CS 6385
ALGORITHMIC ASPECTS OF TELECOMMUNICATION NETWORKS

PROJECT - 1
IMPLEMENTATION OF K-CORE CLUSTER CONCEPT

SUBMITTED BY -
YASHWANTH DEVIREDDY
NetID : yxd210008

CONTENTS

ABSTRACT

In this project, we investigate the k-core clustering concept and also the relation between the number of cores and the edge probability of graphs. By definition of the k-core algorithm, it is the largest subset of nodes where each node has at least k neighbors from the subset. Using the algorithm, we try to analyze, in this project the randomly generated graphs with varying edge probabilities. And for each of the graphs, we calculate the core number (the largest k value) and the averaged result over 10 runs. We represent the relation between the core number and the edge probability visually by plotting the dependency diagrams. By these findings, we suggest that the core number is one of the useful measures for graph connectivity.

ALGORITHM

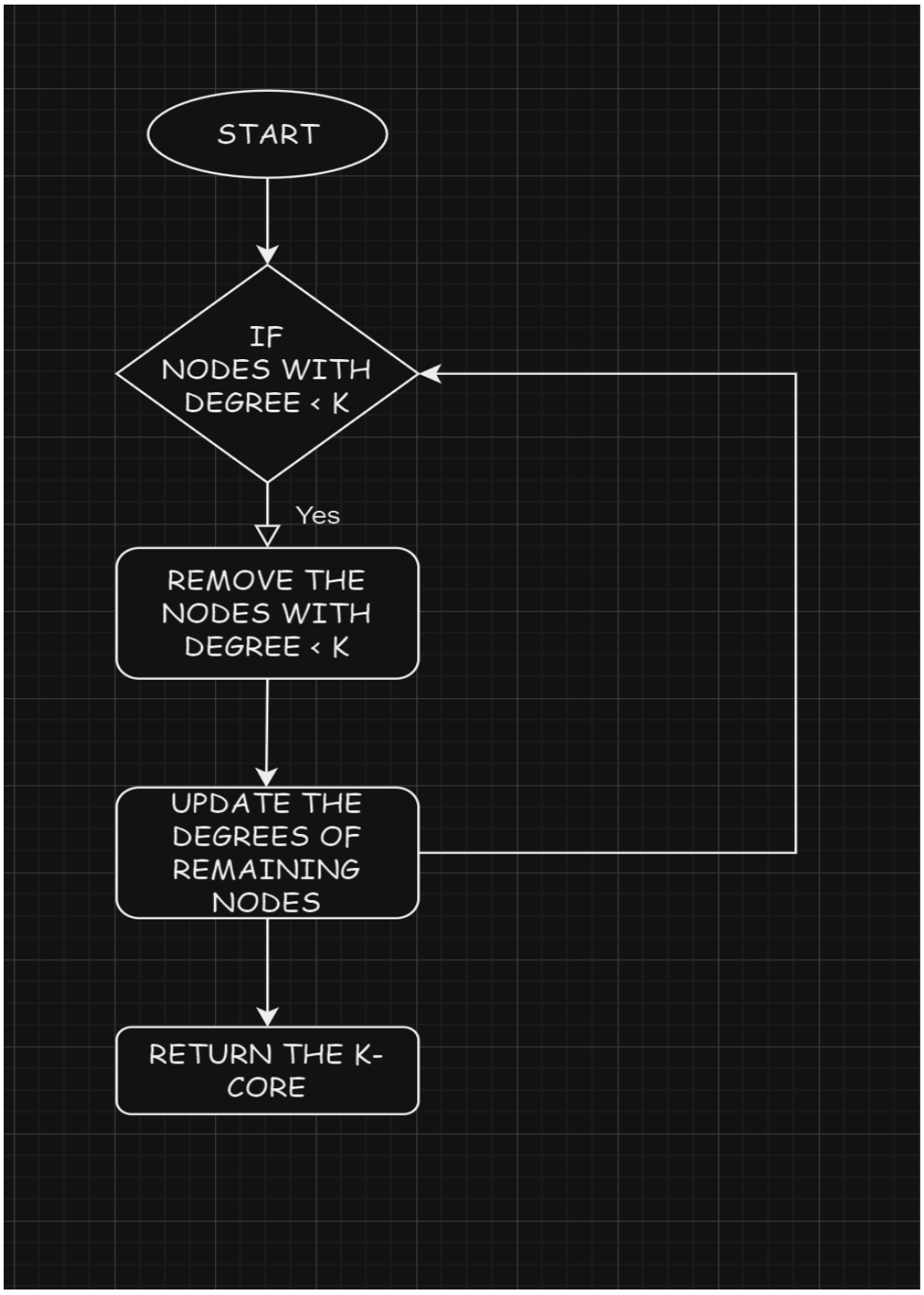
The algorithm to find the k-core of a graph:

