CSC3002 Project Report - Urban Navigation System

Zizhou Zhang (122090787), Yuquan Cai (122090007), Jiayi Yao (120040070), Jiale Yuan (122090702)

Contents

1 Project Abstract

In this project, we have implemented an urban (Longgang District) navigation system with Dijkstra's algorithm.

2 Introduction

Navigation systems play a crucial role in various domains, ranging from naive navigation applications to robotics and autonomous vehicles. These systems rely on sophisticated algorithms to find optimal paths from one location to another, considering various constraints such as obstacles, terrain, and time.

Brute-force Search: While brute-force search (e.g., DFS/BFS) is able to find the optimal path and is very easy to implement, it can be very inefficient since nodes/edges could be visited for multiple times.

Dijkstra's Algorithm: Dijkstra's algorithm, named after Dutch computer scientist Edsger W. Dijkstra, is a widely used algorithm for finding the shortest path between nodes in a graph. It works by iteratively selecting the node with the smallest tentative distance from the source and updating the distances of its neighboring nodes.

A* Algorithm: A* is a heuristic search algorithm that combines the advantages of both breadth-first search and Dijkstra's algorithm. It uses a heuristic function to estimate the cost of reaching the goal from a given node, guiding the search towards the most promising paths. A* evaluates nodes by combining the cost of reaching the node from the start node (g-value) and the estimated cost of reaching the goal from the node (h-value). A* is more sclable than Dijkstra in that a well defined heuristic function could accelerate the search process.

Goal-Biased Rapidly-exploring Random Tree: Goal-Biased Rapidly-exploring Random Tree (RRT) is commonly used for motion planning in robotics and navigation systems. It builds a tree of possible paths from the start to the goal by randomly sampling the state space and connecting the samples to the existing tree. Goal-biased RRT variants bias the sampling towards the goal region, speeding up the exploration process and focusing on finding paths towards the goal. While Goal-biased RRT could be faster than A* search, the former does not guarantee an optimal solution.

3 Our Navigation System

3.1 System Overview

The system is a pathfinding tool designed to provide navigation solutions for Longgang District. It is backended with Dijkstra's algorithm and is written in C++. Note that we utilize Dijkstra's algorithm in that it is suitable for graphs with a moderate number of nodes (i.e. Longgang District). For larger-scale navigation systems or systems used in more complex and dynamic environment, more efficient algorithms like A* algorithm or Goal-based RRT should be considered.

3.2 System Implementation

Data Structures: The Node struct represents points on the map, each with a unique identifier and the corresponding coordinates. The Edge struct represents connections between nodes, with attributes for a unique identifier, ids of the connected nodes, and the length of the edge.

Path-finding algorithm: The findway function uses Dijkstra's algorithm to compute the shortest path. First initializes a distance array distance with infinity and sets the distance of the start node to 0. Then uses a priority queue pq to store nodes sorted by distance for efficient selection of the node with the smallest distance.

In the main loop, While the priority queue is not empty: Dequeues the node with the smallest distance (curr) from the priority queue. Then iterates through all edges starting from node curr. For

