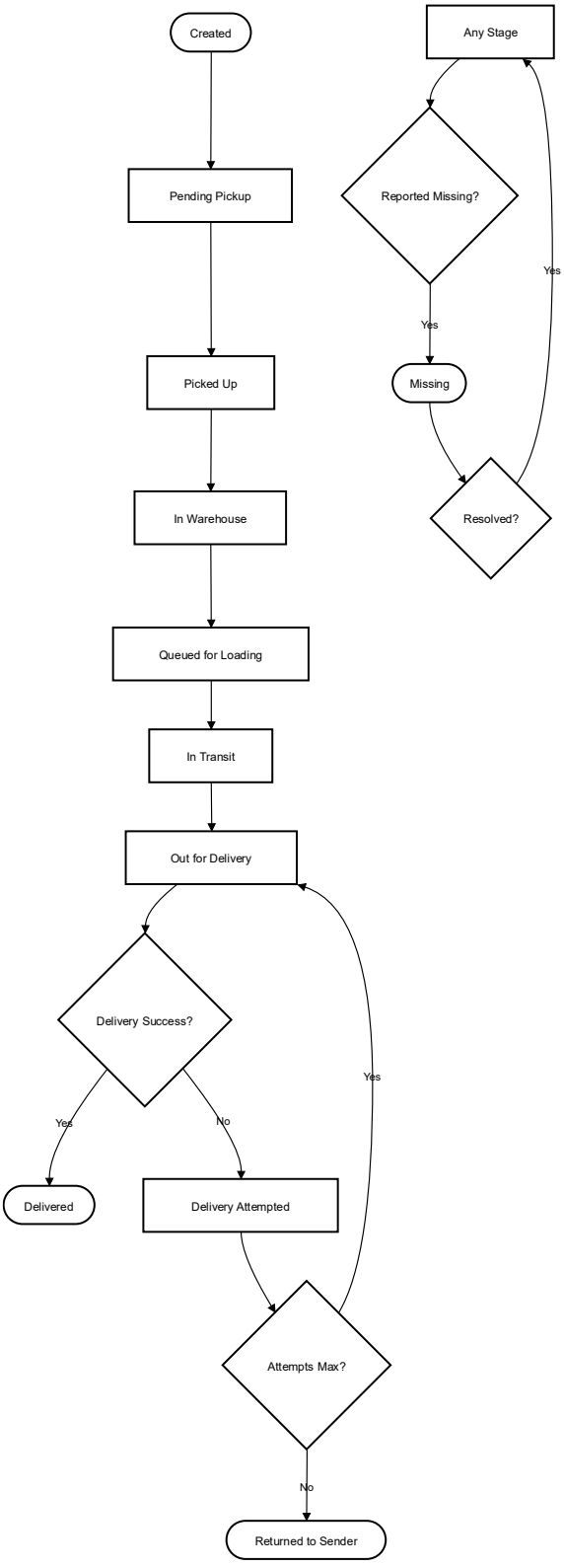

Figure 12 Parcel Lifecycle Flowchart

4.12 Queue Operations (Pickup/Warehouse/Transit) Flowchart

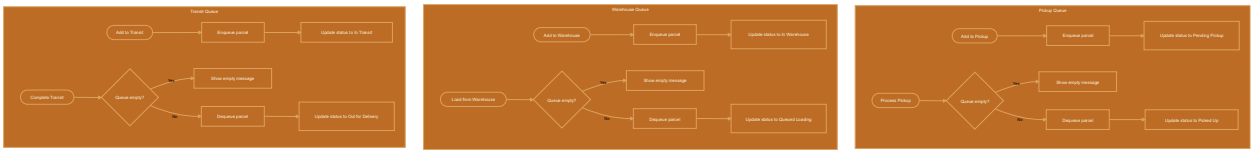

Figure 13 Queue Operations (Pickup/Warehouse/Transit) Flowchart

4.13 MinHeap Insert Operation Flowchart

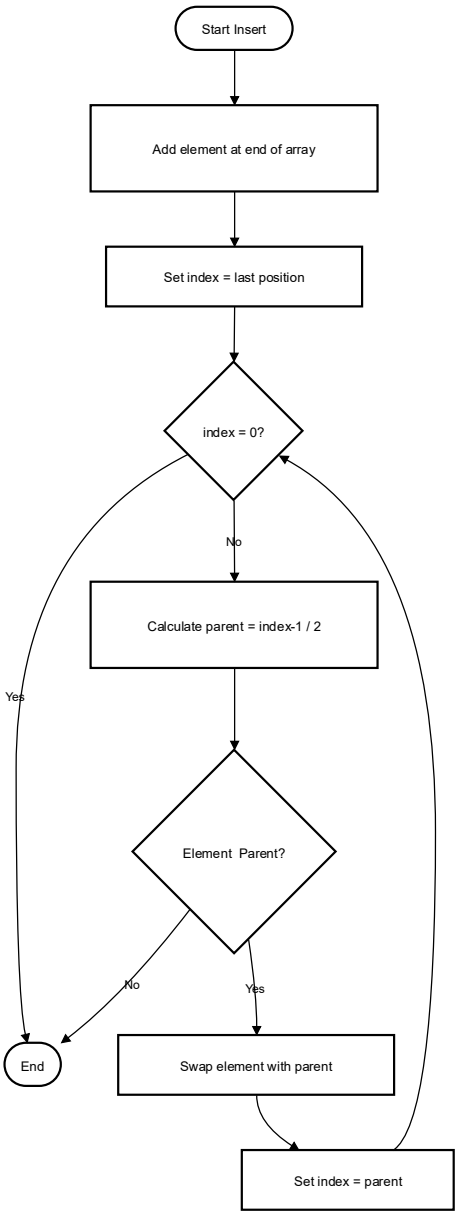

Figure 14 MinHeap Insert Operation

4.14MinHeap Extract Min Operation Flowchart

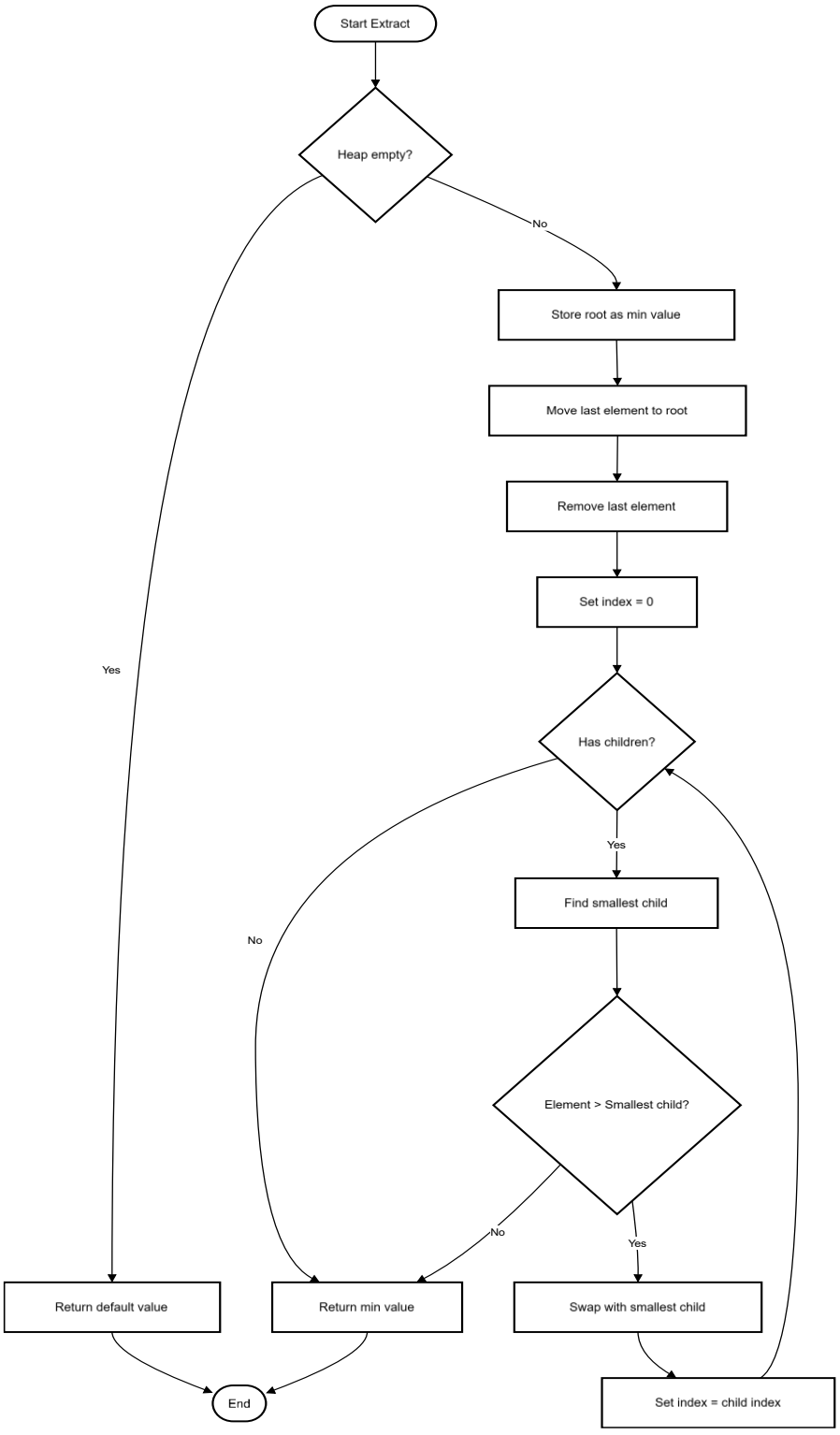

