

# **NETWORK ANALYSIS USING GRAPH**

## **GROUP ASSIGNMENT REPORT**

*Submitted by*

**Anouska Jhunjhunwala (RA2211003010637)**

**Samarth Mishra (RA2211003010659)**

**Jhanvi Singh (RA2211003010625)**

*Under the Guidance of*

**Dr. Arunachalam N**

**Assistant Professor, Department of Computing Technologies**

*In partial fulfillment for the  
Course of*

***21CSC201J – DATA STRUCTURES AND ALGORITHMS***

***in DEPARTMENT OF COMPUTING TECHNOLOGIES***



**FACULTY OF ENGINEERING AND  
TECHNOLOGY SCHOOL OF COMPUTING  
SRM INSTITUTE OF SCIENCE AND  
TECHNOLOGY KATTANKULATHUR**

**NOVEMBER 2023**



**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**  
**(Under Section 3 of UGC Act, 1956)**

**BONAFIDE CERTIFICATE**

Certified that this Group Assignment Report titled “**NETWORK ANALYSIS USING GRAPH**” is the bonafide work done by Anouska Jhunjhunwala (RA2211003010637), Samarth Mishra (RA2211003010659) and Jhanvi Singh (RA2211003010625) who completed the project under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

**SIGNATURE**

Dr. Arunachalam N

**DSA – Course Faculty**

Assistant Professor

Department of Computing Technologies

SRMIST

## Table Of Contents

Serial no.	Title	Page no.
1.	Introduction	2
2.	Data structures	3
3.	Tools and Technologies	4
4.	Source code and Function Explanation	5
5.	Code Output and Interpretation	12
6.	Time Complexity Analysis	14
7.	Conclusion	15

# 1: Introduction

## 1.1 Problem Statement

In this report, we are addressing the problem of identifying network topologies, specifically focusing on detecting whether a given network graph represents a star topology. A star topology is characterized by all nodes being connected to a central node, with no direct connections existing between non-central nodes. This report aims to provide a comprehensive analysis of the code developed for this purpose.

## 1.2 Objectives

The primary objectives of this report can be summarized as follows:

### 1. Explain the Code's Functionality and Implementation in Python

We will delve into a detailed explanation of how the code functions and its implementation in the Python programming language. This includes a breakdown of the algorithms and techniques used to identify a star topology within a network graph.

### 2. Elaborate on the Data Structures Used and Their Justification

The choice of data structures in the code plays a crucial role in its performance. We will discuss the data structures employed and provide justification for their selection, highlighting how they enhance the efficiency of the code.

### 3. Analyze the Code's Output and Discuss Its Significance

An essential part of this report is to analyze the output generated by the code. We will interpret the results and discuss their significance in the context of network topology identification. This will help us understand the practical implications of the code's findings.

### 4. Evaluate the Code's Time Complexity and Efficiency

Efficiency is a critical factor in code designed for network topology identification, especially for large and complex network graphs. We will evaluate the code's time complexity and overall efficiency, discussing any potential areas for optimization.

## 2. Data Structures in Network Topology Detection

Data structures are pivotal in addressing challenges related to network topologies, facilitating the effective representation and manipulation of graphs. Understanding these structures is fundamental when dealing with network topology detection.

### 2.1 The Significance of Adjacency Lists

One of the critical elements in network representation is the choice of data structure. In this section, we will explore the significance of using adjacency lists, a fundamental data structure in graph representation. Adjacency lists provide an efficient means to store connections between nodes, offering scalability and versatility in network topology detection.

#### 2.1.1 What are Adjacency Lists?

Adjacency lists are a way of representing a graph, where each node is associated with a list of its adjacent nodes. This list records all the connections that a given node has, making it a highly efficient structure for working with graphs. In the context of network topology detection, adjacency lists help us organize and access information about how nodes are interconnected.

#### 2.1.2 Efficiency in Storing Connections

One of the significant advantages of adjacency lists is their efficiency in storing connections in a graph. Unlike other representations, such as adjacency matrices, adjacency lists excel in efficiently handling sparse graphs. In network topology, many graphs are indeed sparse, where only a fraction of possible connections exists. Adjacency lists make it easy to represent such graphs without wasting memory on non-existent connections.