1. Initialize the original graph as the k-core and start with it.
2. While there are nodes present in the k-core with degree less than k, do:
   - Remove the nodes with degree less than k from the graph
   - Update the degree of the other nodes by removing the edges connected to the nodes which are deleted.
3. The graph which we have now is the k-core and each node in this graph has >= k neighbors within the subset.

The basic flow and functionality of the program:

1. Take the input p (edge probability), where (0 < p < 1).
2. Generate a random graph with the specified 25 nodes in each. The edges between the nodes should be added based on the value p.
3. For the generated random graph, say G, calculate its core number using the k-core algorithm mentioned above.
4. Run the program 10 times generating a random graph everytime anc calculating the core number.
5. Average the core number we got for 10 times.
6. Display the randomly generated graphs with 3 different p values using the Networkx library.
7. Run the program for various p values, ranging from sparse to dense. Display the graph for the relation between p and average core number.

FLOW CHART


START
IF NODES WITH DEGREE < K
Yes
REMOVE THE NODES WITH DEGREE < K
UPDATE THE DEGREES OF REMAINING NODES
RETURN THE K-CORE

OUTPUT AND RESULTS

Task 1: Output (Average core number over 10 runs for different p values)

For p = 0.1

```
PS C:\Users\Yash> & C:/Users/Yash/AppData/Local/Programs/Python/Python39/python.exe c:/Users/Yash/Desktop
/CS6385_Project1_YXD210008.py
Enter p value:0.1
Average Core Number for p = 0.1 over 10 runs: 2.0
```

For p = 0.25

```
PS C:\Users\Yash> & C:/Users/Yash/AppData/Local/Programs/Python/Python39/python.exe c:/Users/Yash/Desktop
/CS6385_Project1_YXD210008.py
Enter p value:0.25
Average Core Number for p = 0.25 over 10 runs: 4.2
```

For p = 0.5

```
PS C:\Users\Yash> & C:/Users/Yash/AppData/Local/Programs/Python/Python39/python.exe c:/Users/Yash/Desktop
/CS6385_Project1_YXD210008.py
Enter p value:0.5
Average Core Number for p = 0.5 over 10 runs: 9.1
```