Figure 15MinHeap Extract Min Operation Flowchart

4.15 HashMap Insert Operation Flowchart

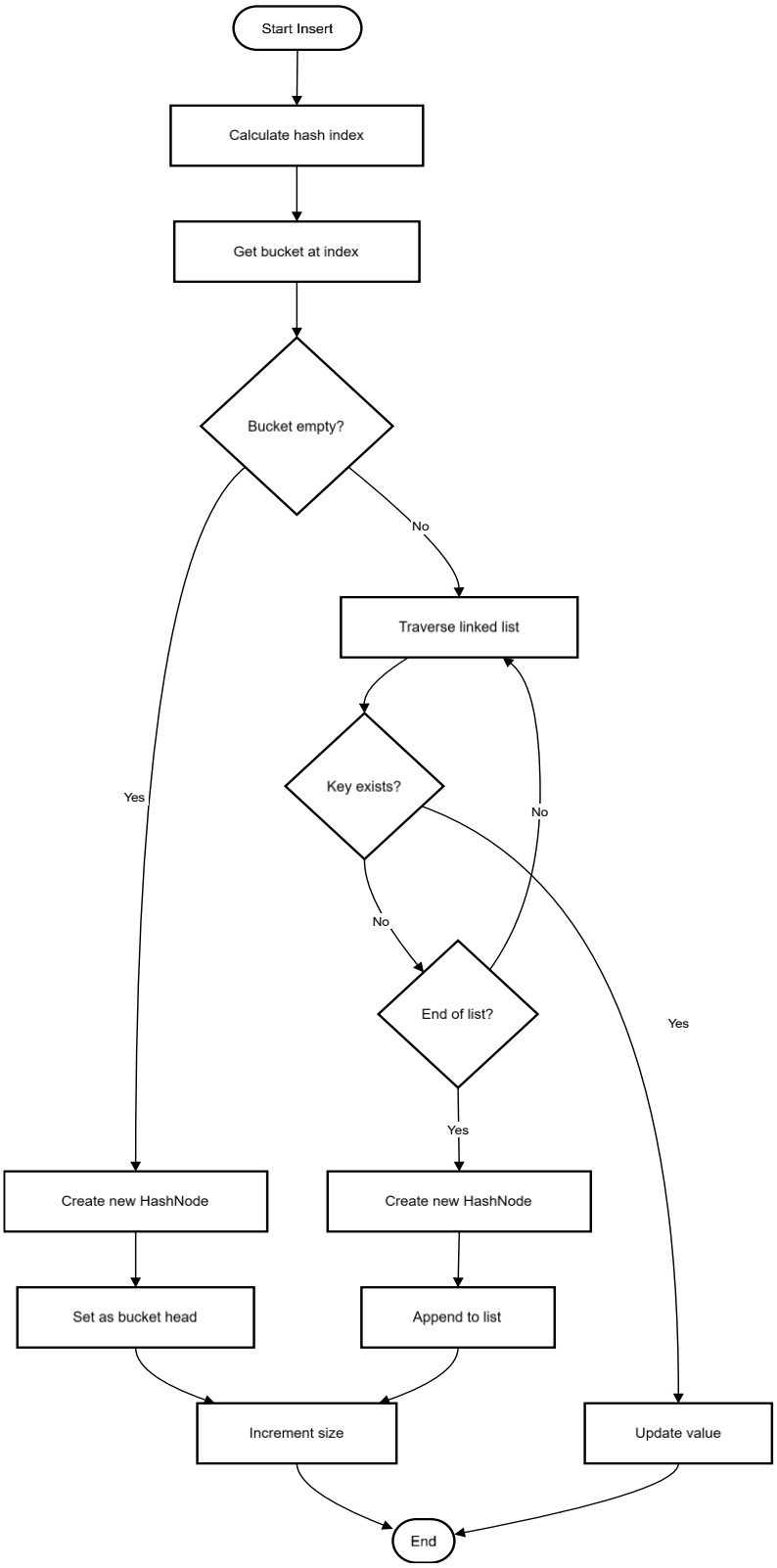

Figure 16 HashMap Insert Operation Flowchart

4.16 Zone Auto-Assignment Flowchart

