FRENCH-AZERBAIJANI UNIVERSITY
UE709 Network Algorithms L3/S5
PW2 - Introduction to Graph Theory
November 2019

Working with Python and NetworkX (cont)

1 Requirements

This practice (continuation of the last practice) uses the file produced for the last Practical Work. Please, retrieve all information related to it before you start.

2 More information on NetworkX

The **shortest path** between two nodes on a graph is called a **geodesic**.

i. We are going to use NetworkX method `shortest_path(G, source=None, target=None, weight=None)` in this practice to find geodesics on a digraph. Note that we are still not discussing the method(s) it uses:

```
import pandas as pd
import networkx as nx

data = pd.read_csv('mydatafile.csv')
G = nx.from_pandas_edgelist(data, source='X', target='Y',
    edge_attr=True, create\_using=nx.DiGraph) # NEW
print(nx.shortest_path(G, source='NodeS', target='NodeD',
    weight='myAttrib'))
```

where `X` and `Y` are the titles (names) of columns that will be considered as the endpoints of each edge (row). Other columns are data for each edge (`edge_attr=True`); `NodeS` and `NodeD` are actual nodes of the graph, as `myAttrib` is the column name for the attribute used as weight.

Note that we are **using the `create_using=nx.DiGraph` option** to create our graph. This time we are considering **directional weight variation between nodes**.

iii. Property `degree` and variations:

```
print(G.degree('NodeS'))
print(G.in\_degree('NodeS'))
print(G.out\_degree('NodeS'))
```

If our graph is a `nx.Graph`, only `degree` is allowed (undirected graph). For `nx.DiGraph` (directed graph), we have all three variations, with `degree` as the sum of `in_degree` and `out_degree`.

iii. To find more information about NetworkX:
https://networkx.github.io/documentation/latest/index.html

3 Activities

3.1 Cities

A very basic dataset is available in `cities_in_az.csv`. Use columns `Origin` and `Destiny` as the endpoints of your graph edges. Column `Hours` is our weight of interest:

1. Plot the graph to visualise it
2. Apply `shortest_path` method to retrieve the path from `'Baku'` to `'Imishli'` without weights.
3. Apply `shortest_path` method to retrieve the path from `'Baku'` to `'Imishli'` with attribute *'Hours'* as the weight.

Is there any difference between those paths? Could you explain the reason?

1. Include in your code:

```
nx.add_path(G, ['Baku', 'Imishli'])
G.edges['Baku', 'Imishli']['Hours'] = 1.29
```

2. Then find shortest paths:
   - from Baku to Imishli
   - from Imishli to Baku

How different they are? Explain the reason.

3.2 Airports

The `airports.csv` dataset have a sample of flights from the USA. The below variables have been provided:

- Origin and destination
- Scheduled time of arrival and departure
- Actual time of arrival and departure

- Date of the journey
- Distance between the source and destination
- Total airtime of the flight

Use columns `Origin` and `Dest`, destinations of flights, as the endpoints of your graph edges. Then:

1. Plot the graph to visualise it
2. Find the shortest path with respect to the distance (column `Distance` of the dataset) from `'CPR'` to `'BOI'` and vice versa
3. Find the shortest path with respect to the time (column `AirTime` of the dataset) from `'CPR'` to `'BOI'` and vice versa

Compare your results. Regarding those metrics:

- **Degree Connectivity** – the number of edges connected to a node. In the case of a directed graph, we can have 2-degree centrality measures: **Inflow Centrality** and **Outflow Centrality**
- **Closeness Centrality** – Of a node is the average length of the shortest path from the node to all other nodes
- **Betweenness Centrality** – Number of times a node is present in the shortest path between 2 other nodes
- **Network Density** – based on how many edges a graph has; we can use the formula for directed graphs:

$$\frac{1}{n \cdot (n-1)} \cdot \sum_{n=1}^{N} \mathbf{deg}(\text{node}_n)$$

- **Network Diameter** – the longest of all its geodesics
- **Network Average Path Length** – the average of all geodesics

Write code to implement your version of those functions. Apply them to the network and present your results and discussions.

Hint: write auxiliary functions whenever need; for example, a function to return a list of shortest paths from a given node to all other nodes.