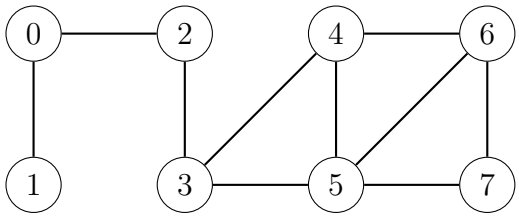
Série 11

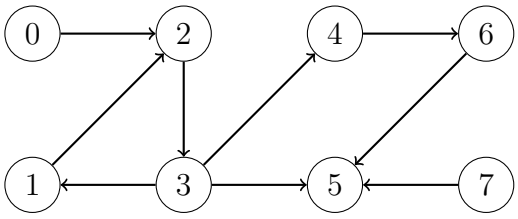
Sauf si spécifié autrement, tous les algorithmes demandés sont à écrire sous forme d'une fonction Python.

Exercice 1

(a) Pour chacun des graphes ci-dessous, donnez sa représentation par matrice d'adjacence et par listes d'adjacence.



G_1



G_2

(b) Le graphe non dirigé G_3 est représenté par la matrice d'adjacence

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

Dessinez G_3 et donnez sa représentation par listes d'adjacence.

(c) Le dictionnaire suivant représente le graphe dirigé G_4 par listes d'adjacence.
`D = {0:[1, 3], 1:[4], 2:[4,5], 3:[1], 4:[3], 5:[]}`.
Dessinez G_4 et donnez sa représentation par matrice d'adjacence.

Exercice 2

(a) Le Jupyter notebook `BFS.ipynb` contient l'algorithme de parcours en largeur vu en cours, avec une instruction `print` qui affiche, en plus du sommet parcouru, la liste des sommets stockés à chaque étape dans `a_parcourir`. Il contient également les représentations par listes d'adjacence de deux graphes G et H que vous pouvez utiliser pour mieux comprendre le fonctionnement de l'agorithme `BFS`.
Exécutez `BFS(G,s)` et `BFS(H,s)` avec divers choix du sommet de départ `s`. Changez l'ordre où les sommets voisins sont stockés dans les listes

d'adjacence données et observez s'il y a une différence dans l'ordre des sommets affichés.

(b) Soient G_1 et G_2 les graphes donnés à l'exercice 1a, et soient `G_1` et `G_2` les représentations par listes d'adjacence que vous avez données de ces graphes.
Faites à la main un parcours en largeur (BFS) du graphe G_1 à partir du sommet 2, c'est-à-dire écrivez les sommets dans l'ordre où vous les visitez.
Vérifiez votre réponse en exécutant `BFS(G_1,2)` dans le Jupyter notebook.

(c) Même question pour un parcours en largeur de G_2 à partir de 0.

Exercice 3

(a) Soit $G = (V, E)$ un graphe **non dirigé** à n sommets et m arêtes.
Pour $v \in V = \{0, 1, \ldots, n-1\}$, on définit le **degré** de v comme étant le nombre d'arêtes incidentes à v, c'est-à-dire le nombre de voisins de v.
Par exemple, pour le graphe G_1 donné à l'exercice 1a, la liste suivante contient les degrés des sommets (`L[u]` contient le degré du sommet `u`) :

```
L = [2, 1, 2, 3, 3, 4, 3, 2].
```

(i) On suppose que G est représenté par une matrice d'adjacence. Donnez un algorithme qui prend en entrée la matrice d'adjacence de G et retourne une liste `L` de taille n, telle que `L[v]` contient le degré du sommet v. Testez votre algorithme avec le graphe G_1 donné à l'exercice 1a.
Quel est le temps de parcours de votre algorithme en fonction de n et m ?

(ii) On suppose maintenant que G est représenté par des listes d'adjacence. Donnez un algorithme qui prend en entrée un dictionnaire de listes d'adjacence de G et retourne la liste `L` décrite au point (i). Testez votre algorithme avec le graphe G_1 donné à l'exercice 1a.
Quel est le temps de parcours de votre algorithme en fonction de n et m ?

(b) On considère maintenant un graphe **dirigé** $G = (V, E)$ à n sommets et m arêtes.
Pour $v \in V = \{0, 1, \ldots, n-1\}$, on définit
— le **degré sortant** de v comme le nombre d'arêtes sortant de v, c'est-à-dire le nombre d'arêtes (v, u) pour u un quelconque autre

sommet de G ;

— le **degré entrant** de v comme le nombre d'arêtes entrant dans v, c'est-à-dire le nombre d'arêtes (u, v) pour u un quelconque autre sommet de G.

Par exemple, pour le graphe G_2 donné à l'exercice 1a, la liste suivante contient les degrés sortants de tous les sommets :

```
L_out = [1, 1, 1, 3, 1, 0, 1, 1]
```

Et la liste suivante contient les degrés entrants de tous les sommets :

```
L_in = [0, 1, 2, 1, 1, 3, 1, 0].
```

(i) On suppose que G est représenté par une matrice d'adjacence. Donnez un algorithme qui prend en entrée la matrice d'adjacence de G et retourne deux listes `L_out` et `L_in` de taille n, telles que `L_out[v]` contient le degré sortant et `L_in[v]` le degré entrant du sommet v. Testez votre algorithme avec le graphe G_2 donné à l'exercice 1a.

Quel est l'ordre du temps de parcours de votre algorithme en fonction de n et m ?

(ii) On suppose maintenant que G est représenté par des listes d'adjacence. Donnez un algorithme qui prend en entrée un dictionnaire de listes d'adjacence de G et retourne les deux listes décrites au point (i). Testez votre algorithme avec le graphe G_2 donné à l'exercice 1a.

Votre algorithme doit avoir temps de parcours $\Theta(n + m)$.

Remarque 1 : Les listes `L_out` et `L_in` sont calculées indépendamment l'une de l'autre.

Remarque 2 : A part pour le calcul de `L_in` au point (b-ii), un algorithme naïf fera l'affaire pour toutes les listes demandées.

Exercice 4

(a) Soit $G = (V, E)$ un graphe dirigé à n sommets et m arêtes. Prouver que

$$0 \leq m \leq n(n-1).$$

(b) Soit $G = (V, E)$ un graphe non dirigé à n sommets et m arêtes. Prouver que

$$0 \leq m \leq \frac{n(n-1)}{2}.$$

Exercice 5 (révision)

Pour chacune des boucles ci-dessous, soit $F(n)$ le nombre de fois que l'instruction `print` est exécutée.

(a) Pour $n = 5$, que vaut $F(n)$ pour chacune des boucles ? Vérifiez votre réponse en exécutant la boucle avec $n = 5$.

(b) Donner, pour chacune des boucles, l'ordre de croissance de $F(n)$ en notation $\Theta(\cdot)$.

```
#boucle 1
for i in range(n):
    for j in range(n):
        print(i, j)

#boucle 2
for i in range(n):
    for j in range(i+1, n):
        print(i, j)

#boucle 3
for i in range(n):
    for j in range(i, i+2):
        print(i, j)

#boucle 4
for i in range(n):
    for j in range(i+1, n):
        for k in range(j+1, n):
            print(i, j, k)

#boucle 5
for i in range(n):
    for j in range(i+1, n):
        for k in range(j, j+1):
            print(i, j, k)
```