M.KUMARASAMY
COLLEGE OF ENGINEERING
NAAC Accredited Autonomous Institution
Approved by AICTE & Affiliated to Anna University
ISO 9001:2015 Certified Institution
Thalavapalayam, Karur – 639 113.

A Project Report

on

SHORTEST PATH FINDING IN MAP

Submitted in partial fulfilment of requirements for the award of the course

of

CGA1121 – DATA STRUCTURES

Under the guidance of

Mrs. K. MAKANYADEVI M.E.,

Assistant Professor/CSE

Submitted By

SASMITHA. S **(927622BCS103)**

DEPARTMENT OF FRESHMAN ENGINEERING

M.KUMARASAMY COLLEGE OF ENGINEERING
(Autonomous)

KARUR – 639 113

MAY 2024

M.KUMARASAMY
COLLEGE OF ENGINEERING
NAAC Accredited Autonomous Institution
Approved by AICTE & Affiliated to Anna University
ISO 9001:2015 Certified Institution
Thalavapalayam, Karur – 639 113.

M. KUMARASAMY COLLEGE OF ENGINEERING

(Autonomous Institution affiliated to Anna University, Chennai)

KARUR – 639 113

BONAFIDE CERTIFICATE

Certified that this project report on **"SHORTEST PATH FINDING IN MAP"** is the bonafide work of **SASMITHA S(927623BCS103)** who carried out the project work during the academic year 2023- 2024 under my supervision.

Signature

Mrs. P. KAYALVIZHI M.E.,

SUPERVISOR,

Department of Computer Science and Engineering,

M. Kumarasamy College of Engineering,

Thalavapalayam, Karur -639 113.

Signature

Dr. K.CHITIRAKALA, M.Sc., M.Phil.,Ph.D.,

HEAD OF THE DEPARTMENT,

Department of Freshman Engineering,

M. Kumarasamy College of Engineering,

Thalavapalayam, Karur -639 113.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION OF THE INSTITUTION

To emerge as a leader among the top institutions in the field of technical education

MISSION OF THE INSTITUTION

- Produce smart technocrats with empirical knowledge who can surmount the global challenges
- Create a diverse, fully-engaged, learner-centric campus environment to provide quality education to the students
- Maintain mutually beneficial partnerships with our alumni, industry, and Professional associations

VISION OF THE DEPARTMENT

To achieve education and research excellence in Computer Science and Engineering

MISSION OF THE DEPARTMENT

- To excel in academic through effective teaching learning techniques
- To promote research in the area of computer science and engineering with the focus on innovation
- To transform students into technically competent professionals with societal and ethical responsibilities

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO 1: Graduates will have successful career in software industries and R&D divisions through continuous learning.

PEO 2: Graduates will provide effective solutions for real world problems in the key domain of computer science and engineering and engage in lifelong learning.

PEO 3: Graduates will excel in their profession by being ethically and socially responsible.

PROGRAM OUTCOMES (POs)

Engineering students will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and recei inst

11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

1. **PSO1: Professional Skills:** Ability to apply the knowledge of computing techniques to design and develop computerized solutions for the problems.
2. **PSO2: Successful career:** Ability to utilize the computing skills and ethical values in creating a successful career.

ABSTRACT

This project aims to implement the shortest path finding algorithm in maps using the C programming language. The project focuses on developing efficient algorithms to find the shortest path between two locations on a map represented as a graph. Utilizing concepts from graph theory and data structures such as graphs, queues, and arrays, the program will enable users to input a map, specify start and end points, and compute the shortest path between them . The project aims to provide a practical tool for route planning and navigation systems.

