Graphes - Exercices

A. Un peu de théorie

Exercice 1. Handshake lemma.

Soit $G = (S, A)$ un graphe non orienté. Démontrer qu'il y a un **nombre pair** de sommets de degré impair.

B. Matrices d'adjacence

Dans cette section on considère la représentation des graphes par **matrice d'adjacence**.

Exercice 2. Création de matrices d'adjacence

*On considère ici des graphes **non orientés**. Vous pourrez réfléchir en complément à ce qu'il faudrait modifier pour gérer des graphes orientés.*

On donne la fonction suivante permettant de créer la matrice d'adjacence d'un graphe à n sommets sans arêtes.

```
def matrice_nulle(n: int) -> list:
    M = [ [0] * n for k in range(n) ]
    return M
```

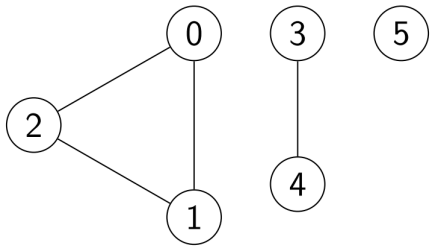
a. Coder et tester la fonction précédente.

b. Écrire une fonction `ajouter_arete(M: list, u: int, v: int) -> None` qui modifie la matrice M pour tenir compte de la *création* de l'arête reliant u et v.

c. Écrire une fonction `effacer_arete(M: list, u: int, v: int) -> None` qui modifie la matrice M pour tenir compte de la *suppression* de l'arête reliant u et v.

d. Ecrire la succession d'instructions permettant de créer la matrice d'adjacence du graphe ci-contre.

Afficher la matrice obtenue.



Exercice 3. Degré et voisins d'un sommet

a. Écrire une fonction `degre(M: list, u: int) -> int` qui retourne le degré du sommet u.

b. Écrire une fonction `voisins(M: list, u: int) -> list` qui retourne la liste des voisins du sommet u.

C. Liste d'adjacence

Dans cette section on utilise la représentation par **liste d'adjacence.**

Exercice 4. Création de listes d'adjacence

*On considère ici des graphes **non orientés**. Vous pourrez réfléchir en exercice à ce qu'il faudrait modifier pour gérer des graphes orientés.*

On donne la fonction suivante permettant de créer la liste d'adjacence d'un graphe à n sommets sans arêtes.

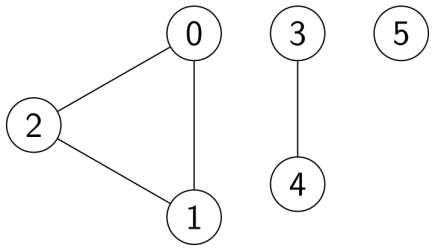
```
def liste_vide(n: int) -> list:
    L = [ [] for k in range(n) ]
    return L
```

a. Coder et tester la fonction précédente.

b. Écrire une fonction `ajouter_arete_liste(L: list, u: int, v: int)-> None` qui modifie la liste L pour tenir compte de la *création* de l'arête reliant u et v.

c. Ecrire la succession d'instructions permettant de créer la liste d'adjacence du graphe ci-contre.

Afficher la liste obtenue.



Exercice 5. Écrire une fonction calculant le nombre d'arcs d'un graphe orienté représenté par liste d'adjacence.

D. Jonglons avec les représentations

Exercice 6. Conversion de représentation

Ecrire deux fonctions `matrice_2_list(M)` et `list_2_matrice(L)` pour convertir une matrice d'adjacence en liste d'adjacence et vice-versa.

Exercice 7. Autour des graphes orientés

Pour cet exercice vous aurez besoin de reprendre les fonctions de création précédentes dans le cadre de graphes orientés.

Si G est un graphe orienté, son graphe *transposé* s'obtient en inversant le sens de tous ses arcs.

a. Si G est représenté par sa matrice d'adjacence M, quelle est la matrice de son graphe transposé ?

Écrire une fonction `transp_mat(M)` qui retourne la matrice du graphe transposé de matrice M.

b. Traiter maintenant le cas d'une représentation par liste d'adjacence.

Indication : on pourra judicieusement effectuer des conversions entre les deux représentations.