Greedy Student Facing Assignment

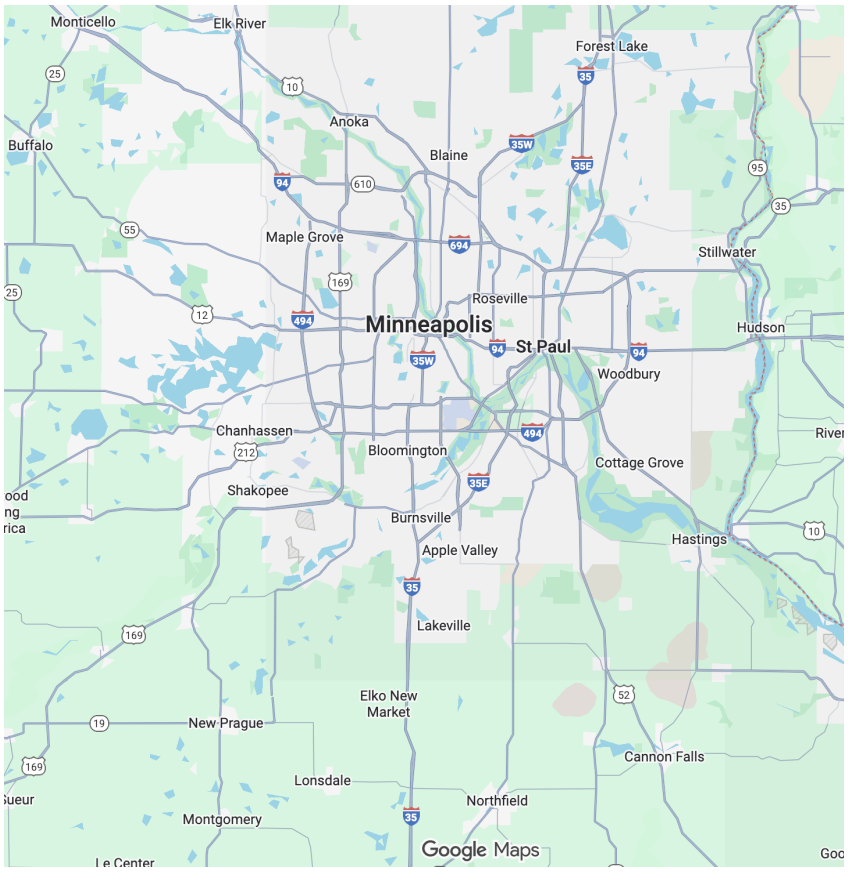
Learning Goals:

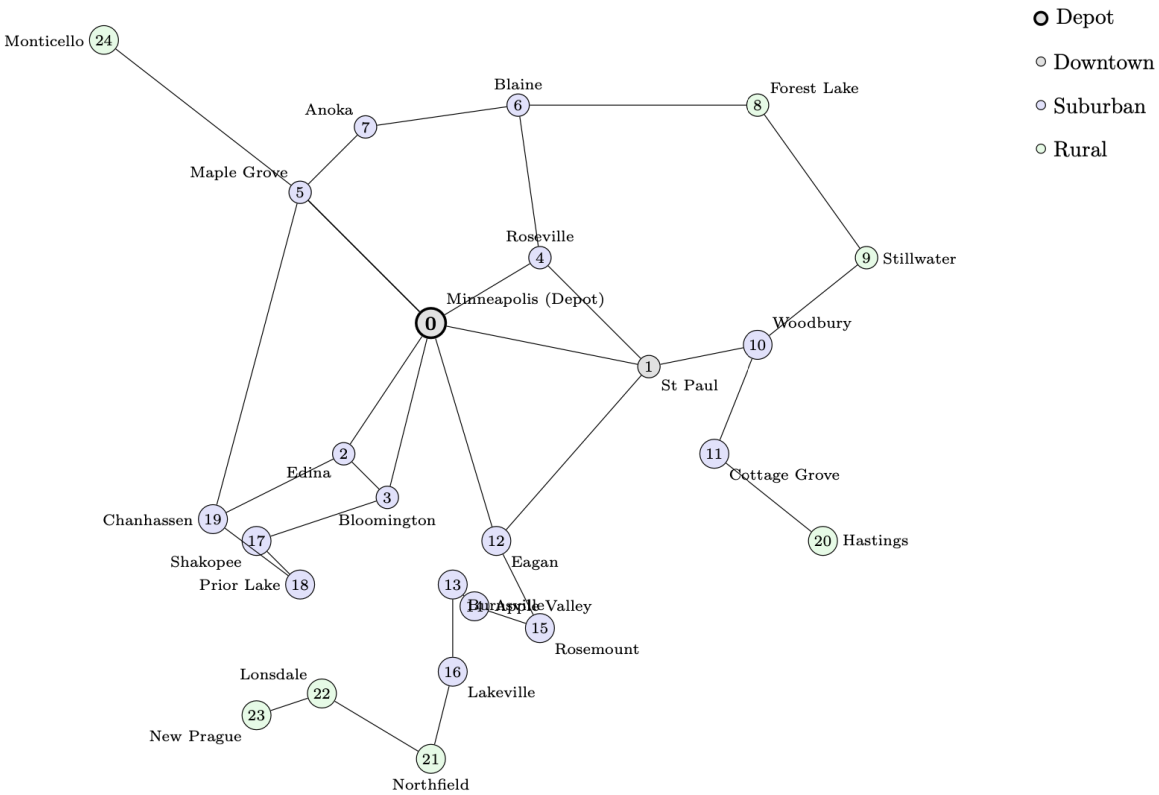
- Implement and prove the correctness of Dijkstra's Algorithm.
- Identify the conditions under which Dijkstra's Algorithm should be applied.
- Discuss the limitations and ethical implications of Greedy Algorithms.

