Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

DepartmentofComputer ScienceandEngineering(DataScience)

Subject: Social Network Analysis Laboratory (DJ19DSL8014)

Bhuvi Ghosh
60009210191

AY: 2024-25

Experiment 3

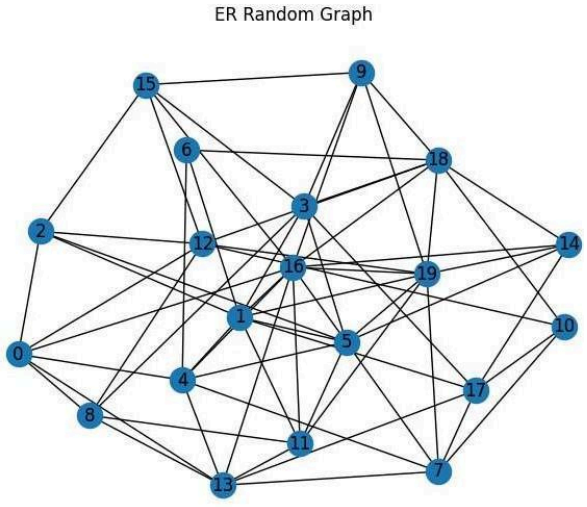
Aim: Implementation of The Erdős-Rényi (ER) random network growth model.

Theory:

1. Introduction

The Erdős-Rényi (ER) model, named after mathematicians Paul Erdős and AlfrédRényi, is central to the field of network science, offering key insights into the random processes that can generate network structures. Unlike the Barabasi-Albert model, which is characterized by the scale-free property and preferential attachment, the ER model is defined by its simplicity and the random nature of connections between nodes. This model is particularly useful in exploring the properties of networks where the probability of connection between any two nodes is uniform.

The Erdős-Rényi model comes in two variants:



G(n, p): A graph of n nodes where each pair of nodes is connected with probability p.

G(n, M): A graph of n nodes and M randomly placed edges.

Implementation Steps for the Erdős-Rényi (G(n, p)) Model

- Implementation Steps for the Erdős-Rényi (G(n, p)) Model
- Initialize an Empty Network: Begin with n nodes but no edges.
- Random Edge Creation: For every pair of distinct nodes, add an edge between them with a fixed probability p.
- Iterative Process: The process of considering pairs and deciding on edge creation is done once for each unique pair.
- Analysis: After the network is generated, analyze its properties such as degree distribution, clustering coefficient, diameter, and path lengths.

Lab assignment:

1. Create a scale-free network using the Erdos-Renyi network.
2. Degree Distribution vs. Degree (k)

   X-axis: Degree k (number of connections a node has)

   Y-axis: Frequency (number of nodes having degree k)

   This plot is called a degree histogram and shows how the degrees are distributed across the nodes.

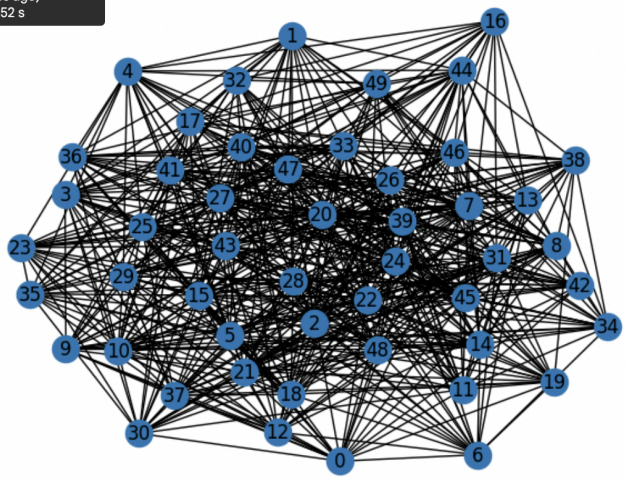
3. Calculate degree distribution, Diameter, Average clustering coefficient, Find out effect of changing pattern of N and P on above parameters and its interpretation.

```
[17] import networkx as nx
     import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
```

```
G= nx.erdos_renyi_graph(50,0.5)
nx.draw(G, with_labels=True)
```

