**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**

[1 Introduction 4](#_Toc218183950)

[2 Project Overview 4](#_Toc218183951)

[3 UML Diagram 5](#_Toc218183952)

[3.1 Class Diagram 5](#_Toc218183953)

[4 Flowcharts 6](#_Toc218183954)

[4.1 Main Application Flowchart 6](#_Toc218183955)

[4.2 Add Parcel Flowchart 7](#_Toc218183956)

[4.3 Dispatch Parcel Flowchart 8](#_Toc218183957)

[4.4 Dijkstra’s Shortest Path Algorithm Flowchart 9](#_Toc218183958)

[4.5 K-Shortest Paths Algorithm Flowchart 10](#_Toc218183959)

[4.6 Record Delivery Attempt Flowchart 11](#_Toc218183960)

[4.7 Assign Parcel to Rider Flowchart 12](#_Toc218183961)

[4.8 Find Best Rider Flowchart 13](#_Toc218183962)

[4.9 Undo Operation Flowchart 14](#_Toc218183963)

[4.10 Block/Unblock Road Flowchart 15](#_Toc218183964)

[4.11 Parcel Lifecycle Flowchart 16](#_Toc218183965)

[4.12 Queue Operations (Pickup/Warehouse/Transit) Flowchart 17](#_Toc218183966)

[4.13 MinHeap Insert Operation Flowchart 17](#_Toc218183967)

[4.14 MinHeap Extract Min Operation Flowchart 18](#_Toc218183968)

[4.15 HashMap Insert Operation Flowchart 19](#_Toc218183969)

[4.16 Zone Auto-Assignment Flowchart 20](#_Toc218183970)

[4.17 Save/Load Parcels Flowchart 21](#_Toc218183971)

[4.18 WebServer Request Handling Flowchart 21](#_Toc218183972)

[5 System Architecture Diagram 22](#_Toc218183973)

[6 System Architecture 22](#_Toc218183974)

[7 File and Module Description 22](#_Toc218183975)

[7.1 main.cpp 22](#_Toc218183976)

[7.2 Parcel.h 23](#_Toc218183977)

[7.3 ParcelManager.h / ParcelManager.cpp 23](#_Toc218183978)

[7.4 Graph.h / Graph.cpp 23](#_Toc218183979)

[7.5 CourierOperations.h / CourierOperations.cpp 24](#_Toc218183980)

[7.6 structures.h 24](#_Toc218183981)

[7.7 ValidationUtils.h 25](#_Toc218183982)

[7.8 WebServer.h / WebServer.cpp 25](#_Toc218183983)

[7.9 JsonUtils.h 25](#_Toc218183984)

[7.10 Input/Output Files 25](#_Toc218183985)

[7.11 public/ (Web UI) 26](#_Toc218183986)

[8 Data Structures and Algorithms Justification 26](#_Toc218183987)

[8.1 Data Structures Used: 26](#_Toc218183988)

[8.2 Algorithms Used: 27](#_Toc218183989)

[8.3 Complexity Analysis: 27](#_Toc218183990)

[9 Conclusion 27](#_Toc218183991)

**Table of Figures**

[Figure 1 Class Diagram 5](#_Toc218183992)

[Figure 2 Main Application Flowchart 6](#_Toc218183993)

[Figure 3 Add Parcel Flowchart 7](#_Toc218183994)

[Figure 4 Dispatch Parcel Flowchart 8](#_Toc218183995)

[Figure 5 Dijkstra's Shortest Path Algorithm Flowchart 9](#_Toc218183996)

[Figure 6 K-Shortest Path Algorithm Flowchart 10](#_Toc218183997)

[Figure 7 Record Delivery Attempt Flowchart 11](#_Toc218183998)

[Figure 8 Assign Parcel to Rider Flowchart 12](#_Toc218183999)

[Figure 9 Find Best Rider Algorithm 13](#_Toc218184000)

[Figure 10 Undo Operation Flowchart 14](#_Toc218184001)

[Figure 11 Block/Unblock Road Flowchart 15](#_Toc218184002)

[Figure 12 Parcel Lifecycle Flowchart 16](#_Toc218184003)

[Figure 13 Queue Operations (Pickup/Warehouse/Transit) Flowchart 17](#_Toc218184004)

[Figure 14 MinHeap Insert Operation 17](#_Toc218184005)

[Figure 15MinHeap Extract Min Operation Flowchart 18](#_Toc218184006)

[Figure 16 HashMap Insert Operation Flowchart 19](#_Toc218184007)

[Figure 17 Zone Auto-Assignment Flowchart 20](#_Toc218184008)

[Figure 18 Save/Load Parcels Flowchart 21](#_Toc218184009)

[Figure 19 WebServer Request Handling Flowchart 21](#_Toc218184010)

[Figure 20 System Architecture Diagram 22](#_Toc218184011)

