MINI PROJECT
On

Travel Planner using Graphs

In
Data Structure & Algorithms

BACHELOR OF TECHNOLOGY
IN
Artificial Intelligence and Machine Learning

SUBMITTED BY

Soham Shirish Kulkarni
(PRN - 23070126129)

Under the Supervision of

Prof. Ashwini Tande

SYMBIOSIS INSTITUTE OF TECHNOLOGY, PUNE – 412115

(A CONSTITUENT OF SYMBIOSIS INTERNATIONAL (DEEMED UNIVERSITY))

2024-25

1. Problem Statement

The shortest path problem is a classic and fundamental question in graph theory, concerned with finding the optimal route between two nodes in a graph such that the sum of edge weights along the path is minimized. This concept has broad applications across various fields, from computer networks to logistics and urban planning. In a travel planning context, solving the shortest path problem allows us to design an intelligent system that can suggest the most efficient route for travelers navigating between destinations. This project models cities as nodes and direct travel routes between them as weighted edges, where edge weights can represent critical travel metrics, such as distance, time, or cost. By leveraging these graph structures, the travel planner system is able to compute optimal routes for users, helping them minimize travel time, reduce transportation costs, or simply travel shorter distances.

In a practical setting, travel planning often requires navigating between multiple locations, each with potentially varying travel options. This becomes particularly complex as the number of possible routes and destinations grows, leading to a need for algorithmic solutions that can efficiently manage these options and select the best route based on specified criteria. In this project, Dijkstra's algorithm is employed to solve the shortest path problem for a single-source to single-destination setup. Dijkstra's algorithm is well-suited for this purpose, as it reliably identifies the shortest path in graphs with non-negative edge weights, making it ideal for planning scenarios where distances or time values are positive.

This project aims to develop a tool that simplifies the decision-making process for travelers by leveraging shortest path algorithms to help them select the best routes. By abstracting travel routes and destinations into graph elements, this travel planner provides a versatile framework for calculating optimized routes across different metrics, whether the goal is to reduce travel time, minimize distance, or lower overall costs.

---

2. Motivation

Efficiently planning travel routes is critical in a range of scenarios, from personal commutes to large-scale logistics. Whether for daily commutes or holiday travel, individuals constantly seek to optimize their routes to save time, reduce costs, or simply take the shortest path. However, when multiple possible destinations and paths are involved, identifying the best route becomes challenging without an organized method. By applying graph theory and algorithmic techniques, such as the shortest path algorithms, a travel planner can help users make smarter decisions by automatically evaluating and recommending routes that best meet their criteria.

In addition to individual travel, businesses and logistics companies rely heavily on route optimization to manage operations effectively. Delivery services, for example, use similar algorithms to streamline the routes taken by their delivery vehicles, thus minimizing fuel usage and reducing time on the road. Ride-sharing platforms, public transportation systems, and even emergency response planning also benefit from route optimization techniques that determine the most efficient paths in complex urban environments. In all of these cases, having a travel planner that can quickly adapt to changes in route preferences—such as prioritizing shortest distance over minimum cost—provides significant value.

This project introduces a streamlined solution for travelers who may not have the resources or time to manually calculate and compare potential routes. By inputting their start and end destinations and specifying their preferences, users can effortlessly receive optimized route

suggestions based on the shortest path calculation. The approach used in this project not only provides travel convenience but also has implications for sustainable travel planning, as shorter or faster routes can directly contribute to reducing overall fuel consumption and environmental impact. This integration of graph-based optimization into a travel planning system demonstrates the power of mathematical algorithms in solving real-world problems and enhancing the efficiency of everyday activities.

---

3. Objectives

Develop a Graph-Based Travel Network:

- **Design and implement a graph structure where cities are represented as nodes and travel routes between them are edges with associated weights (distances).**

Implement Efficient Pathfinding Using Dijkstra's Algorithm:

- **Utilize Dijkstra's algorithm to compute the shortest path between two cities, minimizing travel distance or time, and providing accurate route information to users.**

Provide an Interactive User Interface for City and Route Management:

- **Enable users to dynamically add cities, define routes, and view the current travel map, allowing for easy updates and modifications of the city network.**

Ensure Robust Input Validation and Error Handling:

- **Incorporate input validation to ensure that city names, route distances, and user choices are correctly formatted and logically consistent, while providing helpful error messages when necessary.**

Deliver a Scalable and Flexible Travel Planning Tool:

- **Build a solution that can handle a growing number of cities and routes, with the potential for future expansion, such as incorporating different types of optimization (e.g., travel time, cost) and scalability improvements.**

---

4. Methodology of Implementation

Flowchart / Pipeline Steps:

1. **Input Locations and Routes**
   Users enter city names and routes between cities. Distances are provided for each route, forming weighted edges in the graph structure.
2. **Graph Construction**
   The code uses an adjacency matrix to represent city connections. Each city (node) holds distances to other cities, making up the edges of the graph.
3. **Algorithm Selection**
   For this project, Dijkstra's algorithm was chosen due to its efficiency in finding the shortest path between two nodes when all edges have non-negative weights.
4. **Shortest Path Calculation**
   Dijkstra's algorithm calculates the shortest path from the selected start city to the destination, iteratively finding the minimum cumulative distance.
5. **Output Display**
   The program presents the optimal route to the user, showing the total travel distance and estimated travel time.

Explanation of Pipeline Components:

- **Input Locations and Routes: Cities and routes are entered by the user, and distances are set to represent the cost or time of travel.**
- **Graph Construction: Cities are nodes, and distances between them (or the absence of a connection) create the adjacency matrix. `addRoute` assigns weights between cities.**
- **Algorithm Selection: Dijkstra's algorithm efficiently calculates paths for non-negative weights and is a robust choice for this single-pair pathfinding.**
- **Shortest Path Calculation: Dijkstra's algorithm minimizes path distance, starting from the source city and updating shortest distances to connected cities until reaching the destination.**
- **Output Display: The shortest path and corresponding details are shown, making route options clear to the user.**

---

5. Hardware/Software Used

- **Hardware: Any standard computer.**
- **Software:**
  - **C Programming Language: The core implementation language.**
  - **GCC: For compilation**
  - **Bash: For execution**

---

6. Executable Code

The code implements the graph structure and shortest path calculation using Dijkstra's algorithm. The entire codebase is also available on Github at the following URL:

https://github.com/netherquark/travelplanner

The following functions demonstrate the main components:

```
int addRoute(const char *fromCity, const char *toCity, int distance) {
   if (distance <= 0) {
       printf("Error: Distance should be a positive integer.\n");
       return 0;
   }

   int fromIndex = findCityIndex(fromCity);
   int toIndex = findCityIndex(toCity);

   if (fromIndex == -1) {
       printf("Error: Source city '%s' not found.\n", fromCity);
       return 0;
   }
   if (toIndex == -1) {
       printf("Error: Destination city '%s' not found.\n", toCity);
       return 0;
   }

   cities[fromIndex].distances[toIndex] = distance;
   cities[toIndex].distances[fromIndex] = distance;
   printf("Route from '%s' to '%s' with distance %d added successfully.\n",
fromCity, toCity, distance);
   return 1;
}
```

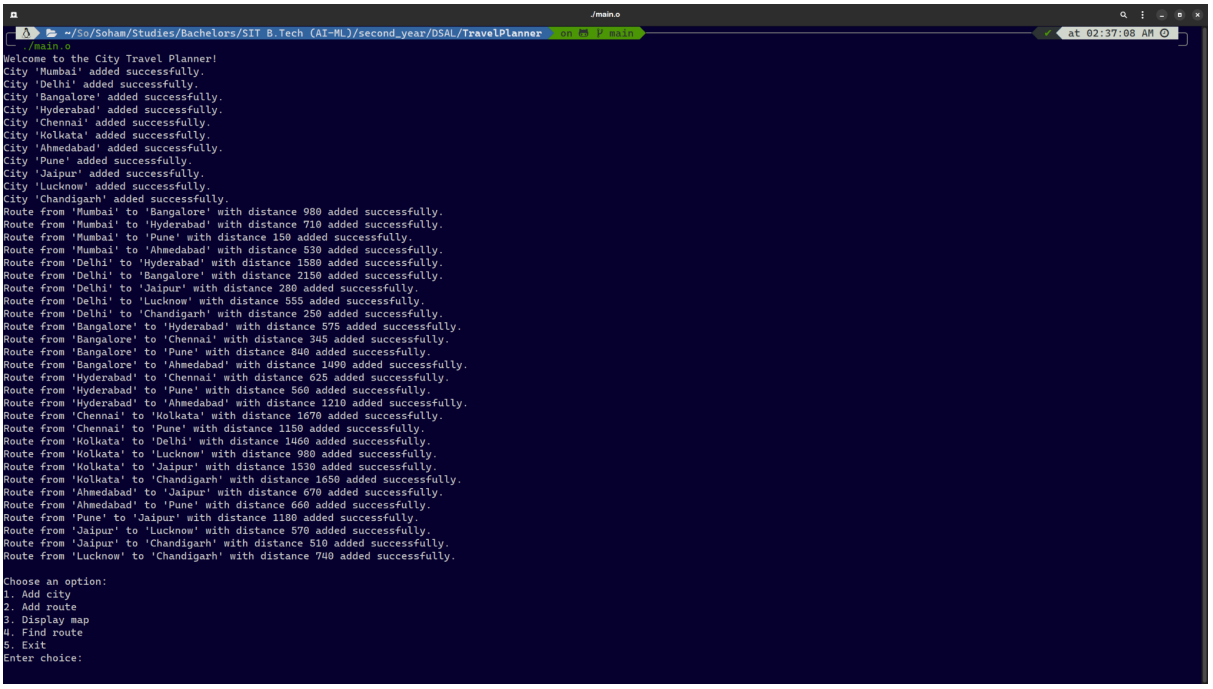
```
void dijkstra(int start, int end) {
   int distance[MAX_CITIES];
   int visited[MAX_CITIES] = {0};
   int parent[MAX_CITIES];
   for (int i = 0; i < cityCount; i++) {
       distance[i] = INF;
       parent[i] = -1;
   }
   distance[start] = 0;
   for (int count = 0; count < cityCount - 1; count++) {
       int min = INF, u = -1;
       for (int v = 0; v < cityCount; v++) {
           if (!visited[v] && distance[v] < min) {
               min = distance[v];
               u = v;
           }
       }
       if (u == -1) break;
       visited[u] = 1;
       for (int v = 0; v < cityCount; v++) {
           if (!visited[v] && cities[u].distances[v] != INF &&
               distance[u] + cities[u].distances[v] < distance[v]) {
               distance[v] = distance[u] + cities[u].distances[v];
               parent[v] = u;
           }
       }
   }
   if (distance[end] == INF) {
       printf("No route exists from '%s' to '%s'.\n", cities[start].name,
cities[end].name);
```

```
        return;

    }



    printf("Shortest path from '%s' to '%s': %d km\n", cities[start].name,
cities[end].name, distance[end]);

    printf("Path: ");

    int path[MAX_CITIES];

    int path_index = 0;

    for (int v = end; v != -1; v = parent[v]) {

        path[path_index++] = v;

    }

    for (int i = path_index - 1; i >= 0; i--) {

        printf("%s", cities[path[i]].name);

        if (i > 0) printf(" -> ");

    }

    printf("\n");

}
```

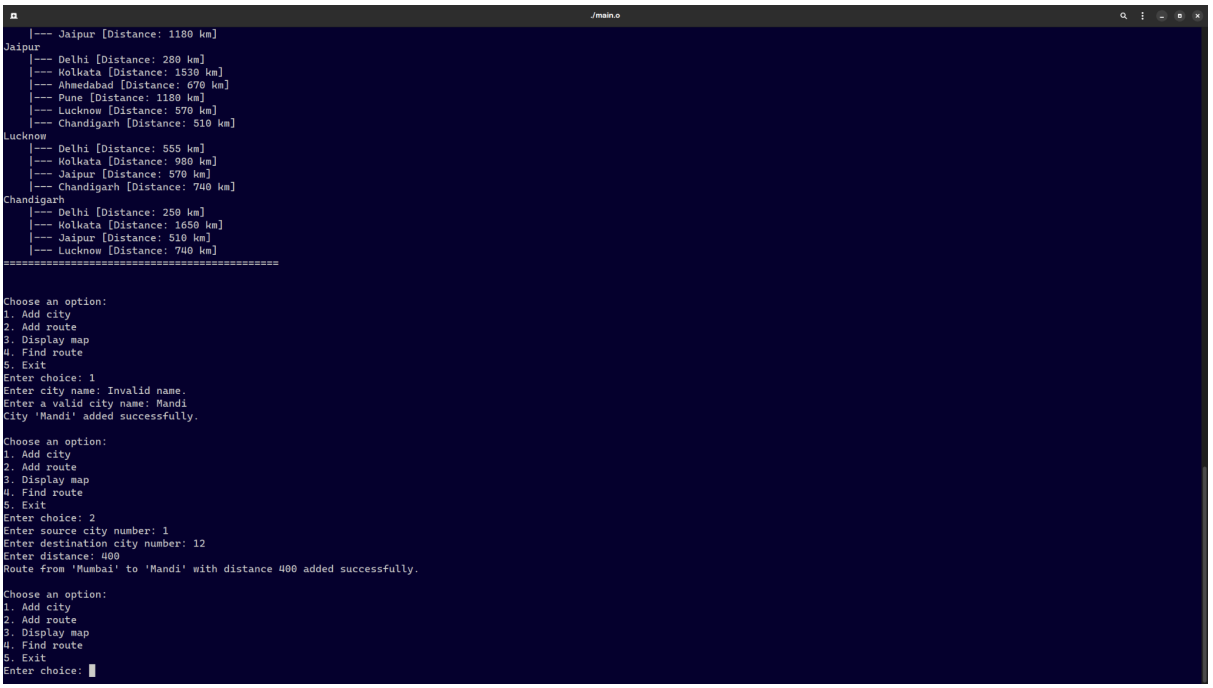
---

7. Result Analysis with Output Screenshot

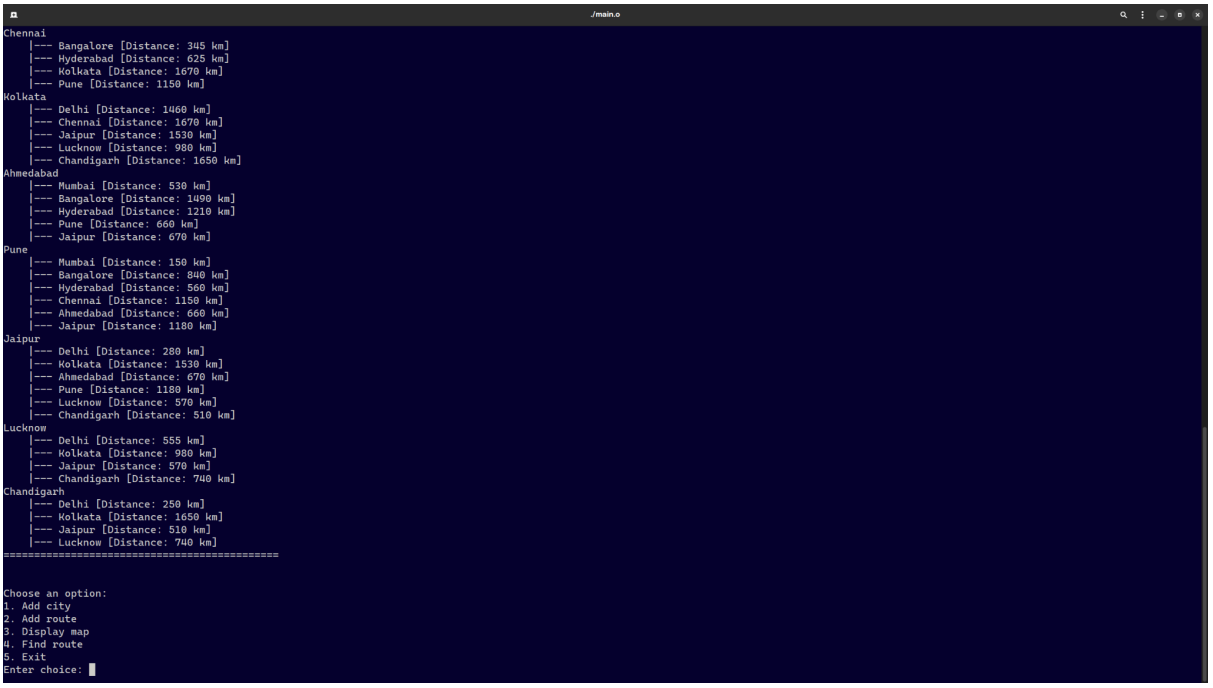
Users can select two cities, and the program computes the shortest path using Dijkstra's algorithm. The output includes the optimal route and total travel distance.