Assignment Overview:

You are helping Amada, an Uber driver in Minnesota, design an algorithm that finds "the cheapest path" between cities. Each city (node) has attributes such as region, traffic. Each road (edge) connects two nodes. The edge object stores only the two endpoints. In other words, there are no stored edge weights as usual; you will compute edge weights yourself from those node attributes.

Your goal is to implement Dijkstra's Algorithm to find the minimum cost route under different perspectives, then extend your model to explore ethical and social implications.


Monticello
Elk River
Forest Lake
Anoka
Buffalo
Blaine
Maple Grove
Stillwater
Roseville
Minneapolis
St Paul
Hudson
Woodbury
Chanhassen
Bloomington
Cottage Grove
Shakopee
Burnsville
Hastings
Apple Valley
Lakeville
Elko New Market
New Prague
Cannon Falls
Lonsdale
Northfield
Montgomery
Le Center
Google Maps


Depot
Downtown
Suburban
Rural
Monticello 24
Anoka 7
Blaine 6
Forest Lake 8
Maple Grove 5
Roseville 4
9 Stillwater
Minneapolis (Depot) 0
Woodbury 10
1 St Paul
2 Edina
3 Bloomington
11 Cottage Grove
Chanhassen 19
17
Shakopee
Prior Lake 18
12 Eagan
20 Hastings
13
15 Rosemount
16 Lakeville
Lonsdale 22
23 New Prague
21 Northfield

Your Task

Part A: Dijkstra for Different Stakeholders

When defining the cost of the trip, companies and drivers like Amada have different factors to consider. In Part A, you will implement dijkstra for the company and modify it for drivers like Amada.

① The Company's Dijkstra Strategy

- **Goal:** Minimize **company's cost** on a single trip.
  - company_cost = Platform costs per ride + mileage * driver base pay per mile
- **Task**: A Dijkstra algorithm written in Python with proper documentation

② The Driver's Dijkstra Strategy

- **Goal**: Minimize **driver's cost** on a single trip.
  - driver_cost = mileage * fuel per mile + parking + maintenance
- **Task:** A Dijkstra algorithm written in Python with proper documentation

Hint:

- Store each node's "lowest cost so far" in a min-priority queue (`heapq` in Python).
- Start with dist[start] = 0 and everything else as infinity.
- Repeatedly pop the node u with the smallest distance, then for each neighbor v:
  `newDist = dist[u] + cost(u, v)`
- Update dist [v] if newDist is lower than original
- Update v's position in the queue until all are processed (or goal reached).
- Rebuild the path.

Part B: Northfield Subsidies!

As a proud Northfield resident, you would like to introduce a subsidy to attract more rideshare drivers. You decided to give out subsidies if Amada picked up a passenger from Northfield. The subsidy ends up exceeding the cost of her trip to Northfield…

- **Goal**: Investigate into negative edge weights in Dijkstra

- **Task**:
  - Copy over your implementation of Part A ② and add:

```
if v = "Northfield":
    driver_cost = -100
```

  - Run your program and reflect briefly (1-2 sentences):
    i. What outcome did you get from running dijkstra with negative edge weights?
    ii. What does this imply about Dijkstra's assumptions? In other words, why can't the graph here contain negative weights?

Part C: Proof

Now it's time to prove your dijkstra algorithm. Before you go and stretch out the proof, it's a great idea to review notes on proving techniques of greedy algorithms.

- **Goal**: Prove Dijkstra with Greedy Stays Ahead for Company's Minimum Cost
- **Task**: A Formal Proof written in LaTeX, output as PDF.