For p = 0.75

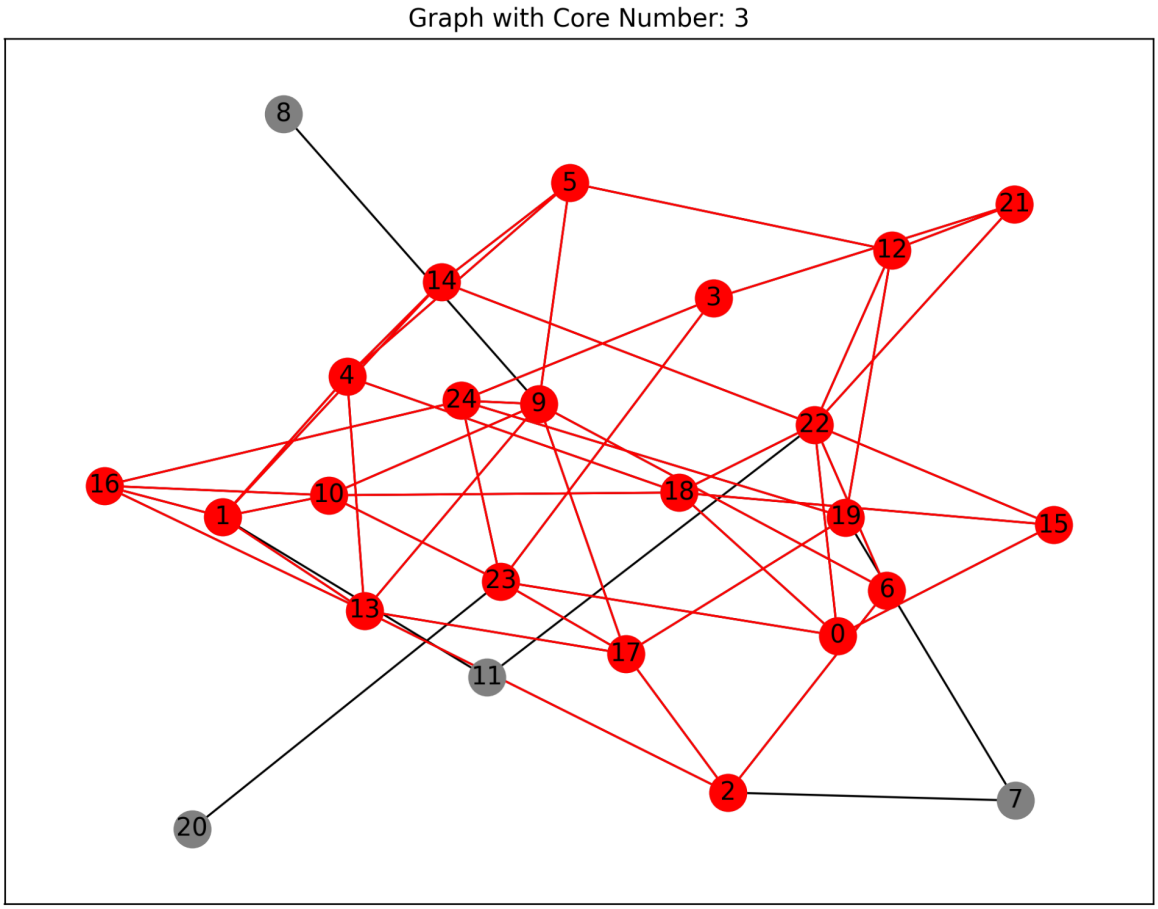
```
PS C:\Users\Yash> & C:/Users/Yash/AppData/Local/Programs/Python/Python39/python.exe
c:/Users/Yash/Desktop/CS6385_Project1_YXD210008.py
Enter p value:0.75
Average Core Number for p = 0.75 over 10 runs: 14.9
```

