Institut National Universitaire Champollion

TP 1 - Représentations d'un graphe. Premières fonctions.

Théorie et Algorithmique des Graphes
L3 INFO - Semestre 6

Le but de ce premier TP est de mettre en oeuvre en Python, la représentation d'un graphe par listes d'adjacences, et de commencer à utiliser cette représentation pour écrire nos premières fonctions simples de traitement des graphes.

Avertissement : Python ne distingue pas les vecteurs des listes (tous les deux sont mis en œuvre à l'aide du type `List` de Python). Cependant, cette façon de faire est assez rare dans les langages de programmation et nous aurons donc le souci de parler de vecteur lorsque la taille est immuable et que l'on souhaite faire des accès directs aux éléments (en lecture ou en écriture) et de parler de liste lorsque la taille de l'objet est variable et que l'on enlève ou ajoute des éléments à cet objet (en début ou en fin de liste).

Nous manipulerons donc toujours les vecteurs et matrices par création et initialisation (directement ou à l'aide de primitives fournies dans le fichier vect.py) puis accès direct à leurs éléments. Ils devront toujours être de taille constante.

Exemple : Le code

```
from vect import *

V = initVect(3,0)
V[2] = 1
print("V = ", V)

W = initVectList(4)
W[1] = [1,2]
print("W = ", W)

M = initMat(2,0)
M[0][0] = 4
print("M = ", M)
```

renvoie

```
V =  [0, 0, 1]
W =  [[], [1, 2], [], []]
M =  [[4, 0], [0, 0]]
```

En revanche, pour les listes, nous utiliserons les méthodes d'insertion, d'ajout ou de suppression avec la syntaxe pointée, après initialisation (souvant à la liste vide).

Exemple :
Le code suivant

```
L = []
L.append(1)
print("L = ", L)

L.pop(0)
print("L =", L)
```

renvoie

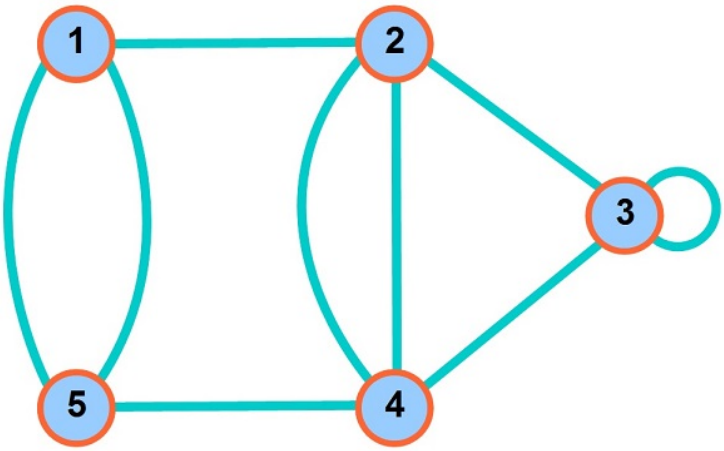
```
L =  [1]
L =  []
```

Nous parlerons donc du **vecteur des listes d'adjacences**.

Les éléments d'un objet de type `List` sont indexées à partir de 0 en Python. Or nous souhaitons pour faciliter la compréhension des algorithmes, que $G[i]$ soit la liste d'adjacence du sommet i. Nous prendrons donc l'habiture de commencer notre vecteur de listes d'adjacence par une première liste vide (indexée par 0).
En revanche la matrice d'adjacence sera bien indexée de 0 à $n-1$, l'élément $M[i-1][j-1]$ concernant l'arête entre les sommets i et j.

1. Les graphes non orientés.

Considérons le multigraphe non orienté G_2 suivant :



Rappelons que ce graphe peut-être représenté en Python par ses listes d'adjacences ou par sa matrice d'adjacence. D'après ce qui a été dit plus haut, le graphe G_2 sera représenté

- par la liste d'arêtes $L = [[1,2],[1,5],[1,5],[2,3],[2,4],[2,4],[3,3],[3,4],[4,5]]$
- par la liste d'adjacence $G = [[],[2,5,5],[1,3,4,4],[2,3,4],[2,2,3,5],[1,1,4]]$
- et par la matrice d'adjacence

$$M = \begin{pmatrix} 0,1,0,0,2 \\ 1,0,1,2,0 \\ 0,1,1,1,0 \\ 0,2,1,0,1 \\ 2,0,0,1,0 \end{pmatrix}$$

Avec ces conventions, le parcours du vecteur des listes d'adjacence s'écrira `for i in range(1,len(G))` et le parcours du voisinage du sommet i s'écrira `for j in G[i] :`.

