graphes

March 3, 2020

[ ]:
```
import networkx as nx
```

1 Construction de graphes

1.1 Comment créer un graphe vide

[ ]:
```
gr = nx.Graph()
```

ATTENTION on a ici créer un graphe non orienté on utilisera `nx.DiGraph` pour avoir un graphe orienté avec des variantes pour les opérations ci-dessous.

1.2 Comment ajouter un noeud

[ ]:
```
gr.add_node("A")
```

ATTENTION : les objets utilisés pour créer les noeuds doivent être `hashable` donc moralement immutable.

1.3 Comment ajouter plusieurs noeuds simultanément

En passant un conteneur d'objets.

[ ]:
```
gr.add_nodes_from("BCDE")
gr.add_nodes_from(["F", "G", "H"])
```

1.4 Comment enlever un/des noeud

[ ]:
```
gr.remove_node("H")
gr.remove_nodes_from("EFH")
```

1.5 Comment ajouter une arrête

[ ]:
```
gr.add_edge("A", "B")
```

ATTENTION les noeuds passés pour définir une arrête seront rajoutés s'il ne sont pas déjà présent dans la liste de noeuds.

[ ]:
```
gr.add_edge("U", "V")
```

1.6 Comment ajouter plusieurs arrêtes simultanément

[ ]:
```
gr.add_edges_from([("A", "C"), ("A", "D"), ("B", "C"), ("A", "B")])
```

1.7 Comment enlever une/des arrêtes

[ ]:
```
gr.remove_edge("U", "V")
gr.remove_edges_from((("A", "C"), ("A", "D")))
```

1.8 Comment ajouter des propriétés aux noeuds

[ ]:
```
gr.nodes["A"]["depart"] = True
```

1.9 Comment ajouter des propriétés aux arrêtes

[ ]:
```
gr.edges["A", "B"]["poids"] = 1
```

1.10 Comment ajouter des arrêtes avec propriétés simplement

[ ]:
```
gr.add_weighted_edges_from([("A", "U", 2), ("A", "V", 3), ("C", "U", 1), ("D",␣
↪"V", 5)], weight="poids")
```

2 Analyse de graphes

2.1 Comment voire le nombre de noeuds/sommets

[ ]:
```
print(gr.number_of_edges())
print(gr.number_of_nodes())
```

2.2 Comment voire les noeuds

[ ]:
```
gr.nodes
```

2.3 Comment voire les arrêtes

[ ]:
```
gr.edges
```

2.4 Comment voire les degrés des noeuds

[ ]:
```
gr.degree
```

2.5 Comment voire les voisinages

[ ]:
```
gr.adj
```

2.6 Comment accéder aux voisins d'un noeud

[ ]:
```
gr["A"]
```

3 Visualisation de graphes

3.1 Comment visualiser un graphe

Il faut d'abord importer pyplot et modifier les options si on le veut

[ ]:
```
import matplotlib.pyplot as plt
```

[ ]:
```
plt.rcParams["figure.figsize"] = (12, 8)
```

[ ]:
```
nx.draw(gr)
```

3.2 Comment visualiser avec les noms des sommets

[ ]:
```
nx.draw(gr, with_labels=True)
```

4 Comment afficher un graphe dans une figure/repère spécifique

pour par exemple sauver la figure dans un fichier avec la méthode `savefig`.

[ ]:
```
fig, rep = plt.subplots()
nx.draw(gr, ax=rep, with_labels=True)
```

4.1 Comment récupérer les positions des sommets

[ ]:
```
positions = nx.spring_layout(gr)
positions
```

4.2 Comment afficher aussi les propriétés des arrêtes

[ ]:
```
nx.draw_networkx(gr, positions)
poids = nx.get_edge_attributes(gr, "poids")
nx.draw_networkx_edge_labels(gr, positions, edge_labels=poids)
```

5 Algorithmes de graphes

De nombreux autres algorithmes sont disponibles en plus de ceux couverts en dessous.

5.1 Comment trouver si le graphe est connexe (i.e. tous les couples de sommets sont reliés par au moins un chemin)

[ ]:
```
nx.is_connected(gr)
```

5.2 Comment récupérer les composantes connexes

[ ]:
```
list(nx.connected_components(gr))
```

5.3 Comment trouver le chemin le plus court (suivant un attribut)

[ ]:
```
print("chemin le plus court de V à C: ", nx.dijkstra_path(gr, "V", "C",␣
 ↪weight="poids"))
print("et sa longueur : ", nx.dijkstra_path_length(gr, "V", "C", weight="poids"))
```

ATTENTION on à utiliser ici l'algorithme de Dijkstra mais d'autres algorithmes et d'autres fonctions sont disponibles.

5.4 Comment trouver le flot maximal d'un graphe

[ ]:
```
nx.flow.maximum_flow(gr, "C", "D", capacity="poids")
```