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

Florent Foucaud
Malika More
Thibault Ralet
Carine Simon
Thierry Trévisan

TP5 - Calculs de distances

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 (Algorithme de Dijkstra - distances dans les graphes valués)**.**

Dans cet exercice, on considère des graphes valués, c'est-à-dire que les arêtes possèdent un **poids** positif ou nul. La *distance* entre deux sommets est la plus petite somme des poids des arêtes parmi les chaînes qui relient les deux sommets. Ces poids sont représentés par l'attribut `weight` des arêtes du graphe dans la librairie `networkx` de Python. De plus, on crée un attribut `distance` pour les sommets qui représentera la distance à la source. Les distances sont d'abord toutes initialisées à `None` comme dans l'exemple ci-dessous :

```
##################################
##Un graphe pour tester Dijkstra
##################################
print("Un graphe pour tester Dijkstra")
G=nx.Graph()
G.add_nodes_from(["A","B","C","D","E","F","G","H","I","J"],distance=None)
print(G.nodes())
G.add_edges_from([("A", "B", {"weight": 4}),("A", "C", {"weight": 2}),
                  ("A", "E", {"weight": 1}),("B", "F", {"weight": 3}),
                  ("C", "G", {"weight": 1}),("C", "H", {"weight": 2}),
                  ("D", "H", {"weight": 1}),("E", "J", {"weight": 5}),
                  ("F", "I", {"weight": 2}),("I", "J", {"weight": 5}),
                  ("H", "J", {"weight": 6})])
print(G.edges())
########################

print(G.edges[("A","B")]["weight"])
print(G.nodes["A"]["distance"])
########################
```

On rappelle ci-dessous l'algorithme de Dijkstra qui calcule les distances à un sommet source donné :

Algorithme de Dijkstra pour le graphe G à partir du sommet source s

- L représentera la frontière. Contient initialement seulement s.
- La valeur de distance d est initialisée à d(s)=0 et à d(v)=∞ pour chaque autre sommet v
- Une liste T contient les sommets qui ont été complètement traités. Initialement T est vide.
- Tant que L n'est pas vide :
  - ⋆ choisir un sommet v dans L qui a une valeur de distance d(v) minimale
  - ⋆ pour tout voisin w de v qui n'est pas dans T :
    - si d(v) plus le poids p de l'arête (v,w) est inférieur à d(w), on fixe d(w)=d(v)+p
    - ajouter w à L
  - ⋆ enlever v de L, ajouter v à T

Implémentez l'algorithme en suivant les étapes suivantes :

1. Écrire une fonction `mise_a_jour_voisin(G,n,v)` qui met à jour la distance du sommet `v` à la source (`v` est supposé voisin de `n` dans le graphe `G`) à partir de celle du sommet `n` (en supposant que la distance du sommet `n` à la source est un nombre).
2. Écrire une fonction `choix_prochain_sommet(G,L)`, qui renvoie le sommet de la frontière `L` qui sera choisi pour la prochaine itération de l'algorithme de Dijkstra.
3. Écrire une fonction `Dijkstra(G,s)` qui détermine les distances entre le sommet `s` et tous les autres sommets dans le graphe valué `G`.

Exercice 2 (Algorithme de Floyd-Warshall - calcul de toutes les distances)**.**

Coder et tester l'algorithme de Floyd-Warshall, rappelé ci-dessous :

`Algorithme de Floyd-Warshall pour le graphe G`

- `Les sommets sont ordonnés :` s_1 `...` s_n `avec` n `le nombre de sommets`
- `On initialise une matrice` D `de taille` $n \times n$`, où` $D(i,j)$ `devra contenir la distance entre le sommet` s_i `et le sommet` s_j`. Pour toute arête entre` s_i `et` s_j `de poids` p_{ij}`, on fixe` $D(i,j) = p_{ij}$`, et pour tout` i `on fixe` $D(i,i) = 0$`. Dans les autres cas, on fixe` $D(i,j) = \infty$`.`
- `Pour` k `allant de` 1 `à` n `:`
  - `Pour` i `allant de` 1 `à` n `:`
    - `Pour` j `allant de` 1 `à` n `:`

      $D(i,j) = \min\{D(i,j), D(i,k) + D(k,j)\}$
- `Renvoyer` D

Pourquoi n'est-t-il pas nécessaire de copier la matrice D à chaque étape ? Vérifier que l'algorithme ne va pas écraser des données importantes en cours de route.