Vers le bac : les graphes

Exercice 1

Dans tout cet exercice, on modélise un groupe de personnes à l'aide d'un graphe. Le groupe est constitué de huit personnes (Anas, Emma, Gabriel, Jade, Lou, Milo, Nina et Yanis) qui possèdent entre elles les relations suivantes :

- Gabriel est ami avec Jade, Yanis, Nina et Milo ;
- Jade est amie avec Gabriel, Yanis, Emma et Lou ;
- Yanis est ami avec Gabriel, Jade, Emma, Nina, Milo et Anas ;
- Emma est amie avec Jade, Yanis et Nina ;
- Nina est amie avec Gabriel, Yanis et Emma ;
- Milo est ami avec Gabriel, Yanis et Anas ;
- Anas est ami avec Yanis et Milo ;
- Lou est amie avec Jade.

Partie A : Matrice d'adjacence

On choisit de représenter cette situation par un graphe dont les sommets sont les personnes et les arêtes représentent les liens d'amitié.

1. Dessiner sur votre copie ce graphe en représentant chaque personne par la première lettre de son prénom entourée d'un cercle et où un lien d'amitié est représenté par un trait entre deux personnes.

Une matrice d'adjacence est un tableau à deux entrées dans lequel on trouve en lignes et en colonnes les sommets du graphe.

Un lien d'amitié sera représenté par la valeur 1 à l'intersection de la ligne et de la colonne qui représentent les deux amis alors que l'absence de lien d'amitié sera représentée par un 0.

2. Recopier et compléter l'implémentation de la déclaration de la matrice d'adjacence du graphe.

```
# sommets :      G, J, Y, E, N, M, A, L
matrice_adj = [[0, 1, 1, 0, 1, 1, 0, 0], # G
               [........................], # J
               [........................], # Y
               [........................], # E
               [........................], # N
               [........................], # M
               [........................], # A
               [........................]] # L
```

On dispose de la liste suivante qui identifie les sommets du graphe :

```
sommets = ['G','J','Y','E','N','M','A','L']
```

On dispose d'une fonction `position(l,s)` qui prend en paramètres une liste de sommets `l` et un nom de sommet `s` et qui renvoie la position du sommet `s` dans la liste `l` s'il est présent et `None` sinon.

3. Indiquer quel seront les retours de l'exécution des instructions suivantes :

```
>>> position(sommets,'G')
>>> position(sommets,'Z')
```

4. Recopier et compléter le code de la fonction `nb_amis(L,m,s)` qui prend en paramètres une liste de noms de sommets `L`, une matrice d'adjacence `m` d'un graphe et un nom de sommet `s` et qui renvoie le nombre d'amis du sommet `s` s'il est présent dans `L` et `None` sinon.

```
1 def nb_amis(L,m,s):
2     pos_s = …
3     if pos_s == None:
4         return ...
5     amis = 0
6     for i in range(len(m)):
7         amis += ...
8     return ...
```

5. Indiquer quel est le retour de l'exécution de la commande suivante :

```
>>> nb_amis(sommets,matrice_adj,'G')
```

Partie B : Dictionnaire de listes d'adjacence

6. Dans un dictionnaire Python `{c : v}`, indiquer ce que représentent `c` et `v`.

On appelle graphe le dictionnaire de listes d'adjacence associé au graphe des amis. On rappelle que Gabriel est ami avec Jade, Yanis, Nina et Milo.

```
graphe = {'G' : ['J','Y','N','M'],
          'J' : ...
          ...
          }
```

7. Recopier et compléter le dictionnaire de listes d'adjacence `graphe` sur votre copie pour qu'il modélise complètement le groupe d'amis.

8. Écrire le code de la fonction `nb_amis(d,s)` qui prend en paramètres un dictionnaire d'adjacence `d` et un nom de sommet `s` et qui renvoie le nombre d'amis du nom de sommet `s`. On suppose que `s` est bien dans `d`.

Par exemple :

```
>>> nb_amis(graphe,'L')
 1
```

Milo s'est fâché avec Gabriel et Yanis tandis qu'Anas s'est fâché avec Yanis. Le dictionnaire d'adjacence du graphe qui modélise cette nouvelle situation est donné ci-dessous :

```
graphe ={'G' : ['J','N'],
         'J' : ['G','Y','E','L'],
         'Y' : ['J','E','N'],
         'E' : ['J','Y','N'],
         'N' : ['G','Y','E'],
         'M' : ['A'],
         'A' : ['M'],
         'L' : ['J']
         }
```

Pour établir la liste du cercle d'amis d'un sommet, on utilise un parcours en profondeur du graphe à partir de ce sommet. On appelle cercle d'amis de *Nom* toute personne atteignable dans le graphe à partir de *Nom*.

