# Kongu Engineering College KONGU ENGINEERING COLLEGE Kongu Engineering College (Autonomous)

(Autonomous)

Perundurai,Erode – 638060

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**FINDING NETWORK TOPOLOGY USING BRUTE FORCE WITH ADJACENCY MATRIX**

**A MICRO PROJECT REPORT**

**FOR**

**DESIGN AND ANALYSIS OF ALGORITHMS(22ITT31)**

**SUBMITTED BY**

**GOWTHAM M (23ITR049)**

** KONGU ENGINEERING COLLEGE**  (Autonomous)

Perundurai,Erode – 638060

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**FINDING NETWORK TOPOLOGY USING BRUTE FORCE WITH ADJACENCY MATRIX**

**A MICRO PROJECT REPORT**

**FOR**

**DESIGN AND ANALYSIS OF ALGORITHMS(22ITT31)**

**SUBMITTED BY**

# GOWTHAM M (23ITR049)

# Kongu Engineering College KONGU ENGINEERING COLLEGE Kongu Engineering College (Autonomous)

# (Autonomous)

Perundurai,Erode – 638060

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**BONAFIED CERTIFICATE**

|  |  |
| --- | --- |
| Name | : GOWTHAM M |
| Course Code | : 22ITT31 |
| Course Name | : DESIGN AND ANALYSIS OF ALGORITHMS |
| Semester | : IV |

Certified that this is a bonafied record of work for application project done by the above student for 22ITT31-DESIGN AND ANALYSIS OF ALGORITHMS during the academic year 2024-2025.

Submitted for the Viva Voice Examination held on \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Faculty Incharge Head of the Department

## ABSTRACT

## This project presents a user-friendly web-based application designed to identify the topology of a network using manual input of an adjacency matrix. Implemented in Python with the Flask framework, the application analyzes node connectivity patterns using a brute-force approach, with an overall time complexity of O(n²). The user inputs the number of nodes and a corresponding adjacency matrix, where each value (0 or 1) indicates the presence or absence of a direct connection between nodes.

## The application processes the matrix to calculate the degree of each node and compares the degree distribution against known patterns to detect common network topologies such as Star, Ring, Mesh, and Bus. It then provides a clear identification of the detected topology along with practical suggestions for potential improvements or upgrades, and displays the computational steps and complexity involved in the process.

## Designed for educational and analytical purposes, this tool aids students, network learners, and instructors in understanding basic network structures and the role of graph theory in topology identification. The project highlights the effectiveness of brute-force algorithms in small-scale applications and demonstrates the value of pairing logical analysis with accessible interfaces to support interactive learning in computer network design.

## TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO** | **TITLE** | **PAGE NO** |
|  | **ABSTRACT** | **ix** |
| **1.** | **INTRODUCTION** | 6 |
|  | 1.1 PURPOSE | 7 |
|  | 1.2 OBJECTIVE | 7 |
|  | 1.3 METHODOLOGY OVERVIEW | 8 |
| **2.** | **PROBLEM STATEMENT** | 9 |
| **3.** | **METHODOLGY** | 10 |
|  | 3.1 Input & Initialization | 10 |
|  | 3.2 **Divide & Compare** | 10 |
|  | 3.3 Recursive Detection | 10 |
|  | 3.4 Visualization & Output | 10 |
| **4.** | **IMPLEMENTATION** | 11 |
|  | 4.1 Input & Initialization | 11 |
|  | 4.2 Divide & Compare | 11 |
|  | 4.3 Recursive Detection | 12 |
|  | 4.4 Visualization & Output | 12 |
| **5.** | **RESULTS** | 15 |

## INTRODUCTION

In the world of computer networks and topology analysis, accurately identifying the structure of a network is a fundamental task that can influence design, performance, and scalability. This project focuses on developing a web-based application that enables users to input an adjacency matrix and instantly determine the underlying network topology, such as Star, Ring, Mesh, or Bus.

Using Python’s Flask framework for the backend and a user-friendly web interface for interaction, the application offers a simple and accessible platform for users to enter node connections, perform computations, and view results without requiring deep technical knowledge. The core logic employs a brute-force method that calculates node degrees and compares them against standard patterns, achieving accurate classification with a time complexity of O(n²).

This project not only demonstrates the practical application of graph theory and brute-force algorithms in solving real-world networking problems but also emphasizes the importance of clarity and usability by providing an intuitive interface. By integrating manual matrix input, automatic degree computation, and topology detection, the application serves as a valuable educational and analytical tool for students, network designers, and IT professionals.

