NATIONAL UNIVERSITY OF COMPUTER & EMERGING SCIENCES ISLAMABAD CAMPUS

DATA STRUCTURES FALL - 2024

Semester Project

Due Date: 8th December, 2024

Time: 11:59 PM

Please follow the following submission instructions. Failure to submit according to the format would result in a deduction of marks. Submissions other than Google classroom (e.g., email etc.) will not be accepted. No late submission will be accepted. **Correct and timely submission of the assignment is the responsibility of every student; hence no relaxation will be given to anyone.**

Instructions: **Total Marks: 100**

1. Make sure that you read and understand each and every instruction. If you have any questions or comments, you are encouraged to discuss your problems with your classmates (and instructors) on google classroom.
2. The student is solely responsible for checking the final .cpp files for issues like corrupt files, viruses in the file, or mistakenly exe sent.
3. If there is a syntax error in the code, zero marks will be awarded in that part of the assignment.
4. Displayed output should be well mannered and well presented. Use appropriate comments and indentation in your source code.
5. **Each project group must have 3 members.**

Honor Policy

Plagiarism is a grave academic offense that can severely undermine your academic integrity and reputation. Any instance of a student found to have plagiarized their assignment, whether from a peer or external source, will be subject to strict consequences. **This may result in a zero score for the current or all assignments, or in severe cases, even a failure in the course**. Furthermore, all instances of plagiarism will be promptly reported to the **Disciplinary Committee** for further action.

GitHub Repository and Commit Practices

For this project, you are required to create a private GitHub repository where you will track your progress and commit code changes regularly. Follow the instructions below to ensure you maintain a professional development workflow.

1. **Private Repository**
    a. Create a private repository on GitHub and add your evaluator as a collaborator to review your progress.
    b. Ensure that no sensitive information or unnecessary files are included in the repository.
2. **Commit Practices**
    a. Commit your progress regularly throughout the development of your project. Each commit should represent a meaningful change, such as the completion of a specific feature or a bug fix.
    b. Use descriptive commit messages that clearly state the purpose of the commit.
    c. Avoid committing incomplete or non-functional code unless it is a work-in-progress commit, clearly labeled as such.
    d. The history of commits will be viewed on the demo day, so ensure consistent progress throughout the assignment.
3. **README and Project Documentation**
    a. Create a well-structured README.md file in your repository.
    b. The README should include:
        i. A brief introduction to the project.
        ii. A list of key features.

iii. Instructions on how to run the code.

iv. Sample outputs or screenshots showing the system in action.

4. **Demo Day**
    a. On the demo day, your commits will be reviewed to ensure consistent progress throughout the development process.
    b. Be prepared to showcase your repository structure, commit history, and README file as part of the demo presentation.

Submission Guidelines

1. Keep a backup of your work always that will be helpful in preventing any mishap and avoid last hour submissions
2. Combine all your work in one folder. The folder must contain .cpp files and README file.
3. Rename the folder as ROLL-NUM_DEGREEPROGRAM_SECTION (e.g., i230001_DS/AI) and compress the folder as a zip file. (e.g., i230001_DS/AI.zip). Strictly follow this naming convention, otherwise marks will be deducted.
4. Submit the .zip file on Google Classroom within the deadline.
5. Include a README with instructions for running the program.
6. Include sample inputs and outputs.

Note: Start early so that you can finish it on time.

Overview

The **Smart Traffic Management System Simulator** is a practical project designed to replicate real-world traffic management in a city. It optimizes urban traffic flow using data structures like graphs, heaps, and priority queues. The system focuses on real-time vehicle routing, traffic signal control, congestion management, and emergency handling without incorporating predictive features.

System Modules

---

1. City Traffic Network (Graph Representation)

- Represent the city's traffic infrastructure as a **weighted, directed graph**:
  - **Nodes**: Represent intersections.
  - **Edges**: Represent roads between intersections, with weights representing travel time or congestion levels.
- **Features**:
  - Support dynamic addition and removal of roads or intersections.
  - Visualize the graph structure in a text-based or graphical form.

---

2. Vehicle Routing System

- Calculate the shortest or fastest route for vehicles dynamically.
- **Features**:
  - Use **Dijkstra's Algorithm** to find the optimal path.
  - Recalculate routes dynamically when traffic conditions change.
  - Support vehicle tracking to monitor their movement across the network.

---

3. Traffic Signal Management

- Control traffic lights at intersections to minimize congestion and ensure smooth flow.
- **Features**:
  - Use a **priority queue** to manage incoming roads based on vehicle density.
  - Dynamically adjust green signal durations to reduce overall wait times.
  - Implement an emergency override system for critical situations.

