SYMBIOSIS INSTITUTE OF TECHNOLOGY,HYDERABAD
A Constituent of Symbiosis International (Deemed University), Pune.
Re-accredited by NAAC with 'A++ Grade |Awarded Category - I by UGC

Ambulance Routing System

Bachelor of Technology

in

Computer Science And Engineering

Submitted by

Mahesh Jella (PRN:24070721017)

Venkatesh K (PRN:24070721014)

For the Fulfillment

of

Data Structure Algorithm Lab Course

Symbiosis Institute of Technology, Hyderabad

Survey Number:292, Off Bangalore Highway, Modallaguda Village,Nandigama Mandal, Ranga Reddy DIST, Near Hyderabad, Telangana- 509217.

Contents

Abstract

In emergency scenarios, ambulances must reach hospitals as quickly as possible, but road congestion often causes critical delays.
The **Smart Ambulance Route Finder** is a C-based system that determines the most efficient route between two places in a city using **Dijkstra's shortest path algorithm**.
The system represents the city as a weighted graph, where intersections are nodes and roads are edges with travel times as weights.
It dynamically updates the road weights to simulate **real-time traffic**, ensuring that the ambulance always follows the fastest route.
The program also includes a **console-based animation** that visually simulates the ambulance's movement step by step.
This project demonstrates the application of **graph theory and data structures** in developing real-world intelligent systems that can optimize emergency response time.

Introduction

Ambulances play a vital role in providing emergency medical care, where every second can mean the difference between life and death.
In urban environments, **traffic congestion** and poor route selection often delay ambulances.
This project addresses the challenge by developing a **route optimization system** that finds the **shortest and fastest path**from the ambulance's current location to the hospital.
The project leverages **Dijkstra's algorithm**, a well-known graph-based approach for computing the minimum distance between nodes in a weighted network.

In this implementation, locations such as *City Center, Tech Park, Airport, Mall,* and *Main Hospital* are treated as vertices, while roads between them are represented by edges with specific travel times.
The system also introduces a **dynamic traffic simulation**, where travel times on roads fluctuate periodically, mimicking real-world conditions.

By recomputing the optimal path after every traffic update, the system ensures that the ambulance always takes the most efficient route available at that moment.

The project is implemented entirely in **C**, demonstrating the use of arrays, structures, loops, and graph algorithms in solving real-time navigation problems.

Methodology

The system's workflow is divided into distinct stages:

3.1 Problem Definition

The main objective is to compute the most efficient route between an ambulance's starting point and a hospital, adapting to real-time changes in road conditions.
The problem is modeled as a **graph**, where:

- **Nodes (vertices)** represent key locations (e.g., Hospital, City Center, Airport).
- **Edges** represent roads between locations with associated **weights** (travel time in minutes).

3.2 Algorithm Used — Dijkstra's Algorithm

Dijkstra's algorithm is chosen because it efficiently determines the shortest path in a **weighted, non-negative graph**.
It uses the concept of a **priority-based selection**, updating distances to neighboring nodes iteratively until the shortest path to all vertices is known.

Time complexity (matrix-based implementation): **$O(V^2)$**
where V = number of locations (vertices).

3.3 Dynamic Traffic Simulation

To mimic real-world conditions, the travel times between locations are periodically adjusted using random values:

```
int change = (rand() % 7) - 3; // -3 to +3 variation
```

This increases or decreases the travel time between connected roads, simulating congestion or clearance.

3.4 Data Representation

The city map is stored in an **adjacency matrix**:

```
int city[V][V] = {
  {0, 10, 0, 0, 15, 0},
  {10, 0, 12, 0, 0, 15},
  {0, 12, 0, 22, 0, 1},
  {0, 0, 22, 0, 2, 0},
  {15, 0, 0, 2, 0, 5},
  {0, 15, 1, 0, 5, 0}
};
```

Each row and column corresponds to a location.
A `0` indicates no direct road between the two places.

3.5 Animation System

After the shortest path is computed, the program animates the ambulance's journey:

- It prints each location with a delay, showing progress step-by-step.
- Between updates, traffic changes are simulated, and the path is recalculated.

This continuous loop reflects a **live simulation** where traffic patterns evolve and the algorithm adapts accordingly.

4. Code and Code explanation

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <limits.h>
#include <time.h>
#include <ctype.h>

#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif

#define V 6   // Number of locations

// List of all places
char places[V][30] = {
    "City Center",
    "Main Hospital",
    "Tech Park",
    "Airport",
    "Mall",
    "Residential Area"
```

```
};

// Function to remove newline from fgets input
void trimNewline(char *str) {
    size_t len = strlen(str);
    if (len > 0 && str[len - 1] == '\n')
        str[len - 1] = '\0';
}

// Convert “centre” → “center” and ignore spaces/case
int findPlaceIndex(char name[]) {
    char cleaned[30], ref[30];
    int i, j;

    for (i = 0, j = 0; name[i] != '\0'; i++) {
        if (name[i] != ' ' && name[i] != '\n' && name[i] != '\r')
            cleaned[j++] = tolower(name[i]);
    }
    cleaned[j] = '\0';

    // Convert "centre" to "center"
    char *ptr = strstr(cleaned, "centre");
    if (ptr) strcpy(ptr, "center");
```

```
    for (int k = 0; k < V; k++) {
        for (i = 0, j = 0; places[k][i] != '\0'; i++) {
            if (places[k][i] != ' ')
                ref[j++] = tolower(places[k][i]);
        }
        ref[j] = '\0';

        if (strcmp(cleaned, ref) == 0)
            return k;
    }

    return -1;
}

// Find vertex with minimum distance
int minDistance(int dist[], bool visited[]) {
    int min = INT_MAX, min_index = -1;
    for (int v = 0; v < V; v++)
        if (!visited[v] && dist[v] <= min)
            min = dist[v], min_index = v;
    return min_index;
}

// Recursively store path
```

```
void buildPath(int parent[], int j, int path[], int *index) {
    if (parent[j] == -1)
        return;
    buildPath(parent, parent[j], path, index);
    path[(*index)++] = j;
}

// Dijkstra's algorithm
int dijkstra(int graph[V][V], int src, int dest, int path[]) {
    int dist[V], parent[V];
    bool visited[V];
    for (int i = 0; i < V; i++) {
        dist[i] = INT_MAX;
        visited[i] = false;
        parent[i] = -1;
    }
    dist[src] = 0;

    for (int count = 0; count < V - 1; count++) {
        int u = minDistance(dist, visited);
        if (u == -1) break;
        visited[u] = true;

        for (int v = 0; v < V; v++) {
```

```
            if (!visited[v] && graph[u][v] && dist[u] != INT_MAX &&
                dist[u] + graph[u][v] < dist[v]) {
                parent[v] = u;
                dist[v] = dist[u] + graph[u][v];
            }
        }
    }

    int idx = 0;
    path[idx++] = src;
    buildPath(parent, dest, path, &idx);
    return idx;
}

// Randomly adjust road times (-3 to +3)
void updateTraffic(int city[V][V]) {
    for (int i = 0; i < V; i++) {
        for (int j = i + 1; j < V; j++) {
            if (city[i][j] > 0) {
                int change = (rand() % 7) - 3;
                int newTime = city[i][j] + change;
                if (newTime < 2) newTime = 2;
                city[i][j] = city[j][i] = newTime;
            }
```

```
        }
    }
}

// Display current traffic table
void displayTraffic(int city[V][V]) {
    printf("\n🚦 Current Traffic (minutes between locations):\n\n");
    printf("%-20s", "");
    for (int i = 0; i < V; i++) printf("%-20s", places[i]);
    printf("\n");

    for (int i = 0; i < V; i++) {
        printf("%-20s", places[i]);
        for (int j = 0; j < V; j++) {
            if (city[i][j] == 0)
                printf("%-20s", "-");
            else {
                char buf[10];
                sprintf(buf, "%d", city[i][j]);
                printf("%-20s", buf);
            }
        }
        printf("\n");
    }
```