9. Donner la liste du cercle d'amis de Lou.

Un algorithme possible de parcours en profondeur de graphe est donné ci-dessous :

```
visités = liste vide des sommets déjà visités
fonction parcours_en_profondeur(d,s)
   ajouter s à la liste visités
   pour tous les sommets voisins v de s :
      si v n'est pas dans la liste visités :
         parcours_en_profondeur(d,v)
   retourner la liste visités
```

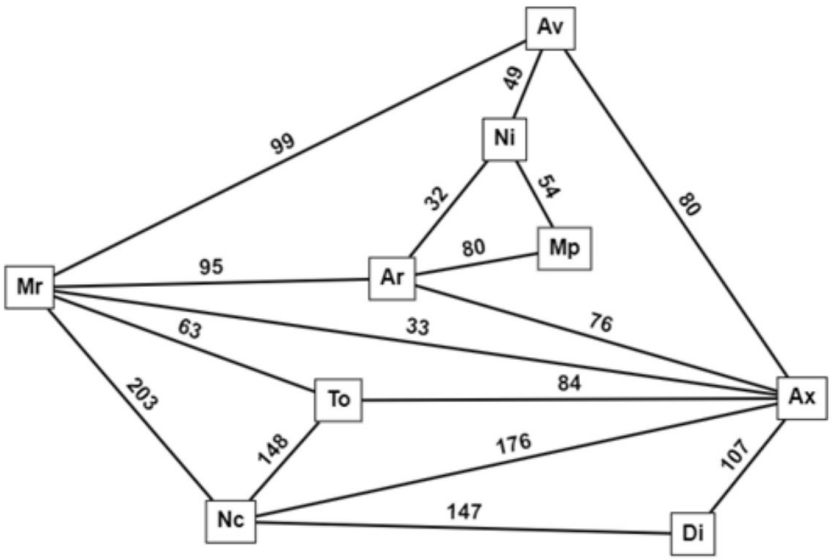
10. Recopier et compléter le code de la fonction `parcours_en_profondeur(d,s)` qui prend en paramètres un dictionnaire d'adjacence `d` et un sommet `s` et qui renvoie la liste des sommets issue du parcours en profondeur du graphe modélisé par `d` à partir du sommet `s`.

```
1  def parcours_en_profondeur(d,s,visites = []):
2     ...
3     for v in d[s]:
4        ...
5           parcours_en_profondeur(d,v)
6     ...
```

Exercice 2

Partie A

Une agence de voyages organise différentes excursions dans une région de France et propose la visite de certaines villes. Ces excursions peuvent être visualisées sur le graphe ci-dessous : les sommets désignent les villes, les arêtes représentent les routes pouvant être empruntées pour relier deux villes et les poids des arêtes représentent des distances, exprimées en kilomètre.

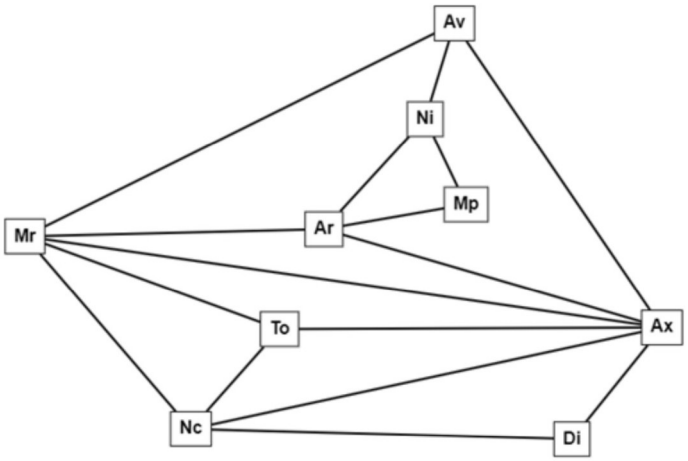


1. Déterminer le plus court chemin allant du sommet Mp au sommet Nc et préciser la longueur, en kilomètres, de ce chemin. Aucune justification n'est attendue.

