TP n° 3 - Plus de graphes

Il vaudrait mieux avoir fini au moins les deux premiers exercices du TP précédent avant de s'attaquer à celui-ci...

Exercice 1. *Graphes pondérés*

On modifie les représentations des graphes vues au TP précédent pour permettre d'ajouter un poids à chaque arête (comme ça a été vu en cours).
Dans le cas des matrices d'adjacences, on remplace les 0 et les 1 par le poids de l'arête, en mettant une valeur *infinie* si les deux sommets ne sont pas reliés (en python, on peut utiliser la valeur `float("infinity")` qui pourra être comparée à des entiers ou des flottants).
Dans le cas des listes d'adjacence, il faut maintenant lister des couples. Ainsi la liste correspondant au sommet `i` contiendra les couples `(j, p)` tels qu'il existe une arête de `i` à `j` de poids `p`.

1. Réécrivez la fonction `voisins(g, i)` qui renvoie la liste des voisins de `i` dans `g`.

2. Ré-écrivez les fonctions `liste_en_matrice(g)` et `matrice_en_liste(g)` qui permettent de passer d'une représentation par matrice à une représentation par listes, et inversement.

Exercice 2. *Algorithme de plus court chemin*

1. Écrivez l'algorithme vu en cours permettant de trouver pour tous couples `(i, j)` de sommets dans un graphe `g` le plus court chemin entre `i` et `j`.
Rappel : Pour chaque valeur de k allant de 0 à n (le nombre de sommets du graphe), on construit le tableau C_k à deux dimensions dont la case (i, j) contient la longueur du plus court chemin du sommet `i` au sommet `j` ne passant que par les sommets de `1` à `k`. On a la relation

$$C_k(i,j) = \min(C_{k-1}(i,j), C_{k-1}(i,k) + C_{k-1}(k,j)$$