Ramdeobaba University, Nagpur
Department of Computer Science and Engineering
Session: 2025-26

Subject: Design and Analysis of Algorithms (DAA) Lab Project

III Semester

LAB PROJECT REPORT

Group Members with Roll Number and Section:

Name Section Batch Roll No.

Prachi Sharnagat A2 B2 20
Mansi Gupta A2 B2 32
Tejal Bhagat A2 B4 63

GITHUB link of Project repo and deployment link (if website): GitHub

Repository: https://github.com/Prachi-Sharnagat/Dijkstra-Solver-website

Deployment Link: dijkstra-solver.netlify.app/

TITLE: Shortest Path Visualizer using Dijkstra's Algorithm

Objectives:

1. To visualize how Dijkstra's algorithm finds the shortest path between nodes.
2. To provide an interactive UI for creating nodes and edges.
3. To calculate and display path distances dynamically.
4. To enhance understanding of graph theory through visualization. 5. To

implement animations and smooth transitions for a better user experience.

Introduction:

Graphs are an important concept in computer science used to represent connections between objects, such as cities connected by roads or networks of computers.
This project, Shortest Path Visualizer, is a web-based application that allows users to:

- Create nodes (vertices) on a canvas.
- Connect nodes with weighted edges.
- Run Dijkstra's Algorithm to find the shortest path from a selected source node to all other nodes.

The visualization helps students understand how Dijkstra's algorithm explores nodes and updates distances in real time, using animations and color transitions for clarity.

Algorithms/Technique used:

Algorithm Name: Dijkstra's Shortest Path Algorithm

Algorithm Steps:

1. Set all nodes' distances to Infinity except the source node, which is set to 0.
2. Mark all nodes as unvisited.
3. Choose the unvisited node with the smallest distance.
4. For each neighbor of this node, calculate the tentative distance and update it if smaller.
5. Mark the current node as visited.
6. Repeat steps 3–5 until all nodes are visited.

Explanation:

Dijkstra's algorithm works by exploring the nearest unvisited node, updating paths to its neighbors, and repeating this until all shortest paths are found.
It ensures the minimum total cost path from the source to every other node in a weighted graph (with no negative weights).

Time Complexity and its explanation:

- **Time Complexity:** $O(V^2)$
  (where V = number of vertices)

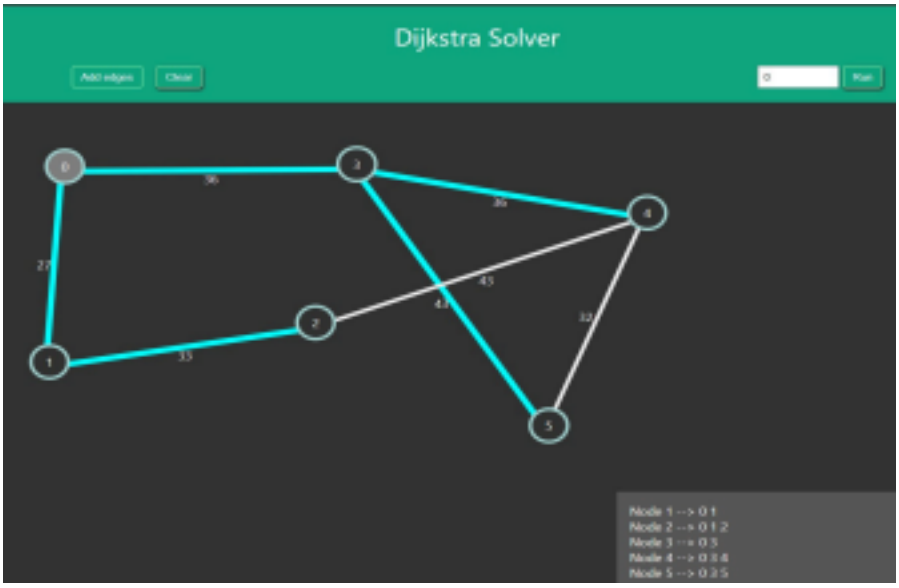
Explanation:

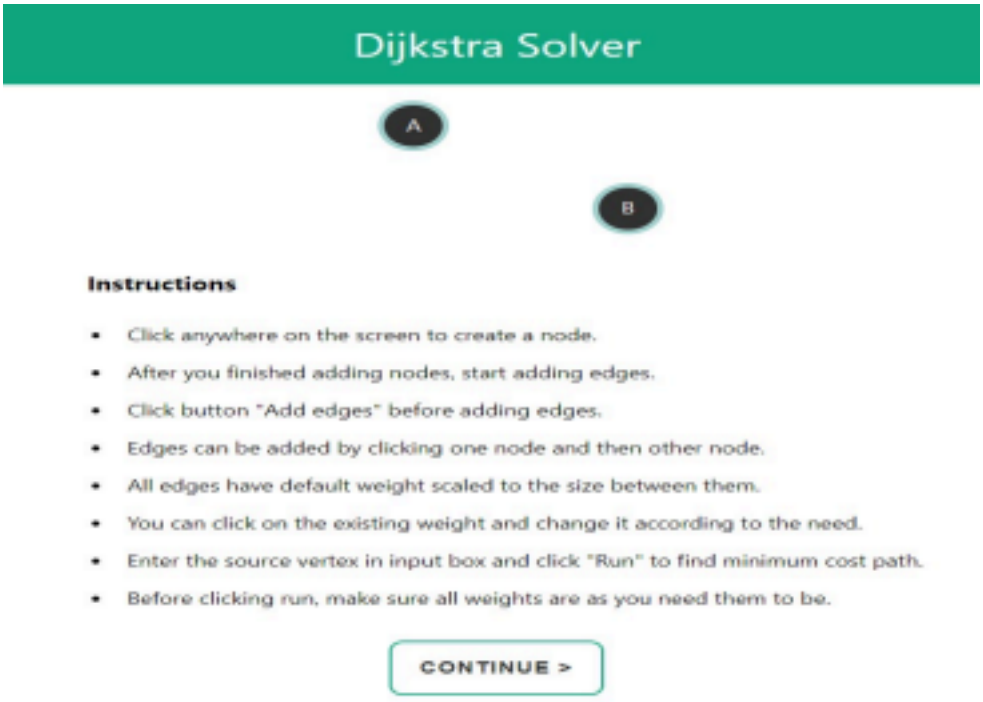
In the current implementation (without using a priority queue):

- We search the minimum distance node linearly → $O(V)$
- We repeat this for all vertices → $O(V^2)$

If implemented using a priority queue (min-heap), the complexity can be improved to **O(E log V)**.

Results:





1. **Main Interface:**
   Shows the canvas where nodes are created.

2. **After Connecting Edges:**
   Displays edges with editable weights.

3. **Running Dijkstra's Algorithm:**
   Highlights the shortest path using animations.

4. **Result Display Section:**
   Shows the computed shortest path from source to all nodes.

Discussion:

The project successfully visualizes the working of Dijkstra's algorithm.
Users can:

- Add nodes dynamically,
- Connect them with weighted edges,
- Edit weights, and
- View the path-finding process step-by-step.

The visualization is enhanced by **GSAP animations** and clear UI design, making learning interactive and engaging.

.

Conclusion and future scope:

Conclusion:

This project demonstrates how algorithms can be visualized interactively using web technologies.
It improves understanding of graph traversal and shortest path algorithms. The use of colors, animations, and dynamic UI provides both educational and visual appeal.

Future Scope:

1. Add other algorithms like Bellman-Ford, Floyd–Warshall, and A* search.
2. Implement directed and weighted graphs separately.
3. Allow saving and loading graphs.
4. Add dark/light theme toggle for better UX.
5. Use a priority queue to optimize the time complexity.