Sujets de concours et théorie des graphes en CPGE ECG

Les premiers concours spécifiques au nouveau programme ont eu lieu récemment, et certains concepteurs ne se sont pas privés : les graphes apparaissent dans le sujet Ecricome et dans le sujet EDHEC.

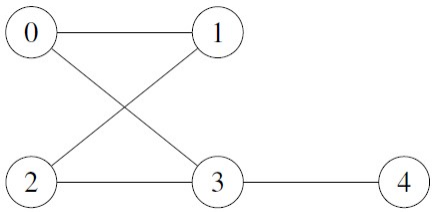
En réalité, dans le sujet EDHEC (page 2), il s'agit simplement de quelques questions pour aborder ensuite l'algèbre linéaire, avec l'étude des valeurs propres de la matrice laplacienne du graphe (matrice définie dans le sujet). En ce qui concerne la théorie des graphes, il est seulement demandé aux candidats de donner la matrice d'adjacence, la matrice laplacienne, et le nombre de chemins reliant deux sommets.

La partie du sujet Ecricome concernant les graphes débute par quelques questions de cours, que nous ne corrigeons pas. Une question concernant le nombre de chemins entre deux sommets permet de faire de lien avec une partie précédente d'algèbre linéaire, où il s'agissait entre autres de calculer la puissance de la matrice d'adjacence. A la page 6, nous nous concentrons sur les questions d'informatique.

EDHEC 2023

On suppose, et c'est valable pour toute l'épreuve, que les librairies `numpy`, `numpy.random` *et* `numpy.linalg` *de* `Python` *sont importées avec les commandes respectives* `import numpy as np`, `import numpy.random as rd` *et* `import numpy.linalg as al`.

Exercice 1 On considère le graphe G suivant et on note A la matrice d'adjacence de G.



1. Déterminer la matrice A en expliquant sa construction.
2. (a) Par lecture du graphe, donner (en listant leurs sommets) les chaînes de longueur 3 reliant les sommets 2 et 3. Combien y en a-t-il ?

   (b) On considère la fonction `Python` suivante :

   ```
   def f(M,k):
       N=al.matrix_power(M,k)
       return N
   ```

   On suppose que l'on a saisi la matrice A et on considère les instructions :

   ```
   B=f(A,...)
   n=B[...]
   print(n)
   ```

   Compléter ces instructions pour qu'elles permettent l'affichage du nombre trouvé à la question 2) a.

On note D la matrice diagonale, appelée matrice des degrés de G, dont l'élément diagonal situé à la ligne i et à la colonne i est le degré du sommet numéro i (ceci étant valable pour tout $i \in [\![1, 5]\!]$).
On définit également la matrice L, appelée matrice laplacienne de G, en posant $L = D - A$.
On note λ_1, λ_2, λ_3, λ_4, λ_5 les valeurs propres non nécessairement distinctes de L et on suppose $\lambda_1 \leq \lambda_2 \leq \lambda_3 \leq \lambda_4 \leq \lambda_5$.

3. (a) Déterminer la matrice D.

   (b) Vérifier que l'on a $L = \begin{pmatrix} 2 & -1 & 0 & -1 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ -1 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix}$

   (c) Pourquoi la matrice L est-elle diagonalisable ?
4. On se propose dans cette question de montrer que les valeurs propres de L sont positives ou nulles et que $\lambda_1 = 0$.

   Soit $X = \begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix}$ et $U = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$

(a) On identifie une matrice de $\mathscr{M}_1(\mathbb{R})$ à un réel.
À quel ensemble appartient la quantité ${}^tXLX$?

(b) Exprimer ${}^tXLX$ en fonction de a, b, c, d et e puis monter que l'on a :

$$ {}^tXLX = (a-b)^2 + (b-c)^2 + (c-d)^2 + (d-a)^2 + (e-d)^2$$

(c) On suppose que X est un vecteur propre de L associé à une certaine valeur propre λ.
Déterminer LX puis ${}^tXLX$ en fonction de λ, a, b, c, d et e.
En déduire que les valeurs propres de L sont positives ou nulles.

(d) Déterminer LU et en déduire que $\lambda_1 = 0$.

5. (a) À l'aide de la question 3) b, montrer l'équivalence :

$$LX = 0 \iff X \in \text{Vect}(U)$$

(b) Conclure que λ_2, λ_3, λ_4, et λ_5 sont des réels strictement positifs.

Ecricome 2023

Exercice 3 - Partie 2.
Soient p un entier naturel non nul et G un graphe non pondéré orienté à p sommets. On note s_0, $s_1, \ldots, s_{p-1}$ les sommets de G.

1. (a) Rappeler la définition de la matrice d'adjacence du graphe G.

   (b) Soient n un entier naturel non nul, i un entier de $[\![1, p]\!]$ et j un entier de $[\![1, p]\!]$.
   Rappeler sans justification l'interprétation du coefficient situé à la ligne i et à la colonne j dans la matrice M^n, où M est la matrice d'adjacence du graphe G.

2. Dans cette question uniquement, on suppose que $p = 4$ et que la matrice d'adjacence du graphe G est la matrice $A = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$ étudiée dans la partie 1.

   (a) Représenter les sommets et les arêtes du graphe G sous forme d'un diagramme.

   (b) Le graphe G est-il connexe ? Justifier votre réponse.

   (c) Soit n un entier naturel non nul.
   Déterminer le nombre de chemins de longueur n menant du sommet s_3 au sommet s_0.

3. Dans cette question et les suivantes, on revient au cas général décrit au début de la partie 2.
   Soit s un sommet de G. On dit que le sommet t est un voisin de s quand $s \neq t$ et (s, t) est une arête du graphe.
   Comme le graphe est orienté, si t est un voisin de s, alors s n'est pas forcément un voisin de t.
   On appelle liste d'adjacence du graphe G, une liste de p sous-listes telle que, pour tout entier k de $[\![0, p-1]\!]$, la sous-liste située à la position k contient tous les numéros des sommets voisins de s_k.
   Par exemple, la liste d'adjacence du graphe étudié à la question 2 est :

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

   Écrire une fonction en langage Python, nommée `matrice_vers_ligne`, prenant en entrée la matrice d'adjacence `A` d'un graphe G (définie sous forme de listes de listes) et renvoyant la liste d'adjacence de G.

4. On cherche à écrire une fonction en langage Python permettant d'obtenir la longueur du plus court chemin menant d'un sommet de départ s_i à chaque sommet du graphe G.
   On souhaite pour cela appliquer un algorithme faisant intervenir les variables suivantes :

   — Une liste `distances` à p éléments, où l'élément situé à la position k sera égal, à la fin de l'algorithme, à la longueur du plus court chemin menant du sommet de départ s_i au sommet s_k.

   — Une liste `a_explorer` contenant tous les sommets restant à traiter.

   — Une liste `marques` contenant tous les sommets déjà traités.

   Nous donnons ci-dessous la description de l'algorithme :

   — Initialisation des trois listes décrites ci-dessus :

   — Initialement, chaque élément de la liste `distances` est égal à p, à l'exception du sommet s_i, auquel on affecte la distance 0.

   — La liste `marques` ne contient initialement que le numéro du sommet de départ s_i.

   — La liste `a_explorer` ne contient initialement que le numéro du sommet de départ s_i.

   — Tant que la liste `a_explorer` n'est pas vide, on répète les opérations suivantes :

   — Nommer `s` le premier sommet de la liste `a_explorer`, et le retirer de cette liste.

   — Pour chaque voisin `v` du sommet `s` : si `v` n'est pas dans la liste `marques`, on l'ajoute à la fin de la liste `a_explorer`, et on lui affecte une distance égale à `distances[s]+1`.

(a) On considère le graphe orienté G étudié à la question 2.
Donner la valeur de la liste `distances` à l'issue de l'exécution de l'algorithme décrit ci-dessus, lorsqu'on l'applique au graphe G en choisissant s_1 comme sommet de départ.

(b) Recopier et compléter la fonction suivante, prenant en entrée la liste d'adjacence `L` du graphe G et le numéro `i0` du sommet de départ s_i, et renvoyant la liste `distances` après exécution de l'algorithme décrit ci-dessus.

```
def parcours(L, i0):
    p = len(L)
    distances = __________
    distances[i0] = 0
    a_explorer  = __________
    marques = __________
    while __________:
        s = _________
        _____________
        for v in __________:
            if v not in marques :
                marques.append(v)
                __________
                __________
    return distances
```

(c) Modifier la fonction précédente pour qu'elle renvoie la liste de tous les sommets s pour lesquels il existe un chemin menant du sommet de départ s_i au sommet s.

Ecricome 2023 - Un corrigé partiel

3. L'idée est d'initialiser une liste `L` à la liste vide, de parcourir chaque ligne de la matrice d'adjacence, et d'ajouter à `L` une liste contenant les indices des coefficients non nuls de la ligne parcourue.

```
def matrice_vers_ligne(A):
    L = []
    for e in A: #e est une liste
        voisins = []
        for i in range(len(e)):
            if e[i]==1:
                voisins.append(i)
        L.append(voisins)
    return L
```

4. (a) A l'issue de l'exécution de l'algorithme, la liste `distances` doit être `[1,0,1,2]`. En effet, (s_1, s_0) et (s_1, s_2) sont deux arêtes du graphe, et (s_1, s_0, s_3) est un chemin du graphe (et (s_0, s_3) n'est pas une arête).

   (b) Voici une solution :

```
def parcours(L, i0):
    p = len(L)
    distances = p*[p]
    distances[i0] = 0
    # Notons que (i0-1)*[p]+[0]+(p-i0)*[p]  conviendrait
    a_explorer = [i0]
    marques = [i0]
    while a_explorer!=[]:
        s = a_explorer[0]
        del a_explorer[0] #commande en annexe du sujet
        for v in L[s]:
            if v not in marques :
                marques.append(v)
                a_explorer.append(v)
                distances[v] = distances[s]+1
    return distances
```

   (c) Dans la liste `distances` donnée par l'algorithme, les sommets non accessibles depuis le sommet initial s_i ont une distance égale à p. Il suffit donc de remplacer la dernière ligne du programme précédent par les deux suivantes :

```
sommets = [i for i in range(p) if distances[i]<p]
return sommets
```