Map Navigator

Omar Abdel Motalb, Ramy Zayed, Mahinour Abdelgawad, Seif Ramy, & Saif Taher

Department of Computer Science and Engineering, The American University in Cairo

Applied Data Structures (CSCE2211)

Spring 2025

Project Report

Abstract

This project presents a robust and feature-rich map navigation application developed in C++ using Qt framework, integrating Dijkstra's algorithm for the optimization of path finding and Levenshtein distance for intelligent input correction as a bonus feature showing the passion of the individuals behind said project. The application is also equipped with a File handling system and the underlying graph class is designed to handle the dynamic interactions between the nodes as well as the edges.

Keywords:

Levenshtein distance, Dijkstra algorithm , Path-finding algorithm, Graph theory, Auto Correction

Introduction

The world became a small house with the rise of technology and smart applications connecting every individual in this blue planet. Everywhere became accessible as if each city represents a node connected to other nodes filling the complex of this smart-based world. The map navigator application aims to present such a concept and develop an intelligent pathfinding system for major American cities so that every user of the application can conveniently find the perfect path to the destination that the user will proceed with.

Problem Definition

People tend to find the most suitable and convenient way to accomplish a task. Travelling to another city is no exception to this rule, which is why map navigation applications came to realization. However, map navigation applications must compute optimal routes between locations while also managing inaccurate user input. Classical systems often fail to handle misspelled or approximate names, resulting in a poor user experience. Additionally, large scale networks require the conjunction of efficient representation and traversal methods. The problem addressed in this project is twofold:

1. Calculating the shortest path between two user-specified nodes in a complex network.
2. Handling typographical errors or approximate string inputs during location searches.

Methodology

The application follows a sophisticated approach which consists of five steps. First, the implementation and design of a Graph class that represents the map as a network of vertices and edges. Applying Dijkstra's algorithm to find the shortest path between nodes. Implementation of Levenshtein distance for correctness and accurate suggestions of the likely string matches of the user's input. Developing a Graphical User interface using Qt as a framework and pushing for a more user-friendly experience. Finally, testing the system of the application with various datasets input of different maps and sizes to ensure the robustness of the application.

Algorithm Specifications

This application relies on three main algorithms:

1. Dijkstra's algorithm: used to compute the shortest path from a starting city to the destination city. The algorithm implemented maintains a distance vector initialized to infinity and iteratively relaxes the edges from the closest unvisited nodes till it constructs the shortest path between the two nodes of the cities given as parameters. It is implemented through a linear search algorithm with a time complexity of $O(n^2)$ where N is the number of vertices and E is the number of edges connected to a certain edge at a period of time, making it perfect for small to medium sized graph maps.

2. Levenshtien distance algorithm: This dynamic algorithm calculates the minimum number of edits which are insertion, deletion and substitution. The number is required to transform one string to another and it is used to correct the user's misspell cities input. The time complexity of this algorithm is O(mxn) where m and n are the lengths of the strings inserted into the algorithm. The algorithm iterates through a dictionary containing the strings of all the American cities of the map and constructs a 2D dynamic programming matrix (DP) with the number of rows and columns of the matrix being the lengths of the input string and the dictionary string respectively. Then it fills the DP matrix with the minimum calculated from the three operations mentioned. After filling the matrix, the Levenshtein distance is stored in the lower-left corner, i.e., dp[m][n]. This keeps recurring over the whole dictionary until the lowest distance is reached and the nearest match is found.

