AMBULANCE PATH FINDER

MAJOR PROJECT

Semester 1

Four Year Undergraduate Program- Design Your Degree

Submitted to:

University of Jammu, Jammu

By:

HEMACHANDRA GROUP

Name	**Roll no.**
Arti Devi	DYD-24-06
Fatima Sayed	DYD-24-10
Nishkarsh Singh Nagra	DYD-24-25
Rudraksh Sharma	DYD-24-29
Shailja Sharma	DYD-24-34

UNDER THE MENTORSHIP OF

Prof. K.S. Charak	Dr. Jatinder Manhas	Dr. Sunil Kumar Bougal
(Dept. of Mathematics)	(Dept. of CA & IT)	(Dept. of Statistics)

CERTIFICATE

The work embodied in this report entitled "Ambulance Path Finder" has been done by Arti Devi, Fatima Sayed, Rudraksh Sharma, Nishkarsh Singh Nagra and Shailja Sharma as a Major Project for Semester I. This work was carried out under the guidance of Prof. K.S. Charak, Dr. Jatinder Manhas and Prof. Sunil Kumar Bougal for the partial fulfilment for the award of the Design Your Degree, Four Year Undergraduate Program, University of Jammu. This project report has not been submitted anywhere for the award of any degree, diploma.

Signature of students:

1. Arti Devi
2. Fatima Sayed
3. Nishkarsh Singh Nagra
4. Rudraksh Sharma
5. Shailja Sharma

Signature of mentors:

Prof. K.S. Charak **Dr. Jatindar Manhas** **Dr. Sunil Kumar Bougal**

Prof. Alka Sharma

SIIEDC, University of Jammu, Jammu

ACKNOWLEDGEMENT

Expressing profound appreciation, we extend our heartfelt gratitude to all who have provided guidance and support during our endeavours. First and foremost, we acknowledge the Almighty for instilling in us the determination and spiritual strength to overcome the challenges we have encountered. Words alone cannot capture the depth of our indebtedness and the profound sense of gratitude we feel towards our mentors Prof. K.S. Charak, Department of Mathematics, Dr. Jatinder Manhas, Department of Computer Application and IT, and Dr. Sunil Kumar Bougal, Department of Statistics at the University of Jammu. Their guidance, encouragement, and well-articulated advice have served as the bedrock of our journey. Despite their demanding schedules, they have generously extended their unwavering support, providing us with invaluable encouragement and wisdom. We consider ourselves truly fortunate to have had the privilege of benefiting from their perpetual motivation. Equally, we are blessed to express our heartfelt gratitude to Prof. Alka Sharma, Director of SIIEDC at the University of Jammu, and member of the advisory committee. Her invaluable guidance, generous assistance, and provision of necessary facilities have played a pivotal role in our pursuit of excellence.

We extend our deepest thanks to all esteemed mentors whose immense support has guided us through the execution of this major project. Reflecting on our journey, we acknowledge the invaluable contributions of our friends, whose steadfast support, affection, and cooperation have brought joy and happiness to our days. Their presence has been a constant source of strength, propelling us forward with renewed determination. Lastly, we extend our gratitude to all individuals, whether mentioned explicitly or not, who have silently wished us well, offered constructive suggestions, and assisted us in achieving our goals. Their contributions, though often unseen, have made an indelible mark on our journey, for which we are sincerely grateful.

Abstract

In India, a significant number of patients lose their lives due to delayed medical assistance, primarily because ambulances do not arrive on time or treatment is delayed. With advancements in technology, it is crucial to leverage modern innovations to solve this problem. Currently in India, when an accident occurs, emergency services are contacted through a designated helpline number. Upon receiving it, the medical team is informed and an ambulance is dispatched to the location. Despite their efficiency, delays often occur due to uncertainties and it is difficult to reach on time.

To address these inefficiencies, a technologically advanced system is proposed that ensures timely ambulance transportation from the accident site to the hospital. The key feature of this system is its ability to automatically determine the shortest and fastest route between the accident location and the nearest hospital, significantly reducing travel time and improving emergency medical response.

The system utilizes **Dijkstra's algorithm**, a well-established shortest-path algorithm, to determine the most efficient route for the ambulance. The nearest hospital is informed about the incoming emergency, ensuring that medical staff are prepared in advance to provide immediate treatment upon arrival. It can lead to reduction in time with efficient time management and result in the immediate treatment of the patient.

As timely medical intervention can save countless lives, and providing technology is the most effective way to improve emergency situations by using Dijkstra's algorithm for shortest path routing. It will ensure that the ambulances reach patients quickly, increasing the chances of saving lives. With a well- structured implementation plan this solution has immense potential to revolutionize emergency healthcare in India, providing faster medical assistance in the near future.

TABLE OF CONTENT

CHAPTER-1

INTRODUCTION

In today's fast-paced world, emergency medical services (EMS) play a pivotal role in saving lives, and the ability to reach a hospital or medical facility in the shortest time possible can make the difference between life and death. One of the most critical aspects of emergency medical services is the speed at which an ambulance can reach the destination, especially when navigating through traffic, road closures, and various obstacles in a city environment. The goal of this project is to develop an efficient system to find the shortest path for an ambulance traveling between different locations in a city. The key challenge lies in determining the fastest route, considering multiple potential paths and road conditions.

The Dijkstra algorithm's main premise is basically as follows: First, create a set S of vertices, and then determine the path weights from the source point (beginning point) S to the vertices in the set. During the operation of the Dijkstra algorithm, different paths are repeatedly chosen, that is, the vertices with the best path estimation are added, and the vertex i is added to S, and the function weights all the path weights of the vertex i, and the final function weight value is chosen. The shortest path is the best path.

The system uses a computational approach to determine the shortest route, and the algorithm employed in this project is **Dijkstra's Algorithm**, a widely-used method for solving the shortest path problem in graph theory. In this case, it consists of a set of locations (represented as vertices or nodes) connected by roads (represented as weighted edges), with the weights corresponding to the distance or time required to travel along each road. The main objective of this project is to leverage Dijkstra's algorithm to help emergency responders choose the optimal route from a starting location (such as a hospital or station) to a destination, minimizing response time and ensuring the quickest delivery of medical assistance.