* 1. **PURPOSE**

The primary purpose of this project is to create an easy-to-use application that identifies the network topology from a user-inputted adjacency matrix. It aims to provide a quick, accurate, and efficient way to analyze network structure based on node connections.

This tool helps students, network learners, and IT professionals to:

* Instantly detect common network topologies like Star, Ring, Mesh, or Bus.
* Understand the connectivity patterns through calculated node degrees.
* Gain insights into network design using an intuitive web interface.
* Demonstrate the practical application of graph theory and brute-force algorithms in real-world networking problems.

## OBJECTIVE

 To develop a user-friendly web application that accepts an adjacency matrix and efficiently identifies the network topology.

 To implement a brute-force algorithm with time complexity O(n²) to accurately analyze node degrees and detect topology patterns.

 To display the detected network topology along with suggestions for network design improvements.

 To provide a clear and intuitive interface that allows users to input network data easily and view results instantly.

 To demonstrate the application of fundamental programming concepts like input validation, graph theory, and web development using Python’s Flask framework.

 To enhance understanding of algorithmic analysis by applying brute-force methods to real-world network structure identification.

## METHODOLOGY OVERVIEW

## The project employs a systematic approach to identify the network topology from a user-input adjacency matrix using a brute-force degree analysis algorithm. The methodology includes:

## Input Collection: The user inputs the network's adjacency matrix through a web form, entering the number of nodes and the corresponding connectivity matrix with rows representing connections between nodes.

## Data Validation and Parsing: The input is validated to ensure the matrix dimensions match the specified number of nodes, and each entry is a valid integer (0 or 1) representing connection status.

## Degree Calculation: The algorithm calculates the degree of each node by summing the entries in each row of the adjacency matrix, representing the number of direct connections per node.

## Topology Detection Algorithm: The algorithm analyzes the degree list to match known patterns of standard network topologies (Star, Ring, Mesh, Bus). This brute-force method checks conditions on the degrees, ensuring a time complexity of O(n²), where n is the number of nodes.

## Result Generation and Suggestions: Based on the detected topology, the application generates a result summary and provides recommendations for network design improvements or validation.

## User Interface: The project uses Python’s Flask framework to build a simple and intuitive web interface that allows users to input data, submit for analysis, and instantly view detailed results and steps followed.

## 2. PROBLEM STATEMENT

In network management and design, accurately identifying the underlying network topology from connection data is crucial for optimizing performance, troubleshooting, and planning upgrades. Manual analysis of network connections can be complex and error-prone, especially for larger networks with many nodes.

This project aims to develop a user-friendly software application that efficiently determines the network topology from a given adjacency matrix representing node connections. The solution should handle input validation, perform topology detection using an algorithm with reasonable time complexity, and display the results clearly along with relevant suggestions. The application must provide an intuitive web interface for ease of use by students, network engineers, and IT professionals.

**3.METHODOLOGY:**

**Methodology (Brute Force Approach):**

1. **Input Collection:**  
   The system accepts the adjacency matrix of the network as input, with each row representing connections of a node to others.
2. **Input Validation:**  
   Validate the input to ensure it is a square matrix with numeric entries (0 or 1), and the size matches the declared number of nodes.
3. **Degree Calculation (Brute Force):**  
   For each node (row in the matrix), sum all entries to calculate its degree, representing the number of connections.
4. **Degree Comparison Process:**  
   Analyze the degree list by checking patterns characteristic of standard topologies:

* Star: one node connected to all others (degree n-1), others connected to only one node (degree 1).
* Ring: all nodes have degree 2.
* Mesh: all nodes connected to every other node (degree n-1).
* Bus: two nodes with degree 1, others with degree 2.

1. **Topology Identification:**  
   Match the degrees with known patterns to identify the network topology.
2. **Output Display:**  
   Display the detected topology, relevant suggestions for network design, the total number of nodes, and the time complexity of the brute force approach (O(n²)).

**4.IMPLEMENTATION :**

from flask import Flask, render\_template, request

app = Flask(\_\_name\_\_)

def calculate\_degrees(adj\_matrix):

    return [sum(row) for row in adj\_matrix]

def detect\_topology\_from\_degrees(degree):

    n = len(degree)

    if degree.count(1) == n - 1 and degree.count(n - 1) == 1:

        return "Star Topology", "O(n²) - Checks degree of all nodes in adjacency matrix."

    if all(d == 2 for d in degree):

        return "Ring Topology", "O(n²) - Checks all nodes have degree 2."

    if all(d == n - 1 for d in degree):

        return "Mesh Topology", "O(n²) - Confirms every node connects to every other node."

    if degree.count(1) == 2 and degree.count(2) == n - 2:

        return "Bus Topology", "O(n²) - Checks specific node degrees characteristic of bus."

    return "Unknown or Irregular Topology", "O(n²) - General check without matching known patterns."

