Visualization of network graphs

K. Bertet

Objectives : The objective of this practical work is to handle and visualize network graphs using :

- NetworkX : a Python package for the creation, manipulation and visualization of the structure of graphs and networks.
  - https://networkx.org/documentation/stable/reference/introduction.html#networkx-basics
  - https://networkx.org/documentation/stable/reference/drawing.html
- Graphviz: an open source graph visualization software
  - http://magjac.com/graphviz-visual-editor/
  - https://graphviz.readthedocs.io/en/latest/examples.html

1) Creation and visualization of graphs

Exercice 1. Edit and view the following graph with Graphviz.

digraph G {
a - > c
a - > d
d - > e
b - > d }

Exercice 2. Test different visualization engine and export the graph in the DOT format to the file graph.dot. Consult the documentation of dot, and in particular the different attributes of arcs and nodes :

- The *color* attribute to color a node or an edge,
- The *label* attribute to associate a label to a node or an edge
- The distinction between a directed and a undirected graph

Exercice 3. Execute the following python code to generate and visualize a random graph

```
import matplotlib.pyplot as plt
from networkx import nx
# Random graph with 10 nodes and 20 edges
G = nx.gnm_random_graph(10,20)
# Visualization of the graph
nx.draw_networkx(G)
plt.show()
```

Exercice 4. Execute the following python code to create a graph from a dot file :

```
x.Graph(read_dot("graph.dot"))
nx.draw_networkx(G)
plt.show()
```

Exercice 5. Implement a function *divGraph(n)* to generate the graph of divisors where nodes are integers between 2 and n, and there is an edge between i and j if i is a multiple of j.

Exercice 6. Implement a function *mydraw(G,edges)* to color some edges of a graph G. You can achieve this by creating an initial drawing using *draw_networkx(G)* and then redraw the edges on top of it (at the same position, see the second parameter of the *draw_networkx_edges* function).

Exercice 7. Implement the *breadth-first search* (G, s) function for a graph that takes a graph and a starting node sas input and returns the list of edges (or arcs) of the breadth-first search tree starting from the given node. We assume that the graph is (strongly) connected, allowing us to implement only the internal loop of the BFS. You can use a set of visited vertices instead of coloring the vertices during the traversal. Thus, if a vertex is in the set, it has already been visited; otherwise, it has not.

Exercice 8. Implement the *depth-first-search (G,s)* function that takes a graph G and a starting node s as input and returns the list of edges (or arcs) of the depth-first search tree of the graph starting from the given node.

2) The graph of the Parisian metro

The file *metro.graph* contains data describing the graph of the Parisian subway :

- Each node corresponds to a station for a given line (for example, République [line 3] and République [line 5] are two different nodes). Each node is associated with the name of the station (character string) and the position of the station on a map (scale: 1~25.7m).
- An oriented and attributed edge connects two nodes if the metro directly connects the corresponding stations. The graph is not symmetrical because of some « one-way » lines, for example at the Porte d'Auteuil station. The edges are attributed by the estimated travel time in seconds (assuming an average speed of 10m/s, or 36km/h).
- Two symmetrical edges connect two nodes if it is possible to walk without changing tickets between the corresponding stations. These nodes are then attributes by an estimate of the average travel and waiting time (120s).

Exercice 9. Create the graph of the Paris subway, with :

- Nodes attributes : position of the stations, name (String)
- Edges attributes : travel time