GRAPHES
Notions élémentaires et TP

I. Notions fondamentales sur les graphes

1) Vocabulaire préliminaire

• Un **graphe non orienté** $G = (S, A)$ est une liste finie S d'éléments, appelés **sommets**, et d'une liste A de paires de sommets, appelés **arêtes** (*rappel : une paire est un ensemble à 2 éléments*).

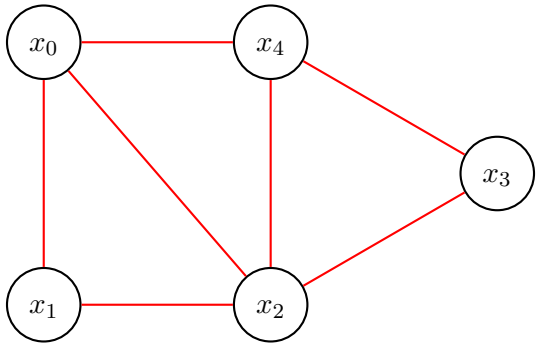

FIGURE 1 – Représentation d'un graphe G_1 non orienté à 5 sommets

• Un **graphe non orienté** $G = (S, A)$ est une liste finie S d'éléments, appelés **sommets**, et d'une liste A de **couples** de sommets, appelés **arcs** (*rappel : un couple est une liste – orientée – 2 éléments*). On distingue alors le sommet origine de l'arc et son extrémité.

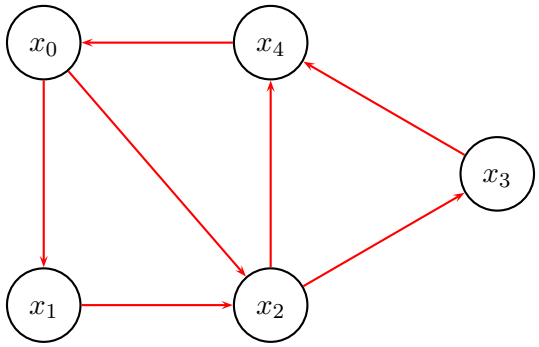

FIGURE 2 – Représentation d'un graphe orienté à 5 sommets

• L'**ordre** d'un graphe est le nombre de sommets de ce graphe.
• Deux sommets d'un graphe reliés par au moins une arête (ou un arc) sont dits **adjacents**.
• Une arête (ou un arc) partant et arrivant au même sommet est appelée **boucle**.
• Un graphe non orienté est dit **simple** s'il est sans boucle et si chaque couple de sommets est relié par au plus une arête (un sommet n'est donc pas adjacent à lui-même).

À partir de maintenant, on travaille avec des graphes *simples et non orientés*.

• Un graphe **complet** est un graphe simple dont tous les sommets sont adjacents.

• Soit G un graphe simple non orienté, de sommets $(x_i)_{i\in I}$, ($I \subset \mathbb{N}$, I fini).
On définit la **matrice d'adjacence** M du graphe G comme la matrice carrée d'ordre n dont le coefficient en ligne $i \in I$, colonne $j \in I$ vaut :
– pour les graphes non pondérés : 1 si le sommet x_i est relié au sommet x_j par une **arête**, 0 sinon.
– pour les graphes pondérés (cf. exemple en FIGURE 3) : le **poids** de l'arête reliant les sommets x_i et x_j.

2) Exercices

Exercice n° 1 : Fonctions élémentaires

On choisit de définir une matrice d'adjacence, en langage Python, par un **tableau Numpy**.