# 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++.

# 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.

# UML Diagram

## Class Diagram



Figure 1 Class Diagram

# Flowcharts

## Main Application Flowchart



Figure Main Application Flowchart

## Add Parcel Flowchart

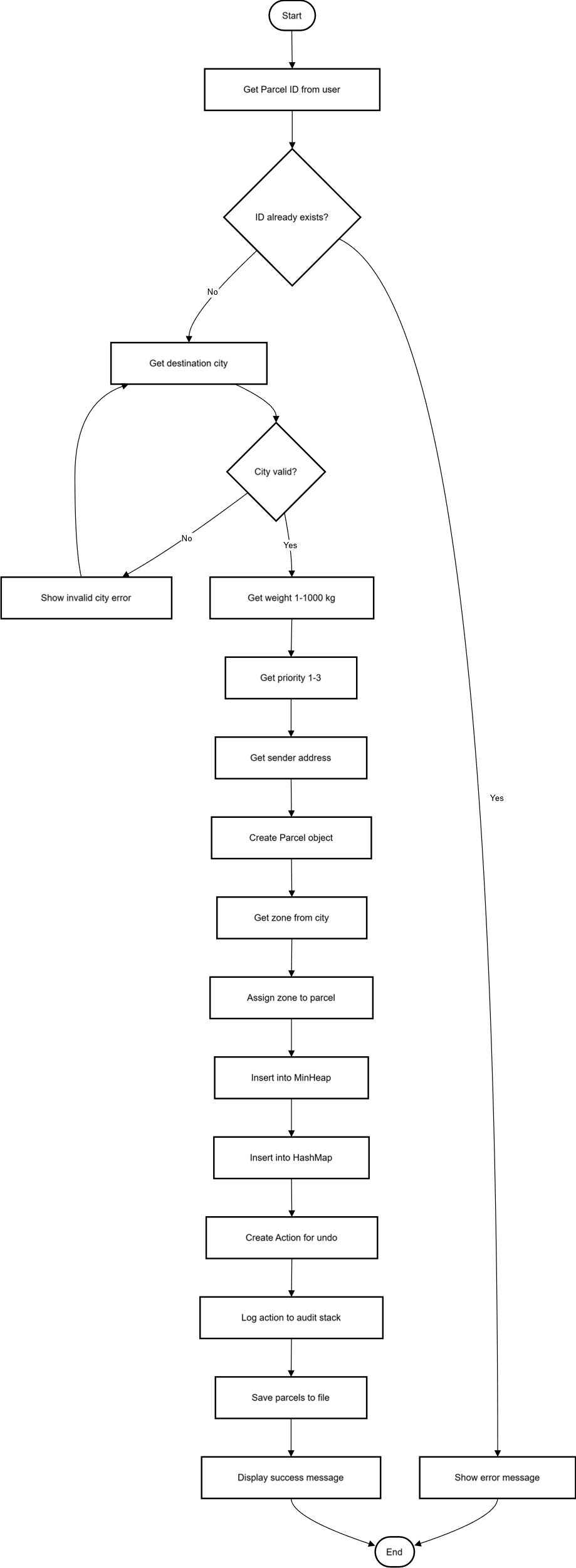


Figure 3 Add Parcel Flowchart

## Dispatch Parcel Flowchart

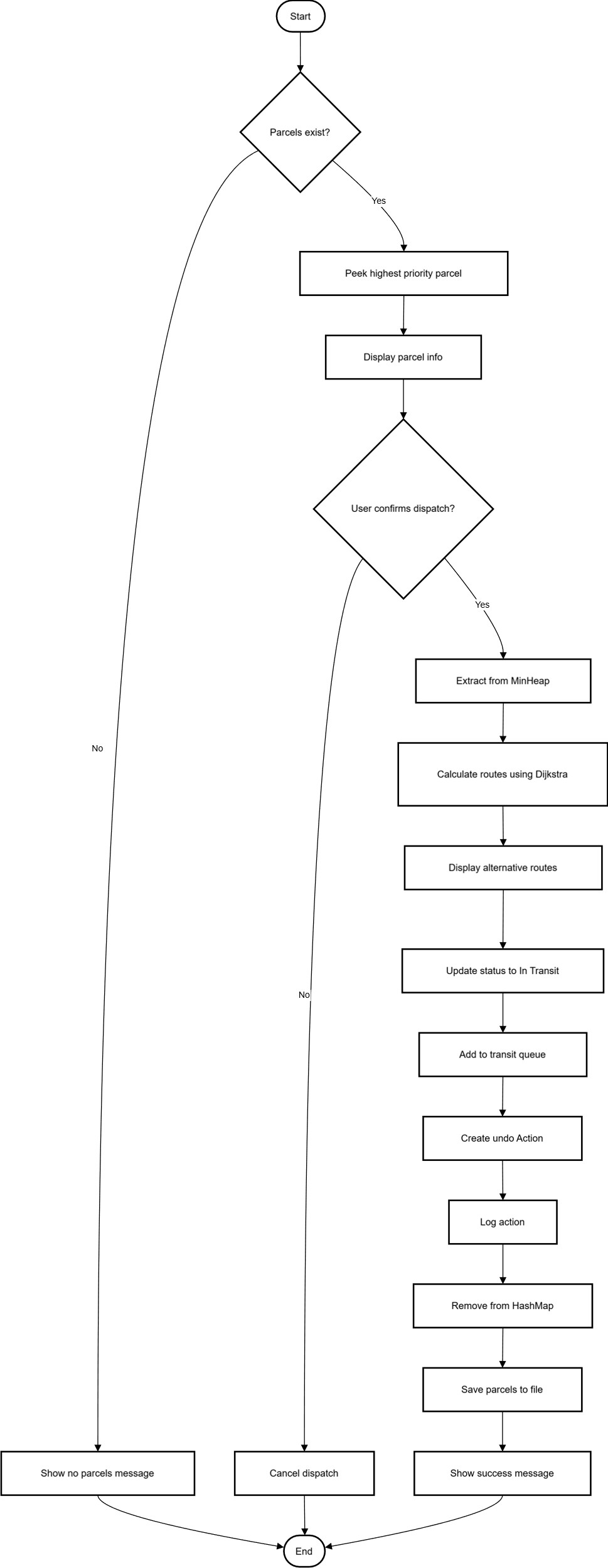


Figure Dispatch Parcel Flowchart

