COMP1002 Final Assignment
Optimizing Urban Parcel Logistics Using Data Structures and Algorithms

School of Electrical Engineering, Computing and Mathematical Sciences
Curtin University

April 25, 2025

Introduction

CityDrop Logistics is expanding its operations in a metropolitan area and requires a suite of inter-connected software modules to optimize its delivery network. As part of their systems development team, your role is to develop a modular delivery optimization system using data structures and algorithms. The core focus will be on runtime efficiency, memory optimization, and clean algorithmic design.

The system is divided into four logically interconnected modules:

1. Graph-Based Route Planning: To determine the most efficient delivery paths.
2. Hash-Based Customer Lookup: To retrieve customer information quickly.
3. Heap-Based Parcel Scheduling: To prioritize deliveries based on urgency and customer priority.
4. Sorting of Delivery Records: To organize the records at the end of the day using Merge and Quick Sort.

Each module has defined inputs and outputs, and together they form a workflow where data flows from one component to another to simulate a real-time logistics system. The associated tasks for each module are described in detail in the following sections.

Module 1: Graph-Based Route Planning Optimization

Problem Description

As part of CityDrop Logistics' expansion plan, you are tasked with developing a delivery route optimization module. This module will allow the system to determine:

- Which delivery zones (intersections or hubs) are accessible from a specific hub,

- Whether any inefficient loops exist in the delivery network,
- The fastest route from a central warehouse to each delivery hub.

The city road network is modeled as a **weighted, undirected graph**, where:

- Each node represents a hub or city intersection.
- Each edge represents a road segment, with weight indicating the estimated travel time in minutes.

Objectives

You will:

- Implement an efficient graph using an **adjacency list**.
- Apply and test two traversal algorithms and one algorithmto identify the shortest path:
  - **Breadth-First Search (BFS)** for identifying reachable zones.
  - **Depth-First Search (DFS)** for detecting cycles (loops) in the delivery network.
  - **Pick one algorithm and implement** for computing shortest travel times from the warehouse to other hubs.

Instructions

1. Design and implement a **Graph class** using an adjacency list. Ensure:
   - It supports dynamic insertion of nodes and weighted edges.
   - Each edge holds a travel time.
2. Implement the following algorithms:
   - **BFS(source)**:
     - Input: Source hub (e.g., 'A')
     - Output: List of reachable hubs and their level (i.e., minimum number of hops)
     - Application: Identifies all delivery zones accessible from the warehouse.
   - **DFS(source)**:
     - Input: Source hub (e.g., 'A'); the function should attempt to traverse the whole graph.
     - Output: Report whether any cycles (loops) exist.
     - Application: Helps CityDrop identify inefficient routes that could cause delivery delays.
   - **Shortest Path Algorithm (source)**:
     - Input: Source hub (e.g., 'A')

  - Output: Shortest path and total travel time from source to each hub.
  - Application: Ensures parcels are routed along the fastest paths. You should cite the source of the algorithm and implement it without using any built-in functions.

3. Construct a sample graph with at least:
   - 8 nodes (A–H),
   - 10–12 edges with travel times (e.g., 'A $\rightarrow$ B (5 minutes)'),
   - At least one cycle and one disconnected node to test robustness.
4. Demonstrate the output of all three algorithms clearly on your test graph.

Expected Output

You must show the following:

- The structure of your graph.
- BFS output: List of reachable hubs from a source, with levels.
- DFS output: Whether any cycles exist; if yes, highlight nodes involved.
- Shortest Path output: Shortest path and cost from a source to every other node.

Module 2: Hash-Based Customer Lookup

Problem Description

The goal of this module is to implement a hash table that allows for retrieval of customer information based on customer ID. This lookup will be crucial for scheduling deliveries, as it provides the customer level required to compute delivery priority. Each delivery has an associated customer. This module should have the functionality to improve customer support and real-time order tracking by allowing fast access to the most up-to-date delivery statuses using customer identifiers.

Tasks

- Implement a hash table using chaining or linear probing. Use a good simple hash function (e.g., modulo-based) for indexing, and explain your choice in the documentation.
- Store fields: Customer ID, Name, Address, Priority Level (1–5), Delivery Status e.g., 'Delivered', 'In Transit', 'Delayed')
- Provide functions to:
  - Insert new customer

- Search customer by ID
- Delete a customer
- Demonstrate collision handling with examples.

Notes:

- Make sure your hash table can handle at least 50 customer records without significant performance degradation.
- You may implement your own LinkedList class if using chaining.
- Clearly document how your implementation works and provide a complexity analysis.

Module 3: Heap Based Parcel Scheduling

Problem Description

This module is responsible for prioritizing and scheduling deliveries based on customer information and estimated delivery time. The hash table implemented in Module 2 provides the necessary customer details—such as Priority Level and Delivery Status—which are used in conjunction with the estimated delivery time from Module 1 (route planning) to compute the scheduling priority for each delivery request.

Input:

- Delivery requests containing:
  - Customer ID (used to fetch priority level from the hash table)
  - Estimated delivery time (from shortest path computation)
- Customer information retrieved via hash table (from Module 2), including:
  - Priority Level (1 = highest, 5 = lowest)
  - Delivery Status (use only for active deliveries)

Priority Calculation:

$$\text{Priority} = (6 - P) + \frac{1000}{T} \tag{1}$$

where:

- P is the **Customer Priority Level** (1 = highest, 5 = lowest),
- T is the **Estimated Delivery Time** in minutes.

The formula ensures that high-priority customers (lower numerical levels) and shorter delivery times result in higher scheduling priority.