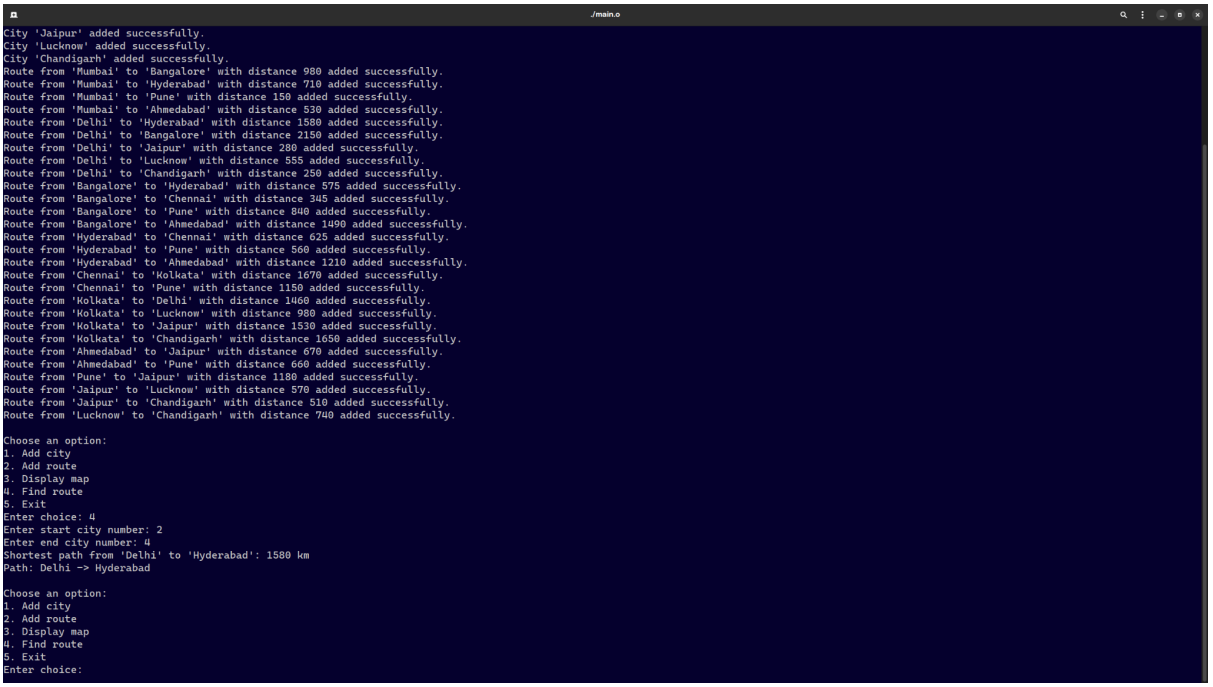
Program Initialisation



Add City & Route



Display Map



Find Route

8. Implementation of Symbol Table and AVL Tree

- **Symbol Table: Could be used for city data storage, allowing efficient retrieval of city information such as names and coordinates. This supports efficient management of travel information.**
- **AVL Tree: For potential extension, an AVL Tree could organize cities by attributes like distance. Its balanced structure supports efficient data retrieval, especially in large-scale travel networks.**

---

9. Learning Outcome

This project provided practical experience in applying graph theory and shortest path algorithms to real-world challenges. Implementing Dijkstra's algorithm enhanced my understanding of graph traversal, while exploring Symbol Tables and AVL Trees broadened my knowledge of modern data management techniques.

References

1. **Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009).** ***Introduction to Algorithms*** **(3rd ed.). MIT Press.**
2. **Dijkstra, E. W. (1959).** ***A note on two problems in connection with graphs*****. Numerische Mathematik, 1(1), 269–271.**
3. **Cherkassky, B. V., Goldberg, A. V., & Radzik, T. (1996).** ***Shortest paths algorithms: Theory and experimental evaluation*****. Mathematics of Operations Research, 22(3), 504–528.**
