

TP : Théorie des graphes

- Le rendu doit être un fichier .zip contenant : un compte rendu contenant les réponses aux questions ainsi que les programmes Python (les fichiers .py).
- Vous pouvez travailler en binôme.

1 Introduction

Dans ce TP nous allons utiliser plusieurs bibliothèques bien connues en Python afin de représenter des graphes. On commencera donc par ajouter en début de script les lignes suivantes :

```
1 import networkx as nx #Cette bibliothèque permettra de manipuler des graphes
2 import matplotlib.pyplot as plt #Cette bibliothèque permettra de les représenter
3 import numpy as np
```

Manipulation de graphes à l'aide du package networkx

Le but de cette section est de vous apprendre à définir et à manipuler des graphes à l'aide du package **networkx**. Néanmoins, si besoin, je vous invite par exemple à visiter ce site où à mener vos propres recherches pour vous familiariser avec cette bibliothèque. Voici donc quelques opérations simples sur les graphes accessibles grâce à cette bibliothèque.

Créer un graphe :

```
1 G = nx.Graph() #Crée un graphe G non orienté
2 G2= nx.DiGraph() #Crée un graphe G orienté
```

Ajouter/supprimer de sommets :

```
1 G.add_node(1) #Ajoute le sommet 1
2 G.add_nodes_from([2,3]) #Ajoute les éléments de la liste comme sommets
3 G.remove_node(1) #Supprime le sommet 1
4 G.remove_nodes_from([1,2]) #Supprime les éléments de la liste comme sommet
```

Ajouter/supprimer d'arrêtes :

```
1 G.add_edge(1,2) #Ajoute l'arrête 1-2 (et ajoute le sommet s'il n'existe pas)
2 G.add_edge(2,4,weight=4) #Ajoute l'arrête 2-4 avec une pondération de 4
3 G[2][4]['weight'] #Retourne le poids de l'arrête 2-4
4 G.add_edges_from([(1,2),(1,3)]) #Ajoute les arrêtes à partir de la liste
5 G.add_weighted_edges_from([(1,2,0.125),(1,3,0.75),(2,4,1.2),(3,4,0.375)])
6 #Ajoute les arrêtes pondérées à partir de la liste
7 G.remove_edge(1,3) #Supprime l'arrête 1-3
8 G.remove_edges_from([(1,2),(0,1)]) #Supprime les arrêtes à partir de la liste
```

Représenter un graphe :

```
1 nx.draw(G) #Représente le graphe G en plaçant les sommets de façon aléatoire
2 plt.show() #Utile surtout si vous tracez plusieurs graphes dans un seul script
```

Charger un graphe depuis un fichier :

```
1 GG=nx.read_adjlist("./test.txt") #Charge le graphe à partir du fichier test.txt
2 nx.draw(GG)
3 nx.write_adjlist(G,"test2.txt") #Sauve le graphe G dans le fichier test2.txt
```

où le fichier text.txt est un fichier texte de la forme

```
1 1 2 3 4
2 2 5
```

où le premier nombre de la ligne est le nom du sommet et ensuite ce sont les noms de ses voisins qui sont renseignés.

Exemples de fonctions utiles :

```
1 G.number_of_nodes() #Nombre de sommets de G
2 G.number_of_edges() #Nombre d'arrêtes de G
3 G.nodes() #Liste des sommets de G
4 G.edges() #Liste des arrêtes de G
```

Exercice 1

Le but de ce premier exercice est simplement de mettre en œuvre les différentes notions introduites précédemment :

1. Créer un graphe G et un graphe orienté DG .
2. Ajouter à chaque graphe les sommets nommés 1, 2, 3, 4 et 5.
3. Afficher les sommets des graphes.
4. Supprimer le sommet 1 du graphe G .
5. Ajouter au graphe G les arêtes $\{2, 3\}$, $\{2, 5\}$, $\{3, 4\}$ et $\{4, 5\}$.
6. Ajouter au graphe DG les arêtes $(1, 3)$, $(2, 3)$, $(2, 4)$, $(2, 5)$, $(4, 5)$ et $(5, 1)$.
7. Tracer le graphe G et le graphe DG . Quelles sont les limites d'un tel affichage ?

Exercice 2

Représenter les graphes suivants :



Exercice 3

Écrire un script qui crée un graphe non orienté dont :

1. Le nombre de sommets est choisi aléatoirement entre 2 et 15.
2. L'existence d'une arête entre deux sommets existants est aléatoire.

Indication : On pourra utiliser la fonction **random** du package **random**.

Améliorations possibles de la représentation graphique

Afin d'obtenir une représentation graphique plus lisible des graphes, la bibliothèque **networkx** permet d'utiliser d'avantage d'options comme la coloration d'un nœud, l'ajout d'une étiquette à un nœud où encore l'ajout du poids à une arête.