---

4. Congestion Monitoring

- Continuously monitor the number of vehicles on each road.
- **Features**:
    - Use a **hash table** to track vehicle counts for each road segment.
    - Identify congested roads and reroute traffic using BFS or DFS algorithms.
    - Display congestion levels visually for analysis.

---

5. Emergency Vehicle Handling

- Provide special routing for emergency vehicles to ensure minimal delays.
- **Features**:
    - Override normal traffic signal operations to clear a path.
    - Use *A Search Algorithm** to calculate the fastest possible route.
    - Restore normal traffic flow once the emergency vehicle has passed.

---

6. Accident and Road Closure Simulation

- Simulate disruptions such as road closures or accidents.
- **Features**:
    - Block specific roads or intersections dynamically.
    - Recalculate affected vehicle routes and update the traffic network.
    - Monitor system performance under these disruptions.

---

7. Simulation Dashboard

- Offer an interactive interface to visualize and control the simulation.
- **Features**:
    - Display traffic flow, congestion levels, and signal statuses.
    - Allow manual addition or removal of vehicles.
    - Generate logs of all system actions, including rerouting and signal changes.

System Workflow

1. **Initialization**:
    - Define the road network as a graph.
    - Assign initial weights to roads based on estimated travel times.
2. **Traffic Simulation**:
    - Vehicles are added to the network with defined start and end points.
    - The system calculates and assigns routes using the current graph state.
3. **Signal and Flow Management**:
    - Monitor vehicle density at intersections.
    - Adjust traffic signals dynamically to prioritize heavily congested roads.
4. **Emergency Handling**:
    - Identify emergency vehicles and adjust traffic flow for their priority.
    - Re-optimize traffic flow once the emergency vehicle has cleared the network.
5. **Congestion Resolution**:
    - Reroute vehicles from congested roads to less busy paths.
    - Recalculate routes dynamically as congestion levels change.

Data Structures and Algorithms

- **Graph**: Represent the city's road network with adjacency lists.
- **Priority Queue**: Manage the order of roads for signal adjustments.
- **Min-Heap**: Efficiently find the road with the most congestion.
- **Hash Table**: Track real-time vehicle counts on roads.
- **Dijkstra's Algorithm**: Calculate shortest paths for vehicles.
- ***A Search Algorithm****: Handle emergency vehicle routing efficiently.
- **BFS/DFS**: Detect congestion zones or inaccessible paths.

Implementation Details:

Programming Language: C++

Data Input:

- Road Network: Use file input (`road_network.csv`) to define the road network graph (nodes as intersections, edges as roads with travel times).
- Vehicles: Initialize vehicle data using file input (`vehicles.csv`) (ID, start, and end points) from the input file.
- Traffic Signals: Define initial signal timings using file input (`traffic_signal_timings.csv`) (intersection and green time) from the file.

- Emergency Vehicles: Specify emergency vehicle details using file input (`emergency_vehicles.csv`) (ID, start, end, priority) from the file.
- Accidents/Closures: Simulate disruptions by dynamically blocking roads as defined in the file (`accidents_or_closures.csv`).

Visualization: Console-based interface displaying:

- Graph structure (intersections and roads)
- Vehicle positions and routes
- Congestion levels on roads
- Traffic signal status (red/green) at intersections

Simulation Logs: Generate real-time logs tracking:

- Vehicle movements and routes
- Congestion levels and locations
- Signal changes and timings
- Emergency vehicle routes and overrides
- Accident/closure occurrences and impact

Deliverables

1. **Source Code**: Modularized and thoroughly documented.
2. **Simulation Scenarios**:
    - Normal traffic flow.
    - Peak-hour congestion.
    - Emergency vehicle routing.
    - Accident or road closure scenarios.
3. **Performance Metrics**:
    - Average vehicle travel time.
    - Signal adjustment efficiency.
    - Total resolved congestion events.

Bonus Task: Finding the Best Route with Changing Traffic

Imagine you're using a GPS app, but the traffic is constantly changing. This task is about helping your simulator find the best route even with those changes.

How it Works:

1. **Flexible Travel Times:** Instead of assuming each road takes a fixed time to travel, we'll allow for different travel times depending on how much traffic there is.

2. **Smart Route Finder:** We'll use a clever method (called "dynamic programming") to find the best route. This method keeps track of all the possible routes and how long they might take, and then picks the fastest one.

3. **Keeping Track of Time:** We'll also consider the time of day, because traffic changes throughout the day.

Making it Visual:

- You can show how the best route changes as traffic gets better or worse.

- You can use colors to show which roads are congested (red for slow, green for fast).

This bonus task helps the simulator be more realistic by considering how traffic changes. It also uses a cool technique to find the best route, which is something used in real-world GPS apps!

HAPPY CODING :)