Université Clermont Auvergne
BUT Info 1A
R207 - Graphes
Année 2021-2022

Florent Foucaud
Malika More
Thibault Ralet
Carine Simon
Thierry Trévisan

TP1 - Découverte des graphes en Python avec NetworkX

Nous allons utiliser la librairie `NetworkX` pour `Python3`.[1] On peut trouver un tutoriel et de la documentation en ligne (en anglais).

Exercice 1 (Prise en main)**.**

1. Tester le code Python suivant.

```
import networkx as nx                          #pour la gestion des graphes
import matplotlib.pyplot as plt                #pour l'affichage
                                               #(on les renomme pour raccourcir le code)
G1 = nx.Graph()                                #crée un graphe non-orienté vide
G1.add_edge(0,1)                               #ajoute une arête entre les sommets 0 et 1
G1.add_edges_from([(3,0),(3,4)])               #ajoute les arêtes d'une liste donnée
nx.add_path(G,[1,2,3])                         #ajoute les arêtes du chemin 1-2-3
G1.add_node(6)                                 #ajoute un sommet
G1.add_node("toto")
G1.add_edge("toto",6)
nx.draw(G1,with_labels=True)                   #on prépare le dessin du graphe
plt.show()                                     #on affiche le dessin
```

2. Tester les fonctions suivantes.

```
G1.remove_node(s) #Supprimez un sommet s de votre choix, puis ré-affichez le graphe.
print(G1.nodes)
print(G1.edges)
```

3. Par défaut, le dessin de graphe utilise un algorithme par "modèle de forces" avec une part de hasard, et peut changer à chaque appel. On peut dessiner le graphe autrement, par exemple avec une disposition circulaire et en changeant les couleurs :

```
nx.draw(G1, with_labels=True, pos=nx.circular_layout(G1), node_color='r', edge_color='b')
```

Ou bien en spécifiant à la main les coordonnées de chaque sommet dans un dictionnaire :

1. Si nécessaire, pour installer NetworkX, il faut taper `pip3 install networkx` dans un terminal. À l'IUT il est normalement déjà installé.

```
dico_positions = {0:(0,0),1:(1,0),2:(0,1),3:(1,1),4:(0,-1),6:(-1,-1),"toto":(0,-1)}
nx.draw(G1, with_labels=True, pos=dico_positions)
```

Exercice 2 (Un exemple de graphe)**.**

Le graphe de la Figure 1 représente le plan d'un bâtiment avec ses six salles.

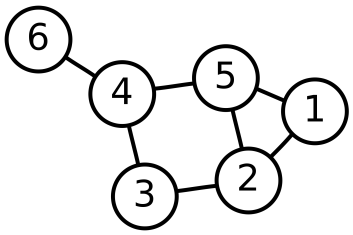

FIGURE 1 – Un graphe non-orienté à six sommets.

1. Écrire le code qui permet d'afficher ce graphe, on l'appellera par exemple `H`.
2. Écrire le code qui permet d'afficher les salles directement connectées à la salle 5, en utilisant la méthode `H.edges` qui renvoie la liste des arêtes de `H`. *(Attention : une arête entre* `i` *et* `j` *peut être stockée sous forme* `(i,j)` *ou* `(j,i)`*.)*
3. Ajouter une connexion entre la salle 5 et la salle 3 et vérifier que le code précédent donne toujours le bon résultat.

Exercice 3 (Fonctions de base pour les graphes)**.**

Coder et tester les fonctions suivantes (en utilisant seulement `G.nodes` et `G.edges` qui renvoient la liste des sommets et des arêtes d'un graphe `G`, respectivement).

1. `nombre_sommets(G)`, qui renvoie le nombre de sommets d'un graphe `G`.
2. `nombre_aretes(G)`, qui renvoie le nombre d'arêtes d'un graphe `G`.
3. `existe_arete(G,s1,s2)`, qui renvoie `True` s'il existe une arête entre `s1` et `s2` dans un graphe `G`, et `False` sinon. *(Attention : l'arête peut être stockée sous forme* `(s1,s2)` *ou* `(s2,s1)`*.)*
4. `voisins(G,s)`, qui renvoie la liste des voisins du sommet `s` dans un graphe `G`. Utiliser la fonction `existe_arete(G,s1,s2)`.
5. `degre(G,s)`, qui renvoie le degré (nombre de voisins) du sommet `s` dans un graphe `G`.
6. `degre_max(G)`, qui renvoie le degré maximum d'un sommet dans un graphe `G`.
7. `sommets_de_degre_max(G)`, qui renvoie la liste des sommets d'un graphe `G` de degré maximum.

Exercice 4 (Graphes orientés)**.**

On peut aussi créer et afficher des graphes orientés, avec le code ci-dessous par exemple. Observer la particularité du dessin des arcs entre les sommets `0` et `1`.

```
GO = nx.DiGraph()                          #on crée un graphe orienté
GO.add_edge(0,1)
GO.add_edge(1,0)
GO.add_edge(1,2)
GO.add_edge(3,2)
GO.add_edge(3,1)
nx.draw(GO,with_labels=True)               #on prépare le dessin du graphe
plt.show()                                 #on affiche le dessin
```

Exercice 5 (Fonctions de base pour les graphes orientés)**.**

Coder et tester les fonctions suivantes (en utilisant seulement les méthodes `G.nodes` et `G.edges` qui renvoient la liste des sommets et des arêtes du graphe orienté `G`, respectivement).

1. `existe_arc(G,s1,s2)`, qui renvoie `True` s'il existe un arc de `s1` à `s2` dans un graphe orienté `G`, et `False` sinon.
2. `voisins_entrants(G,s)`, qui renvoie la liste des sommets qui ont un arc vers `s` dans un graphe orienté `G`. Utiliser la fonction `existe_arc(G,s1,s2)`.
3. `degre_entrant(G,s)`, qui renvoie le degré entrant du sommet `s` dans un graphe `G` (nombre de voisins qui ont un arc vers `s`).
4. `est_source(G,s)`, qui renvoie `True` si le sommet `s` est une *source* du graphe `G` (un sommet sans aucun arc entrant) et `False` sinon.

Exercice 6 (Labyrinthe **)**.**

Considérons le labyrinthe de la Figure 2. Chaque position dans le labyrinthe sera notée par `"i-j"` où i désigne l'indice de ligne et j l'indice de colonne. Avec ces notations, l'entrée du labyrinthe se situe donc en `"4-0"` et la sortie en `"4-5"`.

1. Modéliser le labyrinthe par un graphe non orienté, en respectant les noms des sommets. Il est conseillé d'ajouter les arêtes avec la fonction `nx.add_path(G,[s,...,t])` pour gagner du temps.
2. Afficher ce graphe.
3. Combien y a-t-il de chemins possibles (sans faire demi-tour) entre deux cases quelconques du labyrinthe ? À quelle famille spéciale de graphes appartient le graphe qui représente le labyrinthe ?
4. (**) Coder une fonction `chemin(s)` qui trouve un chemin depuis l'entrée `"4-0"` vers un sommet `s` donné (on peut répéter des cases). On affichera les cases parcourues lors de l'exploration. Lors de l'exploration du labyrinthe, on pourra se souvenir des sommets déjà parcourus. Un itérateur sur les voisins d'un sommet `s` est donné par `G.neighbors(s)` (on obtient la liste des voisins avec `list(G.neighbors(s))`).

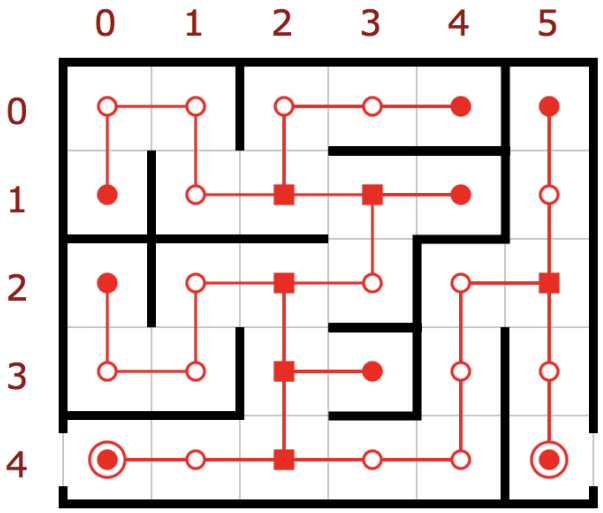

FIGURE 2 – Un labyrinthe.