L3 – Théorie des graphes

CCE01

-

Jeudi 19 octobre 2023

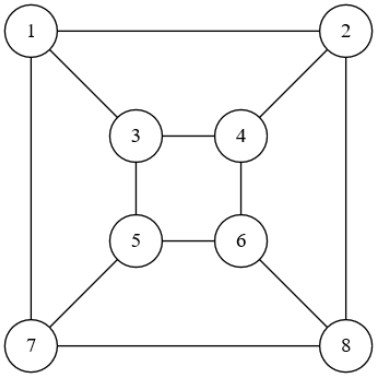
La calculatrice n'est pas autorisée

Durée : 1 h 30

Toutes les réponses sont à donner sur la copie, sauf pour les questions 4 et 5 de l'exercice 7.

Exercice 1.

On considère le graphe non orienté, G_1 suivant :



1) Donner l'ordre et la taille du graphe G_1.

2) Donner une représentation de G_1 sous la forme d'une matrice d'adjacence.

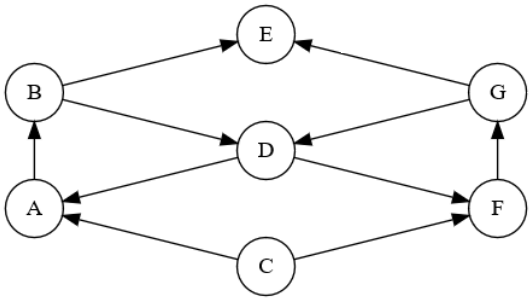
3) Le graphe G_1 est-il complet ? Pourquoi ?

4) Le graphe G_1 admet-il des cycles ? Si oui, en donner un.

5) Que dire des degrés des sommets du graphe G_1 ?

Exercice 2.

On donne la représentation suivante d'un graphe non orienté, G_2



1) Donner une représentation du graphe G_2 sous la forme d'un dictionnaire de listes d'adjacence.

2) Existe-t-il un chemin de B vers C ? de C vers G ? Si oui, expliciter ces chemins.

3) Existe-t-il un sommet à partir duquel tous les sommets du graphe sont accessibles ? Si oui, expliciter, si non, justifier.

Exercice 3.
On considère un graphe G_3 dont la matrice d'adjacence est représentée par le tableau suivant :

	1	2	3	4	5	6	7
1	1	0	1	0	0	1	0
2	0	1	1	0	1	0	1
3	0	1	0	1	0	0	1
4	1	1	1	0	0	0	1
5	0	1	0	0	1	1	0
6	0	0	0	1	0	0	1
7	1	0	1	0	1	0	0

1) Le graphe G_3 est-il orienté ? Justifier.
2) Le graphe G_3 comporte-t-il des boucles ? Justifier.
3) Donner une représentation graphique du graphe G_3.
4) Le graphe G_3 est-il connexe ? Justifier.

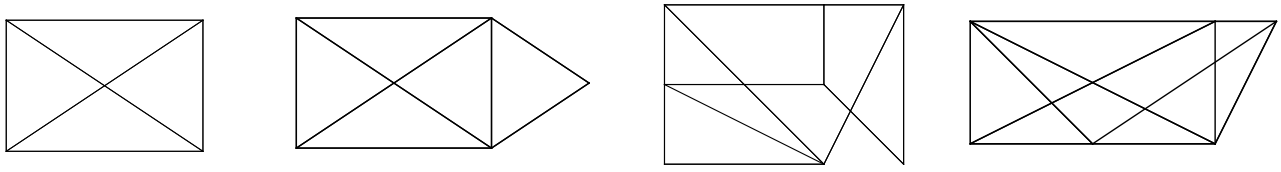
Exercice 4.
On fait les rappels suivants concernant les graphes non orientés :

1. Un graphe $G = (S, A)$ et **biparti** s'il existe une partition de ses sommets en deux sous-ensembles, $\{S_1,\ S_2\}$, tels que $S = S_1 \cup S_2$ et $S_1 \cap S_2 = \emptyset$, telle que chaque arête ait une extrémité dans S_1 et l'autre dans S_2.
2. Un graphe simple est un **graphe sans boucle ni arête multiple**
3. On appelle **graphe biparti complet** entre un ensemble de n sommets et un ensemble de m sommets, le graphe simple tel que chaque sommet du premier ensemble est relié à chaque sommet du deuxième ensemble. On note $K_{n,m}$ ce graphe.

1) Construire le graphe $K_{3,4}$. Compter son nombre d'arêtes.
2) Combien le graphe biparti complet $K_{n,m}$ possède-t-il d'arêtes ? $(m, n \geq 1)$

Exercice 5.
On rappelle qu'un parcours eulérien dans un graphe non orienté est une chaîne simple (*i.e.* ne passant pas deux fois par la même arête) entre deux sommets du graphe, passant par toutes les arêtes du graphe.

1) Donner une condition nécessaire sur les degrés des sommets pour qu'un graphe connexe possède un parcours eulérien.
2) [Application] Est-il possible de tracer les figures suivantes sans lever le crayon (et sans repasser sur un tracé déjà fait) ?



Si oui, reproduire la figure et indiquer clairement les points de départ et d'arrivée du tracé.

Exercice 6.

On considère un graphe dont les sommets sont étiquetés A, B, C, D, E, F, G.

L'ensemble de ses sommets est représenté en machine par la liste

```
LS = ['A', 'B', 'C', 'D', 'E', 'F', 'G']
```

Il est demandé en complément, pour chaque fonction, d'évaluer sa complexité en fonction de l'ordre n et de la taille, m, du graphe.

1) Écrire une fonction Python `numS(L)`, prenant en argument une liste L des noms des sommets d'un graphe et renvoyant un dictionnaire associant à chaque nom de sommet sa position dans la liste L.

On devra par exemple obtenir :

```
>>> numS(LS)
{'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4, 'F': 5, 'G': 6}
```

2) Définir une fonction `complet(n)`, prenant en argument un entier $n \geq 1$ et renvoyant la représentation sous forme de liste de liste de la matrice d'adjacence du graphe simple (sans boucles ni arêtes multiples) non orienté complet d'ordre n.

On devra par exemple obtenir :

```
>>> complet(3)
[[0, 1, 1], [1, 0, 1], [1, 1, 0]]
```

3) Définir une fonction `mat_to_dic(L, M)`, prenant en argument une liste L, des noms des sommets d'un graphe orienté G et une liste de listes `M`, représentant la matrice d'adjacence du graphe G (les sommets du graphe y étant représentés par leur position dans L) et renvoyant un dictionnaire associant à chaque nom de sommet la liste (éventuellement vide) de ses successeurs.

Si besoin, on pourra commencer, en l'indiquant clairement, définir une fonction `mat_to_dic`(L, M) répondant à la question uniquement dans le cas où les noms des n sommets du graphe sont leur numéros de position, de 0 à $n-1$, dans la liste L.

On devra par exemple obtenir :

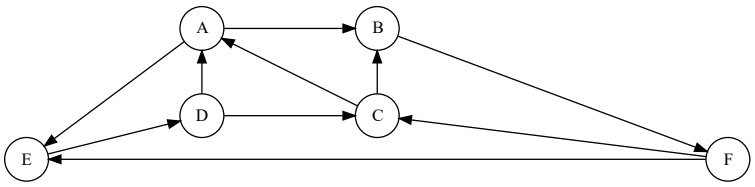
```
>>> mat_to_dic(['A', 'B', 'C'], [[1, 0, 0], [0, 1, 1], [1, 0, 1]])
{'A' : ['A'],  'B' : ['B', 'C'],  'C' : ['A', 'C']}
```

Exercice 7. Parcours de graphe – implémentation itérative

1. Rappeler le fonctionnement de la structure de données de pile. Comment peut-on implémenter, au plus simple, en recourant éventuellement à un module, une pile en Python ?

2. Rappeler le fonctionnement de la structure de données de file. Comment peut-on implémenter, au plus simple, en recourant éventuellement à un module, une file en Python ?

On considère le graphe orienté G suivant :



On considère que le graphe G est représenté en machine par un dictionnaire associant chaque sommet à la liste de ses successeurs, et que ces listes de successeurs sont ordonnées dans l'ordre alphabétique.

Au début d'un parcours itératif, avant toute itération :

- aucun sommet n'est en cours de traitement
- la pile ou la file contient un unique sommet (sommet de départ), qui est marqués comme atteint.

À chaque itération, on traite un sommet sorti de la pile ou de la file. Le traitement d'un sommet consiste à :

- explorer son voisinage
- marquer comme atteint chaque sommet découvert et non encore marqué comme atteint
- empiler ou enfiler les sommets nouvellement découverts

Pour les deux questions suivantes, **on supposera que les successeurs d'un sommet qui le doivent, sont ajoutés à la pile ou à la file, les uns après les autres, dans l'ordre alphabétique**.
Les piles ou les files seront représentées en ligne, avec <u>le sommet de la pile ou la sortie de la file à droite</u>.
La première ligne de chaque tableau décrit la situation avant toute itération.
Chacune des lignes suivantes correspondent à une itération de l'algorithme, et donne le sommet traité à cette itération et décrivent l'état de la pile ou de la file et les sommets marqués à la fin du traitement de ce sommet (fin de l'itération).

3. Combien de lignes comportera chaque tableau, hormis la ligne déjà complétée (état avant toute itération) ? Justifier.

4. Compléter sur votre copie le tableau suivant, correspondant à un **parcours en largeur** du graphe G à partir du sommet E.

Sommet en cours de traitement	File/Pile des sommets à traiter	Sommets marqués
aucun	E	E
E	…	…
…	…	…

5. Compléter sur votre copie le tableau suivant, correspondant à un **parcours en profondeur** du graphe G à partir du sommet D.

Sommet en cours de traitement	File/Pile des sommets à traiter	Sommets marqués
aucun	E	E
E	…	…
…	…	…

On considère la fonction suivante, dans laquelle `dA` représente un dictionnaire de listes d'adjacence décrivant un graphe, et `s0` le sommet de départ d'un parcours

```
1.  def parcours(dA, s0):
2.      L = []
3.      atteints = {}
4.      for sommet in dA.keys():
5.          atteints[sommet] = False
6.      X = [s0]
7.      atteints[s0] = True
8.      while len(X) > 0:
9.          sommet = X.pop()
10.         for succ in dA[sommet]:
11.             if not atteints[succ]:
12.                 X.append(succ)
13.                 atteints[succ] = True
14.     return L
```

6. Évaluer la complexité de la fonction en fonction de l'ordre n et de la taille m du graphe.
7. La fonction implémente-t-elle un parcours en largeur ou en profondeur ? Justifier.
8. En l'état, que renvoie un appel à la fonction parcours si on l'applique à une représentation du graphe G donné en exemple avec pour sommet de départ E ?
9. Quelle modification apporter au code de la fonction pour qu'elle renvoie la liste des sommets présents dans la colonne de gauche du tableau complété précédent correspond au même type de parcours ?
10. Quelles modifications apporter à la fonction pour quelle implémente l'autre type de parcours ?
11. Que dire de la liste L renvoyée si le graphe en entrée est un graphe non orienté non connexe ?
12. Définir une fonction `compo(dA)` renvoyant, dans le cas où le graphe en entrée est non orienté, une liste de listes dont chaque sous-liste est constituée des sommets correspondant à une composante connexe du graphe.

Exercice 8. Parcours de graphe – implémentation récursive

On donne l'implémentation récursive d'un parcours en profondeur d'un graphe :

```
def parcours_prof_rec(dA, s0):
    marquage = {}
    for sommet in dA.keys():
        marquage[sommet] = False
    ##
    def explorer(dA, s):
        marquage[s] = True
        for voisin in dA[s]:
            if not marquage[voisin]:
                explorer(dA, voisin)
    ##
    explorer(dA, s0)
```

Pour le graphe donné en exemple à l'**exercice 7**, représenté par un dictionnaire `dA` des listes d'adjacence de ses sommets, avec les mêmes hypothèses sur l'ordre au sein des listes de successeurs, on exécute l'appel suivant, `parcours_prof_rec (dA, 'D')`.

1. Donner la liste des appels à la fonction `explorer`, dans l'ordre chronologique.
   On pourra abréger un appel `explorer(dA, 'B')` en `exp('B')`.
2. Dans quel ordre les appels se terminent-ils ?

ANNEXE

Nom/prénom :

Exercice 7

2. Compléter sur votre copie le tableau suivant, correspondant à un **parcours en largeur** du graphe G à partir du sommet E.

Sommet en cours de traitement	File/Pile des sommets à traiter	Sommets marqués
aucun	E	E
E	…	…
…	…	…

3. Compléter sur votre copie le tableau suivant, correspondant à un **parcours en profondeur** du graphe G à partir du sommet D.

Sommet en cours de traitement	File/Pile des sommets à traiter	Sommets marqués
aucun	E	E
E	…	…
…	…	…