Deccan Education Society's
Fergusson College (Autonomous), Pune
Department of Computer Science

A

Report
on

"Interactive Visualization of Minimum Spanning Tree Algorithms using Kruskal's and Prim's Methods"

In partial fulfillment of Post Graduate course

in

M.Sc. Computer Science – I

(Semester -I)
2025-2026

CSC-520 Practical I

SUBMITTED BY

Abhishek Jaiswar - 256201
Yogiraj Bhilare – 256228
Tejas Patare - 256259

Index

1. Case Study Description

✓ Project Overview

The MST Visualizer is an interactive web-based application designed to visualize and understand two fundamental graph algorithms: **Kruskal's Algorithm** and **Prim's Algorithm**. The application provides both automated graph generation and manual graph creation capabilities, making it an excellent educational tool for learning about Minimum Spanning Trees.

✓ Algorithm Details

1.1 Kruskal's Algorithm

Concept: Kruskal's algorithm is a greedy algorithm that finds the Minimum Spanning Tree by selecting edges in ascending order of their weights, while avoiding cycles.

Implementation Approach:

- **Data Structure:** Union-Find (Disjoint Set Union) with path compression
- **Time Complexity:** O(E log E) where E is the number of edges
- **Space Complexity:** O(V) where V is the number of vertices

Algorithm Steps:

1. Sort all edges in non-decreasing order of their weights
2. Initialize a parent array for Union-Find operations
3. For each edge in sorted order:
   - Check if adding this edge creates a cycle using find() operation
   - If no cycle is formed, add edge to MST using union() operation
   - If cycle is formed, skip the edge
4. Continue until V-1 edges are added to the MST

Key Functions:

- find(x): Finds the root parent of vertex x with path compression
- union(a, b): Merges two sets if they have different parents
- Returns true if merge successful (no cycle), false otherwise

Visual Feedback:

- **Green edges**: Added to MST
- **Red edges**: Skipped (would create cycle)
- Real-time cost calculation and timing information

1.2 Prim's Algorithm

Concept: Prim's algorithm builds the MST by starting from a source vertex and greedily adding the minimum weight edge that connects a visited vertex to an unvisited vertex.

Implementation Approach:

- **Data Structure:** Set for visited vertices, array for available edges
- **Time Complexity:** O(E log V) with efficient implementation
- **Space Complexity:** O(V + E)

Algorithm Steps:

1. Start with a user-specified source vertex
2. Mark source vertex as visited
3. Add all edges connected to the source to available edges list
4. Repeat until all vertices are visited:
    - Select edge with minimum weight from available edges
    - If both endpoints are visited, skip (would create cycle)
    - Otherwise, add edge to MST
    - Mark new vertex as visited
    - Add all edges from new vertex to unvisited vertices

Key Functions:

- `addEdges(u)`: Adds all edges from vertex u to unvisited neighbors
- Dynamic sorting of available edges by weight
- Visited set tracking to prevent cycles

Visual Feedback:

- **Green edges**: Added to MST
- **Red edges**: Skipped (both endpoints already visited)
- Step-by-step progression with cost tracking

✓ UI Design Details

2.1 Layout Structure

The application features a clean, modern interface divided into four main sections:

Header Section:

- Application title with gradient blue background
- Clear, professional typography

Control Panel (Left Sidebar):

- Graph configuration controls
- Algorithm selection dropdown
- Action buttons for graph generation and algorithm execution
- Source vertex input (conditional display for Prim's algorithm)

Visualization Canvas (Center):

- 500×500px SVG canvas for graph rendering
- Nodes arranged in circular layout for optimal visibility
- Interactive drag-and-drop functionality for edge creation
- Dynamic edge coloring based on algorithm state

Information Panel (Right Sidebar):

- Real-time step-by-step execution log
- Algorithm results display
- Total MST cost calculation
- Execution time tracking
-

2.2 Visual Design Elements

Color Scheme:

- **Primary Blue (#0277bd)**: Nodes and UI accents
- **Light Blue (#90caf9)**: Default edge color
- **Green (#66bb6a)**: MST edges (accepted)
- **Red (#ef5350)**: Rejected edges (cycles)
- **Yellow (#ffd54f)**: Interactive drag line
- **White backgrounds**: Clean, professional appearance

Node Design:

- Circular nodes with 18px radius
- White stroke border for visibility
- Centered letter labels (A, B, C, etc.)
- Pointer cursor in interactive mode

Edge Design:

- Lines with 3px stroke width (5px when in MST)
- Weight labels centered on edges
- White rounded rectangle backgrounds for readability
- Proper z-index management (edges behind nodes)

2.3 Interactive Features

Graph Creation Modes:

1. **Random Graph Generation:**
    - Node count selection (3-10 vertices)
    - Density control (Sparse, Medium, Dense)
    - Automatic random weight assignment (1-20)
    - Ensures graph connectivity
2. **Interactive Manual Mode:**
    - Click and drag between nodes to create edges
    - Modal popup for weight input
    - Validation for positive integer weights
    - Duplicate edge prevention
    - Visual feedback with dashed yellow line during drag

Algorithm Execution:

- Start/Pause toggle button
- Step-by-step execution with 1-second delay
- Real-time update of visualization and logs
- Performance timing (milliseconds)
- Auto-scroll for step logs

User Experience Enhancements:

- Responsive button states
- Clear error messages
- Input validation
- Smooth animations
- Scrollable content areas
- Professional modal dialogs

2.4 Technical Implementation

Technology Stack:

- Pure HTML5, CSS3, and Vanilla JavaScript
- SVG for scalable vector graphics
- No external dependencies
- Client-side execution for instant feedback

Code Architecture:

- Modular function design
- Separate concerns (UI, algorithm logic, event handling)
- Global state management for animation control
- Event-driven interaction model

Key Technical Features:

- DOM manipulation for dynamic SVG generation
- Event listeners for interactive drag-and-drop
- setTimeout-based animation loop
- Union-Find with path compression optimization
- Efficient edge sorting and filtering

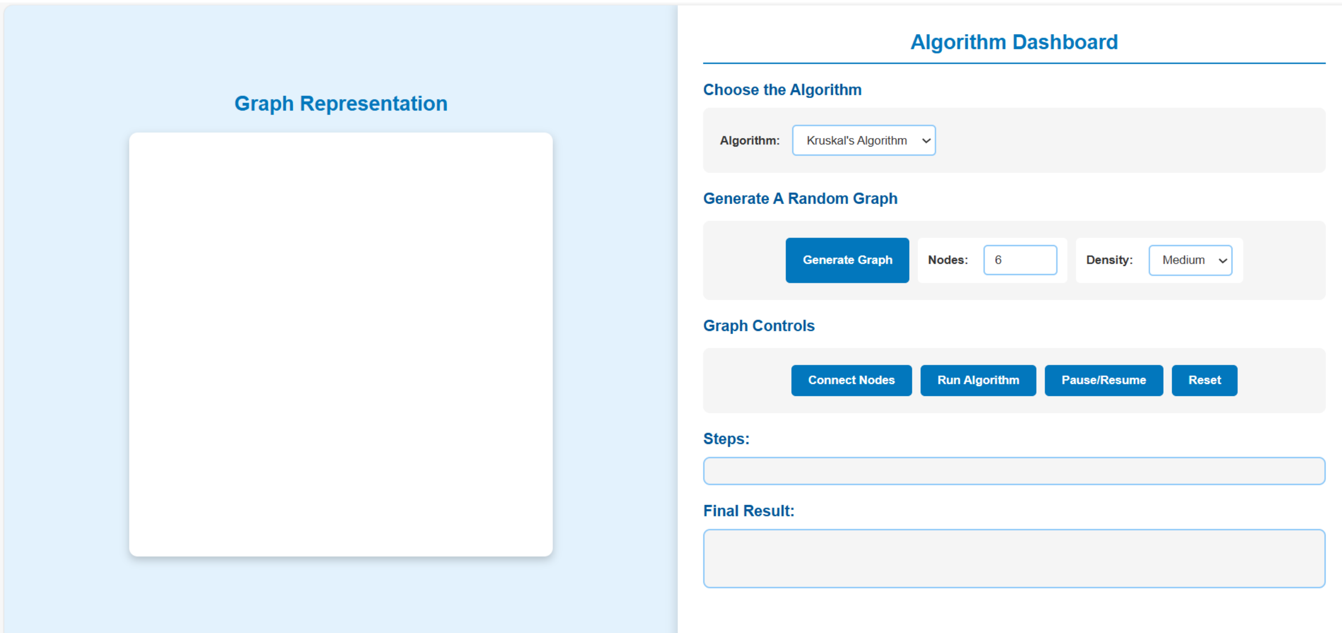
✓ Conclusion

The MST Visualizer successfully demonstrates both Kruskal's and Prim's algorithms with an intuitive, interactive interface. The application serves as an effective educational tool for understanding graph algorithms, providing real-time visualization, step-by-step execution, and performance metrics.

2. Screenshots

2.1 Initial Interface

Main application interface showing control panel, visualization canvas, and information panel

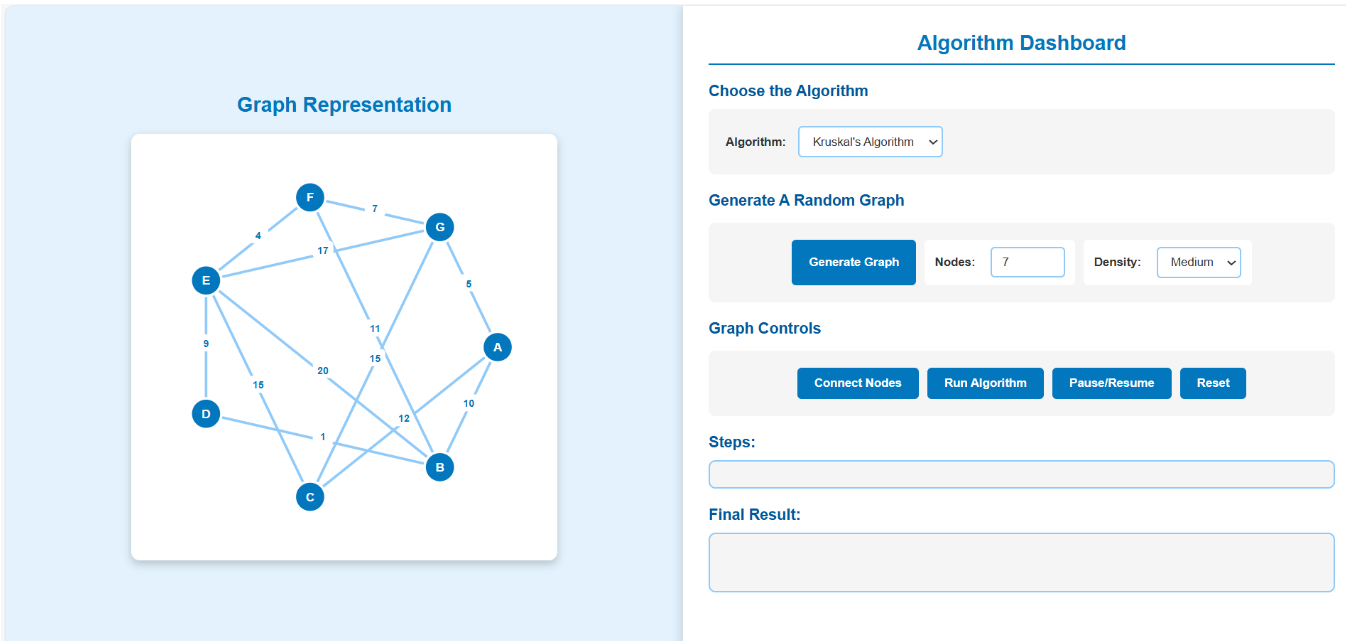


Features Visible:

- Clean layout with organized controls
- Empty SVG canvas ready for graph creation
- Algorithm selection dropdown
- Node count and density controls

2.2 Random Graph Generation

Generated graph with 7 nodes and medium density

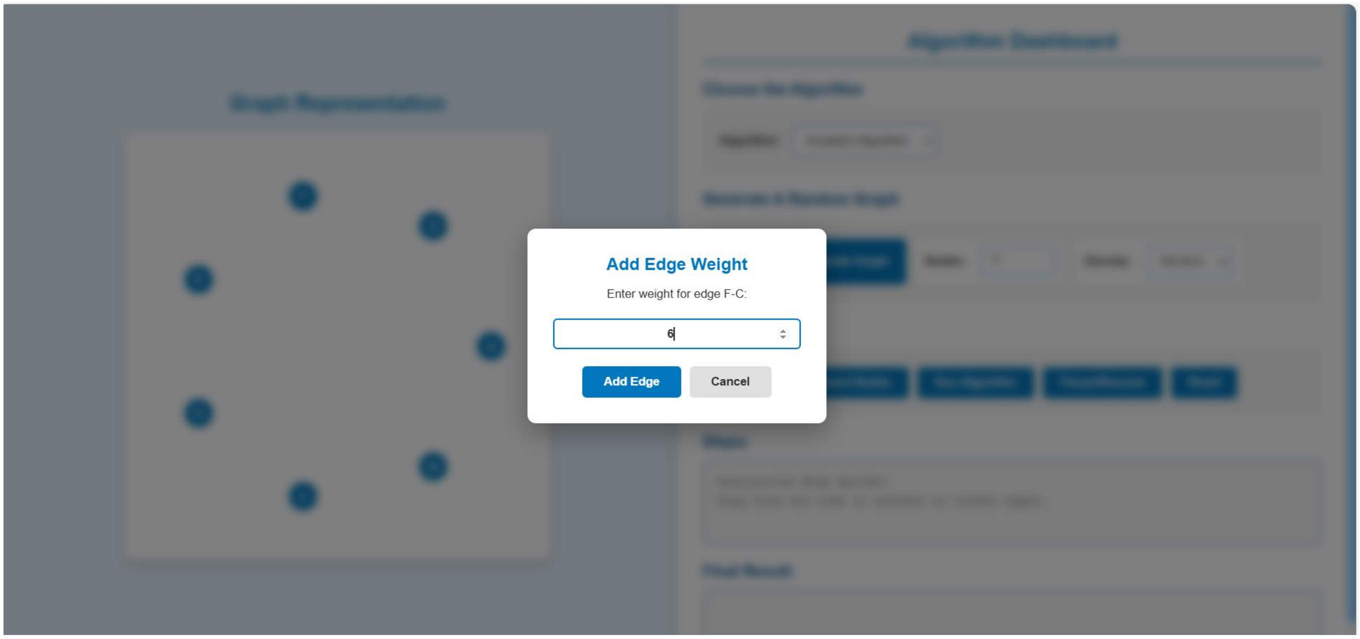


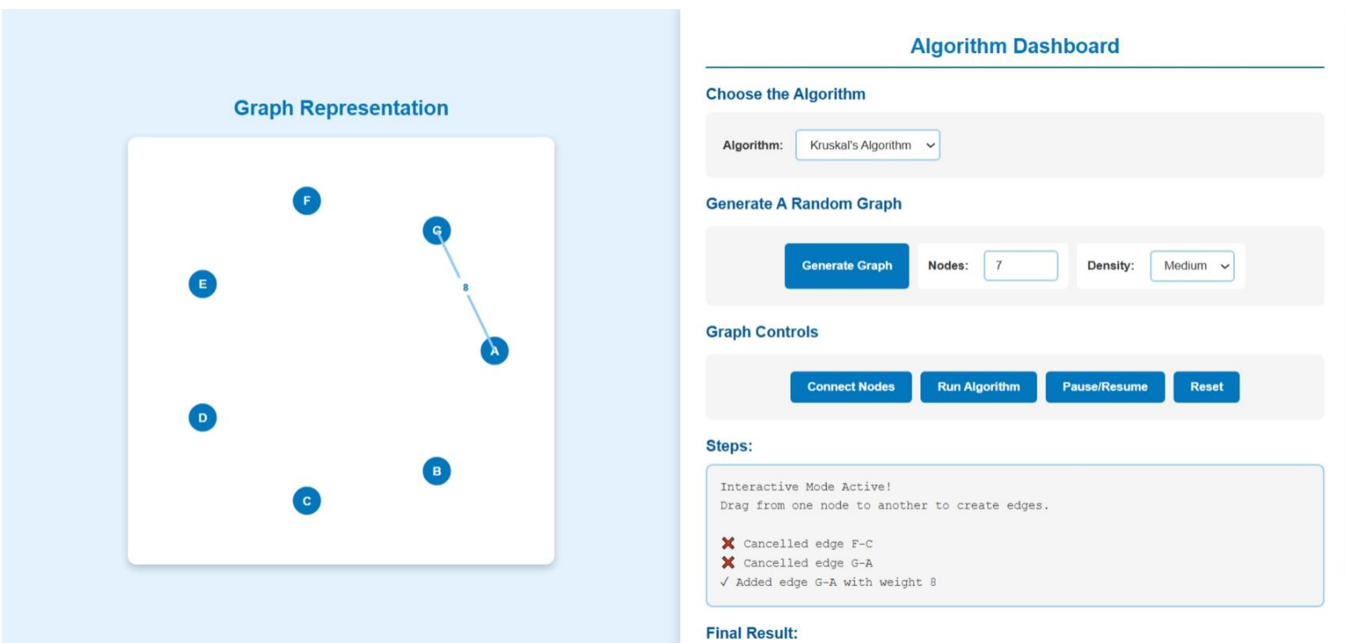
Elements Shown:

- Nodes arranged in circular pattern
- Multiple edges with weight labels
- Blue default edge coloring
- Letter-labeled vertices (A through G)

2.3 Interactive Mode

User creating edges manually with drag-and-drop



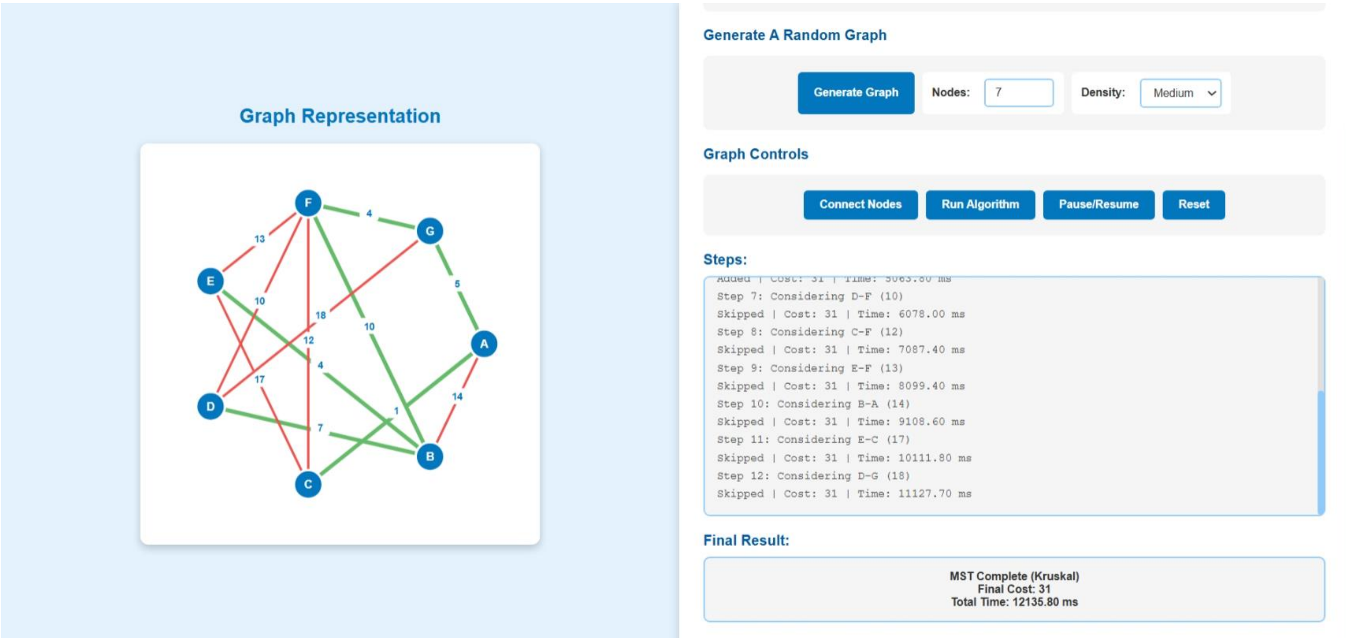


Interaction Display:

- Dashed yellow line during drag
- Weight input modal dialog
- Real-time feedback in steps panel
- Confirmation and cancellation options

2.4 Kruskal's Algorithm Execution

Mid-execution showing accepted and rejected edges

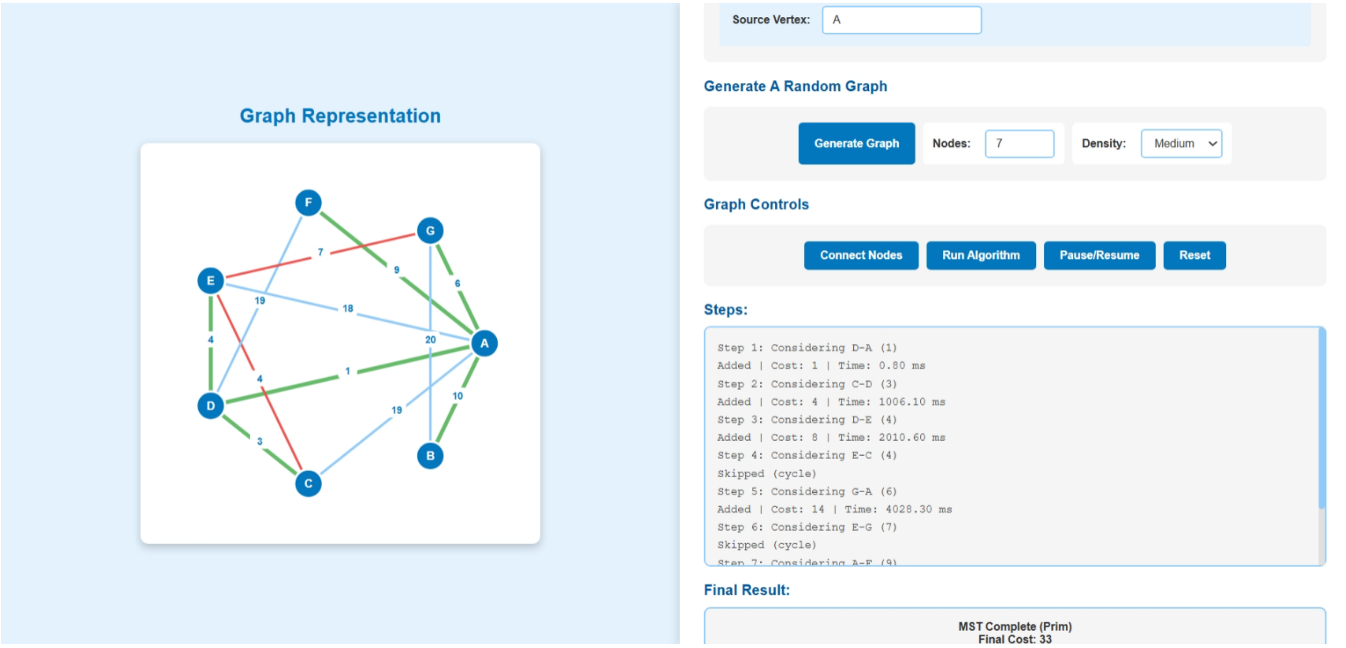


Algorithm Visualization:

- Green edges: Part of MST
- Red edges: Rejected (would create cycles)
- Step-by-step log with edge considerations
- Running cost calculation
- Timestamp for each step

2.5 Prim's Algorithm Execution

Prim's algorithm showing tree growth from source vertex



Visualization Features:

- Source vertex input field
- Progressive tree building
- Green edges showing MST formation
- Red edges showing rejected connections
- Visited vertex tracking

3. References

✓ Algorithm Resources

1. **Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C.** (2009) *Introduction to Algorithms* (3rd ed.) MIT Press
    - Chapter 23: Minimum Spanning Trees
    - Kruskal's Algorithm (Section 23.2)
    - Prim's Algorithm (Section 23.2)
2. **Sedgewick, R., & Wayne, K.** (2011) *Algorithms* (4th ed.) Addison-Wesley
    - Graph Algorithms Chapter
    - Union-Find Data Structure

✓ Web Development Resources

1. **MDN Web Docs - SVG Tutorial** Mozilla Developer Network https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial
    - SVG element creation and manipulation
    - Event handling in SVG
2. **W3C SVG Specification** World Wide Web Consortium https://www.w3.org/TR/SVG2/
    - SVG standards and best practices

✓ Algorithm Visualization Concepts

1. **VisuAlgo** https://visualgo.net/en/mst
    - Inspiration for interactive algorithm visualization
    - UI/UX patterns for educational tools
2. **GeeksforGeeks - Graph Algorithms** https://www.geeksforgeeks.org/
    - Kruskal's Algorithm implementation
    - Prim's Algorithm implementation
    - Union-Find optimization techniques

✓ Design Resources

Material Design Guidelines Google Material Design https://material.io/design

- Color palette selection
- Button and interaction design
- Modal dialog patter

Project completed as part of Algorithm Visualization Study Date: November 2025