## Dijkstra’s Shortest Path Algorithm Flowchart

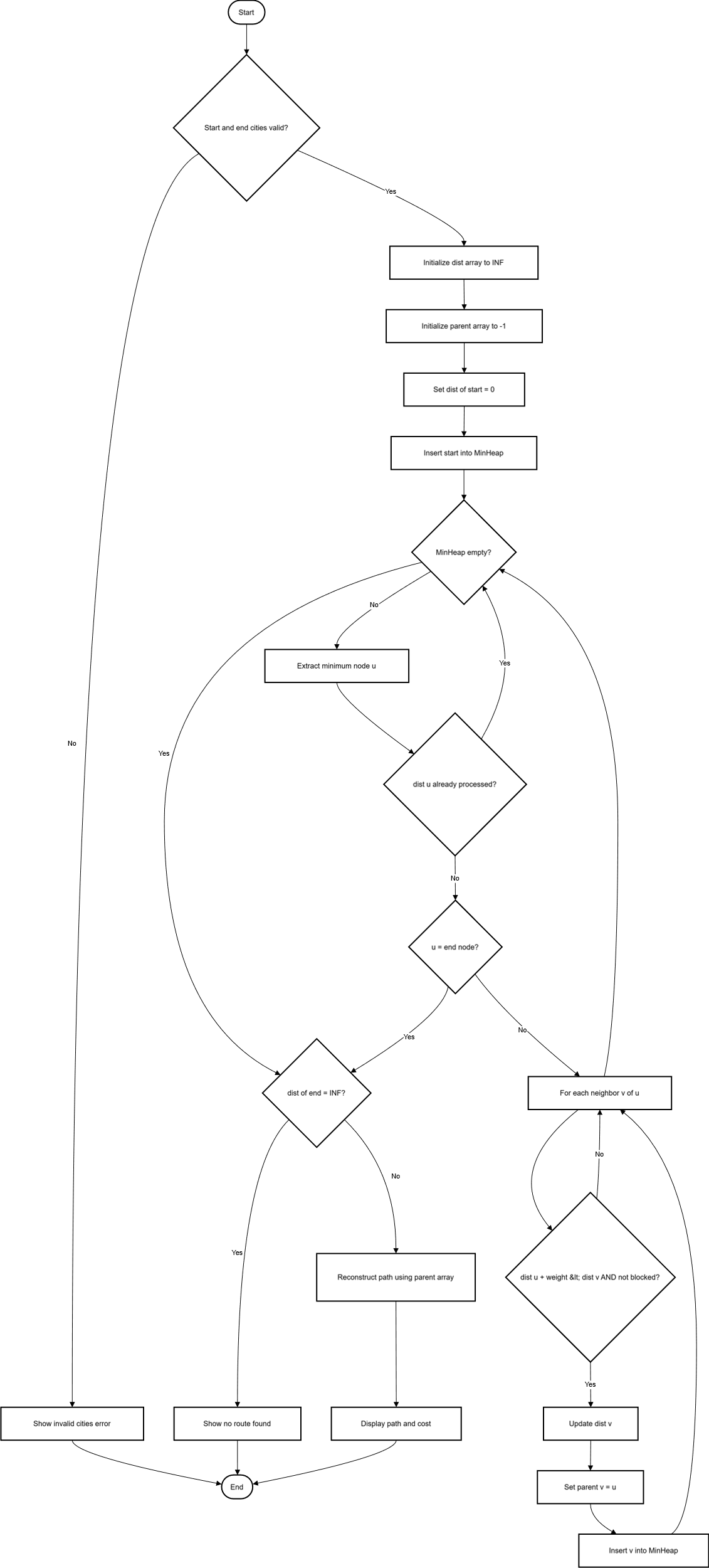


Figure 5 Dijkstra's Shortest Path Algorithm Flowchart

## K-Shortest Paths Algorithm Flowchart

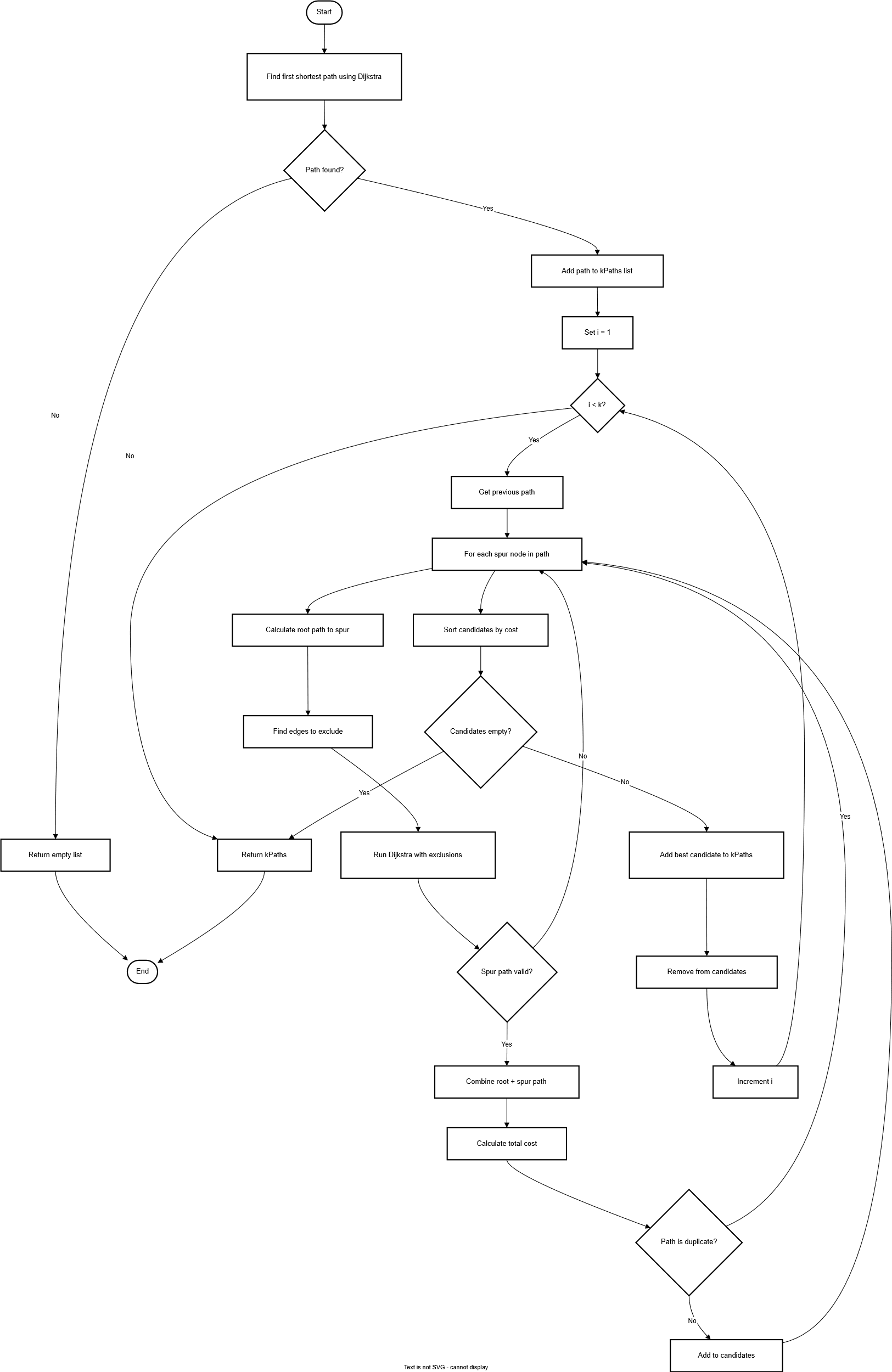


Figure 6 K-Shortest Path Algorithm Flowchart

## Record Delivery Attempt Flowchart

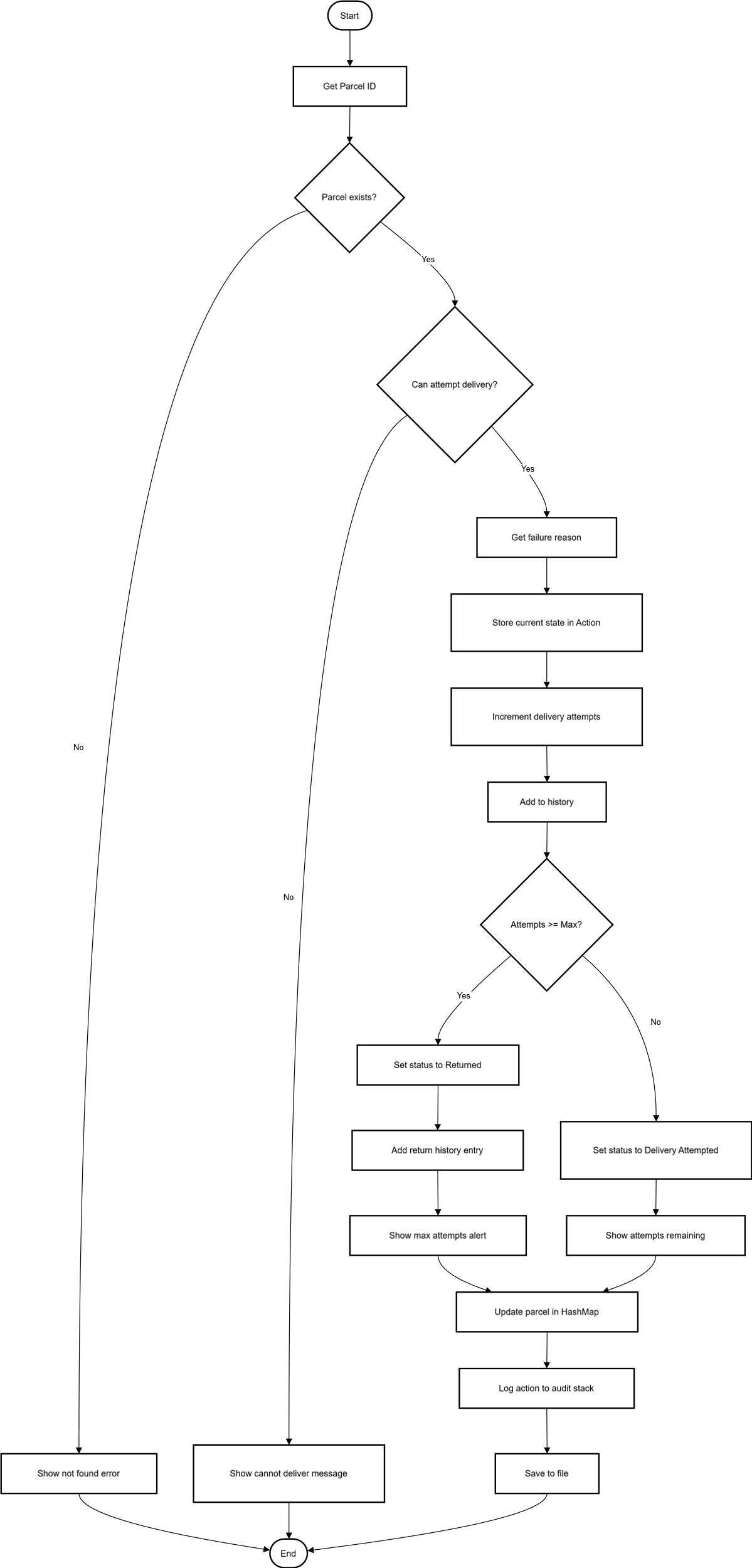


Figure Record Delivery Attempt Flowchart

## Assign Parcel to Rider Flowchart

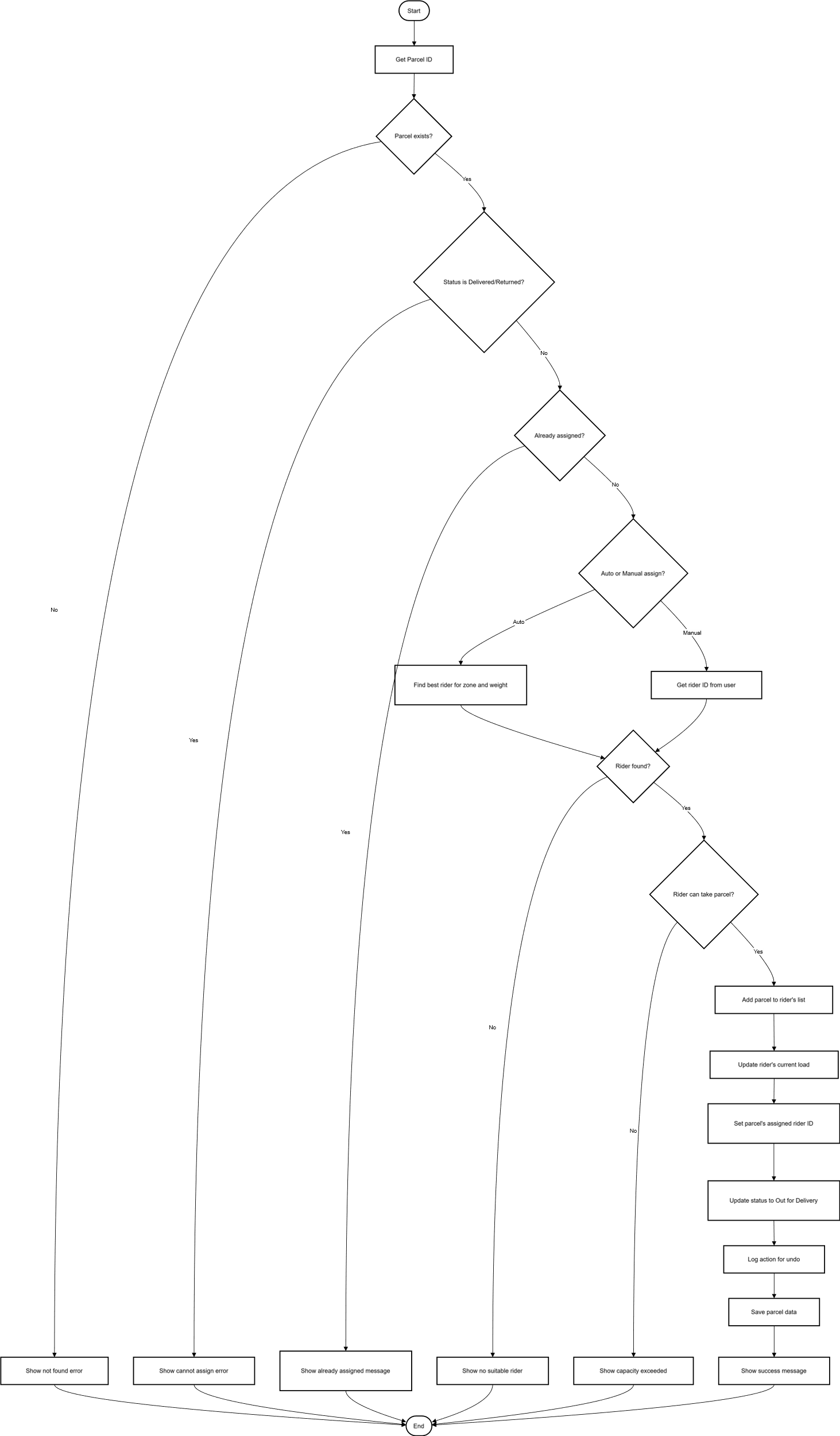


Figure 8 Assign Parcel to Rider Flowchart