#### 2.1.3 Scalability and Versatility

Adjacency lists offer scalability and versatility, making them an excellent choice for various network topologies. Whether dealing with small or large-scale networks, adjacency lists can adapt effectively. They enable quick and easy additions and deletions of edges, which is essential for dynamic network analysis.

### 2.2 Justification for Using Adjacency Lists

The selection of adjacency lists as the primary data structure is not arbitrary but a result of careful consideration. Sparse graphs, frequently encountered in network topology analysis, align perfectly with the capabilities of adjacency lists.

#### 2.2.1 Handling Sparse Graphs

Sparse graphs, where the number of connections is significantly lower than the maximum possible connections, are commonplace in network topology studies. In such cases, using an adjacency matrix to represent the network would be inefficient and memory-intensive. Adjacency lists, however, shine in this scenario as they precisely store the existing connections without wasting resources on non-connections.

### 2.2.2 Flexibility in Representation

Another advantage of adjacency lists is their flexibility in representing different types of networks. They can be easily adapted to model directed or undirected graphs, weighted edges, and can even accommodate metadata about nodes and edges. This adaptability is vital when dealing with the diverse range of network topologies that may be encountered.

## 3: Tools and Technologies in Network Topology Detection

In the realm of solving network topology detection problems, the choice of tools and technologies plays a pivotal role in the success of the implementation. In this section, we will introduce the key tools and technologies employed in the development of the network topology detection code, focusing on the programming language and standard input/output mechanisms.

### 3.1 Python Programming Language

Selecting an appropriate programming language is a fundamental decision when implementing a codebase. In this context, Python stands out as the primary tool for solving the network topology detection problem. Several compelling reasons justify this choice:

#### 3.1.1 Versatility and Power

Python is renowned for its versatility and power, making it an excellent choice for addressing a wide range of problems, including network topology detection. Its clear and concise syntax enhances readability, facilitating code comprehension and maintenance.

#### 3.1.2 Extensive Library Support

Python boasts a rich ecosystem of libraries and packages that simplify various aspects of code development. This wealth of resources accelerates the implementation process, allowing developers to leverage existing solutions for tasks such as graph manipulation and algorithm implementation.

#### 3.1.3 Data Manipulation Tools

The availability of robust data manipulation tools in Python, particularly in libraries like NumPy and pandas, is crucial for processing and analyzing network graph data efficiently. These tools enable the code to manage, preprocess, and extract insights from complex network structures.

### 3.2 Standard Input and Output

Effective interaction with users is an integral aspect of the network topology detection code's functionality. To ensure a user-friendly experience, the code leverages standard input and output features. Here's why these features are vital:

#### 3.2.1 User-Friendly Interaction

Standard input and output mechanisms provide a straightforward and user-friendly means for users to communicate with the code. Users can input essential information, such as the number of vertices and edges, and receive meaningful output, such as the detection of a star topology. This direct interaction enhances the code's utility and accessibility.

## 4: Source Code and Functions Explanation

### 4.1 The addEdge Function

The foundation of the code's functionality lies in the `addEdge` function. This function is crucial for building the connections within the graph. It takes two vertices, `u` and `v`, and establishes a connection between them. This section provides a comprehensive explanation of how the `addEdge` function works, its role in the code, and how it contributes to the construction of the network graph.

```
def addEdge(adj, u, v):  
    adj[u].append(v)  
    adj[v].append(u)
```

### 4.2 The checkStarTopologyUtil Function

At the core of the code for detecting star topologies lies the `checkStarTopologyUtil` function. This function is responsible for evaluating the network's topology and determining whether it exhibits the characteristics of a star topology. In this section, we delve into the logic and operation of the `checkStarTopologyUtil` function. We explain how it inspects the network graph, calculates degrees, and identifies central nodes.

```
9  def checkStarTopologyUtil(adj, V, E):  
10     if E != (V - 1):  
11         return False  
12  
13     if V == 1:  
14         return True  
15  
16     vertexDegree = [0] * (V + 1)  
17     for i in range(V + 1):  
18         for v in adj[i]:  
19             vertexDegree[v] += 1  
20  
21     countCentralNodes = 0  
22     centralNode = 0  
23  
24     for i in range(1, V + 1):  
25         if vertexDegree[i] == (V - 1):  
26             countCentralNodes += 1  
27             centralNode = i  
28  
29     if countCentralNodes != 1:  
30         return False  
31  
32     for i in range(1, V + 1):  
33         if i == centralNode:  
34             continue  
35         if vertexDegree[i] != 1:  
36             return False  
37     return True
```



