

UMM-E-HABIBA IMRAN

01-134241-048

ZAINAB IDREES

01-134241-051

# Data Structures and Algorithms Lab

# Vertex Voyage

**Department of Computer Science**

Lab Instructor: Mohsin Javed Butt

Department of Computer Science

Bahria University, Islamabad

May 2025

**Abstract**

This project, VertexVoyage, presents a graphical application for finding the shortest path between cities in Pakistan using Dijkstra’s algorithm. The underlying data consists of a weighted graph defined by an adjacency list and adjacency matrix, with city nodes connected by edges representing distances. The graph data is stored externally in a structured file (Secret.txt), which details the vertices and their weighted connections. The application supports dynamic switching between adjacency list and adjacency matrix representations to demonstrate algorithm performance and implementation flexibility. A user-friendly interface built with SFML allows input of source, destination, and multiple intermediate stops, providing both user-defined route order and automatically optimized routes that minimize total travel distance. City coordinates for mapping are manually extracted from a high-resolution map image, enabling accurate visualization of routes and key points. The system manages multiple stops with constraints to balance performance and user experience, including validation and feedback for invalid inputs. Overall, VertexVoyage offers an interactive way to explore shortest path computations and route optimizations on a real-world geographical network, combining algorithmic rigor with visual clarity and practical usability.

\

**Chapter 1**

**Introduction**

Efficient route planning is vital in transportation and logistics, especially as city connectivity grows. Dijkstra’s algorithm offers a reliable method for finding shortest paths in weighted networks, where nodes represent cities and edges reflect travel distance or cost. **VertexVoyage** applies this approach to major cities in Pakistan. It builds a graph from real-world data, letting users choose a source, destination, and optional stops. The system offers two routing modes: one that follows the user’s stop order and another that optimizes the route to minimize total travel distance.

* 1. **Problem Statement**

Manual route planning with multiple stops is often inefficient and time-consuming, lacking tools that allow flexible stop order input combined with automated optimization and clear route visualization. Existing navigation solutions may not expose the underlying computations or support comparative analysis of user-defined versus optimized routes. There is a clear need for an interactive system that efficiently computes the shortest paths, supports multiple stops with user control, and visually presents routes on an accurate geographic map.

* 1. **Objectives**

1. To develop VertexVoyage, an interactive graphical application that models city connections as a weighted graph loaded from external data.
2. To implement Dijkstra’s shortest path algorithm supporting both adjacency list and adjacency matrix graph representations, switchable at runtime.
3. To accurately visualize computed routes overlaid on a scaled map using manually extracted city coordinates.
4. To provide functionality for route computation using both user-defined stop order and an optimized stop order that minimizes total travel distance.
5. To handle performance constraints by limiting stops for permutation-based optimization and ensuring smooth user interaction.

**Chapter 2**

**System Design and Architecture**

The VertexVoyage application is designed with a modular architecture to separate concerns between data representation, algorithmic processing, and user interface. The system comprises three main components:

**2.1 Graph Data Structure**

The core of the application is a weighted graph representing cities as nodes and the roads or connections between them as edges weighted by distance. The graph supports two interchangeable representations:

* **Adjacency List:** A map of nodes where each key node stores a map of connected nodes and their edge weights. This representation is memory-efficient for sparse graphs and supports quick iteration over neighbors.
* **Adjacency Matrix:** A 2D vector storing edge weights for all possible node pairs, with infinite weight indicating no direct connection. It enables constant-time edge lookups but consumes more memory, especially for large graphs.

The application allows switching between these representations at runtime to demonstrate differences in algorithmic performance and data access.

**2.2 Data Input and Preprocessing**

Graph data is loaded from an external file (Secret.txt), formatted to specify the total number of vertices followed by adjacency details for each node. Each line includes a node index, followed by pairs of adjacent node indices and edge weights. This modular data input design allows easy modification or extension of the graph without altering the source code.

City coordinates for visualization are manually extracted from a high-resolution Pakistan map image using a pixel mapping tool (e.g., MS Paint). The coordinates map each city node to a fixed (x, y) position on the scaled map displayed in the application.

**2.3 User Interface and Visualization**

The UI, developed with the SFML library, provides interactive input fields for the source city, destination city, and multiple stops. It supports:

* **Input validation:** Ensures entered city names match known nodes.
* **Multiple stop management:** Users can add multiple intermediate stops with feedback on invalid or duplicate entries.
* **Routing options:** Radio buttons allow users to select between following their specified stop order or using an algorithmically optimized stop sequence.

The route computed by Dijkstra’s algorithm is visualized as colored lines connecting city nodes on the map, with distinct markers for the start, stops, and destination. Overlays and fading effects enhance user experience by focusing attention during key interactions.

