



City Graph

Data Structures Project Report

Name: Anvi Trivedi

PRN: 24070721006

Course Name: Data Structures (Theory and Lab)

Course Faculty: Dr. Salakapuri Rakesh

Department: CSE

College: Symbiosis Institute of Technology, Hyderabad

Semester: III

Table of Contents

<i>Sr. No.</i>	<i>Topic</i>	<i>Page</i>
1.	<i>Abstract</i>	3
2.	<i>Introduction</i>	4
3.	<i>Problem Definition</i>	5
4.	<i>Literature Review</i>	6
5.	<i>Methodology</i>	7
6.	<i>Implementation Details</i>	9
7.	<i>Results & Analysis</i>	10
8.	<i>Conclusion</i>	12
9.	<i>References</i>	13

Abstract

The City Graph project addresses the challenge of visualizing and understanding real-world connectivity through the use of graph algorithms. In most navigation systems, users only see a direct route between two points. City Graph expands on this idea by allowing users to explore multiple possible paths, shortest routes, and traversal patterns across city locations. It aims to make graph theory concepts—such as nodes, edges, and weighted paths—more interactive and intuitive through a visual map interface.

The project models the city as a graph, where user-selected map points act as nodes and the connecting routes act as weighted edges. The system uses Dijkstra's algorithm to find the shortest path between two locations and Depth-First Search (DFS) to explore routes through the city network. This approach demonstrates how these algorithms operate in real-world contexts, emphasizing both optimal routing and systematic exploration.

City Graph is built with HTML, CSS, and JavaScript, using the Leaflet.js library for map visualization. It integrates the Open Source Routing Machine (OSRM) for real-world route data and applies custom graph structures to store connections and distances. The interface is designed with a clean, modern glass-style layout using the Inter font for clarity.

The outcome is an interactive tool that visually represents how data structures and algorithms function in real-world navigation. Users can select points, generate routes, and explore paths dynamically, gaining a practical understanding of how algorithms like Dijkstra's and DFS are applied in systems such as mapping and transport networks.

Introduction

Modern cities rely heavily on navigation and route optimization systems, which are built upon core concepts of graph theory and data structures. Each city can be represented as a network of locations (nodes) and connecting roads (edges), where efficient traversal and exploration depend on well-designed algorithms. This project, City Graph, was developed to visualize these graph-based operations interactively using real map data.

The primary objective of City Graph is to illustrate how graph algorithms are applied in real-world contexts. It allows users to mark locations on a map, find the shortest route between them, and explore multiple possible paths. The project applies Dijkstra's algorithm for shortest-path computation and Depth-First Search (DFS) for exploring routes across the city network. These implementations demonstrate the power and practicality of classical data structures in everyday applications, such as navigation systems.

The project's scope focuses on graph traversal and visualization, integrating real maps through Leaflet.js and live routing data from OSRM. While it successfully demonstrates pathfinding and exploration, limitations include dependency on online routing services and simplified handling of alternate paths.

City Graph is directly relevant to Data Structures and Algorithms (DSA) as it applies fundamental concepts such as graphs, weighted edges, adjacency lists, and traversal algorithms. Through interactive visualization, it bridges the gap between theoretical graph operations and their real-world applications in navigation and network design.

Problem Definition

The problem addressed by City Graph is to model and visualize a real-world city as a graph, where locations act as nodes and roads act as weighted edges, in order to explore and determine efficient travel routes between selected points. The challenge lies in dynamically integrating real geographical data and applying graph algorithms to provide meaningful route analysis.

The system accepts user-defined inputs — specific locations (or “homes”) selected on an interactive map. Once these locations are chosen, the program must compute and display either the shortest path between two nodes using Dijkstra's algorithm or all possible exploration routes using Depth-First Search (DFS).

The outputs are the visual representations of these computed routes on a live city map. For the shortest routes, the system highlights the optimal path, while the exploration feature animates the traversal sequence through the graph.

Constraints include:

- Dependence on an active internet connection for real map rendering and routing data.
- Limited accuracy of route alternatives based on the external OSRM service.
- Performance scalability — the system is best suited for smaller, user-defined graphs rather than full-scale city networks.

