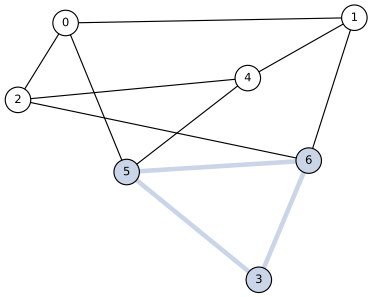
Comptage de triangles dans un graphe

Contexte



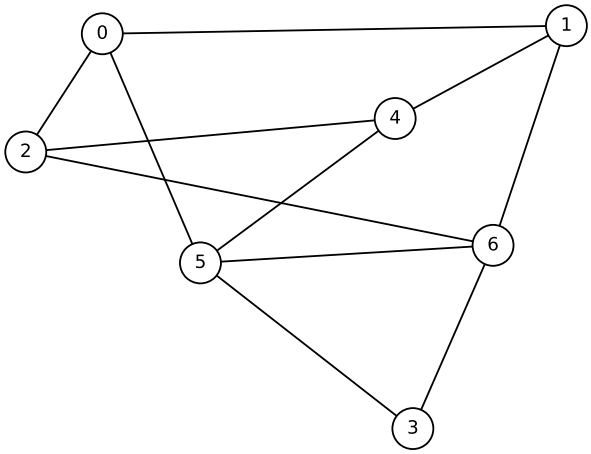
Ce problème étudie le comptage de triangles dans un graphe non orienté, c'est à dire les triplets de sommets formant un cycle de longueur 3. Plusieurs algorithmes sont proposés: des algorithmes de recherche directe et un algorithme basé sur le produit matriciel. On en profite pour étudier l'algorithme de Strassen pour le produit matriciel.

1 Comptage de triangles dans un graphe

On considère dans cet exercice des graphes **non orientés** $G = (S, A)$ où S est l'ensemble des sommets et A l'ensemble des arêtes.
Dans la suite on supposera de plus que l'ensemble des sommets $S = [\![0, n-1]\!]$ où $n \in \mathbb{N}$.
Un graphe sera représenté par un **matrice d'adjacence** M de taille $|S| \times |S|$ tel que pour tous sommets $i, j \in S$, on a $M_{ij} = 1$ si l'arête (i, j) appartient à A et $M_{ij} = 0$ sinon.

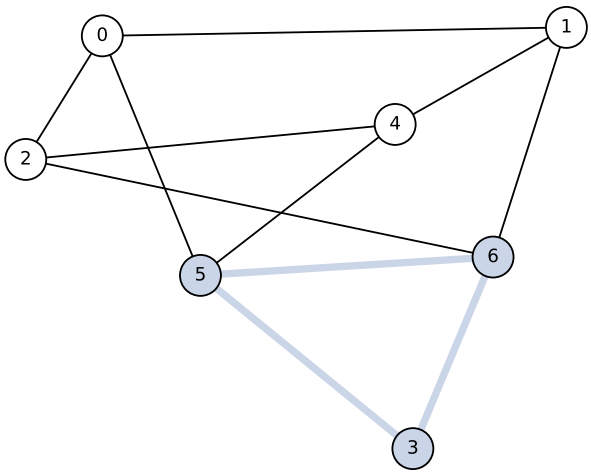
Par exemple le graphe G_0 à droite est représenté par la matrice d'adjacence suivant :

$$M_0 = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$



Étant donné un tel graphe $G = (S, A)$, on dit que les sommets distincts $i, j, k \in S$ forment un **triangle** si les arêtes (i, j), (j, k) et (k, i) appartiennent à A.

Par exemple dans le graphe G_0 précédent:
- Les sommets $3, 5, 6$ forment un triangle car les arêtes $(3, 5)$, $(5, 6)$ et $(6, 3)$ appartiennent à A.
- Les sommets $0, 1, 2$ ne forment **pas** un triangle car l'arête $(1, 2)$ n'appartient pas à A.



1.1. Pour chacun des graphes représentés par les matrice d'adjacence suivants, déterminer s'il admet des triangles, et si oui déterminer lesquels.

$$M_1 = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix}. \qquad M_2 = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Remarque: un schéma suffisamment clair pourra faire office de preuve.

Un premier algorithme de détection de triangles consiste à

- pour tous les triplets de sommets distincts $i, j, k \in S$
- vérifier si les sommets i, j, k forment un triangle en vérifiant si les arêtes (i, j), (j, k) et (k, i) appartiennent à A.

1.2. Écrire une fonction `recherche_triangle_brute_force` prenant en argument une matrice M représentant un graphe, et renvoyant le nombre de triangles dans le graphe représenté par M.

1.3. Quelle est la complexité de l'algorithme précédent en fonction de $|S|$?

Dans le cas où $|A|$ est négligeable devant $|S|^2$, on peut améliorer l'algorithme précédent de la manière suivante:

- pour toutes les arêtes $(i, j) \in A$,
- pour tous les sommets $k \in S$ distincts de i et j,
- vérifier si (i, k) et (j, k) appartiennent à A.

