ECE 309 Project (Fall 2025)

City Transportation Planner

1. Overview

Build a **City Transportation Planner** over a city graph with **string-labeled stops**. You will parse a text dataset, construct an adjacency-list graph, and answer shortest-path queries with nonnegative edge weights.

A few components are not explicitly covered in lecture; they are part of the project's guided exploration, see §10 for links and videos.

2. Teams and Scope

- **Solo (1)**: complete all items in §3.
- **Team of 2**: §3 + choose **one** extension from §4.
- **Team of 3**: §3 + **both** extensions from §4.

3. What You Must Build (Required)

3.1. Data file

The grader provides `./data/edges.txt` (UTF-8, space-separated, single header line `u v length_m`). Each row is an *undirected* edge between string stop IDs with positive length in meters. Stop IDs are single tokens (no spaces), case-sensitive, allowed characters `[A-Za-z0-9_]`.

3.2. Core graph & algorithm

R1. **Adjacency list.** Build a memory-efficient graph structure (you may remap string IDs to compact ints).

R2. **Shortest path.** Implement **Dijkstra** for nonnegative weights.

R3. **Show shortest path.** For each query with a valid route, **show the stop sequence from source to destination** (the shortest path). In your *debug run* and report evidence, print the ordered stop IDs joined by , e.g.,

```
Central_Station BusHub Airport_T5
```

This path printout *must not* interfere with autograder I/O; print it to `stderr` (recommended) or only in a separate debug build.

R4. **Determinism & limits.** Identical inputs → identical outputs. Per-query time limit: 30s. No network access.

3.3. I/O examples (grading interface)

Your program reads from *stdin* and writes to *stdout*; all diagnostics go to `stderr`. No command-line flags are required.

3.3.1 Input (stdin).

Each line contains two stop IDs:

```
src dst
```

3.3.2 Output (stdout).

For every valid input line, print exactly one line:
Success

```
ok <distance_m> <eta_s>
```

where `eta_s` = ⌈`distance_m`/1.4⌉.
Failure

```
error no_path
```

Debug print (stderr). In your debug run and report evidence, also print the shortest path sequence:

```
Central_Station BusHub Airport_T5
```

3.3.3 Examples.

Sample dataset (illustrative; not the real smoke-test data). Download here.

```
Central_Station BusHub 300.0
BusHub Airport_T5 2975.9
Central_Station Museum 450.0
```

```
Museum North_Park 915.4
North_Park Ferry_Pier 800.0
Ferry_Pier Airport_T5 1200.0
Mall_East Hill_Top 600.0
```

Example 1 (single query)

Input
```
Central_Station Airport_T5
```
Output (stdout)
```
ok 3275.9 2340
```
Debug (stderr)
```
Central_Station BusHub Airport_T5
```

Example 2 (unknown stop or disconnected)

Input
```
Mall_East Unknown_Stop
```
Output (stdout)
```
error no_path
```

Example 3 (multiple queries; # = comment)

Input
```
# three OD pairs
Central_Station Airport_T5
Museum North_Park
Hill_Top Ferry_Pier
```
Output (stdout)
```
ok 3275.9 2340
ok 915.4 654
error no_path
```
Debug (stderr)
```
Central_Station BusHub Airport_T5
Museum North_Park
```

Examples above are for format/correctness check only. Gradescope uses hidden tests. **Debug path lines are not part of the stdout**.

3.4. Report (submit as PDF)

Include:

- **Team**: names and emails (match Moodle); brief roles/responsibilities.
- **Scope completed**: checklist of Required / Extension(s) / Bonus.
- **Evidence**: outputs, screenshots, or brief logs supporting correctness, **including the shortest-path printouts** from your debug run.
- **Reproducibility**: exact build/run steps and expected outputs.
- **LLM-free components**: at least **two (solo)** / **three (team)** core modules written without LLMs (e.g., `Graph`, `MinHeap`, `Dijkstra`).

4. Extensions (choose from below)

Choose from the following (see §10 for references):

- **A* with Euclidean heuristic**. Compare with Dijkstra on provided OD pairs (expanded nodes and/or wall-time).
- **Name lookup (nearest-stop by string)**. Implement a simple stop-name lookup (e.g., prefix search or case-insensitive match) using an appropriate structure (hash set + scan, trie, etc.).

