Hanoi University of Science and Technology

School of Informatics and Communication Technology

PROJECT-1 REPORT

Title: Hanoi Routing Application

Full Name: Vu Thuong Tin
Student id: 20230091
Semester: 2025.1

Supervisor: Dr. Dinh Viet Sang

January 15, 2026

Abstract

Traffic congestion and navigation in complex urban areas like Hanoi present significant challenges. This project, titled **Hanoi Routing App** aims to develop a software solution that simulates and visualizes optimal driving routes between real-world locations. By leveraging **OpenStreetMap (OSM)** data for the road network and implementing **Dijkstra's Algorithm** from scratch, the application calculates the shortest path based on physical distance. The system is built using **Python**, with **Streamlit** serving as the web interface and **Folium** providing interactive map visualizations. This report details the system architecture, the algorithmic implementation, and the technical solutions applied to handle large geographic datasets within a version-controlled environment. Demo this app via **GitHub**

Contents

Chapter 1

Introduction

1.1 Background

Navigation systems have become indispensable in modern life. Behind familiar apps like Google Maps or Grab lies the complex application of Graph Theory and shortest-path algorithms. Understanding how these systems process real-world geographic data is fundamental for computer science students. Hanoi, with its intricate network of streets, alleys, and diverse traffic conditions, serves as an excellent case study for developing a routing application.

1.2 Problem Statement

While many routing APIs exist, building a system from scratch poses specific challenges:

- How to convert raw geographic data (latitude/longitude) into a computational graph?
- How to efficiently store and load large map datasets (approx. 160MB for Hanoi) without overloading version control systems like Git.
- How to visualize the mathematical result (a list of node IDs) onto a user-friendly map.

1.3 Objectives

The primary objectives of this project are:

1. To acquire and process OpenStreetMap data for Hanoi's driving network.
2. To implement Dijkstra's algorithm manually to find the shortest path.
3. To develop an interactive web interface allowing users to input start/end locations.
4. To solve technical challenges related to GitHub file size limits.

Chapter 2

Theoretical Background

2.1 Graph Theory in Routing

In this project, the road network of Hanoi is modeled as a weighted directed graph $G = (V, E)$, where:

- **Vertices (V):** Represent intersections or endpoints of road segments (Nodes).
- **Edges (E):** Represent the roads connecting these intersections.
- **Weights (W):** Represent the cost to travel between two nodes. In this application, the weight is primarily defined by the **length** (distance in meters) of the road segment.

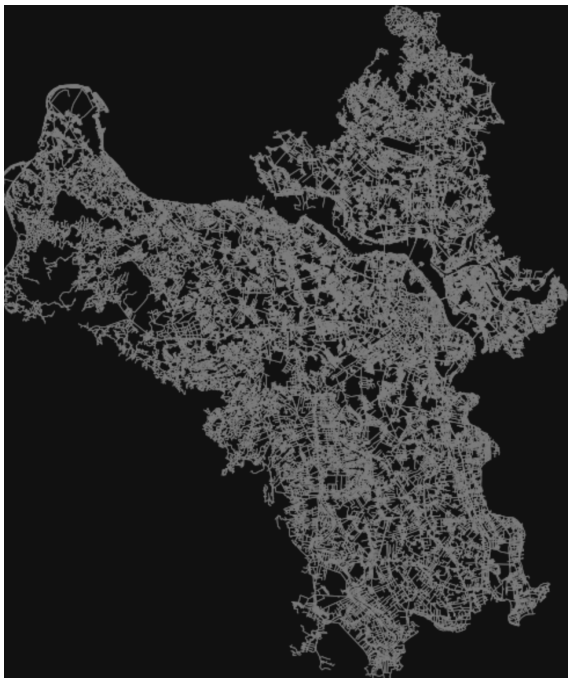

Figure 2.1: Representation of a road network as a Graph (Placeholder)

2.2 Dijkstra's Algorithm

Dijkstra's algorithm is a classic greedy algorithm used to find the shortest path from a source node to all other nodes in a graph with non-negative edge weights.

- **Principle:** It maintains a set of visited nodes and a set of unvisited nodes. It tentatively assigns a distance value to every node (set to infinity initially, zero for the source). It always selects the unvisited node with the smallest tentative distance.
- **Complexity:** With a priority queue (min-heap), the complexity is $O(E + V \log V)$.