1. Écrire une fonction `sont_egaux`, prenant pour paramètres deux tableaux Numpy de même dimension, et retournant le booléen attestant de l'égalité des deux matrices associées.
2. Définir la matrice d'adjacence du graphe G_1 défini en *figure 1*.
3. Programmer alors une fonction `est_complet`, ayant pour paramètre la matrice d'adjacence M (donc carrée) d'un graphe simple non orienté, et retournant un booléen répondant à la question : "le graphe associé est-il complet ?"
   On pourra insérer au début de la fonction la commande : `assert len(M)==len(M[0])`*... et analyser son rôle.*

Exercice n° 2 : Graphe défini par la liste de ses arêtes

Soit G un graphe d'ordre n, de sommets : $\{0, 1, \ldots, n-1\}$.
On peut définir G par son ordre, doté de la liste de ses arêtes, c'est-à-dire la liste des couples $[i, j]$ tels qu'il existe une arête reliant le sommet x_i et le sommet x_j.
On veut savoir comment passer de cette définition à la matrice d'adjacence.

1. *Étude d'un exemple pour $n = 5$* : représenter (sur feuille) le graphe (non orienté) G_2 à 5 sommets, défini par la liste de ses arcs :
   $$A_2 = [[0, 1], [2, 0], [1, 2], [2, 3], [3, 1]].$$
2. *Cas général* : Écrire une fonction `conversion` de paramètres un entier n et une liste A de couples, et qui retourne la matrice d'adjacence du graphe associé.
   On pourra initialiser la matrice nulle d'ordre n par la commande NumPy : `zeros((n,n))`.
   Pourquoi le paramètre n est-il obligatoire ?
3. Tester votre fonction sur le graphe G_2.

Exercice n° 3 : Condition de pondération d'une matrice d'adjacence

1. Écrire une fonction `estPondere()` qui admet une matrice d'adjacence en paramètre et retourne `True` si elle contient au moins 1 coefficient différent de 0 et de 1, `False` sinon.
2. Tester cet algorithme le graphe pondéré :

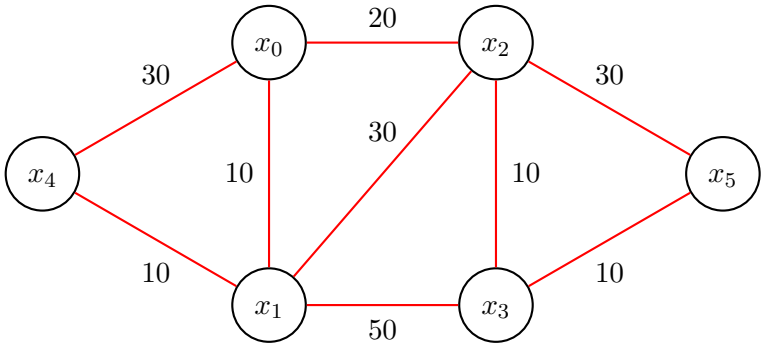

FIGURE 3 – Graphe G_3, simple non orienté mais pondéré

dont la matrice d'adjacence est (copiez-collez !) :

```
G3=np.array([[0,10,20,0,30,0],[10,0,30,50,10,0],[20,30,0,10,0,30],[0,50,10,0,0,10],
[30,10,0,0,0,0], [0,0,30,10,0,0]])
```

II. Théorèmes d'Euler

Dans toute la suite, pour simplifier les notations, les sommets d'un graphe (simple non orienté) G d'ordre n seront notés : $\{0, 1, \ldots, n-1\}$.

1) Vocabulaire préliminaire

• Le **degré** d'un sommet est le nombre d'arêtes (ou d'arcs) dont il est l'une des extrémités[1].

Soient $(i, j) \in [\![0, n]\!]^2$.

• On appelle **voisins d'un sommet** i la liste des sommets qui sont reliés à i (dans un graphe simple, un sommet n'est pas relié à lui-même).

• On appelle **chaîne** de i à j une suite d'arêtes qui permettent de relier les sommets i et j.

• Un **cycle** est une chaîne dont les arêtes sont *distinctes* et dont les extrémités coïncident (elle relie donc un sommet à lui-même).

• Un graphe est dit **connexe** si on peut relier deux quelconques de ses sommets par une chaîne (éventuellement réduite à une arête).