2. On souhaite toujours se rendre du sommet Mp au sommet Nc mais en visitant le minimum de villes. Déterminer les deux chemins possibles.

Partie B

L'agence souhaite proposer un itinéraire permettant de visiter toutes les villes. On appelle G le graphe non pondéré ci-dessous.



On choisit d'implémenter un graphe par listes d'adjacence, à l'aide d'un dictionnaire, en langage Python, dont :

- les clés sont les sommets du graphe ;
- la valeur associée à une clé est la liste des voisins de ce sommet clé.

Les sommets sont de type `str`.

3. Donner l'implémentation, en langage Python, du graphe de la figure 2. Le dictionnaire obtenu sera stocké dans une variable nommée G . Afin de faciliter la notation manuscrite ainsi que la lisibilité, écrire chaque couple clé/valeur sur une nouvelle ligne.

On considère une file.

4. Indiquer la signification des lettres dans les acronymes LIFO et FIFO.

5. Indiquer l'acronyme utilisé pour désigner la structure de file.

Voici, en langage Python, les opérations pouvant être effectuées sur une telle file :

- `creerFile()` : renvoie une file vide ;
- `estVide(F)` : renvoie `True` si la file `F` est vide et `False` sinon ;
- `enfiler(F, e)` : ajoute l'élément `e` dans la file `F` ;
- `defiler(F)` : renvoie l'élément à la tête de la file `F` en le retirant de la file `F`.

On donne la fonction parcours ci-dessous. Cette fonction prend en paramètres un dictionnaire graphe représentant un graphe sous la forme de listes d'adjacence, et une chaîne de caractères sommet représentant un sommet du graphe.

```
1    def parcours(graphe,sommet):
2        f = creerFile()
3        enfiler(f, sommet)
4        visite = [sommet]
5        while not estVide(f):
6            s = defiler(f)
7            for v in graphe[s]:
8                if not (v in visite):
9                    visite.append(v)
10                   enfiler(f, v)
11       return visite
```

6. Donner le résultat renvoyé par l'appel `parcours(G,'Av')`.

7. Recopier, parmi les deux propositions ci-dessous, celle qui correspond au type de parcours de graphe réalisé par la fonction parcours :
   - **proposition A** : parcours en largeur ;
   - **proposition B** : parcours en profondeur.

Dans la suite de l'exercice, la distance entre deux sommets désignera le nombre d'arêtes séparant ces deux sommets. Ainsi définie, la distance entre les sommets Mp et Nc du graphe de la figure 2 est 3.

8. En modifiant la fonction `parcours`, écrire une fonction `distance` ayant pour paramètres `graphe` , un dictionnaire représentant un graphe sous la forme de listes d'adjacence, et une chaîne de caractères `sommet` représentant un sommet du graphe. Cette fonction renvoie un dictionnaire tel que :
   – les clés sont les sommets du graphe ;
   – la valeur associée à une clé est la distance entre ce sommet clé et le sommet d'origine `sommet`.

9. Donner le résultat renvoyé par l'appel `distance(G,'Av')`.

On considère une pile. Voici, en langage Python, les opérations pouvant être effectuées sur une telle pile :
- `creerPile()` : renvoie une pile vide ;
- `estVide(P)` : renvoie `True` si la pile `P` est vide et `False` sinon ;
- `empiler(P, e)` : ajoute l'élément `e` au sommet de la pile `P` ;
- `depiler(P)` : renvoie le sommet de la pile `P` en le retirant de la pile `P`

On donne, ci-dessous, le pseudo-code d'un algorithme de parcours d'un graphe `G` à partir d'un sommet `s` :

```
créer une pile p
empiler s dans p
créer une liste vide visite
tant que p n'est pas vide
   x = depiler p
   si x n'est pas dans la liste visite
      ajouter x à la liste visite
      pour chaque voisin v de x
         empiler v dans p
      fin pour
   fin si
fin tant que
renvoyer visite
```

10. Traduire, dans le corps d'une fonction Python nommée `parcours2`, l'algorithme en pseudo-code donné précédemment. Cette fonction prendra pour paramètres `G` , un dictionnaire représentant un graphe sous la forme de listes d'adjacence, et une chaîne de caractères `s` représentant un sommet du graphe.

11. Donner un résultat possible renvoyé par l'appel `parcours2(G,'Av')`.