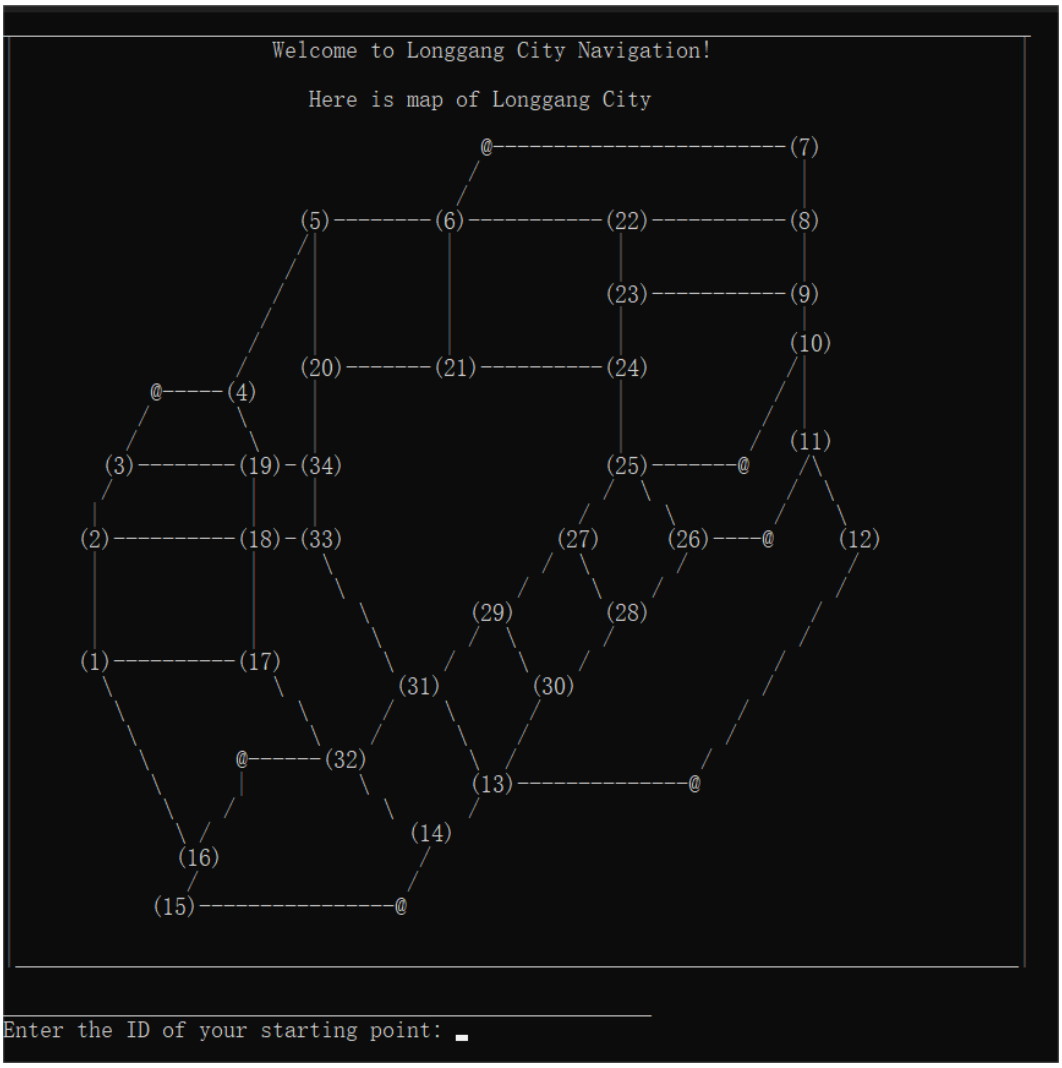

Figure 1: A simple abstracted map of Longgang District

each edge, checks if it starts from node curr. If yes, calculates the new distance to the edge's end node and updates the distance, predecessor, and predecessor edge if the new distance is smaller. After the round Pushes the updated distance and node index into the priority queue for further processing.

After the algorithm finishes, it constructs the shortest path node sequence path and edge sequence edgePath based on predecessor node information. It starts from the end node (endNodeId) and follows predecessor nodes and edges to reconstruct the path. Then reverses the path and edgePath vectors to get the correct order.

System Input/Ouput: Nodes and edges are loaded from text files using readNodeData and readEdgeData functions, respectively. These files are fixed and contain the basic geographical abstraction of Longgang District. Users are all allowed to enter their desired starting point and destination within Longgang District. The output is simply the optimal path (i.e. shortest path) between the user provided origin and the destination.

3.3 Testing and Demo

Figure 1 shows a simplified version of map of Longgang District. Figure 2 shows the user interface, including both the system prompts, user input and the final output optimal path.

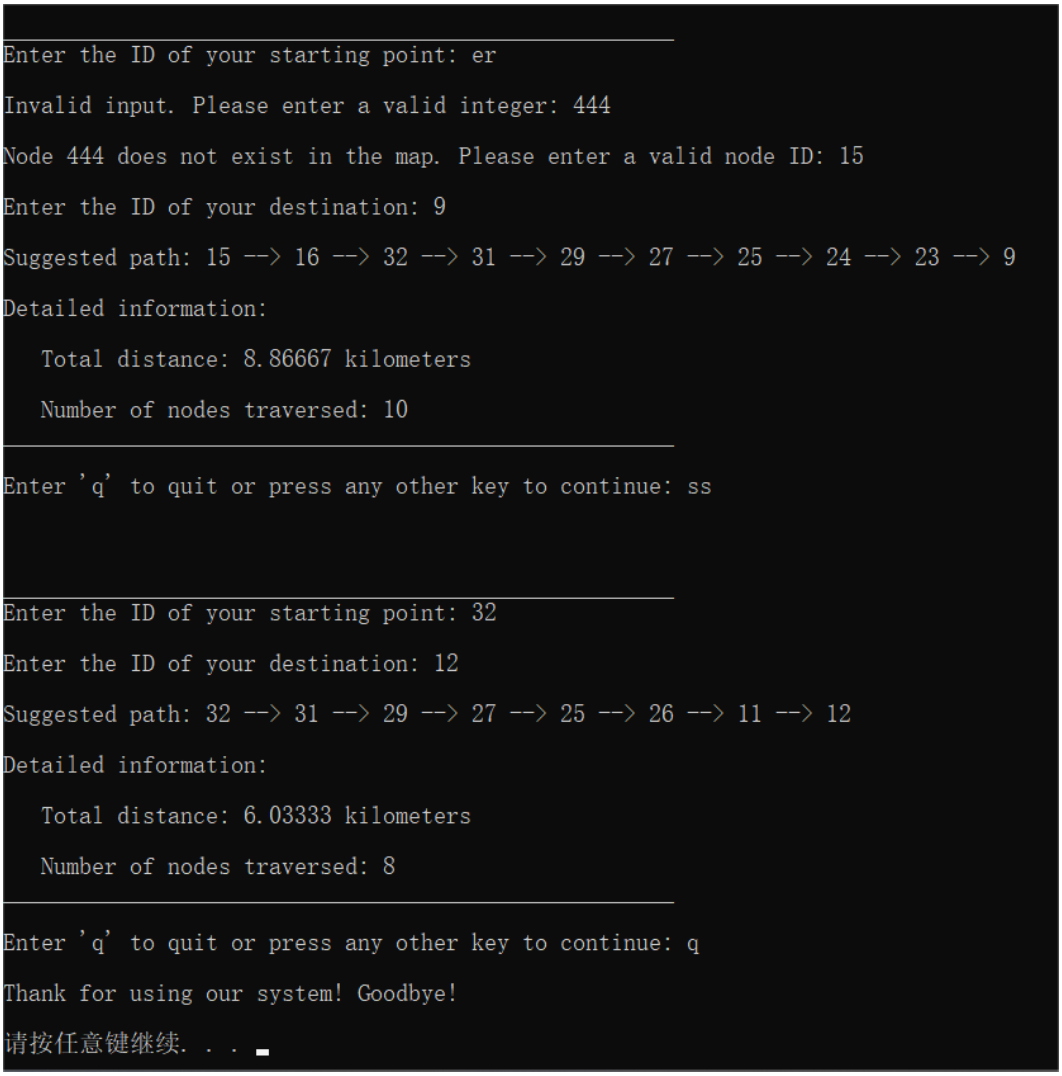

Figure 2: User interface

4 Conclusion

In this project, we have implemented a C++ based navigation system with Dijkstra's algorithm. The current system only focuses on the navigation system in a small area (e.g., Longgang District). The system's search/route-planning efficiency can be further improved by caching (e.g., save the frequently queried (origin, destination) pair). To scale up the system (e.g., Shenzhen city, Guangdong Province), we should leverage a heuristic function that prioritizes searching on the paths that are more likely to be the optimal path (i.e., A* algorithm). Another potential area for improvement is to incorporate other factors such as terrain and real-time traffic/weather information into the existing system.