In [1]:
```
import pandas as pd
import numpy as np

data = pd.read_csv('Airlines_graph.csv')
```

In [2]:
```
import matplotlib.pyplot as plt
import networkx as nx
data.shape


data.dtypes
```

Out[2]:
```
year                 int64
month                int64
day                  int64
dep_time           float64
sched_dep_time       int64
dep_delay          float64
arr_time           float64
sched_arr_time       int64
arr_delay          float64
carrier             object
flight               int64
tailnum             object
origin              object
dest                object
air_time           float64
distance             int64
dtype: object
```

In [3]:
```
nx.__version__
```

Out[3]:
```
'2.7.1'
```

In [4]:
```
# converting sched_dep_time to 'std' - Scheduled time of departure
data['std'] = data.sched_dep_time.astype(str).str.replace('(\d{2}$)', '', regex=True) +
```

In [5]:
```
# converting sched_arr_time to 'sta' - Scheduled time of arrival
data['sta'] = data.sched_arr_time.astype(str).str.replace('(\d{2}$)', '', regex=True) +

# converting dep_time to 'atd' - Actual time of departure
data['atd'] = data.dep_time.fillna(0).astype(np.int64).astype(str).str.replace('(\d{2}$)
```

In [6]:
```
# converting arr_time to 'ata' - Actual time of arrival
data['ata'] = data.arr_time.fillna(0).astype(np.int64).astype(str).str.replace('(\d{2}$)
```

In [7]:
```
data['date'] = pd.to_datetime(data[['year', 'month', 'day']])
```

In [8]:
```
# finally we drop the columns we don't need
data = data.drop(columns = ['year', 'month', 'day'])
```

In [9]:
```
FG = nx.from_pandas_edgelist(data, source='origin', target='dest', edge_attr=True,)
```

In [10]:
```
FG.nodes()
```

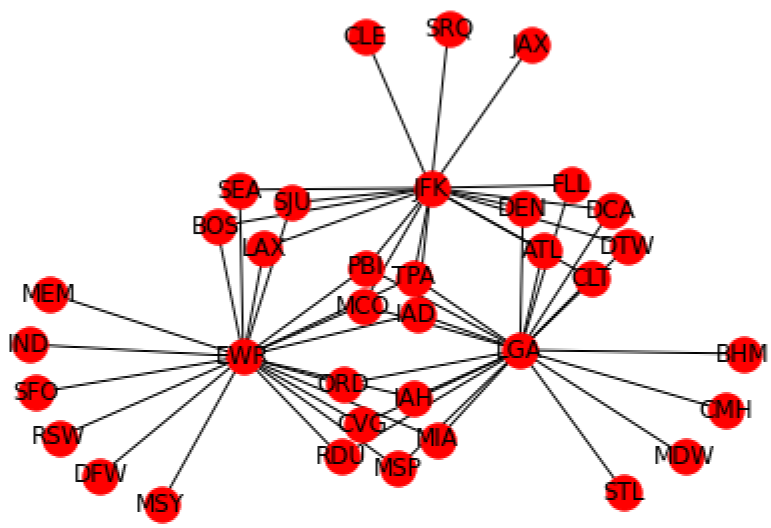
Out[10]:
```
NodeView(('EWR', 'MEM', 'LGA', 'FLL', 'SEA', 'JFK', 'DEN', 'ORD', 'MIA', 'PBI', 'MCO',
'CMH', 'MSP', 'IAD', 'CLT', 'TPA', 'DCA', 'SJU', 'ATL', 'BHM', 'SRQ', 'MSY', 'DTW', 'LA
X', 'JAX', 'RDU', 'MDW', 'DFW', 'IAH', 'SFO', 'STL', 'CVG', 'IND', 'RSW', 'BOS', 'CLE'))
```

In [11]:
```
FG.edges()
```

Out[11]: EdgeView([('EWR', 'MEM'), ('EWR', 'SEA'), ('EWR', 'MIA'), ('EWR', 'ORD'), ('EWR', 'MSP'), ('EWR', 'TPA'), ('EWR', 'MSY'), ('EWR', 'DFW'), ('EWR', 'IAH'), ('EWR', 'SFO'), ('EWR', 'CVG'), ('EWR', 'IND'), ('EWR', 'RDU'), ('EWR', 'IAD'), ('EWR', 'RSW'), ('EWR', 'BOS'), ('EWR', 'PBI'), ('EWR', 'LAX'), ('EWR', 'MCO'), ('EWR', 'SJU'), ('LGA', 'FLL'), ('LGA', 'ORD'), ('LGA', 'PBI'), ('LGA', 'CMH'), ('LGA', 'IAD'), ('LGA', 'CLT'), ('LGA', 'MIA'), ('LGA', 'DCA'), ('LGA', 'BHM'), ('LGA', 'RDU'), ('LGA', 'ATL'), ('LGA', 'TPA'), ('LGA', 'MDW'), ('LGA', 'DEN'), ('LGA', 'MSP'), ('LGA', 'DTW'), ('LGA', 'STL'), ('LGA', 'MCO'), ('LGA', 'CVG'), ('LGA', 'IAH'), ('FLL', 'JFK'), ('SEA', 'JFK'), ('JFK', 'DEN'), ('JFK', 'MCO'), ('JFK', 'TPA'), ('JFK', 'SJU'), ('JFK', 'ATL'), ('JFK', 'SRQ'), ('JFK', 'DCA'), ('JFK', 'DTW'), ('JFK', 'LAX'), ('JFK', 'JAX'), ('JFK', 'CLT'), ('JFK', 'PBI'), ('JFK', 'CLE'), ('JFK', 'IAD'), ('JFK', 'BOS')])

In [12]:
```
# your code is here (Quick view of the Graph.)

# plot the graph using matplotlib
nx.draw(FG, with_labels=True, node_color='red')
plt.show()
```



In [13]:
```
nx.algorithms.degree_centrality(FG) # Notice the 3 airports from which all of our 100 ro
# Calculate average edge density of the Graph

# your code is here
import statistics
statistics.mean(list(nx.algorithms.degree_centrality(FG).values()))
```

Out[13]: 0.09047619047619047

In [14]:
```
nx.average_shortest_path_length(FG) # Average shortest path length for ALL paths in the
```

Out[14]: 2.36984126984127

In [15]:
```
nx.average_degree_connectivity(FG) # For a node of degree k - What is the average of its
```

Out[15]: {20: 1.95, 1: 19.307692307692307, 2: 19.0625, 17: 2.0588235294117645, 3: 19.0}

In [16]:
```
# Let us find all the paths available
for path in nx.all_simple_paths(FG, source='JAX', target='DFW'):
 print(path)
```

```
['JAX', 'JFK', 'DEN', 'LGA', 'ORD', 'EWR', 'DFW']
['JAX', 'JFK', 'DEN', 'LGA', 'PBI', 'EWR', 'DFW']
['JAX', 'JFK', 'DEN', 'LGA', 'IAD', 'EWR', 'DFW']
['JAX', 'JFK', 'DEN', 'LGA', 'MIA', 'EWR', 'DFW']
['JAX', 'JFK', 'DEN', 'LGA', 'RDU', 'EWR', 'DFW']
```

```
['JAX', 'JFK', 'DEN', 'LGA', 'TPA', 'EWR', 'DFW']
['JAX', 'JFK', 'DEN', 'LGA', 'MSP', 'EWR', 'DFW']
['JAX', 'JFK', 'DEN', 'LGA', 'MCO', 'EWR', 'DFW']
['JAX', 'JFK', 'DEN', 'LGA', 'CVG', 'EWR', 'DFW']
['JAX', 'JFK', 'DEN', 'LGA', 'IAH', 'EWR', 'DFW']
['JAX', 'JFK', 'SEA', 'EWR', 'DFW']
['JAX', 'JFK', 'MCO', 'LGA', 'ORD', 'EWR', 'DFW']
['JAX', 'JFK', 'MCO', 'LGA', 'PBI', 'EWR', 'DFW']
['JAX', 'JFK', 'MCO', 'LGA', 'IAD', 'EWR', 'DFW']
['JAX', 'JFK', 'MCO', 'LGA', 'MIA', 'EWR', 'DFW']
['JAX', 'JFK', 'MCO', 'LGA', 'RDU', 'EWR', 'DFW']
['JAX', 'JFK', 'MCO', 'LGA', 'TPA', 'EWR', 'DFW']
['JAX', 'JFK', 'MCO', 'LGA', 'MSP', 'EWR', 'DFW']
['JAX', 'JFK', 'MCO', 'LGA', 'CVG', 'EWR', 'DFW']
['JAX', 'JFK', 'MCO', 'LGA', 'IAH', 'EWR', 'DFW']
['JAX', 'JFK', 'MCO', 'EWR', 'DFW']
['JAX', 'JFK', 'TPA', 'EWR', 'DFW']
['JAX', 'JFK', 'TPA', 'LGA', 'ORD', 'EWR', 'DFW']
['JAX', 'JFK', 'TPA', 'LGA', 'PBI', 'EWR', 'DFW']
['JAX', 'JFK', 'TPA', 'LGA', 'IAD', 'EWR', 'DFW']
['JAX', 'JFK', 'TPA', 'LGA', 'MIA', 'EWR', 'DFW']
['JAX', 'JFK', 'TPA', 'LGA', 'RDU', 'EWR', 'DFW']
['JAX', 'JFK', 'TPA', 'LGA', 'MSP', 'EWR', 'DFW']
['JAX', 'JFK', 'TPA', 'LGA', 'MCO', 'EWR', 'DFW']
['JAX', 'JFK', 'TPA', 'LGA', 'CVG', 'EWR', 'DFW']
['JAX', 'JFK', 'TPA', 'LGA', 'IAH', 'EWR', 'DFW']
['JAX', 'JFK', 'SJU', 'EWR', 'DFW']
['JAX', 'JFK', 'ATL', 'LGA', 'ORD', 'EWR', 'DFW']
['JAX', 'JFK', 'ATL', 'LGA', 'PBI', 'EWR', 'DFW']
['JAX', 'JFK', 'ATL', 'LGA', 'IAD', 'EWR', 'DFW']
['JAX', 'JFK', 'ATL', 'LGA', 'MIA', 'EWR', 'DFW']
['JAX', 'JFK', 'ATL', 'LGA', 'RDU', 'EWR', 'DFW']
['JAX', 'JFK', 'ATL', 'LGA', 'TPA', 'EWR', 'DFW']
['JAX', 'JFK', 'ATL', 'LGA', 'MSP', 'EWR', 'DFW']
['JAX', 'JFK', 'ATL', 'LGA', 'MCO', 'EWR', 'DFW']
['JAX', 'JFK', 'ATL', 'LGA', 'CVG', 'EWR', 'DFW']
['JAX', 'JFK', 'ATL', 'LGA', 'IAH', 'EWR', 'DFW']
['JAX', 'JFK', 'DCA', 'LGA', 'ORD', 'EWR', 'DFW']
['JAX', 'JFK', 'DCA', 'LGA', 'PBI', 'EWR', 'DFW']
['JAX', 'JFK', 'DCA', 'LGA', 'IAD', 'EWR', 'DFW']
['JAX', 'JFK', 'DCA', 'LGA', 'MIA', 'EWR', 'DFW']
['JAX', 'JFK', 'DCA', 'LGA', 'RDU', 'EWR', 'DFW']
['JAX', 'JFK', 'DCA', 'LGA', 'TPA', 'EWR', 'DFW']
['JAX', 'JFK', 'DCA', 'LGA', 'MSP', 'EWR', 'DFW']
['JAX', 'JFK', 'DCA', 'LGA', 'MCO', 'EWR', 'DFW']
['JAX', 'JFK', 'DCA', 'LGA', 'CVG', 'EWR', 'DFW']
['JAX', 'JFK', 'DCA', 'LGA', 'IAH', 'EWR', 'DFW']
['JAX', 'JFK', 'DTW', 'LGA', 'ORD', 'EWR', 'DFW']
['JAX', 'JFK', 'DTW', 'LGA', 'PBI', 'EWR', 'DFW']
['JAX', 'JFK', 'DTW', 'LGA', 'IAD', 'EWR', 'DFW']
['JAX', 'JFK', 'DTW', 'LGA', 'MIA', 'EWR', 'DFW']
['JAX', 'JFK', 'DTW', 'LGA', 'RDU', 'EWR', 'DFW']
['JAX', 'JFK', 'DTW', 'LGA', 'TPA', 'EWR', 'DFW']
['JAX', 'JFK', 'DTW', 'LGA', 'MSP', 'EWR', 'DFW']
['JAX', 'JFK', 'DTW', 'LGA', 'MCO', 'EWR', 'DFW']
['JAX', 'JFK', 'DTW', 'LGA', 'CVG', 'EWR', 'DFW']
['JAX', 'JFK', 'DTW', 'LGA', 'IAH', 'EWR', 'DFW']
['JAX', 'JFK', 'LAX', 'EWR', 'DFW']
['JAX', 'JFK', 'FLL', 'LGA', 'ORD', 'EWR', 'DFW']
['JAX', 'JFK', 'FLL', 'LGA', 'PBI', 'EWR', 'DFW']
['JAX', 'JFK', 'FLL', 'LGA', 'IAD', 'EWR', 'DFW']
['JAX', 'JFK', 'FLL', 'LGA', 'MIA', 'EWR', 'DFW']
['JAX', 'JFK', 'FLL', 'LGA', 'RDU', 'EWR', 'DFW']
['JAX', 'JFK', 'FLL', 'LGA', 'TPA', 'EWR', 'DFW']
['JAX', 'JFK', 'FLL', 'LGA', 'MSP', 'EWR', 'DFW']
['JAX', 'JFK', 'FLL', 'LGA', 'MCO', 'EWR', 'DFW']
```

```
['JAX', 'JFK', 'FLL', 'LGA', 'CVG', 'EWR', 'DFW']
['JAX', 'JFK', 'FLL', 'LGA', 'IAH', 'EWR', 'DFW']
['JAX', 'JFK', 'CLT', 'LGA', 'ORD', 'EWR', 'DFW']
['JAX', 'JFK', 'CLT', 'LGA', 'PBI', 'EWR', 'DFW']
['JAX', 'JFK', 'CLT', 'LGA', 'IAD', 'EWR', 'DFW']
['JAX', 'JFK', 'CLT', 'LGA', 'MIA', 'EWR', 'DFW']
['JAX', 'JFK', 'CLT', 'LGA', 'RDU', 'EWR', 'DFW']
['JAX', 'JFK', 'CLT', 'LGA', 'TPA', 'EWR', 'DFW']
['JAX', 'JFK', 'CLT', 'LGA', 'MSP', 'EWR', 'DFW']
['JAX', 'JFK', 'CLT', 'LGA', 'MCO', 'EWR', 'DFW']
['JAX', 'JFK', 'CLT', 'LGA', 'CVG', 'EWR', 'DFW']
['JAX', 'JFK', 'CLT', 'LGA', 'IAH', 'EWR', 'DFW']
['JAX', 'JFK', 'PBI', 'LGA', 'ORD', 'EWR', 'DFW']
['JAX', 'JFK', 'PBI', 'LGA', 'IAD', 'EWR', 'DFW']
['JAX', 'JFK', 'PBI', 'LGA', 'MIA', 'EWR', 'DFW']
['JAX', 'JFK', 'PBI', 'LGA', 'RDU', 'EWR', 'DFW']
['JAX', 'JFK', 'PBI', 'LGA', 'TPA', 'EWR', 'DFW']
['JAX', 'JFK', 'PBI', 'LGA', 'MSP', 'EWR', 'DFW']
['JAX', 'JFK', 'PBI', 'LGA', 'MCO', 'EWR', 'DFW']
['JAX', 'JFK', 'PBI', 'LGA', 'CVG', 'EWR', 'DFW']
['JAX', 'JFK', 'PBI', 'LGA', 'IAH', 'EWR', 'DFW']
['JAX', 'JFK', 'PBI', 'EWR', 'DFW']
['JAX', 'JFK', 'IAD', 'LGA', 'ORD', 'EWR', 'DFW']
['JAX', 'JFK', 'IAD', 'LGA', 'PBI', 'EWR', 'DFW']
['JAX', 'JFK', 'IAD', 'LGA', 'MIA', 'EWR', 'DFW']
['JAX', 'JFK', 'IAD', 'LGA', 'RDU', 'EWR', 'DFW']
['JAX', 'JFK', 'IAD', 'LGA', 'TPA', 'EWR', 'DFW']
['JAX', 'JFK', 'IAD', 'LGA', 'MSP', 'EWR', 'DFW']
['JAX', 'JFK', 'IAD', 'LGA', 'MCO', 'EWR', 'DFW']
['JAX', 'JFK', 'IAD', 'LGA', 'CVG', 'EWR', 'DFW']
['JAX', 'JFK', 'IAD', 'LGA', 'IAH', 'EWR', 'DFW']
['JAX', 'JFK', 'IAD', 'EWR', 'DFW']
['JAX', 'JFK', 'BOS', 'EWR', 'DFW']
```

