Université de Carthage Institut Préparatoire aux Etudes d'Ingénieurs de Nabeul Département Mathématique	IPEIN المعهد التحضيري للدراسات الهندسية بنابل Institut Préparatoire aux Etudes d'Ingénieurs de Nabeul	Année universitaire : 2023/2024 Filière : Informatique Niveau d'étude : 2 année Semestre : 1 Nombre de pages : 4 Date : 16/12/2023 Durée : 2h

EXAMEN

DOCUMENT NON AUTORISES

Problème POO

Partie 1 (5 pts)

Une matrice à *m* lignes et *n* colonnes est un tableau rectangulaire de $m \times n$ nombres, rangés ligne par ligne. La matrice contient *m* lignes, et dans chaque ligne *n* éléments.

La disposition générale des coefficients d'une matrice *A* de taille (*m*,*n*) est donc la suivante

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}$$

Les coefficients $a_{i,j}$ avec $i = j$ sont dits diagonaux, ceux avec $i \neq j$ sont dits extradiagonaux.

On représente par la suite une matrice par une liste de listes telles que A est une liste de ***m*** sous listes dont chacune contient ***n*** éléments. Par exemple, on représente ci-dessous une matrice *A*, à coefficients entiers, et de dimension (3,4) :

$$A = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \end{pmatrix}$$

⇨ A= **[[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11]]**

Travail demandé

On se propose de créer une classe **Matrice** permettant de représenter la matrice décrite ci-dessus.Un objet de type **Matrice** est caractérisé par les attributs suivant :

- n : nombre de ligne de la matrice de type entier
- m : nombre de colonne de la matrice de type entier
- A : liste contenant les coefficients de la matrice initialisée à vide

1. Définir la classe **Matrice** ainsi que son constructeur.

2. Définir une méthode **Saisie_mat** permettant de remplir une **Matrice** par des coeffecients entiers positifs compris entre deux valeurs entières **deb** et **fin** données.
3. Definir une méthode **__str__** permettant d'afficher une **Matrice** sous la forme :

« **0 1 2 3**
4 5 6 7
8 9 10 11 »

4. Definir une méthode **__add__** permettant de retourner la **Matrice** somme de deux objets Matrices données (qu'on suppose de même taille).
5. Definir une méthode **__mul__** permettant de retourner la **Matrice** produit de deux objets Matrices données (qu'on suppose de tailles compatibles) , la formule de produit est la suivante :

$$AB = \begin{pmatrix} c_{0,0} & \cdots & c_{0,p-1} \\ \vdots & \ddots & \vdots \\ c_{m-1,0} & \cdots & c_{m-1,p-1} \end{pmatrix} \quad \text{avec} \quad c_{ik} = \sum_{j=0}^{n-1} a_{ij} b_{jk}$$

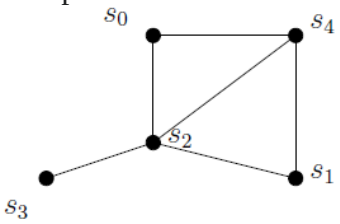
Partie 2 (6 pts)

Dans cette partie nous allons définir une nouvelle structure de données appelée le graphe. Le graphe est un ensemble de sommets interconnectés par des liaisons, chaque liason est appelée **arête** :

- Si deux sommets sont liés alors arête=1
- Sinon arête=0

Un graphe est défini par une **Matrice** appelée matrice d'adjacence. Pour un graphe de n sommets sa matrice d'adjacence est de taille ***(n,n)*** avec chaque coeffecients a_{ij} correspond à la valeur 1 si sommet i et sommet j sont liés, 0 sinon.

Exemple :




```
A = [ [0,0,1,0,1],
      [0,0,1,0,1],
      [1,1,0,1,1],
      [0,0,1,0,0],
      [1,1,1,0,0] ]
```

Travail demandé

On se propose de construire une classe **Graphe** afin de représenter cette structure de données. Cette classe admet comme attributs :

- n : un entier représentant le nombre de sommets d'un objet Graphe
- M : une Matrice représentant la matrice d'adjacence d'un objet Graphe

1. Créer la classe **Graphe** ainsi que son constructeur définissants l'attribut ***n*** et l'attribut ***M*** initialisé à un objet Matrice vide .
2. Définir une méthode **Saisie_Gr** permettant la saisie de la matrice d'adjacence d'un objet Graphe.
3. Définir la méthode **__getitem__** qui , pour un numéro de sommet ***num*** donnée, permet de retourner une liste de numéros des sommets voisins (liés) à ***num.***
4. Définir la méthode **Degre** permettant de retourner le nombre de sommets voisins d'un sommet numéro ***num*** donnée.

