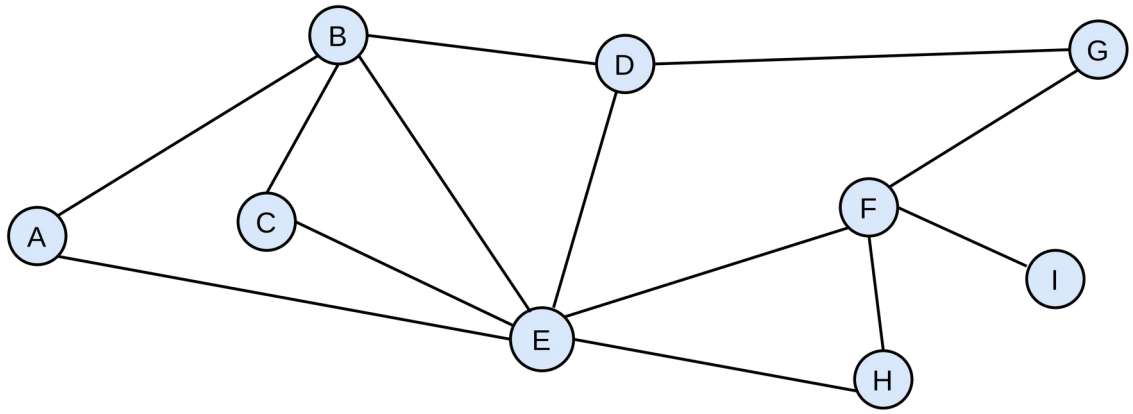
TP – CHAPITRE 11 : Graphes

Sommaire

TP1 - Premier exemple de graphe

- On considère le graphe simple non orienté G1 ci-dessous.



- Compléter ci-dessous le tableau donnant les degrés de chaque sommets de ce graphe.

Sommet	A	B	C	D	E	F	G	H	I	**Total**
Degré										

- En déduire à l'aide d'un calcul, le nombre d'arêtes du graphe G1 :

- Donner le diamètre du graphe G1 : .

- Compléter ci-dessous le tableau donnant l'excentricité de chaque sommets de ce graphe.

Sommet	A	B	C	D	E	F	G	H	I	**Minimum**
Excentricité										

- En déduire le rayon de ce graphe : et les centres de ce graphe :

- Compléter ci-dessous, la matrice d'adjacence de ce graphe.

$$A = \begin{pmatrix} \end{pmatrix}$$

Lever la main pour valider ce TP.

Nicolas Blanc, Michel Courtieu, Lycée Anna de Noailles (Evian)

TP2 - Matrice d'adjacence

- On considère la matrice d'adjacence suivante :

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

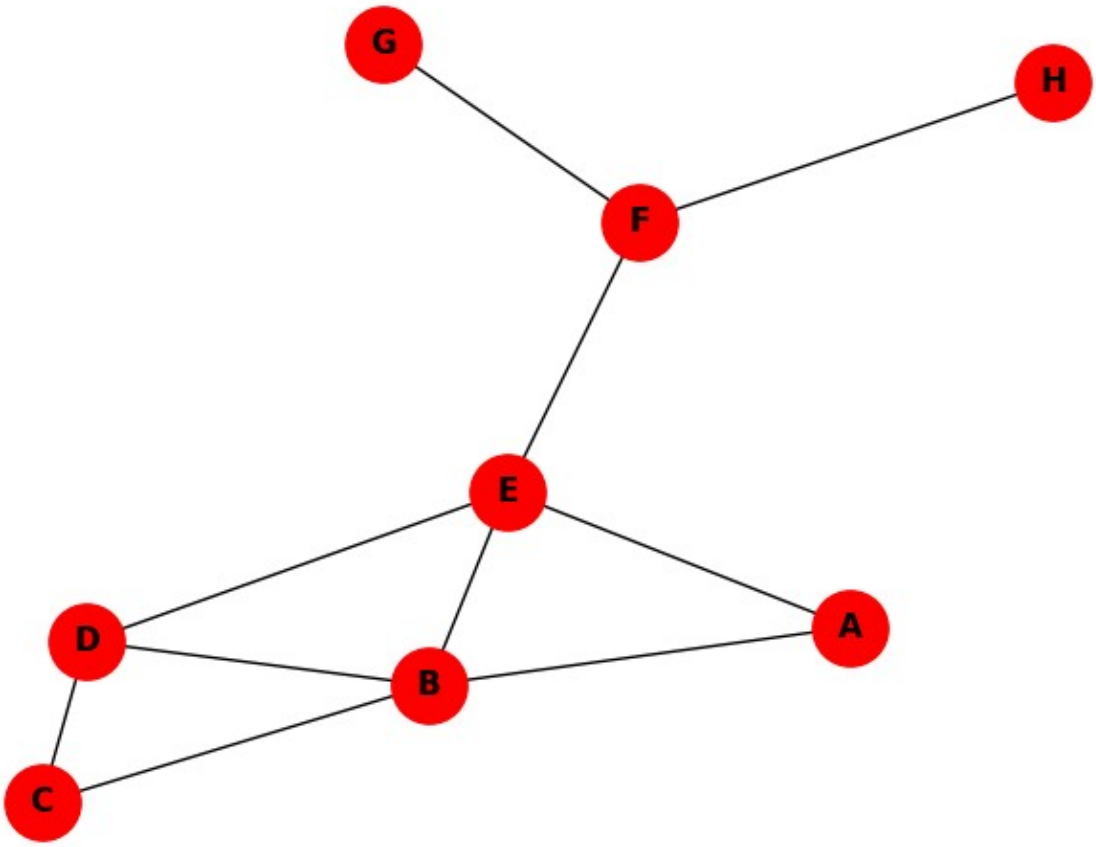
- Sur une feuille de brouillon, dessiner le graphe **simple non orienté** G1 correspondant à cette matrice d'adjacence.
- Sur la feuille de brouillon, dessiner le graphe **simple orienté** G2 correspondant à cette matrice d'adjacence.