## 4.3 The checkStarTopology Function

The ultimate verdict on whether a network represents a star topology or not is delivered by the `checkStarTopology` function. This function leverages the `checkStarTopologyUtil` function to make this determination. In this section, we provide a detailed explanation of how the `checkStarTopology` function utilizes the utility function, its significance in the problem-solving process, and the meaning of its output.

```
39 def checkStarTopology(adj, V, E):
40     isStar = checkStarTopologyUtil(adj, V, E)
41     if isStar:
42         return "YES, This is a STAR TOPOLOGY:)"
43     else:
44         return "NO, This isn't a star topology:("
45
```

## 4.4 Input Handling

Proper input handling is vital to the robustness of the code. In this section, we discuss how the code handles user input. Users are prompted to provide information such as the number of vertices and edges. The input handling process ensures that the code is well-prepared to perform the network topology detection task accurately.

```
47 try:
48     V, E = map(int, input("Enter the number of vertices and edges: ").split())
49     adj = [[] for i in range(V + 1)]
50
51     for i in range(E):
52         u, v = map(int, input("Enter an edge (u v): ").split())
53         addEdge(adj, u, v)
54
55     checkStarTopology(adj, V, E)
56 except ValueError:
57     print("Invalid input. Please enter valid integers for the number of vertices and edges.")
58 except Exception as e:
59     print(f"An error occurred: {e}")
60
```

## 4.5 HTML Code (index.html)

```
index.html X app.py 1 topology.py # style.css JS script.js
templates > <> index.html > html > body > div#output
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Network Topology Detection</title>
7   <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
8 </head>
9 <body>
10   <h1>Network Topology Detection</h1>
11   <form id="input-form">
12     <label for="num-vertices">Number of Vertices:</label>
13     <input type="number" id="num-vertices" required>
14     <label for="num-edges">Number of Edges:</label>
15     <input type="number" id="num-edges" required>
16     <div id="edge-input-container"></div>
17     <button type="submit" id="submit-button">Submit</button>
18   </form>
19   <div id="output">
20     <button id="check-topology" style="display: none;">Check Topology</button>
21     <p id="result"></p>
22   </div>
23   <script src="{{ url_for('static', filename='script.js') }}"></script>
24 </body>
25 </html>
26
```

## 4.6 CSS Code(style.css)

```
index.html app.py 1 topology.py # style.css X JS script.js
static > # style.css > h1
1 body {
2   font-family: 'Times New Roman', serif;
3   background-color: #87CEEB; /* Sky Blue */
4   text-align: center;
5   padding: 20px;
6 }
7
8 h1 {
9   font-size: 24px;
10 }
11
12 form {
13   background-color: #fff;
14   padding: 10px;
15   margin: 20px auto;
16   max-width: 400px;
17   border-radius: 5px;
18   box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
19 }
20
21 label, input {
22   display: block;
23   margin: 10px 0;
24 }
25
26 button {
27   background-color: #0074D9;
28   color: #fff;
29   border: none;
```

```

30     padding: 10px 20px;
31     border-radius: 5px;
32     cursor: pointer;
33 }
34
35 #output {
36     display: none;
37     padding: 20px;
38     background-color: #ffff;
39     max-width: 400px;
40     margin: 0 auto;
41     border-radius: 5px;
42     box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
43 }
44
45 #result {
46     font-size: 18px;
47     margin: 10px 0;
48 }
49
50 #check-topology {
51     background-color: #0074D9;
52     color: #fff;
53     border: none;
54     padding: 10px 20px;
55     border-radius: 5px;
56     cursor: pointer;
57     margin-top: 10px;
58 }

