Network and Algorithms Final Project

Groupmates

Huseyn Naghiyev
Majid Rahmanov
Musakhan Eminov

1. Setup

We will be working in the local environment for this project. We used a Jupyter notebook for coding and editing. We used pandas library for data manipulation and analysis, numpy for working with arrays, `matplotlib for building graphs`, and data obtained from airports.csv and routes.csv files.

2. Task Description

1- select a region (Asia, Europe, ..) and extract the corresponding data from each file;

2- use the longitude/latitude values from the airports file to update the data extracted from routes file by adding the distance attribute for each direct route;

3- use all needed algorithms to define statistics on the airports, like the shortest path between two airports, the importance of an airport (number of routes from and to the airport), and other statistics you judge important in the context.

3. Working with Data

```
In [3]: adf = pd.read_csv("airports.csv")
```

```
In [5]: rdf = pd.read_csv("routes.csv")
```

We used above functions to read .csv files into Dataframe

	Airport ID	Name	City	Country	IATA	ICAO	Latitude	Longitude	Altitude	Timezone	DST	Tz database time zone	Type	Source
0	1	Goroka Airport	Goroka	Papua New Guinea	GKA	AYGA	-6.081690	145.391998	5282	10	U	Pacific/Port_Moresby	airport	OurAirports
1	2	Madang Airport	Madang	Papua New Guinea	MAG	AYMD	-5.207080	145.789001	20	10	U	Pacific/Port_Moresby	airport	OurAirports
2	3	Mount Hagen Kagamuga Airport	Mount Hagen	Papua New Guinea	HGU	AYMH	-5.826790	144.296005	5388	10	U	Pacific/Port_Moresby	airport	OurAirports
3	4	Nadzab Airport	Nadzab	Papua New Guinea	LAE	AYNZ	-6.569803	146.725977	239	10	U	Pacific/Port_Moresby	airport	OurAirports
4	5	Port Moresby Jacksons International Airport	Port Moresby	Papua New Guinea	POM	AYPY	-9.443380	147.220001	146	10	U	Pacific/Port_Moresby	airport	OurAirports

	Airline	Airline ID	Source airport	Source airport ID	Destination airport	Destination airport ID	Codeshare	Stops	Equipment
0	2B	410	AER	2965	KZN	2990	NaN	0	CR2
1	2B	410	ASF	2966	KZN	2990	NaN	0	CR2
2	2B	410	ASF	2966	MRV	2962	NaN	0	CR2
3	2B	410	CEK	2968	KZN	2990	NaN	0	CR2
4	2B	410	CEK	2968	OVB	4078	NaN	0	CR2

We used .str.split() with the parameter " / " to obtain the region value.

```
In [7]: adf['reg'] = adf['Tz database time zone'].str.split('/').str[0]
```

Our selected region became America.

```
In [8]: adf['reg'].unique()

Out[8]: array(['Pacific', 'America', 'Atlantic', 'Africa', 'Europe', 'Arctic',
               'Indian', 'Asia', '\\N', 'Antarctica', 'Australia'], dtype=object)

In [9]: aadf = adf[adf['reg'] == 'America']
```

The angular distance between two places on the surface of a sphere was calculated using the Haversine formula.

The Haversine (or great circle) distance is the angular distance between two points on the surface of a sphere. The first coordinate of each point is assumed to be the latitude, the second is the longitude, given in radians. The dimension of the data must be 2.

```
In [17]: def haversine(lon1, lat1, lon2, lat2):
             lon1, lat1, lon2, lat2 = map(np.radians, [lon1, lat1, lon2, lat2])
             dlon = lon2 - lon1
             dlat = lat2 - lat1
             a = np.sin(dlat/2.0)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/2.0)**2
             result = 6367 * 2 * np.arcsin(np.sqrt(a))
             return result
```

We used the compute_dist(r) function to calculate distance. We have to give 2 longitude/latitude parameters and in case of invalid arguments functions returns None value, indicating there is something wrong with coordinates.

```
In [18]: def compute_dist(r):
             try:
                 src, dst = r['Source airport'], r['Destination airport']
                 src_data = aadf[aadf['IATA'] == src][['Latitude', 'Longitude']].values
                 lon1, lat1 = src_data[0].tolist()
                 dst_data = aadf[aadf['IATA'] == dst][['Latitude', 'Longitude']].values
                 lon2, lat2 = dst_data[0].tolist()
                 return haversine(lon1, lat1, lon2, lat2)
             except:
                 return None

         ardf['distance'] = ardf.apply(compute_dist, axis=1)
```

Finally, our data for building graph has the following look:

Out[21]:

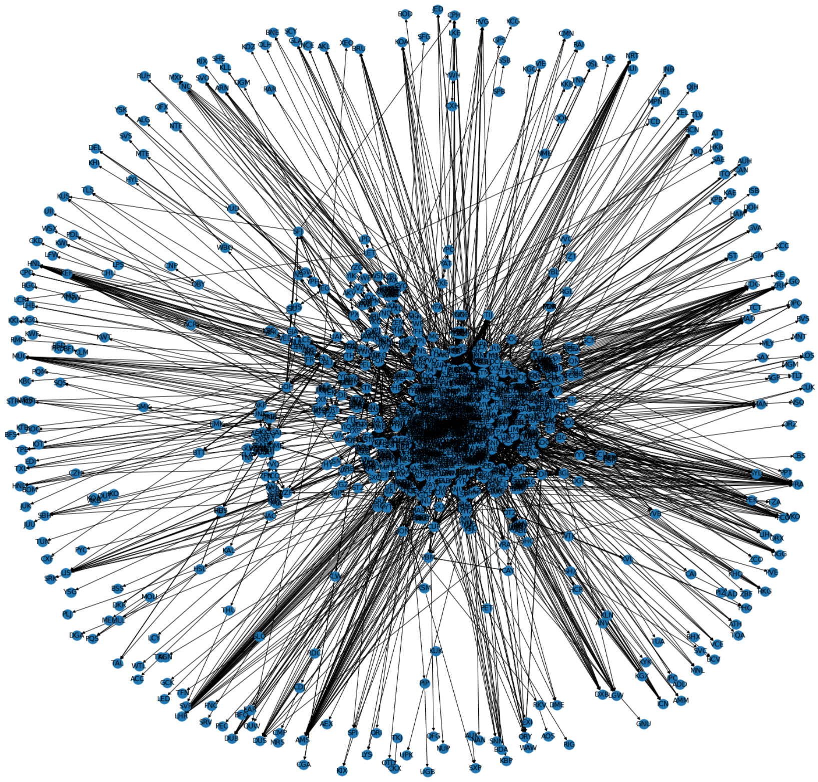
	source	destination	distance
63	AYP	LIM	324.848465
64	CUZ	LIM	576.829573
65	CUZ	PEM	303.082220
66	HUU	LIM	115.024253
67	IQT	PCL	199.177123
...	...	...	...
67565	SHR	DEN	296.170975
67566	SOW	FMN	216.911463
67567	SOW	PHX	225.358353
67569	VIS	LAX	168.109003
67570	WRL	CYS	359.668966

4. Graph

We build graph using previously obtained data

In [22]:
```
G = nx.from_pandas_edgelist(
    ardf, source='source', target='destination',
    edge_attr=True, create_using=nx.DiGraph
)
```

In [26]:
```
plt.figure(figsize=(23, 23))
nx.draw(G, with_labels=True)
```



5. Centrality

We used closeness centrality to detect nodes that are able to spread information very efficiently through a graph. The closeness centrality of a node measures its average farness (inverse distance) to all other nodes.

```
In [31]: ccent_data = nx.closeness_centrality(G)
```

```
In [34]: sorted_ccent = dict(sorted(ccent_data.items(),key=lambda x:x[0]))
```

```
In [53]: ccent_data = nx.closeness_centrality(G)
         sorted_ccent = dict(sorted(ccent_data.items(),key=lambda x:x[1]))
```

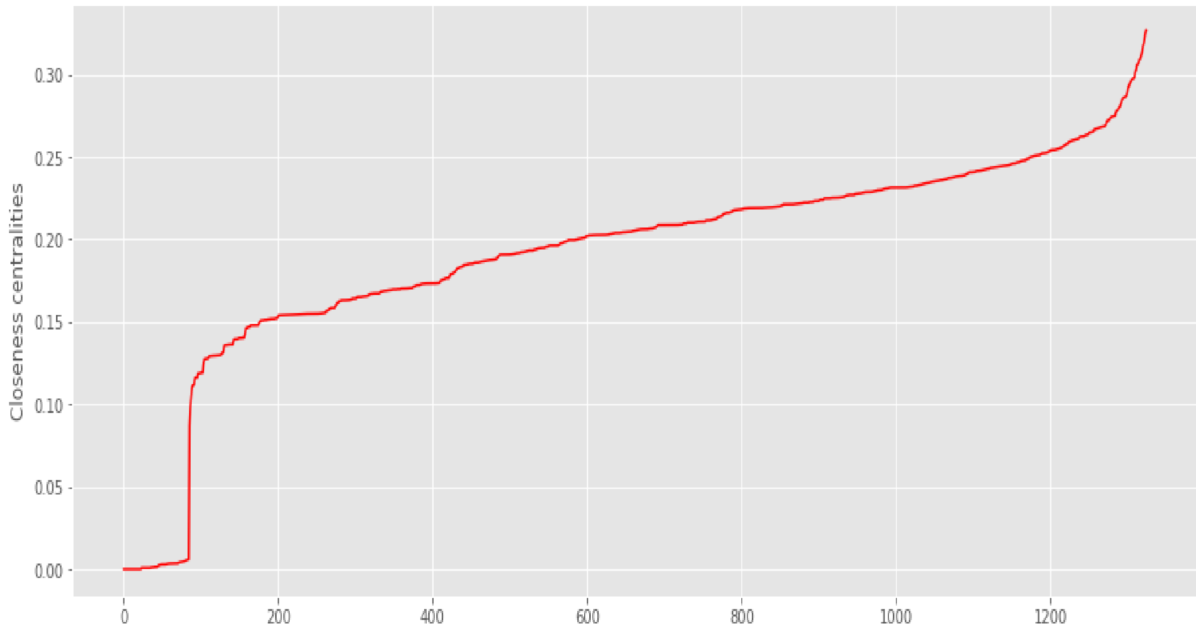
To show the graph stats info, we use the closeness Centrality, and then find top 10 nodes with greatest centrality values.

```
For LAS closeness_centrality is 0.30875191972282057
For MCO closeness_centrality is 0.3090957414597057
For EWR closeness_centrality is 0.3100163542414174
For ORD closeness_centrality is 0.3121079188502425
For LAX closeness_centrality is 0.3137543057620374
For JFK closeness_centrality is 0.31722054380664655
For YYZ closeness_centrality is 0.31855544280506776
For ATL closeness_centrality is 0.32125923128566636
For MIA closeness_centrality is 0.325147960754567 4
For DFW closeness_centrality is 0.326678669082953
```

Plotting Graph Haversine And the last, we shows the increase of closeness

```
In [65]: xs = sorted_ccent.values()
         ys = range(len(xs))

         plt.style.use('ggplot')
         plt.figure(figsize=(15, 6))
         plt.ylabel('Closeness centralities')
         plt.plot(ys, xs, color="red")
         plt.show()
```



Closeness centralities
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
200
400
600
800
1000
1200