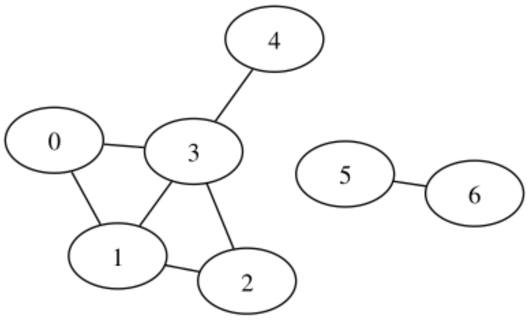
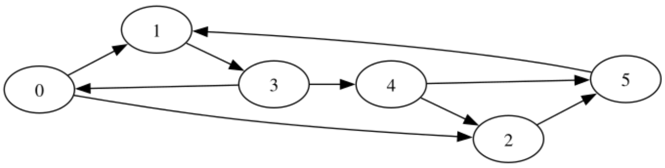
Graphes - Représentations

PCSI/PTSI

I Introduction



G1



G2

Exercice 1. *Liste d'adjacence*

Définir les variables globales `L1` et `L2` représentants les listes d'adjacences des graphes `G1` et `G2`.

Exercice 2. *Matrices d'adjacences*

Définir les variables globales `M1` et `M2` représentants les matrices d'adjacences des graphes `G1` et `G2`.

Exercice 3. *Connexité*

Est-ce que `G1` est connexe ? Justifiez.

Exercice 4. *Chemins*

Combien y a-t-il de chemins différents du sommet 0 au sommet 4, ne passant pas deux fois par le même sommet, dans `G1` ? Dans `G2` ?

Exercice 5. *Degrés*

1. Donnez dans `G1` le degré de chaque sommet.
2. Donnez dans `G2` le degré entrant et sortant de chaque sommet.

II Fonctions utilitaires

Tous les graphes sont considérés comme orientés.

Exercice 6. *Cardinalités*

Écrire deux versions des fonctions `nb_s` et `nb_a` qui prennent en entrée un graphe `G`. Et renvoient respectivement le nombre de sommet et le nombre d'arcs de `G`.

Dans la première version `G` sera représenté par une matrice d'adjacence, dans la seconde par une liste d'adjacence. Donnez la complexité dans les deux cas.

Exercice 7. *Succésseurs*

Écrire deux versions de la fonction `est_successeur` qui prend en entrée un graphe `G` et deux sommets `x` et `y` et qui renvoie un booléen vrai si et seulement si `y` est un successeur de `x`.

Dans la première version `G` sera représenté par une matrice d'adjacence, dans la seconde par une liste d'adjacence. Donnez la complexité dans les deux cas.

Exercice 8. *Degrés orienté*

Écrire deux versions de la fonction `degre` qui prend en entrée un graphe `G` et un sommets `x` qui renvoie le degré de `x`.

Dans la première version `G` sera représenté par une matrice d'adjacence, dans la seconde par une liste d'adjacence. Donnez la complexité dans les deux cas.

Exercice 9. *Degrés non-orienté*

On suppose dans cette question que le graphe est non-orienté.

Écrire deux versions de la fonction `degre_no` qui prend en entrée un graphe `G` et un sommets `x` qui renvoie le degré de `x`.

Dans la première version `G` sera représenté par une matrice d'adjacence, dans la seconde par une liste d'adjacence. Donnez la complexité dans les deux cas.

Exercice 10. *Liste d'adjacence vers matrice d'adjacence*

Écrire une fonction `la2mat` qui prend en entrée la liste d'adjacence d'un graphe et qui renvoie la matrice d'adjacence du même graphe. Précisez sa signature. Calculez sa complexité.

Exercice 11. *Matrice d'adjacence vers liste d'adjacence*

Écrire une fonction `mat2la` qui prend en entrée la matrice d'adjacence d'un graphe et qui renvoie la liste d'adjacence du même graphe. Précisez sa signature. Calculez sa complexité.

III Chemins

Tous les graphes sont considérés comme orientés. On représente un chemin par une liste de sommets.

Exercice 12. *Vérification*

Écrire deux versions de la fonction `est_chemin` qui prend en entrée un graphe `G` et une liste de sommets `c` et qui vérifie si `c` est un chemin dans `G`.

Dans la première version `G` sera représenté par une matrice d'adjacence, dans la seconde par une liste d'adjacence. Donnez la complexité dans les deux cas.

Exercice 13. *Recherche de chemin*

Soit $G = (S, V)$ un graphe orienté **acyclique** (ne possédant pas de cycle). Pour tous sommets $a, b \in S$ on défini

$$C(a, b) = \begin{cases} \texttt{True} & \text{si il existe un chemin entre a et b} \\ \texttt{False} & \text{sinon} \end{cases}$$

On cherche un algorithme qui calcule C.

Soit $u, v \in S$.

1. Calculez $C(u, u)$. On suppose dans la suite $v \neq u$.
2. Si u n'as pas de successeurs que dire de $C(u, v)$?
3. Soit w un successeur de u. Si $C(w, v) = \texttt{True}$, que dire de $C(u, v)$?
4. Exprimez $C(u, v)$ en fonction des $C(w, v)$ avec w successeur de u.
5. Implémentez une fonction récursive prenant en entrée un graphe G et deux sommets u et v et calculant $C(u, v)$ en se basant sur la formule de récurrence obtenue. Vous choisirez la représentation de G adéquate.
6. Montrez que si votre algorithme ne termine pas alors il existe une boucle dans G. Conclure quand à la terminaison de l'algorithme.