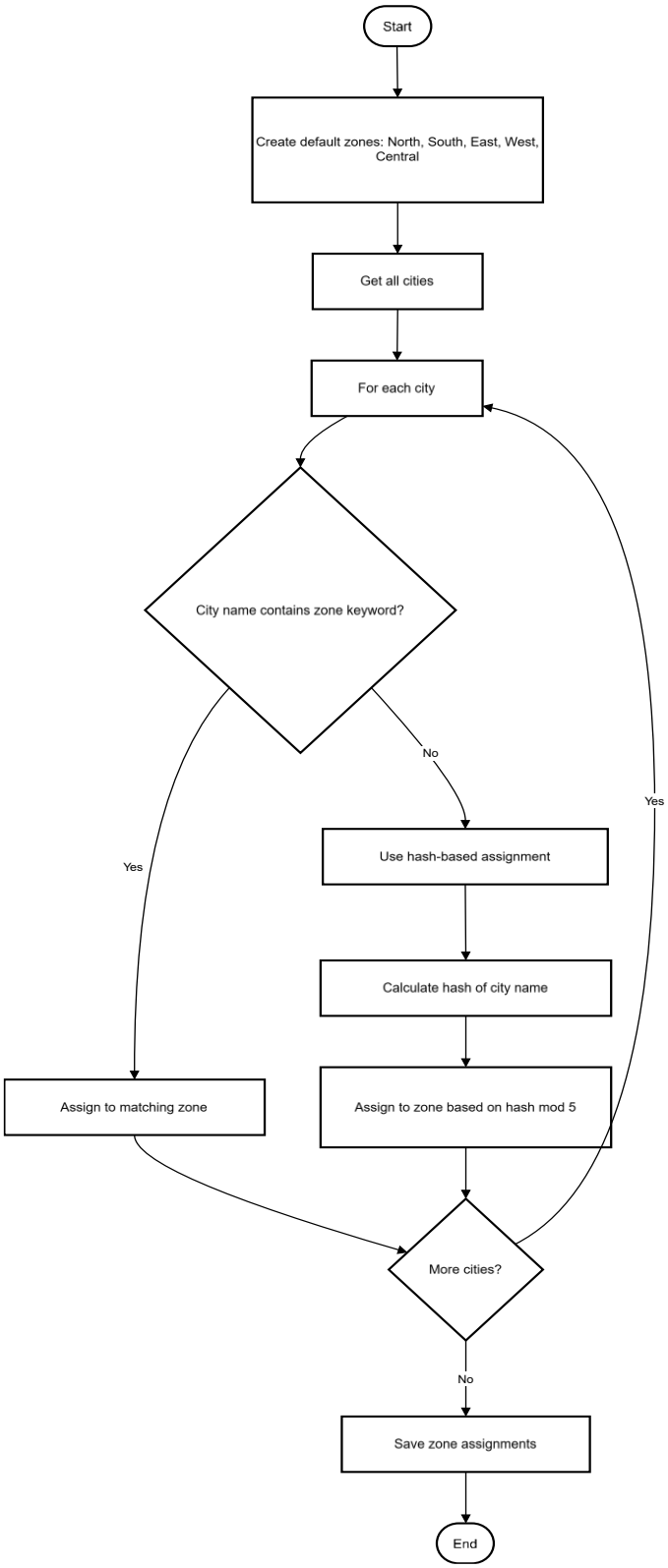

Figure 17 Zone Auto-Assignment Flowchart

4.17 Save/Load Parcels Flowchart

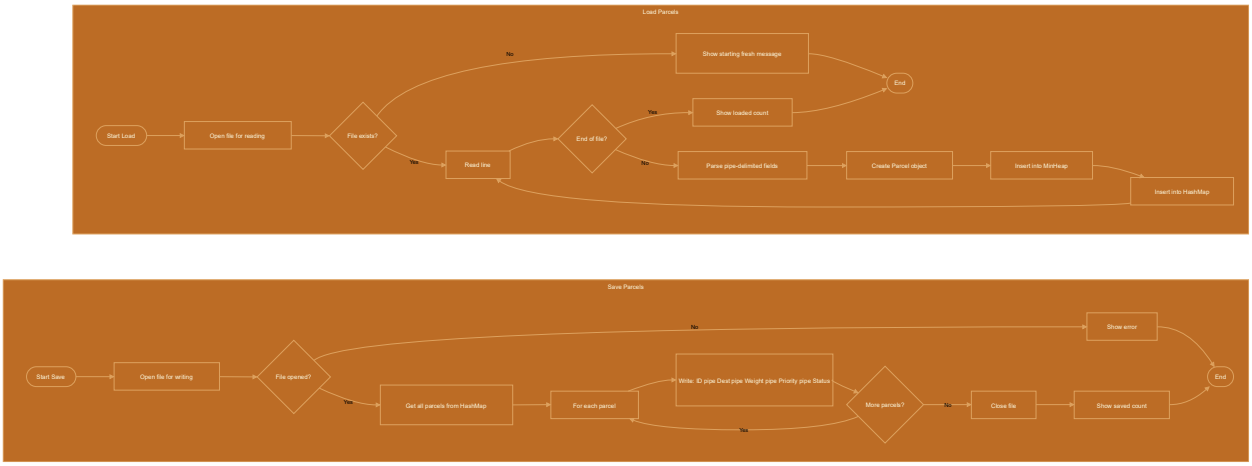

Figure 18 Save/Load Parcels Flowchart

4.18 WebServer Request Handling Flowchart

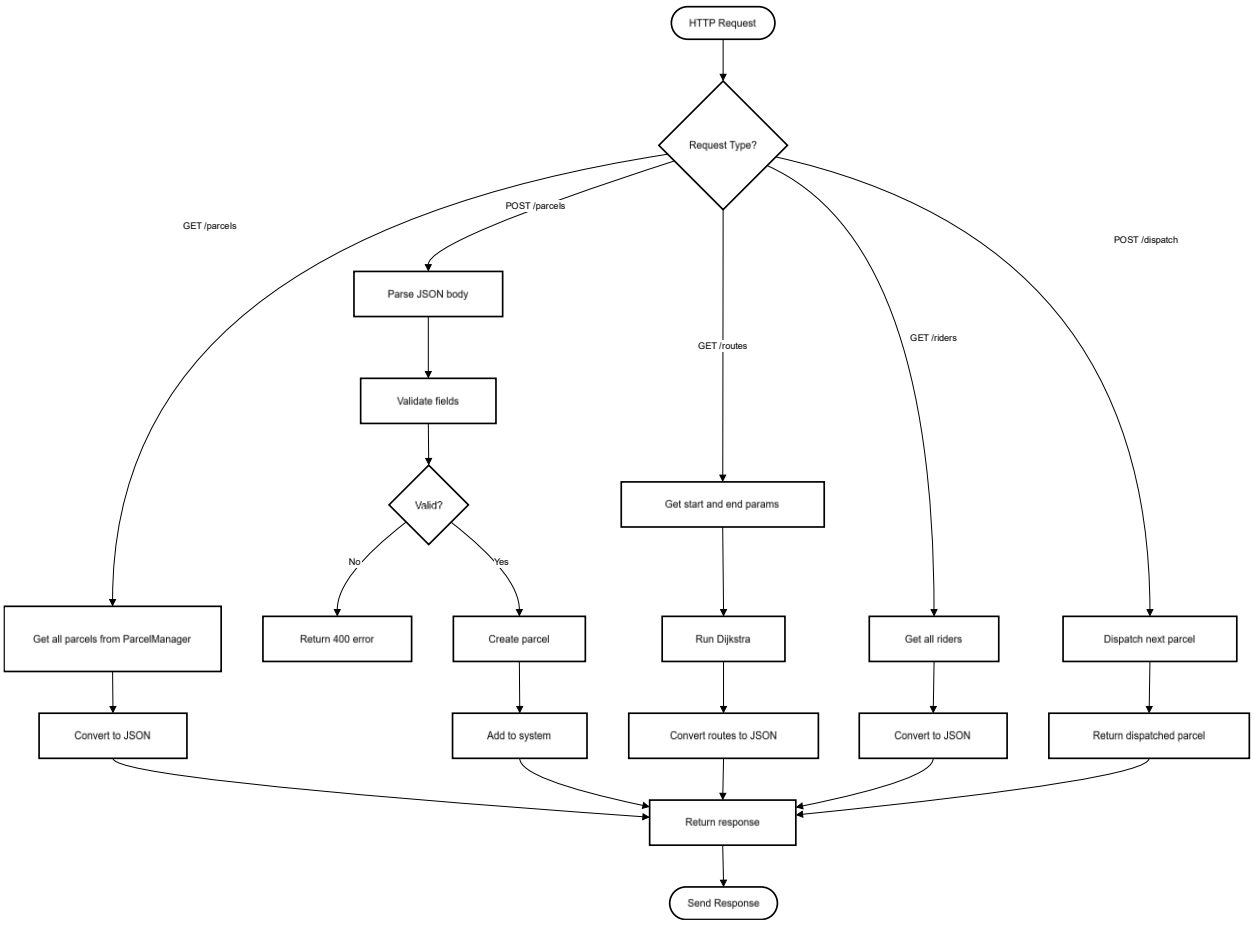

Figure 19 WebServer Request Handling Flowchart

5 System Architecture Diagram

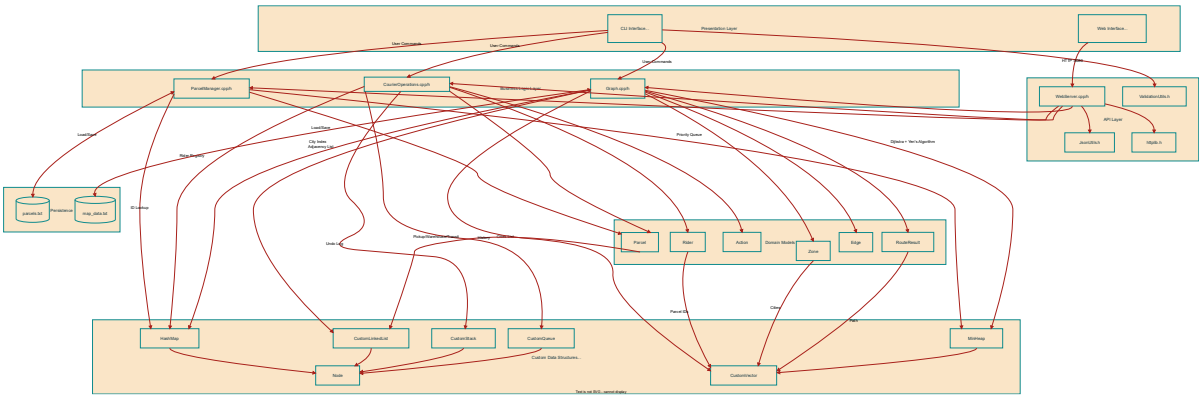

Figure 20 System Architecture Diagram

6 System Architecture

The system follows a modular architecture, where each major functionality is implemented in a separate module. This improves readability, maintainability, and scalability.