Run cell (⌘/Ctrl+Enter)
cell executed since last change

executed by bhuvi ghosh
23:04 (0 minutes ago)
executed in 0.552 s



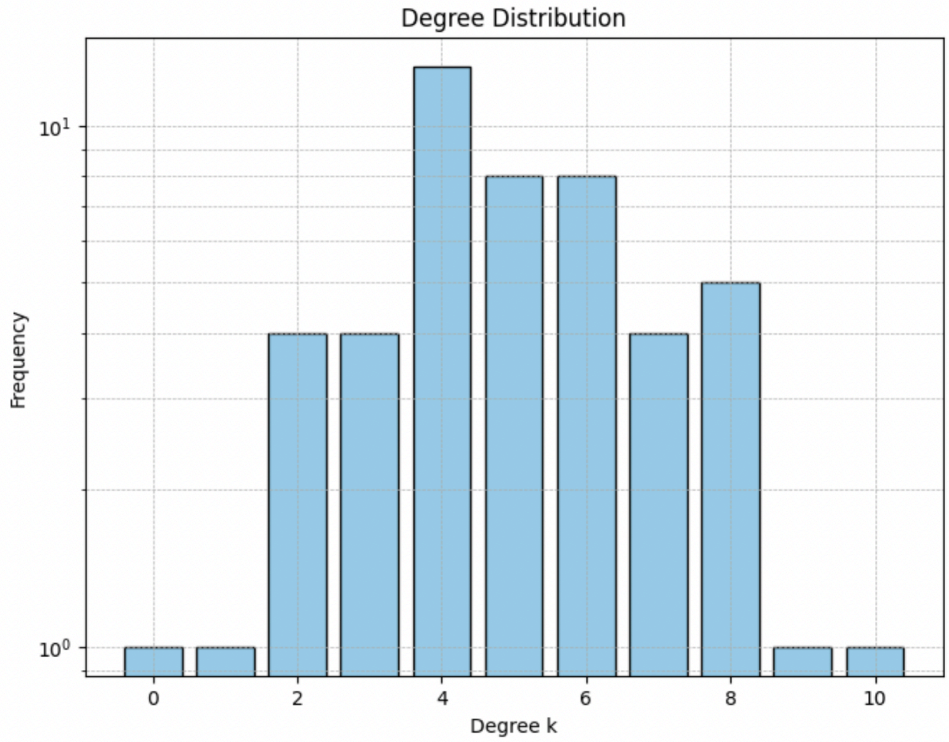
[8]
```
def generate_er_graph(N, P):
    return nx.erdos_renyi_graph(N, P)

def degree_distribution(G):
    degrees = [d for _, d in G.degree()]
    unique_degrees, counts = np.unique(degrees, return_counts=True)
    return unique_degrees, counts

def graph_metrics(G):
    avg_clustering = nx.average_clustering(G)
    if nx.is_connected(G):
        diameter = nx.diameter(G)
        avg_path_length = nx.average_shortest_path_length(G)
    else:
        diameter = float('inf')
        avg_path_length = float('inf')
    return avg_clustering, diameter, avg_path_length
```

[10]
```
def generate_erdos_renyi(N, P):
    G = nx.erdos_renyi_graph(N, P)
    return G

def compute_metrics(G):
    degree_sequence = [d for n, d in G.degree()]
    degree_counts = {k: degree_sequence.count(k) for k in set(degree_sequence)}
    diameter = nx.diameter(G) if nx.is_connected(G) else float('inf')
    avg_clustering = nx.average_clustering(G)
    return degree_counts, diameter, avg_clustering
```