Lever la main pour valider ce TP.

TP3 - Implémentation d'un graphe simple non orienté avec un dictionnaire

- Sur votre compte créer un dossier **TP-Chapitre11** dans lequel on va enregistrer les TP de ce chapitre. Récupérer les fichiers **graphe_dico1.py** et **dessiner_graphe.py** **en ressource** et les enregistrer dans le dossier **TP-Chapitre11**.
- Compléter le **pass** avec du code dans le fichier **graphe_dico1.py** afin d'implémenter le **graphe simple non orienté** dessiné ci-dessous avec une structure de **dictionnaire**.
- Installer la bibliothèque **networkx** pour pouvoir dessiner les graphes. Pour cela, dans **Thonny** cliquer sur l'onglet **Outils** puis **Gérer les paquets…** et rechercher **networkx** puis l'installer .
- Retours attendus après l'appel de la fonction **dessiner1( )** dans la console :

```
>>> dessiner1(G)
```



Lever la main pour valider ce TP.

TP4 - Implémentation d'un graphe simple non orienté avec un dictionnaire

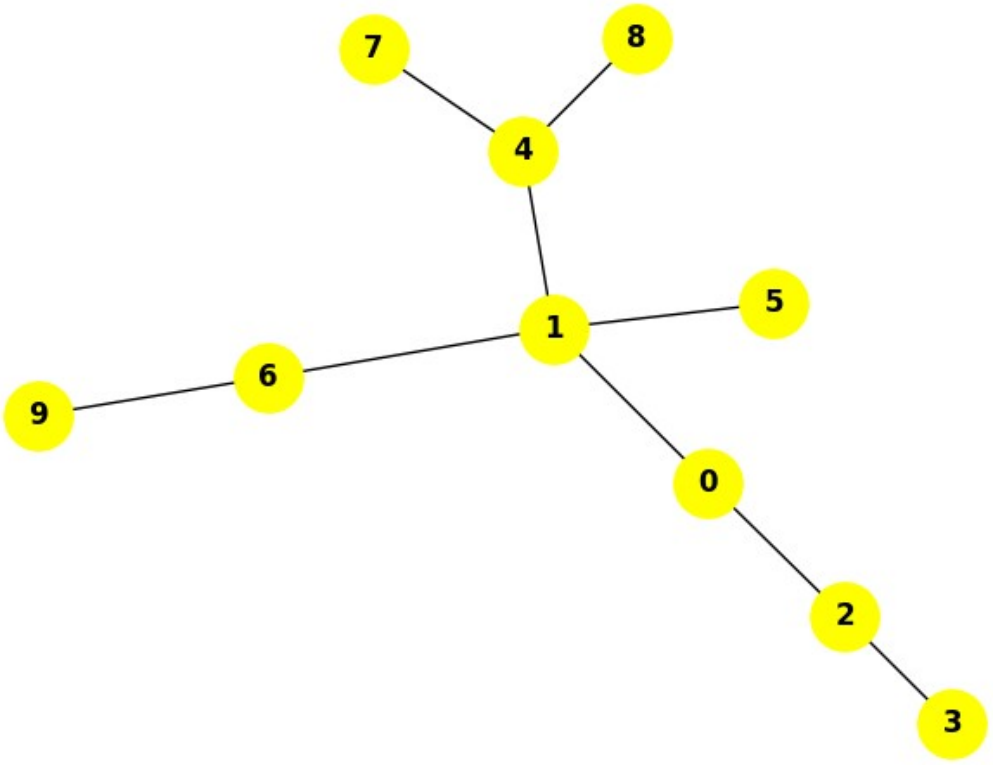
- Sur votre compte dupliquer le fichier **graphe_dico1.py** en **graphe_dico2.py** .
- Modifier le code pour implémenter et dessiner le graphe simple non orienté correspondant à la matrice d'adjacence ci-dessous, toujours avec la structure de **dictionnaire**.

