Experimenting with centralities

You can do the exercises in the order that you prefer.
the *Going further* experiments take more time and I do not expect everyone to do them.

1. Visualizing networks with Gephi
   (a) Using Menu File>Open, open one of the small networks available on the page of the class.
   (b) Using the bottom left panel, apply layouts to automatically position the nodes
   (c) Using the top left panel, assign the **size** of nodes to be proportional to their degree.
   (d) Using the bottom left panel, change the layout to adapt to these new sizes. Check, for instance, *expansion*, *noverlap*, and the *prevent overlap* option of *ForceAtlas 2*.
   (e) Use the button **T** at the bottom to display the name of nodes. Using another option at the bottom, make node names proportional to node size.
   (f) Using the *Statistics* tab of the right panel, compute PageRank
   (g) Using the top left panel, you can now assign a color scale to nodes corresponding to their PageRank score.
   (h) Have a look at the *Data Laboratory* window, accessible by clicking on the button of the same name at the top of your window. Check the data for both Nodes and Edges
   (i) Go back to *Overview* window, and, using the right panel, compute the different statistics. Observe the resulted plots, and then check all the new values available to assign sizes/colors.
   (j) Would you say that the network is a *small world* network ?
   (k) Use the *Filters* tab in the right panel to filter some nodes and/or edges in your graph. For instance, remove the nodes of lower degree, edges of lower weight, edges of higher betweenness, etc.

2. Finding important nodes with python and networkx.

   `networkx` package has a good documentation, but the best way to find what you're searching 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 availbale on my webpage.

   (a) Using networkx, load the airport dataset using `read_edgelist`.
   (b) Compute basic statistics on your graph, i.e., number of nodes and edges, density...

(c) Obtain the list of the 20 nodes of higher and lower degrees. You can use `degree()` and `sorted(X, key=lambda x: x[1]` for instance. Does it mach you expectation?

(d) Compare with the list of the 10 nodes of higher and lower pagerank. You might need to use `items` to transform a dictionary in a list of pair. Observe the differences

(e) 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?

(f) Check with other typical node centralities.

(g) Check edge betweenness.

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

3. 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!

4. Going further : Eigenvectors and Power method

(a) Using `networkx` to model a small graph (use for instance `nx.karate_club_graph()`, compute eigenvector centrality with the power method.

(b) Compute eigenvector centrality on the same graph using `numpy` to compute eigenvectors and eigenvalues. You can get the adjacency matrix of a graph with a networkx function.

(c) Check that both scores are similar.