@app.route("/", methods=["GET", "POST"])

def index():

    result = ""

    suggestion = ""

    time\_complexity = ""

    matrix\_input = ""

    nodes = ""

    steps = []

    if request.method == "POST":

        try:

            nodes = int(request.form["nodes"])

            matrix\_input = request.form["matrix"].strip().split("\n")

            adj\_matrix = []

            for row in matrix\_input:

                nums = list(map(int, row.strip().split()))

                if len(nums) != nodes:

                    result = f"Each row must contain {nodes} numbers."

                    return render\_template(

                        "index.html",

                        result=result,

                        suggestion=suggestion,

                        time\_complexity=time\_complexity,

                        matrix\_input=request.form["matrix"],

                        nodes=nodes,

                        steps=steps,

                    )

                adj\_matrix.append(nums)

            degrees = calculate\_degrees(adj\_matrix)

            topology, time\_complexity = detect\_topology\_from\_degrees(degrees)

            suggestions = {

                "Star Topology": "Suggestion: Upgrade to Mesh for better fault tolerance.",

                "Ring Topology": "Suggestion: Monitor for breaks; consider Mesh for stability.",

                "Mesh Topology": "Suggestion: Already optimal for reliability.",

                "Bus Topology": "Suggestion: Common in older systems; consider upgrading to Ring or Mesh.",

                "Unknown or Irregular Topology": "Suggestion: Network doesn't follow standard topology; review design.",

            }

            result = f"Detected Network Topology: {topology}"

            suggestion = suggestions[topology]

            steps = [

                "Step 1: Calculate degrees of each node from adjacency matrix.",

                "Step 2: Determine topology by analyzing degree patterns.",

            ]

        except Exception as e:

            result = f"Error: {str(e)}"

    return render\_template(

        "index.html",

        result=result,

        suggestion=suggestion,

        time\_complexity=time\_complexity,

        matrix\_input=matrix\_input,

        nodes=nodes,

        steps=steps,

    )

if \_\_name\_\_ == "\_\_main\_\_":

    app.run(debug=True)

<!DOCTYPE html>

<html lang="en">

<head>

  <link rel="stylesheet" href="{{ url\_for('static', filename='style.css') }}">

  <meta charset="UTF-8" />

  <title>Network Topology Detector</title>

  <style>

    body {

      font-family: Arial, sans-serif;

      background-color: #f8f9fa;

      margin: 40px;

      background: linear-gradient(to right, #4facfe, #00f2fe);

    }

    .container {

      background: white;

      padding: 20px 30px;

      border-radius: 10px;

      width: 500px;

      margin: auto;

      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

    }

    textarea,

    input[type="number"] {

      width: 100%;

      margin-top: 10px;

      margin-bottom: 20px;

      padding: 10px;

      font-size: 14px;

    }

    button {

      padding: 10px 20px;

      background-color: #0069d9;

      border: none;

      color: white;

      cursor: pointer;

      font-size: 16px;

    }

    .output {

      margin-top: 20px;

      background-color: #e2e3e5;

      padding: 15px;

      border-radius: 5px;

    }

    .time-complexity {

      margin-top: 10px;

      font-style: italic;

      color: #555;

    }

    .steps {

      margin-top: 15px;

      background-color: #f1f3f5;

      padding: 10px 15px;

      border-radius: 5px;

      font-size: 14px;

      color: #333;

    }

    .steps h4 {

      margin-bottom: 8px;

    }

  </style>

</head>

<body>

  <div class="container">

    <i><h2>Network Topology Detection</h2></i>

    <form method="POST">

      <label>Number of Nodes:</label>

      <input

        type="number"

        name="nodes"

        min="2"

        required

        value="{{ nodes or '' }}"

      />

      <label>Enter Adjacency Matrix (each row on new line):</label>

      <textarea

        name="matrix"

        rows="6"

        placeholder="Example for 3 nodes:

0 1 0

1 0 1

0 1 0"

        required

      >{{ matrix\_input }}</textarea>

      <button type="submit">Detect Topology</button>

    </form>

    {% if result %}

    <div class="output">

      <strong>{{ result }}</strong><br />

      {{ suggestion }}

      {% if time\_complexity %}

      <div class="time-complexity">Time Complexity: {{ time\_complexity }}</div>

      {% endif %}

    </div>

    {% endif %}

    {% if steps %}

    <div class="steps">

      <h4>Detection Process:</h4>

      <ul>

        {% for step in steps %}

          <li>{{ step }}</li>

        {% endfor %}

      </ul>

    </div>

    {% endif %}

  </div>

</body>

</html>

**DIFFERENCE BETWEEN BRUTE FORCE AND LINEAR SCAN FOR NETWORK TOPOLOGY DETECTION:**

**Brute Force Approach:**  
The brute force method involves examining the entire adjacency matrix thoroughly by comparing each node’s connections with every other node’s connections to determine the network topology. This leads to a time complexity of **O(n²)** since it involves checking each element in the matrix multiple times. Though conceptually straightforward, it becomes inefficient and slow when dealing with larger networks because the number of comparisons grows quadratically with the number of nodes.

