TP n°6 - Graphes : différentes représentations

Notions abordées

- Représentation de graphe par une matrice d'adjacence
- Représentation de graphe par liste de voisins (ou de prédécesseurs/successeurs)
- Notion (et calcul) de graphe induit
- Test du caractère biparti d'un graphe
- Tri topologique , détection de cycle

Toutes les fonctions doivent être commentées et testées. Elles doivent notamment être munies d'une description faisant suite à d'éventuelles hypothèses sur leurs arguments placée entres triple guillemets sous la signature de la fonction, (`""" documentation sur plusieurs lignes"""`). Les autres commentaires s'écrivent sur des lignes commençant par dièse : `# ligne de commentaire`.

Graphes non orientés

On rappelle qu'un **graphe non orienté** G est la donnée d'un couple (V, E) où V est un ensemble fini et $E \subseteq \{X \subseteq V \mid \text{Card}(X) = 2\}$. Les éléments de V sont appelés les **sommets** de G (V comme *vertex/vertices* en anglais) et ceux de E les **arêtes** de G (E comme *edge* en anglais). On parle parfois de **graphe simple**, car ces graphes ne contiennent pas de boucles (arête dont les deux extrémités sont en fait le même sommet), ni de multi-arêtes (sans quoi E serait un multi-ensemble).

Exercice 1 Représentation par matrice d'adjacence

Une matrice de booléens symétrique de dimensions $n \times n$ M représente le graphe non orienté $G = (V, E)$ où $V = [1..n]$ et $E = \{\{i, j\} \mid M_{i,j} = \text{vrai}\}$. Dans tout cet exercice on dit "un graphe" pour signifier "un graphe non orienté représenté par une matrice d'adjacence".

Question 1

Dessiner ci-contre le graphe représenté par la matrice d'adjacence ci-dessous.

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Question 2 ✍
Quelles conditions doit vérifier une matrice de booléens pour représenter un graphe non orienté.

Question 3 💻 + ✍
Définir une fonction `est_valide_mat` qui teste si une matrice représente bien un graphe non orienté.

Question 4 💻 + ✍
Définir une fonction `nb_sommets` qui donne le nombre de sommets d'un graphe.
Quelle est sa complexité ?

Question 5 💻 + ✍
Définir une fonction `sont_voisins` qui teste si deux sommets sont voisins dans un graphe.
Quelle est sa complexité ?

Question 6 💻 + ✍
Définir une fonction `nb_arretes` qui donne le nombre d'arrêtes d'un graphe.
Quelle est sa complexité ?

Question 7 💻 + ✍
Définir une fonction `nb_voisins` qui donne le degré d'un sommet dans un graphe.
Quelle est sa complexité ?

Question 8 ✍
Donner une relation entre le nombre d'arêtes et les degrés des sommets dans un graphe non orienté.

Question 9 💻 ∗
Définir une fonction `graphe_induit` qui étant donné un graphe $G = (V, E)$ et un ensemble $W \subseteq V$ calcule le **graphe induit** par G sur W défini comme $(W, \{\{u, v\} \in E \mid u \in W \text{ et } v \in W\})$.

Question 10 💻 ∗
On dit d'un graphe $G = (V, E)$ qu'il est **biparti** s'il existe $\{V_1, V_2\}$ une partition de V telle que les arêtes de G ont toutes une extrémité dans V_1 et l'autre dans V_2. Définir une fonction `est_biparti` qui teste si un graphe est biparti.

Exercice 2 Représentation par listes d'adjacences

Dans cet exercice on s'intéresse à la représentation de graphes non orientés par listes d'adjacence c'est-à-dire par un tableau de listes de voisins (listes sans doublons représentant des ensembles). En effet, un tableau T de taille n de listes d'entiers de $[1..n]$ représente le graphe $G = (V, E)$ où $V = [1..n]$ et $E = \{\{i, j\} \mid i \text{ est présent dans } T_j\}$.[1] Dans tout cet exercice on dit "un graphe" pour signifier "un graphe non orienté représenté par une table de listes d'adjacence". Afin de ne pas créer de conflit avec les fonctions de l'exercice précédent, coder les fonctions de cet exercice dans un nouveau fichier.

Question 1 ✍
Donner la table de listes d'adjacence du graphe non orienté de la question 1 de l'exercice 1.

[1] cette définition est correcte sous réserve des conditions sur T que l'on demande d'expliciter à la question 1.

Question 2 ✍
Quelle condition doit vérifier un tableau de listes d'entiers pour représenter un graphe non orienté.