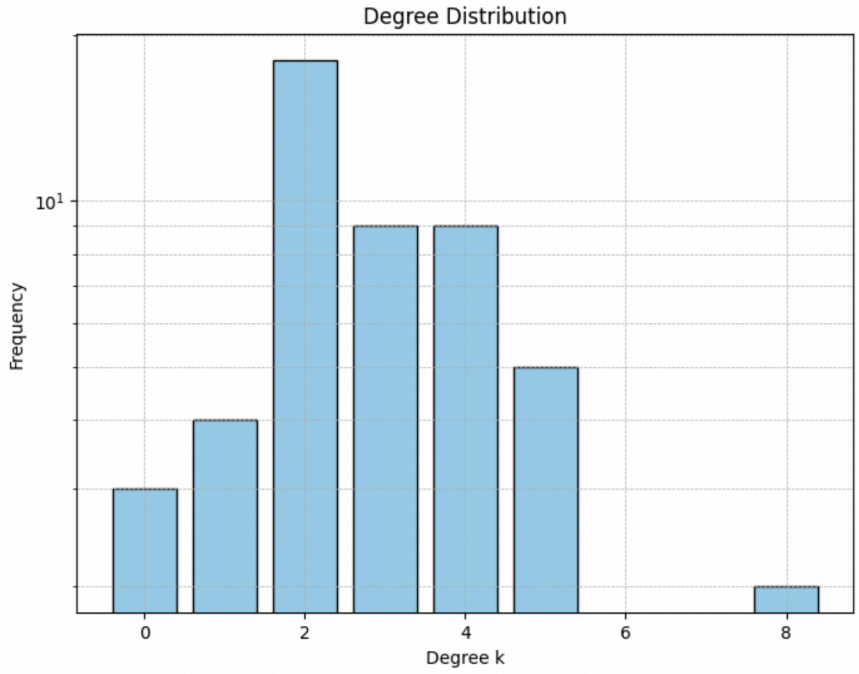
```
[11] def plot_degree_distribution(degree_counts):
         degrees, counts = zip(*sorted(degree_counts.items()))

         plt.figure(figsize=(8, 6))
         plt.bar(degrees, counts, color='skyblue', edgecolor='black')
         plt.xlabel('Degree k')
         plt.ylabel('Frequency')
         plt.title('Degree Distribution')
         plt.yscale('log')  # Log scale to observe patterns in large networks
         plt.grid(True, which='both', linestyle='--', linewidth=0.5)
         plt.show()
```

```
N_values = [50, 100, 200]
P_values = [0.1, 0.05, 0.01]

for N in N_values:
    for P in P_values:
        print(f"\nErdős-Rényi Network (N={N}, P={P}):")
        G = generate_erdos_renyi(N, P)
        degree_counts, diameter, avg_clustering = compute_metrics(G)

        print(f"Diameter: {diameter}")
        print(f"Average Clustering Coefficient: {avg_clustering:.4f}")

        plot_degree_distribution(degree_counts)
```

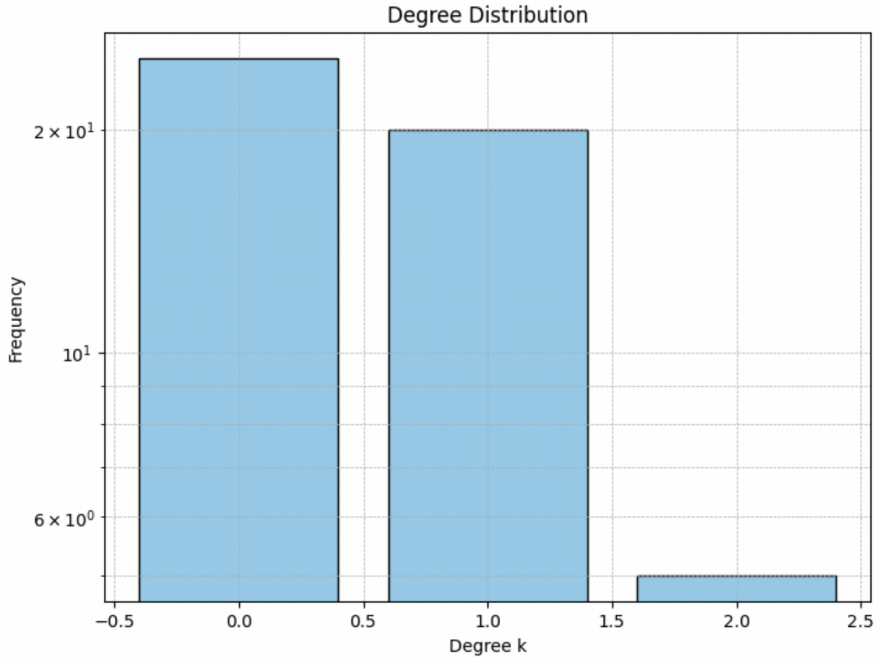
```
Erdős-Rényi Network (N=50, P=0.1):
Diameter: inf
Average Clustering Coefficient: 0.1102
```