Core Modules

- Intelligent Parcel Sorting Module
- Parcel Routing Module
- Parcel Tracking System Module
- Courier Operations Engine Module

7 File and Module Description

7.1 main.cpp

- *Entry point* of the application.
- Supports dual modes:
  - CLI Mode: Menu-driven console interface with 19+ options
  - Web Mode: REST API server (launch with --web flag)
- Coordinates interaction between all modules.
- Loads map data (map_data.txt) and parcel data (parcels.txt) at startup.
- Initializes default riders (Ali, Ahmed, Sara).
- Controls overall system flow including:
  - Parcel management (add, search, dispatch)
  - Delivery operations (attempts, mark delivered, return to sender)
  - Queue management (pickup, warehouse, transit)

- Routing & zone management
- Rider assignment
- Undo functionality

7.2 Parcel.h

- Defines the Parcel class with encapsulated attributes.
- Stores attributes:
  - Parcel ID
  - Priority (1=Overnight, 2=2-Day, 3=Normal)
  - Weight & Weight Category (Light/Medium/Heavy)
  - Destination & Zone
  - Current status (Created, Pending Pickup, In Transit, Delivered, etc.)
  - Delivery attempts tracking (max 3 attempts)
  - Sender address (for return-to-sender)
  - Assigned rider ID
  - History log (using CustomLinkedList)
- Provides getter/setter methods and status constants.
- Includes helper functions: *getWeightCategory(), getWeightCategoryName(), getCurrentTimestamp()*.

7.3 ParcelManager.h / ParcelManager.cpp

- Responsible for parcel storage, sorting, and tracking.
- Uses MinHeap for priority-based parcel queue.
- Uses HashMap for O(1) parcel lookup by ID.
- Supports:
  - Insertion and removal of parcels
  - Priority-based dispatching (dispatchNext())
  - Duplicate ID checking (parcelExists())
  - Status updates throughout parcel lifecycle
  - Re-insertion for undo operations
  - File I/O: saveParcels() / loadParcels() for persistence.

7.4 Graph.h / Graph.cpp

- Implements a weighted Graph representing the delivery network.
- Uses adjacency list representation (CustomVector of CustomLinkedList).
- Uses HashMap for O(1) city name lookup.
- Key features:
  - Nodes represent cities/locations
  - Edges represent roads with weights (distance/cost)

    - Road blocking/unblocking for route simulation
- Routing algorithms:
    - Dijkstra's algorithm for shortest path
    - Yen's K-Shortest Paths algorithm for alternative routes
- Zone Management:
    - Add/manage zones (North, South, East, West, Central)
    - Assign cities to zones
    - Auto-assign zones based on city configuration
- File I/O: loadGraph() / saveGraph() for map persistence. **structures.h**.

7.5 CourierOperations.h / CourierOperations.cpp

- Simulates complete courier workflow using custom data structures.
- Queue Management (using CustomQueue):
    - Pickup Queue: Parcels waiting to be collected
    - Warehouse Queue: Parcels staged for loading
    - Transit Queue: Parcels currently being delivered
- Rider Class: Manages courier capacity, load, zone assignment.
- Rider Management (using HashMap):
    - Add/find riders
    - Find best rider by capacity and zone
    - Assign/release parcels from riders
- Missing Parcel Detection: Track and resolve missing parcels.
- Undo System (using CustomStack):
    - Action class stores operation type and state
    - Supports undo for: add parcel, dispatch, block road, status changes, rider assignment, delivery attempts
    - Full audit log of all actions.

7.6 structures.h

- Contains custom data structure implementations (no STL):
    - Node<T>: Generic node for linked structures
    - CustomVector<T>: Dynamic array with auto-resize
    - CustomLinkedList<T>: Doubly linked list
    - CustomStack<T>: LIFO stack
    - CustomQueue<T>: FIFO queue
    - MinHeap<T>: Priority queue (min-heap)
    - HashMap<K, V>: Hash table with chaining
- Used across all modules for consistency.

- Includes copy constructors and assignment operators.