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

Lever la main pour valider ce TP.

TP5 - Parcours en profondeur récursif

- On considère le graphe G ci-dessous. Sur une feuille de brouillon, dessiner sous forme d'un arbre, les appels récursifs successifs de la fonction **parcours_profondeur( )** que l'on notera **pf (s)** en abrégée, si l'appel initial est **pf(G, 'A', liste)** .
- Donner la liste retournée par l'appel **pf(G, 'A', liste)** .



Lever la main pour valider ce TP.

TP6 - Parcours en profondeur récursif avec un dictionnaire

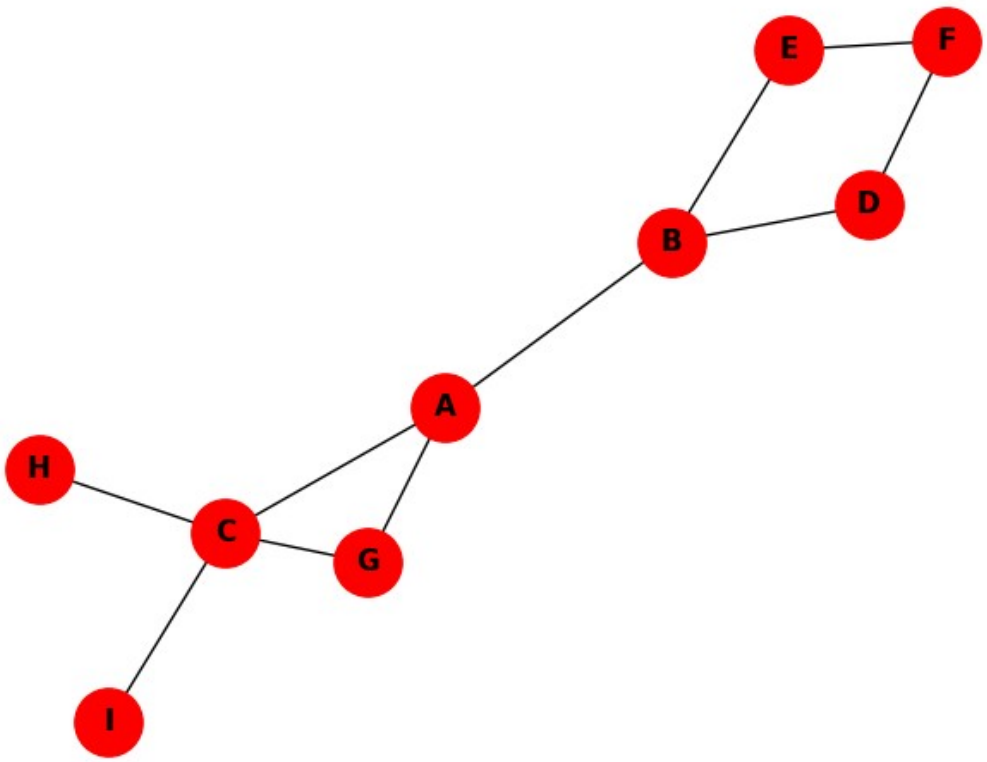
- Dans le fichier **graphe_dico2.py** rédiger une fonction **récursive** nommée : **parcours_profondeur1(G, s, liste)** qui retourne un parcours en profondeur de type liste du **graphe simple non orienté G** en partant du sommet s . (Voir pseudo-code du cours).
- Exemple de retours attendus dans la console :

```
>>> parcours_profondeur1(G, 'A', [])
['A', 'C', 'D', 'B', 'E', 'F', 'G']
>>> parcours_profondeur1(G, 'G', [])
['G', 'E', 'B', 'A', 'C', 'D', 'F']
```

Lever la main pour valider ce TP.

