Assignment2 (Score: 14.0 / 14.0)

1. Test cell (Score: 1.0 / 1.0)
2. Test cell (Score: 1.0 / 1.0)
3. Test cell (Score: 1.0 / 1.0)
4. Test cell (Score: 1.0 / 1.0)
5. Test cell (Score: 1.0 / 1.0)
6. Test cell (Score: 1.0 / 1.0)
7. Test cell (Score: 1.0 / 1.0)
8. Test cell (Score: 1.0 / 1.0)
9. Test cell (Score: 1.0 / 1.0)
10. Test cell (Score: 1.0 / 1.0)
11. Test cell (Score: 1.0 / 1.0)
12. Test cell (Score: 1.0 / 1.0)
13. Test cell (Score: 1.0 / 1.0)
14. Test cell (Score: 1.0 / 1.0)

Assignment 2 - Network Connectivity

In this assignment you will go through the process of importing and analyzing an internal email communication network between employees of a mid-sized manufacturing company. Each node represents an employee and each directed edge between two nodes represents an individual email. The left node represents the sender and the right node represents the recipient. We will also store the timestamp of each email.

In [1]:
```
import networkx as nx

#!head assets/email_network.txt
```

In [2]:
```
!head assets/email_network.txt
```

```
#Sender Recipient       time
1       2       1262454010
1       3       1262454010
1       4       1262454010
1       5       1262454010
1       6       1262454010
1       7       1262454010
1       8       1262454010
1       9       1262454010
1       10      1262454010
```

Question 1

Using networkx, load up the directed multigraph from `assets/email_network.txt` . Make sure the node names are strings.

This function should return a directed multigraph networkx graph.

In [3]:

```
import networkx as nx

def answer_one():
    # Load the directed multigraph from the file
    G = nx.read_edgelist('assets/email_network.txt',
                         nodetype=str,   # This ensures that nodes are tre
d as strings
                         create_using=nx.MultiDiGraph(),  # Create a direc
multigraph
                         data=[('time', int)])  # Add 'time' attribute for
ch edge
    return G

# Test loading the graph
try:
    ans_one = answer_one()
    print(f"Graph loaded successfully with {ans_one.number_of_nodes()} nod
and {ans_one.number_of_edges()} edges.")
except Exception as e:
    print(f"Error loading graph: {e}")
```

```
Graph loaded successfully with 167 nodes and 82927 edges.
```

In [4]:

```
cell-483a56c89f9db231
```

```
ans_one = answer_one()
```

Question 2

How many employees are represented in the network?

How many `sender` -> `recipient` pairs of employees are there in the network such that `sender` sent at least one emai `recipient` ? Note that even if a `sender` sent multiple messages to a `recipient` , they should only be counted once. should **not** exclude cases where an employee sent emails to themselves from this [email] count.

This function should return a tuple with two integers (#employees, # `sender` -> `recipient` pairs).

In [5]:

```
def answer_two():
    # Load the email network graph
    G = answer_one()    # 1. Number of employees (nodes in the graph)
    num_employees = G.number_of_nodes()

    # 2. Number of unique sender -> recipient pairs (unique directed edges
    unique_pairs = set(G.edges(keys=False))  # Convert edges to a set to e.
re uniqueness
    num_unique_pairs = len(unique_pairs)     # Get the number of unique se.
r -> recipient pairs

    # Return the number of employees and the number of unique sender-recip.
t pairs
    return (num_employees, num_unique_pairs)

# Call the function and print the result
ans_two = answer_two()
print(ans_two)
```

```
(167, 5784)
```

In [6]:

```
cell-5b6391549b076d2b
```

```
ans_two = answer_two()
```

Question 3

- Part 1. Assume that information in this company can only be exchanged through email.

  When an employee sends an email to another employee, a communication channel has been created, allowing the se
  to provide information to the reciever, but not viceversa.

  Based on the emails sent in the data, is it possible for information to go from every employee to every other employee

- Part 2. Now assume that a communication channel established by an email allows information to be exchanged both ways.

  Based on the emails sent in the data, is it possible for information to go from every employee to every other employee

This function should return a tuple of bools (part1, part2).