**Linear Scan Approach:**  
The linear scan approach simplifies the problem by calculating the degree of each node once (by summing its adjacency matrix row) and then analyzing the resulting degree list to identify patterns matching known topologies. This reduces the time complexity to **O(n²)** for the adjacency matrix scan but avoids repetitive pairwise comparisons. The pattern recognition step runs in **O(n)**. Overall, the linear scan is much faster and more practical for real-time or large networks.

**Space Complexity:**  
Both approaches require **O(n²)** space to store the adjacency matrix. The degree list requires **O(n)** space.

* **Brute force** checks every possible pair multiple times, leading to redundant work and inefficiency.
* **Linear scan** optimizes by computing node degrees once and performing pattern matching, which is faster and cleaner.

**Time Complexity:**

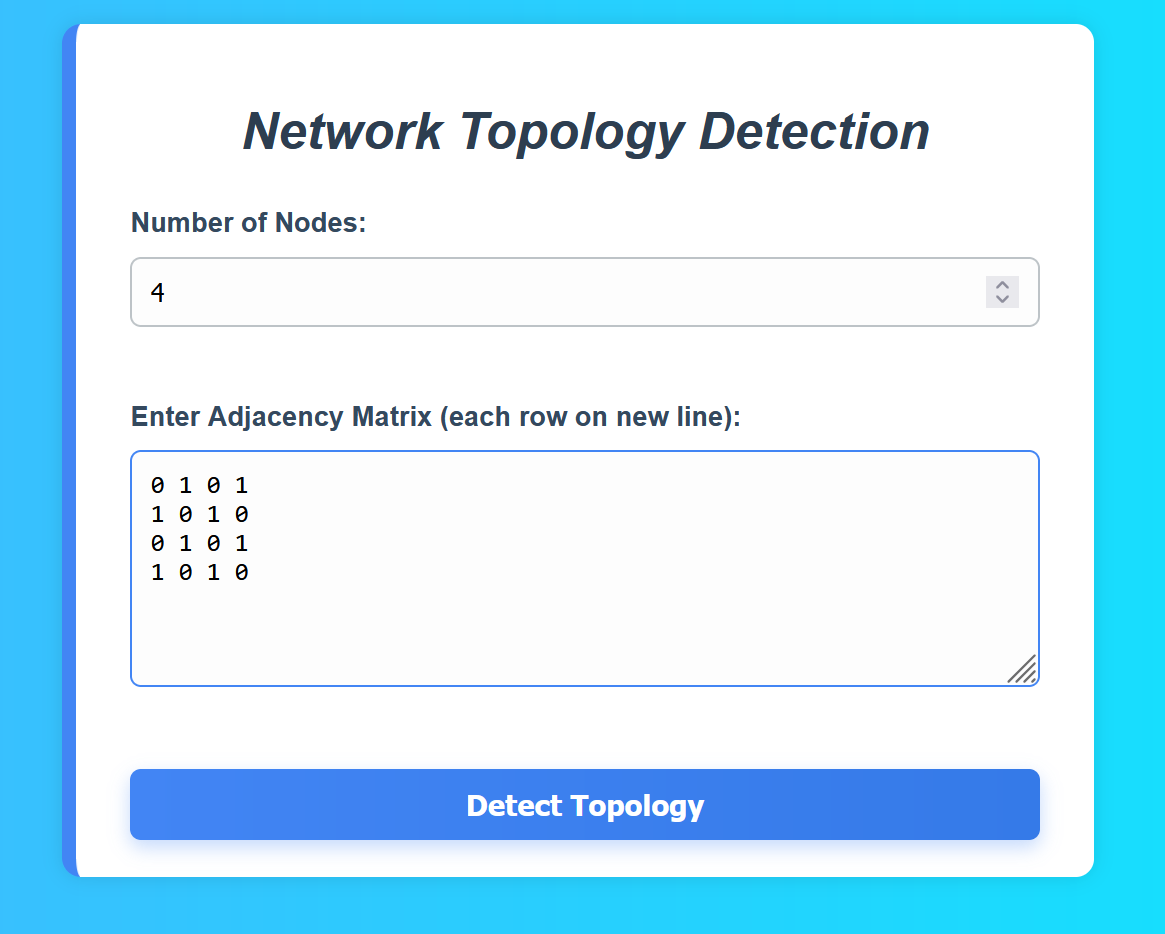
* **Brute Force:** O(n²) comparisons (less efficient for large networks)
* **Linear Scan:** O(n²) to compute degrees + O(n) for pattern detection (more efficient and practical)

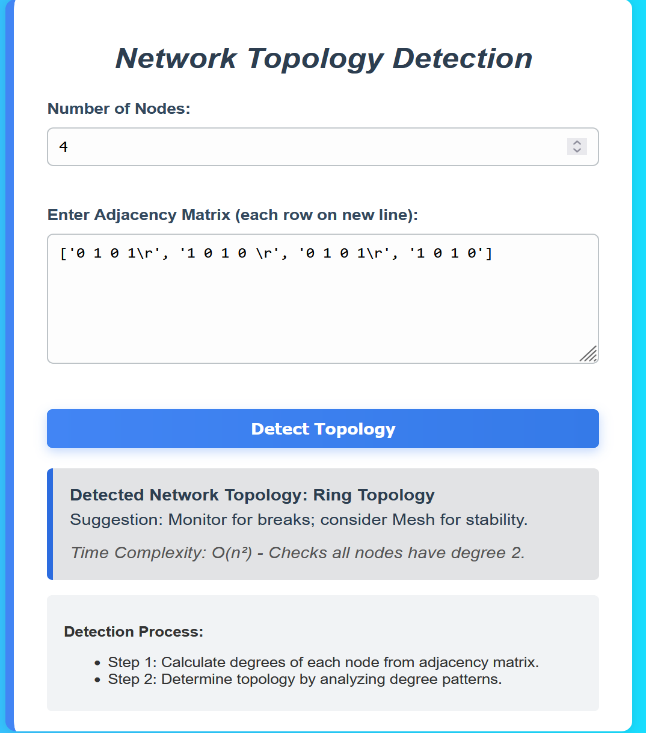
**Pros and Cons:**

| **Approach** | **Pros** | **Cons** |
| --- | --- | --- |
| **Brute Force** | Simple to implement and understand | Inefficient due to redundant comparisons, slow for large networks |
| **Linear Scan** | Efficient degree calculation, faster pattern recognition | Slightly more logic needed for pattern checks |
| **Aspect** | Brute Force Approach | Linear Scan Approach |
| **Basic Idea** | Check every node’s connections against all other nodes repeatedly | Calculate the degree of each node once and analyze degree patterns |
| **Number of Comparisons** | For each node, compare connections with all others → O(n²) comparisons | One pass through adjacency matrix rows to compute degrees → O(n²), then O(n) for pattern check |
| **Time Complexity** | O(n²) — quadratic time due to repeated comparisons | O(n²) to calculate degrees + O(n) for pattern detection (more efficient overall) |
| **Space Complexity** | O(n²) to store adjacency matrix | O(n²) for adjacency matrix + O(n) for degree list |
| **Implementation Complexity** | Conceptually straightforward but involves redundant checks | Slightly more logic to analyze degree patterns but cleaner overall |
| **Performance** | Inefficient for larger networks | More efficient, scalable, suitable for real-time applications |
| **Use Case** | Small networks or when simplicity is preferred | Larger or real-time networks where performance matters |

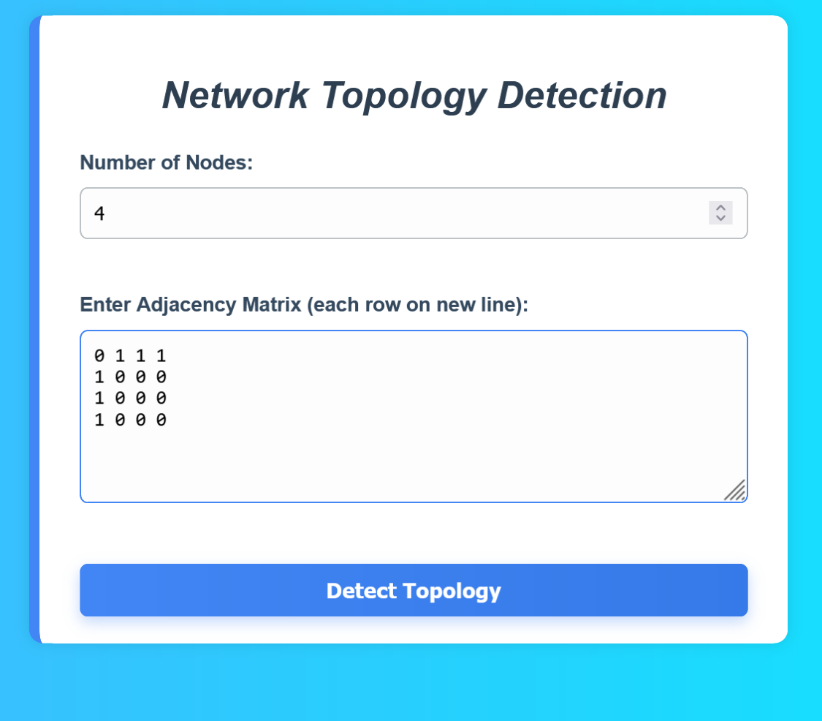
**5.0. RESULTS:**

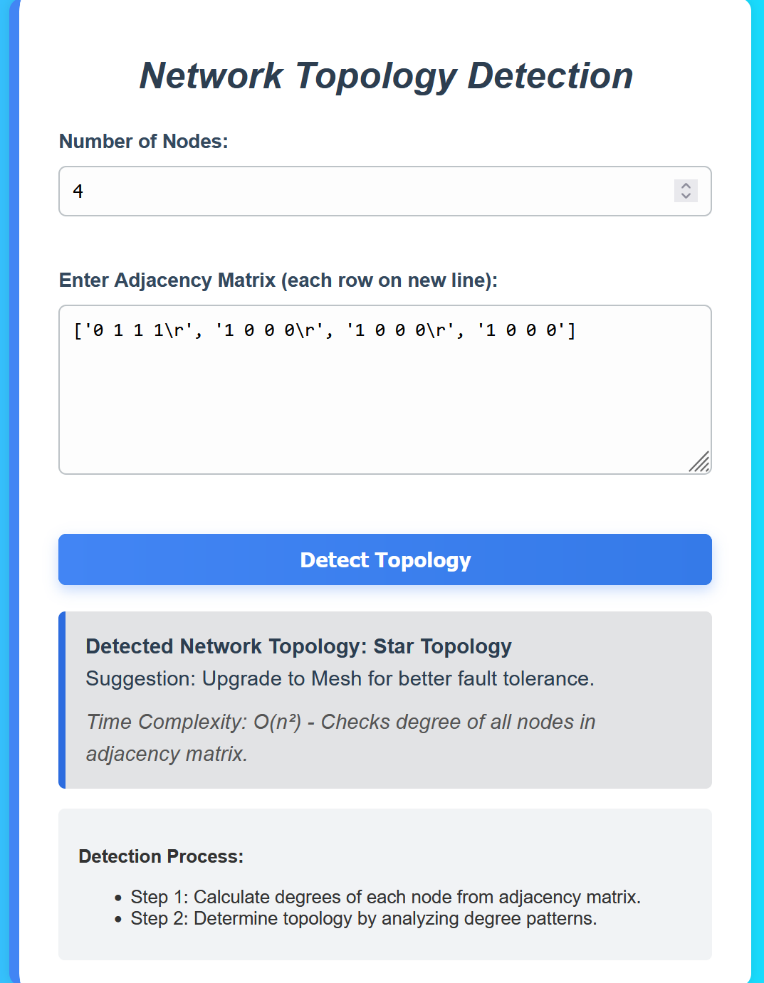
**1.Ring Topology:**



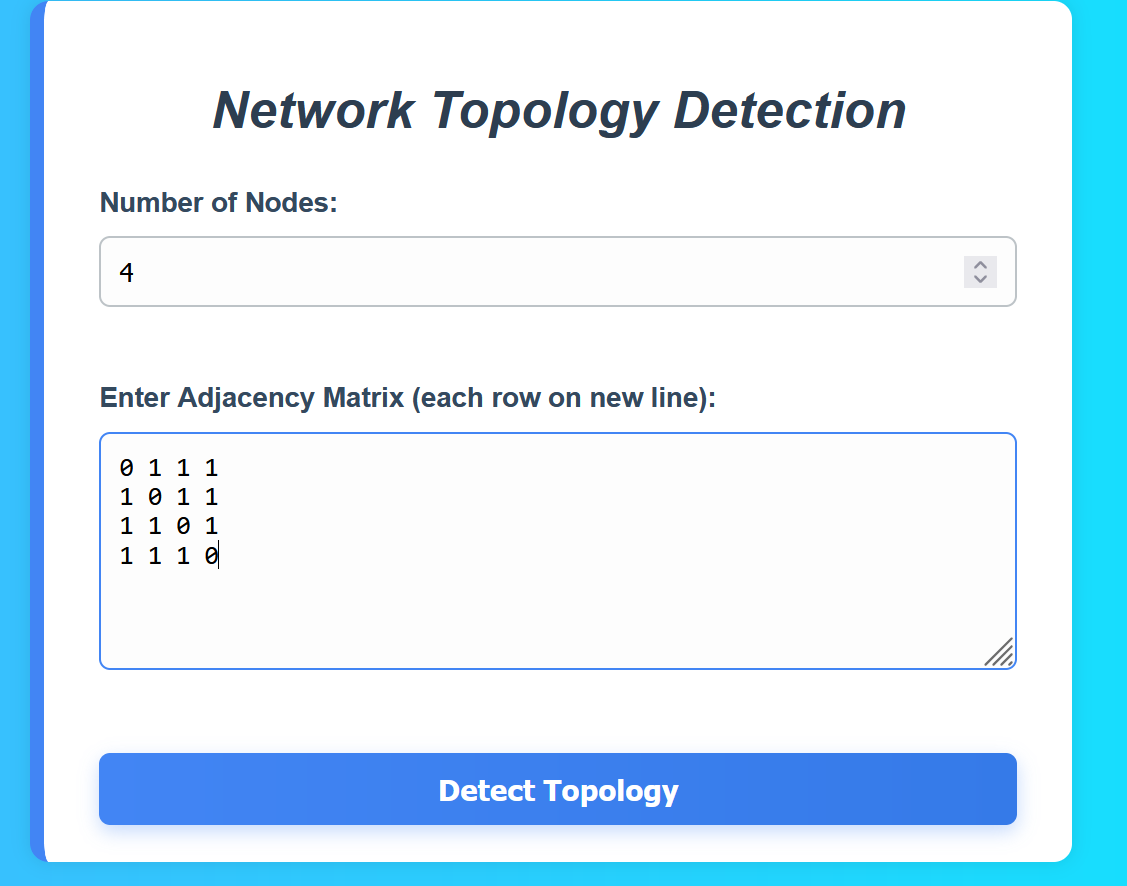


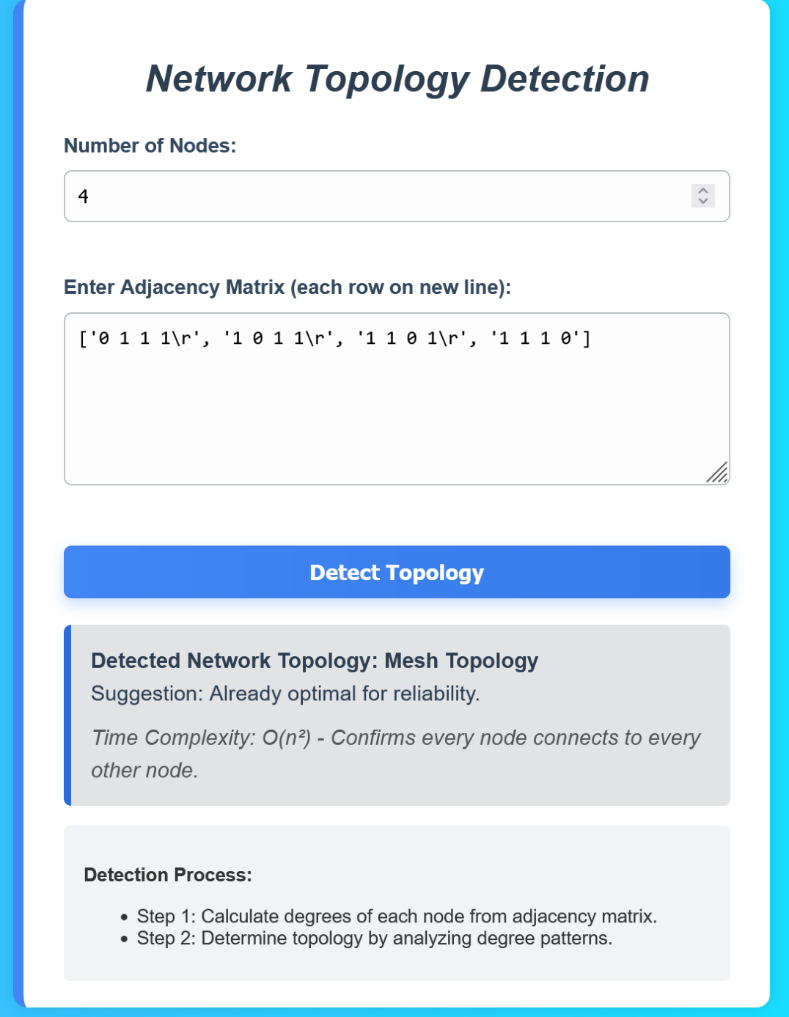
**2.Star Topology:**

****

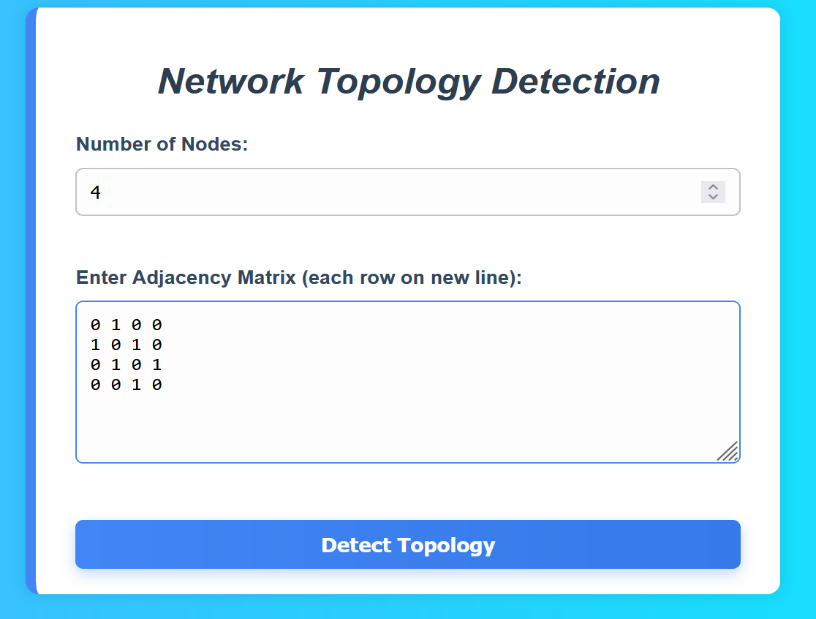
****