TP7 - Parcours en profondeur itératif

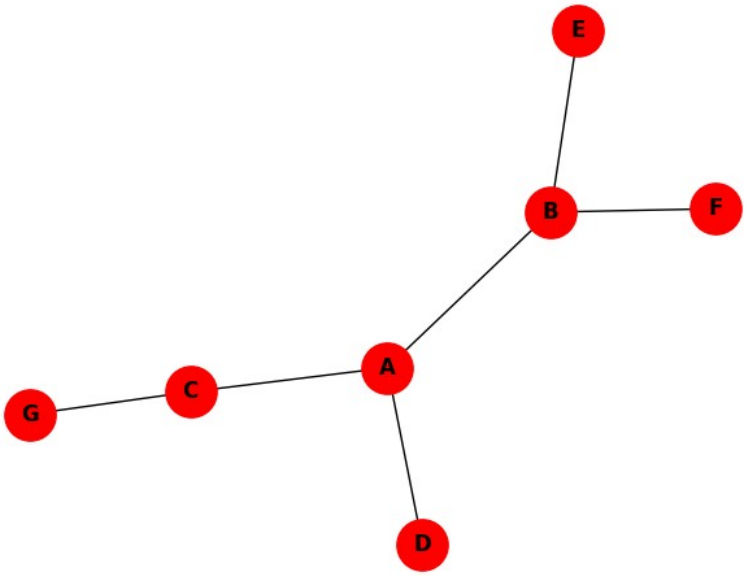
- On considère le graphe G ci-dessous. Sur une feuille de brouillon, détailler l'évolution de la pile et de la liste lors de l'appel de la fonction itérative **parcours_profondeur(G, 'A')** .



Lever la main pour valider ce TP.

TP8 - Parcours en profondeur itératif avec un dictionnaire

- Récupérer le fichier **graphe_dico3.py** et rédiger une fonction **itérative** nommée : **parcours_profondeur2(G, s)** qui retourne un parcours en profondeur de type liste du **graphe simple non orienté G** en partant du sommet s . (Voir pseudo-code du cours).
- Implémenter le graphe G ci-dessous avec la structure de **dictionnaire**.



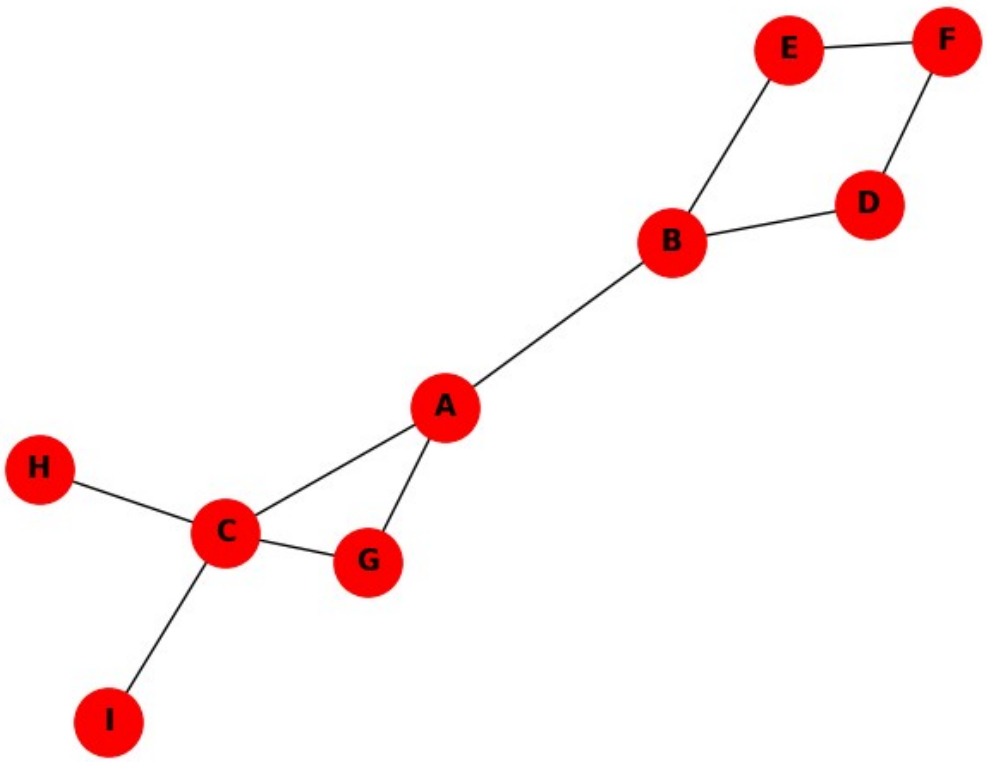
- Exemple de retours attendus dans la console :

```
>>> parcours_profondeur2(G, 'A')
['A', 'D', 'C', 'G', 'B', 'F', 'E']
>>> parcours_profondeur2(G, 'G')
['G', 'C', 'A', 'D', 'B', 'F', 'E']
```

Lever la main pour valider ce TP.

