

# PathSmart-Optimized Routes

**System**

***Project report***

*submitted in partial fulfillment of the requirements for the award of the degree of*

## BACHELOR OF TECHNOLOGY

**in**

## COMPUTER SCIENCE & ENGINEERING

**by**

**Name Roll No.**

Harsh Bhartia R2142221221

Shailav Justa R2142220638

Vinayak Singh R2142221177

***under the guidance of***

**Dr. Virender Kadayan**



# School of Computer Science

**University of Petroleum & Energy Studies Bidholi, Via Prem Nagar, Dehradun, Uttarakhand Aug 2024**

1



## DECLARATION

We hereby certify that the project work entitled **PathSmart-Optimized Routes System** in partial fulfilment of the requirements for the award of the Degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING with specialization in AIML and submitted to the Department of AIML, School of Computer Science, University of Petroleum & Energy Studies, Dehradun, is an authentic record of our work carried out during a period from **Aug 2024** to **till date** under the supervision of **Dr. Virender Kadayan** Assistant Professor (SG)

The matter presented in this project has not been submitted by us for the award of any other degree of this or any other University.

|  |  |  |
| --- | --- | --- |
| **Harsh Bhartia** | **Shailav Justa** | **Vinayak Singh** |
| **R2142221221** | **R2142220638** | **R2142221177** |

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.



# Acknowledgement

We wish to express our deep gratitude to our mentor **Dr. Virender Kadayan**, for all advice, encouragement and constant support he has given us throughout our project work. This work would not have been possible without his support and valuable suggestions.

We sincerely thanks to our respected **Dr. Anil Kumar**(**Cluster Head of AIML),** for his great support our project.

We are also grateful to Dean SoCS UPES for giving us the necessary facilities to carry out our project work successfully. We also thanks to our Course Coordinator, Mr. Pawan Singh Koranaga and our Activity Coordinator **Dr.Rajesh Bhatt** for providing timely support and information during the completion of this project.

# ABSTRACT

**The optimized Routing System is an integrated project designed to calculate shortest path between locations.**

# The system features a user friendly front end for inputting locations & visualising routes while backend handles real time route distance calculation using graph.



**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Contents** | **Page No** |
|  | **1. Introduction** | **1-4** |
|  | 1.1. Background information | 1 |
|  | 1.2. Problem Statement | 1 |
|  | 1.3. Main Objective | 1 |
|  | 1.4. Methodology | 2-3 |
|  | 1.5. Reference software model | 3-4 |
|  | 1.6. Branching Strategy used | 4 |
|  | **2. Implementation** | **5-13** |
|  | 2.1. Algorithm | 5 |
|  | 2.2. Data Structures | 5-6 |
|  | 2.3. UML diagrams | 6-7 |
|  | - Class diagram | 6 |
|  | - Sequence diagram | 7 |
|  | - Data flow diagram | 7 |
|  | 2.4. Code implementation | 8-10 |
|  | 2.5. Related work | 10-11 |
|  | 2.6. Work Done Till Now | 11-12 |
|  | 2.7 Proposed System Design | 12-13 |
|  | 2.8 Literature Review |  |
|  | **3. Analysis** | **13-15** |
|  | 3.1. SWOT analysis | 13 |
|  | 3.2. Results and Discussions | 13-15 |
|  | **4. Future Work** | **16** |
|  | **5. Conclusion** | **17** |
|  | **6. References** | **18** |



# Introduction

## Background information

An **optimized routing system** refer to the use of advanced algorithms, technologies, and methodologies to determine the most efficient paths for travel or data transmission. These systems aim to reduce travel time, fuel consumption, or data transmission delays by selecting routes that avoid obstacles, congestion, or hazards, while considering factors such as distance, speed, traffic, and fuel costs.

An **optimized routing system** refers to a technology-driven solution designed to determine the most efficient paths for traveling from one location to another, typically minimizing factors like time, distance, or fuel consumption. Such systems are crucial in various industries, including transportation, logistics, and personal navigation, enhancing both operational efficiency and user experience.

## Problem Statement

The Optimized navigation and Routing System is a Software which would enable the users to find the shortest distance between any two locations be it any city, also it would enable to navigate to a particular location.

Objective:- Its primary Objective is to develop a software application which would enable users to find the shortest path between any two locations and also it would enable the users to get real time navigation and as the name says optimized routes on the basis of distance.

Key Requirements:-Some of the key requirements of the application would be calculating

interface, providing it reliability and Scalability.

Expected Outcome:- The Successful development of the problem will result in a reliable navigation tool helping users to find the shortest distance.

