TP10

Partie 1 — Graphes

Ex. 1 — Création

Avant de commencer

Vérifier que votre interpréteur dispose bien des librairies `matplotlib` et `networkx` pour ce TP. Dans `Pycharme > File > Settings > Project TPx > Intepreter` vérifier s'ils apparaissent dans la liste, sinon cliquer sur le `+` et installer.

IMPORTANT : Dans ce TP, vous allez parfois devoir dessiner des graphes, ce qui ouvre une fenêtre supplémentaire avec le dessin du graphe demandé. Les fenêtres s'ouvrent une à la fois et il faut donc les fermer une par une pour que le code continue de s'éxécuter. Pendant les corrections automatiques, PyCharm exécute vos fichiers et il risque donc d'ouvrir énormément de fenêtres que vous devrez fermer une par une. Pour éviter cela : - quand on vous demande de dessiner un graphe et que vous voulez voir le résultat, tapez votre ligne de code normalement et **EXÉCUTEZ** le fichier (en faisant Clic Droit/Run) pour voir votre dessin; - quand vous voulez utiliser la correction automatique, **COMMENTEZ** (c'est-à-dire ajoutez un # au début de la ligne) toutes les lignes qui font des dessins (la correction automatique vous comptera quand même juste).

Graphes

Les graphes sont utilisés pour résoudre divers problèmes. Ce sont des modèles abstraits de dessins de réseaux reliant des objets et leurs applications sont nombreuses dans tous les domaines liés à la notion de réseau (réseau social, réseau informatique, télécommunications, etc.) et dans bien d'autres domaines (par exemple génétique).

Un graphe est un ensemble de points nommés nœuds (parfois sommets ou cellules) reliés par des traits (segments) ou flèches nommées arêtes (ou liens ou arcs).

On note $G = (V, E)$ un graphe G composé d'un ensemble de noeuds V et un ensemble d'arêtes E.

Implémentation par matrice d'adjacence

Rappel sur les listes. Vous avez vu lors du TP5 comment manipuler des listes. Pour accéder à l'élément d'indice i d'une liste L, on écrit `L[i]`.

Ici, nous allons manipuler des listes de listes. Par exemple, si `L = [[1,2], [3,4]]`. Dans ce cas, `L[0] = [1,2]`.

On peut donc accéder aux éléments de `L[0]` en faisant par exemple `L[0][1]` qui est égal à 2.

Matrice d'adjacence.

- Une approche pour représenter un graphe est la matrice d'adjacence. L'idée ici est de construire une matrice de taille $n \times n$ où n est le nombre de noeuds. On numérote ensuie les noeuds de 0 à n-1, correspondant aux indices de la liste. La matrice contient les valeurs 1 ou 0:
  - la valeur 1 dans la case du tableau d'indice i,j indique la présence d'une arête entre les noeuds i et j;
  - la valeur 0 indique l'absence d'arête.
- La matrice d'adjacence ci-dessous représente donc un graphe :

```
G=[[0, 1, 1, 0, 0],
   [1, 0, 1, 1, 0],
   [1, 1, 0, 1, 1],
   [0, 1, 1, 0, 0],
   [0, 0, 1, 0, 0]]
```

`G[0][1] = 1` donc les sommets 0 et 1 sont voisins.

A l'inverse, `G[0][3] = 0` ce qui signifie que les sommets 0 et 3 ne sont pas voisins.

Terminologie

- Noeud : généralement représenté par un point dans un graphe. On les numérote $0, 1, 2, ..., n-1$
- Arête : une connexion entre deux noeuds. La ligne connectant 0 et 1 est un exemple d'arête.
- Boucle : quand une arête a pour extrémités le même sommet.
- Degré d'un noeud : le nombre d'arêtes/arcs qui lui sont incidents. Le degré du noeud 2 est 4 : il a une arête vers 0, vers 1, vers 3 et vers 4.
- Adjacence : connexion(s) entre un noeud et ses voisins. Le noeud 2 est adjacent au noeud 4 car il y a une arête qui lie les deux noeuds.

Exercice

La fonction `randint(a,b)` permet de générer des nombres entiers aléatoires entre `a` et `b`.

Question 1 : Écrivez une fonction `random_row(n)` qui crée une liste de taille `n` contenant des `0` et des `1` de manière aléatoire.

Question 2 : Essayez votre fonction pour créer une liste de `6` éléments.

Question 3 : Écrivez une fonction `random_oriented_graph(n)` qui génère un graphe aléatoire contenant `n` sommets.