```

## 4.7 JavaScript Code (script.js)

```

<> index.html  app.py 1  topology.py  # style.css  JS script.js  X
static > JS script.js > document.addEventListener("DOMContentLoaded") callback > inputForm.addEventListener("submit") callback
1  document.addEventListener("DOMContentLoaded", () => {
2      const inputForm = document.getElementById("input-form");
3      const submitButton = document.getElementById("submit-button");
4      const outputDiv = document.getElementById("output");
5      const checkTopologyButton = document.getElementById("check-topology");
6      const resultText = document.getElementById("result");
7      const edgeInputContainer = document.getElementById("edge-input-container");
8
9      inputForm.addEventListener("submit", (event) => {
10         event.preventDefault();
11         const numVertices = parseInt(document.getElementById("num-vertices").value);
12         const numEdges = parseInt(document.getElementById("num-edges").value);
13
14         if (isNaN(numVertices) || isNaN(numEdges)) {
15             alert("Please enter valid integers for the number of vertices and edges.");
16         } else {
17             edgeInputContainer.innerHTML = '';
18             for (let i = 0; i < numEdges; i++) {
19                 const edgeInput = document.createElement("input");
20                 edgeInput.setAttribute("type", "text");
21                 edgeInput.setAttribute("placeholder", `Edge ${i + 1} (u v)`);
22                 edgeInput.setAttribute("name", `edge${i + 1}`);
23                 edgeInputContainer.appendChild(edgeInput);
24             }
25
26             outputDiv.style.display = "block";
27             checkTopologyButton.style.display = "block";
28         }
29     });

```

```

31     checkTopologyButton.addEventListener("click", () => {
32         const numVertices = parseInt(document.getElementById("num-vertices").value);
33         const numEdges = parseInt(document.getElementById("num-edges").value);
34         const edgeInputs = [];
35
36         for (let i = 0; i < numEdges; i++) {
37             const edgeInput = document.querySelector(`[name=edge${i + 1}]`);
38             edgeInputs.push(edgeInput.value);
39         }
40
41         const requestData = {
42             numVertices: numVertices,
43             numEdges: numEdges,
44             edgeInputs: edgeInputs,
45         };
46
47         fetch("/check_topology", {
48             method: "POST",
49             headers: {
50                 "Content-Type": "application/json",
51             },
52             body: JSON.stringify(requestData),
53         })
54         .then((response) => response.json())
55         .then((data) => {
56             resultText.textContent = data.result;
57         })
58         .catch((error) => {
59             console.error("Error:", error);

```

```

60             resultText.textContent = "An error occurred.";
61         });
62     });
63 });
64

```

```

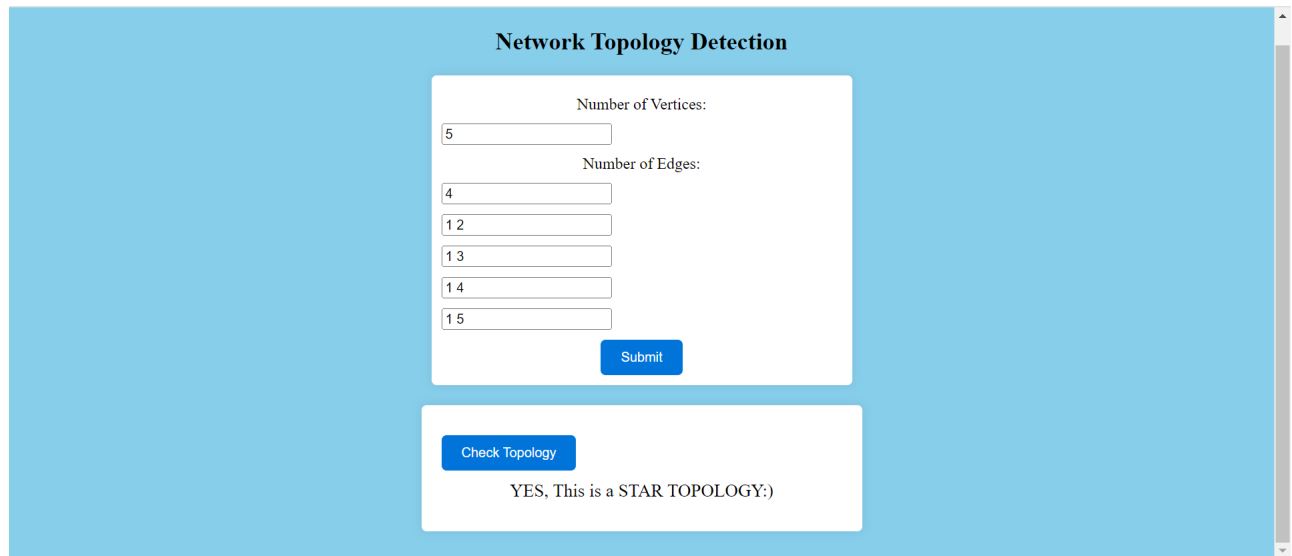
app.py > checkStarTopologyUtil
1  from flask import Flask, render_template, request, jsonify
2
3  app = Flask(__name__)
4
5  def addEdge(adj, u, v):
6      adj[u].append(v)
7      adj[v].append(u)
8
9  def checkStarTopologyUtil(adj, V, E):
10     if E != (V - 1):
11         return False
12
13     if V == 1:
14         return True
15
16     vertexDegree = [0] * (V + 1)
17     for i in range(V + 1):
18         for v in adj[i]:
19             vertexDegree[v] += 1
20
21     countCentralNodes = 0
22     centralNode = 0
23
24     for i in range(1, V + 1):
25         if vertexDegree[i] == (V - 1):
26             countCentralNodes += 1
27             centralNode = i
28
29     if countCentralNodes != 1:

