TP 14
Représentation des graphes – Sujet

Déplacement d'un cavalier sur un échiquier

Un cavalier se déplace, lorsque c'est possible, de 2 cases dans une direction verticale ou horizontale, et de 1 case dans l'autre direction (le trajet dessine une figure en L).

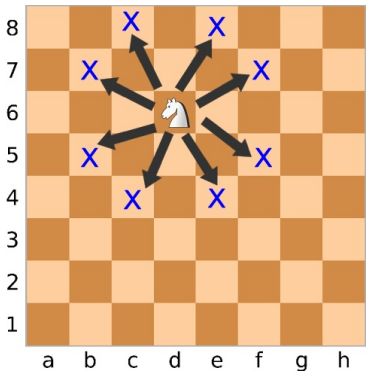

Figure 1 – Illustration du mouvement d'un cavalier sur un échiquier

Dans un premier temps, les cases de l'échiquier sont représentées par des tuples : le couple `(i,j)` désigne la case d'abscisse `i` et d'ordonnée `j`. Un échiquier possède 8 colonnes et 8 lignes, donc `i` et `j` seront compris entre `0` et `7`.

Question 1 Écrire une fonction `estDansEch(i:int, j:int)->bool` qui renvoie `True` si `(i, j)` correspond à une case valide de l'échiquier et `False` sinon.

Question 2 Écrire une fonction `mvtsPossibles(i:int, j:int)->list` qui renvoie la liste des cases où le cavalier peut se déplacer à partir de la case `(i,j)` à l'ordre près.

Question 3 Vérifier que :

- `mvtsPossibles(0,0)` renvoie `[(1, 2), (2, 1)]`,
- `mvtsPossibles(3,5)` renvoie bien `[(1, 4), (1, 6), (2, 3), (2, 7), (4, 3), (4, 7), (5, 4), (5, 6)]`,
- `mvtsPossibles(7,7)` renvoie bien `[(5, 6), (6, 5)]`.

Tous ces résultats sont à l'ordre près.

Question 4 Créer un graphe `G` sous la forme d'un dictionnaire d'adjacence avec pour sommets les différentes cases de l'échiquier et les arêtes qui correspondent à un mouvement possible du cavalier.

Question 5 Vérifiez que vous avez un graphe avec 64 sommets et 168 arêtes.

Le codage des cases d'échiquier se fait classiquement par une chaine de caractères comprenant : une lettre minuscule pour l'abscisse (de `a` à `h`) et un chiffre pour l'ordonnée (de `1` à `8`). La case en bas à gauche de l'échiquier est donc de code `'a1'`.

Remarque

`ord(c)` retourne le codage Unicode correspondant au caractère `c` et `chr(n)` renvoie le caractère dont le codage Unicode est `n`.

Question 6 Écrire une fonction `codage(i:int, j:int)->str` qui renvoie le code correspondant à la case `(i, j)`.

Question 7 Créer un dictionnaire d'adjacence `G2` comme défini précédemment avec les cases maintenant nommées d'après leur code.

Question 8 Vérifier que :

- `G2['a1']` renvoie `['b3', 'c2']`,
- `G2['d6']` renvoie bien `['b5', 'b7', 'c4', 'c8', 'e4', 'e8', 'f5', 'f7']`,
- `G2['h8']` renvoie bien `['f7', 'g6']`.

Tous ces résultats sont à l'ordre près.

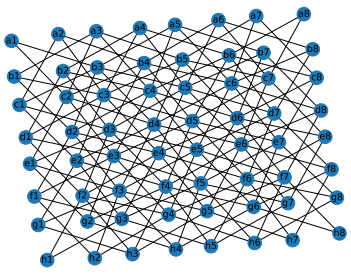
Figure 2 – Représentation graphique du graphe `G2`

Exercice d'application

Soit les graphes suivants appelés `poly`, `maison`, `foo1` et `foo2`.

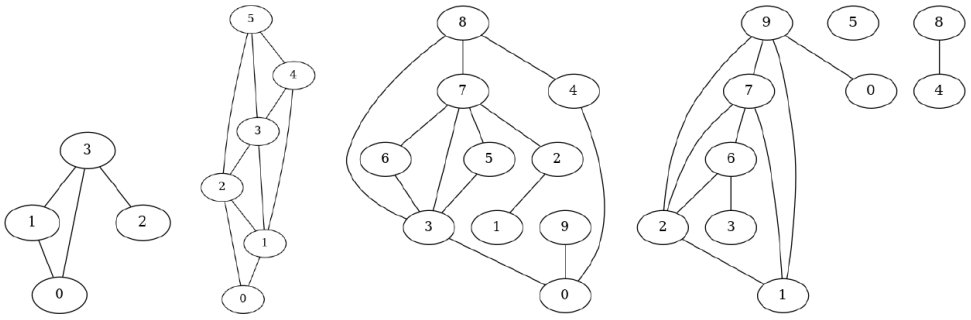

FIGURE 3 – `poly`, `maison`, `foo1` et `foo2`

Question 9 Donner la représentation sous forme de dictionnaire d'adjacence de chacun de ces graphes.

Un graphe connexe est déclaré :

- **semi-eulérien** lorsqu'il possède un chemin passant exactement une fois par chaque arête ;
- **eulérien** lorsqu'il possède un cycle (le sommet de départ est celui d'arrivée) passant exactement une fois par chaque arête ;
- **hamiltonien** lorsqu'il possède un cycle passant exactement une fois par tous les sommets (sauf celui de départ qui est celui d'arrivée).

Question 10 Parmi les graphes ci-dessus, en observer quelques-uns, et dire s'ils sont (semi-)eulériens et/ou hamiltoniens.

Il est remarquable que ces propriétés qui se ressemblent en première approximation sont en fait de complexité très différente à vérifier :

- un graphe est eulérien si, et seulement si, tous ses sommets sont de degré pair (le degré est le nombre d'arêtes issues de ce sommet) ;
- un graphe est semi-eulérien si, et seulement si, tous ses sommets sauf au plus 2 sont de degré pair ;
- pour tester le caractère hamiltonien, on ne sait pas faire bien mieux que de tester tous les chemins possibles, ce qui en fait beaucoup ! e

Attention. Le critère pour les (semi-)eulériens est « un petit peu faux ».

Question 11 Écrire un programme testant le caractère eulérien ou semi-eulérien d'un graphe.

Pour le caractère hamiltonien, on peut tenter de tester toutes les permutations possibles des sommets. On dispose d'une fonction permettant d'engendrer toutes les permutations d'une liste !

```
>>> from itertools import permutations
>>> for p in permutations(list(range(3))):
            print(p, list(p), list(p)+[p[0]])
                (0, 1, 2) [0, 1, 2] [0, 1, 2, 0]
                (0, 2, 1) [0, 2, 1] [0, 2, 1, 0]
                (1, 0, 2) [1, 0, 2] [1, 0, 2, 1]
                (1, 2, 0) [1, 2, 0] [1, 2, 0, 1]
                (2, 0, 1) [2, 0, 1] [2, 0, 1, 2]
                (2, 1, 0) [2, 1, 0] [2, 1, 0, 2]
```

Ainsi, on peut parcourir toutes les permutations ; pour chacune d'entre-elles, on ajoute le sommet de départ à la fin, et on teste la présence dans les arêtes de toutes les jonctions $\left(s_{\sigma(i)}, s_{\sigma(i+1)}\right)$.

Question 12 Écrire une fonction chemin testant si un chemin est présent dans un graphe. Écrire ensuite une fonction testant le caractère hamiltonien d'un graphe.

On peut voir sur ces tests que doit rendre la fonction lorsque le graphe est hamiltonien, un chemin le prouvant :

```
>>> chemin([0, 1, 2, 3], maison)
    True
>>> chemin([0, 1, 2, 4], maison)
    False
>>> hamiltonien(maison)
    (True, [0, 1, 3, 4, 5, 2, 0])
>>> hamiltonien(poly)
    False
```