- **Hint**:
  - Remember our conclusion from PartB, Dijkstra is only valid when our calculated edge weights are non-negative. State this as an assumption for your proof.
  - Use Greedy Stays Ahead with induction on $|S|$ (size of finalized set).
  - When prove the inductive step, decompose any shortest path P from s to u as: $s \rightsquigarrow x \rightarrow y \rightsquigarrow u$ where:
    - $s \rightsquigarrow x$ is entirely within S
    - $x \in S$ is the last finalized node on this path
    - $y \notin S$ is the first node outside S (crosses the boundary)
    - $y \rightsquigarrow u$ is the remaining path

Part D: Ethical Modification (Advance Only)

Ammda is happily testing out your algorithm and she liked her helper a lot. It's true that your helper algorithm picks a cheapest route, however, she does have some urgent concerns:

1. I don't want to be driving with fatigue.
2. Sometimes I wish I could take more passengers from rural cities, but it's too much drive and cost.
3. I need to avoid stormy / heavily raining regions for safe drives.

Now, it's your turn to solve those ethical concerns. You should choose ONE of the three concerns and modify your dijkstra accordingly, and reflect briefly on your results.

- **Goal**: Think about responsibility and reflect them in your implementation
- **Task**:
    - Modify your code in Part A ① to implement ONE of the following ethical rules:
        i. Fatigue: Limit consecutive long-distance drives
        ii. Fairness: add subsidies for rural regions (not as much as in Part B!)
        iii. Weather Safety: Avoid bad weather to include drive safety.
    - Run your ethical rules and reflect briefly (1-2 sentences):
        i. How did the ethical rule you pick account for people.
        ii. Can you identify more factors to incorporate?

- **Hint:**
    - **Fatigue**: Add a small penalty whenever a route segment is unusually long. You can keep track of roads taken and set a filter at 5 nodes.
    - **Fareness**: Reward trips to rural cities slightly by reducing their cost (without making them negative)
    - **Weather Safety:** take a list of stormy nodes to note cities with bad weather as input, and add a penalty whenever your route touches one of them.

Part E Reflections and Responsibilities

Great job in building the helper algorithm for Amada, but before you go, she prepared you with some thoughtful questions in return…

- **Goal**: Reflect your work and extend them to your general understanding as a

computer science student!

- **Task**:
    - Re-run your solutions if needed
    - Reflect briefly (1-2 sentences):
        i. Identify some potential impact of algorithm design? Are they also optimal or objective?
        ii. What should computer scientists do to eliminate such impact?

Assessment

In this assignment, you will be graded based on if you have demonstrated understanding the proficiency / Mastery requirements:

The Proficiency Requirement is:

- ☐ Pass all 5 tests for Part A
- ☐ Complete tasks and reflection in Part B
- ☐ Code somewhat well-styled
- ☐ Satisfy Core requirements for Proof (see below)
- ☐ Complete reflections in Part E

The Mastery Requirement is:

- ☐ Satisfy all Proficiency requirements
- ☐ Satisfy Mastery requirements for Proof (see below)
- ☐ Completed Part D
- ☐ Code well-styled

Part C: Proof Assessment Criteria

The Proficiency Requirement is:

- ☐ State and use the key assumption that all edge weights are non-negative
- ☐ Clearly define the correctness goal (Dijkstra returns true minimum cost distances for all nodes)
- ☐ Use a Greedy-Stays-Ahead argument with induction on the finalized set S
- ☐ Present a valid base case showing the claim holds when S = {s}
- ☐ Include an inductive hypothesis that assumes the claim holds for |S| = k
- ☐ Demonstrate an inductive step that shows the claim holds when adding the (k+1)th node
    - ☐ The algorithm never overestimates a shortest path
    - ☐ Nodes added to S have their true shortest path distance finalized at that moment
- ☐ Be written clearly and formatted in LaTeX, exported as PDF

The Mastery Requirement is:

- [ ] Meet all core requirements
- [ ] Provide a complete "greedy stays ahead" argument by showing dist[u] ≤ d*(u)
- [ ] Explicitly justify why relaxing edges preserves optimality
- [ ] Clearly explain the decomposition of any shortest path $P = s \rightsquigarrow x \rightarrow y \rightsquigarrow u$ with clear explanation of each component
- [ ] Use the inductive hypothesis correctly to establish dist[x] = d*(x)
- [ ] Prove the key inequality chain: dist[u] ≤ dist[y] ≤ d*(y) ≤ d*(u)
- [ ] Be cleanly typeset, well-organized, and professionally written in LaTeX

Get Started

Starter Code: https://github.com/Chloexf795/compsgreedy

You should only be implementing your solution in `part_a.py, part_b.py, part_d.py` separately. You should run each file to see results and reflect on it!

You should start by reading through other files to get an understanding of the basic structure, helper functions you are expected to utilize are in `main.py`. Pay attention to the hints!