Question 3 💻
Définir une fonction `est_valide_tab` qui teste si une table de listes d'adjacence représente bien un graphe non orienté.

Question 4 💻 + ✍
Définir une fonction `nb_sommets` qui donne le nombre de sommets d'un graphe.
Quelle est sa complexité ?

Question 5 💻 + ✍
Définir une fonction `sont_voisins` qui teste si deux sommets sont voisins dans un graphe.
Quelle est sa complexité ? Peut-on réduire cette complexité en remarquant que le fait d'être voisin définit une relation symétrique ?

Question 6 💻 + ✍
Définir une fonction `nb_arretes` qui donne le nombre d'arêtes d'un graphe.
Quelle est sa complexité ?

Question 7 💻 + ✍
Définir une fonction `nb_voisins` qui donne le degré d'un sommet dans un graphe.
Quelle est sa complexité ?

Question 8 💻 ∗
Définir une fonction `graphe_induit` qui étant donné un graphe $G = (V, E)$ et un ensemble $W \subseteq E$ calcule le graphe induit par G sur W.

Question 9 💻 ∗
Définir une fonction `est_biparti` qui teste si un graphe est biparti.

Graphes orientés

On rappelle qu'un **graphe orienté** est un couple $G = (S, A)$ où S est un ensemble fini et $A \subseteq V \times V$. Les éléments de S sont alors appelés les **sommets** et ceux de A les **arcs** du graphe G.

Exercice 3 Représentation par matrice d'adjacence

Une matrice de booléens symétrique de dimensions $n \times n$ M représente le graphe $G = (S, A)$ où $S = [1..n]$ et $A = \{(i, j) \mid M_{i,j} = \text{vrai}\}$. Dans tout cet exercice on dit "un graphe" pour signifier "un graphe orienté représenté par une matrice d'adjacence".

Question 1 💻
Identifier quelles fonctions doivent être modifiées, voir dédoublées et renommées, pour passer des graphes non orientés aux graphe orienté. Recoder alors ces fonctions et copier les autres à partir des réponses à l'exercice 1, le tout dans un nouveau fichier.

Question 2 ✍
Donner une relation entre le nombre d'arêtes et les degrés des sommets dans un graphe non orienté.

Exercice 4 Représentation par listes d'adjacences

Un graphe orienté peut être représenté par un tableau de listes de successeurs de chacun des sommets, ou bien par un tableau de listes de prédécesseurs, ou bien les deux si cela permet de réduire la complexité.

Question 1 ✍
On ne demande pas ici de recoder toutes les fonctions de l'exercice 2, mais il faut savoir les adapter à un graphe représenté par liste de successeurs ou par liste de prédécesseurs, et surtout il faut savoir quelle représentation est la plus efficace. Résumer dans un tableau la complexité des opérations suivantes pour les deux représentations : parcourir les successeurs, parcourir les prédécesseurs, calculer le degré entrant, calculer le degré sortant, tester si i est successeur de j, tester si i est prédécesseur de j.

Question 2 🖳 + ✍
Définir une fonction qui transforme un tableau de listes de prédécesseurs en tableau de listes de successeurs. Définir la transformation réciproque.
Quelle est la complexité de ces transformations.

Exercice 5 Tri topologique

On dit qu'une liste L d'éléments $L_1, L_2, \dots L_n$ est un **tri topologique**. (des sommets) d'un graphe $G = (S, A)$ ssi $\{L_i \,|\, i \in [1..n]\} = S$ et $\forall (i,j) \in [1..n]^2, i < j \Rightarrow (L_j, L_i) \notin A$. On dit on parfois que L est une permutation des sommets dans laquelle aucun sommet n'est placé après l'un de ses prédécesseurs.

Question 1 ✍
Que peut-on dire du premier sommet d'un tri topologique. Combien a-t-il de prédécesseurs ? de successeurs ? Et le dernier ?

Question 2 ✍
Existe-t-il toujours un tri topologique ? Donner une condition nécéssaire et suffisante à l'existence d'un tri topologique dans un graphe.

Question 3 ✍ ∗
Proposer un algorithme en pseudo-code qui permet de calculer un tri topologique s'il en existe un, et qui donne une preuve (au sens témoin, pas au sens démonstration) qu'il n'en existe pas sinon.

Question 4 ✍ ∗
En utilisant la représentation qui permet la complexité la plus intéressante, implémenter l'algorithme proposé. *On attend une complexité pire cas en* $O(n^2)$.