Literature Review

Several existing systems and algorithms address the problem of route finding and map navigation. Common approaches include Dijkstra's algorithm, A* (A-star) search, and Breadth-First Search (BFS) for unweighted or grid-based maps. Modern mapping services, such as Google Maps and OpenStreetMap (OSM), utilize highly optimized versions of these algorithms, combined with extensive road network data and heuristics, to provide real-time directions.

For educational or small-scale visualizations, existing projects often simplify the problem by using static graphs or grid representations without integrating real-world maps. While these methods effectively demonstrate algorithmic concepts, they lack realism and interactivity.

In contrast, City Graph combines classical graph algorithms with real geographic data using the Leaflet.js library and the OSRM routing API. This hybrid approach bridges the gap between theoretical graph traversal and practical, map-based route analysis, allowing users to see data structure concepts (such as nodes, edges, and weighted paths) applied directly to real city layouts.

Limitations

- The project relies on the OSRM (Open Source Routing Machine) API, which provides only a limited number of alternate routes between two points. As a result, not all possible real-world paths are displayed.
- The availability and accuracy of routes depend on OSRM servers and the underlying OpenStreetMap data. Network issues or API rate limits may prevent some routes from loading.
- In cases where OSRM fails to respond, the system uses straight-line fallback paths, which may not represent realistic driving or walking routes.
- The application currently does not save data, so all nodes and routes are cleared once the page is refreshed or closed.
- Handling a large number of nodes and edges can slightly impact performance due to browser-side computation and rendering limitations.
- The interface focuses on functionality rather than detailed city modeling, meaning some visual aspects may not perfectly match real-world urban layouts.

Methodology

The City Graph project is designed to represent real-world city navigation using data structures and algorithms. The system allows users to select locations on a real map and then explore possible routes, find the shortest path, or explore the city using traversal algorithms. The application combines user interaction, algorithmic processing, and real-time visualization.

System Architecture

The system has three main components:

1. **User Interface:**

The user interface is built using HTML, CSS, and Leaflet.js. It provides an interactive city map where users can drop pins to mark locations, select them, and visualize routes.

2. **Processing and Algorithms:**

This part handles the project's main logic. It utilizes data structures, such as graphs, to store nodes (locations) and edges (routes). Algorithms such as Dijkstra's for shortest path and Depth-First Search (DFS) for exploration are implemented here.

3. **Visualization:**

The results of the algorithms are shown on the map using colored lines and animations. Each operation (like finding routes or exploring the city) has a different color and animation style to make it visually clear to the user.

The general flow of the system starts when a user selects or adds locations on the map. Each selected point becomes a node in the graph. When routes are generated, edges are added between these nodes with distances as weights. Based on the selected operation, the system runs the chosen algorithm and displays the results visually on the map.

Algorithms and Data Structures Used

The main data structure used in this project is a graph, represented using an adjacency list. Each node in the graph represents a location, and each edge represents a route between two locations. The edges have weights, which correspond to the distances between locations. The adjacency list structure is efficient for representing real-world road networks, as it enables quick traversal and facilitates easy updates.

For pathfinding, Dijkstra's Algorithm is used. It calculates the shortest distance between two nodes by visiting the nearest unvisited node and updating distances until the destination is reached. It ensures that the optimal path is found based on the edge weights.

For exploration, Depth-First Search (DFS) is implemented. It recursively visits all connected nodes, showing how a user could "explore" all parts of the city network. DFS is ideal for demonstrating traversal through connected components and is visually represented as an animated path on the map.

Additionally, arrays and objects (hash maps) are used to store nodes, edges, and their attributes efficiently. These help in quick access, searching, and updating during algorithm execution.

Reason for Choosing These Structures

Graphs were chosen because city maps naturally represent a network of connected points, which fits perfectly into graph-based modeling. Dijkstra's algorithm efficiently solves the shortest path problem for weighted graphs, while DFS provides a simple yet effective way to explore the entire

network. Together, these demonstrate the use of core data structures and algorithms in practical applications.