As ambulance service is one of the most critical services to be operated anytime, an ambulance reaching a patient at the right time and reaching a hospital at the minimum possible time is very crucial as it is the time that decides the life of a person. In this paper we propose a method of developing a predefined data about the distance to reach an ambulance service using Dijkstra's algorithm, which will aid in picking a wise nearest service.

a) The Problem of Efficient Ambulance Routing

The problem at hand is to determine the shortest route for an ambulance from one location to another in a given city. Consider a scenario where an ambulance needs to travel between two points, such as a hospital and a residential area. The ambulance driver may encounter various challenges such as road traffic, construction, or accidents along the way, which can lead to delays. By calculating the shortest possible path in advance, the ambulance can avoid these potential setbacks and take the quickest route available.

Traditional methods of route planning, such as using a paper map or relying on drivers' knowledge of the area, are insufficient for solving such time-critical problems. With the advancement of technology and computational algorithms, it is possible to automate this process and ensure that the ambulance takes the most efficient route. By employing a computer-based solution, the ambulance team can have immediate access to the best route, saving valuable time.

b) Dijkstra's Algorithm: An Overview

Dijkstra's algorithm, named after its inventor Edsger Dijkstra, is a well-established algorithm in graph theory that solves the single-source shortest path problem for a graph with non-negative edge weights. The algorithm works by incrementally building up the shortest path from the starting point to all other nodes in the graph. It does so by selecting the node with the smallest tentative distance, updating the distances to its neighbouring nodes, and repeating this process until the destination is reached.

In the context of ambulance routing, it represents the city's layout, where each location (e.g., hospital, residential area, airport, etc.) is a node, and each road between two locations is an edge with a weight corresponding to the travel time or distance. The algorithm starts at the source location (e.g., the ambulance station or the current location of the ambulance) and works its way to find the shortest route to the destination.

The key steps involved in Dijkstra's algorithm are as follows:

- **Initialization**: Initially, all nodes are marked with an infinite distance, except for the starting node, which is set to a distance of zero. The algorithm keeps track of the nodes that have been processed (those that are part of the shortest path tree) and those that still need to be explored.
- **Selection of the Minimum Distance Node**: The algorithm repeatedly selects the node with the smallest known distance that has not been processed yet. This node is then marked as processed.
- **Updating Neighbouring Nodes**: Once the minimum distance node is selected, the algorithm examines its neighbouring nodes. For each neighbour, if the total distance from the start node to the neighbour (through the current node) is shorter than the previously known distance to that neighbour, the algorithm updates the neighbour's distance.
- **Repetition**: The algorithm repeats the above steps until all nodes have been processed. The shortest distance to each node is known once the algorithm completes.
- **Completion**: The algorithm terminates when the destination node has been processed, and the shortest path from the source to the destination has been found.

This approach ensures that the ambulance will always take the quickest route, avoiding unnecessary delays and minimizing the overall travel time. Additionally, it is capable of

handling multiple routes and dynamically calculating the optimal path based on the current state of the graph.

c) Importance of finding the optimal path

The importance of discovering optimal pathways in industrial applications cannot be emphasized. It is crucial to resource efficiency, time savings, and cost savings. By shortening travel distances and durations, optimal path planning streamlines operations, conserves energy, and lowers operating costs, resulting in higher productivity. Furthermore, in safety-critical industries such as robotics and emergency response, optimal pathways enable secure and quick movements, while in supply chains, they improve inventory management and distribution efficiency. The solution of optimal pathways leads companies toward increased efficiency, decreased environmental impact, and higher overall performance, from energy networks to communication systems, precision agriculture to transportation logistics.

d) Application to Ambulance Routing

The locations where ambulances need to go (e.g., hospitals, accident sites, fire stations) are modelled as nodes, and the roads connecting them are edges with weights that represent either distance or time required to travel on that road.

Our case consists of five locations- Government Hospital Gandhi Nagar, Satwari, Airport Jammu, Bahu Plaza, and Apsara Road. Each location is represented as a node, and the roads between them are represented by weighted edges, where the weight denotes the travel time or distance.

The program prompts the user to select a starting and destination location, then applies Dijkstra's algorithm to calculate the shortest route between them. It outputs the shortest distance, allowing the ambulance driver to take the quickest path to reach the hospital or any other destination.

These locations are connected by roads of various lengths or travel times. The ambulance needs to travel from one location (say, Satwari) to another (say, Government Hospital Gandhi Nagar). By applying Dijkstra’s algorithm, the ambulance team can determine the shortest possible route based on the available road network, considering all possible paths and their corresponding travel times.

This method allows the ambulance team to receive real-time guidance on the most efficient route, ensuring that the ambulance can reach its destination as quickly as possible, regardless of road conditions or traffic patterns. This is particularly important in urban areas with complex road networks, where multiple possible routes may exist.

e) Key Benefits of Using Dijkstra's Algorithm for Ambulance Routing

- **Optimized Route Selection**: The algorithm ensures that the ambulance always takes the shortest possible route, which is crucial in emergencies.
- **Time-Saving**: By automating the route selection process, valuable time is saved that would otherwise be spent manually determining the best route.
- **Scalability**: The algorithm can be applied to large-scale cities with numerous locations, making it scalable for a wide range of cities and regions.
- **Flexibility**: The algorithm can accommodate changes in road conditions, such as traffic congestion or accidents, allowing it to dynamically adapt to changing situations.
- **Improved Response Time**: Faster response times mean better chances for positive medical outcomes, as the ambulance can reach the destination without unnecessary delays.

CHAPTER-2

LITERATURE SURVEY

Paper:- 1

R. Jadhav, S. Baperkar, S. Kamble, U. Mohite
Shortest Route Finding Ambulance System
Viva- Tech International Journal for Research and Innovation
(Volume: 1, Issue 4), (2021), (Pages: 1-6)

OVERVIEW

The research paper "Shortest Route Finding Ambulance System" focuses on developing a system to minimize ambulance response times during emergencies. It utilizes GPS technology, real-time traffic data, and route optimization algorithm like Dijkstra to identify the fastest path. The system aims to improve patient survival rates by providing efficient routes and real-time updates to both hospitals and users. Challenges like GPS accuracy and traffic data reliability are also addressed.

