TP4 – Manipulating graphs with `NetworkX`[1] `Python` package

Do not hesitate to ask questions about previous TPs.

1 First use of `NetworkX`

The python package `NetworkX` can be installed easily in bash with `python3.8 -m pip install networkx` and be used in `Python3` programs thanks to the line `import networkx as nx`

Question 1.

> In a python script, import the package and create the graphs G and F from respectively Figure (1) and (2). Be careful with the graphs' attributes: are they directed or not, *etc.*?
> You can refer to the package tutorial:
> `https://networkx.org/documentation/stable/tutorial.html`

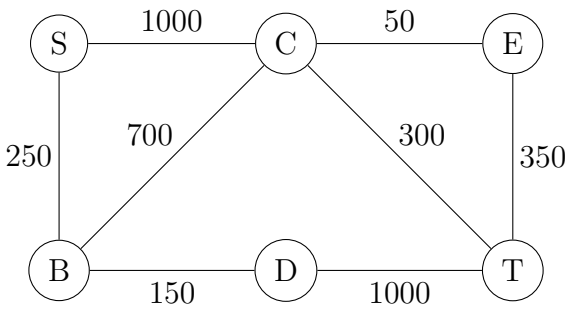

Figure 1: The undirected graph $G = (V, E, w_e)$ such that V is the set of vertices, E the set of edges. $\forall e \in E$, $w_e \in \mathbb{N}^*$ is the weight of the edge e.

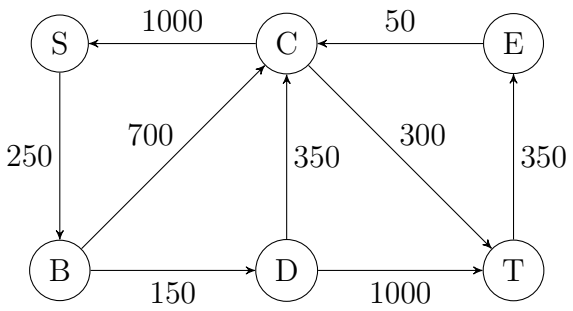

Figure 2: The directed graph $F = (V, E, w_e)$ such that V is the set of vertices, E the set of edges. $\forall e \in E$, $w_e \in \mathbb{Z}^*$ is the weight of the edge e.

2 Find an elementary path in graph

While some algorithms exist to compute the shortest or the longest path in a graph, they can suffer from graph structure: some cannot find a path in a negative weighted graph (*e.g.* Dijkstra algo.), other cannot find if there is a negative cycle (*e.g.* Bellman-Ford algo.).

In this section, we will learn to find a shortest (a lightest) / longest (a heaviest) path from a source to a target in any graph $G = (V, E, x_e)$.

[1] `https://networkx.org/`

An elementary path means that all vertex participating in this path has at most one successor and at most one predecessor. The source is the first vertex so it has only one successor in the path, the target is the last so it has only one predecessor in the path.

Question 2.

Given the node identifiers for the starter s and the target $target$, and from description above, give the variables associated to the problem of finding a path. No code is asked for now. They must describe:

- the starter has only one successor in the path
- the target has only one predecessor in the path
- if a vertex is intermediate in the path
- if an edge is participating to the path

Question 3.

What is the objective function in the case of finding a lightest path? A heaviest path?

Question 4.

What are the constraints to find only one elementary path?

Question 5.

In any graph, it is possible to have some cycles. To avoid them in order to not loop, we can say a vertex in the path is used only one time.
Let $pos_v \in \mathbb{N}$, $\forall\, v \in V$ the position of vertex v in the path such that:

$$pos_v = \begin{cases} 0 \text{ if } v \text{ not in the path} \\ >= 1 \text{ else} \end{cases}$$

Thanks to this new variable and the first ones, how can we avoid cycles? You should find a property on vertices' position in the path.

Question 6.

The linear constraint associated with the previous property is named a Miller-Tucker-Zemlin (MTZ) sub-tour elimination constraint. It consists to use a huge number called "big M" as bound.
How can we use this big M to ensure the property and help the solver to give a correct value to variable pos_v?

3 It is time to code!

Question 7.

> For each objective function in question 3, implement `PuLP` models to find elementary path from start node S to target node T in a given undirected graph.
> You can test your model with graph G from figure (1)

Question 8.

> Do the same as question 7 but for a given directed graph.
> You can test your model with graph F from figure(2)