Pseudocode for Algorithms

Dijkstra's Algorithm:

```
function Dijkstra(Graph, source, destination):
    for each vertex v in Graph:
        distance[v] = infinity
        previous[v] = undefined
    distance[source] = 0

    while unvisited vertices remain:
        u = vertex with smallest distance
        for each neighbor v of u:
            alt = distance[u] + weight(u, v)
            if alt < distance[v]:
                distance[v] = alt
                previous[v] = u

    return shortest path from source to destination
```

Depth-First Search (DFS):

```
function DFS(node):
    mark node as visited
    for each neighbor in Graph[node]:
        if neighbor not visited:
            DFS(neighbor)
```

Implementation Details

This project was implemented using HTML, CSS, and JavaScript, with Leaflet.js for map visualization and OpenStreetMap for real-world mapping data. The code was written and tested using Visual Studio Code, and the system runs directly in a web browser without any external server.

Users can drop pins on the map to mark locations, which are treated as nodes in a graph. The system calculates and displays routes between these nodes using Dijkstra's algorithm for the shortest path and Depth-First Search (DFS) for exploration. All routes are drawn on the map using Leaflet polylines with different colors to represent each feature.

The main modules include:

- **Input:** User clicks to add locations (nodes).
- **Processing:** Graph-based algorithms calculate routes and traversal paths.
- **Output:** Results are displayed as interactive lines on the map.

Challenges Faced:

- **Integrating Real-World Mapping Data:**
Incorporating a live map interface with Leaflet and linking it to OSRM routing data required careful handling of coordinate systems, asynchronous API requests, and map rendering updates.
- **API Limitations and Reliability:**
The OSRM API occasionally failed to return alternate routes or provided incomplete responses, resulting in inconsistencies in route visualization. Handling these errors gracefully was a key challenge.
- **Dynamic Node and Edge Management:**
Implementing smooth user interactions for adding, selecting, and connecting nodes on the map demanded dynamic graph updates and frequent re-rendering without breaking existing connections.
- **Visualization and UI Clarity:**
Designing a clean, glass-effect interface while ensuring markers, paths, and controls remained visible against various map backgrounds required multiple iterations to strike a balance between style and usability.
- **Performance Optimization:**
As the number of nodes and routes increased, the browser's performance dropped slightly. Optimizing redraws and managing map layers efficiently helped maintain responsiveness.

Results & Analysis

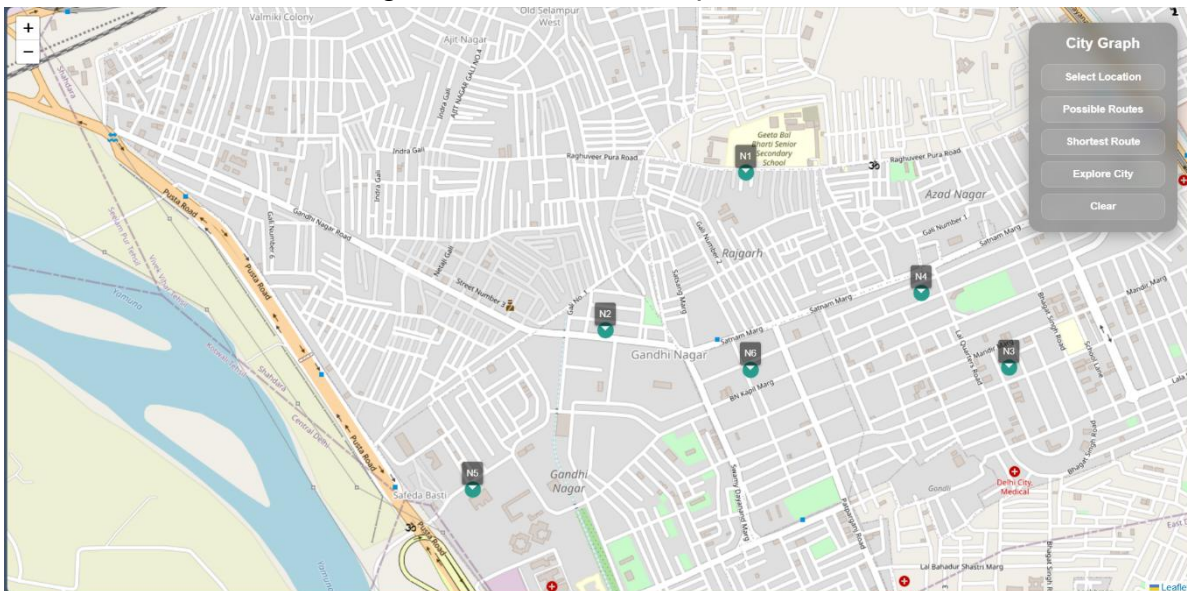
Several test cases were conducted to evaluate the functionality and accuracy of the City Graph system. The application was tested by selecting multiple city locations and generating both shortest and possible routes.