TP9 - Parcours en largeur itératif

- On considère le graphe G ci-dessous. Sur une feuille de brouillon, détailler l'évolution de la file et de la liste lors de l'appel de la fonction itérative **parcours_largeur(G, 'A')** .



Lever la main pour valider ce TP.

TP10 - Parcours en largeur avec un dictionnaire

- Récupérer le fichier **graphe_dico4.py** et rédiger une fonction **itérative** nommée : **parcours_largeur(G, s)** qui retourne un parcours en largeur de type liste du graphe simple non orienté G en partant du sommet s . (Voir pseudo-code du cours).
- Exemple de retours attendus dans la console :

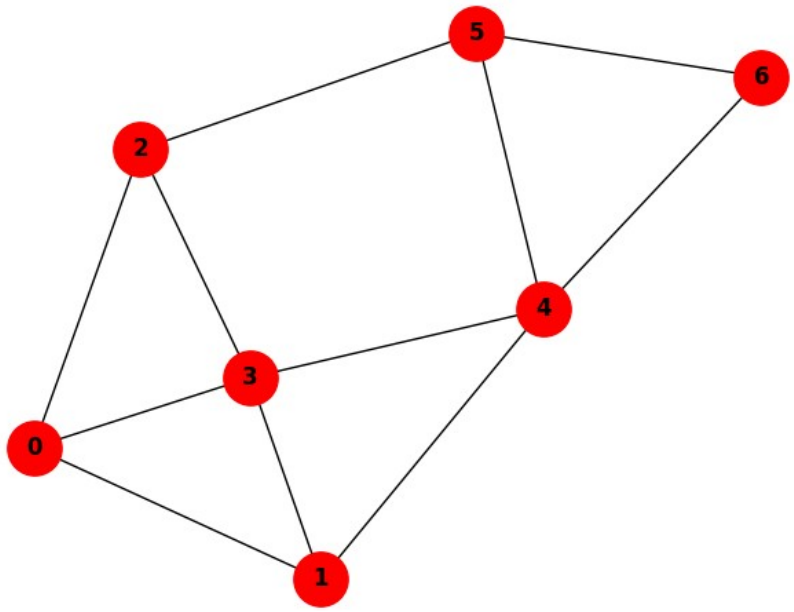
```
>>> parcours_largeur(G, 'A')
['A', 'B', 'C', 'D', 'E', 'F']
>>> parcours_largeur(G, 'F')
['F', 'C', 'D', 'E', 'A', 'B']
```

Lever la main pour valider ce TP.

TP11 - Implémentation d'un graphe avec une matrice

- Récupérer le fichier **graphe_matrice1.py** en ressource et compléter le premier **pass** avec du code dans le fichier **graphe_matrice1.py** afin d'implémenter le graphe dessiné ci-dessous avec une structure de **matrice**.
- Retours attendus après l'appel de la fonction **dessiner2( )** dans la console :

