Dm - Nsi

Les graphes CORRIGE

Cet exercice porte sur les graphes.

Dans cet exercice, on modélise un groupe de personnes à l'aide d'un graphe.

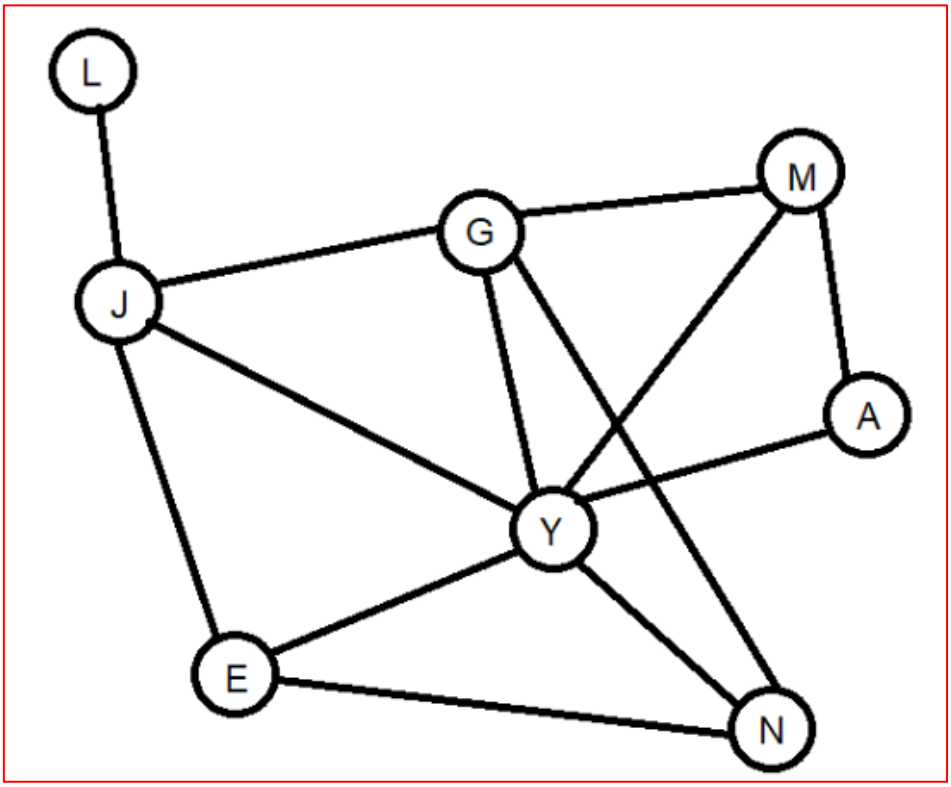
Le groupe est constitué de huit personnes (Anas, Emma, Gabriel, Jade, Lou, Milo, Nina et Yanis) qui possèdent entre elles les relations suivantes :

- 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
```

```
matrice_adj = [
    #G, J, Y, E, N, M, A, L
    [0, 1, 1, 0, 1, 1, 0, 0], # G
    [1, 0, 1, 1, 0, 0, 0, 1], # J
    [1, 1, 0, 1, 1, 1, 1, 0], # Y
    [0, 1, 1, 0, 1, 0, 0, 0], # E
    [1, 0, 1, 1, 0, 0, 0, 0], # N
    [1, 0, 1, 0, 0, 0, 1, 0], # M
    [0, 0, 1, 0, 0, 1, 0, 0], # A
    [0, 1, 0, 0, 0, 0, 0, 0]] # 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')
```

L'exécution de position(sommets , 'G') renvoie 0 car 'G' est l'élément de la liste qui a un index de 0.

L'exécution de position(sommets , 'Z') renvoie None car 'Z' n'est pas un élément de la liste

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 ...
```

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

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

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

L'exécution de nb_amis(sommets, matrice_adj, 'G') renvoie 4 car le nœud 'G' a 4 voisins.

Partie B : Dictionnaire de listes d'adjacence

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

Dans ce dictionnaire c est la clé et v est la valeur.

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.

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

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
```

```
1 def nb_amis(d , s) :
2     amis = len(d[s])
3     return amis
```

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.

Le cercle d'amis de Lou comprend Jade, Gabriel, Yanis, Emma, et Nina.

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     ...
```

```
1 def parcours_en_profondeur(d , s , visites=[]) :
2     visites.append(s)
3     for v in d[s] :
4         if v not in visites :
5             parcours_en_profondeur(d , v) :
6     return visites
```