In [7]:

```
def answer_three():
    # Load the email network graph
    G = answer_one()

    # Part 1: Check if the directed graph is strongly connected
    part1 = nx.is_strongly_connected(G)

    # Part 2: Convert the graph to an undirected version and check if it i
onnected
    G_un = G.to_undirected()
    part2 = nx.is_connected(G_un)

    # Return the results for part1 and part2
    return (part1, part2)

# Call the function and print the result
ans_three = answer_three()
print(ans_three)
```

```
(False, True)
```

In [8]:

```
cell-82b3f0bc45e2895f
```

```
ans_three = answer_three()
```

Question 4

How many nodes are in the largest weakly connected component of the graph?

This function should return an int.

In [9]:
```
def answer_four():
    # Load the email network graph
    G = answer_one()

    # Find all weakly connected components
    weakly_connected_components = nx.weakly_connected_components(G)

    # Find the largest weakly connected component by size
    largest_component = max(weakly_connected_components, key=len)

    # Return the size of the largest weakly connected component
    return len(largest_component)

# Call the function and print the result
ans_four = answer_four()
print(ans_four)
```

```
167
```

In [10]:
```
        cell-2b1b7b06ecfa751d

ans_four = answer_four()
```

Question 5

How many nodes are in the largest strongly connected component?

This function should return an int

In [11]:
```
def answer_five():
    # Load the email network graph
    G = answer_one()

    # Find all strongly connected components
    strongly_connected_components = nx.strongly_connected_components(G)

    # Find the largest strongly connected component by size
    largest_scc = max(strongly_connected_components, key=len)

    # Return the size of the largest strongly connected component
    return len(largest_scc)

# Call the function and print the result
ans_five = answer_five()
print(ans_five)
```

```
126
```

In [12]:
```
cell-b0524f7dc1fbdec4
```

```
ans_five = answer_five()
```

Question 6

Using the NetworkX functions `strongly_connected_components` and `subgraph` , find the subgraph of nodes in the largest strongly connected component. Call this graph G_sc.

This function should return a networkx MultiDiGraph named G_sc.

In [13]:
```
def answer_six():
    # Load the email network graph
    G = answer_one()

    # Find all strongly connected components
    strongly_connected_components = nx.strongly_connected_components(G)

    # Find the largest strongly connected component
    largest_component = max(strongly_connected_components, key=len)

    # Create a subgraph from the largest strongly connected component
    G_sub = G.subgraph(largest_component).copy()  # Copy to ensure it's a
graph object

    # Convert the subgraph to a MultiDiGraph explicitly
    G_sc = nx.MultiDiGraph(G_sub)

    return G_sc

# Call the function and print the result
ans_six = answer_six()
print(ans_six)
```

```
MultiDiGraph with 126 nodes and 82130 edges
```

In [14]:
```
cell-cf148ef273b3b19c
```

```
ans_six = answer_six()
assert type(ans_six) == nx.MultiDiGraph , "Your return type should be a Mu
DiGraph object"
```

Question 7

What is the average distance between nodes in G_sc?

This function should return a float.

In [15]:
```
def answer_seven():
    # Get the largest strongly connected component subgraph
    G_sc = answer_six()

    # Calculate the average shortest path length
    avg_distance = nx.average_shortest_path_length(G_sc)

    return avg_distance

# Call the function and print the result
ans_seven = answer_seven()
print(ans_seven)
```

```
1.6461587301587302
```

In [16]:
```
        cell-5b374fdd48f37e02

ans_seven = answer_seven()
```

Question 8

What is the largest possible distance between two employees in G_sc?

This function should return an int.

In [17]:
```
def answer_eight():
    # Get the largest strongly connected component subgraph
    G_sc = answer_six()

    # Calculate the diameter of the graph
    diameter = nx.diameter(G_sc)

    return diameter

# Call the function and print the result
ans_eight = answer_eight()
print(ans_eight)
```

```
3
```

In [18]:
```
        cell-c5714787854ef644

ans_eight = answer_eight()
```

Question 9

What is the set of nodes in G_sc with eccentricity equal to the diameter?

This function should return a set of the node(s).

In [19]:
```
def answer_nine():
    # Get the largest strongly connected component subgraph
    G_sc = answer_six()

    # Calculate the diameter of the graph
    diameter = nx.diameter(G_sc)

    # Get the eccentricity of all nodes
    ecc = nx.eccentricity(G_sc)

    # Find nodes whose eccentricity equals the diameter
    nodes_with_diameter_ecc = {node for node, e in ecc.items() if e == dia
er}

    return nodes_with_diameter_ecc

# Call the function and print the result
ans_nine = answer_nine()
print(ans_nine)
```

```
{'97', '134', '129'}
```

In [20]:
```
        cell-77c9ca0b94df3d6f

ans_nine = answer_nine()
assert type(ans_nine) == set, "Student answer must return a set"
```

Question 10

What is the set of node(s) in G_sc with eccentricity equal to the radius?