## Find Best Rider Flowchart

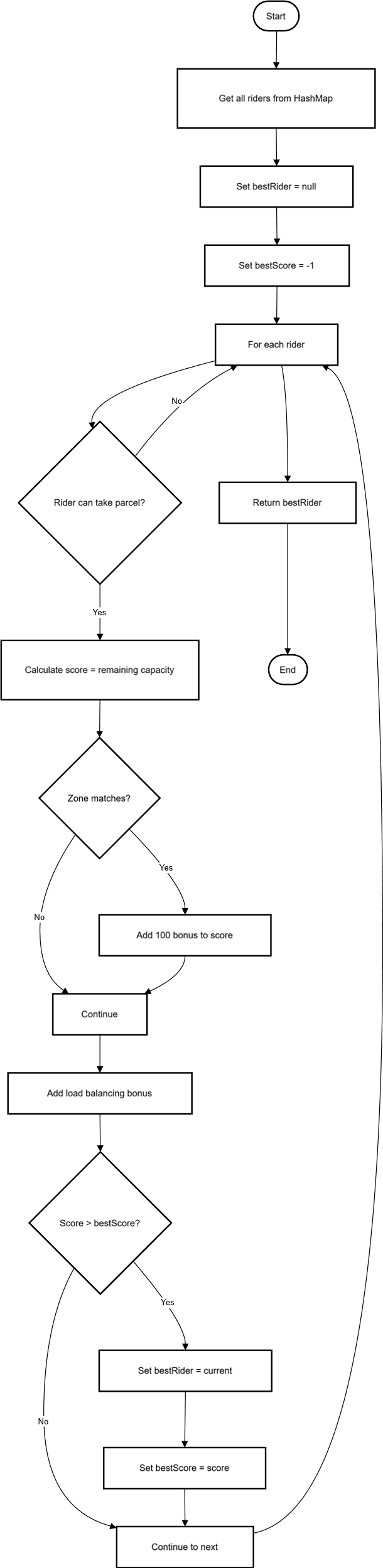


Figure 9 Find Best Rider Algorithm

## Undo Operation Flowchart

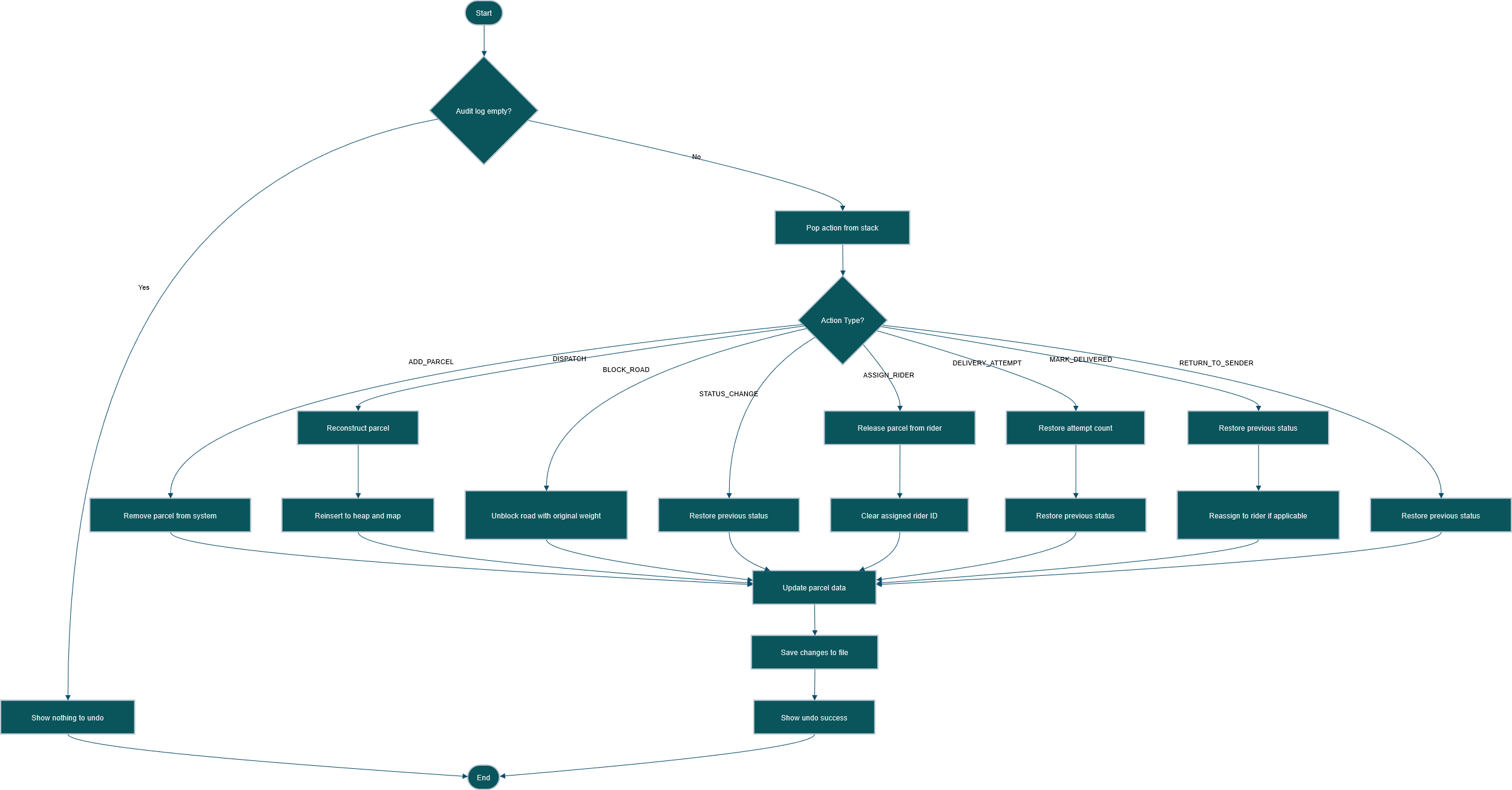


Figure 10 Undo Operation Flowchart

## Block/Unblock Road Flowchart

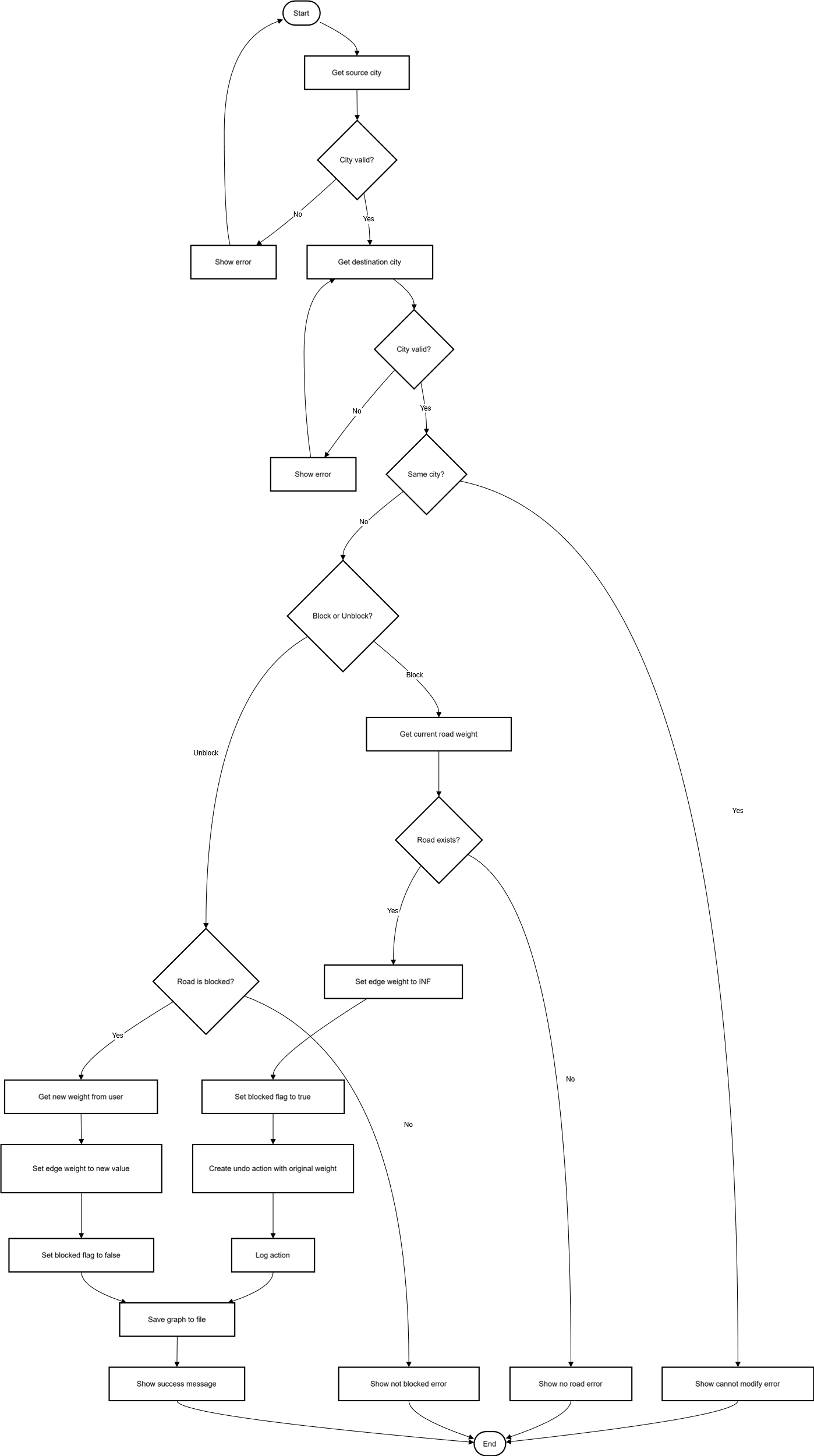


Figure Block/Unblock Road Flowchart

## Parcel Lifecycle Flowchart

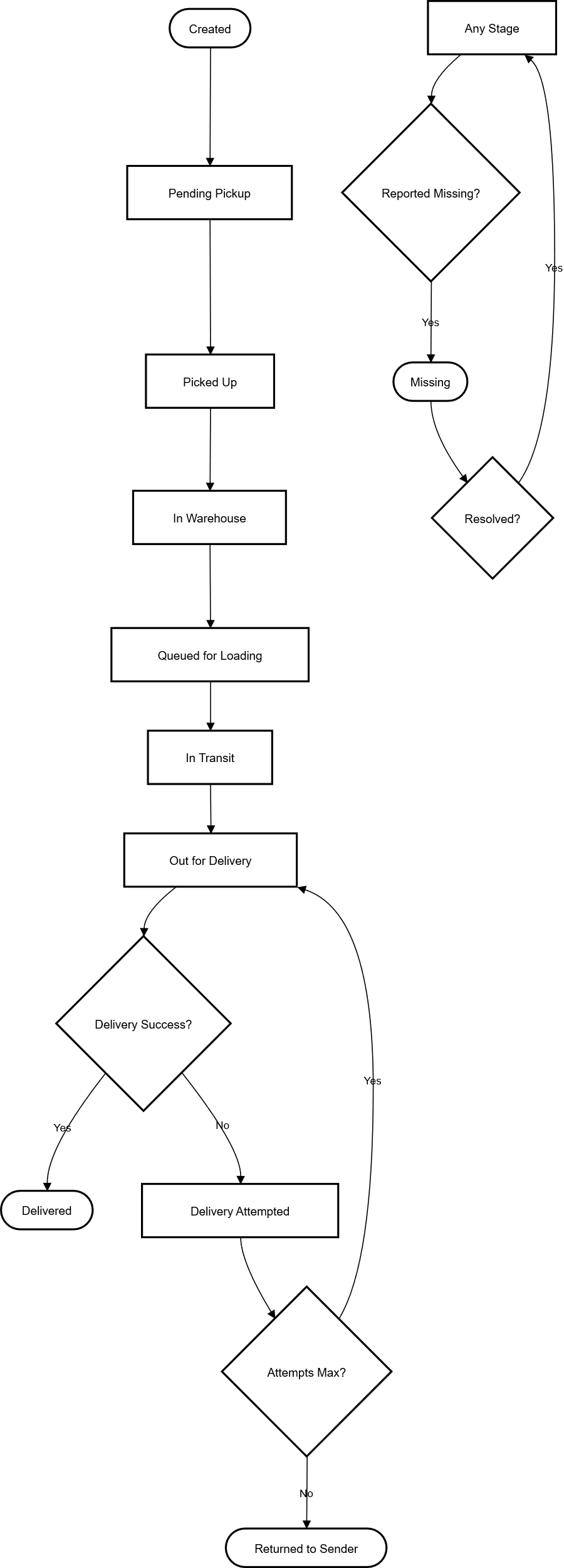


Figure Parcel Lifecycle Flowchart

## Queue Operations (Pickup/Warehouse/Transit) Flowchart

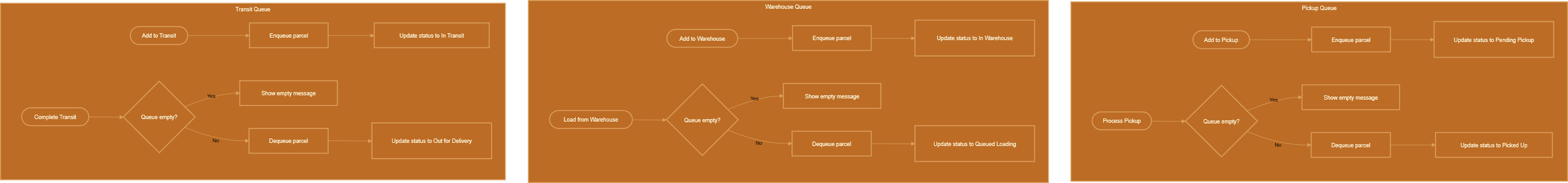


