

Title:

Shortest Path Finder Using Dijkstra's Algorithm (Varanasi Area)

1. Introduction

In today's world of smart cities and intelligent transportation systems, efficient route planning plays a crucial role in minimizing travel time and resource consumption. This project implements a **Shortest Path Finder** using **Dijkstra's Algorithm**, applied to a sample map of the **Varanasi region** and nearby cities such as **Sarnath, Mirzapur, Prayagraj, Jaunpur, Ghazipur, and Bhadohi**.

The program utilizes a **graph-based model** to represent cities and their road distances. The system is built with **Python**, **Tkinter GUI**, and **NetworkX** for graph visualization. Users can interactively select a **starting city** and a **target city**, and the system visually displays the **shortest route** between them with the total distance.

2. Objective

- To develop a **graphical interface** that demonstrates **Dijkstra's shortest path algorithm** in real-world geographical terms (Varanasi area).
 - To provide a **clear, visual understanding** of how Dijkstra's algorithm works in computing the optimal path.
 - To demonstrate the **integration of algorithms with GUI programming** and data visualization using **Tkinter and Matplotlib**.
 - To enable users (students, travelers, or learners) to quickly find the **most efficient route** between two cities.
-

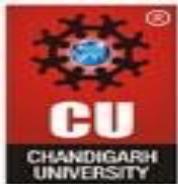
3. Problem Statement

Finding the shortest path between two cities manually can be time-consuming, especially when multiple routes and intersections are involved.

A computational solution is required to automate this process efficiently.

The challenge is to:

- Represent cities and distances using a **graph structure**.
- Implement a reliable algorithm (Dijkstra) to compute the shortest distance.
- Display results in an **intuitive graphical format**, not just as plain text output.



4. Existing System

Earlier, shortest path calculations were done using manual distance tables or static Google Maps lookups. Existing approaches lack:

- Algorithmic demonstration for educational purposes.
- Localized map customizability (like smaller regions).
- Integration into a **lightweight, offline tool** for quick simulation and learning.

5. Proposed System

The proposed system offers an interactive GUI that:

1. Accepts **user-selected cities** as source and destination.
2. Applies **Dijkstra's algorithm** to calculate the minimum travel distance.
3. Visually highlights the computed **shortest path** on a **graph plot**.
4. Displays the **total distance** and route sequence in a popup message.

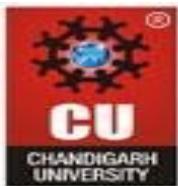
This makes the system useful for both **educational visualization** and **basic navigation simulation**.

6. Technologies Used

Component	Technology
Programming Language	Python 3
GUI Framework	Tkinter
Graph Library	NetworkX
Visualization Library	Matplotlib
Data Structure	Weighted Graph (Dictionary)
Algorithm	Dijkstra's Algorithm
IDE Used	VS Code / PyCharm / IDLE

7. Algorithm Used – Dijkstra's Algorithm

Overview



Dijkstra's algorithm finds the shortest path between nodes in a graph where all edge weights are non-negative. It maintains a set of nodes whose shortest distance from the source is known and expands outward to find the next nearest node.

Steps:

1. Assign a tentative distance value to every node: 0 for the start node, infinity for all others.
2. Set the start node as the current node.
3. For the current node, consider all unvisited neighbors and calculate their tentative distances.
4. If the new distance is smaller than the previously recorded one, update it.
5. Mark the current node as visited and remove it from the unvisited set.
6. Repeat steps 3–5 until all nodes have been visited or the target node is reached.

Time Complexity:

- Using a priority queue (heap): $O((V + E) \log V)$
where V = number of vertices and E = number of edges.

8. System Architecture

Modules:

1. **Graph Data Module:**
Defines the nodes (cities) and weighted edges (distances).
2. **Algorithm Module:**
Implements Dijkstra's algorithm for distance computation and path reconstruction.
3. **GUI Module:**
 - City selection (source/target)
 - Buttons for “Find Path” and “Reset”
 - Result display with message boxes
4. **Visualization Module:**
Draws the graph and highlights the computed shortest path in **red** with **orange nodes**.

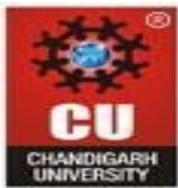
9. System Design

Input Design:

- Dropdown menus for Start and Target cities.

Process Design:

- Executes Dijkstra's algorithm on button click.



Output Design:

- Displays:
 - The shortest route (Varanasi → Sarnath → Jaunpur → Ghazipur)
 - Total distance (e.g., 145 km)
 - Graph visualization with highlighted route.

10. Implementation Details

Key Libraries:

```
import tkinter as tk
from tkinter import ttk, messagebox
import networkx as nx
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import heapq
```

Core Functions:

- `dijkstra(graph, start)`: Computes shortest distances and predecessors.
- `get_path(prev, start, target)`: Reconstructs route from the predecessor map.
- `DijkstraGUI` class: Handles GUI creation, drawing, and user interaction.

Visualization:

- Nodes: Represent cities (e.g., Varanasi, Sarnath)
- Edges: Represent distances (weights).
- Highlighted Path: Shown in red edges and orange nodes.

11. Sample Output

When user selects:

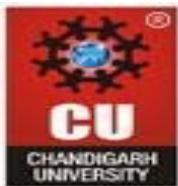
Start: Varanasi

Target: Ghazipur

Output:

Shortest path: Varanasi → Jaunpur → Ghazipur
Distance: 125 km

Graph visualization shows:



- Path highlighted in **red edges**.
 - Nodes on the path in **orange**.
-

12. Advantages

- Simple, interactive, and educational.
 - Demonstrates algorithmic working visually.
 - Uses open-source Python libraries.
 - Can be extended to real maps (with GPS data).
 - Offline and lightweight.
-

13. Limitations

- Fixed dataset (Varanasi area only).
 - Not integrated with real-time GPS or live map APIs.
 - Only supports undirected graphs with static weights.
-

14. Future Enhancements

- Integrate **Google Maps API** for real-world coordinates.
 - Add **A* algorithm** for faster performance.
 - Include **real-time traffic data** or variable edge weights.
 - Expand dataset to include **entire UP or India regions**.
 - Provide **distance units and estimated travel time** dynamically.
-

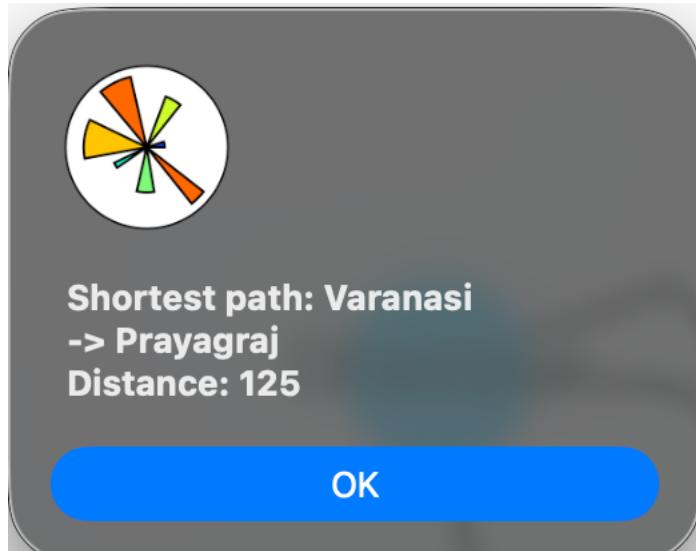
15. Conclusion

This project successfully demonstrates how Dijkstra's Algorithm can be applied to real-world route optimization problems using Python.

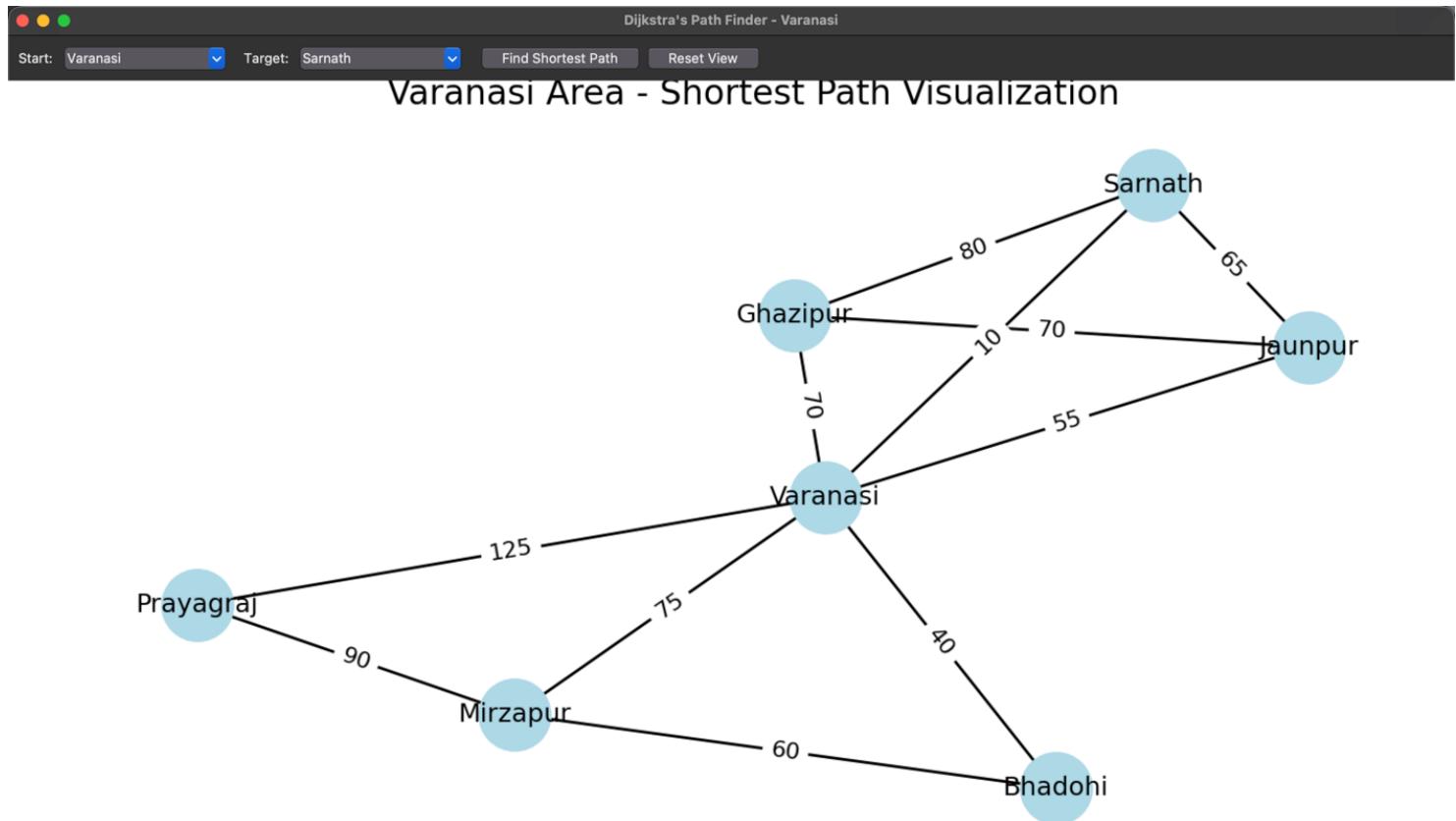
It provides an intuitive GUI and visualization that help users understand graph traversal and shortest-path computation.

The project effectively bridges **theoretical algorithms** and **practical applications** in urban routing, logistics, and computer networks.

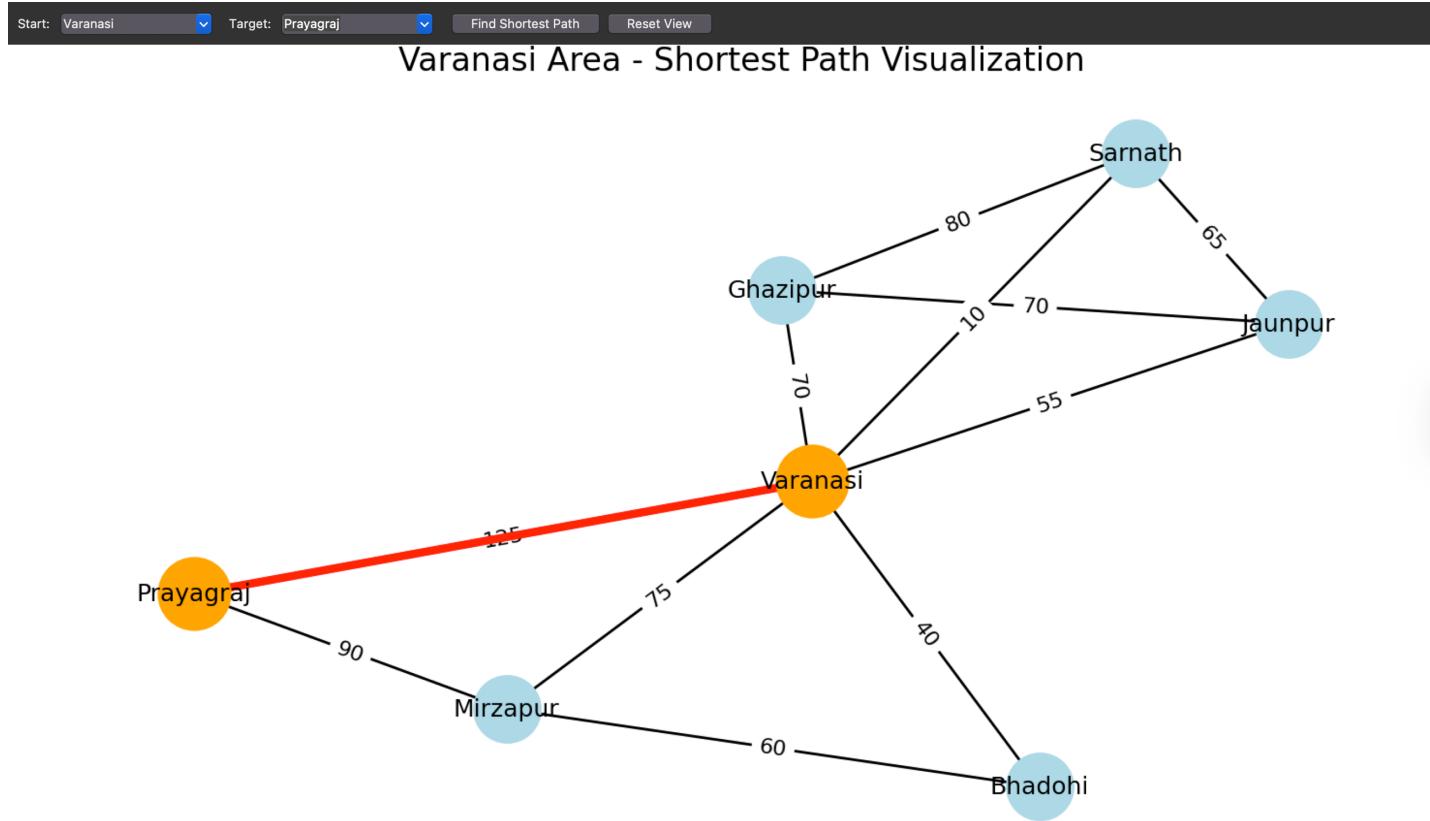
16. Screenshots



Main Page :



Output Page:



16. References

- E. W. Dijkstra, "A note on two problems in connexion with graphs", *Numerische Mathematik*, 1959.
- Python Official Documentation: <https://docs.python.org/3/>
- NetworkX Documentation: <https://networkx.org/documentation/stable/>
- Matplotlib Documentation: <https://matplotlib.org/stable/>
- Tkinter Official Guide: <https://docs.python.org/3/library/tkinter.html>