Question 4 : Affichez un graphe orienté aléatoire de 4 sommets généré par votre fonction.

Dans le cas d'un graphe non orienté, la matrice d'adjacence est symétrique. En effet, une arête entre i et j est aussi une arête entre j et i. On donne la fonction `symetrize(M)` qui permet de modifier la matrice donnée pour la rendre symétrique en conservant sa partie triangulaire supérieure.

On considèrera de plus que nos graphes n'ont pas de boucles, ce qui signifie que la diagonale de la matrice d'adjacence doit ne contenir que des 0.

Question 5 : Écrivez une fonction `random_graph(n)` qui génère un graphe non orienté contenant `n` sommets. Pour cela, générez un graphe orienté quelconque avec votre fonction `random_oriented_graph`, mettez des 0 sur la diagonale puis rendez-là symétrique en utilisant la fonction `symetrize`.

Question 6 : Affectez le résultat de votre fonction pour `6` sommets à une variable `G` et affichez la variable `G`.

Question 7 : Écrivez une fonction `print_graph_matrix(g)` qui afficher la matrice du graphe sous la forme :

```
[0, 1, 1, 0, 0]
[1, 0, 1, 1, 0]
[1, 1, 0, 1, 1]
[0, 1, 1, 0, 0]
[0, 0, 1, 0, 0]
```

c'est-à-dire qui affiche une ligne après l'autre.

Question 8 : Affichez votre graphe `G` avec la fonction `print_graph_matrix`.

On fournit la fonction `draw_graph(matrix)` qui prend en argument une matrice d'adjacence et dessine le graphe non orienté associé.

Question 9 : Dessinez votre graphe en utilisant la fonction `draw_graph`. (pensez à exécuter votre code pour voir le dessin puis à le commenter comme expliqué au début de l'énoncé pour utiliser la correction automatique)

Question 10 : Affichez l'évaluation d'une expression booléenne qui vérifie si le sommet 0 et le sommet 3 sont voisins.

Question 11 : Affichez l'évaluation d'une expression booléenne qui vérifie si le sommet 2 et le sommet 4 sont voisins.

Correction

```
# Les deux lignes prochaines permettent d'ajouter de nouvelles fonctions dont vous aurez bes
from tp10_tools import draw_graph, symetrize
from random import randint


# Q1
def random_row(n):
    R = []
    for i in range(n):
        R.append(randint(0, 1))
    return R


# Q2
print(random_row(6))


# Q3
def random_oriented_graph(n):
    G = []
    for i in range(n):
        G.append(random_row(n))
    return G


# Q4
print(random_oriented_graph(3))


# Q5
def random_graph(n):
    G = random_oriented_graph(n)
    for i in range(len(G)):
        G[i][i] = 0
    G = symetrize(G)
    return G


# Q6
G = random_graph(6)
print(G)

# Q7
def print_graph_matrix(g):
    for line in g:
```

```
        print(line)


# Q8
print_graph_matrix(G)

# Q9
#draw_graph(G)

# Q10
print(G[0][3] == 1)

# Q11
print(G[2][4] == 1)
```

Ex. 2 — Modification de graphe

Ajout d'arête

On donne le graphe G = [[0,1,1,1,0], [1,0,1,0,0,], [1,1,0,1,1], [1,0,1,0,0],[0,0,1,0,0]]

Question 1 : Représentez graphiquement `G` avec la fonction donnée `draw_graph`.

Question 2 : Écrivez une fonction `add_edge(G,i,j)` qui permet d'ajouter l'arête (i,j) à un graphe G donné.

Question 3 : Ajoutez l'arête (3,4) au graphe `G`.

Question 4 : Représentez graphiquement le graphe après modification et vérifiez que l'arête a bien été ajoutée.

Suppression d'arête

Question 5 : Écrivez une fonction `delete_edge(G,v1,v2)` qui permet de supprimer l'arête (v1,v2) d'un graphe G donné.

Question 6 : Supprimez l'arête (3,4) du graphe `G`.

Question 7 : Représentez graphiquement le graphe après modification et vérifiez que l'arête a bien été supprimée.

Recherche des voisin

Question 8 : Écrivez une fonction `neighbours(G, node)` qui renvoie la liste des voisins du noeud `node` dans un graphe `G`.

Question 9 : Affichez les voisins du noeud 2 dans G.

Ajout d'un noeud

Question 10 : Écrivez une fonction `add_node(G, neighbours)` qui prend en argument un graphe et la liste d'adjacence du noeud que l'on veut créer et qui ajoute au graphe G un noeud ayant les voisins demandés. Par exemple, si on veut ajouter un noeud avec pour voisins 1,3 et 4, la liste neighbours vaudra [0,1,0,1,1].