5. Définir la méthode **Nbr_arete** permettant de retourner le nombre d'arêtes d'un objet Graphe en appliquant la formule suivante :

$$nombre\ d'aretes = \frac{\sum_{i=1}^{n} degre(S¿¿i)}{2} ¿$$

6. Définir la méthode **__setitem__** qui permet de rajouter l'arête entre deux sommets ***s*** et ***t*** données d'un objet Graphe.

Partie 3 (9 pts)

Les parcours d'un graphe

Parcourir un graphe consiste à lister tous ses sommets une seule fois.

On peut parcourir un graphe :

- soit pour lister simplement tous ses sommets;
- soit pour trouver un chemin pour relier un sommet à un autre.

Parcours en largeur

L'idée du **parcours en largeur** repose sur l'utilisation d'une **file** de la manière suivante :

Voici la traduction en pseudo-code du parcours en largeur d'un graphe décrit ci-dessus.

```
F est une file vide
On enfile le sommet de départ
On initialise une liste vide
Tant que F n'est pas vide
        S = Tête de F
        On enfile les voisins de S qui ne sont pas déjà présents dans la file et qui n'ont pas été déjà défilés
        On défile F
        On ajoute l'élément défilé à la liste
Fin Tant Que
```

Parcours en profondeur

L'idée du **parcours en profondeur** repose sur l'utilisation d'une **pile** de la manière suivante :

Voici la traduction en pseudo-code du parcours en profondeur d'un graphe.

P est une pile vide

On empile le sommet de départ

On initialise une liste vide

Tant que P n'est pas vide

S = dépile(P)

On ajoute S à la liste

On empile les voisins de P qui ne sont pas déjà présents dans la pile et qui n'ont pas été déjà dépilés

Fin Tant Que

Travail demandé

1. Créer la classe **Pile** qui admet :
   - ✶ un attribut une liste L intialement vide
   - ✶ les méthodes suivantes :
     - ➢ **Vide_P** : qui permet de tester si un objet Pile vide ou non.
     - ➢ **Sommet** : qui permet d'afficher le sommet de l'objet Pile.
     - ➢ **Empiler** :qui permet d'ajouter un élément x à l'objet pile.
     - ➢ **Depiler** : qui permet de retirer un élément x de l'objet pile.
     - ➢ **__contains__** : qui permet de tester l'existence d'un élément x donnée dans un objet Pile.
2. Créer la classe **File** qui admet :
   - ✶ un attribut une liste L intialement vide
   - ✶ les méthodes suivantes :
     - ➢ **Vide_F** : qui permet de tester si un objet File vide ou non.
     - ➢ **Sommet** : qui permet d'afficher le sommet de l'objet File.
     - ➢ **Emfiler** :qui permet d'ajouter un élément à l'objet File.
     - ➢ **Defiler** : qui permet de retirer un élément de l'objet File.
     - ➢ **__contains__** : qui permet de tester l'existence d'un élément x donnée dans un objet File.
3. Ecrire la fonction python **Parcours_larg** qui permet de retourner une liste nommée **listes_sommets** contenant les différents numéros de sommets rencontrés dans un parcours en largeur d'un graphe **G** à partir d'un sommet **numS** donnée.
   La fonction doit retourner None en cas de non-existence du sommet de départ
   NB : il faut utiliser un objet de la classe File.
4. Ecrire la fonction python **Parcours_prof** qui permet de retourner une liste nommée **listes_sommets** contenant les différents numéros de sommets rencontrés dans un parcours en largeur d'un graphe **G** à partir d'un sommet **numS** donnée.
   La fonction doit retourner None en cas de non-existence du sommet de départ
   NB : il faut utiliser un objet de la classe Pile.