Université Polytechnique Hauts-de-France
Institut National des Sciences Appliquées Hauts-de-France **2022-2023**
Spécialité IcY - 3A Graphes et Algorithmes
Semestre 6

Fiche d'exercices pratiques

Consignes & modalités

Les TPs sont à réaliser en présentiel.
Le travail individuel est encouragé mais les binômes sont autorisés. Dans tous les cas vous devez indiquer en haut de votre (vos) fichier(s) source(s) vos nom(s) et prénom(s).
Ne négligez pas les commentaires pour expliquer vos choix et les fonctions mises en œuvre.

Vous devez remettre une version de votre travail avant la fin de chaque séance sur `Moodle`. Celle-ci n'est composée que des fichiers sources réalisés pour le TP. Si vous ne déposez pas de travail la note associée sera 0 (**aucun retard accepté**). Une version finale pourra être déposée à partir du lendemain de la dernière séance, et dans la semaine qui suit celle-ci. Si elle est déposée, cette version finale sera prise en compte en complément dans la notation (**attention :** il s'agit bien d'un complément, et pas d'un remplacement).

Le langage de programmation est libre. **Attention** cependant, il n'est bien entendu pas autorisé d'utiliser des bibliothèques ou d'autres outils implémentant les fonctions demandées dans les exercices.

Dans tout le TP nous considérons des graphes **non orientés**.

Le respect des consignes sera pris en compte dans la notation.

Exercice 1 : structures de données

Nous considérerons deux représentations distinctes, celle par la matrice d'adjacence et celle par listes d'adjacence. Certains éléments sont communs aux deux représentations : le nombre de sommets, noté `n` dans la suite, et le nombre d'arêtes du graphe, noté `m`. Un sommet est caractérisé par un identifiant numérique (un entier) et un nom (stocké via une chaîne de caractères). L'identifiant numérique doit être unique.

Pour rappel, la représentation par matrice d'adjacence nécessite la définition d'une matrice carrée d'ordre `n` contenant un 1 dans la case d'indices `i` et `j` si les sommets `i` et `j` sont reliés par une arête, un 0 sinon (la matrice est symétrique dans le cas non orienté).
La représentation par listes d'adjacences nécessite la définition d'un ensemble de `n` listes de taille éventuellement différentes. La liste du sommet `i` contient les sommets reliés à `i` par une arête (les voisins de `i`).

Pour chacune des deux représentations, implémenter les structures de données permettant de regrouper toutes les informations utiles précisées ci-dessus, puis les fonctions suivantes.

1. `graphe_vide()` : crée un graphe vide sans sommet ni arête.
2. `add_sommet(G, i)` : ajoute le sommet `i` dans le graphe `G`, s'il n'existe pas déjà. Ce sommet n'a aucune arête incidente.
3. `add(G, i, j)` : ajoute l'arête `(i,j)` dans le graphe `G` si elle n'existe pas déjà.
4. `supp(G, i, j)` : supprime l'arête `(i,j)` dans le graphe `G` si elle existe.
5. `est_voisin(G, i, j)` : retourne 1 si le sommet `j` est voisin du sommet `i` dans le graphe `G`, 0 sinon.
6. `load(nom)` : crée un graphe via les informations contenues dans le fichier de nom physique `nom`.

7. `save(G, nom)` : sauvegarde les informations du graphe `G` dans le fichier de nom physique `nom`.

Pour les questions 6 et 7 le fichier dont le nom physique est en paramètre est un fichier texte ayant le format suivant : la première ligne contient uniquement le nombre de sommets du graphe (la valeur de `n`). Chacune des `n` lignes suivantes contient l'identifiant et le nom d'un sommet, séparés par un espace. Enfin, les `m` lignes suivantes contiennent les deux sommets extrémités de chacune des arêtes du graphe, séparés par un espace.

8. Implémenter une fonction `matrice_to_liste(G)` qui construit un graphe représenté par ses listes d'adjacence à partir du graphe `G` représenté par sa matrice d'adjacence.
9. Implémenter une fonction `liste_to_matrice(G)` qui fait l'opération inverse.

Attention

Dans toute la suite lorsqu'il est demandé d'implémenter une fonction avec les deux représentations il n'est pas autorisé d'utiliser une fonction des questions 8 et 9. Autrement dit il est attendu de proposer deux versions bien distinctes et prenant en compte les deux structures de données pour répondre à la question.

Exercice 2 : graphe partiel et sous-graphe

Quelques rappels du cours :

Définition 1 *Soient* `G = (S, A)` *et* `G' = (S, A')` *deux graphes.* `G` *est un* ***graphe partiel*** *de* `G'` *si* `A` *est inclus (strictement) dans* `A'`.

Définition 2 *Soient* `G = (S, A)` *et* `G' = (S', A')` *deux graphes. On dit que* `G` *est un* ***sous-graphe*** *de* `G'` *si :*

- `S` $\subset$ `S'`
- `A` $\subset$ `A'`
- `A` $=$ { `(i,j)` | `i` $\in$ `S`, `j` $\in$ `S`, `(i,j)` $\in$ `A'` }

Dans cet exercice en prendra en compte les noms des sommets dans la comparaison des ensembles de sommets. Cela signifie que deux graphes peuvent avoir les mêmes sommets (mêmes noms) mais pas avec les mêmes identifiants. En représentant les sommets par un couple `(numéro, nom)` on pourrait avoir par exemple :
`S = { (1, "valenciennes"), (2, "lille"), (3, "lens") }`, et
`S' = { (1, "lille"), (2, "lens"), (3, "valenciennes") }`.
Dans ce cas on considérera que `S = S'`.

Implémenter les fonctions suivantes, pour les deux représentations.

1. `inclus_sommet(G, G', strict)` qui renvoie 1 si les sommets du graphe `G` sont inclus dans les sommets du graphe `G'`, 0 sinon. Le paramètres `strict` indique si l'inclusion doit être stricte (dans ce cas il a la valeur 1) ou non (valeur 0).
2. `inclus_aretes(G, G')` qui renvoie 1 si les arêtes du graphe `G` sont incluses (strictement) dans les arêtes du graphe `G'`, 0 sinon.
3. `est_partiel(G, G')` qui renvoie 1 si le graphe `G` est un graphe partiel du graphe `G'`, 0 sinon.
4. `est_sous_graphe(G, G')` qui renvoie 1 si le graphe `G` est un un sous-graphe du graphe `G'`, 0 sinon.

Définition 3 *Soient* `G = (S, A)` *et* `G' = (S', A')` *deux graphes.* `G` *est un* ***sous-graphe partiel*** *de* `G'` *si* `G` *est un graphe partiel d'un sous-graphe de* `G'`.

Définition 4 *Une **clique** est un sous-graphe complet d'un graphe* `G = (S, A)`.

Définition 5 *Un **stable** est un sous-graphe d'un graphe* `G = (S, A)` *sans arêtes.*

Implémenter les fonctions suivantes, pour les deux représentations.

5. `est_sous_graphe_partiel(G, G')` qui renvoie 1 si le graphe `G'` est un sous-graphe partiel du graphe `G`.
6. `est_clique(G, G')` qui renvoie 1 si `G'` est une clique du graphe `G`.
7. `est_stable(G, G')` qui renvoie 1 si `G'` est un stable du graphe `G`.

Exercice 3 : rayon, diamètre et centre

Les graphes sont très utilisés dans l'analyse des réseaux sociaux. Certaines applications visant notamment à savoir comment faire diffuser au mieux une information reposent sur les définitions suivantes, dans lesquelles la distance entre deux sommets fait référence au nombre d'arêtes formant une plus courte chaîne entre les deux sommets.
Dans cet exercice on suppose que le graphe `G = (S,A)` utilisé est connexe.