Figure 13 Queue Operations (Pickup/Warehouse/Transit) Flowchart

## MinHeap Insert Operation Flowchart

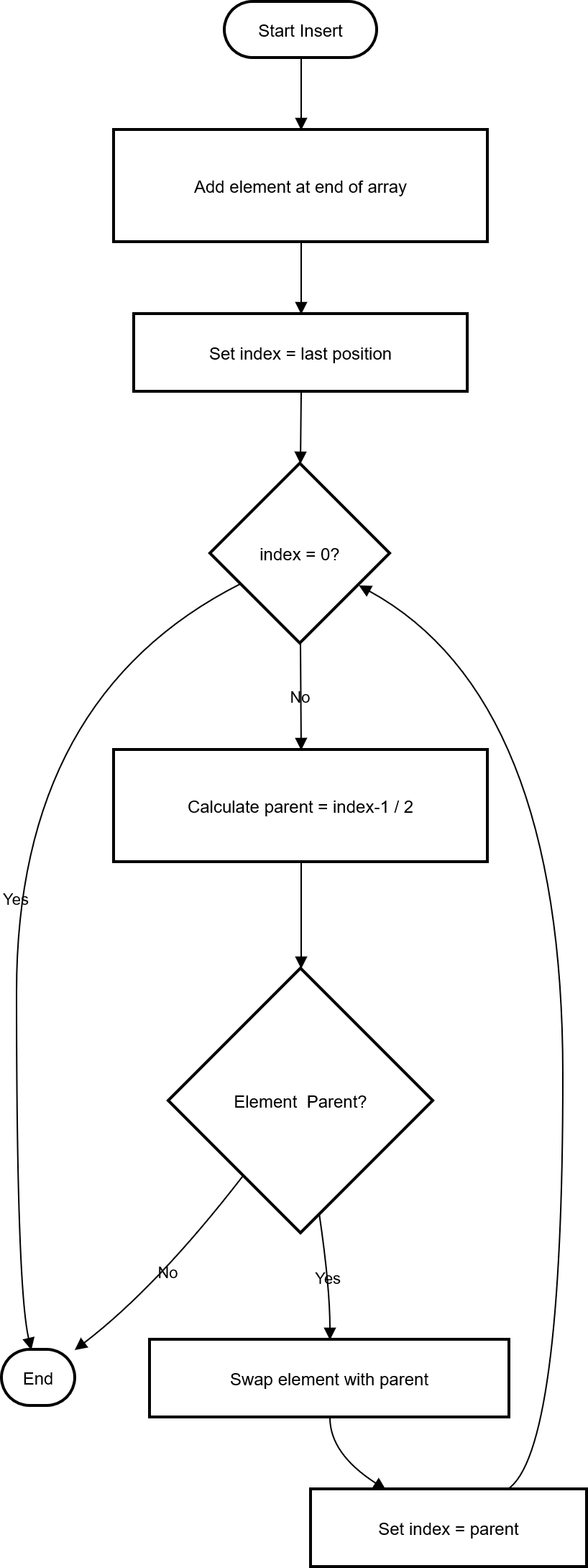


Figure 14 MinHeap Insert Operation

## MinHeap Extract Min Operation Flowchart

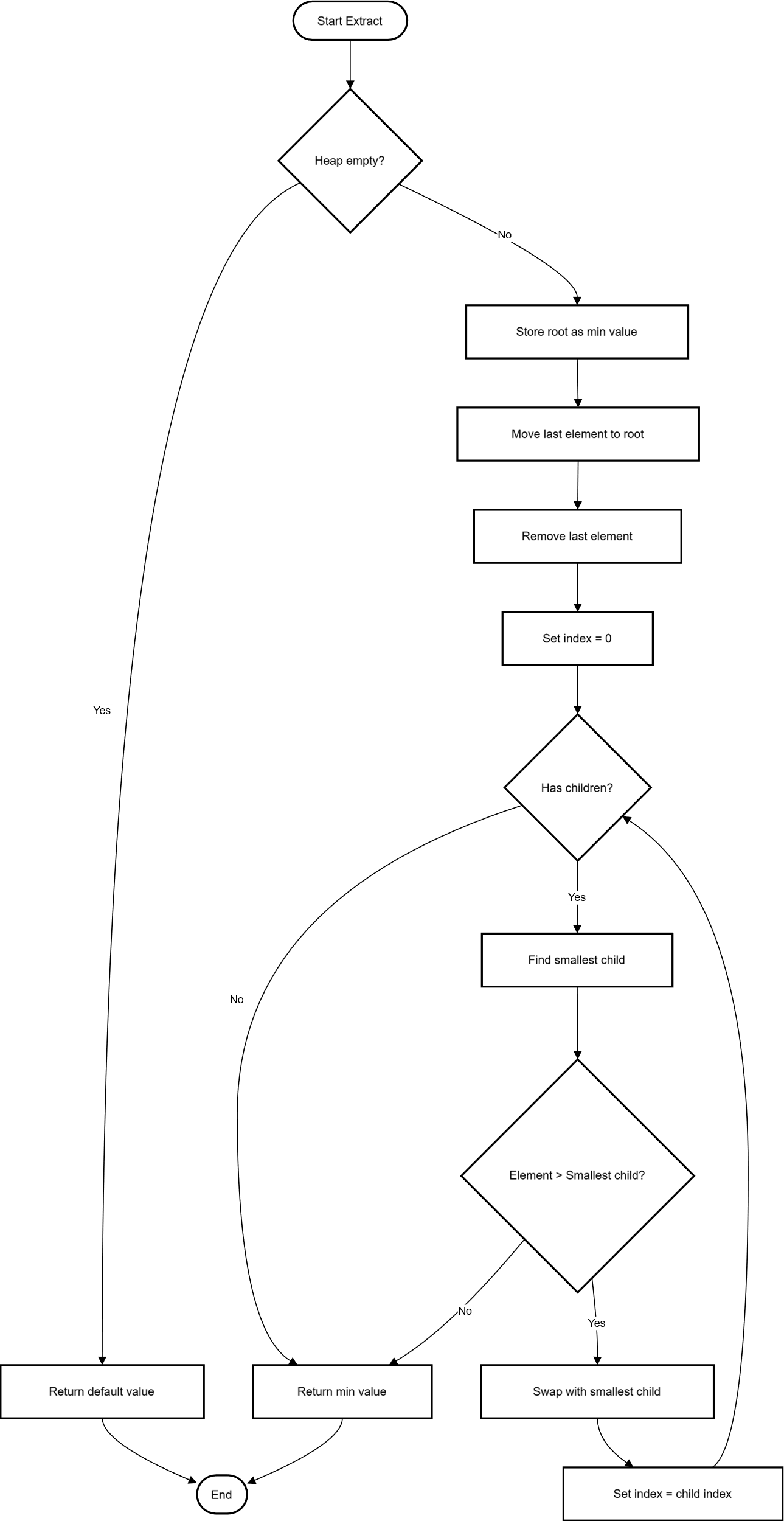


Figure 15MinHeap Extract Min Operation Flowchart

## HashMap Insert Operation Flowchart

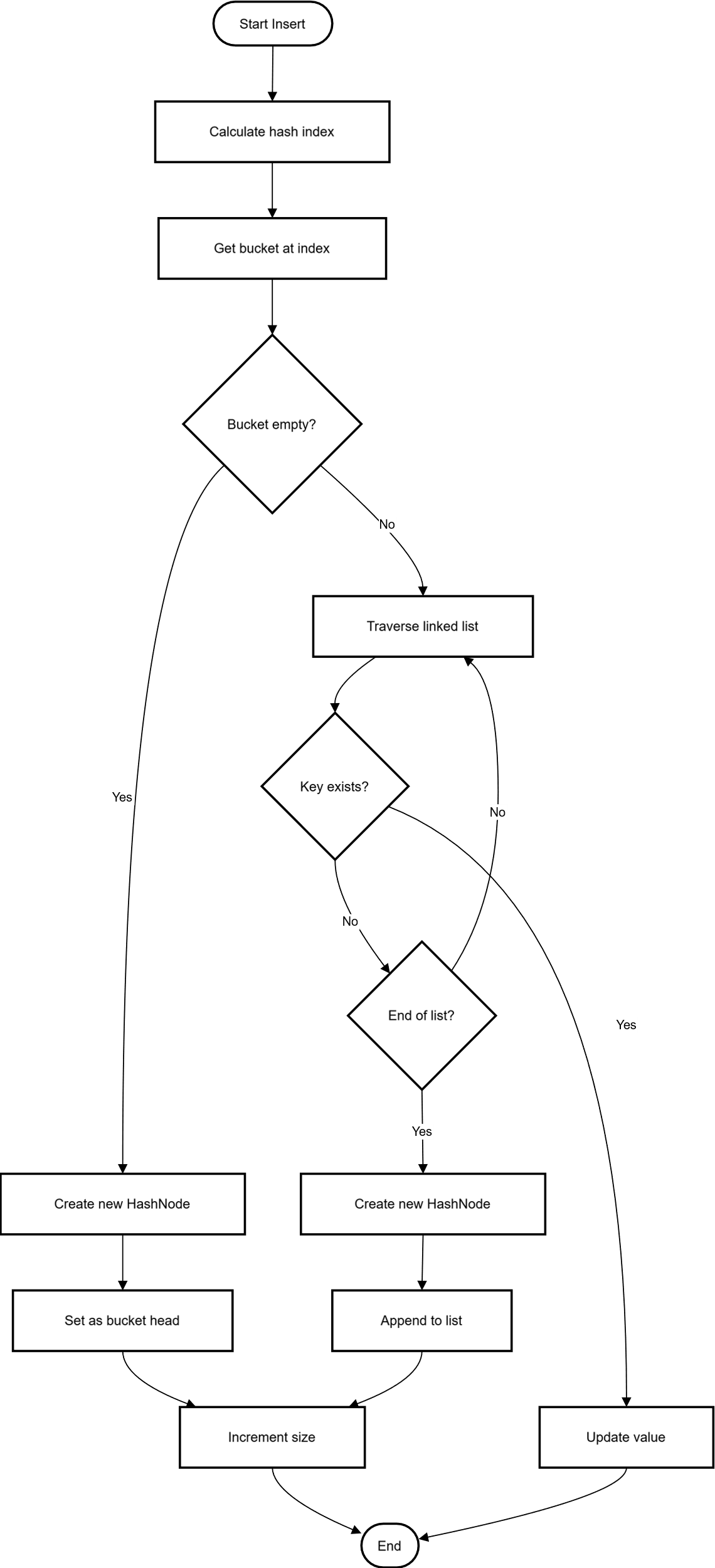


Figure 16 HashMap Insert Operation Flowchart

