Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

A Report on '**Lab Work 5'** [COMP 314]

Submitted by:

Chandan Kumar Mahato (31)

III-year, II semester

Submitted to:

Dr. Rajani Chulyadyo

Department of Computer Science and Engineering

Submission Date: June 4, 2023

Contents

Table of Figures

Qlab5

To be familiar with JavaScript Graph Libraries.

1. ***Explore Graph Libraries***
2. ***Network Repository Data Collection***
3. ***Graph Visualization***
4. ***Compute Graph Property***
5. ***Compare Graph Property***

Solution:

Implemented graph using visualization libraries in node.js **"vis.js".** Data from network repository. Graph Properties like, no of nodes, no of edges, average density, average degree, diameter, average clustering coefficient and degree distribution graph has been calculated. All the calculations to **compute property** are **done manually without using any library methods**. Libraries are used only for plotting the graph. **vis.js** to plot graph. And **nodeplotlib** to plot degree distribution graph.

GitHub Link:

https://github.com/ChandankMahato/DSA_Lab_6th_Sem/tree/master/Lab5

Scripts:

//inside root directory "DSA_Lab_6th_Sem

npm install --save

//inside Lab5

npm run drawQ1

npm run drawQ2G1

npm run drawQ2G2

npm run drawQ2G3

npm run drawQ2G4

npm run drawQ2G5

1. Downloading Data from Network Repository

These Screenshots are showing the data that has been used in this lab work. In the screenshot data name is included and all the property for that graph data. And that property has been matched with my calculation.

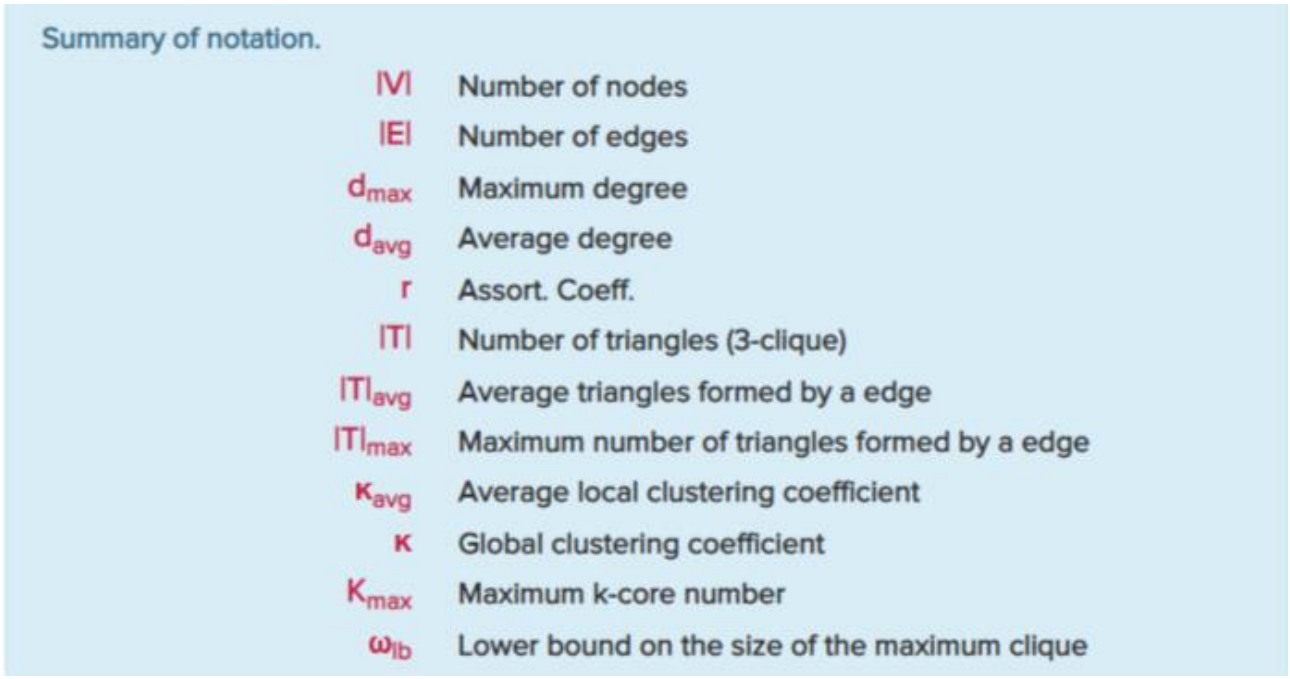


Figure 1 Graph Notations

Graph Name	|V|	|E|	d_{max}	d_{avg}	r	|T|	T_{avg}	T_{max}	κ_{avg}	κ	K	ω_{heu}	Size
aves-sparrow-social	52	516	43	19	0.03	8K	158	433	0.85	0.63	16	10	3 KB

Figure 2 Graph Data Q1 References

Graph Name	|V|	|E|	d_{max}	d_{avg}	r	|T|	T_{avg}	T_{max}	κ_{avg}	κ	K	ω_{heu}
insecta-ant-colony5-day12	133	6K	120	91	0.06	454K	3K	5K	0.80	0.79	73	38
insecta-ant-colony5-day13	133	6K	122	90	0.04	440K	3K	5K	0.79	0.78	67	37
insecta-ant-colony5-day14	133	6K	132	93	0.00	472K	4K	6K	0.80	0.78	71	36
insecta-ant-colony5-day15	133	6K	129	90	-0.02	419K	3K	6K	0.77	0.76	69	27
insecta-ant-colony5-day16	133	6K	127	89	-0.02	422K	3K	6K	0.79	0.77	66	32

Figure 3 Graph Data Q2 References

2. Driver Code

```
const data = extractData("./Q1/52_nodes.edges");
const nodes = data.nodes;
const edges = data.edges;
const property = computerProperty(nodes, edges);
drawDegreeDistributionGraph(nodes, edges, "Graph Q1 - 52 Nodes");
fs.writeFile(
  "./Q1/index.html",
  createHtmlContent(nodes, edges, property, "Q1"),
  (err) => {
    if (err) {
      console.error(err);
      return;
    }
    console.log("./Q1/index.html file created successfully.");
    exec("start ./Q1/index.html", (error) => {
      if (error) {
        console.error(`Error opening file: ${error}`);
      }
    });
  }
);
```

2.1. Extraction of Data

Code showing the extraction of data in required format from graph data file.

```
function extractData(path) {
  const networkData = fs.readFileSync(path, "utf-8");
  const edges = networkData.split("\n").map((line) => {
    const [source, target] = line.split(" ");
    return { from: source, to: target, distance: "1" };
  });
  const nodes = [];
  edges.forEach((edge) => {
    if (!nodes.includes(edge.from)) {
      nodes.push(edge.from);
    }
    if (!nodes.includes(edge.to)) {
      nodes.push(edge.to);
    }
  });
  return { edges: edges, nodes: nodes };
}
```