For p = 0.9

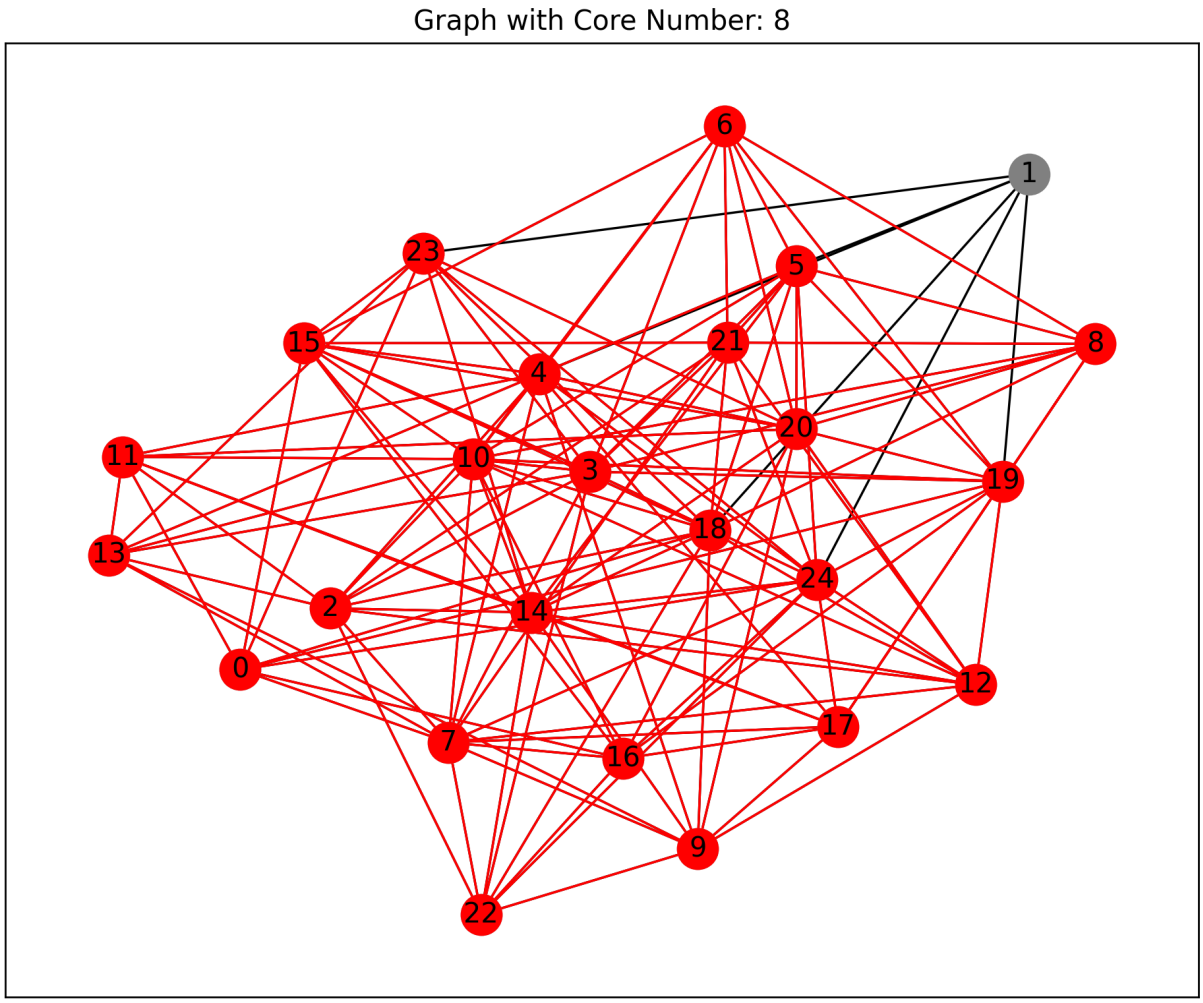
```
PS C:\Users\Yash> & C:/Users/Yash/AppData/Local/Programs/Python/Python39/python.exe
c:/Users/Yash/Desktop/CS6385_Project1_YXD210008.py
Enter p value:0.9
Average Core Number for p = 0.9 over 10 runs: 19.2
```