## Zone Auto-Assignment Flowchart

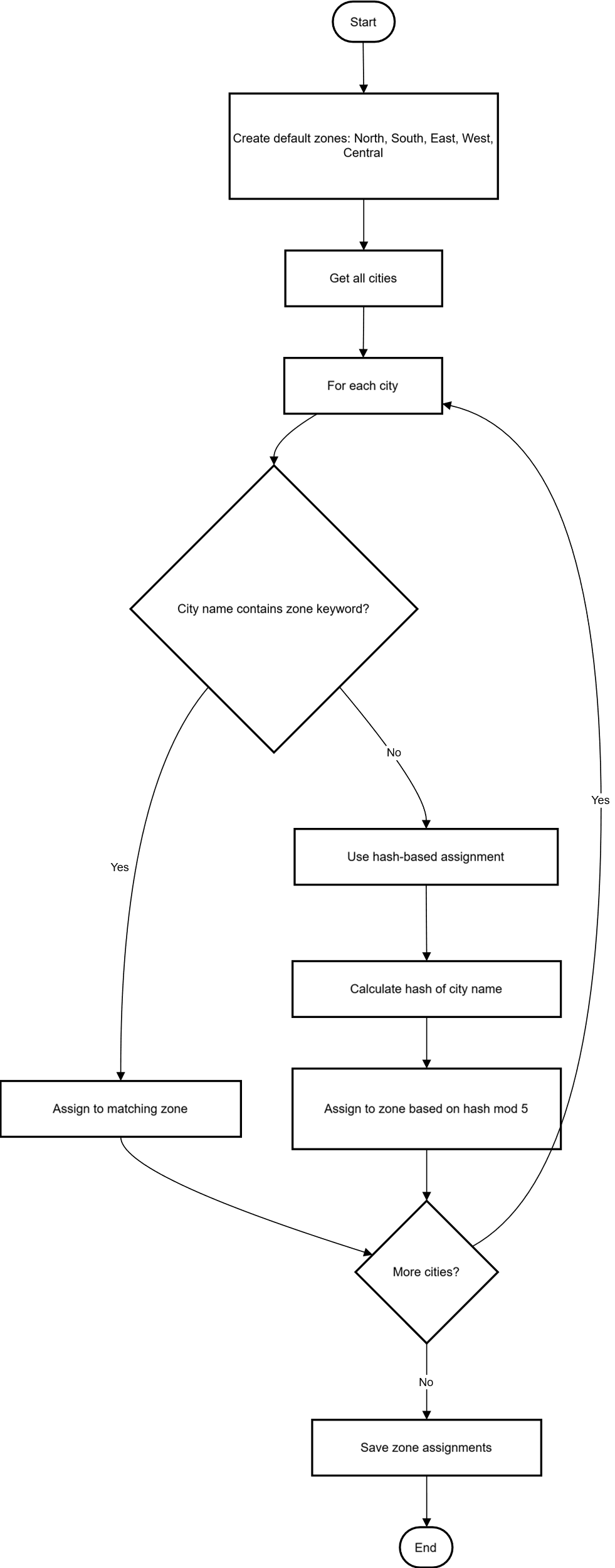


Figure 17 Zone Auto-Assignment Flowchart

## Save/Load Parcels Flowchart

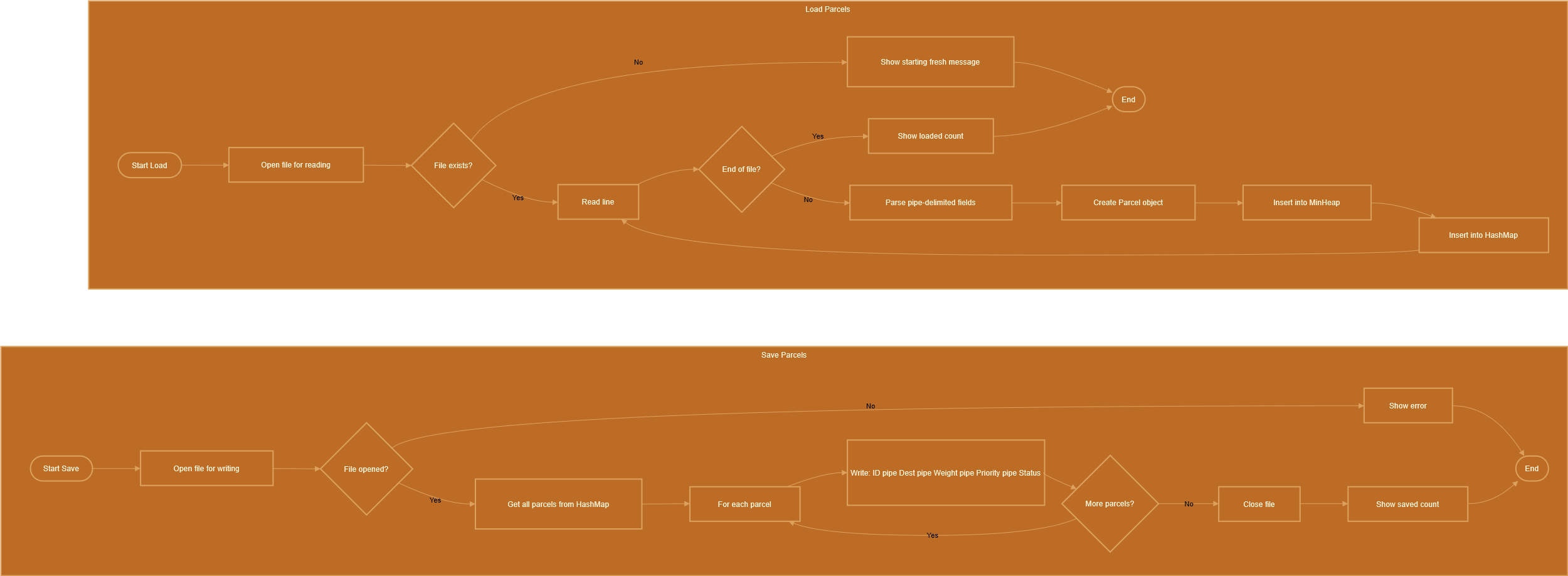


Figure 18 Save/Load Parcels Flowchart

## WebServer Request Handling Flowchart

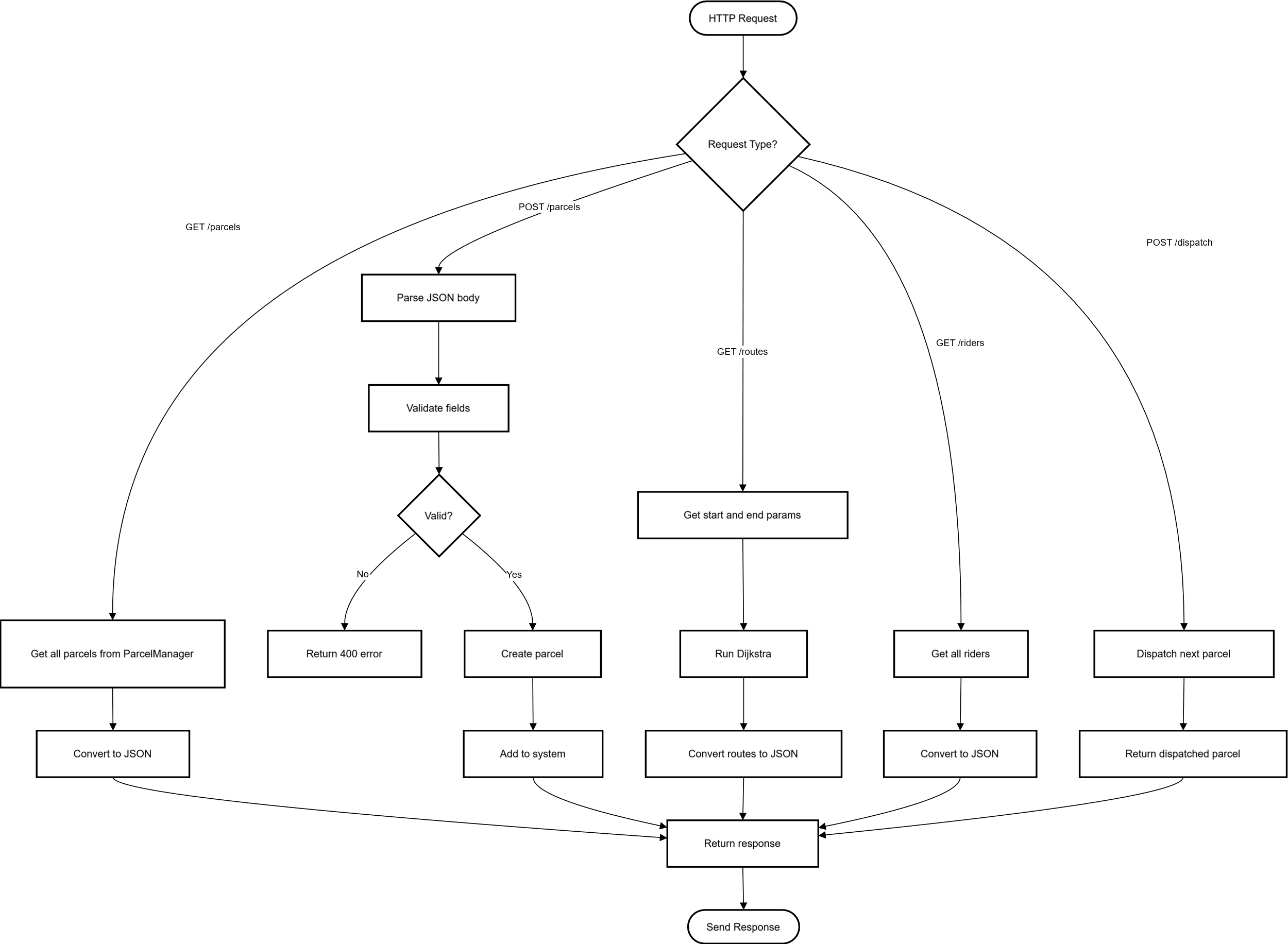


Figure 19 WebServer Request Handling Flowchart

# System Architecture Diagram



Figure 20 System Architecture Diagram

# 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

# File and Module Description

## 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

## 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()*.

## 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.

## 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**.

## 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.

## 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.

## 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.

## 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.

## JsonUtils.h

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

## 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

## public/ (Web UI)

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

# Data Structures and Algorithms Justification

## 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 |

## 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 |

## 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 |

# 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++.