• Une chaîne est dite **eulérienne** si elle satisfaisait aux conditions suivantes :

– elle contient toutes les arêtes du graphe ; – chaque arête n'est parcourue qu'une seule fois.

• Un cycle eulérien est un cycle qui est une chaîne eulérienne.

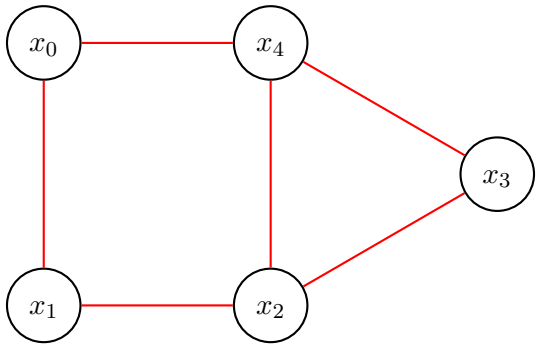

FIGURE 4 – Graphe contenant une chaîne eulériennen par exemple : 4 - 0 - 1 - 2 - 4 - 3 - 2

1. Pour les graphes non simples et non orientés, une boucle compte pour 2. Pour les graphes orientés, on définit parfois la notion de *degré entrant* et *degré sortant*.

REMARQUE : un graphe contient un cycle eulérien si l'on peut le parcourir entièrement "sans lever le crayon" et sans passer 2 fois par la même arête.

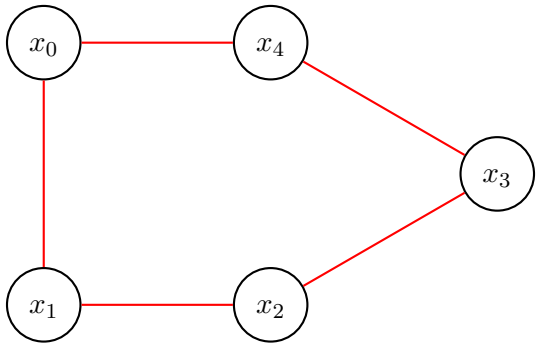

FIGURE 5 – Graphe contenant un cycle eulérien

2) Exercices sur machine

Exercice n° 4 : Degré des sommets - chaine et cycle eulérien

Dans cet exercice, on considère des graphes connexes, simples, non orientés et non pondérés.

1. Proposer une fonction `degre(A,i)`, de paramètres une matrice d'adjacence A et un sommet i, et qui renvoie le degré de ce sommet.
2. Écrire une fonction `voisins(A,i)`, de paramètres une matrice d'adjacence A et un sommet i, et retournant la liste de ses voisins.

   On admet à présent les 2 **théorèmes d'Euler**

   THEOREME 1 *:*

   Un graphe connexe admet un cycle eulérien si et seulement si tous les sommets sont de degré pair.

   THEOREME 2 *:*

   Un graphe connexe admet une chaîne eulérienne entre les sommets A et B si et seulement si A et B sont les seuls sommets de degré impair.

3. Programmer alors une fonction `cycleEulerien(A)` admettant en paramètre une matrice d'adjacence A d'un graphe connexe et qui renvoie `True` si le graphe associé admet *un cycle eulérien*, `False` sinon.

4. Programmer alors une fonction `chaineEulerienne(A)` admettant en paramètre une matrice d'adjacence A d'un graphe connexe et qui retourne `True` si le graphe associé admet *une chaîne eulérienne*, `False` sinon.

Exercice n° 5 : Connexité d'un graphe

1. Écrire une fonction `famille(M,i)`, de paramètres une matrice d'adjacence M et un sommet i, et retournant la liste de TOUS les sommets qu'on peut atteindre depuis le sommet i par une chaîne.

