Open and virtualized networks - Python Exercises - Part 3

These exercises sets covers some aspects you will find useful for the final exam software development. These exercises are NOT part of the final exam and their goal is to make you get used to Python programming. You are strongly encouraged to find yourself a solution to the presented problems.

Final Exercise

The aim of these exercises is to build a software abstraction of the network represented in Fig.1 and simulate the signal propagation from an input node to an output node. In the whole exercise, define the 'constructor' and the 'getter' and 'setter' methods for all the class's attributes, for any class.

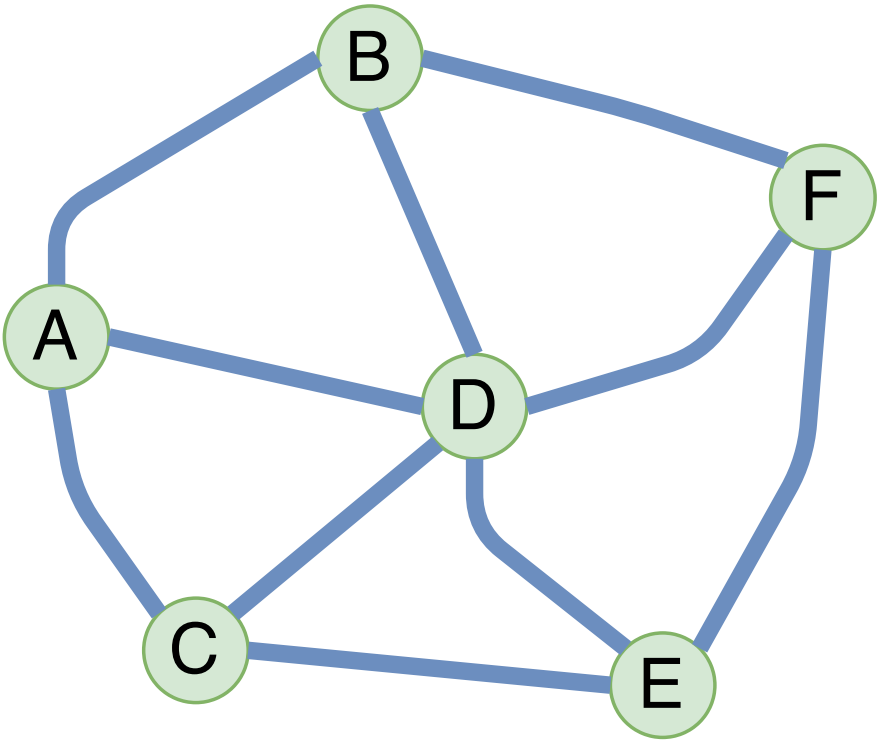

Figure 1: Network

1. Define the class **Signal_information** that has the attributes:
    - **signal_power**: float

- **noise_power**: float
- **latency**: float
- **path**: list[string]

such that its constructor initializes the signal power to a given value, the noise power and the latency to zero and the path as a given list of letters that represents the labels of the nodes the signal has to travel through. The attribute **latency** is the total time delay due to the signal propagation through any network element along the path. Define the methods to update the signal and noise power and the latency given an increment of these quantities. Define a method to update the path once a node is crossed.

2. Define the class **Node** that has the attributes:

   - **label**: string
   - **position**: tuple(float, float)
   - **connected_nodes**: list[string]
   - **successive**: dict[Line]

   such that its constructor initializes these values from a python dictionary input. The attribute **successive** has to be initialized to an empty dict. Define a propagate method that update a signal_information object modifying its path attribute and call the successive element propagate method, accordingly to the specified path.

3. Define the class **Line** that has the attributes:

   - **label**: string
   - **length**: float
   - **successive**: dict[Node]

   The attribute **successive** has to be initialized to an empty dict. Define the following methods that update an instance of the **signal_information**:

   - **latency_generation**(): float
   - **noise_generation**(signal_power): signal_power/(2*length)

   The light travels through the fiber at around 2/3 of the speed of light in the vacuum. Define the line method **latency_generation** accordingly. Define a propagate method that updates the signal information modifying its noise power and its latency and call the successive element propagate method, accordingly to the specified path.

4. Define the class **Network** that has the attributes:

   - **nodes**: dict[Node]

- **lines**: dict[Lines]

both the dictionaries have to contain one key for each network element that coincide with the element **label**. The value of each key has to be an instance of the network element (**Node** or **Line**). The constructor of this class has to read the given JSON file 'nodes.json', it has to create the instances of all the nodes and the lines. The line labels have to be the concatenation of the node labels that the line connects (for each couple of connected nodes, there would be two lines, one for each direction, e.g. for the nodes 'A' and 'B' there would be line 'AB' and 'BA'). The lengths of the lines have to be calculated as the minimum distance of the connected nodes using their positions. Define the following methods:

- **draw**(): this function has to draw the network using matplotlib (nodes as dots and connection as lines).
- **find_paths**(string, string): given two node labels, this function returns all the paths that connect the two nodes as list of node labels. The admissible paths have to cross any node at most once
- **connect()**: this function has to set the successive attributes of all the network elements as dictionaries (i.e., each node must have a dict of lines and each line must have a dictionary of a node)
- **propagate**(signal_information): this function has to propagate the signal_information through the path specified in it and returns the modified spectral_information

5. For all possible paths between all possible node couples, create a pandas dataframe that contains the path string as "A->B-> ...", the total accumulated latency, the total accumulated noise and the signal to noise ratio obtained with the propagation through the paths of a spectral_information with a signal_power of 1. Calculate the signal to noise ratio in dB using the formula $10\log(signal\_power/noise\_power)$