Experimenting with networkx

The **Going further** section is not thought to be done in class. Check it only if you have finished earlier or if you want to know more.

1. Simple network analysis with python and networkx.

   `networkx` package has a good documentation. To get started, do the beginning of the tutorial: `https://networkx.org/documentation/stable/tutorial.html`. After that, the best way to find what you're searching for is to ask google. For instance, if you wonder how to compute the betweenness centrality with networkx, just search *betweenness networkx* in google and your first result will certainly contain the answer.

   I highly recommend to use Jupyter Notebook to make those experiments. Notebooks allow to run pieces of codes without reloading the whole data everytime. If you don't have it installed yet, you can use online notebooks, such as `https://colab.research.google.com`.

   The airport dataset is a network of connections between airports, i.e., an edge exist between two airports if a company offer a direct flight between the two. It is available on the webpage of the class.

   (a) Using networkx, load the airport dataset using `read_graphml`.

   (b) Compute the number of nodes and edges of your graph. A simple way to go is to use `len` on `g.nodes` and `g.edges`

   (c) Compute the density and the clustering coefficent (`nx.density`, `nx.transitivity`)

   (d) Compute the average shortest path length and the diameter of the graph. You'll encounter a connected component issue. Can you understand why? As a solution, you need to apply those methods on the largest connected component, that you can extract with
   `cc = g.subgraph(sorted(nx.connected_components(g), key=len, reverse=True)[0])`

   (e) Obtain the list of the 20 nodes of highest and lowest degrees. You can use `degree()` and `sorted(X, key=lambda x: x[1]` for instance.

   (f) Compute the list of the 20 nodes of highest and lowest Pagerank. You might need to use `items` to transform a dictionary in a list of pair. Observe the differences.

   (g) Do the same for the betweenness. Where in hell is **Anchorage**? And **Port_Moresby**? Investigate a little to understand what is going on. How many neighbors do these nodes have, and who are they?

   (h) Check with other typical node centralities.

   (i) Check edge betweenness.

   (j) Would you say that the network is a *small world* network ?

   (k) Plot the network using `draw_networkx`. Check the documentation to see how you could improve your plot (node colors, size, layouts, etc.) Note that in many cases, it is simpler to do it with Gephi.

(l) Use the `pos` argument to plot nodes according to their geographical position. You can access node attributes either by using `get_node_attributes`, or by simply accessing the node, i.e.: `G.node['node_name']` gives existing attributes of this node, `G.node['node_name']["attribute1"]` gives the value of an attribute for a node

(m) It can be useful to export a graph with computed node or edges properties. Add their PageRank score to nodes as an attribute using `set_node_attributes`.

(n) Save the graph in graphml format using `write_graphml`. Check that you can open this file with Gephi and that the PageRank score is available as a node property.

2. Going further : Robustness

> When analyzing networks of infrastructure, transport or telecommunication, the *robustness* or *resilience* of the network is a popular investigation topic. Robustness can be defined as the capacity of the network to resist attacks and/or failures. The resistance can be measured in several ways, and attacks and failures can be modeled differently too. We will briefly investigate this question on the airport dataset.

(a) Let's define a simple measure of how *efficient* the network is as the fraction of nodes belonging to the largest connected component. Write a function computing this score, given a network.

(b) Write a function which, given an ordered list of nodes, remove them one by one from the network, compute the robustness measure at each step, and return the result.

(c) First, let's compute the resistance of the network to *failures*, i.e., random node removals. Generate a vector containing all nodes in a random order, and call the function defined above on it. Plot the results, i.e., robustness as a function of nodes removed. Repeat a few times to see how much it depends on the order of nodes.

(d) Let's now test the resistance to *attacks*, i.e., we will remove nodes in a specific order, and search for the most damaging strategy. Pick your favorites centralities, and check which one is the most efficient.

(e) Could you think of other strategies that could be even more efficient? Well, test them!