2.2. Computing Property

2.2.1. Driver code to compute Property

```
function computerProperty(nodes, edges) {
  let noOfNodes = nodes.length;
  let noOfEdges = edges.length;
  let avgDegree = calculateAvgDegree(nodes, edges);
  let density = calculateDensity(noOfNodes, noOfEdges);
  let path = longestShortestPath(nodes, edges);
  let mostDistant = mostDistantNode(path[1]);
  let shortestPath = path[1];
  let diameter = path[0].toFixed(2);
  let clusteringCoffecient = clusteringCoff(nodes, edges);
  return {
    noOfNodes, noOfEdges, avgDegree,
    density, mostDistant,
    shortestPath,
    diameter,
    clusteringCoffecient,
  };
}
```

2.2.2. Computing Average Degree

```
function calculateAvgDegree(nodes, edges) {
  let degreeSum = 0;
  nodes.forEach((node) => {
    let degree = edges.filter(
      (edge) => edge.from === node || edge.to === node
    ).length;
    degreeSum += degree;
  });
  return (degreeSum / nodes.length).toFixed(2);
}
```

2.2.3. Computing Graph Density

```
function calculateDensity(noOfNodes, noOfEdges) {
  if (noOfNodes === 0) {return 0;}
else {
    return (density = (2 * noOfEdges) / (noOfNodes * (noOfNodes - 1))).toFixed(
      2
    );
  }
}
```

2.2.4. Computing Graph Diameter

```
class Graph {

  constructor() {
      this.nodes = new Map();
  }

  addNode(name) {
    this.nodes.set(name, new Map());
  }

  addEdge(source, destination, weight) {
    this.nodes.get(source).set(destination, weight);
    this.nodes.get(destination).set(source, weight);
  }

  dijkstra(startNode) {
    const distances = new Map(); const visited = new Set();
    const previousNodes = new Map();
    for (const node of this.nodes.keys()) {
      distances.set(node, Infinity);
    }
    distances.set(startNode, 0);
    while (visited.size < this.nodes.size) {
      const currentNode = this.getMinDistanceNode(distances, visited);
      visited.add(currentNode);
      for (const [neighbor, weight] of this.nodes.get(currentNode)) {
        const distance = distances.get(currentNode) + weight;
        if (distance < distances.get(neighbor)) {
          distances.set(neighbor, distance);
          previousNodes.set(neighbor, currentNode);
        }
      }
    }
    return { distances, previousNodes };
  }
  getMinDistanceNode(distances, visited) {
    let minDistance = Infinity; let minNode = null;
    for (const [node, distance] of distances) {
      if (!visited.has(node) && distance < minDistance) {
        minDistance = distance;
        minNode = node;
      }
    }
    return minNode;
  }
```

```
  shortestPath(startNode, endNode) {
    const { distances, previousNodes } = this.dijkstra(startNode);
    const path = [endNode];
    let currentNode = endNode;

    while (currentNode !== startNode) {
      currentNode = previousNodes.get(currentNode);
      path.unshift(currentNode);
    }
    return { path, distance: distances.get(endNode) };
  }
}

function longestShortestPath(nodes, edges) {
  let currDistance = 0;
  let currPath = "";
  const graph = new Graph();

  nodes.forEach((node) => {
    graph.addNode(node);
  });
  edges.forEach((edge) => {
    graph.addEdge(edge.from, edge.to, parseFloat(edge.distance));
  });
  nodes.forEach((node) => {
    nodes.forEach((otherNode) => {
      if (node !== otherNode) {
        const { path, distance } = graph.shortestPath(node, otherNode);
        if (currDistance < distance) {
          currDistance = distance;
          currPath = path;
        }
      }
    });
  });

  currPath = currPath.join(" => ");
  return [currDistance, currPath];
}
```

2.2.5. Computing Graph Average Clustering Coefficient

```
function clusteringCoff(nodes, edges) {
  let numNode = nodes.length;
  let totalCi = 0;
  nodes.forEach((nodeId) => {
    totalCi += findClusteringCoefficient(edges, nodeId);
```

```
  });
  return (totalCi / numNode).toFixed(2);
}
function findClusteringCoefficient(edges, nodeId) {
  const neighbors = [];
  const neighborConnection = [];
  edges.forEach((edge) => {
    if (edge.from === nodeId) {
      neighbors.push(edge.to);
    }
    if (edge.to === nodeId) {
      neighbors.push(edge.from);
    }
  });
  const numNeighbors = neighbors.length;
  if (numNeighbors <= 1) {
    return 0;
  }
  for (let i = 0; i < numNeighbors; i++) {
    for (let j = i + 1; j < numNeighbors; j++) {
      edges.forEach((edge) => {
        if (
          (neighbors[i] == edge.from && neighbors[j] == edge.to) ||
          (neighbors[i] == edge.to && neighbors[j] == edge.from)
        ) {
          neighborConnection.push(edge);
        }
      });
    }
  }
  let Ki = neighbors.length;
  let Ei = neighborConnection.length;
  const Ci = (2 * Ei) / (Ki * (Ki - 1));
  return Ci;
}
```

2.2.6. Computing Degree Distribution of Graph

```
function findDegreeDistribution(nodes, edges) {
  const numNode = nodes.length;
  const degreeFrequency = {};
  const degreeDistribution = {};
  nodes.forEach((node) => {
    let degree = edges.filter(
      (edge) => edge.from === node.toString() || edge.to === node.toString()
    ).length;
    if (degreeFrequency[degree]) {
```

```
      degreeFrequency[degree] += 1;
    } else {
      degreeFrequency[degree] = 1;
    }
  });
  Object.keys(degreeFrequency).forEach((key) => {
    degreeDistribution[key] = degreeFrequency[key] / numNode;
  });
  return degreeDistribution;
}
function drawDegreeDistributionGraph(nodes, edges, Q) {
  const degreeDistribution = findDegreeDistribution(nodes, edges);
  let X = Object.keys(degreeDistribution);
  let Y = [];
  X.forEach((key) => {
    Y.push(degreeDistribution[key]);
  });
drawGraph(
      X,
      Y,
      `Degree Distribution Graph of${Q}`,
      "Degree (K)",
      "Fraction of nodes P(k)"
    );
}
```