**Chapter 3**

**Implementation Details**

**3.1 Graph Representation (Adjacency List and Matrix)**

The VertexVoyage project implements the graph using two interchangeable data structures:

* **Adjacency List:** This is a map of maps where each key corresponds to a city node, and the value is a map of adjacent nodes with their respective edge weights. It provides efficient memory usage, especially for sparse graphs, and facilitates fast iteration over neighbors.
* **Adjacency Matrix:** A 2D vector is employed to represent the graph as a matrix where each cell [i][j] stores the distance between city i and city j. If no direct connection exists, the cell holds an infinite value. The matrix allows constant-time edge lookups but consumes more memory, especially for large graphs.

**3.2 Data Input and Parsing (Secret.txt)**

The graph data is loaded from an external file named Secret.txt. The file starts with the total number of vertices (cities), followed by lines where each line corresponds to a node and lists pairs of adjacent node indices and their edge weights.

For example, a line like:

0 1 421 2 534 4 353

indicates that city 0 is connected to city 1 with a distance of 421 km, city 2 with 534 km, and city 4 with 353 km.

Importantly, the data used in Secret.txt is based on real distances between major cities in Pakistan, not hypothetical or estimated values. These distances are sourced from reliable geographic data and verified maps, ensuring the application’s practical relevance and accuracy.

**3.3 City Coordinates and Mapping**

To visualize routes accurately, city coordinates were manually extracted from a detailed map image of Pakistan using pixel measurement tools such as MS Paint. Each city node is assigned a fixed (x, y) coordinate corresponding to its approximate location on the scaled map. These coordinates enable the application to draw paths and markers that reflect real world geographic relationships between cities.

**3.4 User Interface Design**

The user interface is built with the SFML graphics library and provides:

* **Text input boxes** for source city, destination city, and multiple stop cities, featuring real-time input handling and validation.
* **Buttons and radio toggles** to add stops, find paths, and switch between user-defined stop orders and optimized routing.
* **Visual feedback** such as error messages for invalid inputs or performance warnings when too many stops are added.
* **Dynamic overlays and animations** to improve user experience during computations and path displays.
* **Map display** with the underlying Pakistan map scaled appropriately, over which routes are drawn with colored lines and city markers.

**3.5 Route Computation and Optimization**

Shortest path computation leverages Dijkstra’s algorithm implemented to work with both graph representations. The algorithm calculates minimal distances between nodes, building routes segment by segment. The application offers two routing modes:

* **User-defined stop order:** The path is computed respecting the order in which stops were added.
* **Optimized stop order:** The system permutes stop sequences to find the route with the shortest total distance, akin to solving a constrained Traveling Salesman Problem for the limited set of stops.

Due to the factorial growth of permutations, the application limits the number of stops to maintain responsiveness.

**Chapter 4**

**Features and Functionality**

**4.1 Multiple Stops Handling**

VertexVoyage supports adding multiple intermediate stops between the source and destination cities. Users can enter multiple stop cities separated by commas, with real-time validation to ensure city names are recognized and not duplicated. To maintain optimal performance, the application limits the number of stops to a manageable count, preventing excessive computation time during route optimization.

**4.2 User Defined vs Optimized Routes**

The application offers two routing options:

* **User-Defined Stop Order:** Routes are computed by following the exact sequence in which the user adds the stops. This mode gives users full control over their travel itinerary.
* **Optimized Stop Order:** The system automatically calculates the shortest possible route by permuting the stops to minimize the total travel distance. This option leverages combinatorial optimization to find a route closer to the theoretical shortest path, improving travel efficiency.

Users can toggle between these modes to compare their planned route with the algorithmically optimized alternative.

**4.3 Mode Switching (Adjacency List vs Matrix)**

To demonstrate different graph representations and their effects on algorithm performance, VertexVoyage allows switching between:

* **Adjacency List Mode:** Efficient for sparse graphs, providing faster neighbor traversal with lower memory usage.
* **Adjacency Matrix Mode:** Enables constant-time edge lookups but uses more memory and can be less efficient for sparse graphs.

This flexibility offers educational insight into data structure impacts on pathfinding algorithms and supports performance experimentation.

**Chapter 5**

**Challenges and Solutions**

During the development of VertexVoyage, several challenges were encountered and addressed to ensure functional robustness and user experience:

**5.1 Handling Multiple Stops and Performance Constraints**

**Challenge:** Permuting multiple stops to find the shortest possible route introduces factorial time complexity, which can quickly become computationally infeasible with an increasing number of stops.

**Solution:** The application limits the number of stops to a practical maximum (e.g., eight) to maintain responsiveness. Additionally, clear feedback is provided to users when too many stops are entered, advising them of potential performance issues.

**5.2 Accurate Geographic Visualization**

**Challenge:** Mapping abstract graph nodes to real geographic locations requires precise coordinates to reflect realistic routes on the displayed map.

**Solution:** City coordinates were manually extracted from a high-resolution map using pixel measurement tools to assign accurate (x, y) positions. This approach enabled visually coherent route rendering aligned with the real-world map.

