TD – Graphes

1ère partie

Introduction et objectifs

Dans cette première partie du TD consacré aux graphes, l'objectif principal est de se familiariser avec les concepts généraux relatifs aux **graphes simples non orientés** (ainsi, dans le sujet de ce TD, le mot « graphe » désignera systématiquement un tel graphe) : degré, chaînes, connexité, ...

On utilisera les tableaux (objets `array`) de la bibliothèque `numpy`.

Représentation des graphes

Exercice N°1 – Construction de la matrice d'adjacence

Construire une fonction `fMatAdj` qui renvoie la matrice d'adjacence `MatAdj` d'un graphe $\mathscr{G}$.

La fonction `fMatAdj` recevra en argument :

- Le nombre de sommets du graphe, ceux-ci étant numérotés de `0` à `n-1`.
- Une liste `L` dont chaque élément sera une liste de deux entiers correspondant à deux sommets adjacents.

  Par exemple, avec un graphe de 6 sommets :

  ```
  L = [[3,5],[1,5],[2,3],[2,1]]
  ```

On note que le 1er et le 5ème sommet n'apparaissent pas dans `L` : il s'agit de sommets isolés.

Avec notre exemple, l'appel à `fMatAdj` sera : `fMatAdj(6,L)`.

Pendant la construction de la matrice d'adjacence, la fonction `fMatAdj` vérifiera la validité de la liste `L` (le cas échéant, elle renverra un message d'erreur du type « `Graphe non valide` »). En effet, celle-ci ne doit comporter :

- Ni élément de la forme `[i,i]` (pas de boucle) ;
- Ni doublon : si `[i,j]` est un élément de `L` alors on ne doit pas avoir `[j,i]` dans `L`.

Exercice N°2 – Degrés des sommets

On suppose que l'on dispose de l'ordre `n` d'un graphe $\mathscr{G}$ et de sa matrice d'adjacence `MatAdj`.

On demande d'écrire une fonction Python Python déterminant le degré de chaque sommet.

La fonction `fDegSom` recevra comme arguments l'entier `n` et la matrice `MatAdj`. Elle renverra une liste `D` comportant `n` éléments, l'élément `D[i]` correspondant simplement au degré du sommet `i`.

Exercice N°3 – Coloration

On cherche à attribuer une couleur à chaque sommet d'un graphe $\mathscr{G}$ de telle sorte que deux sommets adjacents ne soient pas de la même couleur. Le nombre de couleurs minimum requis est appelé « **nombre chromatique** » du graphe.

On demande d'écrire une fonction Python `WP` implémentant l'algorithme suivant (algorithme de Welsh et Powell.) :

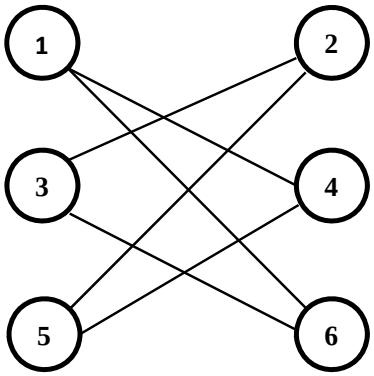
1ère étape.

Déterminer les degrés des sommets et les classer dans un ordre décroissant. On devra obtenir une liste `DD` dont les éléments sont des listes à deux éléments de la forme `[i,d(i)]` où `i` est le numéro du sommet et `d(i)` son degré.

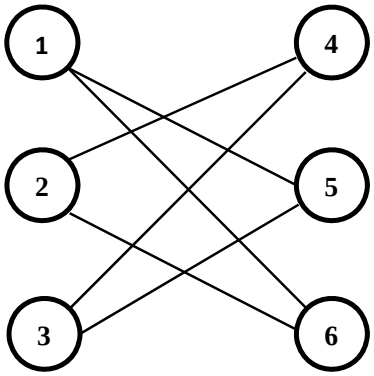
2ème étape.

1) Attribuer au premier sommet de la liste `DD` une couleur (dans votre fonction, il s'agira de l'entier 0).
2) Attribuer cette même couleur aux sommets de `DD` non encore coloriés et qui ne sont pas adjacents aux sommets coloriés de cette couleur.
3) Dans la liste `DD`, considérer alors le premier sommet S non colorié et lui attribuer une nouvelle couleur.
4) Attribuer cette même couleur aux sommets de `DD` non encore coloriés et qui ne sont pas adjacents aux sommets coloriés de cette couleur.
5) Répéter les étapes 3) et 4) tant que la liste `DD` contient au moins un sommet non encore colorié.

On testera la fonction sur le graphe suivant :



Supposons maintenant que les sommets du graphe soient numérotés comme suit :



Que renvoie votre fonction ? Que peut-on en conclure quant à l'algorithme de Welsh-Powell ?

Exercice N°4 – Chaînes

Comme vu en cours, la matrice d'adjacence A d'un graphe $\mathscr{G}$ d'ordre *n* permet, via sa puissance *m*ième, de déterminer directement le nombre de chaînes de longueur *m* reliant les sommets S_i et S_j.

Ecrire une fonction Python <u>récursive</u> `chain` recevant en argument :

- Le numéro `s1` (entre 0 et $n-1$) du premier sommet.
- Le numéro `s2` du deuxième sommet (on peut avoir `s1=s2` lorsque l'on a $m \geq 2$).
- La longueur `LCinit` des chaînes cherchées.

Les trois valeurs initiales de ces variables seront demandées à l'utilisateur dans le programme principal.

La fonction `chain`, au fur et à mesure des appels récursifs, modifiera <u>sur place</u> une liste `C` contenant les sommets de la chaîne en cours de construction et affichera `C` chaque fois que le cas de base favorable sera rencontré.

La liste `C` contiendra initialement les `LCinit+1` éléments suivants :

```
C = [s1,s1,s1,…,s1,s2]
```

On testera le programme avec le graphe d'ordre 5 défini comme suit :

```
L = [[0,2],[2,3],[3,1],[0,4],[2,4],[1,2]]
```

Enfin, avant l'affichage proprement dit de la(des) chaîne(s), le programme affichera le nombre total de chaînes cherchées grâce à la puissance appropriée de la matrice d'adjacence MA du graphe. Cette puissance pourra être calculée à l'aide de la fonction `linalg.matrix_power` de la bibliothèque `numpy`.

Par exemple, avec mon programme et le graphe ci-dessus, on obtient :

```
Veuillez saisir le numéro du premier sommet : 3
Veuillez saisir le numéro du deuxième sommet : 4
Veuillez saisir la longueur des chaînes entre ces sommets : 4

Les 7 chaînes sont :

[3, 1, 2, 0, 4]
[3, 1, 3, 2, 4]
[3, 2, 0, 2, 4]
[3, 2, 1, 2, 4]
[3, 2, 3, 2, 4]
[3, 2, 4, 0, 4]
[3, 2, 4, 2, 4]
```

Exercice N°5 –Connexité

Soit $\mathscr{G}$ un graphe d'ordre *n* et de matrice d'adjacence A.

Expliquer pourquoi la matrice $C = \sum_{k=0}^{n-1} A^k$ permet de savoir si $\mathscr{G}$ est ou non connexe.

Programmer une fonction Python `fGraphConnex` qui reçoit en argument la matrice A et retourne un booléen selon que le graphe $\mathscr{G}$ est ou non connexe.

Exercice N°6 –Distance – diamètre

La distance d_{ij} entre deux sommets S_i et S_j d'un graphe connexe $\mathscr{G}$ de matrice d'adjacence $A = (a_{ij})$ est donnée par :

$$d_{ij} = \min_{k \in \mathbb{N}^*} \left\{ k \,/\, a_{ij}^{(k)} \neq 0 \right\}$$

où $a_{ij}^{(k)}$ désigne le coefficient d'indices *i* et *j* de A^k, puissance *k*ième de la matrice A.
On peut considérer la matrice des distances entre les sommets : $D = d_{ij}$. On note qu'il s'agit encore d'une matrice réelle symétrique et qu'elle est du même ordre que A.

Le diamètre d'un graphe $\mathscr{G}$ connexe est la plus grande distance entre sommets du graphe :

$$\delta(\mathscr{G}) = \max_{(i,j) \in [\![1,n]\!]^2} \left\{ d_{ij} \right\} = \max_{(i,j) \in [\![1,n]\!]^2} \left\{ \min_{k \in \mathbb{N}^*} \left\{ k \,/\, a_{ij}^{(k)} \neq 0 \right\} \right\}$$

1. Ecrire une fonction Python `fGraphDist` qui recevra en argument les numéros `s1` et `s2` de deux sommets d'un graphe connexe $\mathscr{G}$ et renverra leur distance.
2. Ecrire une fonction Python `fGraphDiam` qui recevra en argument la matrice d'adjacence d'un graphe connexe $\mathscr{G}$ (on ne demande pas que la fonction valide que le graphe est effectivement connexe) et renverra le diamètre du graphe.
   La fonction devra construire la matrice des distances entre les sommets.