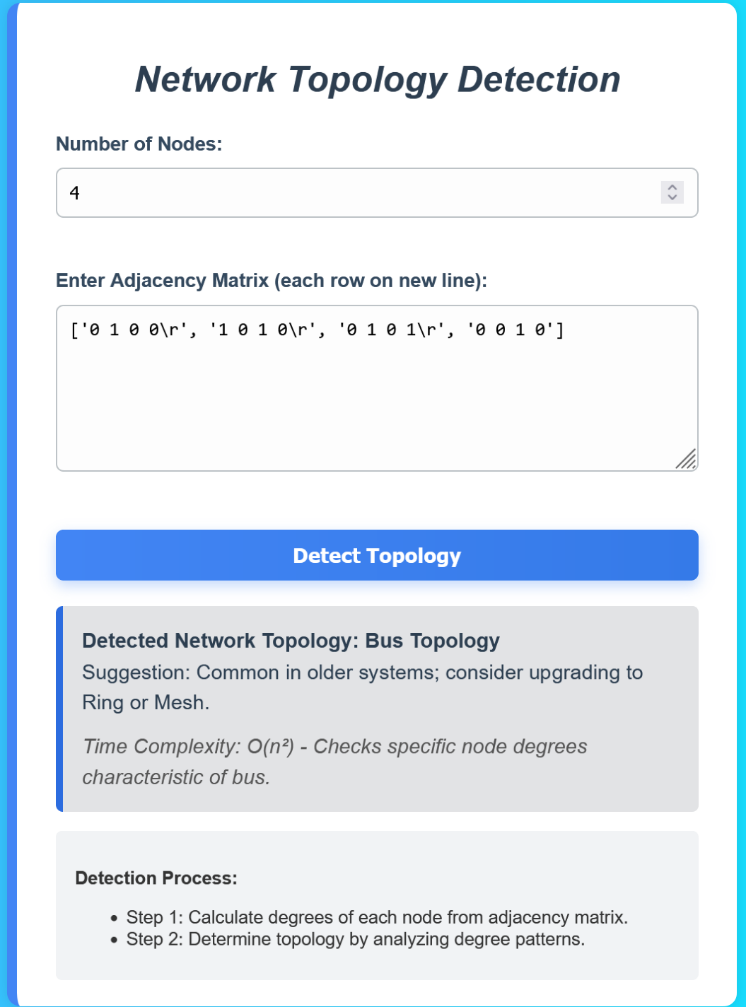
**3.Mesh Topology:**

****

****

**4.Bus Topology:**

****

****

**GITHUB LINK:** https://github.com/GowthamM49/daa\_project