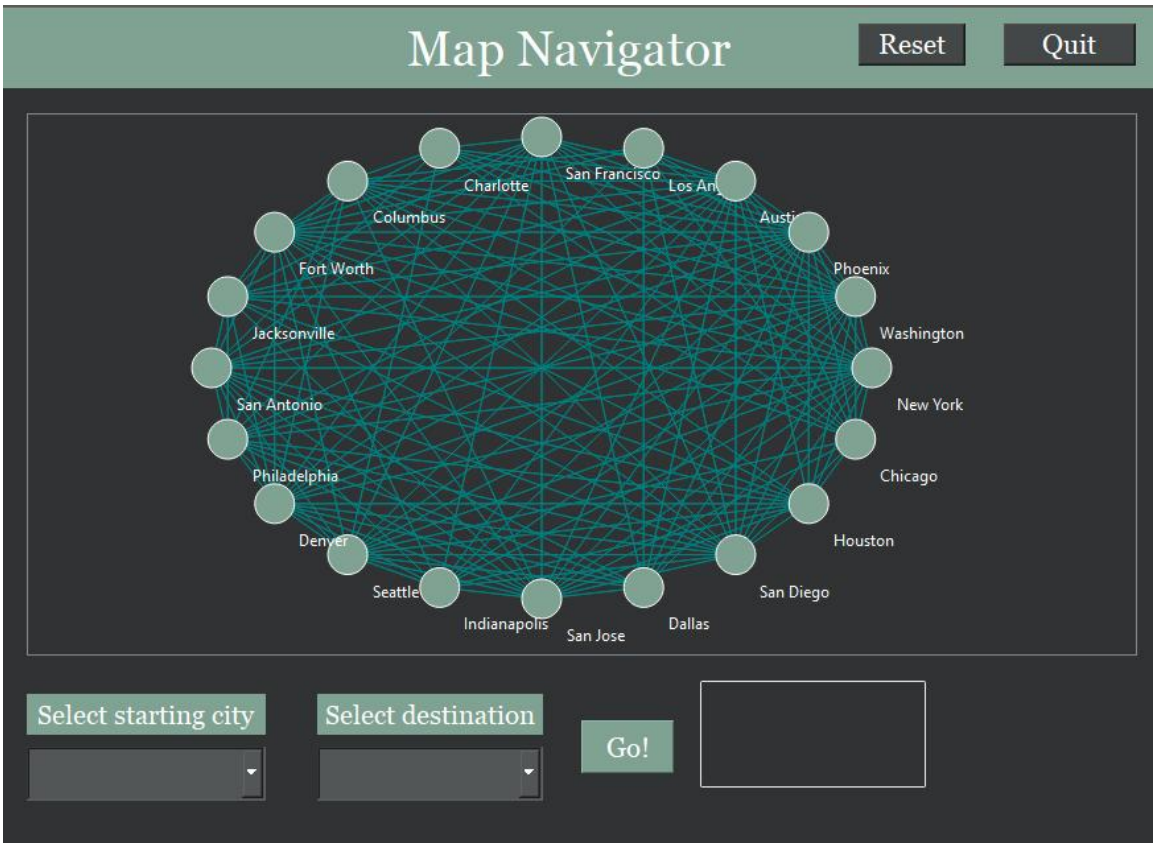
3. Graph construction from File (File Handler): A helper routine function that reads and trims text lines containing American city names and edge weights. The file handler ignores the empty lines and comments. It automatically trims whitespace from city names and adds nodes as well as edges dynamically.

Data Specifications

1. A list of nodes e.g. cities and their weighted edges (distances). This is provided by the txt file which is handled robustly by the File Handler class.
2. User input strings, handled by the Spell Checker class, or inserting of an item from a checkbox provided for more leeway and a more friendly user experience.