On pourra utiliser la fonction `voisins` *de l'exercice précedent, et supprimer à la fin les voisins qui apparaissent plusieurs fois dans la liste.*

2. Écrire enfin une fonction `connexe()`, de paramètres une matrice d'adjacence d'un graphe simple, non orienté, non pondéré, et qui retourne True si le graphe associé est connexe, False sinon.

III. Algorithme de Dijkstra

Cet algorithme permet de déterminer le plus court chemin entre deux sommets d'un graphe.

Pour la présentation de l'algorithme sur l'étude d'un exemple, voir le pdf donné dans votre dossier de TP. Pour plus interactif, on pourra consulter (ultérieurement le tutoriel) :

`https://www.youtube.com/watch?v=k7MsXexTIgE`

Exercice n° 6 : Plus court chemin - Algorithme de Dijkstra

1. On travaille à nouveau sur le graphe pondéré G_3.

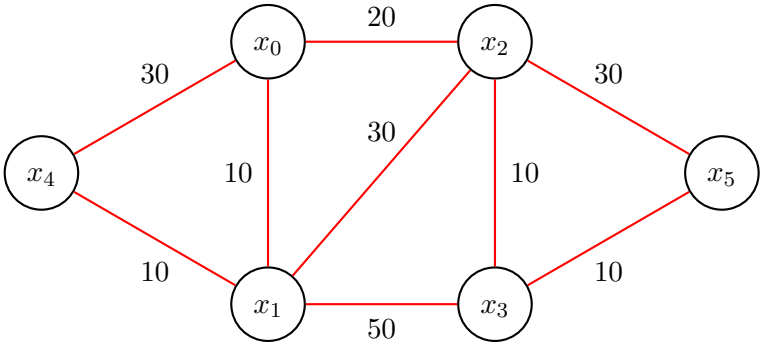

FIGURE 6 – graphe simple non orienté

Appliquer sur papier l'algorithme de Dijkstra pour obtenir le (ou les) plus court(s) chemin(s) du sommet x_4 au sommet x_5.

2. On donne ci-derrière, et dans le dossier du TP, un code de l'algorithme de Dijkstra.

L'analyser et le compléter.

```
1  def Dijkstra(G,sdebut):
2      x,y = np.shape(G)
3      assert x==y
4      n=x
5      assert sdebut < n-1
6      dinfini = 1000
7
8      SProv = []       #Création de la liste des couples [d,sp]
9           #( [distance,sommet précédent]) des sommets provisoirement pondérés ("PP")
10
11     S = []           #Création de la liste des sommets marqués
12
13     tableauAnalyse = []         #Création du tableau d'analyse
14
15
16     T = list(range(0,n))       #Initialisation de la liste des sommets "PP"
17
18     for i in T:
19         SProv.append([dinfini,0])    #Création de la 1re liste des couples [d,sp]
20
21     SProv[sdebut][0] = 0             #La distance du sommet de début est fixée à 0
22     SProv[sdebut][1] = sdebut         #et son prédecesseur est lui-même
23
24
25     T.remove(sdebut)          #Le sommet de début est marqué définitivement
26                              # et donc enlevé de la liste des sommets provisoires
27     S.append(sdebut)      #Il est ajouté à l'ensemble des sommets marqués
28
29
30
31     #Chaque sommet adjacent est marqué du poids de l'arête la reliant à sdebut
32     for i in T:
33         print(i,G[sdebut][i])
34         if G[sdebut][i] != 0:
35             print ("Détection pour " ,i)
36             SProv[i][0] = G[sdebut][i]
37             SProv[i][1] = sdebut
38
39     print ("Initialisation", T,SProv)
40
41     #On termine en créant la première ligne du tableau d'analyse
42
43     tableauAnalyse.append(SProv.copy())
44
45     ligne = 0 #ligne courante du tableau d'analyse
46
47     while len(T)>0:
48
49                 #A vous de coder
50
51     return tableauAnalyse,S
```

3. Coder un protocole de test validant la résolution sur papier.