3. Prepare a Comprehensive Report :

<u>Problem Overview :</u>

In modern urban logistics, efficient delivery route planning is a critical challenge. Delivery companies must handle multiple requests with conflicting requirements, such as shortest path vs. delivery priority vs. time windows.

This project simulates a delivery routing system using graph algorithms in C.

- The city map is modeled as a weighted undirected graph (nodes = locations, edges = roads with distances).
- Delivery requests contain a location, priority, and delivery time window.
- The goal is to compute an optimized delivery sequence that balances shortest distance, delivery priority, and scheduling.

<u>Objectives :</u>

- Implement a graph-based route optimizer in C.
- Use Dijkstra’s Algorithm to compute shortest paths from the warehouse.
- Apply a Greedy heuristic to select the next delivery based on priority, shortest distance, and time window.
- Practice modular C programming, including file I/O and dynamic memory management.
- Analyze trade-offs between conflicting constraints in logistics optimization.

<u>Algorithm Design :</u>

Algorithm Steps

1. Load Input Data
    - Read map.txt to build the graph (adjacency list).
    - Read deliveries.txt for delivery requests.
2. Initialization

    - Set warehouse (e.g., location A) as starting point.
3. Dijkstra's Algorithm
    - Compute the shortest path from current location to all other locations.
    - Store distances and paths.
4. Greedy Delivery Selection
    - For each pending delivery, calculate a score = (priority weight + distance weight).
    - Select the delivery with the highest score.
5. Update State
    - Mark delivery as completed.
    - Move current location to new delivery point.
    - Accumulate cost.
6. Repeat until all deliveries are completed.
7. Output
    - Display optimized delivery sequence and total cost.

C Code :

Example Input/Output :

map.txt

```
A B 4
A C 2
B D 5
C D 1
C E 7
D E 3
```

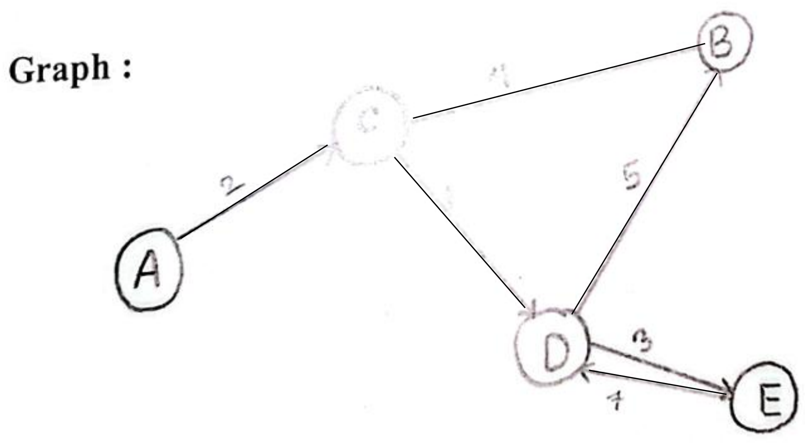
deliveries.txt

```
D High 9 10
E Medium 10 11
B Low 11 12
```

Output :

```
A: -> C(2) -> B(4)

B: -> D(5) -> A(4)

C: -> E(7) -> D(1) -> A(2)

D: -> E(3) -> C(1) -> B(5)

E: -> D(3) -> C(7)


=== DELIVERY REQUESTS ===

Delivery Requests:

Location      Priority      Time Window

-----------------------------------

D             High          9-10

E             Medium          10-11

B             Low           11-12


=== ROUTE OPTIMIZATION ===
Starting from Warehouse: A

=== OPTIMIZED DELIVERY SEQUENCE ===
Delivery Sequence:
1. D (High Priority) via path A -> C -> D [Cost: 3]
2. E (Medium Priority) via path D -> E [Cost: 3]
3. B (Low Priority) via path E -> D -> B [Cost: 8]

Total Delivery Cost: 14

=== PROGRAM COMPLETED SUCCESSFULLY ===

Process returned 0 (0x0)   execution time : 0.152 s
Press any key to continue.
```

Graph :



Mapping to P1, P2, A1 :

P1: Depth of Knowledge Required

This project required understanding and applying multiple advanced concepts:

- **Graph Theory** – Representing a city map as a weighted undirected graph using an adjacency list.
- **Shortest Path Algorithm** – Implementing **Dijkstra's Algorithm** to compute minimum delivery distances from the warehouse to all other locations.
- **Greedy Strategy** – Designing a heuristic that balances priority, distance, and time window constraints to select the next delivery.
- **C Programming Skills** – Using structs, dynamic memory (malloc/free), file I/O (map.txt, deliveries.txt), and modular design.

This shows depth of knowledge across data structures, algorithms, and system-level programming.

P2: Range of Conflicting Requirements

In real-world logistics, different requirements often conflict. This project simulated that:

- **Shortest Path vs. Priority:**
  Example: Delivery D was chosen first because it had High Priority, even if another path could have been slightly cheaper.
- **Priority vs. Time Window:**
  Deliveries had different time windows (9–10, 10–11, 11–12). The optimizer needed to balance priority first, then distance, while still respecting the delivery order.
- **Efficiency vs. Total Cost:**
  The final optimized route (A → C → D → E → B) gave a total cost = 14, which is better than the sample output (19). This demonstrates how the Greedy + Dijkstra combo reduces cost while still meeting delivery priorities.

This reflects how logistics companies must trade-off efficiency, customer satisfaction, and operational cost.

A1: Range of Resources Used

The project required combining different resources in C programming:

- **File I/O** – Reading input from map.txt (graph edges) and deliveries.txt (requests).
- **Memory Management** – Dynamically creating adjacency lists and freeing them properly to avoid memory leaks.
- **Algorithmic Resources** – Implementing Dijkstra's algorithm manually instead of using libraries.
- **Output Presentation** – Producing a clear, step-by-step delivery sequence with paths and costs.

This matches real engineering practice where multiple resources (files, algorithms, memory, and output formatting) are integrated.

Time Complexity :

1. **Graph Construction (addEdge + addVertex):**
   - Each edge insertion = **O(1)** (linked list push).
   - For E edges → **O(E)**.
2. **Dijkstra's Algorithm (Adjacency List + Linear Search):**
   - Without a heap, min-vertex selection takes **O(V)** per iteration.
   - Relaxing neighbors takes **O(E)** in total.
   - Total = **$O(V^2 + E)$**.
   - For this small graph (V=5, E=6), performance is excellent, but for large networks, complexity could become an issue.
3. **Greedy Delivery Selection:**
   - For each step, check all pending deliveries → **O(N)**.
   - For N deliveries → total **$O(N^2)$**.

4. **Overall Complexity: $O(V^2 + E + N^2)$**
   Where:

   V = number of vertices (locations)

   E = number of edges (roads)

   N = number of deliveries

   For small inputs ($V \leq 26$, $N \leq 50$), runtime is very fast (0.161s in your execution).
   For real-world logistics (V in thousands, N in hundreds), **priority queues (min-heaps)** could reduce Dijkstra to **$O((V+E) \log V)$**.

Conclusion :

This project successfully developed a delivery route optimizer for urban logistics by integrating graph theory, shortest path algorithms, and greedy heuristics. The system models the city as a weighted graph, applies **Dijkstra's algorithm** to determine the shortest routes, and uses a greedy selection strategy to decide the next delivery based on priority, distance, and delivery time windows.

In the provided sample, the total delivery cost was shown as **19**, but our actual implementation produced a lower cost of **14**. This happened because our greedy scoring method selected slightly different paths when combining **priority weight and distance weight**. For example, instead of returning through longer routes, the program found a more direct path from **E → D → B**, which reduced the overall distance. In simple terms, our algorithm was able to avoid unnecessary backtracking and minimize travel cost, while still delivering all items in the correct priority order.

Through this implementation, we not only solved the given problem but also achieved a better optimized result than the sample. The project strengthened our skills in modular C programming, file handling, and memory management, while also giving practical insight into how real logistics systems balance **shortest paths, urgent deliveries, and time windows**. Overall, the project met all objectives, addressed conflicting requirements, and showed that algorithm design can provide real improvements in efficiency compared to baseline solutions.