**5.3 Flexible Graph Representation Switching**

**Challenge:** Supporting both adjacency list and adjacency matrix representations and switching between them at runtime complicated the graph implementation and algorithm integration.

**Solution:** A unified Graph class was designed with internal management of both data structures. The Dijkstra algorithm accesses edges based on the active mode, allowing seamless switching without disrupting computations.

**5.4 User Input Validation and Error Handling**

**Challenge:** Ensuring that user-entered city names match existing nodes and handling invalid or duplicate inputs without crashing the program required robust validation.

**Solution:** The system employs case-insensitive matching, trimming of whitespace, and duplicate detection. Clear error messages inform users of invalid inputs, enhancing usability and preventing runtime errors.

**5.5 Visual Feedback and User Interface Responsiveness**

**Challenge:** Managing user expectations during path computation, especially when optimization may take time, and providing a visually appealing interface.

**Solution:** The interface incorporates overlay fading effects during computation, colored route lines, and intuitive input controls. These elements improve user engagement and clarity about the application’s state.

**Chapter 6**

**Results and Evaluation**

The VertexVoyage application successfully computes and visualizes shortest paths between major cities in Pakistan, demonstrating both algorithmic accuracy and practical usability.

**6.1 Path-Finding Accuracy**

Using real-world distance data loaded from the Secret.txt file, the application consistently generates the shortest paths verified against known geographic routes. Both adjacency list and adjacency matrix modes produce identical results, confirming the correctness of the implementation.

**6.2 Performance**

* **Adjacency List Mode:** Offers faster computation and lower memory usage, especially beneficial for sparse graphs like the one modeled.
* **Adjacency Matrix Mode:** Provides constant-time edge lookups but with increased memory footprint and slightly slower performance due to processing the entire matrix.

Switching between these modes allows users and developers to understand trade-offs in graph representation.

**6.3 Multiple Stops and Route Optimization**

The feature of inputting multiple stops and toggle between user-defined and optimized stop orders was evaluated for effectiveness:

* For a small number of stops (up to 6), the permutation-based optimization finds routes that reduce travel distance compared to user input.
* Beyond a certain number of stops, computation time increases factorially, necessitating limits to maintain user experience.

**Chapter 7**

**Limitations**

Despite its strengths, VertexVoyage has several limitations:

1. **Scalability of Stop Optimization:** The brute-force permutation approach for optimizing multiple stops leads to factorial growth in computation time. This restricts practical use to a limited number of stops (typically fewer than eight) to maintain responsiveness.
2. **Static Map Visualization:** The current visualization uses fixed city coordinates on a static map image, without support for zooming, panning, or dynamic resizing, which limits detailed geographic exploration.
3. **Input Flexibility:** City name matching is case-insensitive but requires exact spelling without tolerance for minor typos or alternate spellings, which may affect user convenience.
4. **Lack of Real-Time Data:** Distance data is static and based on predefined values in Secret.txt. The application does not integrate real-time traffic, road closures, or alternative routing criteria.
5. **Limited Route Customization:** The system focuses on shortest distance routing and does not consider other factors such as travel time, road quality, or user preferences.

**Chapter 8**

**Conclusion**

VertexVoyage successfully demonstrates the practical application of graph algorithms, specifically Dijkstra’s algorithm, for shortest path computation and route optimization between cities in Pakistan. By integrating flexible graph representations, user-friendly multi stop input, and clear geographic visualization, the project provides both educational insight and functional utility in route planning. The ability to compare user-defined stop orders with algorithmically optimized routes empowers users to make informed travel decisions. Although limited by computational constraints and static visualization, VertexVoyage lays a strong foundation for more advanced and scalable routing systems. Overall, the project highlights the importance of combining algorithmic rigor with intuitive interfaces to solve real-world problems effectively.

**Chapter 9**

**Future Work**

Building upon the current capabilities of VertexVoyage, several enhancements can be pursued to improve functionality, scalability, and user experience:

1. **Advanced Route Optimization Algorithms:** Incorporate heuristic or metaheuristic algorithms such as Genetic Algorithms, Ant Colony Optimization, or Simulated Annealing to efficiently handle a larger number of stops beyond the factorial limitation of brute-force permutations.
2. **Dynamic Map Interaction:** Implement features such as zooming, panning, and dynamic resizing of the map to allow users better geographic exploration and route inspection.
3. **Enhanced Input Handling:** Integrate fuzzy matching and autocomplete for city names to accommodate minor spelling errors and improve usability.
4. **Real-Time Data Integration:** Incorporate real-time traffic data, road conditions, and alternate routing preferences to provide more practical and up-to-date route recommendations.
5. **Additional Route Metrics:** Extend optimization criteria beyond shortest distance to include travel time, fuel consumption, or scenic routes, offering users personalized travel options.
6. **Mobile and Web Application Development:** Adapt the system for deployment on mobile and web platforms to increase accessibility and convenience.