METHODOLOGY

The proposed ambulance system improves response times by automatically detecting accidents and dispatching the nearest ambulance. When an accident occurs, the vehicle sends its GPS coordinates to a server, which then calculates the distance to all nearby ambulances. Using Dijkstra's algorithm, the closest ambulance is selected and sent a GPS request. The system also optimizes route selection to the nearest hospital using real-time traffic data and IoT-enabled traffic signal control to clear paths for ambulances. The entire process is automated, with communication through smartphones and GPS for both the patient and the ambulance driver, ensuring faster response and timely medical care.

TECHNOLOGY

GPS (Global Positioning System), GSM (Global System for Mobile Communications), IoT (Internet of Things), Dijkstra's Algorithm, RF (Radio Frequency) Communication, Android Application, REST API, Traffic Signal Control.

MODEL

Dijkstra's Algorithm (for finding the shortest path), RF Communication Protocol (for communication between the ambulance and traffic signals), IoT-based Traffic Signal Control (for prioritizing ambulances at traffic intersections), GPS-based Location Tracking (for tracking the patient's and ambulance's location), REST API (for real-time traffic and hospital data exchange)

RESULTS

The results of the proposed ambulance system show a significant reduction in response time, with the system effectively identifying and dispatching the nearest ambulance using real-time GPS data. The integration of Dijkstra's algorithm for route optimization, combined with IoT-based traffic signal control, reduces travel time by up to 20%. This leads to faster ambulance arrival at accident sites, improving the chances of patient survival. Additionally, the automated communication between the patient, ambulance, and hospital ensures timely and efficient emergency responses.

DATA SET DESCRIPTION

TEXT BASED

FUTURE SCOPE

The future scope of the proposed ambulance system includes integrating machine learning to predict traffic patterns and optimize routes even further, incorporating AI-based health monitoring to assess the patient's condition during transit, and expanding the system to cover larger geographic areas with real-time data sharing between multiple cities. Additionally, incorporating drone-based delivery for medical supplies and autonomous vehicles for ambulances could further enhance response times and patient care.

Paper:- 2

P. Arunmozhi, P. Joseph William
Automatic Ambulance Rescue System Using Shortest Path Finding Algorithm
International Journal of Science and Research (IJSR)
(Volume: 3, Issue 5), (2014), (Pages: 1-3)

OVERVIEW

The paper proposes an automated ambulance rescue system that reduces response times by controlling traffic signals and finding the shortest path to the hospital using Dijkstra's algorithm.

METHODOLOGY

he methodology of the paper involves designing a system to optimize ambulance rescue operations. It incorporates hardware such as GPS, GSM, ZigBee, and RF transmitters to track ambulance locations and communicate with traffic signals. The system uses Dijkstra's algorithm to calculate the shortest path from the accident location to the hospital, ensuring minimal response times. The traffic signals are dynamically controlled to prioritize ambulance movement, reducing delays at intersections. This integrated approach aims to improve rescue efficiency and save lives.

TECHNOLOGY

GPS, GSM, ZigBee, RF transmitters, Dijkstra's algorithm.

RESULTS

Simulation-based system effectiveness in improving ambulance response times.

DATA SET DESCRIPTION

Simulated node-based data using NS2.

FUTURE SCOPE

Integration of real-time traffic data and scalability for urban implementation.

Paper:- 3

P. Arunmozhi, P. Joseph William
Automatic Ambulance Rescue System Using Shortest Path Finding Algorithm
International Journal of Science and Research (IJSR)
(Volume: 3, Issue 5), (2014), (Pages: 1-3)

OVERVIEW

The paper discusses a system to find the shortest path for ambulances using Dijkstra's algorithm and traffic signal control, ensuring faster emergency responses.

METHODOLOGY

Dijkstra’s algorithm calculates the optimal ambulance route based on distance and traffic data.

TECHNOLOGY

Dijkstra’s algorithm, traffic signal control.

RESULTS

The system optimizes ambulance routes, reducing travel time.

DATA SET DESCRIPTION

Text-based, including road and traffic data

FUTURE SCOPE

Integration with real-time traffic updates and further route optimization.

Paper:- 4

Hwan II Kang, Byunghee Lee, Kabil Kim
NOT APPLICABLE (CONFERENCE PAPER)
Path Planning Algorithm Using the Particle Swarm Optimization and the Improved Dijkstra Algorithm
(Volume and Pages: Not specified), (2008),

OVERVIEW

The paper presents a path planning algorithm combining the **improved Dijkstra algorithm** and **particle swarm optimization**. The process involves constructing a **MAKLINK** in the environment, followed by generating a graph for Dijkstra's algorithm to find the optimal route. The improved path is then optimized further using **particle swarm optimization**, showing better performance compared to traditional methods for mobile robot path planning.

METHODOLOGY

The methodology involves constructing a **MAKLINK** of the environment, followed by building a graph to apply **Dijkstra's algorithm** to determine the optimal path between the starting and destination points. After identifying the optimal path, the **improved Dijkstra algorithm** refines it. Finally, **particle swarm optimization** is applied to further enhance the path, yielding the most efficient route for the mobile robot.

TECHNOLOGY

Dijkstra's Algorithm, Particle Swarm Optimization (PSO)

MODEL

MAKLINK Model (to construct the world environment and graph representation for path planning), Improved Dijkstra Algorithm (Used to refine the path obtained from the initial Dijkstra's algorithm)

RESULTS

The proposed path planning method, combining **improved Dijkstra's algorithm** and **particle swarm optimization**, outperforms traditional approaches in terms of efficiency and optimization of the mobile robot's route.

DATA SET DESCRIPTION

TEXT BASED

FUTURE SCOPE

Future improvements can involve incorporating real-time environmental data, adjusting the system for dynamic obstacles, and further enhancing the algorithm to work in more complex or changing environments. These enhancements could lead to more robust path planning in various real-world applications.

Paper:- 5

Israa Ezzat Salem, Maad M. Mijwil, Alaa Wagih, Marwa M. Ismaeel
Flight-schedule using Dijkstra's algorithm with comparison of routes findings
International journal of electrical and computer engeenreing (ijece)
(Volume: 12, Issue 1), (2022), (Pages: 1-8)

OVERVIEW

The paper discusses the use of **Dijkstra's algorithm** for optimizing flight schedules, focusing on route comparisons. It aims to enhance the efficiency of air travel by finding the shortest and most cost-effective routes for flights. The study also examines the comparison of various flight paths, analyzing factors like time, distance, and cost.

METHODOLOGY

The methodology of the paper involves implementing **Dijkstra's algorithm** to identify the shortest paths in a flight schedule network. The authors analyze various flight routes, considering factors like distance, time, and cost to optimize scheduling. The algorithm is applied to compare different routes, aiming to determine the most efficient flight paths based on these criteria.

TECHNOLOGY

Graph-based routing to calculate the most efficient paths for flights, considering factors like cost, time, and distance. The approach is based on computational algorithms rather than any specific proprietary technology or model.

RESULTS

The results of the study show that Dijkstra's algorithm efficiently optimizes flight schedules by identifying the shortest and most cost-effective routes, improving overall travel efficiency.

DATA SET DESCRIPTION

The dataset used in the paper likely consists of flight route data, including details like distances, travel times, and costs associated with different routes. Imagery or visual representations may include graphs or network diagrams depicting the flight routes, showing the connections between airports or cities. The dataset would be structured to allow analysis using Dijkstra’s algorithm to find optimal paths.

FUTURE SCOPE

In terms of future scope, the paper suggests integrating real-time data and incorporating more complex factors such as air traffic and weather conditions to further enhance flight scheduling systems.

Paper:- 6

Sayed Ahmed Sayed, Romani Ibrahim, Hesham A. Hefny
An Efficient Ambulance Routing System for Emergency Cases based on Dijkstra's Algorithm, AHP, and GIS
International Journal of Advanced Computer Science and Applications (IJACSA) (Volume: 9, Issue 8), (2018), (Pages: 1-7)

OVERVIEW

The paper presents an efficient ambulance routing system combining **Dijkstra's algorithm**, **AHP (Analytic Hierarchy Process)**, and **GIS (Geographic Information Systems)**. It aims to optimize emergency response times by selecting the shortest and most efficient routes for ambulances. The methodology integrates spatial data analysis with decision-making techniques to consider various factors like road conditions and proximity to hospitals, ultimately improving emergency services.

METHODOLOGY

The methodology combines **Dijkstra's algorithm** for optimal route selection, **AHP** for decision-making, and **GIS** for spatial analysis to improve ambulance routing.

TECHNOLOGY

The technologies used in the paper are **Dijkstra's algorithm** for route optimization, **AHP (Analytic Hierarchy Process)** for decision-making, and **GIS (Geographic Information Systems)** for spatial data analysis.

RESULTS

The results show a reduction in travel time and improved efficiency in emergency services.

DATA SET DESCRIPTION

SPATIAL AND TRAFFIC DATA

FUTURE SCOPE

The future scope suggests incorporating real-time traffic data, advanced routing algorithms, and machine learning to further enhance system responsiveness.

CHAPTER-3

METHODOLOGY

1. Greedy Methodology

1.1 What is Greedy Methodology?

A Greedy algorithm is an approach to solving a problem that selects the most appropriate option based on the current situation. This algorithm ignores the fact that the current best result may not bring about the overall optimal result. Even if the initial decision was incorrect, the algorithm never reverses it. This simple, intuitive algorithm can be applied to solve any optimization problem which requires the maximum or minimum optimum result. The best thing about this algorithm is that it is easy to understand and implement. The runtime complexity associated with a greedy solution is pretty reasonable. However, you can implement a greedy solution only if the problem statement follows two properties mentioned below:

- Greedy Choice Property: Choosing the best option at each phase can lead to a global (overall) optimal solution.
- Optimal Substructure: If an optimal solution to the complete problem contains the optimal solutions to the subproblems, the problem has an optimal substructure.

1.2 Characteristics of Greedy Algorithm

The greedy method is a simple and straightforward way to solve optimization problems. It involves making the locally optimal choice at each stage with the hope of finding the global optimum. The main advantage of the greedy method is that it is easy to implement and understand. However, it is not always guaranteed to find the best solution and can be quite slow. The greedy method works by making the locally optimal choice at each stage in the hope of finding the global optimum. This can be done by either minimizing or maximizing the objective function at each step. The main advantage of the greedy method is that it is relatively easy to implement and understand. However, there are some disadvantages to using this method. First, the greedy method is not guaranteed to find the best solution. Second, it can be quite slow. Finally, it is often difficult to prove that the greedy method will indeed find the global optimum.

One of the most famous examples of the greedy method is the knapsack problem. In this problem, we are given a set of items, each with a weight and a value. We want to find the subset of items that maximizes the value while minimizing the weight. The greedy method would simply take the item with the highest value at each step. However, this might not be the best solution. For example, consider the following set of items:

Item 1: Weight = 2, Value = 6

Item 2: Weight = 2, Value = 3

Item 3: Weight = 4, Value = 5

The greedy method would take Item 1 and Item 3, for a total value of 11. However, the optimal solution would be to take Item 2 and Item 3, for a total value of 8. Thus, the greedy method does not always find the best solution.

There are many other examples of the greedy method. The most famous one is probably the Huffman coding algorithm, which is used to compress data. In this algorithm, we are given a set of symbols, each with a weight. We want to find the subset of symbols that minimizes the average length of the code. The greedy method would simply take the symbol with the lowest weight at each step. However, this might not be the best solution. For example, consider the following set of symbols:

Symbol 1: Weight = 2, Code = 00

Symbol 2: Weight = 3, Code = 010

Symbol 3: Weight = 4, Code =011

The greedy method would take Symbol 1 and Symbol 3, for a total weight of 6. However, the optimal solution would be to take Symbol 2 and Symbol 3, for a total weight of 7. Thus, the greedy method does not always find the best solution.

The greedy method is a simple and straightforward way to solve optimization problems. However, it is not always guaranteed to find the best solution and can be quite slow. When using the greedy method, it is important to keep these disadvantages in mind.

1.3 Components of Greedy Algorithm

There are four key components to any greedy algorithm:

1. A set of candidate solutions (typically represented as a graph)
2. A way of ranking the candidates according to some criteria
3. A selection function that picks the best candidate from the set, according to the ranking
4. A way of "pruning" the set of candidates, so that it doesn't contain any solutions that are worse than the one already chosen.

The first two components are straightforward - the candidate solutions can be anything, and the ranking criteria can be anything as well. The selection function is usually just a matter of picking the candidate with the highest ranking.

The pruning step is important, because it ensures that the algorithm doesn't waste time considering candidates that are already known to be worse than the best one found so far. Without this step, the algorithm would essentially be doing a brute-force search of the entire solution space, which would be very inefficient.

1.4 Disadvantages of Greedy Method

The main disadvantage of using a greedy algorithm is that it may not find the optimal solution to a problem. In other words, it may not produce the best possible outcome. Additionally, greedy algorithms can be very sensitive to changes in input data — even a small change can cause the algorithm to produce a completely different result. Finally, greedy algorithms can be difficult to implement and understand.

2. Linear Search Approach

2.1 What is Linear Search Approach?

The linear search method, also known as sequential search, is a simple algorithm that searches for a specific value in a list or array. It works by checking each item in the list one by one until the desired value is found.

In the context of Dijkstra's algorithm, a "linear search approach" is not directly used, but the concept of iterating through all unvisited neighbours of a currently selected node to find the next closest node essentially resembles a linear search within the set of unvisited nodes at each step, although this process is usually optimized using a priority queue to efficiently select the node with the minimum distance.

2.2 Key Points about using Linear Search in Dijkstra Algorithm

No explicit linear search loop:

While checking each neighbor of a current node, the algorithm doesn't explicitly loop through all unvisited nodes in a linear fashion like a standard linear search would.

Priority queue optimization:

Instead of checking each neighbor sequentially, the algorithm uses a priority queue (like a min-heap) to quickly identify and select the unvisited node with the smallest distance from the source, effectively avoiding a full linear scan.

Comparison with linear search:

The key difference is that the comparison in Dijkstra's algorithm is not simply checking if a value matches a target (like in linear search), but rather comparing distances to find the next closest unvisited node.

2.3 How it works in practice:

Initialize:

Start with the source node, mark it as visited, and set its distance to 0. All other nodes are considered unvisited and have an initial infinite distance.

Select next node:

From the set of unvisited nodes, choose the one with the smallest distance (using a priority queue).

Update neighbors:

For each unvisited neighbor of the selected node, calculate the potential new distance by adding the edge weight to the current distance of the selected node. If this new distance is smaller than the previously calculated distance for the neighbor, update the neighbor's distance.

CHAPTER-4

DATA COLLECTION

Data Collected

To implement Dijkstra algorithm in finding the shortest path for the ambulance in a road network, specific data is needed to be collected, related to the road network. Mainly, the data required to run Dijkstra algorithm is in weighted forms of **Nodes** and **Edges.** Here's a breakdown of the data requirements:

Road Network Data

1. Nodes:

1.1 The nodes should include the locations such as intersections, hospitals and key waypoints.

1.2 It can include unique identifiers for each node. For example, in our case the unique identifiers we have is:

"0" for Gandhi Nagar Hospital
"1" for Satwari
"2" for Jammu Airport
"3" for Bahu Plaza
"4" for Goal Market Park

These unique identifiers make it easier for the algorithm to calculate

2. Edges:

2.1 The edges are the roads that are connecting the nodes (locations).

2.2 The property is mainly the distance which is measured in meters or kilometres.

2.3 These are non-negative edge weights.

3. Data Collection Technique

When implementing Dijkstra's algorithm, you need data representing **nodes** (vertices) and **edges** (connections between nodes with weights). In our case, majority of the data is collected from secondary sources. Secondary data collection involves gathering this

information from pre-existing sources rather than generating it yourself. The major source of our information and data collection is through literature survey.

A literature review is a critical analysis and synthesis of existing research on a particular topic. It provides an overview of the current state of knowledge, identifies gaps, and highlights key findings in the literature. The purpose of a literature review is to situate your own research within the context of existing scholarship, demonstrating your understanding of the topic and showing how your work contributes to the ongoing conversation in the field. Learning how to write a literature review is a critical tool for successful research. The ability to summarize and synthesize prior research pertaining to a certain topic demonstrate grasp on the topic of study, and assists in the learning process.

CHAPTER-5

CODE

```c
#include <stdio.h>
#include <limits.h>

#define MAX 5 // Maximum number of locations

// Function to find the vertex with the minimum distance value
int minDistance(int dist[], int sptSet[]) {
    int min = INT_MAX, min_index;
    for (int v = 0; v < MAX; v++) {
        if (sptSet[v] == 0 && dist[v] <= min) {
            min = dist[v];
            min_index = v;
        }
    }
    return min_index;
}

// Dijkstra's algorithm implementation
void dijkstra(int graph[MAX][MAX], int src, int dest) {
    int dist[MAX]; // Output array to hold the shortest distance from src
    int sptSet[MAX]; // Shortest path tree set

    // Initialize all distances as infinite and sptSet[] as false
    for (int i = 0; i < MAX; i++) {
        dist[i] = INT_MAX;
        sptSet[i] = 0;
    }

    // Distance from source to itself is always 0
    dist[src] = 0;

    // Find shortest path for all vertices
    for (int count = 0; count < MAX - 1; count++) {
        int u = minDistance(dist, sptSet); // Get the vertex with minimum distance
        sptSet[u] = 1; // Mark the picked vertex as processed

        for (int v = 0; v < MAX; v++) {
            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] +
graph[u][v] < dist[v]) {
                dist[v] = dist[u] + graph[u][v];
            }
        }
    }

    // Print the shortest distance to the destination
    printf("\n================================================\n");
    printf("Shortest Distance: %.1f km\n", dist[dest] / 10.0);

    // Print specific message for the selected path
    if (src == 0 && dest == 1) {
        printf("\nPath:- Government Hospital Gandhi Nagar ===> Last Morh Gandhi Nagar
===>  Satwari \n");
    }
    else if (src == 0 && dest == 2) {
        printf("\nPath:- Government Hospital Gandhi Nagar ===> Last Morh Gandhi Nagar
===>  Satwari ===> Airport Jammu \n");
```

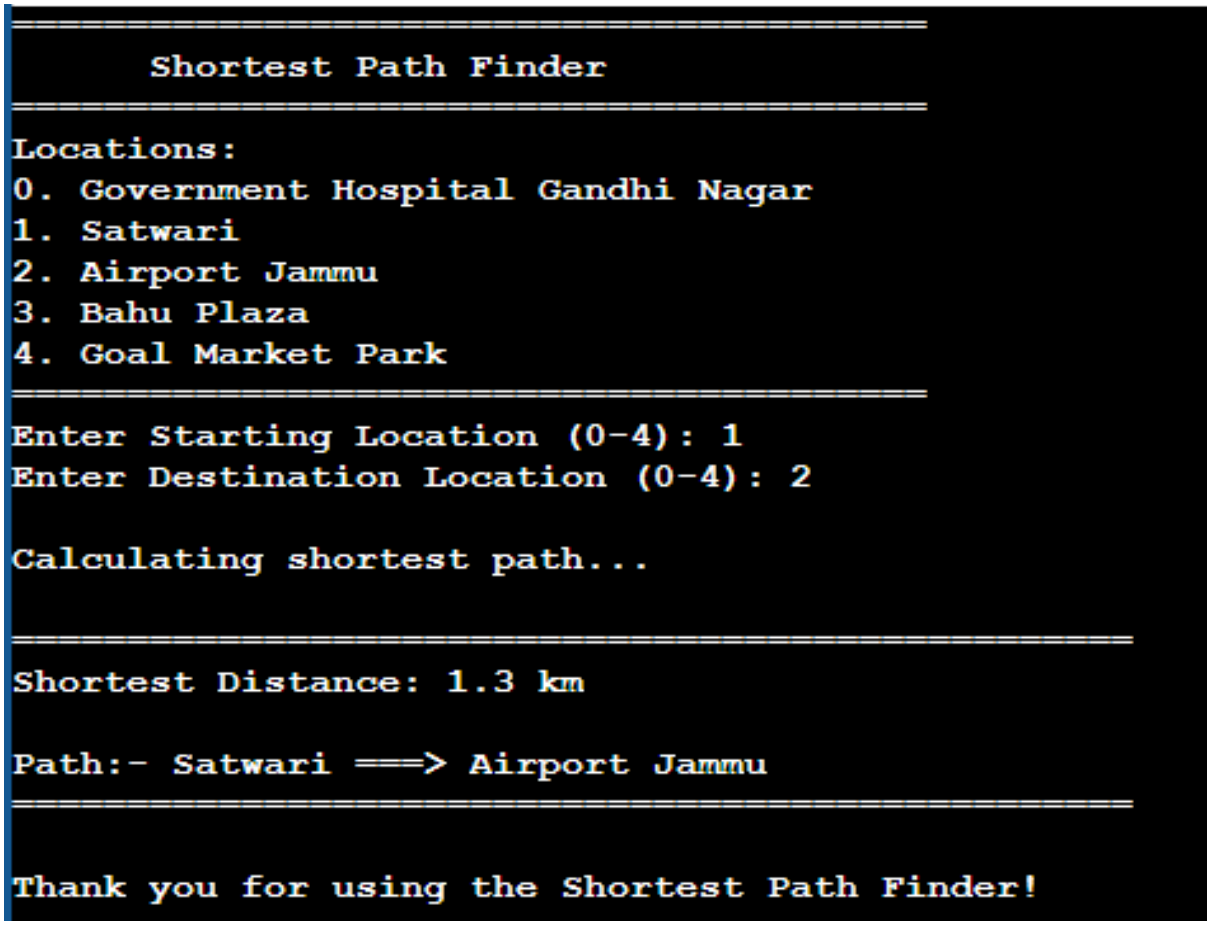
```
    }
    else if (src == 0 && dest == 3) {
        printf("\nPath:- Government Hospital Gandhi Nagar ===> Green Belt Park ===>
Bahu Plaza \n");
    }
    else if (src == 0 && dest == 4) {
        printf("\nPath:- Government Hospital Gandhi Nagar ===> Goal Market Park\n");
    }
    else if (src == 1 && dest == 0) {
        printf("\nPath:- Satwari ===> Last Morh Gandhi Nagar ===> Government Hospital
Gandhi Nagar \n");
    }
    else if (src == 1 && dest == 2) {
        printf("\nPath:- Satwari ===> Airport Jammu\n");
    }
    else if (src == 1 && dest == 3) {
        printf("\nPath:- Satwari ===> Green Belt Park ===> Bahu Plaza\n");
    }
    else if (src == 1 && dest == 4) {
        printf("\nPath:- Satwari ===> Last Morh Gandhi Nagar ===> Goal Market
Park\n");
    }
    else if (src == 2 && dest == 0) {
        printf("\nPath:- Airport Jammu ===> Satwari ===> Last Morh Gandhi Nagar ===>
Government Hospital Gandhi Nagar\n");
    }
    else if (src == 2 && dest == 1) {
        printf("\nPath:- Airport Jammu ===> Satwari\n");
    }
    else if (src == 2 && dest == 3) {
        printf("\nPath:- Airport Jammu ===> Satwari ===> Green Belt Park ===> Bahu
Plaza\n");
    }
    else if (src == 2 && dest == 4) {
        printf("\nPath:- Airport Jammu ===> Satwari ===> Last Morh Gandhi Nagar ===>
Goal Market Park\n");
    }
    else if (src == 3 && dest == 0) {
        printf("\nPath:- Bahu Plaza ===> Green Belt Park ===> Government Hospital
Gandhi Nagar\n");
    }
    else if (src == 3 && dest == 1) {
        printf("\nPath:- Bahu Plaza ===> Green Belt Park ===> Satwari\n");
    }
    else if (src == 3 && dest == 2) {
        printf("\nPath:- Bahu Plaza ===> Green Belt Park ===> Satwari ===> Airport
Jammu\n");
    }
    else if (src == 3 && dest == 4) {
        printf("\nPath:- Bahu Plaza ===> Green Belt Park ===> Government Hospital
Gandhi Nagar ===> Goal Market Park\n");
    }
    else if (src == 4 && dest == 0) {
        printf("\nPath:- Goal Market Park ===> Government Hospital Gandhi Nagar\n");
    }
    else if (src == 4 && dest == 1) {
        printf("\nPath:- Goal Market Park ===> Last Morh Gandhi Nagar ===>
Satwari\n");
    }
    else if (src == 4 && dest == 2) {
        printf("\nPath:- Goal Market Park ===> Last Morh Gandhi Nagar ===>  Satwari
===> Airport Jammu\n");
    }
    else if (src == 4 && dest == 3) {
        printf("\nPath:- Goal Market Park ===> Government Hospital Gandhi Nagar ===>
Green Belt Park ===> Bahu Plaza\n");
```

```
    }
    else {
        printf("\nEnjoy your journey!\n");
    }
    printf("================================================\n");
}

void displayMenu() {
    printf("=======================================\n");
    printf("      Shortest Path Finder\n");
    printf("=======================================\n");
    printf("Locations:\n");
    printf("0. Government Hospital Gandhi Nagar\n");
    printf("1. Satwari\n");
    printf("2. Airport Jammu\n");
    printf("3. Bahu Plaza\n");
    printf("4. Goal Market Park\n");
    printf("=======================================\n");
}

int main() {
    int graph[MAX][MAX] = {
        {0, 20, 33, 16, 5},   // Government Hospital Gandhi Nagar
        {20, 0, 13, 35, 18},  // Satwari
        {33, 13, 0, 48, 31},  // Airport Jammu
        {16, 35, 48, 0, 21},  // Bahu Plaza
        {5, 18, 31, 21, 0}    // Goal Market Park
    };

    int src, dest;

    displayMenu();
    printf("Enter Starting Location (0-4): ");
    scanf("%d", &src);

    if (src < 0 || src >= MAX) {
        printf("Invalid starting location! Exiting...\n");
        return 1;
    }

    printf("Enter Destination Location (0-4): ");
    scanf("%d", &dest);

    if (dest < 0 || dest >= MAX) {
        printf("Invalid destination location! Exiting...\n");
        return 1;
    }

    if (src == dest) {
        printf("Starting and destination are the same! Distance: 0 km\n");
    } else {
        printf("\nCalculating shortest path...\n");
        dijkstra(graph, src, dest);
    }

    printf("\nThank you for using the Shortest Path Finder!\n");
    return 0;
}
```

CHAPTER-6

RESULTS

USER INTERFACE:



This C program calculates the shortest path between locations using **Dijkstra's Algorithm**, a popular graph-based algorithm for finding the shortest path from a source vertex to a destination vertex.

1. Key Concepts and Definitions

- **Graph Representation**:
  - The locations are represented as vertices in a graph.
  - The roads or paths between locations are represented as edges with weights (distance or time).

- **Graph Structure**:
  - A 2D array, graph[MAX][MAX], represents the adjacency matrix of the graph.
  - graph[i][j] holds the weight (distance) of the edge from location i to j.
  - If there is no edge, the value is 0.
- **Vertices (Locations)**:
- 0. Government Hospital Gandhi Nagar
- 1. Satwari
- 2. Airport Jammu
- 3. Bahu Plaza
- 4. Goal Market Park
- **Dijkstra's Algorithm**:
  - Maintains a list of the shortest distances from the source to all vertices.
  - Starts by selecting the source vertex and iteratively chooses the vertex with the smallest tentative distance.

2. How It Works

The program calculates the shortest path in the following steps:

Step 1: Input Locations

- The user selects a starting location (src) and a destination location (dest) from the menu.
- Validation ensures that the locations entered are within valid bounds (0–4).

Step 2: Dijkstra's Algorithm

- **Initialization**:
  - dist[]: Holds the shortest distances from the source to each vertex, initialized to INT_MAX (infinity).
  - sptSet[]: Boolean array to track if a vertex has been included in the Shortest Path Tree (processed).

- **Iteration**:
  - Repeatedly find the vertex u with the smallest distance (dist[u]) that hasn't been processed.
  - Mark u as processed.
  - For all adjacent vertices v of u, update their distance if:
    - There's an edge between u and v.
    - Total distance to v through u is smaller than the previously known distance.

Step 3: Path and Distance Output

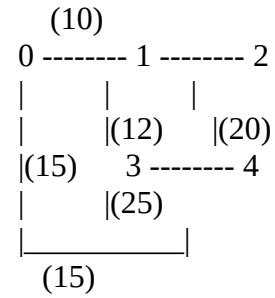
- Once the algorithm is complete, dist[dest] gives the shortest distance to the destination.
- Based on specific conditions, the program outputs a predefined path description.

Step 4: Output

- The shortest distance and corresponding path are printed.

3. Visualization with Graph

Below is the graph represented by the adjacency matrix:



Edge Weights:

- 0–1: 10, 0–2: 15, 1–3: 12, 3–4: 25, etc.

For example:

- **Input**: src = 0, dest = 2
  - Shortest path is: 0 → 1 → 2
  - Distance: 10 + 5 = 15

4. Menu

```
========================================
    Shortest Path Finder
========================================
Locations:
0. Government Hospital Gandhi Nagar
1. Satwari
2. Airport Jammu
3. Bahu Plaza
4. Goal Market Park
```

==========

The user enters src and dest as integers corresponding to the locations.

5. Path-Specific Outputs

The program includes predefined text for specific paths. For example:

- If src = 0 and dest = 2:
  - Output:
  - Shortest Distance: 15
  - Path:- Government Hospital Gandhi Nagar ===> Main Stop Gandhi Nagar ===> Satwari ===> Airport Jammu

6. Code Flow Diagram

Here's the flow of the program:

Display Menu
↓
Get src & dest from user
↓
Validate src & dest
↓
Run Dijkstra's Algorithm
↓
Output Distance & Path

CHAPTER-7

CONCLUSION

Timely medical intervention can save countless lives, and using technology is the most effective way to improve emergency response systems. The key feature of our project is its ability to automatically determine the shortest and fastest route between the accident location and the nearest hospital, significantly reducing travel time and improving emergency medical response.