1.4. Écrire une fonction python `liste_aretes` prenant en argument une matrice M représentant un graphe, et renvoyant la liste de toutes les arêtes du graphe représenté par M.

1.5. Écrire une fonction python `recherche_triangle_variante` prenant en argument une matrice M représentant un graphe, et renvoyant le nombre de triangles dans le graphe représenté par M.

1.6. Quelle est la complexité de l'algorithme précédent en fonction de $|S|$ et $|A|$?
Commenter par rapport à la complexité de l'algorithme précédent.

1.7. Soit $G = (S, A)$ un graphe représenté par une matrice d'adjacence M et $i \in S$.
Déterminer une relation entre $(M^3)_{ii}$ et le nombre de triangles du graphe G contenant le sommet i.
Rappel: M^3 est le produit matriciel usuel MMM.

On suppose dans les questions **1.8.** et **1.9.** que l'on dispose d'une fonction python `produit_matrice` prenant en argument deux matrices carrées M et N de même taille n et renvoie leur produit matriciel MN.

On suppose de plus qu'il existe $\alpha \geq 2$ tel que `produit_matrice` a une complexité en $O(n^\alpha)$.

1.8. En déduire une fonction python `recherche_triangle_matrice` prenant en argument une matrice M représentant un graphe, et renvoyant le nombre de triangles dans le graphe.

1.9. Quelle est la complexité de l'algorithme précédent en fonction de $|S|$ et α ?

On cherche désormais à obtenir un produit matriciel efficace.

1.10. Écrire une fonction python `produit_matrice_naif` prenant en argument deux matrices carrées M et N de même taille n et renvoyant leur produit matriciel MN en utilisant la formule

$$\forall (i, j) \in [\![0, n-1]\!], (MN)_{ij} = \sum_{k=0}^{n-1} M_{ik} N_{kj}.$$

1.11. Commenter la complexité obtenue à la question **1.9.** si on choisit d'utiliser la fonction `produit_matrice_naif` pour effectuer le produit matricel.

Un algorithme plus efficace (algorithme de Strassen) est basé sur la remarque de produit par blocs suivant:
Soient $A_{11}, A_{12}, A_{21}, A_{22} \in \mathrm{M}_n(\mathbb{Z})$, $B_{11}, B_{12}, B_{21}, B_{22} \in \mathrm{M}_n(\mathbb{Z})$.
On pose

$$\begin{cases} M_1 = (A_{11} + A_{22})(B_{11} + B_{22}) \\ M_2 = (A_{21} + A_{22})B_{11} \\ M_3 = A_{11}(B_{12} - B_{22}) \\ M_4 = A_{22}(B_{21} - B_{11}) \\ M_5 = (A_{11} + A_{12})B_{22} \\ M_6 = (A_{21} - A_{11})(B_{11} + B_{12}) \\ M_7 = (A_{12} - A_{22})(B_{21} + B_{22}). \end{cases}$$

On remarque alors (il n'est pas demandé de vérifier le produit par blocs) que

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{pmatrix}. \quad (1)$$

On en déduit l'algorithme récursif suivant pour les produits matriciels de matrices carrées A et B de taille 2^k:

- Si $k = 0$ alors on renvoie la matrice $(a_{11}b_{11})$.
- On extrait les blocs $A_{11}, A_{12}, A_{21}, A_{22}$ de la matrice A et les blocs $B_{11}, B_{12}, B_{21}, B_{22}$ de la matrice B.
- On calcule récursivement les produits matriciels et les sommes pour déterminer matrices $2^{k-1} \times 2^{k-1}$, $M_1, \ldots, M_7$ de l'équation (1).
- On en déduit la matrice AB.

On suppose que l'addition de deux matrices de tailles $2^k \times 2^k$ a une complexité en $O((2^k)^2)$ et que de même l'extraction d'un bloc de taille $2^{k-1} \times 2^{k-1}$ a une complexité en $O((2^{k-1})^2)$.

On note $C(k)$ la complexité de l'algorithme précédent pour des matrices de taille $2^k \times 2^k$.

1.12. Montrer qu'il existe $\gamma > 0$ tel que pour tout $k \in \mathbb{N}^*$, on a

$$C(k) \leq 7C(k-1) + \gamma 4^k.$$

1.13. En déduire une constante $\alpha \in]2,3[$ telle que

$$C(k) = O_{k\to+\infty}\left((2^k)^\alpha\right).$$

1.14. Écrire une fonction python `produit_matrice_strassen` prenant en argument deux matrices carrées M et N de même taille 2^k, et renvoyant leur produit matriciel MN en utilisant l'algorithme de Strassen.

On suppose que l'on peut étendre l'algorithme de Strassen à des matrices carrées de taille $n \times n$ pour tout entier $n \geq 1$, et que la complexité de l'algorithme est en $O(n^\alpha)$ pour la constante α précédente.

1.15. Commenter la complexité obtenue en utilisant la fonction `produit_matrice_strassen` pour effectuer le produit matriciel dans la fonction `recherche_triangle_matrice`.