2.2.7. Creating Html Content to display graph and property

```
function createHtmlContent(nodes, edges, property, type) {
  return `
<!DOCTYPE html>
<html>
<head>
  <title>Graph Visualization</title>
  <style>
    #network-container {
      display: flex;
      width: 100%;
      height: 85vh;
      border: 2px solid lightgray;
    }
    #property {
      margin-left: 50px;
      font-size: 20px;
      font-weight: 300;
    }
```

```
    #nav-bar {
      width: 100%;
      height: 60px;
    }
    h1 {
      text-align: center;
    }
  </style>
</head>
<body>
  <div id="nav-bar">
    <h1>Network Graph ${type}</h1>
  </div>
  <div id="network-container">
     <div id="property">
      <h2>Properties</h2>
      <p>No of Nodes: ${property.noOfNodes}</p>
      <p>No of Edges:  ${property.noOfEdges}</p>
      <p>Average Degree: ${property.avgDegree}</p>
      <p>Network Density: ${property.density}</p>
      <p>Most Distant Nodes: ${property.mostDistant}</p>
      <p>longest Shortest Path: ${property.shortestPath}</p>
      <p>Diameter: ${property.diameter}</p>
      <p>Clustering Coffecient: ${property.clusteringCoffecient}</p>
    </div>
    <div id="graph-container"></div>
  </div>
  <script src="https://unpkg.com/vis-network/standalone/umd/vis-
network.min.js"></script>
  <script>
    const container = document.getElementById('graph-container');
    const data = {
      nodes: ${JSON.stringify(
        nodes.map((node) => ({ id: node, label: node }))
      )},
      edges: ${JSON.stringify(edges)}
    };
    const options = { physics: false};
    new vis.Network(container, data, options);
  </script>
</body>
</html>
`;
}
```

3. Results and Findings:

Our objective was to download a network dataset from the network repository and import a small graph with approx. 50 nodes and fewer than 1000 edges in question no1. And 5K nodes in question no 2 but my device could not handle 5k nodes so I have worked with 133 nodes for question no 2.

I choose to utilize the Animal Social Network Section of the repository to obtain an appropriate dataset. I have provided the link in the **reference section.**

Once the dataset was imported, I proceeded to extract data in required format and then computed all the necessary property as per question and then visualized the graph using vis.js library. The resulting graph with property and degree distribution graph are in **Output Section.**

Upon Visual inspection, we observed that the graph displayed the desired characteristics. The nodes represented entities, while the edges represented relationship between them.

To validate the accuracy of the property computed for the given dataset, I have compared it to a reference dataset **(chapter 1 figures).** The reference dataset provided a benchmark for evaluating the correctness of my methods to compute the property and visualize the graph.

Based on these results, I have successfully completed the task as mentioned in lab work 5.

4. Analysis: *(Reference to this analysis is from **Output Section**)*

Q2 (a) From the network properties, what can you say about the networks you have selected?

⇨ Based on the network properties that I have computed for the selected networks from the network repository, we can draw several conclusions about these networks:

- The number of nodes provides insight into the size or scale of the networks. Networks with a higher number of nodes tend to be more extensive and can potentially represent complex system or interactions.
- The number of edges reflects the level of connectivity within the networks. Networks with a greater number of edges indicates stronger relationships or interactions between the nodes.
- The average Density of a network quantifies the proportion of actual edges to the total number of possible edges. Higher density values suggest that the nodes in the network are more interconnected.
- The average Degree of a network measures the average number of edges per node. It provides an indication of the overall connectivity or the average no. of relationships each node has.
- The diameter of a network represents the maximum shortest path length between any two nodes in the network. It provides a measure of the network's overall size or how spread out the nodes are. Larger diameters suggest greater distances between nodes.
- The clustering coefficient measures the extent to which nodes in a network tend to form clusters or groups. A higher clustering coefficient indicates that nodes within the network have a tendency to form tightly interconnected communities or subgraphs.
- The degree distribution describes the frequency of nodes with a given degree in the networks. It provides insights into the overall structure of the network and can help identify patterns such as power-law distribution (indicating scale-free network) or Gaussian distribution (indicating more homogeneous networks).

Q2 (b) Did you find any pattern in the degree distributions of the networks? In any case, can you come to any conclusion about the networks from their degree distribution?

⇨ After looking at the nature of each graph. These are the specific thing that stands out in each graph.

➢ **Graph 1**:

- The majority of nodes in Graph 1 have degrees between 20 and 110.
- There is a peak around degrees 99 and 103, indicating a group of nodes with high connectivity.
- The distribution has a long tail with a few nodes having degrees greater than 100.

Similar nature has been observed in all other graph as well.

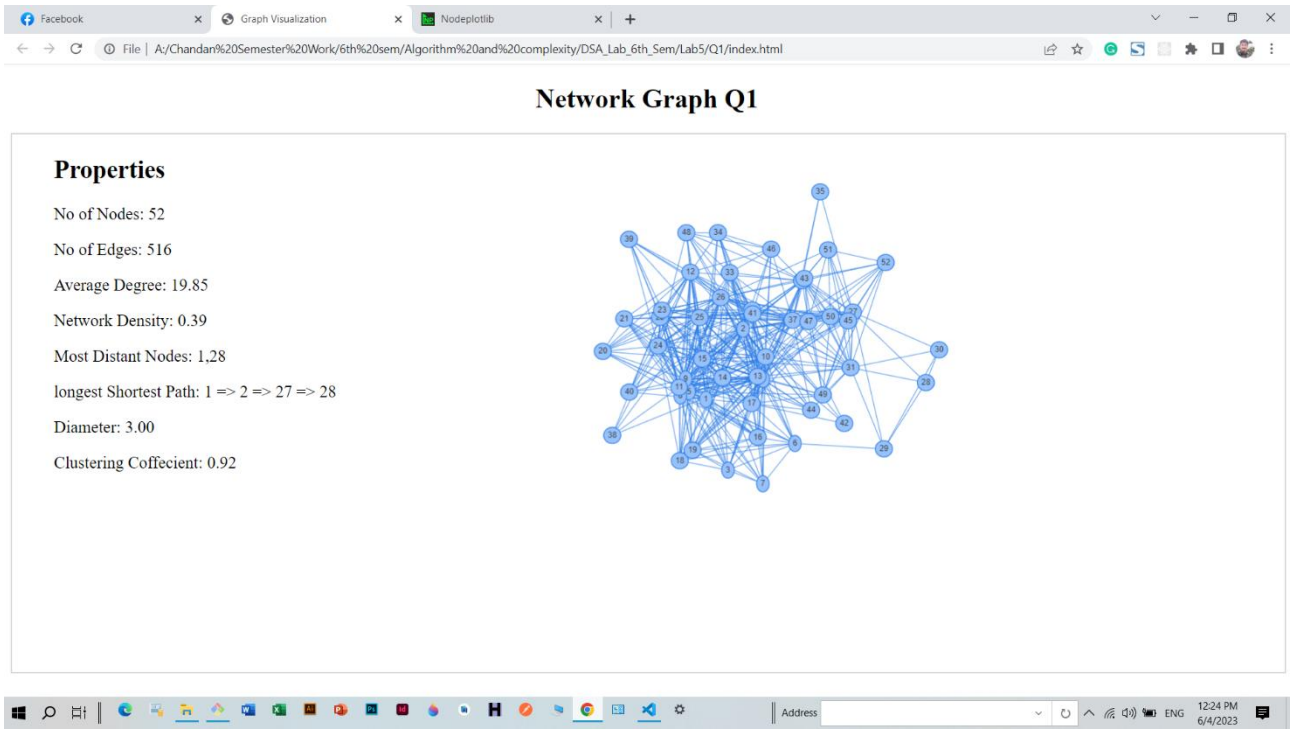
Therefore, from the degree distributions, we can observe that all five networks exhibit similar patterns with a majority of nodes having degrees within a specific range. Additionally, there are peaks in the distributions, indicating the presence of highly connected nodes within each network. The long tail is in the distributions suggest the existence of a few nodes with extremely high degrees.