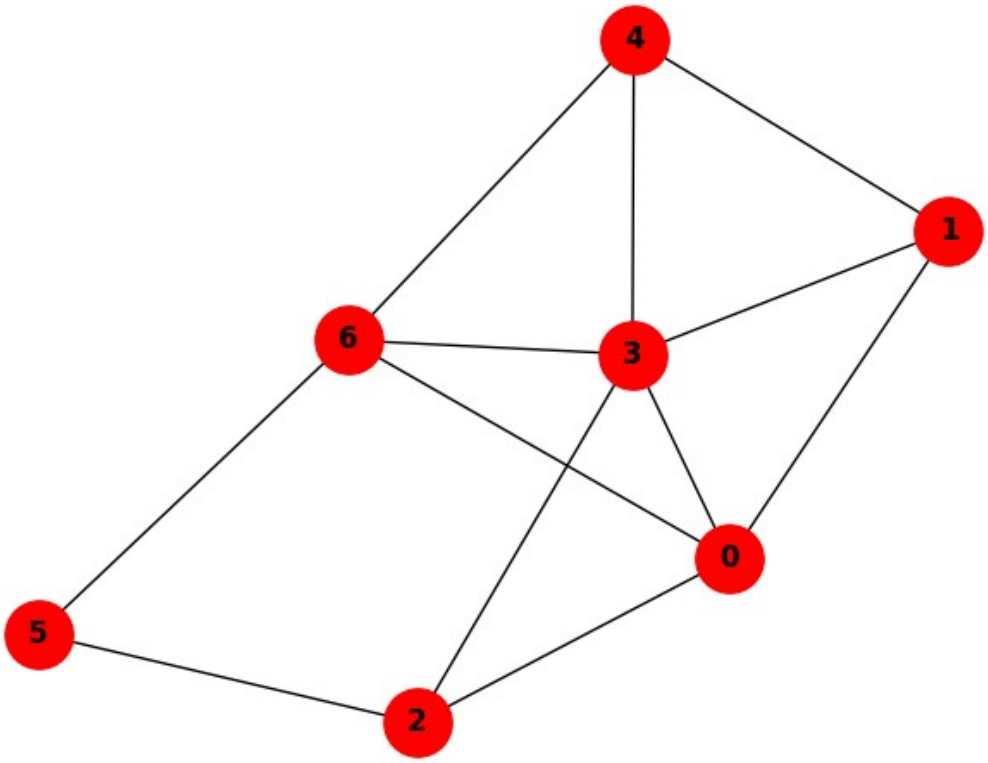
```
>>> dessiner2(G)
```



Lever la main pour valider ce TP.

TP12 - Parcours en profondeur récursif avec une matrice

- Dupliquer le fichier **graphe_matrice1.py** en **graphe_matrice2.py** puis compléter le code de la fonction **récursive** nommée : **parcours_profondeur3(G, s, liste)** qui retourne un parcours en profondeur de type liste du graphe G en partant du sommet s (ici s sera de type int) . (Voir pseudo-code du cours).
- Implémenter le graphe G ci-dessous avec la structure de **matrice**.



- Exemple de retours attendus dans la console :

```
>>> parcours_profondeur3(G, 0, [])
[0, 1, 3, 2, 5, 6, 4]
>>> parcours_profondeur3(G, 6, [])
[6, 0, 1, 3, 2, 5, 4]
```

Lever la main pour valider ce TP.

TP13 - Sommets adjacents avec une matrice

- Dans le fichier **graphe_matrice2.py** , rédiger une fonction nommée : **adjacents(G, s)** qui retourne la liste de tous les sommets du graphe G adjacents au sommet s.
- Exemple de retours attendus dans la console :

```
>>> adjacents(G, 0)
[1, 2, 3, 6]
>>> adjacents(G, 6)
[0, 3, 4, 5]
>>> adjacents(G, 5)
[2, 6]
```

Lever la main pour valider ce TP.

TP14 - Parcours en profondeur itératif avec une matrice

- Dans le fichier **graphe_matrice2.py** , rédiger une fonction **itérative** nommée : **parcours_profondeur4(G, s)** qui retourne un parcours en profondeur de type liste du graphe G en partant du sommet s (ici s sera de type int) . (Voir pseudo-code du cours).
- Exemple de retours attendus dans la console :

```
>>> parcours_profondeur4(G, 0)
[0, 6, 5, 2, 3, 4, 1]
>>> parcours_profondeur4(G, 6)
[6, 5, 2, 3, 4, 1, 0]
```

Lever la main pour valider ce TP.

TP15 - Parcours en largeur avec une matrice

- Dans le fichier **graphe_matrice2.py** , rédiger une fonction **itérative** nommée : **parcours_largeur(G, s)** qui retourne un parcours en largeur de type liste du graphe G en partant du sommet s . (Voir pseudo-code du cours).
- Exemple de retours attendus dans la console :