In [17]:

```
# Let us find the dijkstra path from JAX to DFW.
# You can read more in-depth on how dijkstra works from this resource - https://courses.
dijpath = nx.dijkstra_path(FG, source='JAX', target='DFW')
dijpath
```

Out[17]:

```
['JAX', 'JFK', 'SEA', 'EWR', 'DFW']
```

In [18]:

```
# Let us try to find the dijkstra path weighted by airtime (approximate case)
shortpath = nx.dijkstra_path(FG, source='JAX', target='DFW', weight='air_time')
shortpath
```

Out[18]:

```
['JAX', 'JFK', 'BOS', 'EWR', 'DFW']
```

ASSIGNMENT-4 (100 Points)

Please use the Airlines_graph.csv for the following questions.

1. Please fill "your code here" sections on above cells (10 Points).
2. How many maximal cliques we can spot in this airline network? (20 Points)
3. List the most busiest/popular airport. (20 Points)
4. As a thought leader, identify 6 new routes to recommend. Hint: Think if the pairs are symmetric or not and make your assumption/observation accordingly i.e. whether ORD-LAX and LAX-ORD two separate routes? (50 Points)

2. How many maximal cliques we can spot in this airline network? (20 Points)

A clique is a subset of nodes in a graph where each node is connected to every other node in the subset. A maximal clique is a clique that cannot be extended by adding any other node in the network without violating the condition that every node in the subset must be connected to every other node.

In the context of an airline network, a maximal clique would correspond to a set of airports where there are direct flights between every pair of airports in the set. In other words, it would be a group of airports where passengers could travel directly between any pair of airports without needing to make a connecting flight.

In [19]:
```
# Calculate the maximal cliques in FG: cliques
cliques = nx.find_cliques(FG)

# Count and print the number of maximal cliques in G
print("We can spot {0} maximal cliques in this airline network.".format(len(list(cliques
```

```
We can spot 57 maximal cliques in this airline network.
```