Based on the degree distribution alone, we can conclude that the networks are likely to be **scale-free networks**, as they exhibit a **power-law distribution** with a few nodes having significantly higher degrees compared to the reset of the network. This indicates the presences of **hubs or highly influential nodes** in the network.

5. Conclusion:

I have successfully imported several graphs and computed various network properties, including the number of nodes, number of edges, average density, average degree, clustering coefficient, diameter, and degree distribution. The findings revealed that the selected networks exhibited specific characteristics, such as size, connectivity, and clustering tendencies. These insights contribute to our understanding of the networks and provide a foundation for further analysis. Overall, our analysis provides valuable insights into the nature of the networks under study.

6. Output:



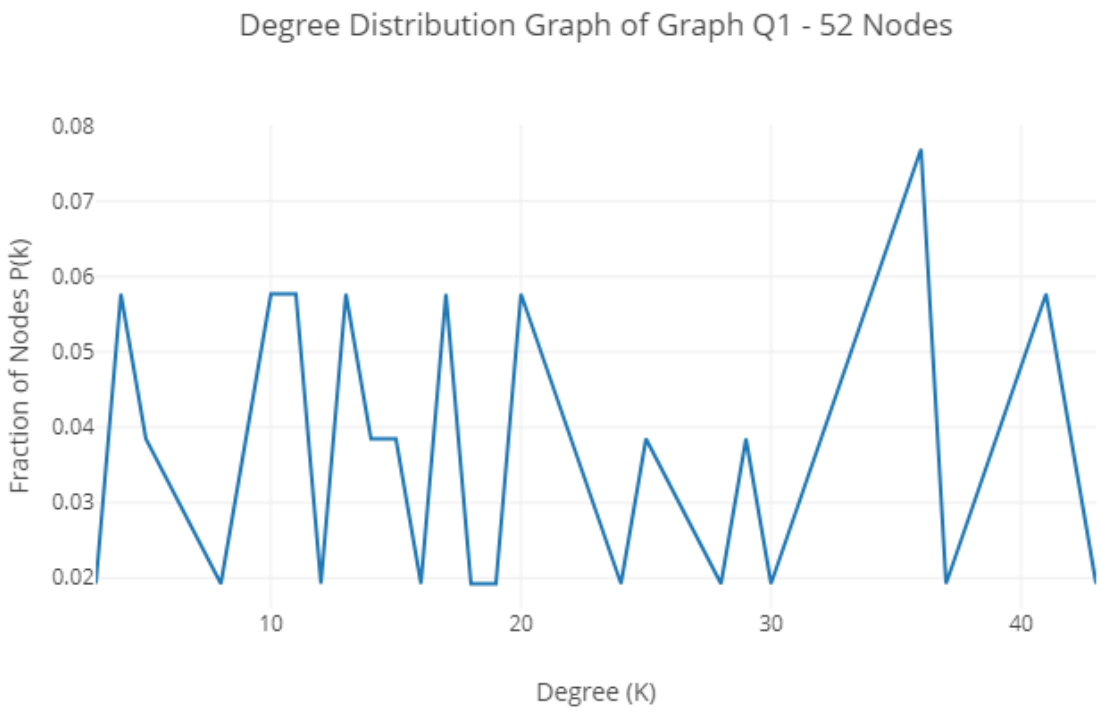
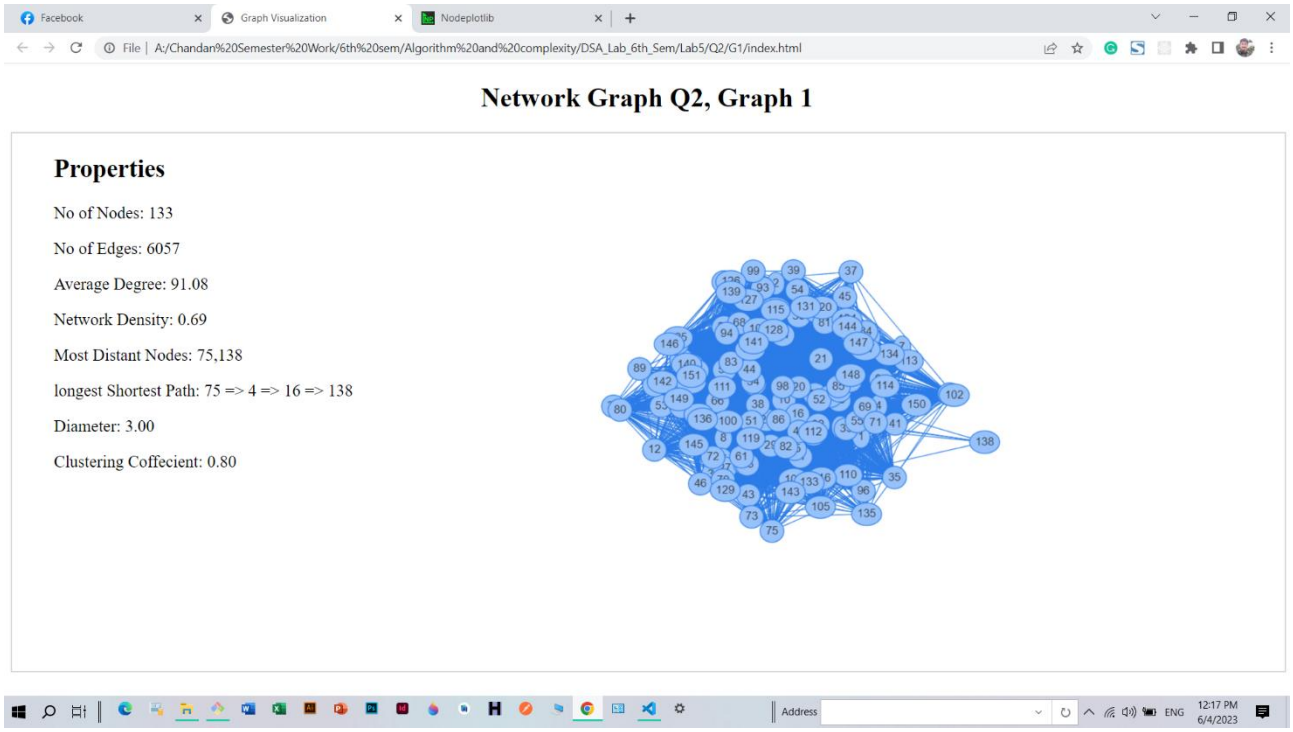


Figure 4 Network Graph Q1 with Properties



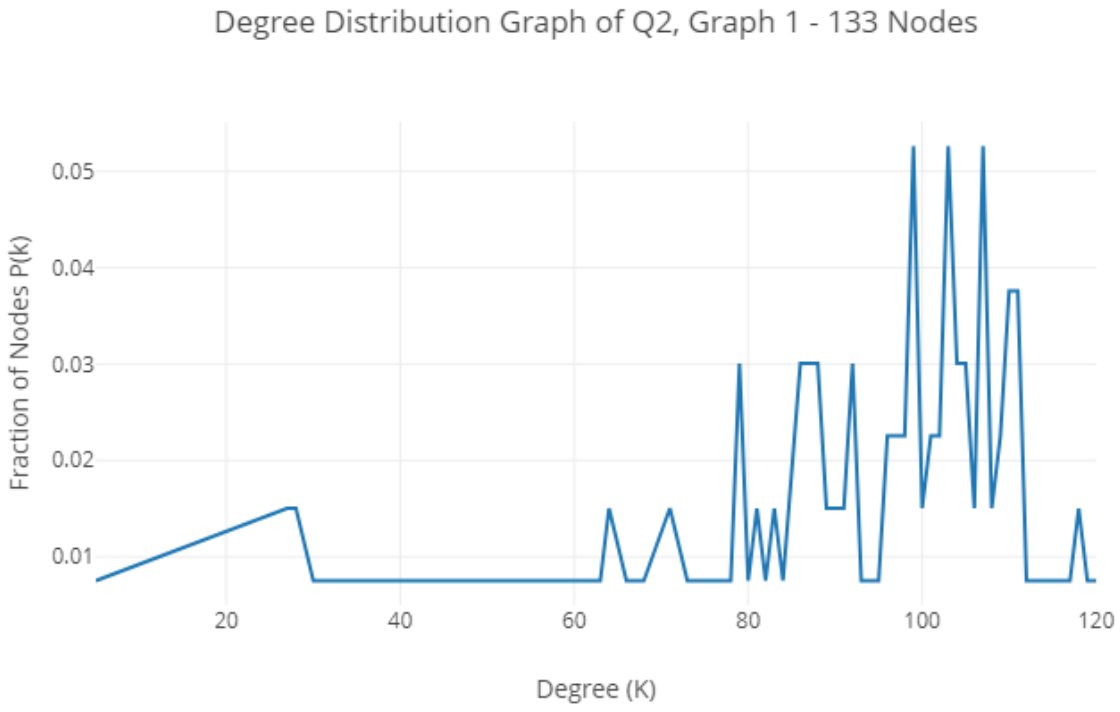
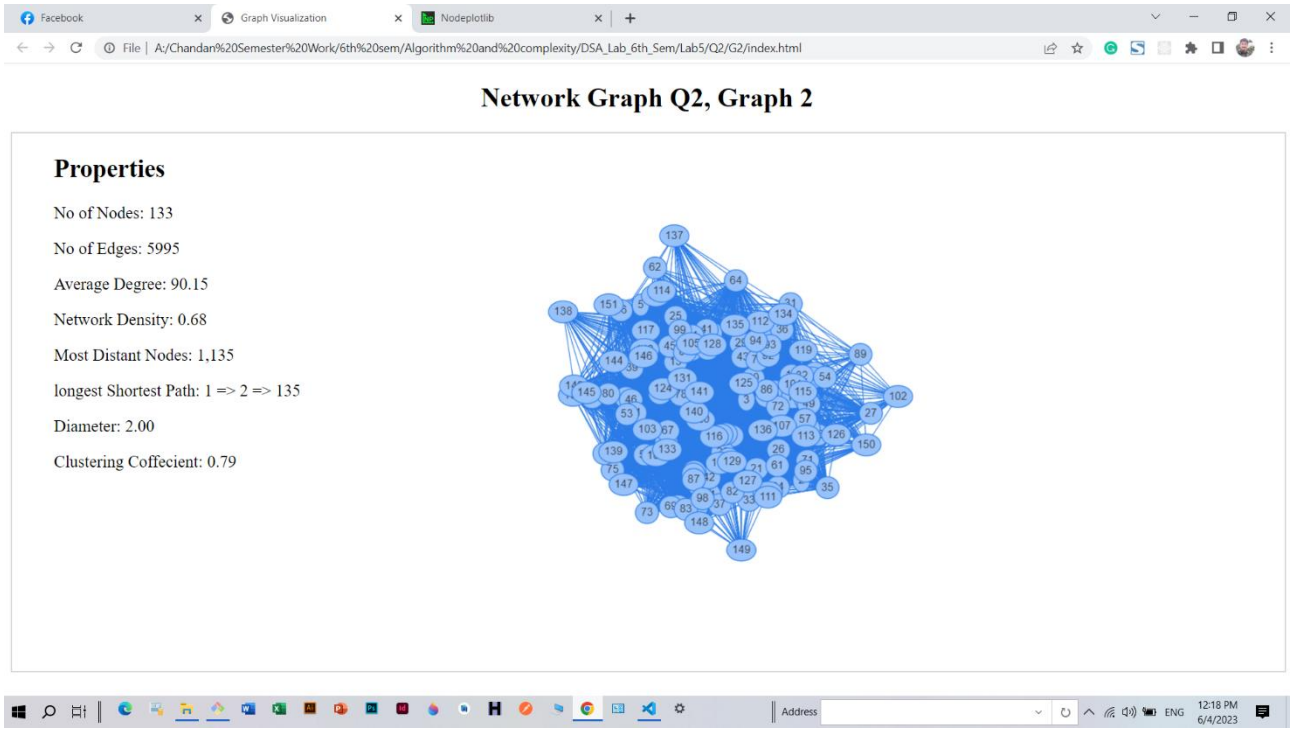