2.3 OpenStreetMap and OSMnx

OSMnx is a Python package that lets users download spatial geometries from OpenStreetMap and model them as NetworkX graphs. This project uses OSMnx to download the "drive" network of Hanoi.

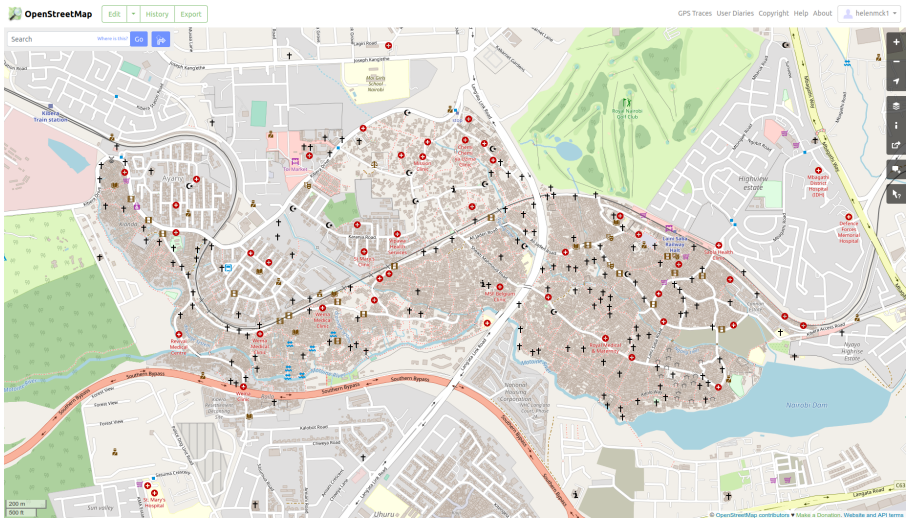

Figure 2.2: Open Street Map

Chapter 3

System Design and Architecture

3.1 Technology Stack

The project is built using the following technologies:

- **Programming Language:** Python 3.9+
- **Data Processing:** OSMnx, NetworkX, Pandas, NumPy.
- **Web Framework:** Streamlit (for rapid UI development).
- **Visualization:** Folium (Leaflet.js wrapper for Python).
- **Version Control:** Git & GitHub.

3.2 System Architecture

The application follows a modular architecture:

1. **Data Layer:** Handles the '.graphml' file storage. Includes logic to check file existence and auto-download from OSM if missing.
2. **Logic Layer:** Contains the 'DijkstraSolver' class and helper functions.
3. **Presentation Layer:** The 'app.py' script which renders the sidebar, map, and results metrics.

Figure 3.1: System Architecture Diagram (Placeholder)

3.2.1 Data Acquisition (map_loader.py)

This is a critical module designed to handle the large dataset problem. Since the Hanoi graph file exceeds 100MB, the code includes a fail-safe mechanism.

```
import osmnx as ox
import os

def load_graph(self):
    if not os.path.exists(GRAPH_FILE):
        print("Downloading data from OSM...")
```

```
        # Download driving network for Hanoi
        G = ox.graph_from_place("Hanoi, Vietnam", network_type='drive')
        # Save locally for future use
        ox.save_graphml(G, GRAPH_FILE)
    else:
        G = ox.load_graphml(GRAPH_FILE)
    return G
```

Listing 3.1: Map Loader Logic

3.3 Algorithm Implementation (algorithms.py)

Instead of relying solely on libraries, a custom `DijkstraSolver` was implemented using Python's `heapq`.

```
import heapq

def find_shortest_path(self, start_node, end_node):
    priority_queue = [(0, start_node)]
    distances = {node: float('inf') for node in self.G.nodes}
    distances[start_node] = 0
    predecessors = {}

    while priority_queue:
        current_dist, current_node = heapq.heappop(priority_queue)

        if current_node == end_node:
            break # Target reached

        for neighbor in self.G.neighbors(current_node):
            weight = self.G[current_node][neighbor][0].get('length', 1)
            new_dist = current_dist + weight

            if new_dist < distances[neighbor]:
                distances[neighbor] = new_dist
                predecessors[neighbor] = current_node
                heapq.heappush(priority_queue, (new_dist, neighbor))

    return self.reconstruct_path(predecessors, end_node)
```

Listing 3.2: Dijkstra Algorithm Implementation

3.4 Demo

To test the application, we simulated a route from **"Ho Guom"** (Sword Lake) to **"HUST University"**.

