TP

Parcours de Graphes

Exercice 1 – Distance à partir d'une case du cavalier sur un échiquier

Ressources de KOVALTCHOUK Thibaut.

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).

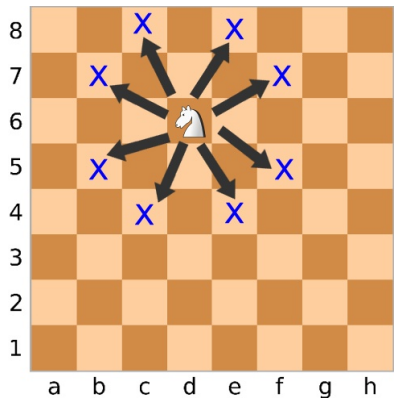

FIGURE 1 – Illustration du mouvement d'un cavalier sur un échiquier

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)` *qui renvoie la liste des cases (sous forme de tuple) où le cavalier peut se déplacer à partir de la case* `(i, j)` *à l'ordre après.*

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 *Écrire une fonction* `make_graphe` *renvoyant un graphe ayant pour sommets les différentes cases de l'échiquier et pour arêtes le mouvement possible du cavalier. Le graphe sera un dictionnaire donc les clefs seront des tuples correspondant aux cases de l'échiquier. Les valeurs associées seront des listes de tuples correspondant aux positions*

accessibles.

Question 5 *Écrire une fonction* `largeur_dist(G, dep)` *qui prend en entrée un graphe codé par un dictionnaire d'adjacence* `G` *et un sommet de départ* `dep` *et renvoie un dictionnaire de distances à partir du sommet* `dep`*. Pour ce faire, vous vous inspirerez du parcours en largeur fourni. Si un sommet n'est pas atteignable depuis le depuis* `dep`*, la valeur associée doit être de* `-1`*.*

Question 6 *Vérifier que vous obtenez le même résultat que sur la figure 2 en affichant les différentes valeurs de distance depuis* `dep = (0, 0)` *et* `dep = (4, 3)`*. La fonction* `print` *peut ne pas revenir à la ligne si on précise un argument optionnel* `end` *différent de* `\n`*. Suivant vos implémentations, la table des distances peut être « orientée » différemment.*

5	4	5	4	5	4	5	6	4	3	2	3	2	3	2	3
4	3	4	3	4	5	4	5	3	2	3	2	3	2	3	2
3	4	3	4	3	4	5	4	2	3	4	1	2	1	4	3
2	3	2	3	4	3	4	5	3	2	1	2	3	2	1	2
3	2	3	2	3	4	3	4	2	3	2	3	0	3	2	3
2	1	4	3	2	3	4	5	3	2	1	2	3	2	1	2
3	4	1	2	3	4	3	4	2	3	4	1	2	1	4	3
0	3	2	3	2	3	4	5	3	2	3	2	3	2	3	2

FIGURE 2 – Distances depuis `(0, 0)` et `(4, 3)`

Question 7 *Pourquoi le parcours en profondeur n'est pas adapté à la résolution de ce problème ?*

Exercice 2 – Traversée d'une rivière avec un loup, une chèvre et un choux

Ressources de KOVALTCHOUK Thibaut.

Sur la rive d'un fleuve se trouvent un loup, une chèvre, un chou et un passeur. Le problème consiste à tous les faire passer sur l'autre rive à l'aide d'une barque, menée par le passeur, en respectant les règles suivantes :

- la chèvre et le chou ne peuvent pas rester sur la même rive sans le passeur ;
- la chèvre et le loup ne peuvent pas rester sur la même rive sans le passeur ;
- le passeur ne peut mettre qu'un seul « passager » avec lui.

FIGURE 3 – Illustration du problème

On décide de représenter le passeur par la lettre P, la chèvre par la lettre C, le loup par L et le chou par X. Une situation est représentée par une chaine de caractères correspondant à l'ensemble des entités présentes sur la rive gauche.

Un fichier `traversee_etudiant.py` est dans votre espace de classe partagé. Il définit un graphe `G` sous la forme d'un dictionnaire d'adjacence avec pour sommets les différentes situations sous forme de chaine de caractères, et, pour arêtes, les liens possibles en effectuant un trajet unique en barque d'une rive à l'autre.

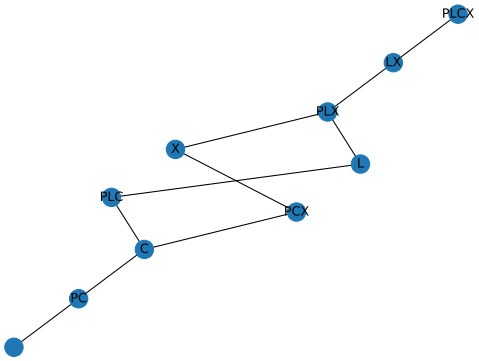

FIGURE 4 – Représentation graphique du graphe `G`

Question 1 *Écrire une fonction* `existence_chemin(G, dep, arr)` *qui prend en entrée un graphe codé par un dictionnaire d'adjacence* `G`*, un sommet de départ* `dep`*, un sommet d'arrivée* `arr` *et renvoie un booléen :* `True` *si un chemin existe entre* `dep` *et* `arr`*,* `False` *sinon. Pour ce faire, vous vous inspirerez du parcours en profondeur fourni.*

Question 2 *Vérifier que* `existence_chemin(G, "PLCX", "")` *renvoie* `True`.

Question 3 *Écrire une fonction* `trouver_chemin(G, dep, arr)` *qui renvoie une liste de sommets correspondant au chemin entre* `dep` *et* `arr`*. Vous pouvez par exemple créer un dictionnaire d'antécédent pour mémoriser le sommet d'où on vient.*