In [15]:
```
#Pandas for excel
import pandas as pd

#Openpyxl for data manipulation
from openpyxl import load_workbook

#Numpy for Lists and Arrays
import numpy as np

#For plotting plots
import matplotlib.pyplot as plt
```

In [16]:
```
#Read and store excel sheet 0 (first) into list xl
xl = pd.read_excel('dat.xlsx', sheet_name=0)
```

In [17]:
```
oc = xl['Origin City']
dc = xl['Destination City']

oc, dc

#We are storing 'Origin' and 'Destination' from excel into 'oc' and 'dc' in pythom
```

Out[17]:
```
(0       3
1       3
2      10
3      11
4      15
       ..
170     1
171     1
172     3
173    13
174    12
Name: Origin City, Length: 175, dtype: int64,
0       5
1      13
2       6
3      15
4       9
       ..
170     7
171     9
172     4
173     1
174    12
Name: Destination City, Length: 175, dtype: int64)
```

In [18]:
```
#Func Declaration with 'Origin', 'Destination' and 'Aircraft/FLEET Number' as inputs
def aircraft (oc, dc, fleet):

    #'a1' will have our data
    a1=[]

    #'A1' will have all the 'OD' pairs of 'each day' of current aircraft-'Fleet'
    for i in range(fleet, len(oc), 25):
        a1.append((oc[i], dc[i]))
```

```
    #Just Calculations for counting number of cities for each fleet
    a1.sort()
    check=[] #Empty list

       #For loop to arrange 'od' pairs in a single column
    for i in range(0, len(a1)):
        check.append(a1[i][0])
        check.append(a1[i][1])

    a=[]

    #For loop to delete duplicates and have only no. of cities
    for i in check:
        if i not in a:
            a.append(i)

        #create 'dictionary' to store each fleets OD pairs in 'a1', total cities in 'a'
    airfleet = dict()
    airfleet['a1'] = a1
    airfleet['a'] = a
    airfleet['maxi'] = max(a)
    return airfleet
```

In [19]:

```
#empty list called airfleet
airfleet = []

#segregate data fleet by fleet in a loop for all 25 fleets
#which stores all OD pairs for all 7 days for each AIRCRAFT/FLEET
for i in range(0, 25):

    airfleet.append(aircraft(oc, dc, i))
```

In [20]:

```
#airfleet
#Use 'a1' for OD pair
#Use 'a' for all cities in route (no dups)
airfleet[0].get('a1')
```

Out[20]: [(1, 6), (3, 5), (3, 13), (5, 1), (6, 3), (13, 14), (14, 3)]

In [21]:

```
#create class graph for Adjacency list with Initialization
#takes inputs 'Num_Nodes' and 'edges'(ODpairs/'a1')
#This class creates Adjacency List
#From Lab
class Graph:
    def __init__(self, num_nodes, edges):
        self.num_nodes = num_nodes
        self.data = [[] for _ in range(num_nodes)]
        for n1, n2 in edges:
            self.data[n1].append(n2)
            self.data[n2].append(n1)


    def __repr__(self):
        return "\n".join(["{}: {}".format(n, neighbors) for n, neighbors in enumerate(s
```

```
    def __str__(self):
        return self.__repr__()
```

In [22]:

```
#Create nodes/fleet = 25
num_nodes = 25
#Create empty list graphs to store adjacency list
graphs = []

#For loop to find adjacency for each aircraft/fleet by callng class 'Graph'
#Airfleet[i].get('a1') gets the ODpairs for each fleet i
#Results appended in list 'graphs'
for i in range(0, len(airfleet)):
    graphs.append(Graph(num_nodes, airfleet[i].get('a1')))
```

In [23]:

```
#graphs - to display all
#graphs[i] - to display for each aircraft/fleet 'i' (0,24)
graphs[24]
```

Out[23]:

```
0: []
1: []
2: [2, 2, 12, 4]
3: []
4: [2, 15]
5: []
6: []
7: []
8: []
9: []
10: []
11: []
12: [2, 12, 12, 13]
13: [12, 15]
14: []
15: [13, 4]
16: []
17: []
18: []
19: []
20: []
21: []
22: []
23: []
24: []
```

In [24]:

```
#Depth First Search
#Creating function 'DFS' with inputs 'graph'(adjacency) and 'root' (each city for each

def dfs(graph, root):
    #Create stack to stack data
    stack = []

    #Discovered to store visited nodes
    discovered = [False] * len(graph.data)

    #Create result for storing results
    result = []

    #Start appending stacks
    stack.append(root)
```

```
    #Keep looping until all stacks or OD pairs are discovered
    while len(stack) > 0:
        #Get current data ready
        current = stack.pop()
        #If Un-Discovered - Then discover it
        if not discovered[current]:
            discovered[current] = True
            #Then append that
            result.append(current)

            #Visit each nodes only one time based on discovered list
            for node in graph.data[current]:
                if not discovered[node]:
                    stack.append(node)

    return result
```

In [25]:
```
#create 'dfsres' to store DFS results of all 25 aircrafts for all 7 days
dfsres=[]

#For airfleets 0 to 25
for i in range(0, 25):

    #create root to store all cities for aircraft 'i' (no dups)
    root = []
    root = airfleet[i].get('a')

    #Col to store column wise
    col = []

    #Loop to do DFS for current aircraft 'i' from each of its origin city 'j'
    for j in range(0, len(root)):
        #Append results in 'COl' calling function 'dfs' with inputs
        col.append(dfs(graphs[i], root[j]))

    #Finally append 'col' with dfsres
    dfsres.append(col)
```

In [34]:
```
dfsres[0]
```

Out[34]:
```
[[1, 5, 3, 14, 13, 6],
 [6, 3, 14, 13, 5, 1],
 [3, 14, 13, 6, 1, 5],
 [5, 1, 6, 3, 14, 13],
 [13, 14, 3, 6, 1, 5],
 [14, 3, 6, 1, 5, 13]]
```

In [27]:
```
#Create object 'exl' using pandas 'pd' creating a 'DataFrame' of our dfsres
exl = pd.DataFrame(dfsres)
```

In [28]:
```
#Create 'write' to call function 'ExcelWriter' and specify Excel File 'data.xlsx'
write = pd.ExcelWriter('data.xlsx')
#Func 'to_excel' calling 'write' and sheetname 'Results'
exl.to_excel(write, 'Results')
write.save()
```

In [29]:

```
#Plotting the graph -Pizzazz
#Create 'ploty' to store single dfs for each aircraft
ploty = []

#For each aircraft
for i in range (0, len(dfsres)):
    #Append only one DFS fot each flight i
    ploty.append(dfsres[i][0])

#Create 'Figure' to overwrite plots to produce our networks
plt.figure()

#For each aircraft
for i in range (0, len(ploty)):
    #Cities store all city for each aircraft
    cities = range (0, len(ploty[i]))
    #Plot Cities-by-dfs for each aircraft i
    plt.plot (cities, ploty[i], color='black', marker='o', markersize=2, linewidth=1)

#Display plot
plt.show()
```