```

## 4.8 Flask Code(app.py)

```
30         return False
31
32     for i in range(1, V + 1):
33         if i == centralNode:
34             continue
35         if vertexDegree[i] != 1:
36             return False
37     return True
38
39 def checkStarTopology(adj, V, E):
40     isStar = checkStarTopologyUtil(adj, V, E)
41     if isStar:
42         return "YES, This is a STAR TOPOLOGY:)
43     else:
44         return "NO, This isn't a star topology:("
45
46 @app.route('/')
47 def index():
48     return render_template('index.html')
49
50 @app.route('/check_topology', methods=['POST'])
51 def check_topology():
52     try:
53         data = request.json # Access the data as JSON
54         num_vertices = data.get('numVertices')
55         num_edges = data.get('numEdges')
56         edge_inputs = data.get('edgeInputs')
57
58         if num_vertices is None or num_edges is None or not isinstance(edge_inputs, list):
59             return jsonify({"result": "Invalid input. Please provide valid data."})
60
61         adj = [[] for _ in range(num_vertices + 1)]
62
63         for i in range(num_edges):
64             edge_input = edge_inputs[i]
65             if edge_input:
66                 u, v = map(int, edge_input.split())
67                 addEdge(adj, u, v)
68
69         result = checkStarTopology(adj, num_vertices, num_edges)
70
71         return jsonify({"result": result})
72     except (ValueError, TypeError):
73         return jsonify({"result": "Invalid input. Please provide valid data."})
74
75 if __name__ == '__main__':
76     app.run(debug=True)
77
```

## 5: Code Output and Interpretation



Network Topology Detection

Number of Vertices:  
5

Number of Edges:  
4  
1 2  
1 3  
1 4  
1 5

Submit

Check Topology

YES, This is a STAR TOPOLOGY:)

Figure 5.1: GUI

### 5.1 Interpretation of Code Output

The moment of truth arrives when the code delivers its verdict. The output of the code can be one of two possibilities: "YES, This is a STAR TOPOLOGY:)" or "NO, This isn't a star topology:(". In this section, we delve into the interpretation of these outcomes and what they mean in the context of network topology.

#### 5.1.1 "YES, This is a STAR TOPOLOGY:)"

When the code produces the affirmative output, it signifies the successful identification of a star topology within the given network graph. In a star topology, all nodes are directly connected to a central node, and there are no direct connections between non-central nodes. The presence of such a configuration can have implications for network management and reliability.