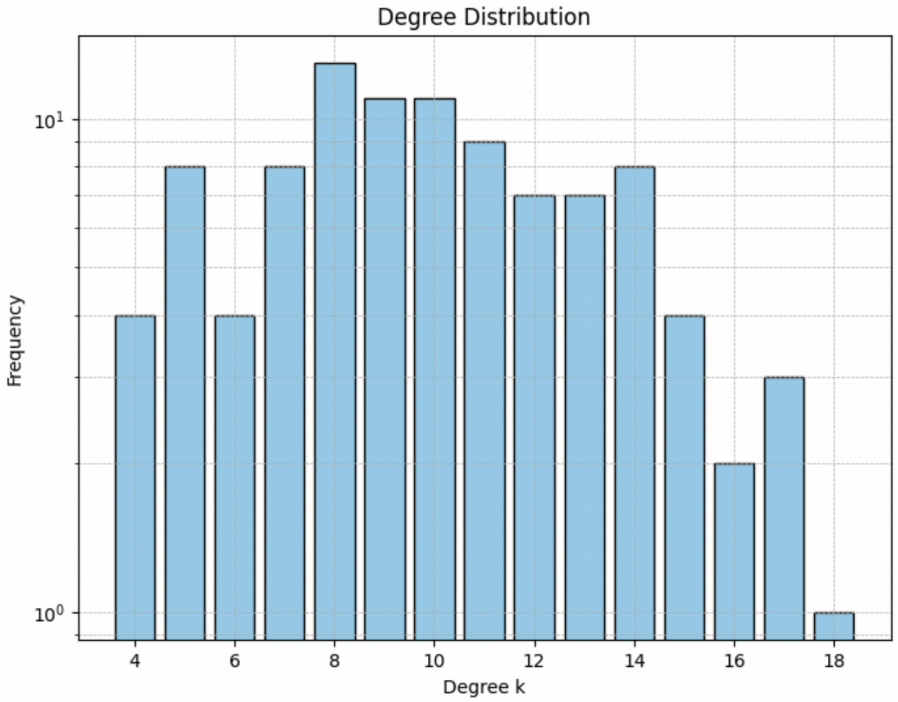
```
Erdős-Rényi Network (N=50, P=0.05):
Diameter: inf
Average Clustering Coefficient: 0.0241
```



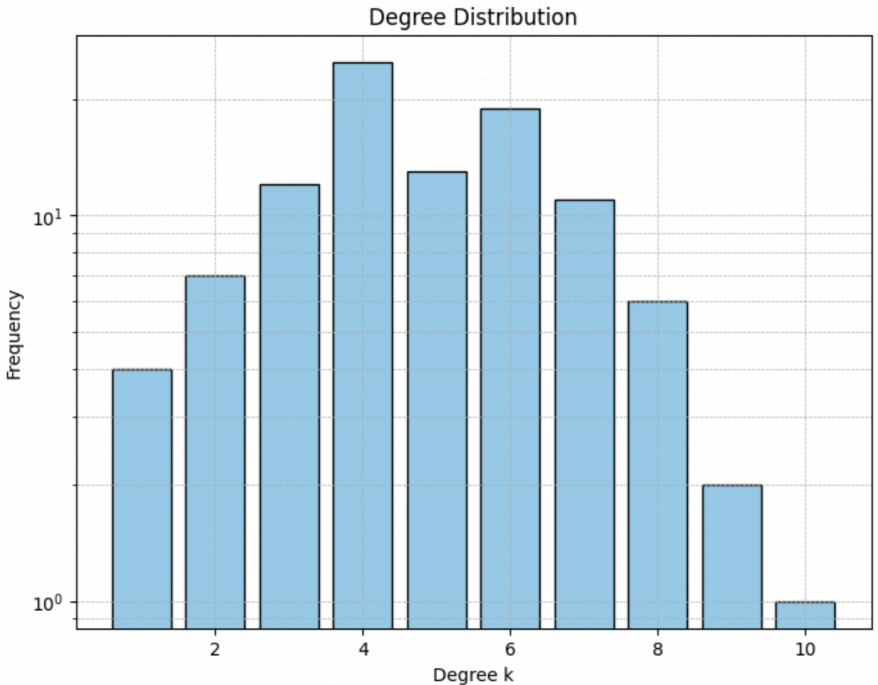

```
Erdős-Rényi Network (N=50, P=0.01):
Diameter: inf
Average Clustering Coefficient: 0.0000
```