## Main Objective

The primary objective of an optimized routing system and navigation is to efficiently determine the best possible routes for transportation or data transmission by considering variable, such as distance. The goal is to minimize delays, fuel consumption, and resource use while maximizing convenience, safety, and operational efficiency.

## Specifically, the objectives include:

1. **Minimization of Travel Time and Distance**: To calculate and recommend the fastest or shortest paths between origin and destination, ensuring users or vehicles spend the least time in transit.
2. **Real-Time Adaptability**: To provide dynamic routing that can adjust to real-time conditions such as traffic congestion, road closures, accidents, or weather changes, ensuring continued optimization during the journey.
3. **Environmental Sustainability**: To reduce carbon emissions and environmental impact by promoting eco-friendly routes and optimizing for fuel efficiency, which is crucial for sustainable transportation systems.

## Methodology

Developing an optimized routing system and navigation involves a multi-disciplinary approach that integrates advanced algorithms, proces, and user-centric design. The methodology can be broken down into several key stages, each focused on achieving optimal performance in routing and navigation under dynamic conditions.

The methodology for developing an optimized routing system and navigation is multi- faceted, involving data collection, algorithm design, real-time data integration, testing, and user-centric deployment. Through this systematic approach, routing systems are designed to deliver real-time, adaptive, and efficient navigation solutions that meet the diverse needs of users in dynamic environments. The integration of machine learning, real-time data processing, and multi-criteria optimization ensures these systems can continue to evolve and provide optimal performance in increasingly complex transportation and communication networks.

## Reference software model

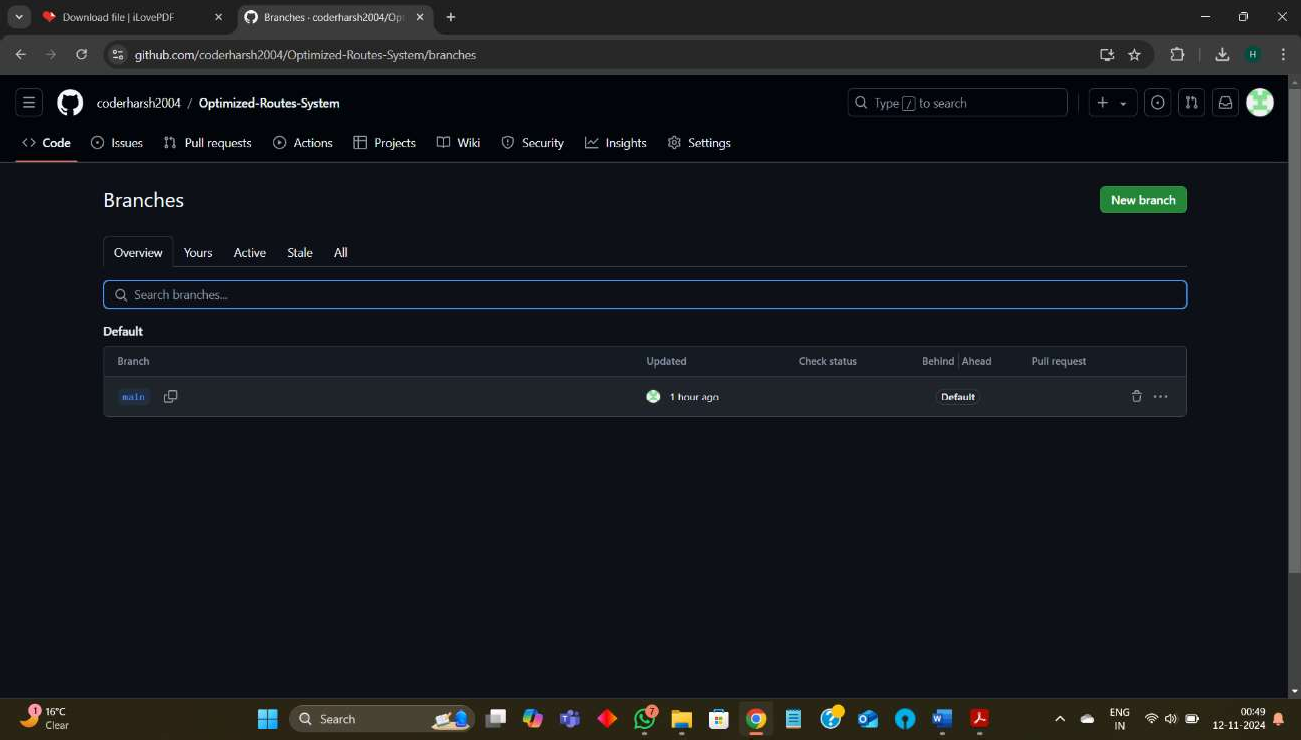
In this project, the **Route Optimization** focuses on achieving the primary objective of optimizing distance within graphs using the Dijkstra s algorithm.

The steps involved in this approach are broken down into key goals, with each guiding the development and implementation of the project. By setting these clear, goal-driven milestones, the project ensures focused progress toward creating a faster, more efficient search mechanism in linked lists, with continuous testing and improvements at each stage.

We are maintaining our goals log using **github** where we can assign goals and set mile stones as well.

## Branching strategy used

We have used a single main branch in the github version control and worked in collaboration by directly committing into the main branch resolving git conflicts if any arise and using git pull to pull the code.





# Implementation

## Algorithm(Dijkstra s Algorithm- Works on Greedy Approach):-

### Initialize:

Create two vectors, dist and parent, of size n.

* + dist[i] will hold the minimum distance from the source to node i. Initialize all distances as INT\_MAX (representing infinity) except dist[source] = 0.
  + parent[i] will hold the previous node for path reconstruction, initialized to -1 for all nodes.

Initialize a priority queue pq to store nodes with their current shortest distance from the source.

### Insert Source into Priority Queue:

Push a Node with dist = 0 and vertex = source into the priority queue pq.

### Process Nodes in Priority Queue:

While the priority queue pq is not empty:

### Extract Node with Minimum Distance:

Pop the top element from pq, which contains the node u with the smallest currentDist.

### Skip Outdated Distances:

If currentDist is greater than dist[u], continue to the next iteration, as this means a shorter path to u has already been found.

### Relax Edges:

For each neighbor v of u (loop through all nodes):

If there is an edge between u and v (graph[u][v] != -1) and dist[u] + graph[u][v] < dist[v], update dist[v] and parent[v]:

Set dist[v] = dist[u] + graph[u][v].

Set parent[v] = u (indicating that the shortest path to v comes through u).

Push a Node with dist[v] and vertex = v into pq.

### Output Results:

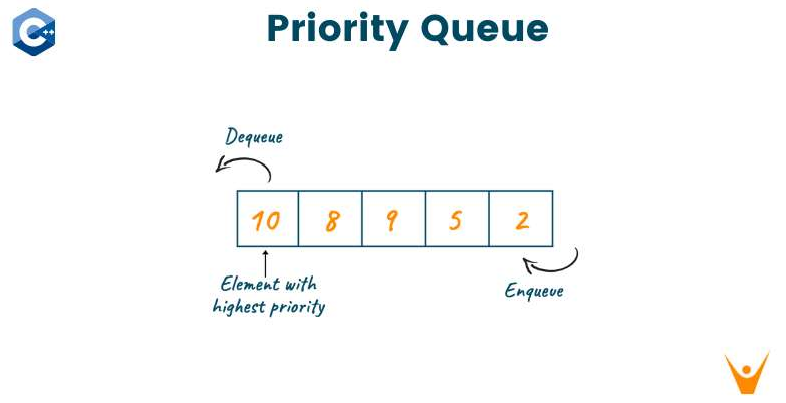
After the algorithm terminates, check dist[destination]:

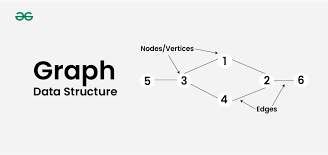
* + If dist[destination] == INT\_MAX, output INF, indicating that the destination is unreachable.
  + Otherwise, output dist[destination], which is the shortest distance from the source to the destination.



## Data structures

* + 1. **Node**: The fundamental unit of the linked list, containing the data and a reference to the next node.
    2. **Priority Queue:** A **priority queue** is a specialized data structure that operates similarly to a regular queue, but with a key difference: each element has a priority level, and elements are dequeued based on their priority rather than their order of arrival. This allows elements with higher priority (or lower numerical priority values in some cases) to be processed first, regardless of their position in the queue.
    3. **Graph**: A **graph** is a data structure that consists of a set of nodes (also called vertices) and edges connecting these nodes. Graphs are used to represent relationships or connections between objects, making them ideal for modeling real-world networks like roads, social networks, web links, and more.