- **Result:**
  - The application successfully Geocoded the locations.
  - The computed path followed major roads: Tran Dai Nghia → Dai Co Viet → Le Dai Hanh → ... → Dinh Tien Hoang.
  - **Distance:** Approximately 3.91 km.

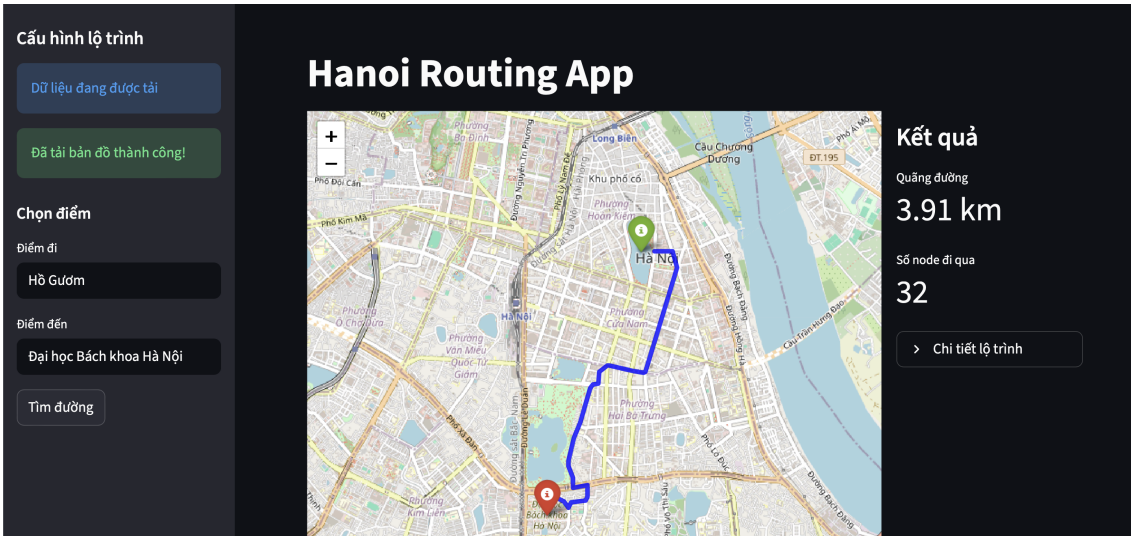

Figure 3.2: The application displaying the optimal route on the map (Placeholder)

Chapter 4

Conclusions

4.1 Summary

The **Hanoi Routing App** project successfully met its initial objectives. We built a functional, interactive routing application that works with real-world data. The project demonstrated the practical application of Data Structures (Graphs, Priority Queues) and Algorithms (Dijkstra) in solving a real-world navigation problem. Furthermore, the experience of managing large files and resolving Git conflicts provided valuable lessons in software engineering workflows.

4.2 Limitations

Despite the successful implementation, some limitations remain:

- Many nodes in the raw OSM data do not have associated names. We implemented a heuristic in
- **Algorithmic Efficiency:** While Dijkstra's algorithm guarantees the shortest path, it explores nodes uniformly in all directions (uninformed search). For long-distance routing across the entire city, this approach is computationally expensive compared to heuristic-based algorithms like A* (A-Star), which directs the search towards the target.
- **Geocoding Dependency:** The accuracy of the start and end points depends heavily on the external Nominatim API. Vietnamese addresses can be unstructured, leading to occasional failures in converting place names to precise coordinates.

4.3 Future Work

Future enhancements could include the implementation of the **A* Search Algorithm** for faster pathfinding using heuristics, and the integration of live traffic APIs to provide dynamic routing. Additionally, expanding the dataset to cover all of Vietnam would increase the utility of the application.

References

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms*. 3rd edition. MIT Press.

[2] Boeing, G. (2017). "OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks." *Computers, Environment and Urban Systems*, 65, 126-139.

[3] Haklay, M., and Weber, P. (2008). "OpenStreetMap: User-Generated Street Maps." *IEEE Pervasive Computing*, 7(4), 12-18.

[4] Streamlit Documentation (2024). "Streamlit Library: The fastest way to build and share data apps." [Online]. Available: https://docs.streamlit.io.