Déclaration du graphe :

```
1 import matplotlib.pyplot as plt
2 import networkx as nx
3 from numpy import array
4 # Définition du graphe
5 G = nx.Graph()
6 # Définition des noeuds
7 G.add_node(0, label='A', col='pink')
8 G.add_node(1, label='B', col='red')
9 G.add_node(2, label='C', col='white')
10 G.add_node(3, label='D', col='white')
11 G.add_node(4, label='E', col='white')
12 G.add_node(5, label='F', col='blue')
13 # Définition des aretes
14 G.add_edge(0, 1, weight=6)
15 G.add_edge(0, 2, weight=5)
16 G.add_edge(0, 4, weight=1)
```

```

17 G.add_edge(4,1,weight=5)
18 G.add_edge(4,2,weight=1)
19 G.add_edge(4,3,weight=3)
20 G.add_edge(2,3,weight=8)
21 G.add_edge(4,5,weight=6)
22 G.add_edge(3,5,weight=9)
23 # Représentation du graphe basique
24 nx.draw(G)
25 plt.show()

```

Traitement des nœuds :

```

1 # Colorer les nœuds :
2 liste = list(G.nodes(data='col'))
3 colorNodes = {}
4 for noeud in liste:
5     colorNodes[noeud[0]]=noeud[1]
6 print(colorNodes)
7 # Pour colorer les nœuds, le format doit être une liste :
8 colorList=[colorNodes[node] for node in colorNodes]
9 print(colorList)
10 # Etiquetter les nœuds : le format doit être un dictionnaire :
11 liste = list(G.nodes(data='label'))
12 labels_nodes = {}
13 for noeud in liste:
14     labels_nodes[noeud[0]]=noeud[1]
15 print(labels_nodes)

```

Traitement des sommets :

```

1 # Pour les étiquettes des arrêtes, le format doit être un dictionnaire :
2 labels_edges = {}
3 labels_edges = {edge:G.edges[edge]['weight'] for edge in G.edges}
4 print(labels_edges)

```

Une autre rédaction si celle ci-dessus pose problème :

```

1 # Pour les étiquettes des arrêtes , le format doit être un dictionnaire :
2 liste_edge = list(G.edges(data='weight'))
3 labels_edges = {}
4 for edge in liste_edge :
5     labels_edges[edge[0],edge[1]]= edge[1]
6 print(labels_edges)

```

Représentation du graphe :

```

1 # Positions des sommets :
2 pos = nx.spring_layout(G)
3 # Sommets :
4 nx.draw_networkx_nodes(G, pos, node_size=700,node_color=colorList,alpha=0.9)
5 # Etiquettes des sommets :
6 nx.draw_networkx_labels(G, pos, labels=labels_nodes, \
7                         with_labels=True,font_size=20, \
8                         font_color='black', \
9                         font_family='sans-serif')
10 # Arrêtes :
11 nx.draw_networkx_edges(G, pos,width=1)
12 nx.draw_networkx_edge_labels(G, pos, width=1, edge_labels=labels_edges,
13 font_color='red')
14 plt.axis('off')
15 plt.show()

```

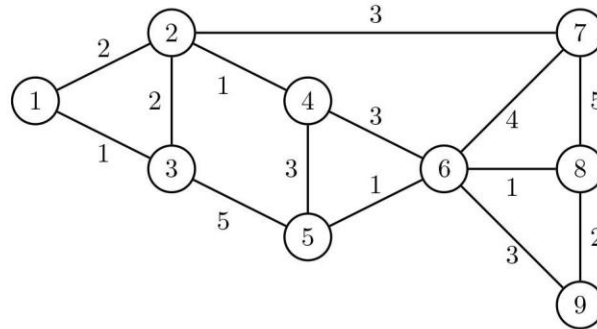

Question 1

Expliquez en détail ce que fait chacune des parties de ces scripts.

2 Recherche d'un arbre couvrant de poids minimal

Exercice 5

On considère le graphe G suivant :



1. Représenter ce graphe en faisant apparaître les étiquettes des sommets et les poids des arrêtes (on pourra utiliser la fonction **representation** de l'exercice précédent).
2. Écrire une fonction **testnonisole** prenant en argument un graphe G et renvoyant True si le graphe G ne possède pas de sommet isolé et False si le graphe G possède un sommet isolé (c'est-à-dire relié à aucun autre sommet). *Indication* : On pourra utiliser la fonction **nx.degree**.
3. Expliquer ce que fait la fonction **nx.cycle_basis**.
4. Écrire une fonction **AlgoACM** prenant en entrée un graphe G et retournant un arbre couvrant de poids minimal ainsi que son poids. *Indication* : On pourra utiliser la fonction **nx.is_connected(G)**.
5. Appliquer cette fonction au graphe G précédent et représenter l'arbre couvrant de poids minimal obtenu à l'aide de la fonction **representation** de l'exercice précédent.