Figure 5 Network Graph Q2 G1 with Properties



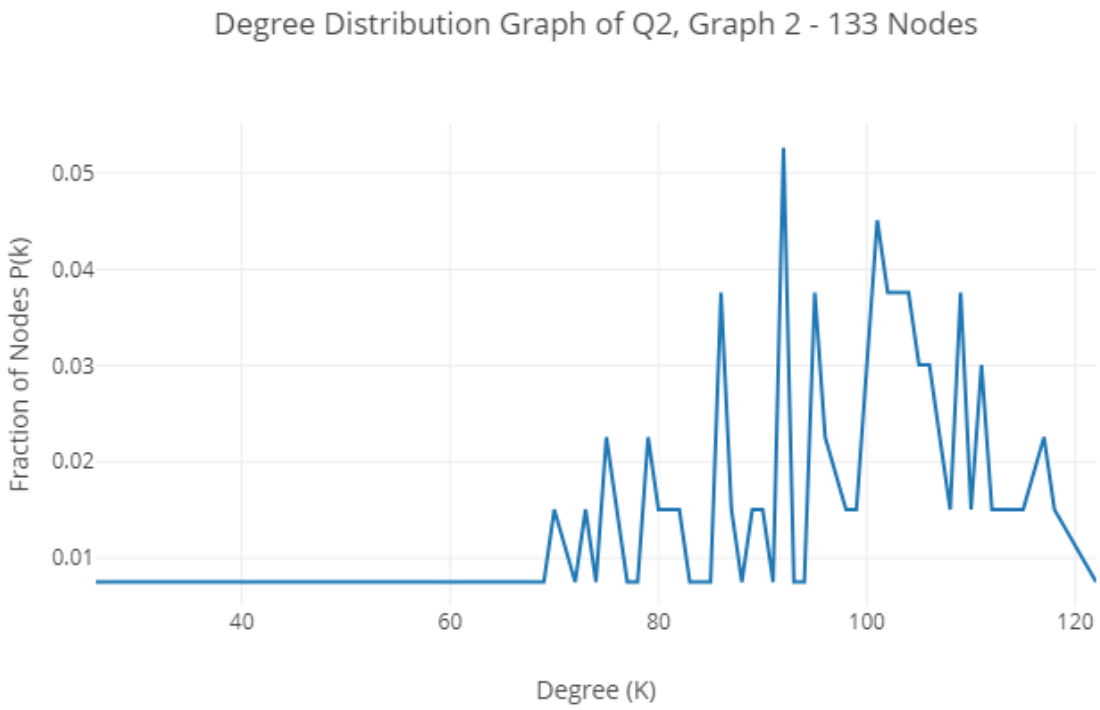
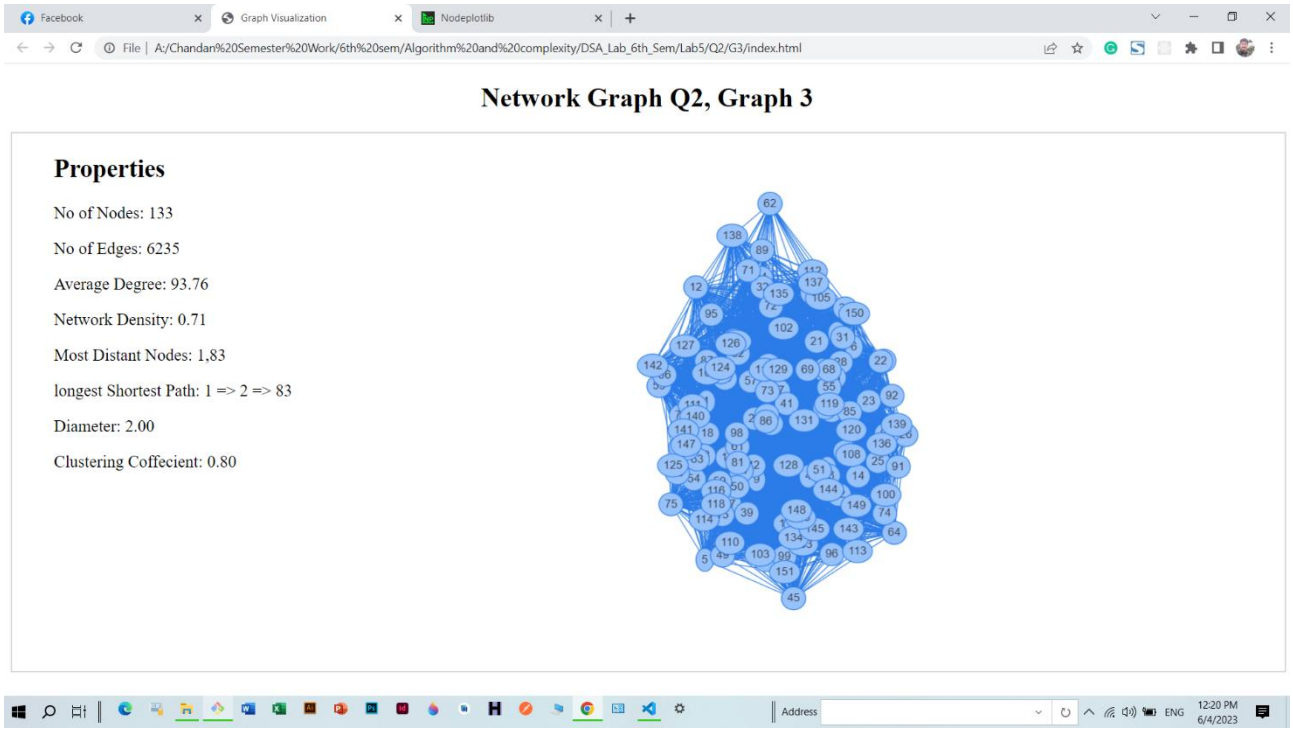


Figure 6 Network Graph Q2 G2 with Properties



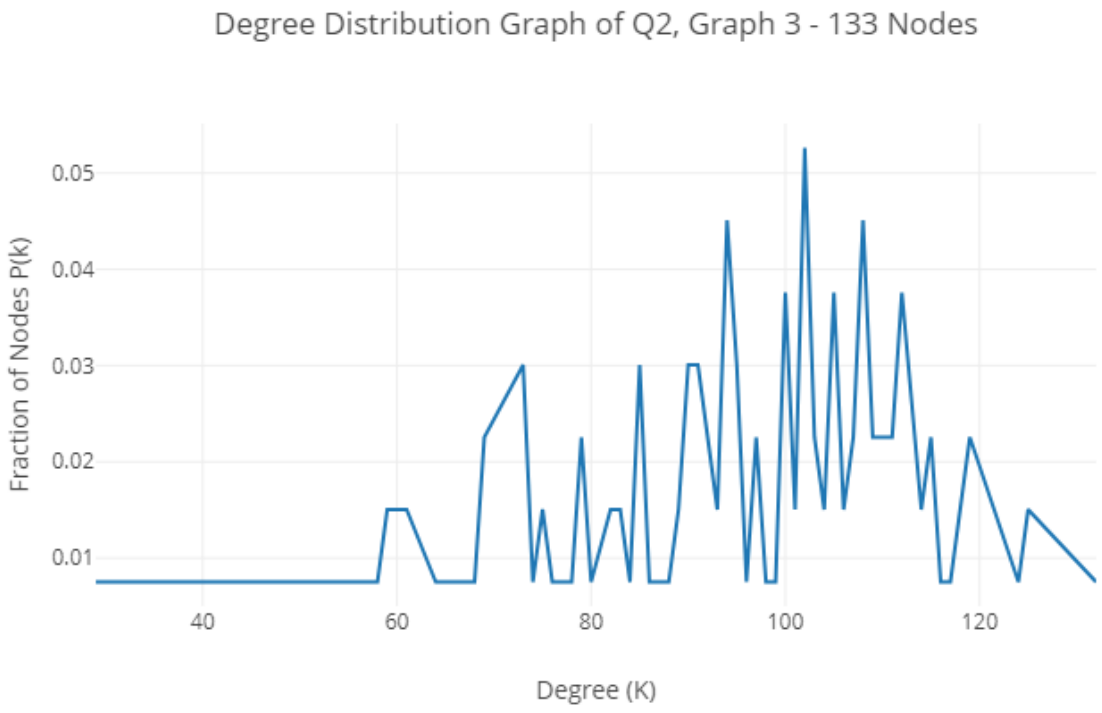
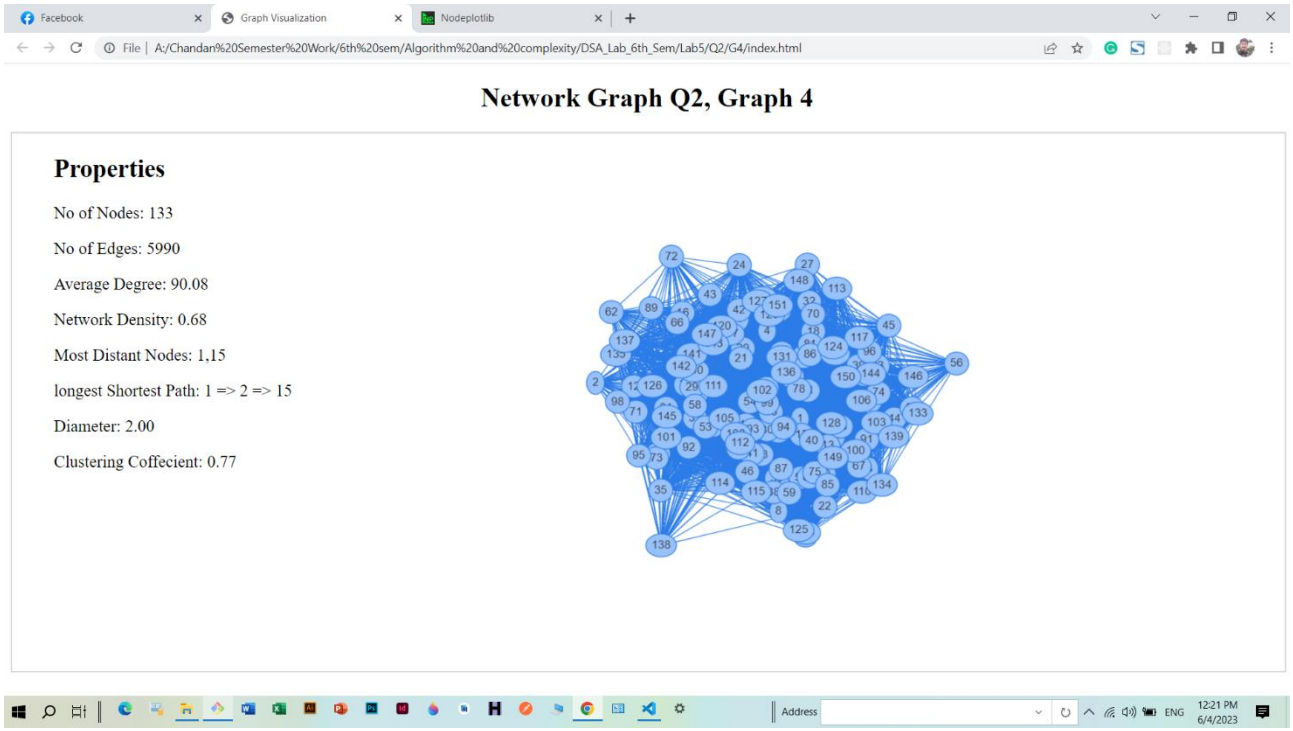


Figure 7 Network Graph Q2 G3 with Properties



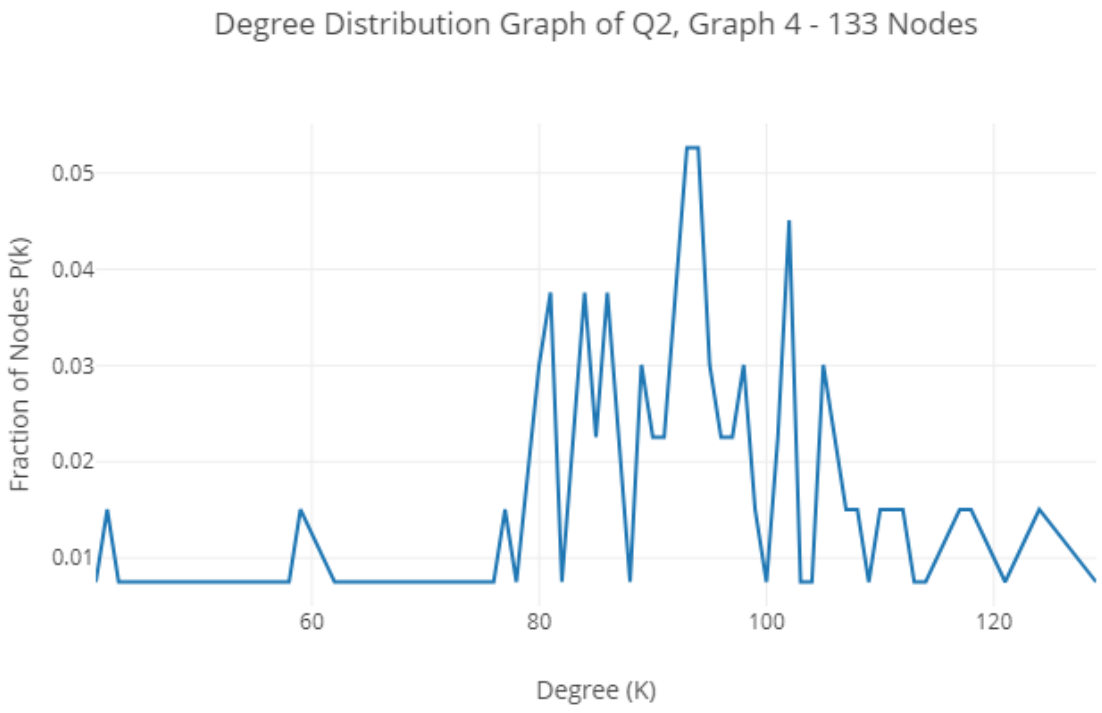
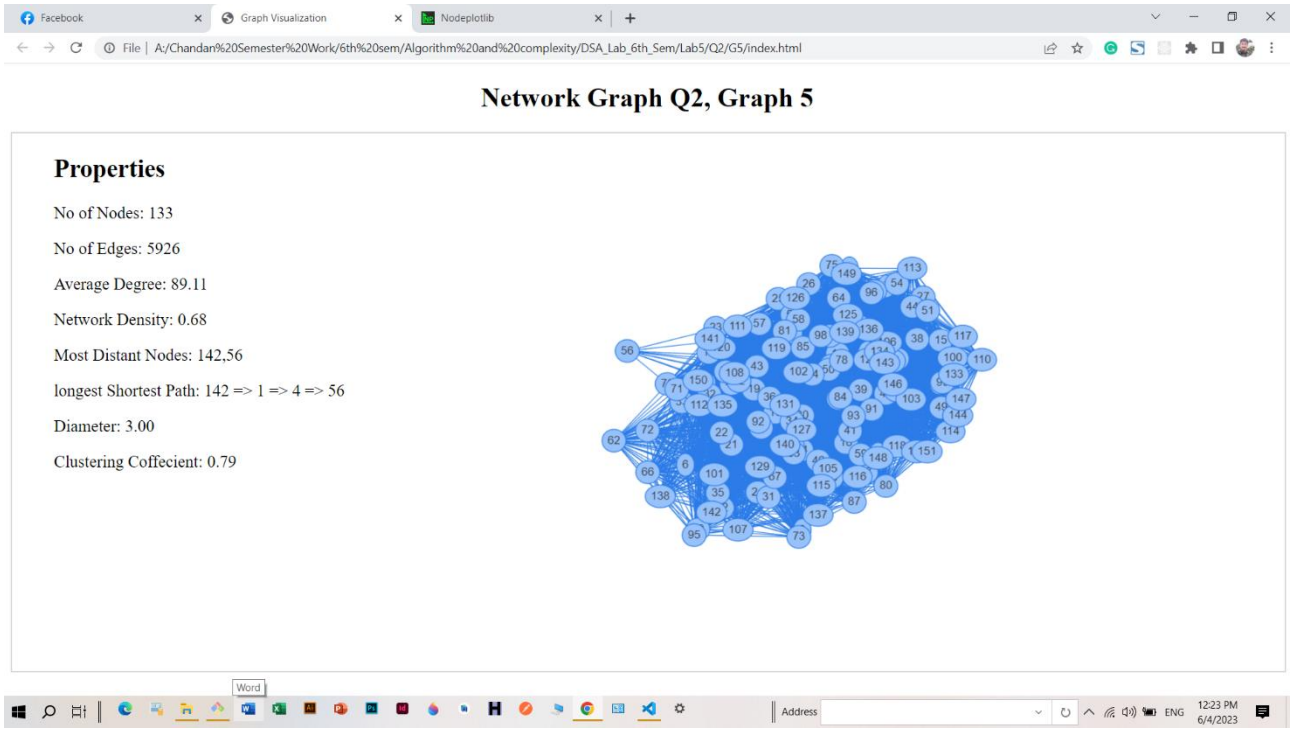


Figure 8 Network Graph Q2 G4 with Properties



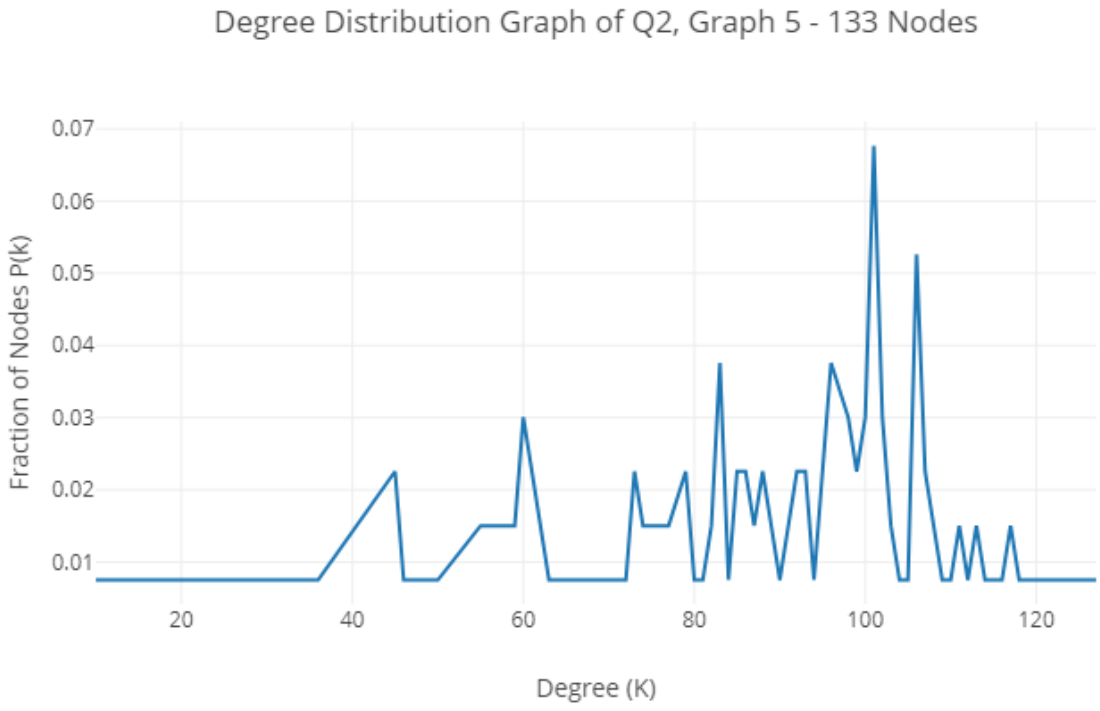


Figure 9 Network Graph Q2 G5 with Properties

7. References:

https://networkrepository.com/

https://networkrepository.com/asn.php

https://visjs.org/

https://visjs.github.io/vis-network/docs/network/