#### 5.1.2 "NO, This isn't a star topology:("

Conversely, when the code negates the presence of a star topology, it indicates that the network graph does not adhere to the characteristics of a star configuration. In such cases, there are direct connections between non-central nodes, deviating from the star topology model. Understanding this outcome prompts further exploration of the network's configuration and may lead to optimizations for specific network requirements.

In this section, we establish the meaning and implications of the code's output, providing readers with a clear understanding of how to interpret the results and their significance in the context of network topology.

## **5.2 Discussion of Network Topology**

In this section, we delve deeper into the world of network topologies. We explain what constitutes a star topology and why it holds significance. By connecting the code's output to the broader domain of network topologies, we provide a comprehensive understanding of the code's findings and how they relate to real-world network configurations.

### **5.2.1 What is a Star Topology?**

A star topology is a fundamental network configuration where all nodes or devices are directly connected to a central hub or switch. This central hub serves as a focal point for data transmission, and there are no direct connections between non-central nodes. The star topology is favored for its simplicity and ease of management.

### **5.2.2 Significance of Star Topologies**

Understanding and identifying a star topology is essential because it has several practical applications. Star topologies are commonly used in local area networks (LANs) and are known for their reliability. They offer centralized control and easy troubleshooting, making them a preferred choice for many organizations.

### **5.2.3 Connecting Code Output to Real-World Networks**

The code's output, whether it affirms the presence of a star topology or not, holds value in the real world. By recognizing star topologies, organizations can ensure efficient network management and troubleshooting. Conversely, the absence of a star topology may prompt further exploration of network configurations to optimize performance and reliability.

In summary, this section provides a bridge between the code's output and its practical implications in the context of network topologies, specifically the significance of star configurations.

## **6: Time Complexity Analysis**

### **6.1 Complexity of Constructing Adjacency List**

Efficiency in handling large network graphs is paramount. In this section, we provide a detailed analysis of the time complexity involved in constructing the adjacency list. This process includes the initialization of the list and the calculation of vertex degrees, with a time complexity of  $O(V + E)$ . Understanding this complexity is essential for optimizing the code's performance with varying graph sizes.

### **6.2 Time Complexity of Topology Detection**

The efficiency of the code in detecting network topologies is closely examined in terms of time complexity. Checking for a star topology involves a time complexity of  $O(V)$ . We delve into the implications of this time complexity and discuss the code's performance in relation to different graph sizes.

## **6.3 Efficiency Insights**

The discussion extends to provide insights into how the code's time complexity impacts its efficiency. By considering the various complexities involved in network topology detection, we gain a better understanding of the code's performance in practical applications.



## 7: Conclusion

In the concluding section of this report, we summarize the key points discussed throughout the document, providing a comprehensive overview of the entire endeavor.

We began by introducing the problem of network topology detection, focusing on the identification of star topologies. The objectives of the report were established, which included explaining the code's functionality, elaborating on the data structures used, analyzing the code's output, and evaluating its time complexity and efficiency.

The significance of data structures, specifically adjacency lists, in solving network topology problems was highlighted, along with the justification for their selection. Python was introduced as the chosen programming language, and the role of standard input and output for user interaction was discussed.

The source code and functions, including `addEdge`, `checkStarTopologyUtil`, and `checkStarTopology`, were explained, along with the importance of input handling for code robustness.

We interpreted the code's output, distinguishing between "YES, This is a STAR TOPOLOGY:)" and "NO, This isn't a star topology:". We also discussed the broader context of network topologies and why a star topology is of interest.

A comprehensive time complexity analysis was presented, covering the construction of the adjacency list and topology detection. This analysis provided insights into the code's efficiency and performance for varying graph sizes.

In conclusion, this report offers a thorough understanding of network topology detection and the code's role in solving this problem. The code's functionality, data structures, and efficiency were all explored, contributing to a valuable tool for network analysis. The report emphasizes the relevance of network topology detection and the applicability of the code in addressing real-world network configurations.

## 8: References

1. *"Computer Networking: Principles, Protocols, and Practice"* by Olivier Bonaventure
2. *"Data Communications and Networking"* by Behrouz A. Forouzan
3. *"Networks: An Introduction"* by Mark Newman
4. *"Networks, Crowds, and Markets: Reasoning About a Highly Connected World"* by David Easley and Jon Kleinberg