7.7 ValidationUtils.h

- Provides input validation utilities in Utils namespace.
- Functions include:
  - getIntInput(): Validated integer input with range checking
  - getStringInput(): Non-empty single word input
  - getLineInput(): Full line input with spaces
  - getValidCity(): Validates city exists in graph
  - getAlphanumericInput(): Alphanumeric string validation
  - clearInputBuffer(): Safe input buffer clearing (handles EOF)
- Ensures safe and correct user input handling.
- Prevents invalid data processing in CLI mode.

7.8 WebServer.h / WebServer.cpp

- Implements REST API server using httplib library.
- Endpoints for:
  - Parcel management (CRUD operations)
  - Queue operations (pickup, warehouse, transit)
  - Rider management
  - Route finding and zone management
  - System status
- Serves static files from public/ folder.
- Enables web-based UI interaction.

7.9 JsonUtils.h

- Provides JSON serialization utilities for web API.
- Converts Parcel, Rider, and Route objects to JSON format.
- Used by WebServer for API responses.

7.10 Input/Output Files

- map_data.txt → Stores delivery network:
  - CITY <name>: Define city nodes
  - ROUTE <from> <to> <weight>: Define weighted edges
  - ZONE <name>: Define delivery zones
  - ZONECITY <zone> <city>: Assign city to zone
- parcels.txt → Stores parcel data (pipe-delimited):
  - Format: ID | Destination | Weight | Priority | Status

7.11 public/ (Web UI)

- index.html: Main web interface page
- styles.css: Styling for web UI

8 Data Structures and Algorithms Justification

8.1 Data Structures Used:

MODULE	DATA STRUCTURE	PURPOSE
PARCEL MANAGEMENT	MinHeap	Priority-based parcel dispatching (O(log n) insert/extract)
PARCEL MANAGEMENT	HashMap<int, Parcel>	O(1) parcel lookup by ID
ROUTING	Graph (Adjacency List)	Represents road network with weighted edges
ROUTING	HashMap<string, int>	O(1) city name to index lookup
PARCEL TRACKING	CustomLinkedList	Maintains parcel status history log
PICKUP OPERATIONS	CustomQueue	FIFO processing of pickup requests
WAREHOUSE OPERATIONS	CustomQueue	FIFO staging of parcels for loading
TRANSIT OPERATIONS	CustomQueue	FIFO tracking of in-transit parcels
UNDO SYSTEM	CustomStack	LIFO audit log for operation reversal
RIDER MANAGEMENT	HashMap<int, Rider>	O(1) rider lookup and management
MISSING PARCELS	HashMap<int, Parcel>	Track potentially missing parcels
ZONE MANAGEMENT	HashMap<string, Zone>	Zone storage and lookup
CITY-ZONE MAPPING	HashMap<string, string>	O(1) city to zone resolution
DYNAMIC STORAGE	CustomVector	Auto-resizing arrays for flexible storage

8.2 Algorithms Used:

ALGORITHM	MODULE	PURPOSE
DIJKSTRA'S ALGORITHM	Graph.cpp	Find shortest path between cities
YEN'S K-SHORTEST PATHS	Graph.cpp	Find K alternative routes for delivery
HEAP OPERATIONS (HEAPIFY)	structures.h	Maintain priority queue property
HASH FUNCTION (CHAINING)	structures.h	Collision resolution in HashMap
LINEAR SEARCH	CustomVector	Find elements in dynamic arrays
DOUBLY LINKED LIST OPS	CustomLinkedList	Efficient insert/delete at both ends

8.3 Complexity Analysis:

OPERATION	TIME COMPLEXITY	DATA STRUCTURE
ADD PARCEL	O(log n)	MinHeap
DISPATCH NEXT (PRIORITY)	O(log n)	MinHeap
FIND PARCEL BY ID	O(1) average	HashMap
SHORTEST PATH	O((V+E) log V)	Graph + MinHeap
K-SHORTEST PATHS	O(K * V * (V+E) log V)	Yen's Algorithm
ENQUEUE/DEQUEUE	O(1)	CustomQueue
PUSH/POP (UNDO)	O(1)	CustomStack
ADD TO HISTORY	O(1)	CustomLinkedList

9 Conclusion

The Intelligent Courier Logistics Engine successfully simulates a real-world courier system using core Data Structures and Algorithms. Through modular design, efficient data handling, and algorithmic decision-making, the project demonstrates strong problem-solving skills and practical application of DSA concepts in C++.