Task 2: Graphs for 3 different p values (one small, one medium, and one large value) sparse to dense.

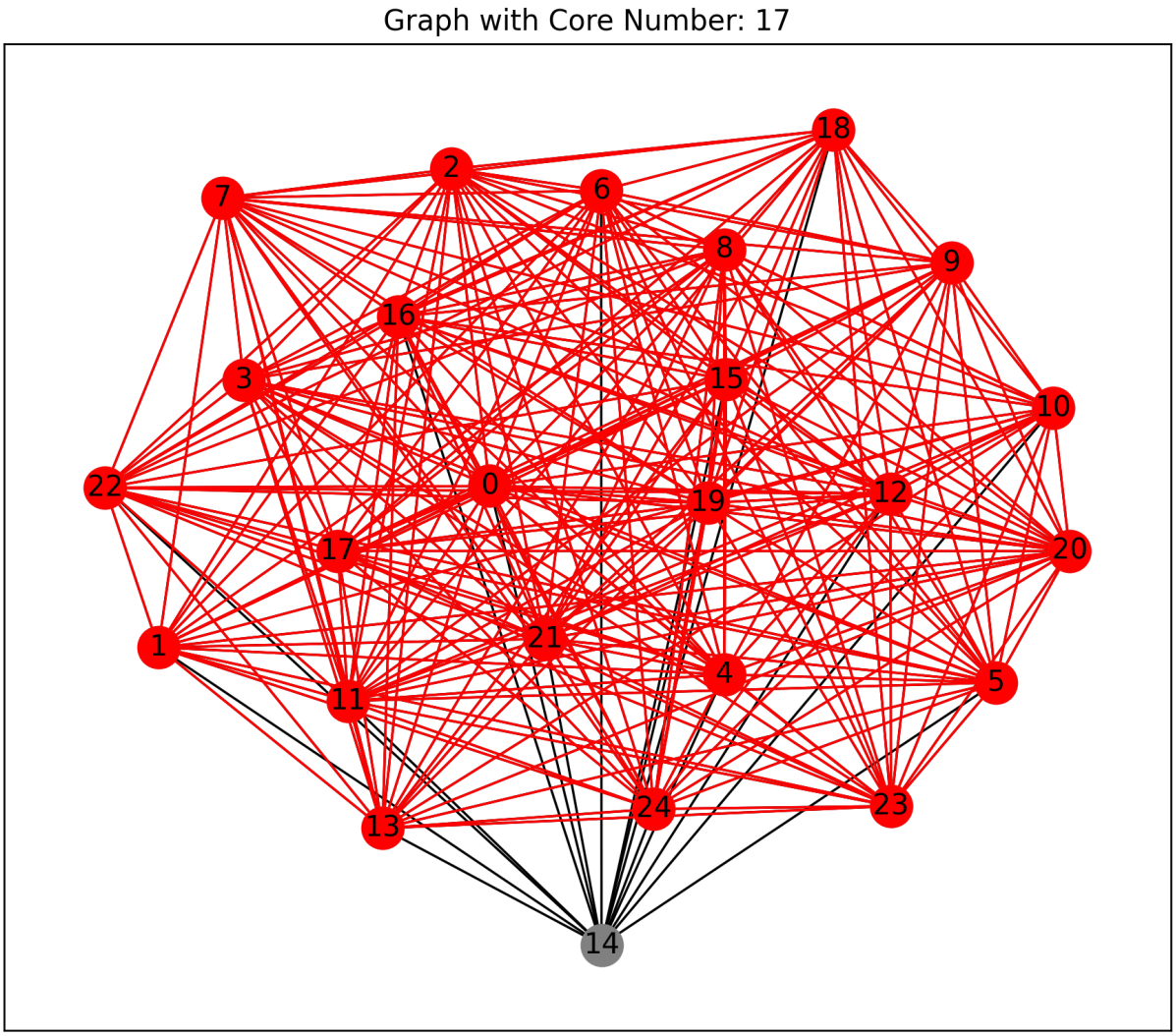
For p = 0.15



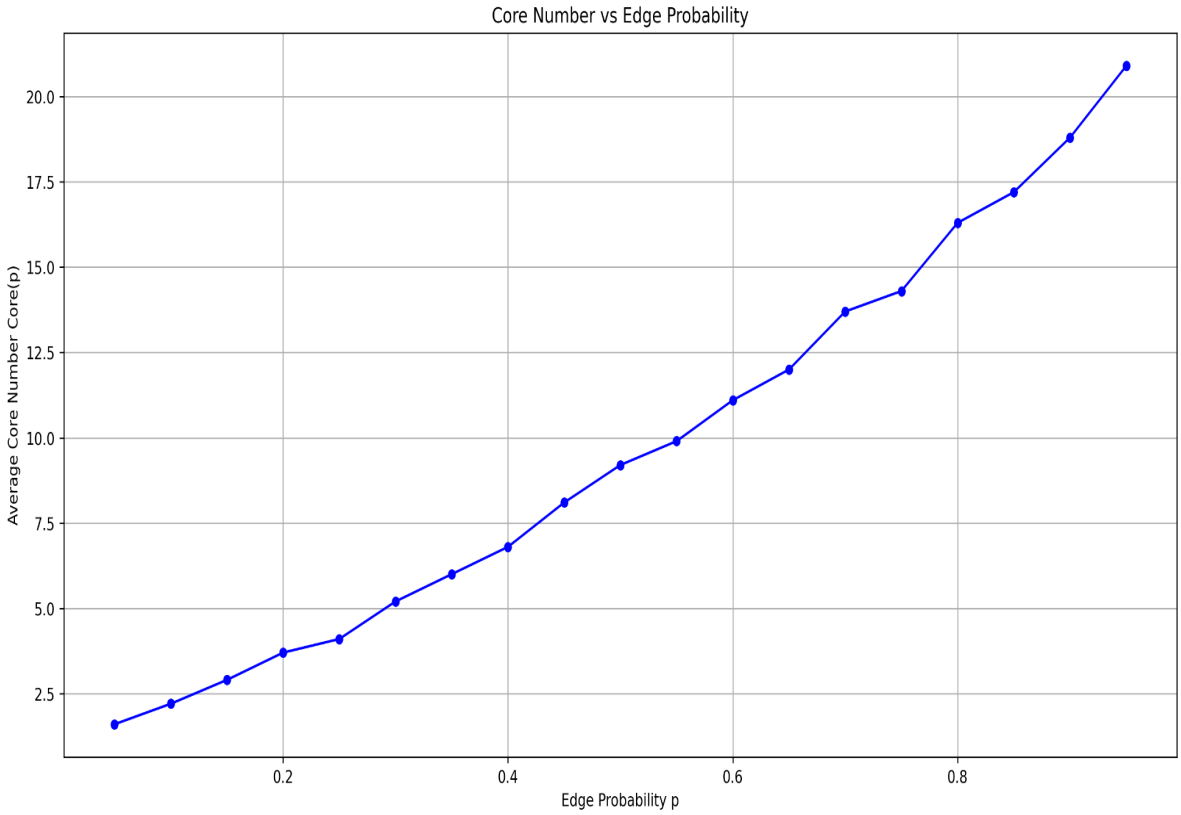
For p = 0.5


Graph with Core Number: 8
6
1
23
5
15
21
8
4
20
11
10
3
19
18
13
24
2
14
0
12
17
7
16
9
22