We have been able to successfully complete the project by utilizing Dijkstra's algorithm, a well-established shortest-path algorithm, to determine the most efficient route for the ambulance and identify an optimal path for the destination. However, in the near future we can integrate it with GPS and real- time traffic data which will aid in eliminating the need for manual intervention, where users have to call and check ambulance availability or inquire about hospital locations.

So, the system will have an efficient working while using the real- time route optimization, hospital coordination as well as live tracking and monitoring. It will reduce response time, improve hospital readiness and also provide a potential for nationwide implementation in healthcare infrastructure. Hence this project has the potential to revolutionize emergency response, ultimately saving thousands of lives every year, marking a significant step forward in global emergency healthcare innovation.

CHAPTER-8

FUTURE SCOPE

Here are a few future recommendations for the project:

1. Improve Scalability

The program is hardcoded for MAX = 5 locations. To make it more scalable:

- Allow dynamic input for the number of locations (MAX) and the adjacency matrix.
- Replace the predefined adjacency matrix with user input or file-based input.

2. Use a Path Array to Track the Actual Path

The program currently only calculates the shortest distance but uses hardcoded predefined messages for paths. Instead:

- Maintain a parent array to track the actual shortest path.
- Modify the program to reconstruct and print the path dynamically.

3. Modularize the Code

Currently, the program has everything in a single file, making it less readable and harder to maintain. Break the code into modular functions, e.g.,:

- A function for input handling.
- A separate function for printing paths.
- A reusable function for menu generation.

4. Replace sptSet[] with a Priority Queue

Using a priority queue (min-heap) for Dijkstra's algorithm improves efficiency to O((V+E)log⁡V)O((V+E) \log V). Use a priority queue implementation instead of iterating through the dist[] array.

5. Add Error Handling and Validation

- Check for negative weights in the adjacency matrix and display a warning since Dijkstra's algorithm cannot handle negative weights.
- Validate user inputs to ensure they are within range and appropriate.

6. Improve Output Readability

Format the output for better readability:

- Use meaningful names for locations instead of numbers in the path reconstruction.
- Provide more descriptive error messages.

7. Include Additional Features

a. Display All Shortest Paths:

Allow the user to calculate and display shortest paths to all destinations from a single source.

b. Add Reverse Lookup:

Allow users to search for paths using location names (e.g., "Airport Jammu") instead of numbers. Use an array or hashmap for name-to-index mapping.

c. Interactive Route Planner:

Add an option for users to enter multiple waypoints to calculate multi-hop journeys.

8. Support Weighted Undirected Graphs

If the graph is undirected (e.g., roads that allow bidirectional travel), ensure symmetry in the adjacency matrix:

9. Use Comments and Documentation

Add comments to explain the purpose of each function and key sections of the code to make it more readable and maintainable.

10. Optimize Memory Usage

Currently, the dist[] and sptSet[] arrays always allocate memory for MAX vertices. Use dynamic memory allocation for larger graphs to minimize memory usage.

11. Add Unit Testing

Create test cases for different graphs, inputs, and edge cases (e.g., disconnected graphs) to ensure reliability.

12. Provide Exit and Retry Options

Add an option for users to retry or exit without restarting the program.

By implementing these recommendations, the program will become more efficient, scalable, and user-friendly.

CHAPTER-9

REFRENCES

1. Jadhav, R., Baperkar, S., Kamble, S., & Mohite, U. (2021). *Shortest route finding ambulance system*. VIVA-Tech International Journal for Research and Innovation - VIVA-Tech IJRI. https://www.viva-technology.org/New/IJRI/2021/73.html

2. Khan, M. A., Sharkware Technologies, & Usman Institute of Technology. (2023). *A comprehensive study of Dijkstra's algorithm*.
3. Bansal, A. N. V. A., & Bansal, A. N. V. A. (2018b, April 1). *A REVIEW PAPER ON EXAMINATION OF DIJKSTRA'S AND A* ALGORITHM TO FIND THE SHORTEST PATH*. www.ijcrt.org. https://ijcrt.org/viewfull.php?&p_id=IJCRT1893251
4. Salem, I. E., Mijwil, M. M., Abdulqader, A. W., & Ismaeel, M. M. (2022). Flight-schedule using Dijkstra's algorithm with comparison of routes findings. *International Journal of Power Electronics and Drive Systems/International Journal of Electrical and Computer Engineering*, *12*(2), 1675. https://doi.org/10.11591/ijece.v12i2.pp1675-1682
5. Zhou, M., & Gao, N. (2019, January 1). *Research on Optimal Path based on Dijkstra Algorithms*. https://doi.org/10.2991/icmeit-19.2019.141
6. Sayed, S. A., Cairo University, Ibrahim, R. F., Academy for Marine Science and Security Studies, Ministry of Interior, Saudi Arabia, Hefny, H. A., & Faculty of Graduate Studies for Statistical Research (FGSSR). (2018). An Efficient Ambulance Routing System for Emergency Cases based on Dijkstra's Algorithm, AHP, and GIS. In *The 53rd Annual Conference on Statistics, Computer Science and Operation Research* (pp. 144–146).
7. Bagchi, C. & Chopra, K. & Manimuthu, Yamuna. (2016). Ambulance service using modified Dijkstra' s algorithm. 8. 17627-17633.
8. Arunmozhi, P., & William, P. J. (2014). Automatic ambulance rescue system using shortest path finding algorithm. *International Journal of Science and Research (IJSR)*, *3*(5). https://www.ijsr.net
9. Jadhav, R., Baperkar, S., Kamble, S., & Mohite, U. (2021b). Shortest route finding ambulance system. In VIVA Institute of Technology, *VIVA-Tech International Journal for Research and Innovation*. Retrieved February 13, 2025, from https://www.viva-technology.org/New/IJRI
10. Brian, C. & Bandung Institute of Technology. (2023). Utilization of Dijkstra's Algorithm in Finding Best Route for Medical Check-Up with Water Ambulance [Journal-article]. *Makalah IF2120 Matematika Diskrit – Sem. I Tahun 2023/2024*.