From a performance standpoint, the operations were efficient for small to medium graphs (up to 20–30 nodes). The time complexity of the shortest path calculation using Dijkstra's algorithm is $O(V^2)$ for the current implementation, which is suitable for interactive use. The DFS-based exploration operates with a time complexity of $O(V + E)$, ensuring linear scalability with respect to the number of nodes and edges.

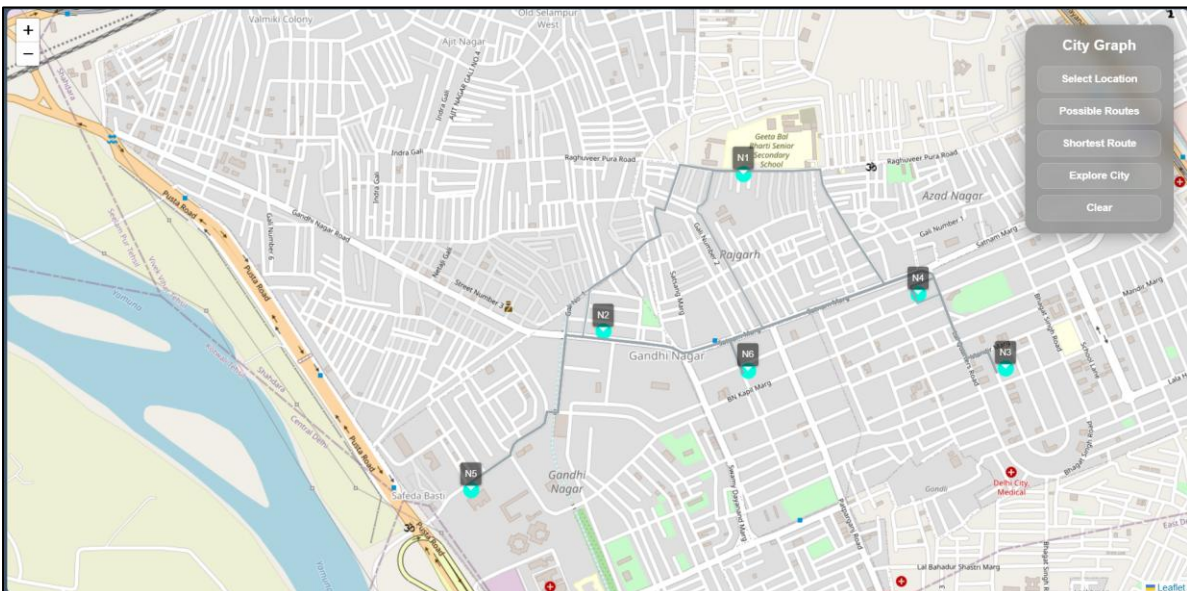
In terms of efficiency, the visual rendering and path computations occurred almost instantly for typical city-scale tests. The results indicate that the system provides accurate and responsive route visualization with minimal computational overhead, making it practical for educational and analytical purposes