Output:

- A list of deliveries sorted by priority using a Heap structure.

Tasks:

- Implement a Heap to schedule parcels based on their computed priority. Justify your choice of selecting max/ min heap structure in the report.
- Provide the following functions:
  - `insert(request)`: Adds a new delivery request to the heap.
  - `extract_priority()`: Retrieves and removes the highest-priority delivery.
- Show the state of the heap after each insertion and extraction operation.
- Log each request's priority calculation based on data from Module 1 and Module 2.

Notes:

- Use the heap class following the lecture slides.
- Provide detailed print logs for heap operations and priority calculations.
- Ensure that your implementation handles dynamic updates efficiently.

Module 4: Sorting Delivery Records

Problem Description

At the end of each day, delivery records are stored and need to be sorted for reporting purposes. This module has the sorting functionality that enhances the efficiency of generating end-of-day reports, verifying delivery performance, and analyzing operational metrics.

Requirements

1. Implement both Merge Sort Quick Sort using the algorithms from lecture slides.
2. Apply sorting to datasets of 100, 500, and 1000 parcels based on estimated delivery time.
3. Measure and compare execution time
4. Analyze performance on different input conditions: random, nearly sorted, reversed.

Expected Output

- Sorted list output
- Comparative analysis graphs
- Reflection on algorithm choice based on input nature

Submission Guidelines

All students must submit the following components to successfully complete the Final Assignment. No group working and submissions are allowed for final assignment. Ensure that your submission meets the formatting and content expectations outlined below.

Submission Deadline

- **Due Date:** Thursday, 29 May 2025, 11:59 PM AWST. Additional one week for students with Curtin Access Plan (CAP).
- Late submissions without approved extension are not accepted.

Submission Format

- All files must be submitted as a **single ZIP file** named: `COMP1002_Assignment_YourStudentID.zip`
- Upload via the Blackboard assignment submission link.
- Include a README with module-wise instructions to run/test each module.
- Include a technical report in PDF format.
- Include sample input files. Consider different case scenarios.

Source Code Requirements

- You may use Python/ Java. You can reuse the code from practical submissions and self cite. For shortest path algorithm, cite the resource.
- Avoid using any built-in functions. 100% penalty will be applied for using built-ins on any part.
- All code must be modular, with meaningful variable and function names
- Include inline comments and a header block for each file.
- No AI generated code is accepted. 100% penalty will be applied if found.
- Include errors and exceptions check in each module.

Technical Report

Submit a technical report in PDF format containing;

1. **Overview:** Summary of your solution and modular design.
2. **Data Structures Used:** Justification for using graph, hash table, heap, sorting algorithm.
3. **Module Integration:** Include a flowchart or diagram/ UML illustrating the system's architecture, detailing the components' interactions.
4. **Algorithm Complexity:** Time and space analysis for each module.
5. **Sample Output:** Screenshot or formatted output for each module (use the sample input provided).
6. **Reflection:** Reflection on what you learned.
7. **Limitations & Assumptions:** Any known issues or simplifications.
8. Use clear section headers, numbered pages, and consistent formatting throughout.

Code Demonstration

- **Demo Date:** 3rd June - 10th June
- **Demo Time:** Book a 20-minute slot from Blackboard (link will be posted in Blackboard with venue details)
- You must be ready to:
  - Run each module live
  - Explain your code logic and structure
  - Answer basic questions on algorithms and complexity
- Demo is **compulsory** and without the demonstration the submission will not be considered completed.

Academic Integrity

- All work must be your own.
- Plagiarism or code sharing will be reported and penalized according to university policy.
- You may discuss ideas with peers but do not share code.

Support

Feel free to reach out to your unit coordinator if you have any concern.

Marking Guide (Total: 40 Marks)

Students must demonstrate not only correct implementation but also clarity, modular design, and integration across components. The detailed marking rubric is attached with the submission link.

Criteria	Details	Marks
Graph-Based Route Planning	Graph construction, BFS, DFS, and shortest path implementation	**8**
	– Adjacency list-based Graph class design	2
	– BFS: reachable nodes and level output	2
	– DFS: correct cycle detection	2
	– Shortest path: from-scratch implementation and result accuracy	2
Hash-Based Customer Lookup	Hash table for customer data using chaining or linear probing	**6**
	– Insert, search, and delete operations working correctly	3
	– Collision handling demonstrated and explained	1
	– Effective hash function and scalability for 50+ customers	2
Heap-Based Parcel Scheduling	Heap design and parcel prioritization integration	**6**
	– Insert and extract operations correctly implemented	2
	– Correct priority score calculation	2
	– Heap state changes and logs presented clearly	2
Sorting Delivery Records	Merge Sort and Quick Sort application and evaluation	**6**
	– Sorting correctness and clear output presentation	3
	– Discussion of performance	3
Technical Report	Clear explanation, structure, and completeness with no AI-generated/plagiarized content	**8**

	– Overview and justification of data structures used	2
	– Flowchart/UML with module interaction explanation	2
	– Time and space complexity analysis + student reflection	2
	– Screenshots, assumptions, limitations	2
Code Quality & Compliance	Code style, documentation, restrictions compliance	**6**
	– Modular, well integrated, readable, documented code, no built-ins	2
	– Output trace with sample inputs, edge cases tested	2
	– Proper error handling and exception safety	2

Note:

- **Demonstration is mandatory.** Non-attendance results in zero mark for the entire assignment.
- **Plagiarism or AI-generated content:** Any evidence will result in **zero marks** for affected sections or the entire assignment.