ABSTRACT WITH POs AND PSOs MAPPING

ABSTRACT	POs MAPPED	PSOs MAPPED
This project aims to implement the shortest path finding algorithm in maps using the C programming language. The project focuses on developing efficient algorithms to find the shortest path between two locations on a map represented as a graph. Utilizing concepts from graph theory and data structures such as graphs, queues, and arrays, the program will enable users to input a map, specify start and end points, and compute the shortest path between them . The project aims to provide a practical tool for route planning and navigation systems.	**PO1(2)** **PO2(3)** **PO3(2)** **PO4(2)** **PO5(3)** **PO6(1)** **PO7(3)** **PO8(2)** **PO9(3)** **PO10(3)** **PO11(2)** **PO12(2)**	**PSO1(3)** **PSO2(2)**

Note: 1- Low, 2-Medium, 3- High

SUPERVISOR **HEAD OF THE DEPARTMENT**

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Introduction

The project addresses the pressing need for efficient route finding in maps, crucial for various applications such as GPS navigation, logistics, and urban planning. By leveraging advanced data structures and algorithms, the project aims to provide a robust solution for optimizing travel routes and minimizing traversal time. The locations are represented as nodes(vertices) and the connection between them as edges .This allows for application of graph algorithms to solve problems related to the map such as finding the shortest path between two locations .Tools planning routes and navigating from one location to another efficiently , utilizing the implemented algorithm for finding the shortest path on a map.

1.2 Objective

The objective of this project is to design and implement an efficient system for route optimization using graph-based data structures. This system will:

1. Provide fast and accurate route calculations for various applications.
2. Minimize travel time and distance by finding the shortest path between locations.
3. Handle a large number of locations and connections, ensuring scalability.
4. Integrate seamlessly with existing navigation and logistics systems.
5. Improve overall user experience by delivering reliable and timely route information.

1.3 Data Structure Choice

Given the requirements and objectives of the project, the choice of data structure is critical. For representing the map and solving the shortest path problem, the following data structure is chosen:

Graph Data Structure

Graphs are well-suited for representing maps, where locations are nodes and connections are edges. This choice is driven by several factors:

6. **Flexibility:** Graphs can model complex relationships between locations.
7. **Efficiency**: Graph algorithms, such as Dijkstra's or A* algorithm, are efficient for finding shortest paths.
8. **Scalability**: Graphs can handle a large number of nodes and edges, which is essential for real-world applications.
9. **Compatibility**: Graph data structures are compatible with various algorithms and can be easily integrated with existing systems.

The chosen graph data structure will be implemented using an adjacency list, which provides efficient storage and fast access to neighbors of a node. This structure allows the project to meet its performance and scalability goals, ensuring optimal route finding and navigation.

By utilizing this data structure, the project aims to deliver a robust and efficient solution for route optimization, addressing the needs of modern navigation and logistics applications.

CHAPTER 2

PROJECT METHODOLOGY

2.1 1 The project methodology for developing an efficient shortest path finding system in maps using the C programming language incorporates several key phases and features to ensure a systematic and successful development process.

Structured Phases: The methodology encompasses well-defined phases, including Requirements Analysis, System Design, Implementation, Testing, Deployment, and Evaluation/Maintenance. Each phase progresses in a structured manner to facilitate clear communication, timely issue identification, and goal achievement.

Requirement Analysis: The project begins with a comprehensive analysis of requirements, involving stakeholders to gather detailed information, define project scope, and identify constraints and dependencies. This phase culminates in the creation of a requirements specification document and use case diagrams, ensuring alignment with user needs and project objectives.

System Design: Following requirements analysis, the System Design phase focuses on developing a detailed system architecture, including the design of the graph data structure to represent nodes and edges. Appropriate technologies and tools are selected, and integration plans for graph algorithms like Dijkstra's and A* are outlined. Deliverables include system architecture diagrams, data structure design documents, and algorithm design documents.

Implementation: In the Implementation phase, the development environment is set up, and the system is coded based on the design specifications. This involves implementing the graph data structure using C, developing shortest path finding algorithms, and creating tools for route planning and navigation. Key structures for nodes and edges are defined, and functions are implemented for managing the graph and running the shortest path algorithms.