OUTPUTS

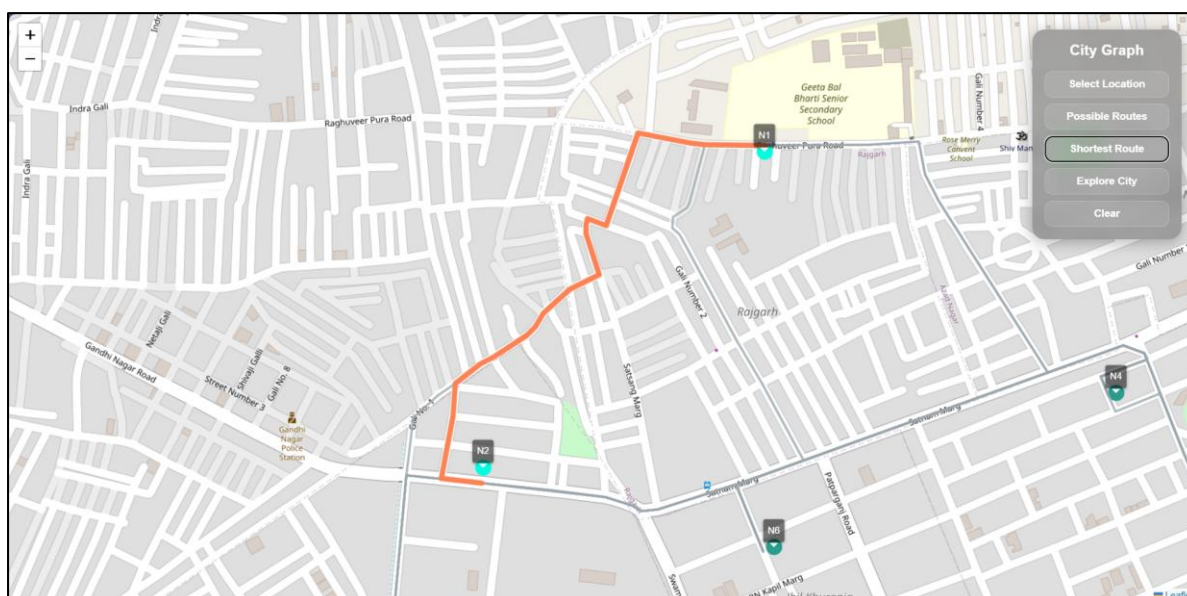
1. Select Location: selecting a location on the map



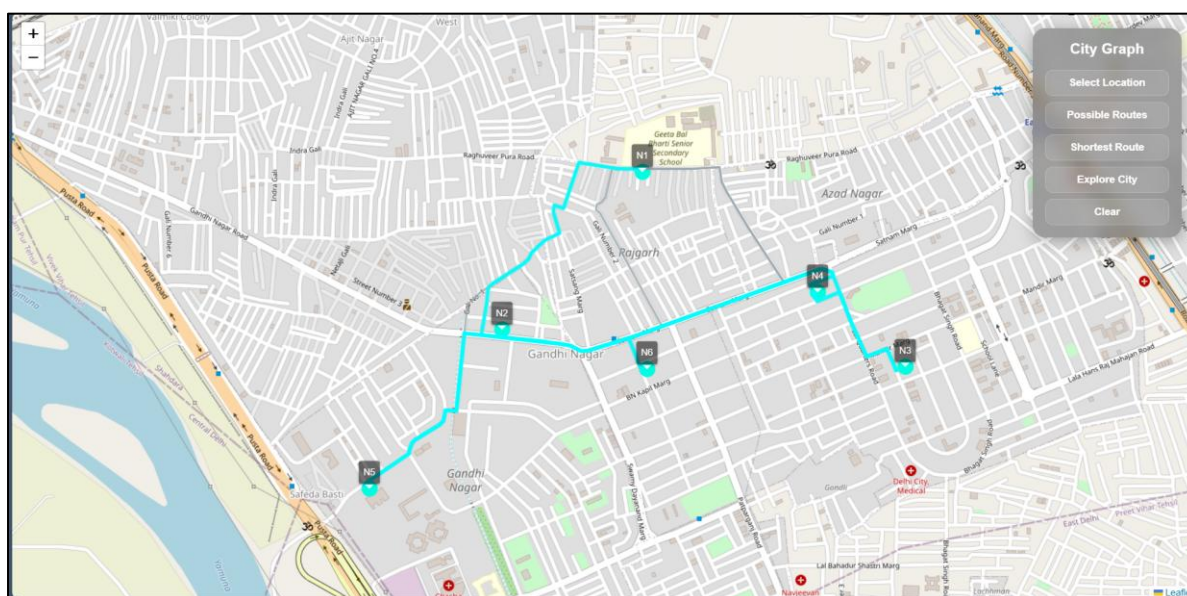
2. Possible Routes: selecting the nodes/location tags (at least two) to find the possible route(s)