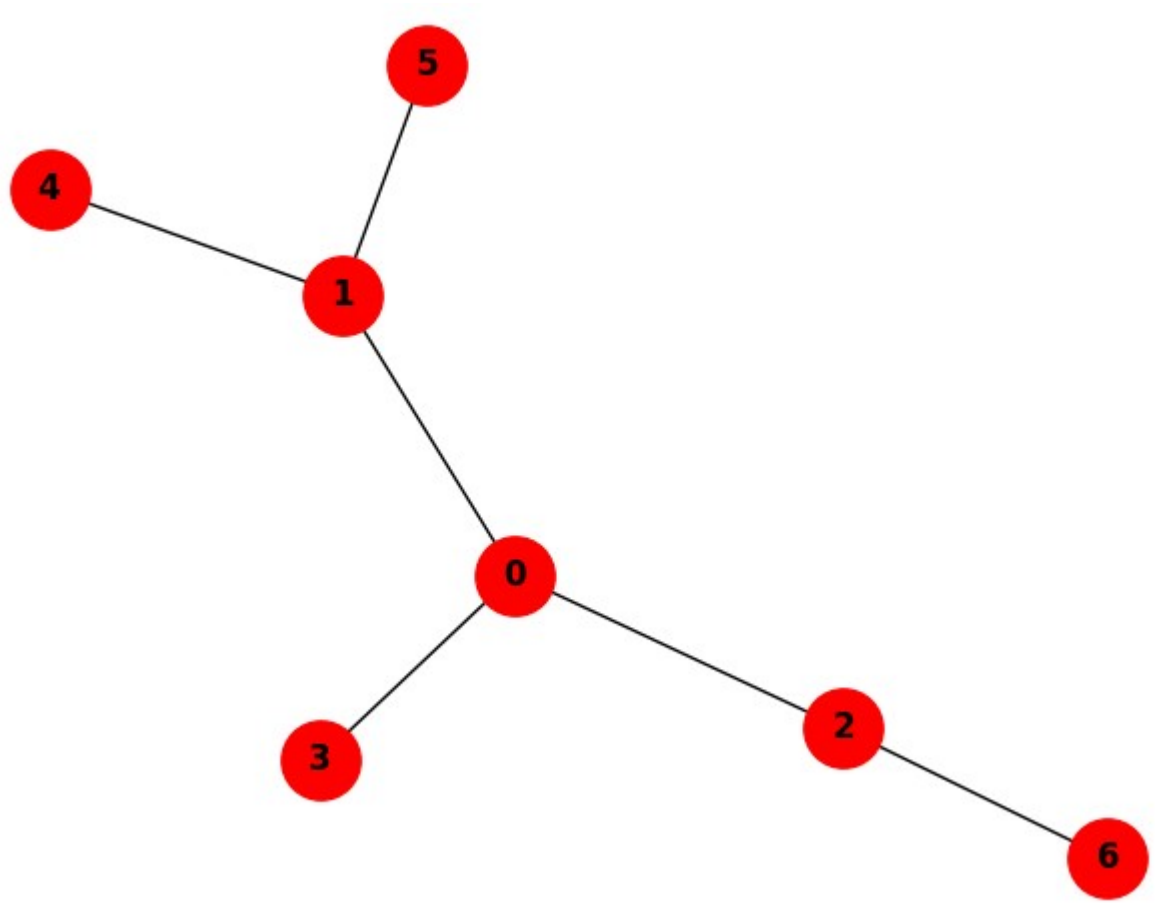
```
>>> parcours_largeur(G, 0)
[0, 1, 2, 3, 6, 4, 5]
>>> parcours_largeur(G, 6)
[6, 0, 3, 4, 5, 1, 2]
```

Lever la main pour valider ce TP.

TP16 - Passer de l'implémentation dictionnaire à l'implémentation matrice

- Récupérer le fichier **dico_matrice.py** en ressource et l'enregistrer dans le dossier **TP-Chapitre11**.
- Compléter le code de la fonction **matrice(G)** qui prend un graphe au format dictionnaire en paramètre et qui retourne un graphe au format matrice.
- Exemple de retours attendus dans la console avec dessin du graphe :

```
>>> G2 = matrice(G)
>>> dessiner2(G2)
```

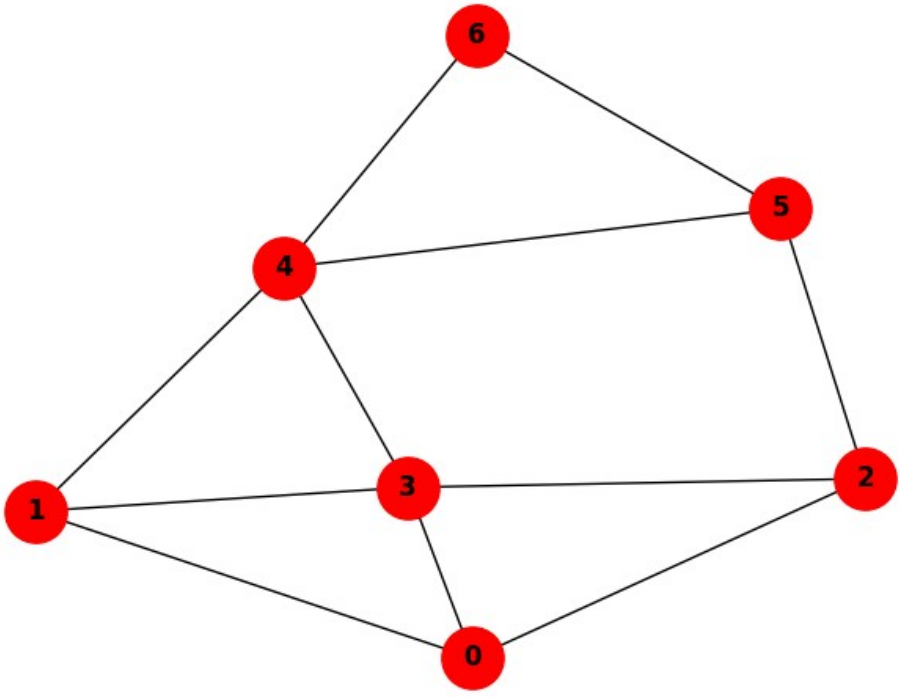


Lever la main pour valider ce TP.

