EXERCICE 1 (6 points)

Cet exercice porte sur les structures de données FILE et PILE, les graphes et les algorithmes de parcours.

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.

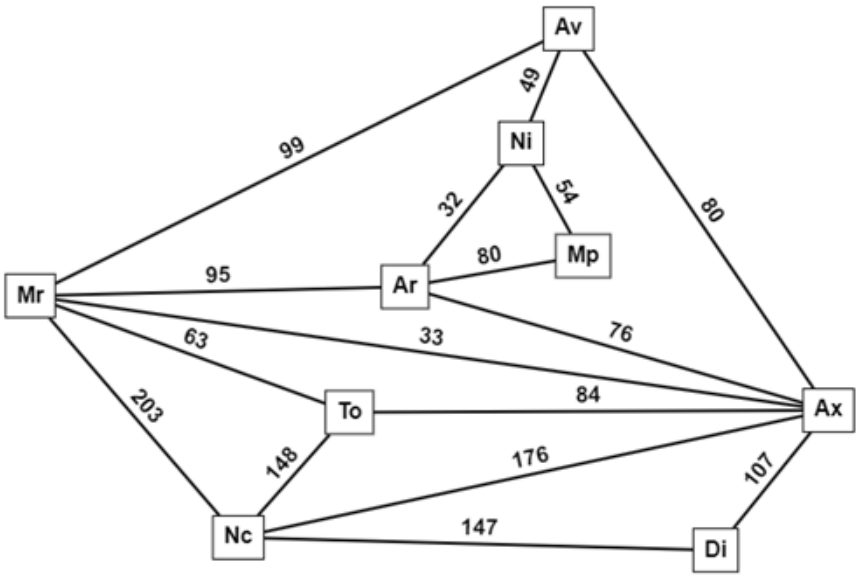

Figure 1. Graphe pondéré

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.

On souhaite toujours se rendre du sommet Mp au sommet Nc mais en visitant le minimum de villes.

2. 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.

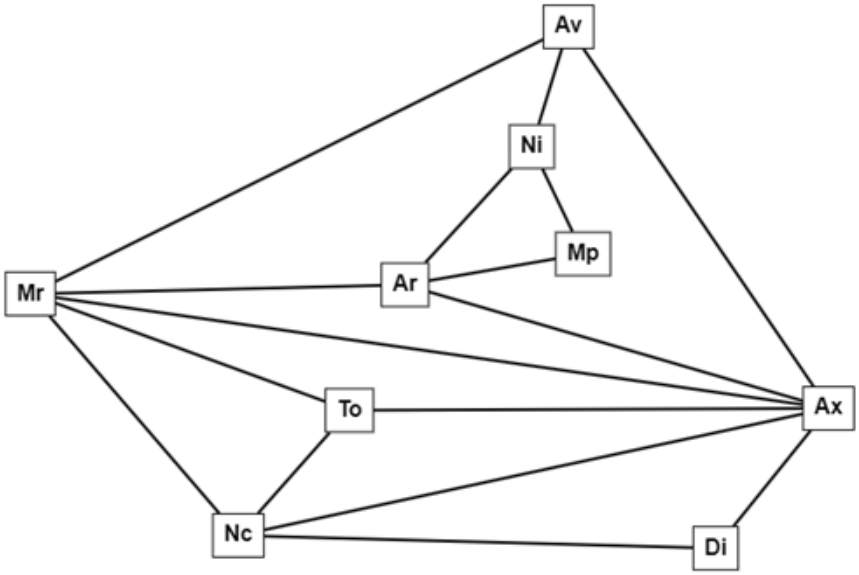

Figure 2. Graphe non pondéré

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` :

```
1  créer une pile p
2  empiler s dans p
3  créer une liste visite contenant s
4  tant que p n'est pas vide
5      x = depiler p
6      si x n'est pas dans la liste visite
7          ajouter x à la liste visite
8          pour chaque voisin v de x
9              empiler v dans p
10         fin pour
11     fin si
12 fin tant que
13 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')`.

Exercice 1, partie B, page 5/15

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

1 créer une pile p

2 empiler s dans p

3 créer une liste visite contenant s »

Remplacer la ligne 3 du pseudo-code par :

« créer une liste visite vide »