Question 11 : Ajoutez un noeud à votre graphe avec comme seul voisin le noeud 2.

Question 12 : Représentez graphiquement le graphe après modification.

Suppression d'un noeud

Question 13 : Écrivez une fonction `delete_node(G, node)` qui prend en argument un graphe et le nom d'un noeud et qui supprime le noeud donné du graphe G. On rappelle qu'on peut utiliser del G[i] pour supprimer l'élément d'indice i de G.

Question 14 : Supprimez le noeud 5 de votre graphe.

Question 15 : Représentez graphiquement le graphe après modification.

Correction

```
# La ligne prochaine permet d'ajouter une nouvelle fonction dont vous aurez besoin après. Ne
from tp10_tools import draw_graph

G = [[0, 1, 1, 1, 0], [1, 0, 1, 0, 0, ], [1, 1, 0, 1, 1], [1, 0, 1, 0, 0], [0, 0, 1, 0, 0]]

# Q1
# draw_graph(G)


# Q2
def add_edge(G, v1, v2):
    if G[v1][v2] == 0:
        G[v1][v2] = 1
        G[v2][v1] = 1


# Q3
add_edge(G, 3, 4)

# Q4
```

```
# draw_graph(G)


# Q5
def delete_edge(G, v1, v2):
    if G[v1][v2] == 1:
        G[v1][v2] = 0
        G[v2][v1] = 0


# Q6
delete_edge(G, 3, 4)

# Q7
# draw_graph(G)


# Q8
def neighbours(G, node):
    l = []
    for i in range(len(G)):
        if G[node][i] == 1:
            l.append(i)
    return l


# Q9
print(neighbours(G, 2))


# Q10
def add_node(G, neighbours):
    for i in range(len(G)):
        G[i].append(neighbours[i])
    neighbours.append(0)
    G.append(neighbours)

# Q11
add_node(G, [0, 0, 1, 0, 0])

# Q12
# draw_graph(G)


# Q13
def delete_node(G, node):
```

```
    if node < len(G):
        del (G[node])
    for i in range(len(G)):
        del (G[i][node])


# Q14
delete_node(G, 5)

# Q15
# draw_graph(G)
```

Ex. 3 — Graphe eulérien

On donne le graphe G = [[0,1,1,1,0], [1,0,1,0,0,], [1,1,0,1,1], [1,0,1,0,0],[0,0,1,0,0]].

On définit le degré d'un noeud v du graphe G par le nombre d'arêtes ayant v comme extrémité.

Degré d'un noeud

Question 1 : Écrivez une fonction `node_degree(G, node)` qui renvoie le degré du noeud `node` dans le graphe G.

Question 2 : Affichez le degré du noeud 2 du graphe G.

Existence d'un cycle eulérien

Vous avez vu en Introduction à l'informatique qu'un graphe non orienté connexe admet un cycle qui traverse chaque arête exactement une fois (un cycle eulérien) si et seulement si chaque sommet est de degré pair.

Question 3 : Écrivez une fonction `eulerian_cycle(G)` qui renvoie `True` si le graphe `G` contient un cycle eulérien et `False` sinon.

Question 4 : On donne G = [[0,1,1,1], [1,0,1,1], [1,1,0,1], [1,1,1,0]] . Affichez si G contient un cycle eulérien ou non.

Question 5 : On donne G = [[0,1,1,1,1], [1,0,1,1,1], [1,1,0,1,1], [1,1,1,0,1],[1,1,1,1,0]]. Affichez si G contient un cycle eulérien ou non.

Correction

```
G = [[0,1,1,1,0], [1,0,1,0,0,], [1,1,0,1,1], [1,0,1,0,0],[0,0,1,0,0]]
```

```
# Q1
def node_degree(G, node):
    cpt = 0
    for edge in G[node]:
        if edge==1:
            cpt = cpt + 1
    return cpt


# Q2
print(node_degree(G,2))


# Q3
def eulerian_cycle(G):
    for node in range(len(G)):
        if node_degree(G, node) % 2 == 1:
            return False
    return True


# Q4
print(eulerian_cycle([[0,1,1,1], [1,0,1,1], [1,1,0,1], [1,1,1,0]]))


# Q5
print(eulerian_cycle([[0,1,1,1,1], [1,0,1,1,1], [1,1,0,1,1], [1,1,1,0,1],[1,1,1,1,0]]))
```