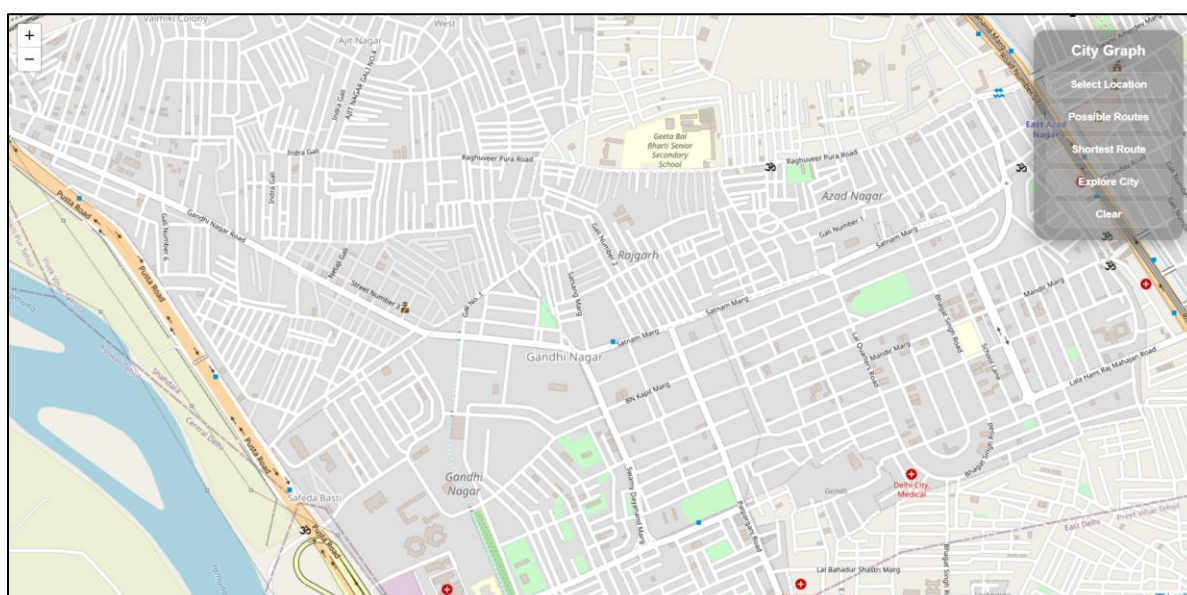
3. Shortest Route: finds the shortest path between 2 selected nodes by Dijkstra's algorithm



4. Explore City: finds the longest route for the selected nodes/location tags using DFS algorithm



5. Clear: clears the nodes/location tags and all the routes from the map



Conclusion

This project demonstrated how graph algorithms can be applied to real-world navigation using an interactive city map. With Dijkstra's algorithm for shortest paths and Depth-First Search (DFS) for exploration, it effectively visualized key data structure concepts such as nodes, edges, and weighted graphs.

Through this work, important skills were developed in algorithm implementation, API integration, and front-end design. Major challenges included handling API limits and ensuring smooth map interactions, which were resolved through testing and optimization.

Future Scope

- **Enhanced Route Generation:**
Integrate advanced routing algorithms (like A* or Yen's K-Shortest Paths) to display multiple realistic alternative routes beyond the OSRM limitations.
- **Offline Map and Routing Support:**
Implement offline caching or local routing logic for areas with poor connectivity or API downtime.
- **Persistent Data Storage:**
Allow users to save and reload their created city nodes and routes using local storage or a database.
- **Interactive Traffic and Cost Analysis:**
Extend the system to include live traffic data or assign weights based on factors such as travel time, fuel cost, or congestion.
- **Improved Visualization:**
Introduce animations, route comparison overlays, and better color-coding to enhance readability and aesthetics.
- **Scalability and Performance Upgrades:**
Optimize the graph handling and rendering for larger datasets or more complex city networks.
- **Mobile Responsiveness and Accessibility:**
Refine the UI for mobile devices and make controls accessible to all users.

References

- Leaflet.js – *An open-source JavaScript library for interactive maps*. Available at: <https://leafletjs.com>
- OpenStreetMap – *Open geographic data used for map rendering*. Available at: <https://www.openstreetmap.org>
- Dijkstra, E. W. (1959). *A note on two problems in connexion with graphs*. Numerische Mathematik, 1(1), 269–271.
- Depth-First Search (DFS) Algorithm – *Standard graph traversal method*. In: Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms*. MIT Press.
- MDN Web Docs – *Official JavaScript and Web API documentation*. Available at: <https://developer.mozilla.org>