1.1. Premières fonctions

G est un graphe ou un multigraphe non orienté, représenté par liste d'adjacence. Écrire les fonctions suivantes :

1. `nbSommets(G)` : retourne le nombre de sommets du graphe G
2. `nbArete(G)` : retourne le nombre d'arêtes du graphe G
3. `ajoutArete(G,i,j)` : ajoute l'arête (i,j) au graphe G
4. `enleveArete(G,i,j)` : enlève une arête (i,j) du graphe G (si elle existe !)
5. `deg(G,i)` : retourne le degré du sommet i
6. `degre(G)` : retourne un vecteur D tel que $D[i-1]$ est le degré du sommet i
7. `nonOriente(M)` : vérifie qu'une matrice d'adjacence est bien symétrique (et peut donc représenter un graphe non orienté)
8. `kuratowski(n)` : retourne le vecteur des listes d'adjacences représentant le n-ième graphe de Kuratowski

N'oubliez pas de tester vos fonctions !

1.2. Conversions entre représentations

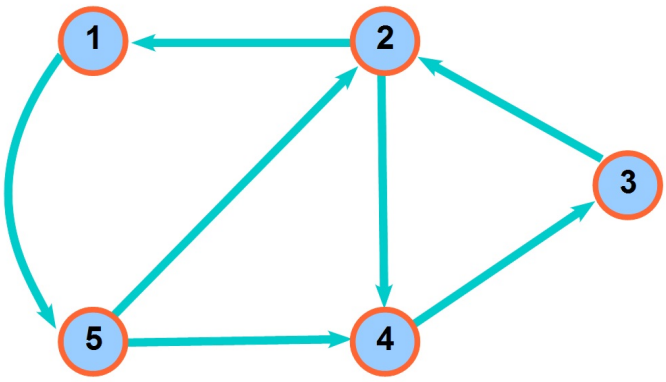
Écrire les fonctions de conversions suivantes :

1. `listeToMatrice(G)` : retourne la matrice d'adjacence représentant G donné par listes d'adjacences
2. `areteToListe(n,L)` : retourne le vecteur des listes d'adjacences représentant un graphe donné par sa liste L d'arêtes
3. `matToListe(M)` : retourne le vecteur des listes d'adjacences représentant un graphe donné par sa matrice d'adjacence

Tester vos fonctions.

2. Graphes orientés

Considérons le graphe G_1 suivant :



Le graphe G_1 est représenté par

- sa liste d'arêtes : $L = [[1,5],[2,1],[2,4],[3,2],[4,3],[5,2],[5,4]]$
- sa liste d'adjacences : $G = [[],[5],[1,4],[2],[3],[2,4]]$
- et sa matrice d'adjacences :

$$M = \begin{pmatrix} 0,0,0,0,1 \\ 1,0,0,1,0 \\ 0,1,0,0,0 \\ 0,0,1,0,0 \\ 0,1,0,1,0 \end{pmatrix}$$

2.1. Premières fonctions

En vous inspirant du paragraphe précédent, écrire les fonctions suivantes (G désigne cette fois un graphe simple orienté) :

1. `nbSommets(G)` : retourne le nombre de sommets du grape G
2. `nbArcs(G)` : retourne le nombre d'arcs du graphe
3. `ajoutArc(G,i,j)` : ajoute l'arc (i,j) au graphe G
4. `enleveArc(G,i,j)` : enlève l'arc (i,j) du graphe G (s'il existe !)
5. `degS(G,i)` : retourne le degré sortant du sommet i
6. `degreS(G)` : retourne un vecteur D tel que $D[i-1]$ soit le degré sortant du sommet i
7. `degE(G,i)` : retourne le degré entrant du sommet i
8. `degreE(G)` : retourne un vecteur D tel que $D[i-1]$ soit le degré entrant du sommet i

Tester vos fonctions.

2.2. Conversions entre représentations

G désigne toujours un graphe simple orienté. En vous inspirant du paragraphe §1.2, écrivez les fonctions suivantes :

1. `listeToMatrices(G)` : retourne la matrice d'adjacence représentant G donné par listes d'adjacences
2. `arcsToListe(n,L)` : retourne le vecteur des listes d'adjacences représentant un graphe donné par sa liste L d'arcs
3. `matToListe(M)` : retourne le vecteur des listes d'adjacences représentant un graphe donné par sa matrice d'adjacence M