Testing: Rigorous testing is conducted in the Testing phase to ensure the functionality, reliability, and performance of the system. A comprehensive testing plan, including unit tests, integration tests, and system tests, is executed, and results are documented. Bugs and performance issues are identified and resolved to meet required standards.

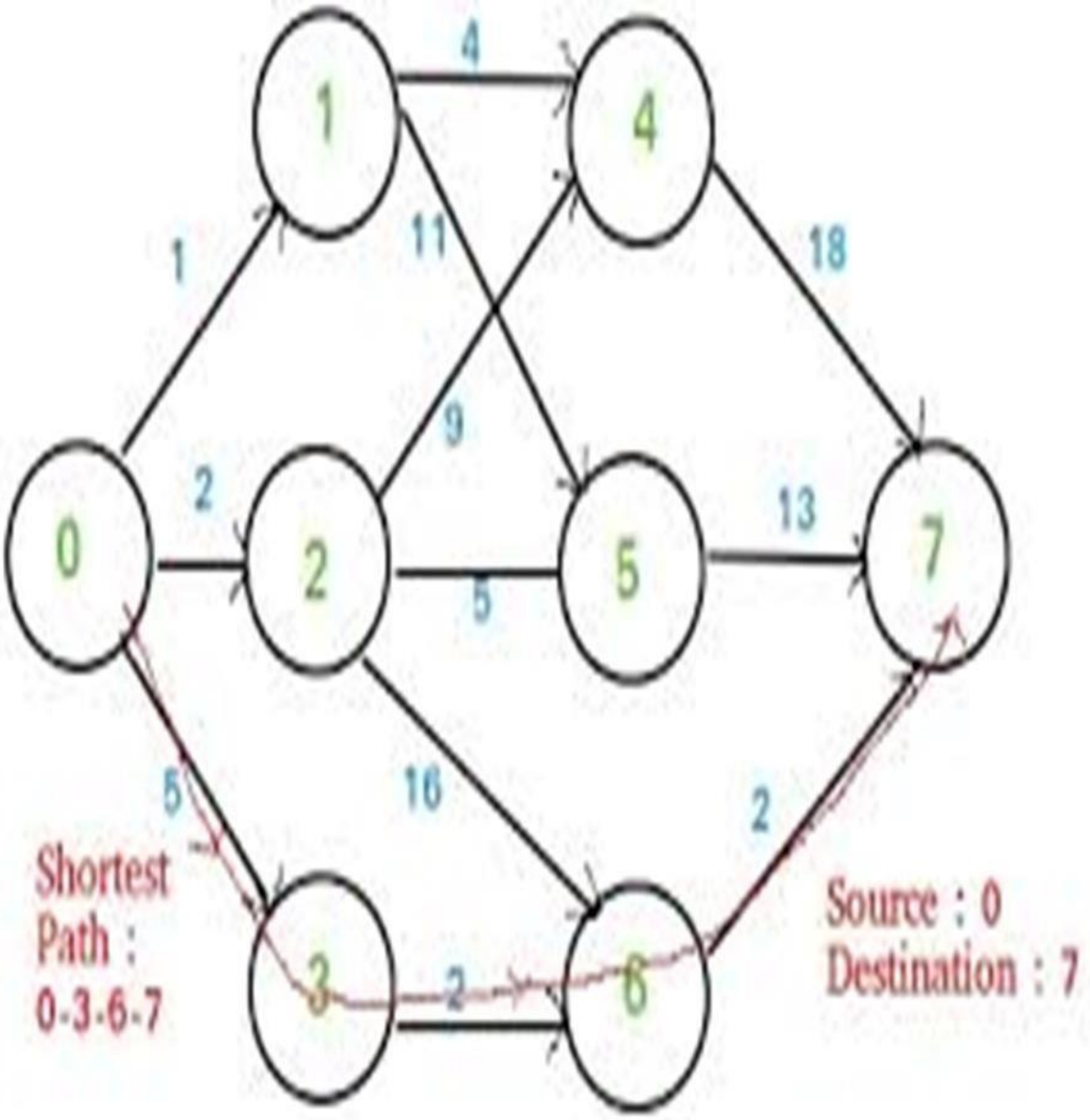
Deployment: The Deployment phase involves preparing the deployment environment, configuring the system, and performing final testing to ensure seamless

operation in the production environment. The deployed system is thoroughly documented to facilitate future maintenance and enhancements.

Evaluation and Maintenance: Following deployment, the Evaluation and Maintenance phase focuses on monitoring system performance, collecting user feedback, and conducting regular maintenance. Plans for future scalability and enhancements are made to ensure the system continues to meet evolving user needs and performs optimally.

By adhering to this structured methodology, the project aims to deliver a robust and efficient shortest path finding system in maps using C, addressing the requirements of modern navigation and logistics applications. Clear communication, systematic progression, and adherence to project objectives are emphasized throughout the project lifecycle.

2.2 Block Diagram



CHAPTER 3
MODULES

3.1 Input Module:

The input module in the provided C program serves as the gateway for user-provided data. It initiates a dialogue with the user, requesting crucial information such as the number of edges in the graph and details of each edge, including the start node, end node, and weight. Additionally, it prompts the user to specify the start and end nodes for the shortest path computation. Through a series of input prompts and scanf functions, this module ensures the program receives essential data in a structured manner, preparing it for further processing in subsequent modules.

3.2 Graph Representation Module:

At the heart of the program lies the graph representation module, responsible for converting user-provided input data into a tangible graph structure. It achieves this by utilizing functions like `add edge` and `get node index` to construct a representation where nodes are characterized by characters and their connected edges stored within arrays nested in a `Node` struct. This module's functionality is pivotal in establishing the foundation for subsequent path finding operations, enabling efficient traversal and analysis of the graph.

3.3 Shortest Path Algorithm Module:

The shortest path algorithm module, encapsulated within the `dijkstra` function, forms the computational core of the program. It implements Dijkstra's algorithm, a renowned method for finding the shortest path between nodes in a graph. By iteratively exploring nodes and updating their shortest path distances based on edge weights, this module identifies the optimal route from the specified start node to the target end node. Through meticulous

tracking of distances and previous nodes, it facilitates the reconstruction of the shortest path, ensuring accurate and efficient path finding capabilities.

3.4 Output Module:

While embedded within the `dijkstra` function, the output module plays a crucial role in presenting the computed shortest path to the user. Following the execution of Dijkstra's algorithm, this module formats and displays the resultant path on the console. It meticulously showcases the nodes traversed along the shortest path and provides vital information such as the total distance covered. Through clear and concise output statements, this module offers users a comprehensive understanding of the optimal route identified by the algorithm, enhancing the program's usability and transparency.

