CS56 Web Application: StoreMapper

Introduction

StoreMapper is a web-based application desgined to manage and visualize location data, primarily focusing on stores and restaurants. It uses OpenStreetMap (OSM) data structures for its mapping and routing features.

Project Overview

StoreMapper utilizes Leaflet.js to display an interactive map to users. Users are able to use the map to both search and filter for locations based on OSM tags. After locations are selected, the application determines the fastest path between locations using Djikstra's algorithm.

Key Features and Achievements

1. OSM Data Integration
2. Advanced Routing Algorithms
3. Interactive UI
4. Scalable Architecture
5. Efficient Data Handling
6. Comprehensive Testing
7. Cross-Environment Compatibility
8. Performance Optimization

A. Names of Group Members

- Carson Bell
- Wei Che
- David Dickinson
- Beckham Lee
- Adityan Vairavel

B. Java Version

JDK 21.0.3 (OpenJDK on Ubuntu/WSL2 and DigitalOcean, Oracle JDK LTS on Windows)

C. IDEs

- Eclipse IDE for Enterprise Java and Web Developers (includes Incubating components)
    - Version: 2024-06 (4.32.0)
    - Build ID: 20240606-1231
- IntelliJ IDEA 2024.1.4 (Ultimate Edition)

D. Challenges Faced

Integrating OpenStreetMap (OSM) data structures

We needed to transition from a simple Location model to a more complex system based on OpenStreetMap (OSM) data by incorporating Nodes and Ways. We used Python starter code from the 6.009 problem set as a template to convert OSM data into Java objects. We also had to separately create Node and Way classes to represent OSM data, as well as implement a Graph class to represent the data.

Redesigning the service layer

We needed to update MapService and LocationService to work with the OSM-based graph structure. Additionally, we had to ensure backwards compatibility with the original MapService and LocationService implementations.

Implementing Pathfinding

We used Djikstra's algorithm to find the shortest path between two locations. To do this, we utilized way tags about speed limits and types of roadways to map way segments to speed limits. We then ultimately used these in a modified Djikstra's algorithm to find the fastest path between two locations.

UI/UX Challenges

We needed to incorporate an interactive route display using Leaflet.js, which consisted of learning the specifics of Leaflet controls, layers, and markers. We also needed to add styling to certain elements for a cleaner UI: route information, color-coded route segments, and clickable markers on locations. We also implemented a search feature for the "Store" and "Restaurant" categories, which involved extracting unique sets from the OSM data and passing this data from the backend to the front-end. Finally, we encountered challenges learning how to use the JavaServer Pages (JSP) templating.

Testing & Validation

We struggled to create a consistent runtime environment between our team members. Some of us used different operating systems or IDE's, which made consistent configuration difficult.

Data & Memory Management

Our maps and ground truth paths were sourced from the "Frugal Maps" Fall 2020 MIT 6.009 problem set. The large files in this dataset (~ 500 MB for nodes, ~ 180 MB for ways) were placed into the /data directory and in Github's Large File Storage (LFS) system. However, continuous integration tests by GitHub workflow led to the data being constantly redownloaded, which ate through our 1.5GB LFS quota. To resolve this, we moved the data to Cloudflare R2 and had to workflow download data on-demand.

Environment Setup

We configured Tomcat to run in different environments (standalone in Ubuntu, Eclipse integrated Tomcat, command line in WSL2 / Digital Ocean). This involved understanding and managing environment variables, like JAVA_HOME and CATALINA_HOME across different setups.

Deployment

We deployed the application as `ROOT.war` to serve at the root context. This had to be done manually in the command line for Ubuntu+WSL2 and Digital Ocean. Additionally, we configured Eclipse to serve the application at the root context.

E. Current Limitations and Attempted Solutions

Performance Issues

When processing very large OSM datasets, we encountered very slow performance. To resolve this, we limited ourselves to small/medium datasets (~ 500-1000 MB Nodes and Ways). For larger datasets, we would need to incorporate a more efficient chunking strategy, spatial indexing, to find the nearest node to given coordinates.

Memory Management

Searching without qualifiers on the 4GB DO droplet leads to Heap Overflow errors.

UI Responsiveness

The front-end crashes if the user loads all locations in West Los Angeles at once. To resolve this, we would need to implement a Level-of-Detail strategy to load only the visible locations at once, and/or selectively show a subset for the current zoom view.

F. Future Work

Database Integration

We would like to move from an in-memory `MapService` to a database-backed solution. We would also like to implement PostGIS and PostgreSQL to store and query OSM data efficiently and optimize spatial queries.

Performance Optimization

We would like to implement advanced spatial indexing (R-tree) for faster nearest-neighbor queries in addition to developing a caching strategy for frequently accessed routes and locations. Ultimately, we hope to optimize our algorithms for larger datasets.

Advanced Routing Features

To make the data we show users even more robust and detailed, we feel that incorporating alternate modes of transport, like walking and cycling, would be benefical. On top of this, it would be interesting to incorporate multi-modal routing where a user can walk and bike to a destination. Real-time traffic data would be useful, as well as multi-stop routes.

User Experience Improvements

We would like to implement a more sophisticated UI that works well with mobile devices. User account systems will be created to save favorite locations and routes. In the future, we hope to add support for user-generated content, such as reviews and ratings.

Data Management and Updates

We would like to develop a system for regular updates of OSM data. In addition, we would need to implement a data validation/cleaning pipeline for user-submitted data. Lastly, we would like to create an

administrative interface to manage location data and user content.

Conclusion

The StoreMapper project has been a significant undertaking, challenging us to integrate complex data structures, implement efficient algorithms, and create a user-friendly web interface. Through this process, we've gained valuable experience in: working with real-world geographical data and the challenges it presents, implementing and optimizing graph-based algorithms for pathfinding, balancing performance and functionality in web application development, and collaborative problem-solving and version control in a team environment.