Définition 6 *Soit* `G = (S, A)` *un graphe. L'**excentricité** d'un sommet* `s` *de* `S` *est la distance maximale entre* `s` *et les autres sommets du graphe.*

Définition 7 *Soit* `G = (S, A)` *un graphe. Le **centre** d'un graphe est le ou les sommets dont l'excentricité est la plus petite.*

Définition 8 *Soit* `G = (S, A)` *un graphe. Le **rayon** de* `G` *est l'excentricité d'un de ses centres.*

Définition 9 *Soit* `G = (S, A)` *un graphe. Le **diamètre** de* `G` *est la distance maximale entre deux sommets* `s` *et* `s'` *de* `S`.

Nous pouvons voir avec ces quelques définitions qu'il peut être intéressant de connaître les centres d'un graphe représentant un réseau (social ou autre). Ces sommets sont de bons points d'entrée pour atteindre le plus rapidement possible l'intégralité des sommets d'un réseau. Certains problèmes en lien avec cette notion de diffusion d'informations dans un réseau sont $\mathcal{NP}$-Difficiles, comme le problème de l'influence maximale dans les réseaux sociaux. En effet il est un peu réducteur de se dire qu'en passant par le centre d'un réseau l'information arrivera le plus vite à l'ensemble des sommets. La vitesse de propagation dans un réseau social dépend de l'activité des différents sommets et de leur influence pour que l'information soit diffusée. D'autres applications existent comme en cybersécurité où nous pouvons facilement comprendre l'intérêt de connaître les centres d'un réseau. Dans ce TP nous resterons à un niveau relativement simple.

Implémenter les fonctions suivantes, pour les deux représentations.

1. `calcul_distances(G)` qui calcule les plus courtes distances (en nombre d'arêtes) entre tout couple de sommets du graphe `G`.
2. `donne_diametre(G, D)` qui renvoie le diamètre du graphe `G` à partir de la structure de données `D` (au choix) contenant les plus courtes distances entre tout couple de sommets de `G`.
3. `donne_centres(G, D)` qui retourne le nombre de centres du graphe `G`, la liste des centres (structure de données au choix), et le rayon de `G`.

Il faut noter qu'il existe d'autres définitions de la notion de "centre" d'un graphe, notamment en fonction du contexte. En fait tout dépend de la mesure qui est introduite pour départager les sommets. Dans certains applications c'est le degré des sommets qui est considéré : le centre du graphe est le

sommet ayant le plus grand degré.

Implémenter les fonctions suivantes, pour les deux représentations.

1. `calcul_degres(G)` qui détermine le degré de chaque sommet du graphe `G`.
2. `donne_centres_degre(G, D)` qui retourne le nombre de centres du graphe `G` en considérant le degré maximum comme critère, la liste des centres (structures de données au choix), et le degré maximum d'un sommet de `G`.

Remarque : la notion de centralité de vecteur propre est une autre mesure utilisée par exemple dans l'algorithme *PageRank* sur Internet qui analyse les liens et génère le classement des pages Web lors d'une recherche via `Google`. Il s'agit d'une extension de la mesure basée sur le degré qui prend en compte les connexions aux sommets non voisins directs (en pénalisant les sommets les plus éloignés).

Exercice 4 : graphes aléatoires

Dans cet exercice vous allez implémenter un algorithme de génération aléatoire de graphes. Différentes techniques existent dans la littérature. Parmi celles-ci nous utiliserons une des deux méthodes proposées par Erdös-Rényi, celle notée $G(n, p)$. Le principe peut être résumé comme suit :

— Choisir une probabilité p ;

— Créer n sommets ;

— Chaque paire (x, y) de sommets est connectée de façon indépendante selon la probabilité p.

Cette méthode peut être implémentée de façon assez simple. Le paramètre p permet d'impacter sur la densité du graphe : plus p est proche de 1 et plus le graphe est dense (avec les cas extrêmes $p = 0$ et $p = 1$ correspondant à un graphe sans connexion et un graphe complet, respectivement).

Implémenter une version de cette approche. Le graphe généré est sauvegardé dans un fichier texte respectant le format utilisé dans l'exercice 1. Vous pouvez alors tester les algorithmes des exercice précédents sur des graphes de différentes tailles et densités, afin de vérifier le bon fonctionnement des méthodes et l'impact des caractéristiques du graphe sur leur complexité.