```
}

// Animate ambulance movement
void animateAmbulance(int path[], int len) {
    printf("\n🚑 Ambulance in Motion:\n\n");
    for (int i = 0; i < len; i++) {
        printf("🟢 Currently at: %s\n", places[path[i]]);
        if (i < len - 1)
            printf("➡️  Heading towards: %s...\n", places[path[i + 1]]);
        else
            printf("🏥 Reached: %s (Destination)\n", places[path[i]]);
#ifdef _WIN32
        Sleep(1000);
#else
        sleep(1);
#endif
    }
}

int main() {
    srand(time(0));

    int city[V][V] = {
        {0, 10, 0, 0, 15, 0},
```

```
    {10, 0, 12, 0, 0, 15},
    {0, 12, 0, 22, 0, 1},
    {0, 0, 22, 0, 2, 0},
    {15, 0, 0, 2, 0, 5},
    {0, 15, 1, 0, 5, 0}
};

printf("🚨 SMART AMBULANCE ROUTE FINDER 🚨\n");
printf("---------------------------------\n");
printf("Available locations:\n");
for (int i = 0; i < V; i++)
    printf(" - %s\n", places[i]);

char srcName[30], destName[30];
printf("\nEnter ambulance starting place: ");
fgets(srcName, sizeof(srcName), stdin);
trimNewline(srcName);

printf("Enter hospital (destination): ");
fgets(destName, sizeof(destName), stdin);
trimNewline(destName);

int src = findPlaceIndex(srcName);
int dest = findPlaceIndex(destName);
```

```
if (src == -1 || dest == -1) {
    printf("\n❌ Invalid location entered. Please enter correct place names.\n");
    return 0;
}

printf("\n🌐 Simulation started... (Press Ctrl+C to stop)\n");

while (1) {
    displayTraffic(city);

    int path[V];
    int pathLen = dijkstra(city, src, dest, path);

    printf("\n🚑 Shortest Route from %s to %s:\n", places[src], places[dest]);
    for (int i = 0; i < pathLen; i++) {
        printf("%s", places[path[i]]);
        if (i != pathLen - 1) printf(" -> ");
    }
    printf("\n");

    animateAmbulance(path, pathLen);

    printf("\n🔄 Updating traffic in 5 seconds...\n");
```

```
#ifdef _WIN32
        Sleep(5000);
#else
        sleep(5);
#endif
        updateTraffic(city);
        printf("\n-----------------------------------------\n");
    }

    return 0;
}
```

4. Code Explanation

The program is modular and divided into specific functional components:

4.1 Header Files and Constants

The program uses:

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

```
#include <ctype.h>
```

`#define V 6` defines the total number of locations (vertices).

4.2 Data Representation

- The `city[V][V]` adjacency matrix stores travel times.
- The `places[V][30]` array stores location names:

```
char places[V][30] = {
  "City Center",
  "Main Hospital",
  "Tech Park",
  "Airport",
  "Mall",
  "Residential Area"
};
```

4.3 Input Handling

The program accepts location names (case-insensitive).
It also includes logic to recognize “centre” as “center”:

```
if (strstr(cleaned, "centre"))
    strcpy(cleaned + strlen(cleaned) - 6, "center");
```

4.4 Dijkstra’s Algorithm Implementation

This function finds the shortest path using iterative relaxation of distances.
It maintains:

- `dist[]` → shortest known distance to each node.
- `visited[]` → whether a node is finalized.
- `parent[]` → stores the route to reconstruct the path.

Core logic:

```
for (int v = 0; v < V; v++) {
  if (!visited[v] && graph[u][v] && dist[u] + graph[u][v] < dist[v]) {
    parent[v] = u;
    dist[v] = dist[u] + graph[u][v];
  }
}
```

4.5 Traffic Update Function

```
void updateTraffic(int city[V][V]) {
  for (int i = 0; i < V; i++) {
    for (int j = i + 1; j < V; j++) {
      if (city[i][j] > 0) {
        int change = (rand() % 7) - 3;
        int newTime = city[i][j] + change;
        if (newTime < 2) newTime = 2;
        city[i][j] = city[j][i] = newTime;
```

