Graphes et représentations

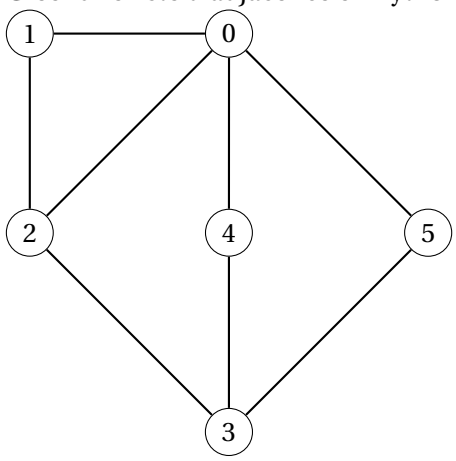
INFORMATIQUE COMMUNE - TP n° 2.3 - Olivier Reynet

À la fin de ce chapitre, je sais :

- ☞ représenter un graphe en machine par une liste d'adjacence ou une matrice d'adjacence
- ☞ transformer une représentation en une autre
- ☞ calculer les degrés des sommets d'un graphe
- ☞ transposer un graphe
- ☞ caractériser un graphe d'après sa séquence de degrés ou son spectre

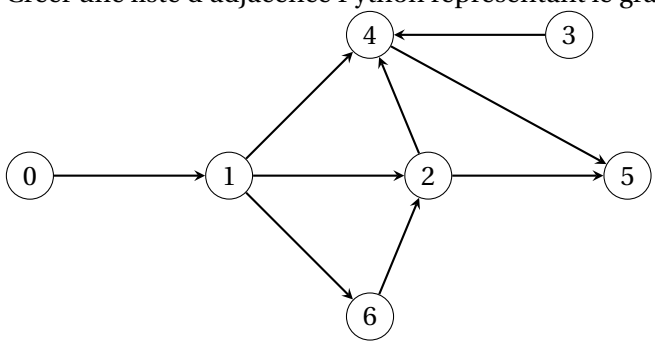
A Représentation et transformation d'un graphe

A1. Créer une liste d'adjacence en Python qui représente le graphe suivant :

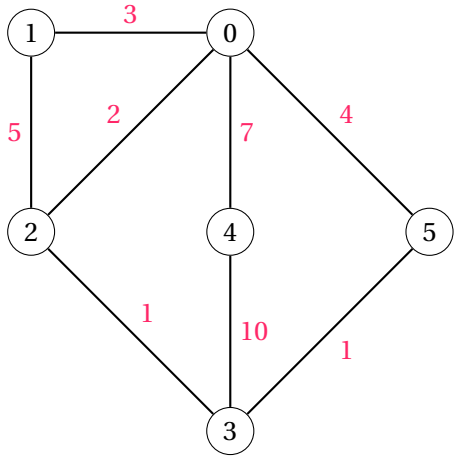


A2. Dessiner le graphe correspondant à la liste d'adjacence :
`[[1],[0,2,4,6],[1,4,5,6],[4],[1,2,3,5],[2,4],[1,2]]`

A3. Créer une liste d'adjacence Python représentant le graphe orienté suivant :



A4. Créer une liste d'adjacence Python représentant le graphe pondéré suivant :



A5. Modéliser les graphes des questions précédentes en utilisant le concept de matrice d'adjacence.

A6. Écrire une fonction de prototype `ladj_to_madj(g)` qui prend comme paramètre un graphe (non pondéré) sous la forme d'une liste d'adjacence et renvoie le même graphe sous la forme d'une matrice d'adjacence. Le type retourné est une liste de listes Python.

A7. Écrire une fonction de prototype `ladj_to_madj_n(g)` qui prend comme paramètre un graphe (non pondéré) sous la forme d'une liste d'adjacence et renvoie le même graphe sous la forme d'une matrice d'adjacence. Le type retourné est un tableau Numpy.

A8. Écrire une fonction de prototype `madj_to_ladj(g)-> list[list[int]]` qui prend comme paramètre un graphe (non pondéré) sous la forme d'une matrice d'adjacence et renvoie le même graphe sous la forme d'une liste d'adjacence. La matrice d'entrée pourra indifféremment être donnée sous la forme d'un tableau Numpy ou d'une liste de liste. La matrice de sortie sera une liste de listes.

A9. Écrire une fonction de prototype `transpose_m(g)` qui prend en paramètre un graphe orienté sous la forme d'une matrice d'adjacence et qui renvoie le graphe transposé correspondant, c'est-à-dire le graphe dont les arcs sont dirigés dans le sens opposé. Quelle est la complexité de cette fonction ?

A10. Écrire une fonction de prototype `transpose(g)` qui prend en paramètre un graphe orienté sous la forme d'une liste d'adjacence et qui renvoie le graphe transposé correspondant, c'est-à-dire le graphe dont les arcs sont dirigés dans le sens opposé. Quelle est la complexité de cette fonction ?

A11. Écrire une fonction de prototype `degrees(g)` qui renvoie la liste des degrés des sommets d'un graphe non orienté. Le paramètre est donné sous la forme d'une liste d'adjacence. Par exemple pour le graphe de la première question, la fonction renvoie : `[4, 2, 3, 3, 2, 2]`.

A12. Écrire une fonction de prototype `in_degrees(g)` qui renvoie la liste des degrés entrants des sommets d'un graphe orienté. Le paramètre est donné sous la forme d'une liste d'adjacence. Par exemple, pour le graphe orienté de la question 3, la fonction renvoie `[0, 1, 2, 0, 3, 2, 1]`.

B Deux graphes sont-ils isomorphes ?

Cette section s'attache à déterminer si deux graphes sont isomorphes. Deux approches successives sont proposées :

1. utiliser la séquence des degrés,
2. utiliser le spectre de la matrice d'adjacence.

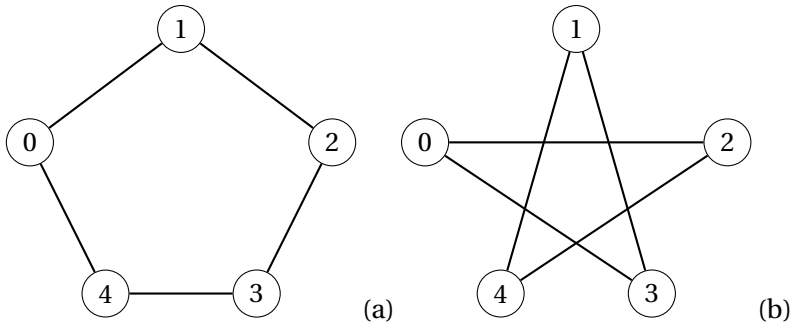


FIGURE 1 – Deux exemples de graphe : le pentagone (a), l'étoile (b)

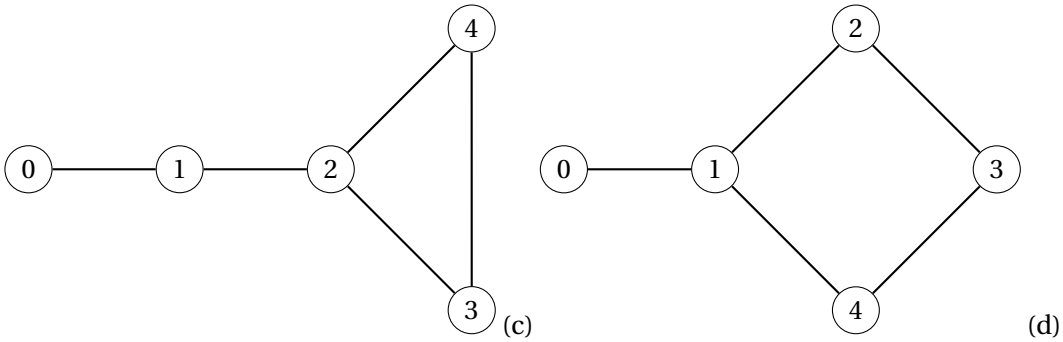


FIGURE 2 – Deux exemples de graphe : queue de poisson(c), cerf-volant (d)

B13. Donner les listes d'adjacence des deux graphes de la figure 1. Sont-ils isomorphes ?

