FRENCH-AZERBAIJANI UNIVERSITY
UE709 Network Algorithms L3/S5
PW3 - Network and Algorithms
November 2019

Working with Python and NetworkX (cont.2)

1 Requirements

This practice (continuation of the last practice) uses the file produced for the last Practical Works. Section 2 gather some information needed to this practice.

2 More information on NetworkX

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.

iii. About nodes and edges:

- `G.nodes()` returns a list with every node of the graph G
- `G.nodes[node_identifier]` returns a dictionary with all attributes for the `node_identifier` of G
- `G.nodes[node_identifier][attribute]` returns a dictionary with all attributes for that node

- `G.edges()` returns a list of tuples constituted by the endpoints of all unique edges (undirected) of G
- `G.edges[node_id1, node_id2]` returns a dictionary with all attributes for the edge `(node_id1, node_id2)` of G
- `G.edges[node_id1, node_id2][attribute]` returns the value of the attribute for that edge `(node_id1, node_id2)`
- `list(G.neighbors(node_identifier))` returns a list with all nodes that share an edge with the `node_identifier` (don't need to convert to list if you need to iterate through it)

iv. Property `degree` and variations:

- `G.degree('NodeS')`
- `G.in_degree('NodeS')`
- `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`.

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

3 Activities

3.1 Airports

The `airports.csv` dataset have a sample of flights from the USA.

Use columns `Origin` and `Dest`, destinations of flights, as the endpoints of your graph edges. There are two routing algorithms to be explored in order to find shortest paths:

1. Implement the Bellman-Ford_algorithm

   https://en.wikipedia.org/wiki/Bellman-Ford_algorithm

2. Implement the Dijkstra's algorithm

   https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

3. Find the shortest path with respect to the distance (column `Distance` of the dataset) from `'CRP'` to `'BOI'` and vice versa with both Bellman-Ford and Dijkstra's algorithms; compare their performance

4. Find the shortest path with respect to the time (column `AirTime` of the dataset) from `'CRP'` to `'BOI'` and vice versa with both Bellman-Ford and Dijkstra's algorithms; compare their performance