Extensions and Bonus are *not* autograded on Gradescope. We will manually run your program and review your report. Please:

- In your report (§3.4), **explicitly state which extension(s) and bonus(es)** you completed.
- Provide **reproducible build/run commands** and any required flags/inputs.
- Attach **evidence** (screenshots and/or logs) demonstrating the extension.
- Briefly **explain your approach/design**.

5. Bonus (extra credit)

Pick at most two. These are optional and *do not* replace Required/Extensions.

- **Batch benchmarking mode**: read multiple OD pairs and write a CSV with columns `src, dst, distance_m, eta_s, time_ms`.
- **Edge-closure re-routing**: support a small blocked-edge list and return "no path" or a re-computed route.
- **Simple route export**: write the found path as a stop-ID sequence to `path.txt`.

Extra credit policy. Each completed bonus *adds +1 point of extra credit* to your **overall course total** (not this project percentage).

Example: if your course total is 86.0 and you complete two bonuses, your new course total becomes 88.0.

6. Smoke Test Protocol & Dataset Scale

Coverage. The smoke test *fully subsumes* data ingestion/validation, graph construction, routing correctness, and I/O discipline.

Dataset scale. The hidden smoke test uses a single large graph of approximately

$$|V| \approx 10{,}000 \quad \text{and} \quad |E| \approx 30{,}000$$

(undirected edges).

Timing. Per-query limit is **30s**. Staying within memory/process limits is required.

> Unlimited submissions on Gradescope until the deadline; **the last submission before the deadline is graded**. Public samples are illustrative only.

7. Grading Rubric

Solo (1 student)	
Smoke test (includes ingestion/graph/routing/I/O)	60%
Report	40%
Total	**100%**

Team of 2	
Smoke test (includes ingestion/graph/routing/I/O)	40%
One extension	20%
Report	40%
Total	**100%**

Team of 3	
Smoke test (includes ingestion/graph/routing/I/O)	40%
Extension A (A*)	15%
Extension B (Name Lookup)	15%
Report	30%
Total	**100%**

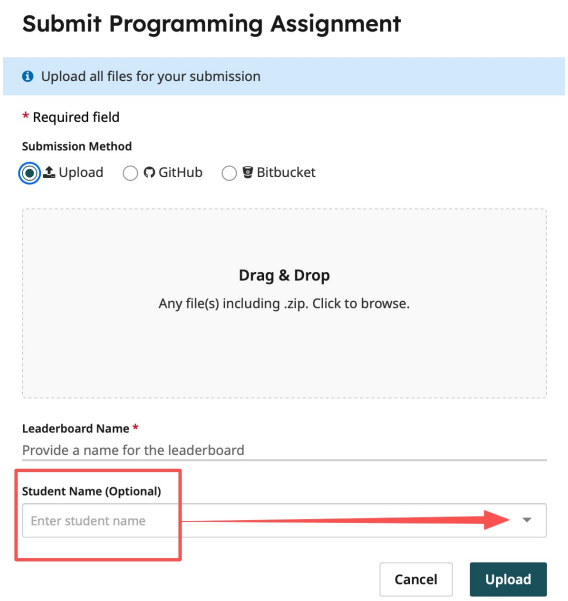
8. Submission, Platforms & Group Selection

Gradescope (Code / Autograder)

- **What to submit**: source files only (`.cpp/.h/Makefile`); **do not** upload report or ZIP here.
- On the upload page, **add Group Members** (select all teammates). **Only one teammate** uploads on behalf of the team; all listed members receive the same score.

Moodle (Report + Code Bundle for Extensions/Bonus)

- **What to submit**: `Report.pdf` *and* a `.zip` code bundle for Extensions/Bonus (scripts, logs, and sources needed to reproduce).
- **One per team**: **Only one teammate** uploads the final PDF and ZIP on Moodle; other members **do not** duplicate the submission.



9. Generative AI / LLM Usage

LLMs may assist brainstorming, proofreading, and minor scaffolding. **Not allowed**: generating or completing core algorithmic implementations (e.g., Dijkstra/A*). Identify LLM-free components in your report (§3.4).

10. Resources

- Red Blob Games — *A* pathfinding*: intro | implementation
- Sebastian Lague — *A* Pathfinding* overview: video
- VisuAlgo — interactive *Dijkstra*: Dijkstra

Questions? Use the course forum or contact TA at mli55@ncsu.edu.