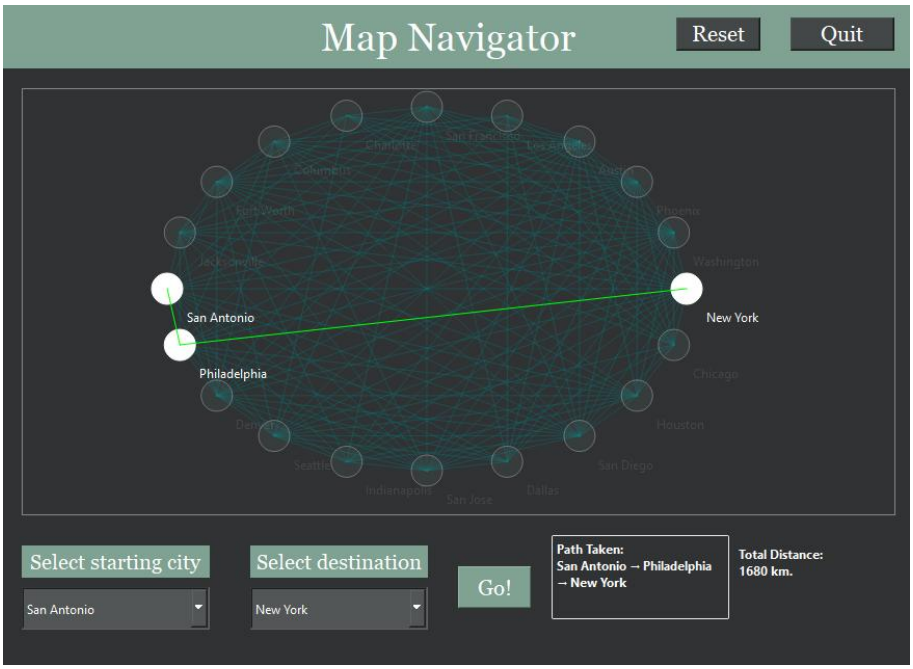
Experimental Results

The figure below shows the entire subspace of the map which simulates major American cities as nodes filling the complex of the graph map:



The second figure below demonstrates the practical use of the map navigator application. The user made a typo in writing the American city “San Antonio”. Hence, the spell checker implemented in the program tackled the issue and made the proper adjustments to the city name.

The last figure below shows the main use of the application where it figured the shortest distance between the two cities is by taking a shortcut route through the city of Philadelphia.



Analysis and Critique

The system successfully demonstrates a reliable routing technique with intelligent output and robust input correction. At its core, the application provides an efficient pathfinding technique using Dijkstra's algorithm, robust input handling with Levenshtein distance and scalable graph representation of the American cities using a user-friendly graphical interface.

The use of Dijkstra's algorithm ensures optimal pathfinding by calculating the shortest distance between the starting city and the destination city by visiting each node and relaxing all the edges till the optimal route is stored in the previous vector. The implementation of the Spell Checker ensures the robustness of the program using the Levenshtein distance as a means to its validity. Through the calculation of the minimum number of edits needed to change a string to another using the dynamic programming matrix, all user typos are corrected and the program has a meaningful layer of fault tolerance such as identifying and correcting "San Antonio" from a misspelled input. The application integrates the Qt framework as its interface which enhances user interactivity and the usability of the system.

The only limitation that the application conceives is that the current Dijkstra's algorithm uses linear search approach, resulting in a time complexity of $O(n^2)$. While this is totally acceptable for this application since it is made only to navigate through small to medium-sized datasets such as the major American cities. But if the dataset keeps on increasing, the performance significantly degrades as the number of nodes increases. So, the application is limited to a medium or lower sized number of graphs. Since with a larger number of datasets, it degrades the GUI performance and becomes less user friendly.

Conclusion

This project effectively demonstrates how algorithmic techniques and software engineering can be integrated to build a smart and user-friendly map navigation system. Through the use of Dijkstra's algorithm and Levenshtein distance, the application provides reliable routing even in the presence of input errors. The project highlights the importance of clean architecture, efficient data structures, and human-centered design. This project came to realization thanks to the amazing team it had as well as the tutoring of the professor of the course and other algorithmic scientists referenced below.

References

Javaid, A. (2013). Understanding Dijkstra's algorithm. *Available at SSRN 2340905*.

Miller, Frederic P., Agnes F. Vandome, and John McBrewster. "Levenshtein distance: Information theory, computer science, string (computer science), string metric, damerau? Levenshtein distance, spell checker, hamming distance." (2009).

Appendix

The GitHub repository that stores the map navigation application code:

https://github.com/IOI-dot/Map-Navigator.git