TP17 - Passer de l'implémentation matrice à l'implémentation dictionnaire

- Dans le fichier **dico_matrice.py** compléter le code de la fonction **dico(G)** qui prend un graphe au format matrice en paramètre et qui retourne un graphe au format dictionnaire.
- Exemple de retours attendus dans la console avec dessin du graphe :

```
>>> G1 = dico(M)
>>> dessiner1(G1)
```

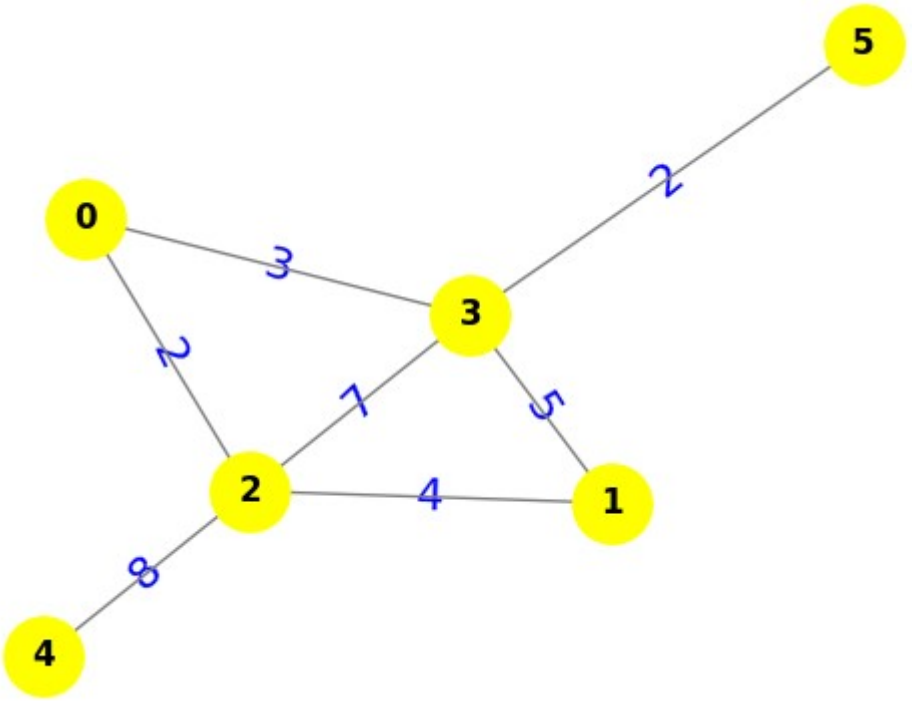


Lever la main pour valider ce TP.

TP18 - Implémentation d'un graphe pondéré

- Récupérer le fichier **graphe_pondere1.py** en ressource et compléter le code dans le fichier **graphe_pondere1.py** afin d'implémenter le **graphe pondéré** dessiné ci-dessous avec une structure de **matrice** (en utilisant la matrice des poids).
- Retours attendus après l'appel de la fonction **dessiner3( )** dans la console :

```
>>> dessiner3(G)
```



Lever la main pour valider ce TP.

TP19 - Implémentation d'un graphe pondéré

- Sur votre compte dupliquer le fichier **graphe_pondere1.py** en **graphe_pondere2.py** .
- Modifier le code pour implémenter et dessiner le graphe pondéré correspondant à la matrice des poids ci-dessous.

$$A = \begin{pmatrix} 0 & 1 & 4 & 0 & 0 & 8 & 0 \\ 1 & 0 & 3 & 1 & 3 & 0 & 0 \\ 4 & 3 & 0 & 5 & 0 & 1 & 0 \\ 0 & 1 & 5 & 0 & 8 & 0 & 0 \\ 0 & 3 & 0 & 8 & 0 & 2 & 7 \\ 8 & 0 & 1 & 0 & 2 & 0 & 4 \\ 0 & 0 & 0 & 0 & 7 & 4 & 0 \end{pmatrix}$$

Lever la main pour valider ce TP.

TP20 - * Algorithme de Dijskstra

- Sur votre compte dupliquer le fichier **graphe_pondere2.py** en **graphe_pondere3.py** .
- Rédiger une fonction nommée **dijkstra(G, i, j)** qui prend en paramètres un graphe pondéré **G** au format matrice, et deux entier i et j correspondant à deux sommets du graphe G. Cette fonction doit retourner un **tupple** du type (liste, p) avec liste la liste des sommets ordonnés de la plus courte chaîne reliant les sommets i et j et p le poids de cette plus courte chaîne. Il faut bien sûr implémenter l'algorithme de Dijkstra pour cela. (Voir chap-titre 8 sur les réseaux : paragraphe 2.8).

Lever la main pour valider ce TP.