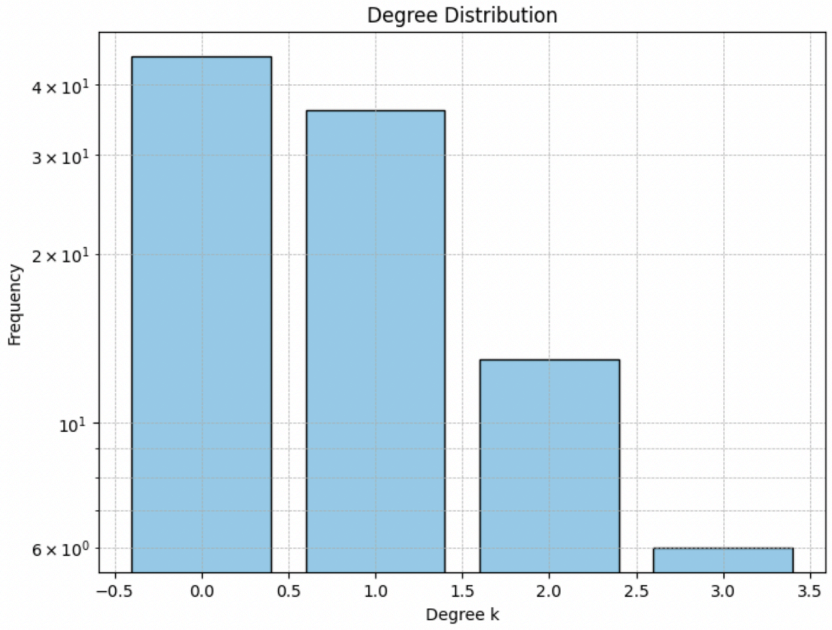
```
Erdős–Rényi Network (N=100, P=0.1):
Diameter: 4
Average Clustering Coefficient: 0.0901
```



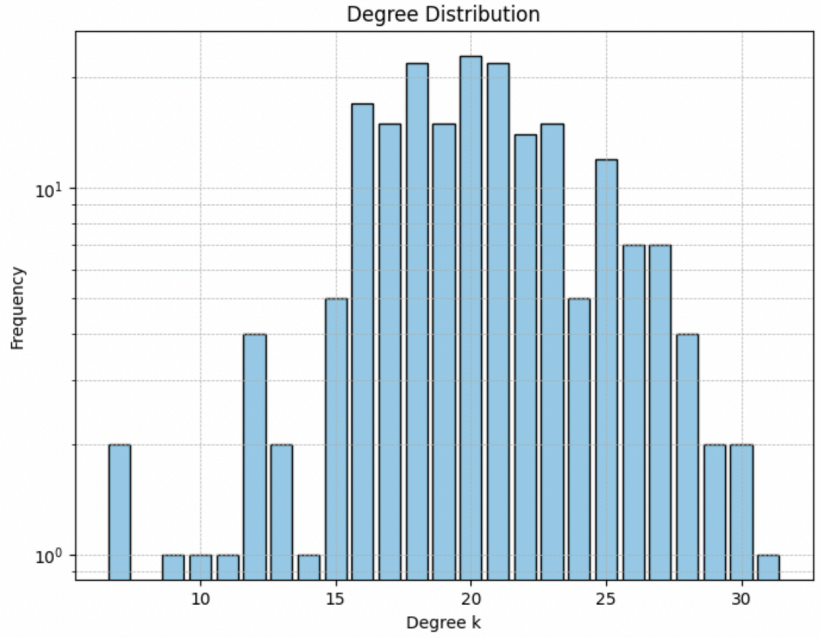

```
Erdős–Rényi Network (N=100, P=0.05):
Diameter: 6
Average Clustering Coefficient: 0.0435
```