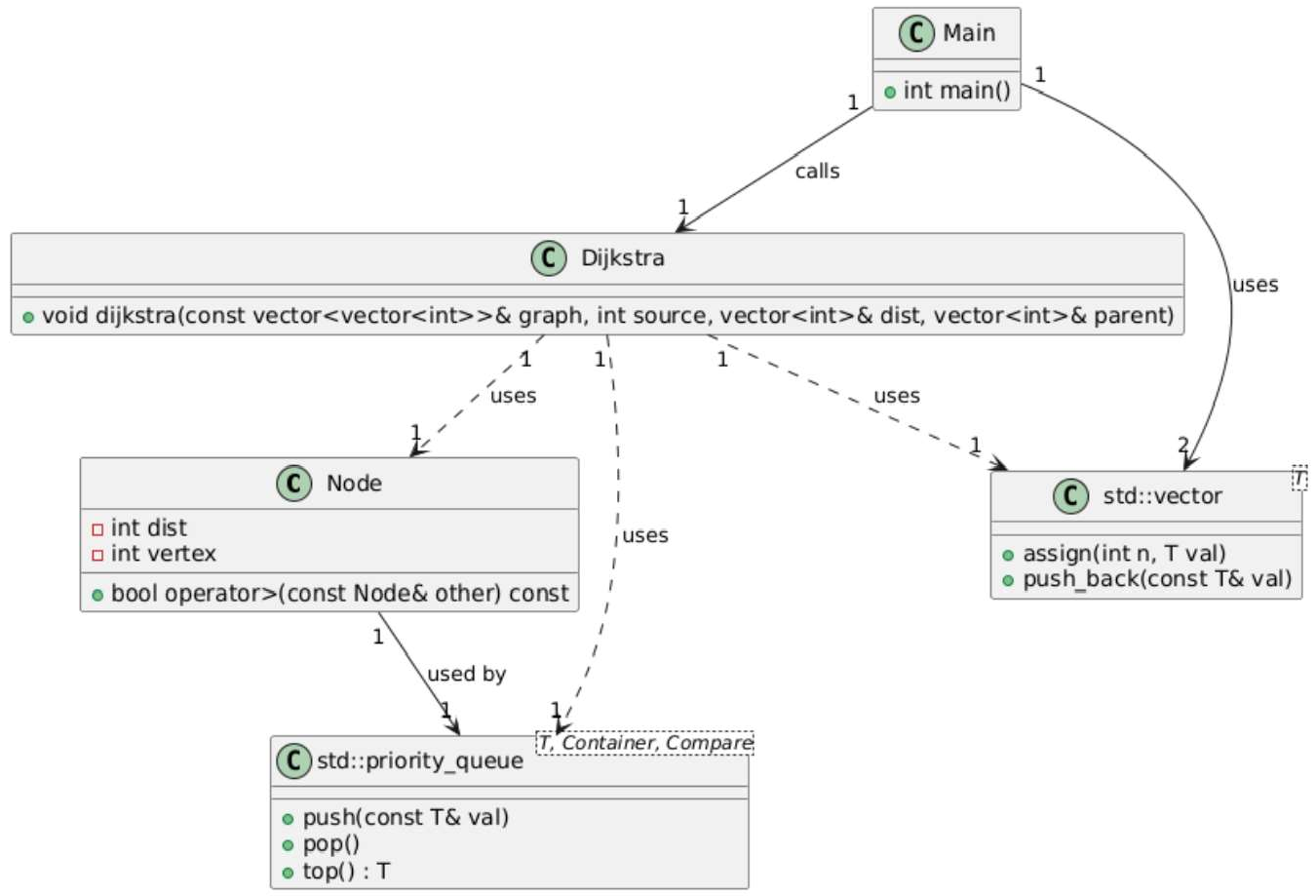






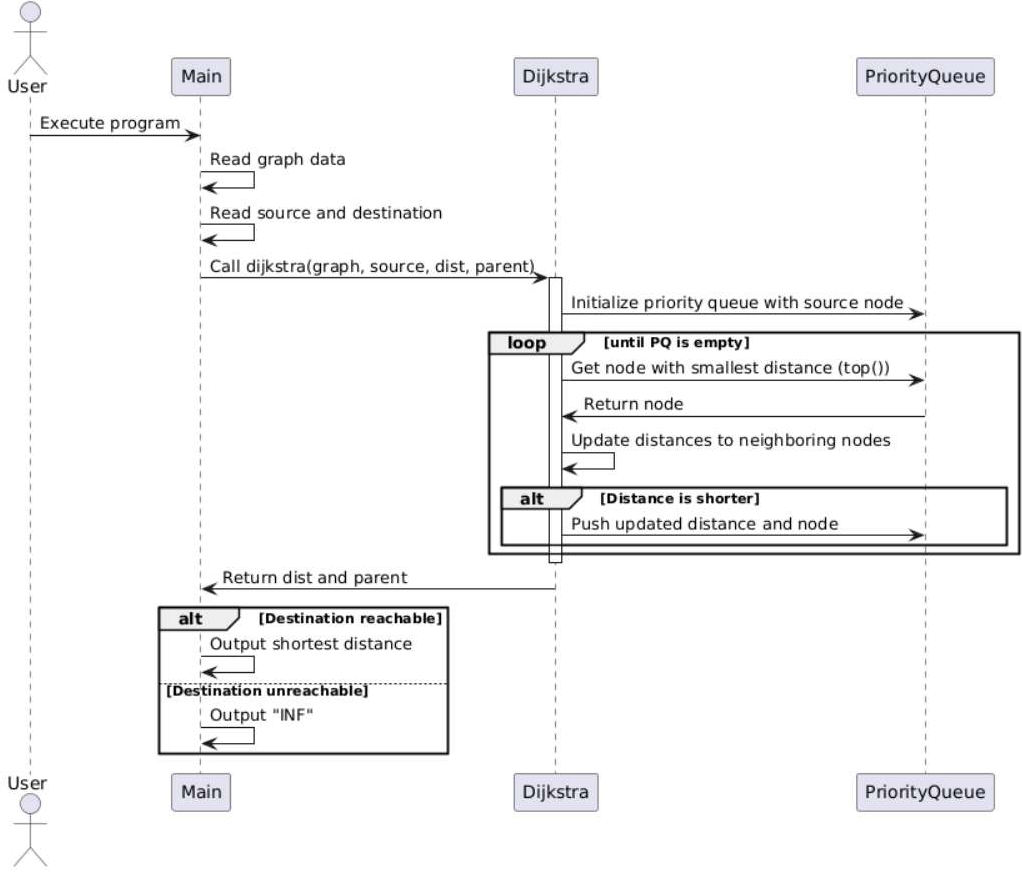
.

## UML diagrams Class Diagram:-

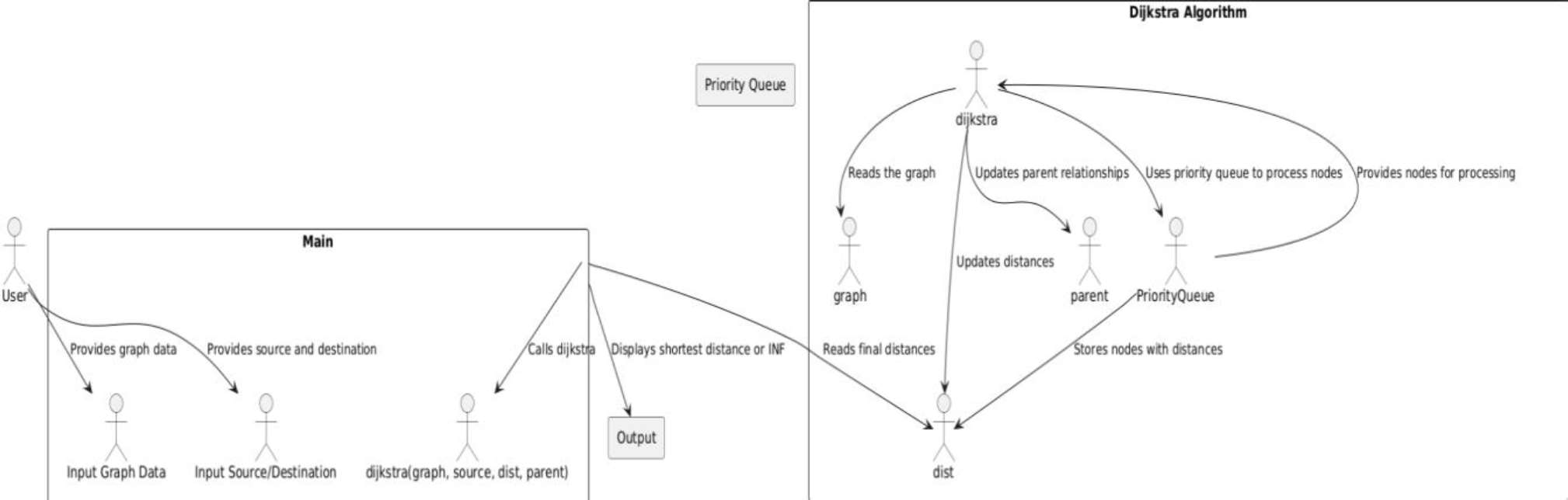




### Sequence diagram



**Data flow diagram**





## Code implementation

#include <iostream> #include <vector> #include <climits> #include <queue> #include <sstream> using namespace std;

// Define a structure for the priority queue element struct Node {

int dist; // The distance from the source int vertex; // The node index

// Comparison operator to prioritize nodes with smaller distances bool operator>(const Node& other) const {

return dist > other.dist;

}

};

// Dijkstra's algorithm using a priority queue

void dijkstra(const vector<vector<int>>& graph, int source, vector<int>& dist, vector<int>& parent) { int n = graph.size();

dist.assign(n, INT\_MAX); parent.assign(n, -1);

dist[source] = 0;

// Priority queue to store {distance, node} pairs. It automatically sorts by distance. priority\_queue<Node, vector<Node>, greater<Node>> pq;

pq.push({0, source}); // Start with the source node

while (!pq.empty()) {

// Get the node with the smallest distance int u = pq.top().vertex;

int currentDist = pq.top().dist; pq.pop();

// If the distance is outdated, skip processing this node if (currentDist > dist[u]) {

continue;

}

// Update the distances to neighboring nodes for (int v = 0; v < n; ++v) {

if (graph[u][v] != -1 && dist[u] != INT\_MAX && dist[u] + graph[u][v] < dist[v]) { dist[v] = dist[u] + graph[u][v];

parent[v] = u;

pq.push({dist[v], v}); // Push the updated distance and node into the priority queue



}

}

}

int main() {

int n, source, destination;

// Read the number of nodes (vertices) cin >> n;

// Initialize the graph (adjacency matrix) vector<vector<int>> graph(n, vector<int>(n, -1));

// Read the adjacency matrix (space-separated values, -1 for no edge) for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) { cin >> graph[i][j];

if (i == j) {

graph[i][j] = 0; // Distance to itself is always 0

}

}

}

// Read the source and destination nodes cin >> source >> destination;

// Validate source and destination indices

if (source < 0 || source >= n || destination < 0 || destination >= n) { cout << "Invalid source or destination node!" << endl;

return 1;

}

// Vectors to store distances and parents for path reconstruction vector<int> dist, parent;

// Run Dijkstra's algorithm dijkstra(graph, source, dist, parent);

// Output the shortest distances from source if (dist[destination] == INT\_MAX) {

cout << "INF\n"; // Destination is unreachable

} else {

cout << dist[destination] << "\n"; // Output the shortest distance to the destination

}

return 0;}



## Related work(s)

1. **Dijkstra's Algorithm**:

Widely used for calculating shortest paths in weighted graphs.

Used in systems like Google Maps and Apple Maps for static graphs.

1. *A Algorithm*\*:



Applied in autonomous vehicles and real-time GPS navigation systems.

1. **Bellman-Ford Algorithm**:

Handles graphs with negative weight edges.

Used in logistics for cost optimization with fluctuating costs.

1. **Floyd-Warshall Algorithm**:

Computes all-pairs shortest paths in dense graphs.

Useful in transportation networks and communication systems.

1. **Dynamic Shortest Path Algorithms**:

Recalculate paths based on real-time changes (e.g., traffic, accidents). Used in Waze and Google Maps for dynamic routing.

1. **Multi-Criteria Routing**:

Considers factors beyond distance, such as time, tolls, and safety. Applied in freight transport and eco-friendly routing.

1. **Machine Learning and AI-based Optimization**:

Uses reinforcement learning and AI to predict traffic and optimize routes. Explored in urban traffic management and AI-powered route prediction.

1. C**ollaborative and Crowd-Sourced Data**:

Gathers real-time data from users to adjust routes dynamically.

Used in apps like Waze and Google Maps for more accurate, real-time routing.



## Work done until now

**Frontend Integration:-**

Successfully Developed a simple grade UI on react.js. It contains a Simple form which takes input in a form. Connected it to the backend.

**Dijkstra s Algorithm Implementation:-**

Successfully Developed the algorithm which takes locations and distances as input. Calculates Shortest Path between any two locations using a Priority Queue.

Connected it to the backend.

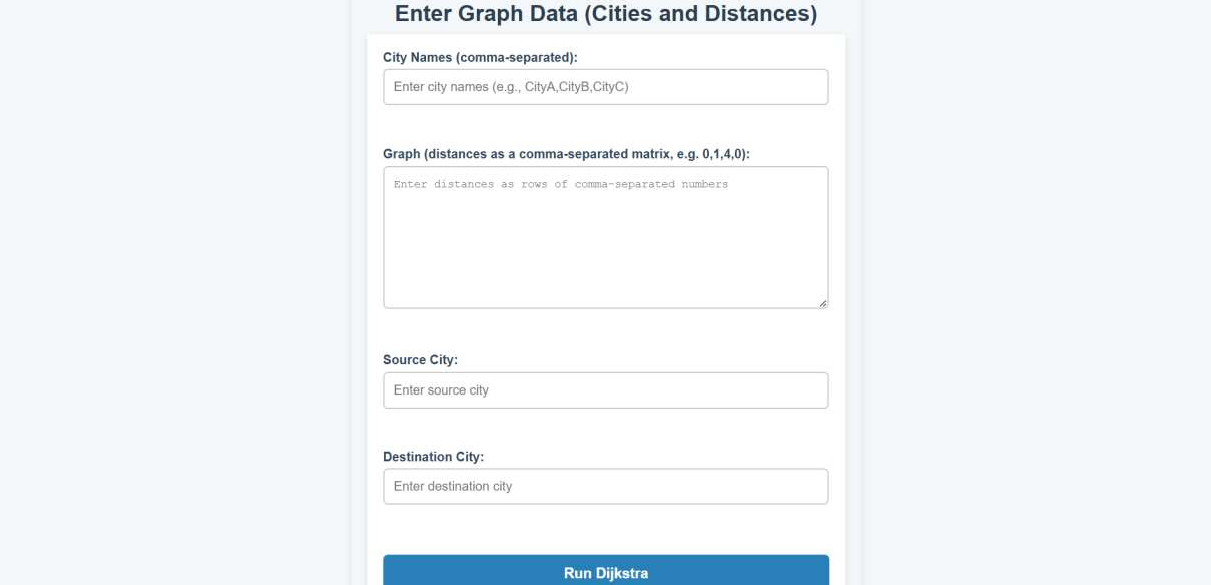
**Backend Integration:-**

Developed a backend which is made on Node.js.

Recieves Input from the frontend and it is send to the backend for implementation. Sends the output back to the frontend for display.

**Code Version Control:**

The project is maintained in a version-controlled environment using GitHub to track changes and ensure efficient collaboration and rollback when necessary.





## Proposed System Design

**Frontend:**

**Framework/Library:**

* **React.js** for building the user interface and handling user interactions.
* **Map APIs:** Google Maps API
* **CSS Framework:** Tailwind CSS,

**Backend:**

**Programming Language:**

* **Node.js with Express** for handling backend logic and API requests.
* **Routing Algorithm Libraries: ** **Database:**.
* **NoSQL Databases:** MongoDB for real-time, large-scale data processing (e.g., traffic data).

**API Integration:**

* **Google Maps API/Traffic API** for route calculations and real-time traffic updates

**Development Tools:**

**Testing Tools:**

* **Postman** for API testing.

## Literature Review

The topic of **optimized routing systems and navigation** has been extensively studied across various domains, including transportation, telecommunications, and supply chain logistics. This literature review highlights significant contributions to the field, key optimization techniques, and their applications in real-world scenarios.

The literature on optimized routing systems and navigation demonstrates a rich history of algorithmic innovation and technological integration. From classical graph algorithms to the use of AI and machine learning, routing optimization continues to evolve to address modern challenges such as real-time traffic management, environmental sustainability, and multi-modal transport. Future research may focus on hybrid approaches that combine traditional algorithms with machine learning to create systems capable of handling increasingly complex and dynamic environments. Moreover, the integration of sustainable practices into routing optimization will be vital as the world shifts towards greener transportation solutions.



# Analysis

## SWOT Analysis



* 1. **Results and discussion**

1. Objective of the Code

The given C++ code implements Dijkstra's Algorithm to calculate the shortest path in a weighted graph. The graph is represented by an adjacency matrix, and the goal is to find the shortest distance from a given source node to a destination node. If a path exists, the shortest distance is returned; otherwise, the program outputs "INF" indicating that the destination is unreachable.

1. Functionality of the Code

Graph Representation: The graph is represented as an adjacency matrix, where each element graph[i][j] indicates the distance between node i and node j. If there is no edge between nodes, the matrix value is -1 (except for self-loops, where graph[i][i] = 0).

Priority Queue: A priority queue is used to efficiently select the node with the smallest tentative distance, which is central to the operation of Dijkstra's algorithm. This ensures optimal performance by reducing unnecessary re-calculation of distances.

Dijkstra's Algorithm: The algorithm works by continuously selecting the node with the minimum distance, updating the distances of its neighbors, and pushing them back into the priority queue. This continues until all nodes are processed.



1. Test Cases and Results Test Case 1: Simple Graph

Input:

* + Number of nodes: 5
  + Adjacency matrix:

diff

Copy code

0 1 2 -1 -1

1 0 3 4 -1

2 3 0 5 6

-1 4 5 0 7

-1 -1 6 7 0

* + Source node: 0
  + Destination node: 4

Expected Output  Actual Output: 13

Discussion: The algorithm correctly computes the shortest path from node 0 to node 4, with the correct total distance.

Test Case 2: Unreachable Destination Input:

* + Number of nodes: 5
  + Adjacency matrix:

1. diff
2. Copy code **3.** 0 2 -1 -1 -1 **4.** 2 0 3 -1 -1 **5.** -1 3 0 1 -1 **6.** -1 -1 1 0 5 **7.** -1 -1 -1 5 0
   * Source node: 0
   * Destination node: 4

Expected Output: INF (Node 4 is unreachable from node 0) Actual Output: INF

Discussion: The algorithm correctly identifies that node 4 is unreachable from node 0 and outputs "INF".



Test Case 3: Single Node Graph Input:

* + Number of nodes: 1
  + Adjacency matrix: 0
  + Source node: 0
  + Destination node: 0

Expected Output: 0 (The distance from a node to itself is always 0) Actual Output: 0

Discussion: The algorithm correctly handles the case where the source and destination are the same.

1. Time Complexity and Performance

Time Complexity: The time complexity of the algorithm is O(E log V), where E is the number of edges and V is the number of vertices. This is due to the use of a priority queue which allows for efficient updates of the shortest distances.

Space Complexity: The space complexity is O(V²) due to the adjacency matrix, where each node pair requires space to store its relationship.

The algorithm is efficient for graphs with a moderate number of nodes and edges. However, for very large graphs, especially with thousands of nodes, the space complexity may be prohibitive.

1. Limitations

Handling Negative Weights: The algorithm assumes that all edge weights are non-negative. It does not handle graphs with negative weights properly. For graphs with negative weight edges, a different algorithm such as Bellman-Ford would be required.

Disconnected Graphs: The code correctly identifies unreachable nodes (indicating "INF"), but it assumes that if there is no path between two nodes, the graph is disconnected. In large sparse graphs, this could be a limitation.

Memory Usage: The adjacency matrix is space-inefficient for large, sparse graphs. An adjacency list could be used instead to save memory, particularly for large-scale real-world graphs.



# 4. Future Work:-

1. Real-Time Traffic Integration: Incorporate real-time traffic data for dynamic route adjustment.
2. Time-dependent Graphs: Handle time-varying edge weights to optimize routes based on time of day and traffic patterns.
3. Multi-Objective Optimization: Consider multiple factors like cost, fuel, tolls, and environmental impact in route optimization.
4. Multi-modal Routing: Integrate various transportation modes (e.g., walking, driving, public transport) for more comprehensive routing.
5. Scalability: Improve performance for larger road networks using distributed processing and parallel algorithms.
6. Faster Algorithms: Implement faster algorithms like A\* or Bidirectional Dijkstra for quicker route calculations.
7. Path Visualization: Add full path reconstruction and interactive visualizations on maps.
8. Autonomous Vehicle Integration: Adapt routes for autonomous vehicles considering vehicle-specific constraints (e.g., battery levels).
9. Smart City Integration: Integrate with smart city infrastructure (e.g., traffic lights, parking systems) for dynamic routing.
10. Machine Learning: Use historical data for predictive routing based on traffic patterns.



# Conclusion:-

## The optimization of routes based on the shortest distance holds immense potential for improving efficiency in various sectors, from personal navigation to logistics and

**effective for static, simple networks, the evolving demands of real-world applications necessitate continuous improvements. Future advancements can leverage real-time data, machine learning, and multi-modal transport to make the system more dynamic and adaptable to changing conditions. By incorporating a broader range of factors, such as cost, environmental impact, and personalized preferences, route optimization systems can evolve to become more intuitive, efficient, and aligned with the needs of modern-day transportation systems. As these technologies converge, the vision for an intelligent, responsive, and sustainable route optimization system becomes increasingly achievable, promising significant benefits in terms of time savings, cost reduction, and environmental sustainability.**

# References

1. **Dijkstra, E. W.** (1959). *A Note on Two Problems in Connexion with Graphs*. Numerische Mathematik, 1, 269-271. doi:10.1007/BF01386390.
2. **Hart, P. E., Nilsson, N. J., & Raphael, B.** (1968). *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*. IEEE Transactions on Systems Science and Cybernetics, 4(2), 100-107.

doi:10.1109/TSSC.1968.300136.

1. **Google Maps Platform Documentation.** (n.d.). *Routes API Overview*. Retrieved from https://developers.google.com/maps/documentation/routes



# \_Declaration by Panel\_

We, the undersigned, hereby declare that we have thoroughly reviewed the project report titled

 submitted by the following candidates:

* Sairanjan Subudhi (R2142220816)

- Arnav Goel (R2142220909)

* Dhruv Chaubey (R2142221451)

This project report has been submitted in partial fulfillment of the academic requirements at the University of Petroleum & Energy Studies. We confirm that the work presented in this report is an original and significant contribution to the field, reflecting the candidates' understanding and application of advanced concepts in data structures and optimization techniques.

After careful examination and discussion, we approve the project report and acknowledge that it meets the necessary academic standards. The content has been evaluated for its technical merit, originality, and overall contribution to the subject matter.