In [20]:
```
print("The maximal cliques are : \n", list(nx.find_cliques(FG)))
```

```
The maximal cliques are :
 [['BOS', 'JFK'], ['BOS', 'EWR'], ['LGA', 'DCA'], ['LGA', 'IAH'], ['LGA', 'MCO'], ['LG
A', 'CVG'], ['LGA', 'ATL'], ['LGA', 'MDW'], ['LGA', 'FLL'], ['LGA', 'PBI'], ['LGA', 'MI
A'], ['LGA', 'IAD'], ['LGA', 'STL'], ['LGA', 'MSP'], ['LGA', 'TPA'], ['LGA', 'CLT'], ['L
GA', 'RDU'], ['LGA', 'CMH'], ['LGA', 'DEN'], ['LGA', 'BHM'], ['LGA', 'DTW'], ['LGA', 'OR
D'], ['CLE', 'JFK'], ['MEM', 'EWR'], ['IND', 'EWR'], ['MSY', 'EWR'], ['RSW', 'EWR'], ['S
FO', 'EWR'], ['JFK', 'DCA'], ['JFK', 'TPA'], ['JFK', 'MCO'], ['JFK', 'ATL'], ['JFK', 'CL
T'], ['JFK', 'LAX'], ['JFK', 'SEA'], ['JFK', 'SJU'], ['JFK', 'DEN'], ['JFK', 'DTW'], ['J
FK', 'JAX'], ['JFK', 'FLL'], ['JFK', 'SRQ'], ['JFK', 'PBI'], ['JFK', 'IAD'], ['LAX', 'EW
R'], ['SEA', 'EWR'], ['SJU', 'EWR'], ['EWR', 'MSP'], ['EWR', 'IAH'], ['EWR', 'MCO'], ['E
WR', 'TPA'], ['EWR', 'CVG'], ['EWR', 'ORD'], ['EWR', 'RDU'], ['EWR', 'DFW'], ['EWR', 'PB
I'], ['EWR', 'MIA'], ['EWR', 'IAD']]
```

3. List the most busiest/popular airport. (20 Points)

Degree centrality measures a node's local connectivity.

Betweenness centrality measures a node's role in connecting different parts of the network.

Eigenvector centrality measures a node's influence based on its connections to other important nodes in the network.

In [21]:
```
# Define find_nodes_with_highest_deg_cent()
def find_nodes_with_highest_deg_cent(G):

    # Compute the degree centrality of G: deg_cent
    deg_cent = nx.degree_centrality(G)

    # Compute the maximum degree centrality: max_dc
    max_dc = max(list(deg_cent.values()))

    nodes = set()

    # Iterate over the degree centrality dictionary
    for k, v in deg_cent.items():

        # Check if the current value has the maximum degree centrality
        if v == max_dc:

            # Add the current node to the set of nodes
            nodes.add(k)
    return nodes
```

```
# Find the node(s) that has the highest degree centrality in T: top_dc
top_dc = find_nodes_with_highest_deg_cent(FG)
print(top_dc)

# Write the assertion statement
for node in top_dc:
    assert nx.degree_centrality(FG)[node] == max(nx.degree_centrality(FG).values())
```

```
{'LGA', 'EWR'}
```

In [22]:
```
# Define find_node_with_highest_bet_cent()
def find_node_with_highest_bet_cent(G):

    # Compute betweenness centrality: bet_cent
    bet_cent = nx.betweenness_centrality(G)

    # Compute maximum betweenness centrality: max_bc
    max_bc = max(list(bet_cent.values()))

    nodes = set()

    # Iterate over the betweenness centrality dictionary
    for k, v in bet_cent.items():

        # Check if the current value has the maximum betweenness centrality
        if v == max_bc:

            # Add the current node to the set of nodes
            nodes.add(k)

    return nodes

# Use that function to find the node(s) that has the highest betweenness centrality in t
top_bc = find_node_with_highest_bet_cent(FG)
print(top_bc)

# Write an assertion statement that checks that the node(s) is/are correctly identified.
for node in top_bc:
    assert nx.betweenness_centrality(FG)[node] == max(nx.betweenness_centrality(FG).valu
```

```
{'EWR'}
```