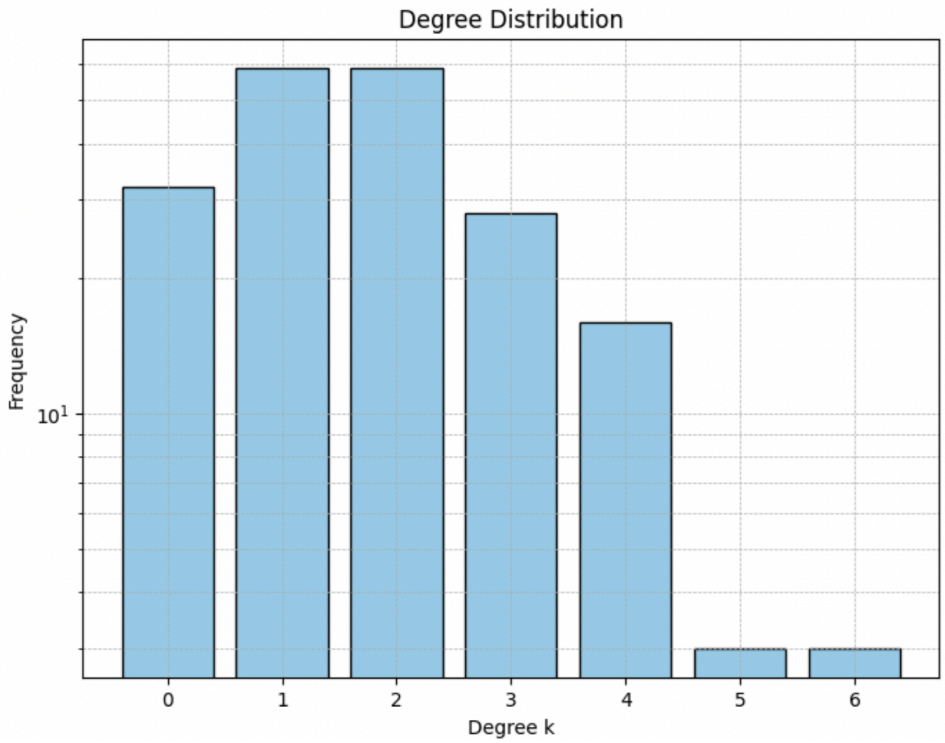
```
Erdős-Rényi Network (N=100, P=0.01):
Diameter: inf
Average Clustering Coefficient: 0.0000
```




```
Erdős-Rényi Network (N=200, P=0.1):
Diameter: 3
Average Clustering Coefficient: 0.0979
```



```
Erdős-Rényi Network (N=200, P=0.01):
Diameter: inf
Average Clustering Coefficient: 0.0000
```



```python
def graph_metrics(G):
    clustering = nx.average_clustering(G)
    if nx.is_connected(G):
        diameter = nx.diameter(G)
        avg_path_length = nx.average_shortest_path_length(G)
    else:
        diameter = float('inf')
        avg_path_length = float('inf')
    return clustering, diameter, avg_path_length

N_values = [10, 20, 50]
P_values = [0.1, 0.5]

results = []

for N in N_values:
    for P in P_values:
        G = generate_er_graph(N, P)
        degrees, counts = degree_distribution(G)
        clustering, diameter, avg_path_length = graph_metrics(G)
        degree_dist = dict(zip(degrees, counts))
        results.append([N, P, clustering, diameter, avg_path_length, degree_dist])
df = pd.DataFrame(results, columns=["N", "P", "Avg Clustering Coefficient", "Diameter", "Avg Path Length", "Degree Distribution"])
print(df)
```

Effect of Changing N and P on Graph Parameters

N	P	Degree Distribution	Clustering Coefficient	Diameter	Avg Path Length
10	0.1	{0: 2, 1: 5, 2: 2, 3: 1}	0.000000	inf	inf
10	0.5	{1: 1, 4: 3, 5: 5, 6: 1}	0.266667	3.0	1.600000
20	0.1	{0: 4, 1: 8, 2: 5,	0.075000	inf	inf

		3: 2, 4: 1}			
20	0.5	{7: 6, 8: 3, 9: 2, 10: 2, 11: 4, 12: 2, 14: 1}	0.434592	2.0	1.521053
50	0.1	{2: 2, 3: 3, 4: 10, 5: 10, 6: 10, 7: 6, 8: 6, ...	0.077127	5.0	2.379592
50	0.5	{15: 1, 19: 1, 20: 1, 21: 3, 22: 7, 23: 5, 24:...	0.508569	2.0	1.491429

Conclusion on the Effect of Changing NNN and PPP on Graph Parameters

1. **Degree Distribution:**
   - For low PPP values (0.1), many nodes have very few connections, and some nodes may even be isolated (k=0k=0k=0). This results in a fragmented network.
   - For higher PPP values (0.5), the degree distribution shifts toward higher values, meaning more nodes have higher degrees, and the network becomes denser.
2. **Clustering Coefficient:**
   - When P=0.1P = 0.1P=0.1, the clustering coefficient is low because connections are sparse, and there are fewer closed triplets (triangles).
   - When P=0.5P = 0.5P=0.5, the clustering coefficient increases significantly because the probability of forming triangles is higher.
3. **Diameter and Average Path Length:**

- For low PPP, the network is often disconnected, leading to an infinite diameter and path length.
- For higher PPP, the diameter decreases, and the network exhibits the **small-world property** where most nodes are just a few steps apart.

4. **Impact of NNN:**
   - Increasing NNN at low PPP still results in a fragmented network with large diameters.
   - At higher PPP, increasing NNN results in a more connected and dense network, reducing the diameter and average path length.