```
        }
    }
  }
}
```

This ensures that travel times change continuously to reflect live road conditions.

4.6 Animation Function

```
void animateAmbulance(int path[], int pathLen) {
  for (int i = 0; i < pathLen; i++) {
    printf("🟢 Currently at: %s\n", places[path[i]]);
    if (i < pathLen - 1)
      printf("➡️  Heading towards: %s...\n",
places[path[i + 1]]);
    else
      printf("🏥 Reached: %s (Destination)\n",
places[path[i]]);
    sleep(1);
  }
}
```

This visually simulates the ambulance moving along its route in real time.

4.7 Main Function Flow

- Display all available locations.

- Accept user input for source and destination.
- Compute and display the best route.
- Animate the ambulance movement.
- Periodically update traffic and recompute the route.

5. Output

🚨 SMART AMBULANCE ROUTE FINDER 🚨

Available locations:

- City Center
- Main Hospital
- Tech Park
- Airport
- Mall
- Residential Area

Enter ambulance starting place: City Centre

Enter hospital (destination): Airport

🌐 Simulation started... (Press Ctrl+C to stop)

🚦 Current Traffic (minutes between locations):

City Center Main Hospital Tech Park Airport Mall Residential Area

City Center	-	10	-	-	15	-
Main Hospital	10	-	12	-	-	15
Tech Park	-	12	-	22	-	1
Airport	-	-	22	-	2	-
Mall	15	-	-	2	-	5
Residential Area	-	15	1	-	5	-

🚑 Shortest Route from City Center to Airport:

City Center -> Main Hospital -> Residential Area -> Tech Park -> Airport

🚑 Ambulance in Motion:

🟢 Currently at: City Center

➡️ Heading towards: Main Hospital...

🟢 Currently at: Main Hospital

➡️ Heading towards: Residential Area...

🟢 Currently at: Residential Area

➡️ Heading towards: Tech Park...

🟢 Currently at: Tech Park

➡️ Heading towards: Airport...

🏥 Reached: Airport (Destination)

🔄 Updating traffic in 5 seconds...

🚦 Current Traffic (minutes between locations):

	City Center	Main Hospital	Tech Park	Airport	Mall	Residential Area
City Center	-	12	-	-	14	-
Main Hospital	12	-	13	-	-	15
Tech Park	-	13	-	24	-	3
Airport	-	-	24	-	3	-
Mall	14	-	-	3	-	4
Residential Area	-	15	3	-	4	-

🚑 Shortest Route from City Center to Airport:

City Center -> Mall -> Airport

🚑 Ambulance in Motion:

🟢 Currently at: City Center

➡️ Heading towards: Mall...

🟢 Currently at: Mall

➡️ Heading towards: Airport...

🏥 Reached: Airport (Destination)

🔄 Updating traffic in 5 seconds...

6. Conclusion

The Smart Ambulance Route Finder project successfully showcases the power of Dijkstra's algorithm in optimizing emergency navigation.
By representing the city as a graph and continuously updating edge weights to simulate live traffic, the system provides real-time route adjustments.
This ensures that the ambulance always follows the most time-efficient path.
The project not only demonstrates strong application of data structures (graphs, arrays) and algorithms, but also creativity in simulating dynamic environments.
It stands as a practical example of how computational logic can contribute to life-saving solutions.

7. References

[1] Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language*, 2nd Edition, Prentice Hall, 1988.

[2] E. Horowitz, S. Sahni, and S. Anderson-Freed, *Fundamentals of Data Structures in C*, Universities Press, 2008.

[3] GeeksforGeeks, "Dijkstra's Shortest Path Algorithm in C." https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-in-c/

[4] TutorialsPoint, "C Programming Structures and Pointers." https://www.tutorialspoint.com/cprogramming/

[5] Lecture Notes, *Data Structures and Algorithms*, Symbiosis Institute of Technology, Hyderabad, 2025.