In [23]:
```
# Define find_nodes_with_highest_ev_cent()
def find_nodes_with_highest_ev_cent(G):

    # Compute the eigen vector centrality of G: ev_cent
    ev_cent = nx.eigenvector_centrality(G,max_iter=1000)

    # Compute the maximum eigen vector centrality: max_evc
    max_evc = max(list(ev_cent.values()))

    nodes = set()

    # Iterate over the degree centrality dictionary
    for k, v in ev_cent.items():

        # Check if the current value has the maximum eigen vector centrality
        if v == max_evc:

            # Add the current node to the set of nodes
            nodes.add(k)
    return nodes

# Find the node(s) that has the highest eigen vector centrality in T: top_evc
top_evc = find_nodes_with_highest_ev_cent(FG)
```

```
print(top_evc)

# Write the assertion statement
for node in top_evc:
    assert nx.eigenvector_centrality(FG,max_iter=1000)[node] == max(nx.eigenvector_centr
```

```
{'LGA'}
```

In a flight network, we can use **degree centrality to compute the busiest airport**. The reason for this is that degree centrality measures the number of connections that a node has, which in this case would correspond to the number of flights arriving or departing from an airport. Therefore, an airport with a high degree centrality would have a large number of flights, indicating that it is busier than other airports in the network.

In [24]:

```
# Compute the degree centrality for each node in the graph
degree_centrality = nx.degree_centrality(FG)

# Identify the busiest airports based on their degree centrality
busiest_airports = [k for k, v in degree_centrality.items() if v >= 0.1]
print("The busiest airports are : ", busiest_airports)
```

```
The busiest airports are :  ['EWR', 'LGA', 'JFK']
```

4. As a thought leader, identify 6 new routes to recommend. Hint: Think if the pairs are symmetric or not and make your assumption/observation accordingly i.e. whether ORD-LAX and LAX-ORD two separate routes? (50 Points)

We know that there are no direct routes between the top 3 busiest airports EWR, LGA and JFK. We could find new routes in the airline network as follows :

In [25]:

```
from itertools import combinations

df = pd.DataFrame(columns=['Origin', 'Destination', 'Path', 'Distance'])

for n in FG.nodes():
    for n1, n2 in combinations(FG.neighbors(n), 2):
        # Recommending potential routes by identifying the open triangles in the network
        if n1!= n2 and not FG.has_edge(n1, n2):

            # Compute shortest path from node origin to destination in the open triangle
            shortest_path = nx.shortest_path(FG, source=n1, target=n2)

            # Compute the distance between the origin and destination airports for each 
            distance = nx.shortest_path_length(FG, source=n1, target=n2, weight='distanc

            # Print the origin, destination, paths, and weight
            origin = shortest_path[0]
            destination = shortest_path[-1]
            paths = "->".join(shortest_path)
            df.at[len(df)] = [origin, destination, paths, distance]

# Remove duplicate routes if any
df.drop_duplicates(subset=['Path'], inplace = True)

# Rank the potential routes based on their distances
df.sort_values(by='Distance', ascending=True, inplace = True)

# Output the top 6 new recommended routes
result = df.head(6)
result
```

Out[25]:

	Origin	Destination	Path	Distance

523	JFK	EWR	JFK->SEA->EWR	387
381	EWR	JFK	EWR->SEA->JFK	387
481	DCA	BOS	DCA->JFK->BOS	400
170	IAD	BOS	IAD->EWR->BOS	412
518	JFK	LGA	JFK->FLL->LGA	427
380	LGA	JFK	LGA->DEN->JFK	427

Even though there is no direct flight from the origin to the destination through the third airport in each open triangle, we can still find the most efficient indirect route that passes through the third airport by using the **shortest path algorithm** to find the best route.

By computing the shortest path between the origin and the destination, we can find the most optimal third airport that requires the **shortest amount of time and distance to reach**.

Using the shortest paths, we can construct a new route that passes through the third airport and connects the origin and destination. This new route is not only more efficient than flying through the busiest airports, but also enables passengers to **travel directly from the origin to the destination, without the need for layovers or connections.**

In [26]:
```
print("The 6 new routes recommended are : \n")
for ind in result.index:
    print(result['Origin'][ind] +" -> "+ result["Destination"][ind])
```

```
The 6 new routes recommended are :

JFK -> EWR
EWR -> JFK
DCA -> BOS
IAD -> BOS
JFK -> LGA
LGA -> JFK
```

In [ ]: