Université Clermont Auvergne
BUT Info 1A
R207 - Graphes
Année 2022-2023

Florent Foucaud
Malika More
Thibault Ralet
Carine Simon
Thierry Trévisan

TP3 - Les parcours de graphes

Fonctions de base dans NetworkX (graphes non-orientés)

Tutoriel et documentation de NetworkX en ligne (en anglais).
Dans ce TP quelques fonctions de bases pourraient être utiles :

```
H=nx.Graph()                      #crée un graphe
H.add_edge(0,1)                   #ajoute une arête entre les sommets 0 et 1
H.add_edges_from([(3,0),(3,4)]) #ajoute les arêtes d'une liste donnée
H.add_node("toto")                #ajoute un sommet nommé "toto"
H.remove_node(s)                  #supprime le sommet s
H.nodes                           #sommets du graphe (attention, pour en faire
                                   une vraie liste Python, écrire: list(H.nodes))
H.edges                           #arêtes du graphe (attention, pour en faire
                                   une vraie liste Python, écrire: list(H.edges))
H.edges(s)                        #les arêtes qui touchent le sommet s

H.neighbors(s)                    #un itérateur sur les voisins du sommet s dans H
                                   pour obtenir une liste, écrire: list(H.neighbors(s))
H.nodes[s]["attri"]               #accède à l'attribut nommé "attri" du sommet s
                                   (tant en lecture qu'en écriture)
                                   exemple: H.nodes[s]["attri"]=2
                                   ou alors print(H.nodes[s]["attri"])
```

Exercice 1 (La frontière)**.**

Pour parcourir un graphe, l'important est la structure de données qui stocke la frontière, c'est-à-dire l'ensemble des sommets visités mais pas encore entièrement explorés. Dans cet exercice, vous allez coder des fonctions permettant de réaliser les opérations élémentaires sur cette structure de données : ajouter un élément, enlever un élément, tester si la structure de données est vide, accéder au prochain élément dans la structure de données.

Dans tous les cas, on vous demande d'utiliser une liste `Python3`**.** Utilisez les méthodes `append` et `pop` pour ajouter et enlever des éléments d'une liste. On écrit par exemple `L.append(x)` pour ajouter un élément `x` en fin de liste `L`. Par ailleurs, `L.pop(i)` retire et

renvoie l'élément à l'indice `i` de la liste `L` (`L.pop(-1)` retire le dernier élément). On peut aussi utiliser `del L[i]` pour supprimer l'élément de `L` à l'indice `i`. Le nombre d'éléments d'une liste `L` est donnée par `len(L)`.

1. Quelle est la structure de données qui permet d'effectuer un parcours en profondeur ? Un parcours en largeur ?
2. Pour une file `F`, et un élément `v`, la fonction ci-dessous ajoute l'élément `v` à la file `F`.

   ```
   def ajouter_file(F,v):
     F.append(v)
     return
   ```

   Écrire les fonctions `enlever_tete_file(F)`, `est_file_vide(F)`, `valeur_tete_file(F)`.
3. Pour une pile `P`, et un élément `v`, la fonction ci-dessous ajoute l'élément `v` à la pile `P`.

   ```
   def ajouter_pile(P,v):
     P.append(v)
     return
   ```

   Écrire les fonctions `enlever_sommet_pile(P)`, `est_pile_vide(P)`, `valeur_sommet_pile(P)`.

Exercice 2 (Les parcours en largeur et en profondeur)**.**

On rappelle l'algorithme de parcours générique, vu en cours.

Parcours du graphe G à partir du sommet ''source'' s
- V est la liste des sommets visités. Contient initialement seulement s.
- F est la frontière : les sommets visités mais qui ont peut-être encore des voisins non visités. Contient initialement seulement s.
- Répéter tant que la frontière F est non-vide :
  - ⋆ Considérer un sommet x de la frontière F.
  - ⋆ Si il existe un voisin y de x pas encore dans V :
    - ajouter y dans V et dans F
  - ⋆ Sinon :
    - enlever x de F
- Retourner V

- Dans le parcours en largeur, la frontière F est une file. Pour considérer un sommet de la frontière on regarde toujours la tête de file. Pour enlever un sommet on défile, et pour en ajouter un, on enfile. Ainsi, l'algorithme inspecte tous les voisins d'un sommet avant de passer au suivant.
- Dans le parcours en profondeur, la frontière F est une pile. Pour considérer un sommet de la frontière on regarde toujours le haut de la pile. Pour enlever un sommet on dépile, et pour en ajouter un, on empile. Ainsi, l'algorithme va de sommet en sommet jusqu'à être bloqué et revenir sur ses pas.

À l'aide des fonctions écrites dans le premier exercice :

1. écrire la fonction `parcours_largeur(G,s)` qui effectue un parcours en largeur du graphe `G` à partir du sommet `s`, et qui retourne la liste des sommets visités.

2. écrire la fonction `parcours_profondeur(G,s)` qui effectue un parcours en profondeur du graphe `G` à partir du sommet `s`, et qui retourne la liste des sommets visités.

Ne pas oublier de tester vos programmes, par exemple sur le graphe `G` suivant :

```
####################
G=nx.Graph()
G.add_edges_from([(1,2),(1,5),(2,3),(2,5),(3,4),(4,5),(4,6),(5,0)])

nx.draw(G,with_labels=True)
plt.show()
####################
```

Exercice 3 (Variante : les trésors)**.**

Le code ci-dessous a pour effet de créer un graphe `G1` avec des trésors sur les sommets (un trésor est un entier).

```
#########################
G1=nx.Graph()
G1.add_edges_from([("a","f"),("a","b"),("f","c"),("f","e"),("f","d"),
                   ("c","e"),("c","b"),("c","d"),("e","d")])
G1.nodes["a"]["tresor"]=3
G1.nodes["f"]["tresor"]=4
G1.nodes["c"]["tresor"]=5
G1.nodes["e"]["tresor"]=1
G1.nodes["d"]["tresor"]=2
G1.nodes["b"]["tresor"]=0

## pour afficher le graphe avec les trésors associés à chaque sommet
dico_positions_sommets = {"a":(0,0),"b":(0,2),"c":(2,2),
                          "d":(4,2),"e":(4,0),"f":(2,0)}
dico_positions_tresors = {"a":(0,-0.1),"b":(0,2.1),"c":(2,2.1),
                          "d":(4,2.1),"e":(4,-0.1),"f":(2,-0.1)}
nx.draw(G1,dico_positions_sommets,with_labels=True)
nx.draw_networkx_labels(G1,dico_positions_tresors,
                        labels=nx.get_node_attributes(G1,"tresor"))
plt.show()
#########################
```

1. Écrire une fonction `tresor(G,s)` prenant en entrée un graphe `G` et un sommet `s` et renvoyant le trésor présent sur le sommet `s`.
2. Écrire une fonction `parcours_tresor(G,s)` qui prend en entrée un graphe `G` et un sommet `s` de départ, et parcourt le graphe en choisissant de traiter à chaque étape le sommet de la frontière (déjà connu mais non traité) ayant le plus grand trésor. La fonction doit renvoyer la liste des sommets traités dans l'ordre de traitement.

Exercice 4 (Variante : l'incendie)**.**

On veut simuler la propagation d'un incendie dans un bâtiment. On représente le bâtiment par un graphe, dont les sommets sont les salles, et une arête entre deux salles indique qu'elles communiquent, et que le feu va se propager de l'une à l'autre. On considère que chaque minute, le feu envahit toutes les salles qui communiquent avec une salle en feu ; les salles qui sont en feu continuent de brûler indéfiniment.

1. Écrire une fonction `propagation(G,E)` qui, à partir d'un graphe `G` et d'un ensemble `E` de sommets représentant des salles en feu, donne l'ensemble `F` des salles qui seront en feu 1 minute plus tard.
2. Écrire une fonction `propagation_temps(G,E,t)` qui indique quelles salles seront en feu dans `t` minutes si `E` est l'ensemble des salles en feu maintenant.
3. Si une seule salle est en feu dans le bâtiment, quel est l'ensemble des salles qui prendront feu à un moment ou à un autre ?
4. Représenter graphiquement la propagation du feu à partir d'une salle.