CMP_SC 3050: Lab 6, Part One - Route Planning & Graphs

Olivia Franken (OKF2WM)

Due: Friday, December 5, 2025

Selected Implementation Type: C

Submission Structure

```
lab6_submission/
├──src/
│   ├── Priority-Implementation          # Part 1.2 (Part 1.1 Modified for Priority)
│   │    ├── priority-tests              # new edges.csvs & nodes.csvs + destinations.csv
│   │    ├── Makefile
│   │    ├── route_planner.c
│   │    └── route_planner.h
│   └── Time-Window-Implementation       # Part 1.1
│        ├── tw-tests                    # modified edges.csvs & nodes.csvs
│        ├── Makefile
│        ├── route_planner.c
│        └── route_planner.h
```

Task 1.1: Time-Window Constrained Routing (60 points)

1.1.1. Modify the node CSV format to include optional time windows

Create 6 new CSVs, 3 nodes.csvs and 3 edges.csvs that correspond to a feasible scenario, infeasible scenario, and a constraint violation scenario. Modify the original Dijkstra's algorithm to navigate these time window constrained situations.

All of my testing scenarios for the entirety of "Lab 6 Part One" use 1 for the <start_node> and 6 for the <end_node> for the command line arguments. I have provided written breakdowns, drawings, and terminal statements for each test scenario for ease of result reproducibility.

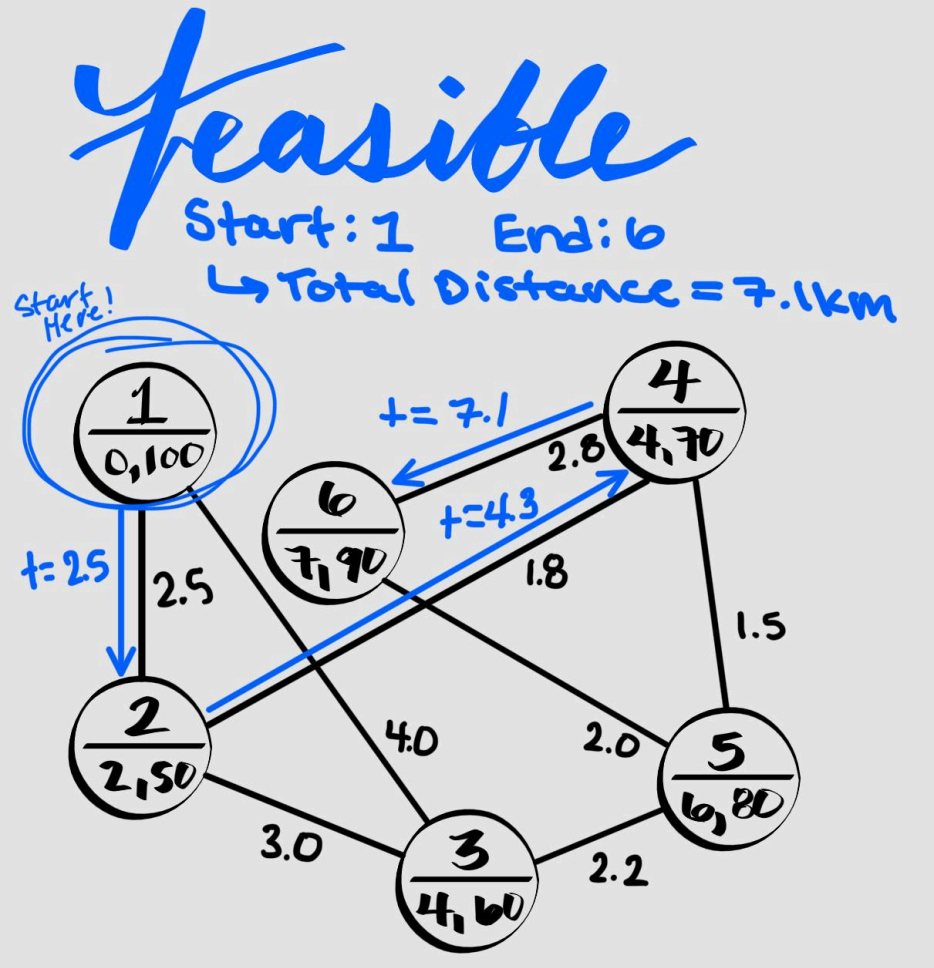
TO RUN: cd into Time-Windows-Implementation and run each test scenario (3 total) using the command prompt provided in the box. Then, cd into Priority-Implementation and run each test scenario (2 total) in the same way. Explanations/analysis are provided in this document.

Olivia Franken
CMP_SC 3050: Lab 6 Part One

Time Window Test Case 1: Feasible Scenario

- A feasible testing scenario is one where a "shortest path" is found which falls within the earliest/latest time constraints for the final node (time > earliest && time < latest).

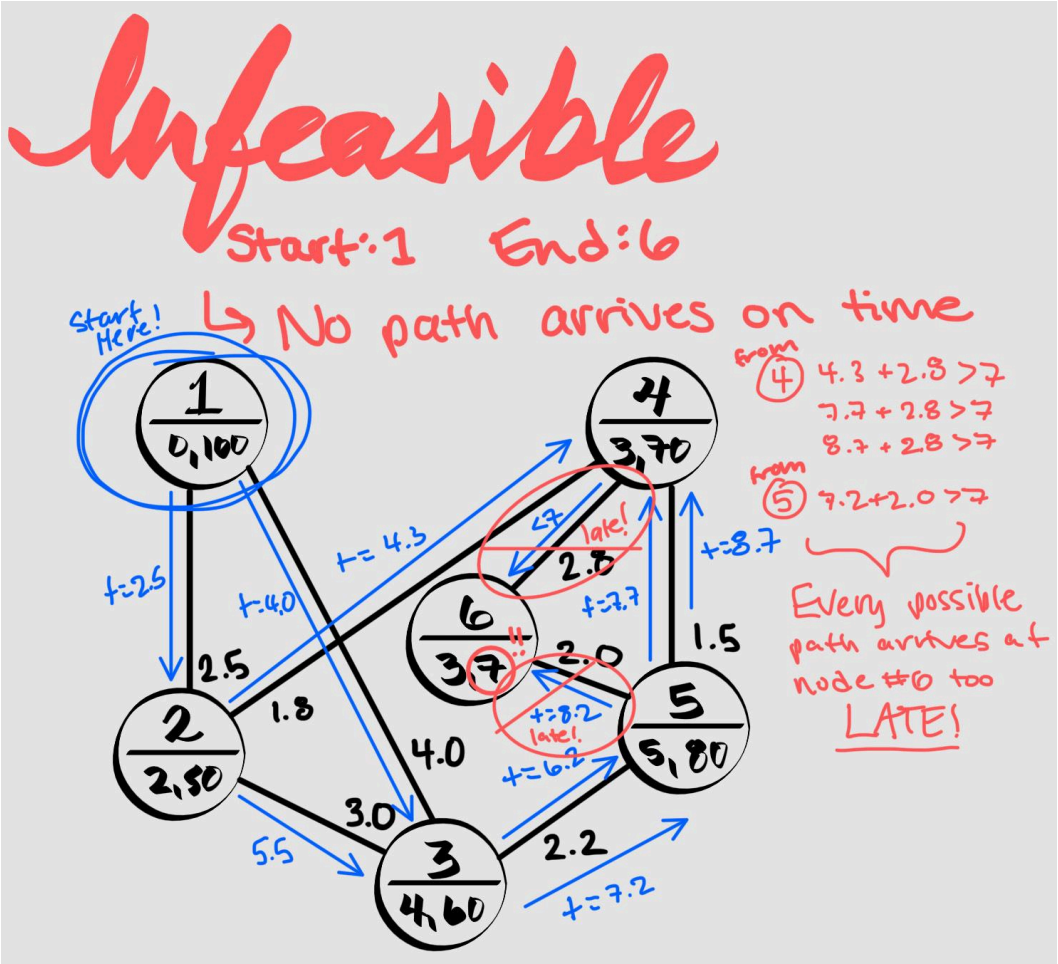
```
./route_planner tw-tests/feasible_nodes.csv tw-tests/feasible_edges.csv 1 6 dijkstra
```



Time Window Test Case 2: Infeasible Scenario

- An infeasible testing scenario is one where the "shortest path" determined still arrives at the target node too late when every possible path has been exhausted (time > latest for <end_node>).
- This scenario satisfies Part 1.1.3's requirement:
    - *Report "No feasible path satisfying time constraints"*

```
./route_planner tw-tests/infeasible_nodes.csv tw-tests/infeasible_edges.csv 1 6 dijkstra
```

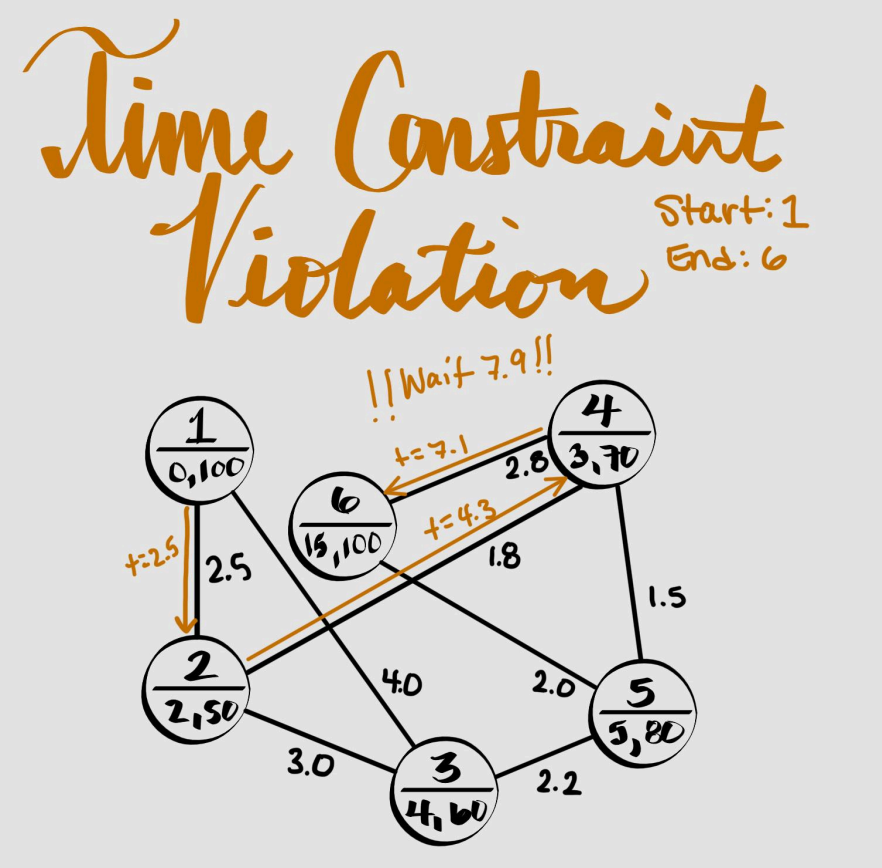


Olivia Franken

Time Window Test Case 3: Constraint Violation Scenario

- A constraint violation testing scenario is one where the “shortest path” determined arrives at the target node too early when every path has been exhausted (time < earliest)
- This scenario satisfies Part 1.1.3’s requirements:
    - *Identify which constraint(s) were violated*
    - *Suggest the "closest" path (minimize constraint violations)*
        - Involves “waiting” until the earliest time is reached.

```
./route_planner tw-tests/shortvc_nodes.csv tw-tests/shortvc_edges.csv 1 6 dijkstra
```



Addressing Part 1.1 Key Challenges

- How do time windows affect the optimality guarantee of Dijkstra?

- Time windows make the optimality guarantee of Dijkstra less achievable by imposing additional constraints. This creates testing situations where "waiting" might become necessary in order to visit an ideal node, where time is wasted. This is a functionality that was not taken into consideration during the initial design of Dijkstra's algorithm.

- When can you prune paths that are shorter but arrive too early/late?

- As soon as a path arrives at the target end node too late, that path can be discarded. However, when a path is found that arrives too early, it must be considered a potential (but less ideal) solution, and should only be thrown out once a fully feasible solution presents itself.

- How does this change the state space you need to track?

- This requires more memory storage to keep track of the "best infeasible path" or "best constraint violation path". Updating and managing these considerations is more resource intensive than traditional Dijkstra.

Task 1.2: Priority-Based Multi-Destination Routing (60 points)

1.2.1 Implement a system that routes through multiple destinations with different priority levels.

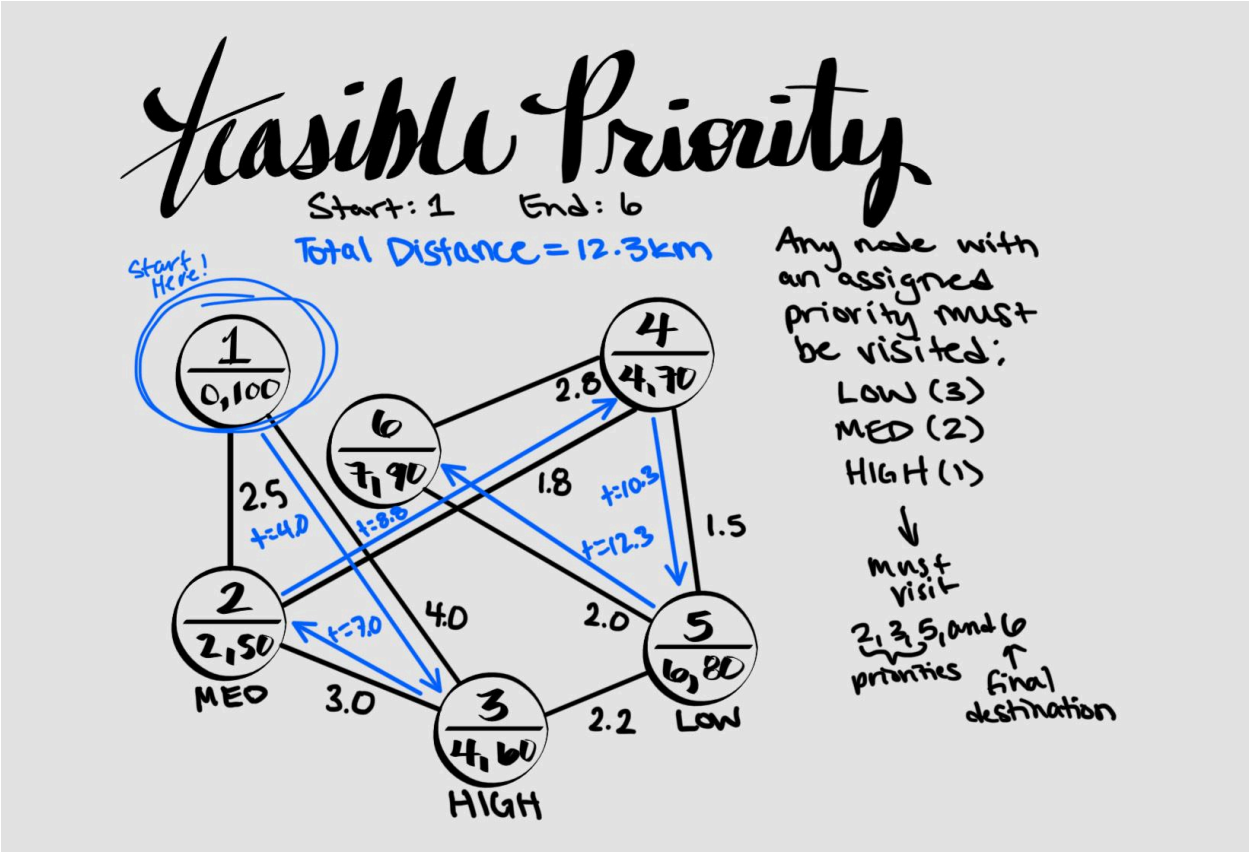
Create a destinations.csv file that is read in alongside respective nodes.csv and edges.csv files. Destinations will contain a list of required nodes to visit, as well as their priority. ***Priority will be assigned such that 1 = High, 2 = Medium, 3 = Low.*** *Ideally, all high priority nodes will be visited first, then all mediums, then all lows. Consider scenarios where satisfying this priority constraint creates significant time delays, and allow the user to specify a threshold parameter which establishes a break point for the priority constraint (where it will be overruled by time).*

Olivia Franken
CMP_SC 3050: Lab 6 Part One

Priority Test Case 1: Feasible Windows and Destination List

- With threshold=0% (no distance constraint), the program accepts all destinations in strict priority order.
- It calculates: Node 3 (priority 1) at 4.0 km from start, then Node 2 (priority 2) at 3.0 km further, then Node 5 (priority 3) at 3.3 km further, finally reaching Node 6 at 2.0 km. The complete path 1→3→2→4→5→6 totals 12.30 km. Since threshold=0 allows unlimited detours, all destinations are mandatory and visited sequentially (in priority HIGH -> LOW order) regardless of the 73% increase over the optimal 7.10 km direct path.

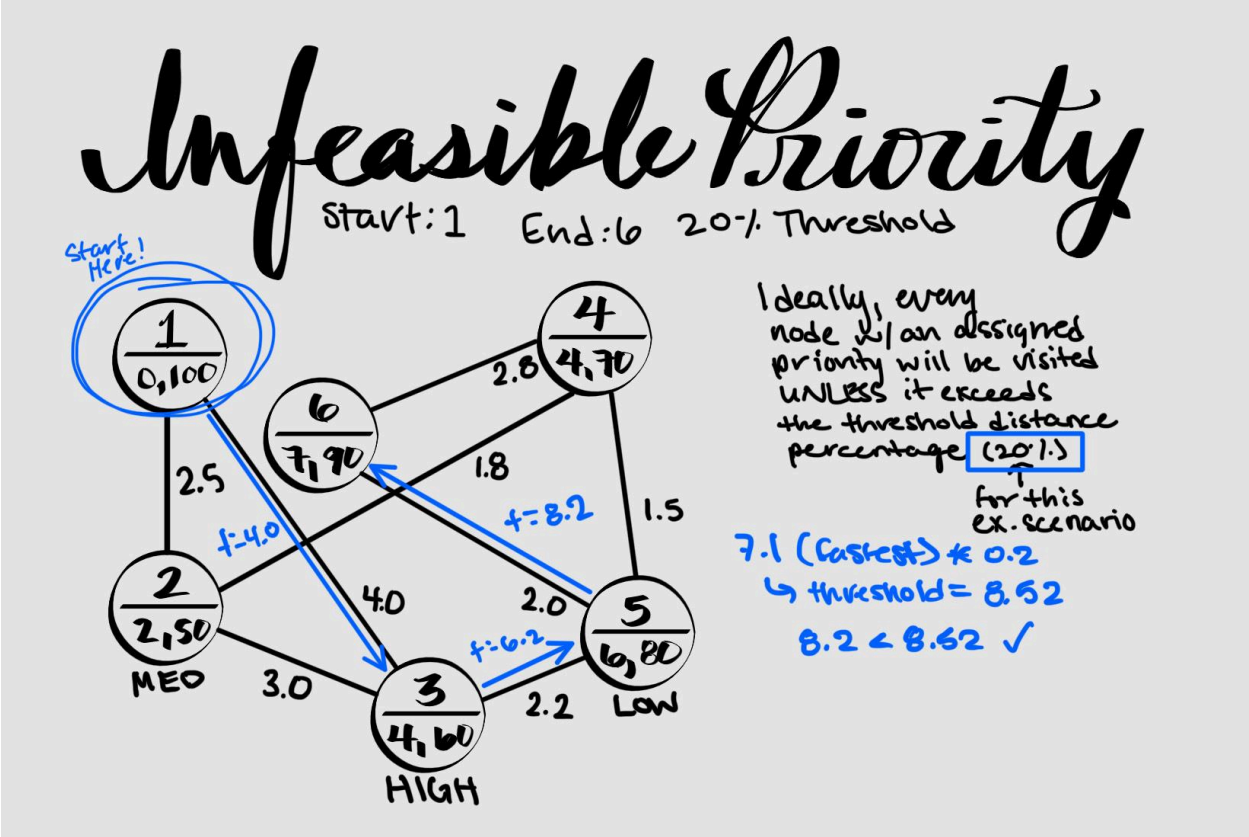
```
./route_planner priority-tests/feasible_nodes.csv priority-tests/feasible_edges.csv
priority-tests/feasible_destinations.csv 1 6 0 dijkstra
```



Priority Test Case 2: Infeasible Destination List

- The program uses threshold enforcement. It calculates the optimal baseline distance based on basic time window traversal, multiplied by the threshold, and then evaluates waypoints/destinations in priority order.
- Node 5 fits within the limit so it's kept. Adding Node 2 would make the path 9.70 km, exceeding 8.52 km, so it's skipped. Result: shortest feasible path respecting highest-priority destination only.

```
./route_planner priority-tests/feasible_nodes.csv priority-tests/feasible_edges.csv
priority-tests/infeasible_destinations.csv 1 6 20 dijkstra
```



Olivia Franken

Questions for Part 1.2:

1. **How is total distance minimized while still obeying priority constraints?**
    a. For each candidate destination, the algorithm calculates: cumulative_distance_to_waypoint + distance_from_waypoint_to_end. If this exceeds threshold_limit, the waypoint is skipped (not visited). This keeps high-priority waypoints while dropping lower-priority ones when they would violate the threshold, minimizing total distance while maximizing priority satisfaction.
2. **Are there any instances where the priority constraint (where high priorities are visited before mediums before lows)?**
    a. Yes, violations occur in Test Scenario #2. The algorithm skips Node 2 (priority 2) because visiting it would exceed the defined threshold above the shortest path. This is intentional so that the threshold constraint takes precedence over strict priority ordering when a conflict arises. In this case, only 2 out of the 3 "required" destinations are visited due to the imposed threshold.
3. **How does the program handle the case where strict priority ordering makes the path significantly longer than the shortest path when only time windows are considered?**
    a. In Test #1 (threshold=0, meaning no threshold priority-breaking constraint), the program accepts ALL destinations regardless of distance increase. It visits them in strict priority order: 1→3→2→5→6 (12.30 km vs. 7.10 km optimal). The program does NOT completely minimize distance in this situation, it instead prioritizes visiting all high-priority destinations, accepting the 73% distance penalty. The break point is defined by the user in the threshold command line parameter.

Olivia Franken

Addressing Part 1.2 Key Challenges

- How much longer can the path be than the shortest path to respect priority order?

- The path can be up to (1 + threshold%) times longer than the shortest path. For threshold=0%, the path can theoretically be infinite (any detour for the listed destinations is allowed). For threshold=20%, the path can be at most 20% longer than the most ideal path.

- Should you allow priority violations if the distance savings is significant?

- Yes, the algorithm does allow priority violations when the distance savings is significant AND the threshold constraint requires it. In Test #2, it skips lower-priority destinations (Node 2 with medium priority) while still hitting the highest priority destination in order to stay within the 20% threshold. This is an explicit design choice: threshold constraints override strict priority ordering, placing more control in the user's hand.

- How do you balance optimality vs. priority satisfaction?

- The program uses a threshold-based hierarchy to maintain a balance between optimality and priority:
    - If threshold=0: Prioritize ALL destinations; distance is irrelevant
    - If threshold>0: Prioritize destinations/waypoints up to the threshold limit; skip lower priorities if they exceed it
- The strategy is to keep the highest-priority destinations first, then skip lower-priority ones that violate the threshold. This is a practical compromise: you get your most important destinations visited while bounding the path length increase.