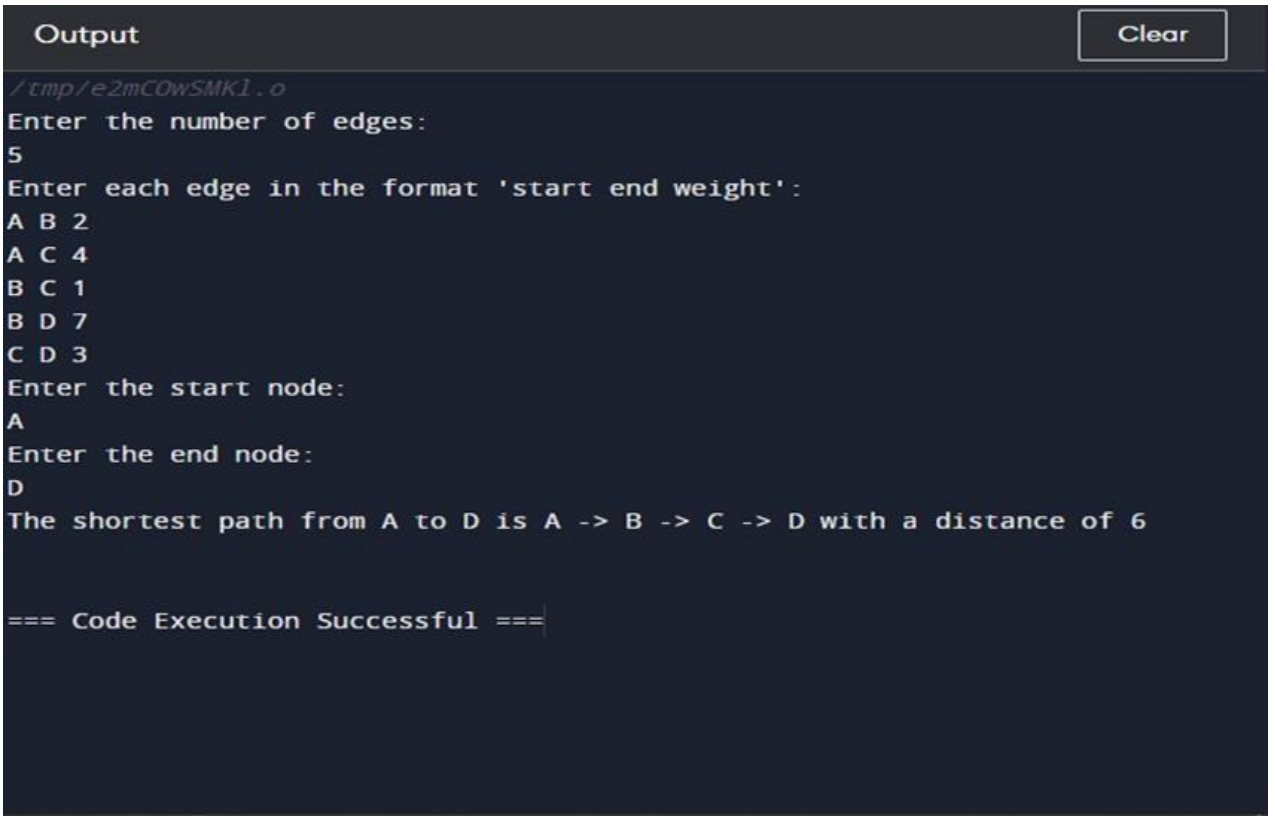
3.5 User Interface Module:

Although not distinctly delineated, the program's command-line interface serves as its user interface module. This module facilitates interaction between the program and the user, guiding them through the process of inputting graph data and specifying the start and end nodes for path finding. Through a sequence of prompts and textual feedback, it ensures a seamless user experience, enabling users to provide input and receive output in a structured manner. While the current interface operates in a command-line environment, potential enhancements could involve implementing a more intuitive graphical or interactive interface for heightened user engagement and accessibility.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Results



4.2 Discussion

The provided C program embodies fundamental concepts in graph theory and algorithmic problem-solving. It tackles the classic shortest path problem, a fundamental challenge in computer science and network optimization. By employing Dijkstra's algorithm, the program efficiently navigates through the graph, determining the optimal path from a designated start node to a target end node. This algorithm's effectiveness lies in its ability to greedily select the nearest unvisited node at each iteration, progressively refining distance estimates until the shortest path is identified. Through careful implementation and manipulation of graph representations, the program demonstrates robustness and scalability, capable of handling diverse graph structures and input scenarios

CHAPTER 5
CONCLUSION

In conclusion, the provided C program showcases the practical application of graph theory and algorithmic techniques in solving real-world problems. By seamlessly integrating input processing, graph representation, and shortest path computation, the program offers a comprehensive solution for path finding tasks. Through Dijkstra ' s algorithm, it efficiently identifies the shortest path in a graph, providing valuable insights into network routing, transportation planning, and resource allocation. While the program's current implementation excels in command-line environments, future enhancements could focus on enhancing user interaction and extending functionality to accommodate more complex graph algorithms and visualization capabilities.

REFERENCES

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms (3rd ed.)*. The MIT Press.
2. Sedgewick, R., & Wayne, K. (2011). *Algorithms (4th ed.)*. Addison-Wesley.
3. Harel, D., & Tarjan, R. E. (1984). Fast Algorithms for Finding Nearest Common Ancestors. *SIAM Journal on Computing, 13*(2), 338–355. https://doi.org/10.1137/021302.

APPENDIX

```
#include  <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>

#define MAX_NODES 100
#define INF INT_MAX

typedef struct {
    int weight;
    int end;
} Edge;

typedef struct {
    Edge edges[MAX_NODES];
    int edge_count;
} Node;

typedef struct {
    int distance;
    int previous;
} Path;

Node graph[MAX_NODES];
```

```
Path paths[MAX_NODES];
int  visited[MAX_NODES];
int node_count = 0;
char nodes[MAX_NODES];

int get_node_index(char node) {
    for (int i = 0; i < node_count; i++) {
        if (nodes[i] == node) {
            return i;
        }
    }
    nodes[node_count] = node;
    return node_count++;
}

void add_edge(char u, char v, int weight) {
    int u_index = get_node_index(u);
    int v_index = get_node_index(v);
    graph[u_index].edges[graph[u_index].edge_count].end = v_index;
    graph[u_index].edges[graph[u_index].edge_count].weight = weight;
    graph[u_index].edge_count++;
    graph[v_index].edges[graph[v_index].edge_count].end = u_index;
    graph[v_index].edges[graph[v_index].edge_count].weight = weight;
    graph[v_index].edge_count++;
}

void dijkstra(int start_index, int end_index) {
    for (int i = 0; i < node_count; i++) {
        paths[i].distance = INF;
```

```
    paths[i].previous = -1;
    visited[i] = 0;
}
paths[start_index].distance = 0;

for (int i = 0; i < node_count; i++) {
    int min_distance = INF;
    int u = -1;
    for (int j = 0; j < node_count; j++) {
        if (!visited[j] && paths[j].distance < min_distance) {
            min_distance = paths[j].distance;
            u = j;
        }
    }
    if (u == -1) break;
    visited[u] = 1;

    for (int j = 0; j < graph[u].edge_count; j++) {
        int v = graph[u].edges[j].end;
        int weight = graph[u].edges[j].weight;
        if (!visited[v] && paths[u].distance + weight < paths[v].distance) {
            paths[v].distance = paths[u].distance + weight;
            paths[v].previous = u;
        }
    }
}

if (paths[end_index].distance == INF) {
    printf("No path found\n");
```

```
        return;
    }

    printf("The shortest path from %c to %c is ", nodes[start_index],
nodes[end_index]);

    int path[MAX_NODES];
    int count = 0;
    for (int v = end_index; v != -1; v = paths[v].previous) {
        path[count++] = v;
    }
    for (int i = count - 1; i >= 0; i--) {
        printf("%c", nodes[path[i]]);
        if (i > 0) {
            printf(" -> ");
        }
    }
    printf(" with a distance of %d\n", paths[end_index].distance);
}

int main() {
    int edges_count;
    printf("Enter the number of edges:\n");
    scanf("%d", &edges_count);

    printf("Enter each edge in the format 'start end weight':\n");
    for (int i = 0; i < edges_count; i++) {
        char u, v;
        int weight;
```

```
        scanf(" %c %c %d", &u, &v, &weight);
        add_edge(u, v, weight);
    }

    char start, end;
    printf("Enter the start node:\n");
    scanf(" %c", &start);
    printf("Enter the end node:\n");
    scanf(" %c", &end);

    dijkstra(get_node_index(start), get_node_index(end));

    return 0;
}
```