B14. Écrire une fonction de signature `seq_degres(g: list[list[int]])-> list[int]` qui renvoie la séquence des degrés d'un graphe triée dans l'ordre croissant.

B15. En déduire une fonction `isomorphique_par_seq_deg` qui teste si deux graphes sont isomorphes d'après leur séquence de degrés.

B16. Appliquer cette fonction aux deux graphes de la figure 1 et 2. Que constatez-vous ?

B17. Donner les matrices d'adjacence des deux graphes de la figure 1.

B18. En utlisant la bibliothèque Numpy et notamment la fonction `eigvalsh` du module `numpy.linalg`, écrire une fonction de signature `spectre(matrice)` qui renvoie le spectre trié dans l'ordre croissant des valeurs propres de la matrice d'adjacence d'un graphe.

B19. En déduire une fonction `isomorphique_par_spectre(m1, m2)` qui teste si deux graphes représentés sous forme de matrice d'adjacence sont isomorphes d'après leur spectre. On pourra utiliser la fonction `allclose` pour comparer deux vecteurs de flottants.

B20. Appliquer cette fonction aux deux graphes (c) et (d) de la figure 2. Que constatez-vous ?

B21. Tester si les deux graphes de la figure 3 sont isomorphes d'après leur degré et/ou d'après leur spectre. Que pouvez-vous en conclure ?

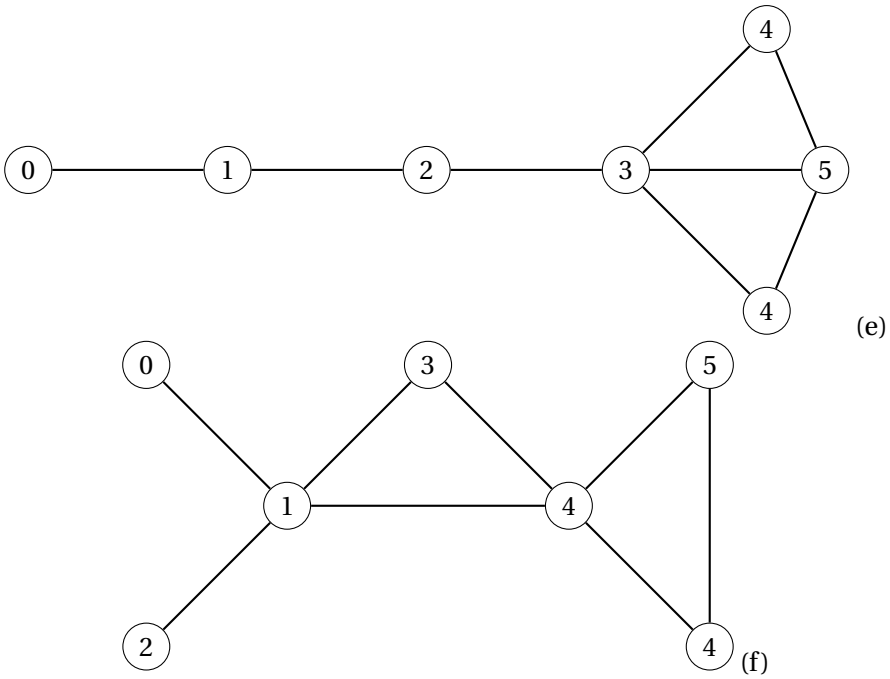


FIGURE 3 – Deux exemples de graphe : grand cerf-volant(e), passe-partout (f)