For p = 0.85


Graph with Core Number: 17
18
2
7
6
8
9
16
3
15
10
22
0
19
12
17
20
1
21
4
5
11
24
23
13
14

Task 3: edge probability (p) dependency on Core(p)



APPENDIX
PROGRAM

 # Task 1:

```
# Importing the networkx library for graph analysis
import networkx as nx
# Importing the matplotlib library for visualization
import matplotlib.pyplot as plt

""" Function for generating an Erdos-Renyi random graph with 25 nodes
and edge probability p."""
def random_graph_generator(p):
    # Return the random generated graph
    return nx.erdos_renyi_graph(25, p)

# Function for returning the length of k-core of the graph.
def kcore(G, k):
    # Extracting the k-core subgraph
    core_set = nx.k_core(G, k)
    # Return the length of the nodes in the subset
    return len(core_set)

# Function to calculate the core number of a graph.
def core_num(G):
    k = 1
    while kcore(G, k) != 0:
        k += 1
    return k - 1
```

```
""" Function to calculate the average of the core number over the
number of runs specified. """
def average_core_num(p, totalruns=10):
    total = 0
    for i in range(totalruns):
        G = random_graph_generator(p)
        total += core_num(G)
    # Return the average
    return total / totalruns

# Taking the edge probability as input
p = float(input("Enter p value:"))
# Printing the output for Task 1
print(f"Average Core Number for p = {p} over 10 runs:
{average_core_num(p)}")
```

 # Task 2:

```
""" Function to visualize the graph with nodes and edges belonging to
k-core colored in red. """
def visualize_graph(G):

    # Calculating the core number of the graph to print in the graph.
    k = core_num(G)
    core_subgraph = nx.k_core(G, k)

    # Positioning of the nodes in the graph
```

```
    pos = nx.spring_layout(G)

    # Making the color of all the nodes and edges of the graph to gray.
    nx.draw_networkx_nodes(G, pos, node_color="gray", node_size=300)
    nx.draw_networkx_edges(G, pos, alpha=1)

    """ Making the color of all the core nodes and edges of the graph to
red."""
    nx.draw_networkx_nodes(core_subgraph, pos, node_color="red",
node_size=300)
    nx.draw_networkx_edges(core_subgraph, pos, edge_color="red")
    nx.draw_networkx_labels(G, pos)

    # Plotting the graph with a title having the core number of it.
    plt.title(f'Graph with Core Number: {k}')
    plt.show()

# Displaying the graphs for three different p values
for p in [0.15, 0.5, 0.85]:
    """ Generating the random graph using the random graph generator
function."""
    G = random_graph_generator(p)
    visualize_graph(G)
```

 # Task 3:

```
""" Function to plot the connectivity between the edge probablity and
the average core number calculated from the task 1."""
```

```
def core_dependency_graph():

    # Taking the p values as a list from 0.05 to 0.95 in increments of 0.05.
    p_values = [i * 0.05 for i in range(1, 20)]
    # Calculating the average core numbers for each value of the p.
    core_values = [average_core_num(p) for p in p_values]

    """ Plotting the graph with edge probability on x axis and average
core number on y axis. """
    plt.plot(p_values, core_values, '-o', markersize = 5, color="blue")
    plt.xlabel('Edge Probability p')
    plt.ylabel('Average Core Number Core(p)')

    # Give the plot a title and display.
    plt.title('Core Number vs Edge Probability')
    plt.grid(True)
    plt.show()

core_dependency_graph()
```

README

1. Make sure that python is installed (version 3.x recommended).
2. Install the required libraries (networkx and matplotlib) using the commands:
   - pip install networkx
   - pip install matplotlib
3. Copy the whole program given in the previous section and save it. Run the program.
4. The program asks for input value of p (edge probability). Give it a value greater than 0 and less than 1.
5. For task 2 the program displays the graphs one after the other. So, the 2nd one will open when we close the 1st one and so on.

REFERENCES

- https://app.diagrams.net/
- https://networkx.org/documentation/stable/reference/index.html