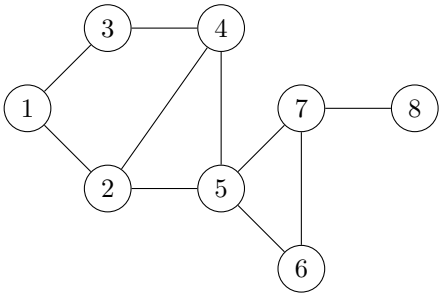
Graphes et Algorithmes

Travaux Pratiques n°1 (Correction)

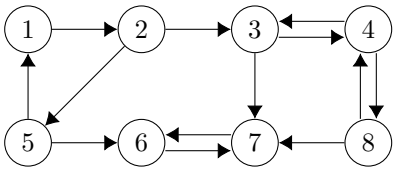
Université de la Nouvelle-Calédonie, 2024

Introduction

Durant ce TP, nous allons – enfin vous allez – chercher à définir et manipuler un graphe. Dans la théorie des graphes, on retrouve deux types de graphes : les graphes non orientés et les graphes orientés. Les arêtes des graphes peuvent être associées à des valeurs. Si aucune valeur n'est affichée, on considérera que sa valeur vaut 1 (pour représenter la présence d'un lien entre deux sommets).



Graphe non orienté.



Graphe orienté.

Exercice 1

Construire la matrice d'adjacence associée au graphe orienté ci-dessus.
Vous pouvez utiliser `numpy.zeros((`m`,`n`))` pour instancier un tableau (rempli de 0) à deux dimensions avec $m \times n = 0$.

Solution :

```python
import numpy as np

M = np.zeros((8,8))

M[0, 1] = 1 # 1 - 2
M[1, 4] = 1 # 2 - 5
M[1, 2] = 1 # 2 - 3
M[2, 0] = 1 # 3 - 1
M[2, 3] = 1 # 3 - 4
M[3, 2] = 1 # 4 - 3
M[4, 0] = 1 # 5 - 1
M[3, 7] = 1 # 4 - 8
M[7, 3] = 1 # 8 - 4
M[5, 6] = 1 # 6 - 7
M[6, 5] = 1 # 7 - 6
M[2, 6] = 1 # 3 - 7
M[7, 6] = 1 # 8 - 7

# ou

M = np.array([
    [0, 1, 0, 0, 0, 0, 0, 0],
    [0, 0, 1, 0, 1, 0, 0, 0],
    [0, 0, 0, 1, 0, 0, 1, 0],
    [0, 0, 1, 0, 0, 0, 0, 1],
    [1, 0, 0, 0, 0, 1, 0, 0],
    [0, 0, 0, 0, 0, 0, 1, 0],
    [0, 0, 0, 0, 0, 1, 0, 0],
    [0, 0, 0, 1, 0, 0, 1, 0]
])
```

Exercice 2

Écrire une fonction `is_directed(M)` qui renvoie `True` si le graphe associé à la matrice d'adjacence M est orienté. Sinon, il renvoie `False`. Pour cet exercice, n'utilisez aucune méthode de `numpy` !

Solution :

```
# M est la matrice d'adjacence

def is_directed(M):
    for i in range(M.shape[0]):
        for j in range(i+1, M.shape[0]):
            if M[i,j] != M[j,i]:
                return True
    return False
```

Exercice 3

Écrire une fonction `remove_edge(`x`, `y`, `M`)` qui supprime l'arête allant du noeud x vers le noeud y dans une matrice d'adjacence M.

Solution :

```
def remove_edge(x, y, M):
    res = np.array(M, copy=True)
    res[x-1, y-1] = 0
    return res

matAdjacence = remove_edge(x, y, matAdjacence)
```

Exercice 4

Écrire une fonction `has_path(`x`, `y`, `k`, `M`)` qui renvoie `True` si dans un graphe associé à la matrice d'adjacence M, il existe (au moins) un chemin de longueur k entre les noeuds x et y. Sinon, il renvoie `False`.

Solution :

```
def has_path(x, y, k, G):
    Gk = np.linalg.matrix_power(G, k)
    return Gk[x-1, y-1] > 0
```

Exercice 5

Écrire une fonction `vertex_degree(`x`, `G`)` qui renvoie sous la forme d'un tuple, le degré entrant et le degré sortant du noeud x dans le cas d'un graphe orienté, et sous la forme d'un entier le degré du noeud dans le cas d'un graphe non orienté.

Solution :

```
def vertex_degree(x, G):
    if is_directed(G):
        return (sum(G[:, x-1]), sum(G[x-1, :]))
    return sum(G[:, x-1])
```

Exercice 6

Un graphe est dit complet si et seulement si tous les sommets sont adjacents deux à deux. Écrire une fonction `is_complete(M)` qui renvoie `True` si la matrice d'adjacence en entrée est associée à un graphe complet. Sinon, il renvoie `False`. N'oubliez pas de prendre en compte le sens des arêtes du graphe.

Solution :

```python
# M est la matrice d'adjacence
def is_complete(M):
    for i in range(M.shape[0]):
        for j in range(i+1, M.shape[0]):
            if (M[i,j] == 0) or (M[j,i] == 0):
                return False
    return True
```