This function should return a set of the node(s).

In [21]:

```
def answer_ten():
    # Get the largest strongly connected component subgraph
    G_sc = answer_six()

    # Calculate the radius of the graph (minimum eccentricity)
    radius = nx.radius(G_sc)

    # Get the eccentricity of all nodes
    ecc = nx.eccentricity(G_sc)

    # Find nodes whose eccentricity equals the radius
    nodes_with_radius_ecc = {node for node, e in ecc.items() if e == radiu

    return nodes_with_radius_ecc

# Call the function and print the result
ans_ten = answer_ten()
print(ans_ten)
```

```
{'38'}
```

In [22]:

```
cell-bfd2ee304bc25264

ans_ten = answer_ten()
assert type(ans_ten) == set, "Student answer must return a set"
```

Question 11

Which node in G_sc has the most shortest paths to other nodes whose distance equal the diameter of G_sc?

For the node with the most such shortest paths, how many of these paths are there?

This function should return a tuple (name of node, number of paths).

In [23]:
```
def answer_eleven():

    G_sc = answer_six()
    d = nx.diameter(G_sc)
    peripheries = nx.periphery(G_sc)

    dic = {}

    for node in peripheries:
        length_to_all = nx.shortest_path_length(G_sc, node).values()
        equal_count = list(length_to_all).count(d)
        dic[node] = equal_count

    max_key = max(dic, key=dic.get)
    max_value = max(dic.values())

    return (max_key, max_value)


answer_eleven()
```

Out[23]: ('97', 63)

In [24]:
```
        cell-f79b06650f61cf37
```

```
ans_eleven = answer_eleven()
assert type(ans_eleven) == tuple, "Student answer must be a tuple"
```

Question 12

Suppose you want to prevent communication flow from the node that you found in question 11 to node 10. What is the smallest number of nodes you would need to remove from the graph (you're not allowed to remove the node from the prev question or 10)?

This function should return an integer.

In [25]:

```
def answer_twelve():
    # Get the largest strongly connected component subgraph
    G_sc = answer_six()

    # Define the node found in Question 11 (you need to specify which node
    source_node = '97'  # Assuming node '97' is from Question 11
    target_node = '100'

    # Find the minimum number of nodes to remove to disconnect source_node
om target_node
    min_cut_set = nx.minimum_node_cut(G_sc, source_node, target_node)

    # Return the size of the node cut (i.e., the minimum number of nodes t
emove)
    return len(min_cut_set)

# Call the function and print the result
ans_twelve = answer_twelve()
print(ans_twelve)
```

```
1
```

In [26]:

```
cell-509cfa9f4136124d
```

```
ans_twelve = answer_twelve()
```

Question 13

Convert the graph G_sc into an undirected graph by removing the direction of the edges of G_sc. Call the new graph G_ur

This function should return a networkx Graph.

In [27]:

```
def answer_thirteen():
    G_sc = answer_six()
    undir_subgraph = G_sc.to_undirected()
    G_un = nx.Graph(undir_subgraph)
    return G_un
ans_thirteen = answer_thirteen()
print(ans_thirteen)
```

```
Graph with 126 nodes and 3107 edges
```

In [28]:
```
cell-d1c0627a327cd774
```

```
ans_thirteen = answer_thirteen()
assert type(ans_thirteen) == nx.Graph , "Your return type should be a Grap
bject"
```

Question 14

What is the transitivity and average clustering coefficient of graph G_un?

This function should return a tuple (transitivity, avg clustering).
Note: DO NOT round up your answer.

In [29]:
```
def answer_fourteen():

    G_un = answer_thirteen()
    transitivity = nx.transitivity(G_un)
    avg_clustering = nx.average_clustering(G_un)

    return (transitivity, avg_clustering)


answer_fourteen()
```

Out[29]: (0.570111160700385, 0.6975272437231418)

In [30]:
```
cell-41dda91202f58e7e
```

```
ans_fourteen = answer_fourteen()
assert type(ans_fourteen) == tuple, "Student answer must be a tuple"
```

In [ ]: