Informatique CUPGE – Travaux Pratiques

Séance 11: Réseau Social

Université de Paris Cité

Objectifs:

- Manipulation des listes de listes d'entiers
- Boucles imbriquées

Nous voulons dans ce TP modéliser un réseau social dans lequel plusieurs utilisateurs sont inscrits. Les utilisateurs sont nommés avec les numéros $0, 1, 2 \dots n$. Nous allons représenter le réseau par une liste de listes d'entiers `R` (une matrice de deux dimensions) où chaque case `R[i][j]` vaut :

- `1` si l'utilisateur `i` est ami avec l'utilisateur `j`
- `0` sinon.

Par exemple, un réseau `R` dans lequel il y a quatre utilisateurs peut valoir la liste de listes suivante :

```
[[0,0,0,1],[0,0,0,1],[0,0,0,0],[1,1,0,0]]

```

On pourrait formater le code pour qu'il ressemble à une matrice de deux dimensions :

```
[[0,0,0,1],
 [0,0,0,1],
 [0,0,0,0],
 [1,1,0,0]]

```

Les utilisateurs 0 et 1 sont amis avec l'utilisateur 3 (car `R[0][3]=1` et `R[3][0]=1`, et `R[1][3]=1` et `R[3][1]=1`), mais pas entre eux, l'utilisateur 2 n'est ami avec personne et l'utilisateur 3 est ami de 0 et 1, mais pas de 2.

Notez qu'il y a un *invariant*, c'est-à-dire une propriété qui doit être satisfaite par toutes les listes de listes qui représentent un réseau social :

1. un utilisateur n'est jamais ami avec lui-même
2. pour chaque couple d'utilisateurs `i` et `j` : si `i` est l'ami de `j` alors `j` est aussi l'ami de `i`.

Toutes les fonctions et procédures qui créent des réseaux sociaux ou qui les modifient doivent conserver cet invariant.

Exercice 1 (Les fonctionnalités de base, $\star - \star\star$)

1. *Vous connaissez* `[0]*n` *qui permet de créer un tableau de longueur* `n`. *Dire pourquoi le code ci après ne permet pas de créer un tableau de tableaux permettant de gérer un réseau social :*

```
def blank_graph_wrong(n):
    return [[0]*n]*n
```

code/blank_graph_wrong.py

2. *Écrivez une fonction* `random_graph(n)` *qui renvoie un réseau sous la forme de liste de listes comme vu ci-dessus. Il y aura dans cette liste un nombre* `n` *d'utilisateurs et l'amitié entre deux utilisateurs est aléatoire (disons une chance sur deux). Rappelez-vous la fonction* `randint(a,b)` *des TP précédents. Rappelez-vous aussi que la liste des listes renvoyée par* `random_graph` *doit respecter les invariants 1 et 2 décrits ci-dessus.*
3. *Écrivez une fonction* `nb_friends(R,a)` *qui, pour un réseau* `R` *et un utilisateur* `a`*, renvoie le nombre d'amis que possède* `a` *dans le réseau. Par exemple, si* `R` *est le réseau donné en exemple ci-dessus,* `nb_friends(R,2)` *renvoie* `0`*, alors que* `nb_friends(R,3)` *renvoie* `2`*.*
4. *Écrivez une fonction* `ls_friends(R,a)` *qui, pour un réseau* `R` *et un utilisateur* `a`*, renvoie une liste contenant tous les amis de* `a` *dans le réseau et une liste vide s'il n'en possède aucun. Par exemple, si* `R` *est le réseau donné en exemple ci-dessus,* `ls_friends(R,2)` *renvoie la liste vide, alors que* `ls_friends(R,3)` *renvoie* `[0,1]` *(ou la liste* `[1,0]`*, l'ordre de ses éléments n'ayant pas d'importance).*
5. *Écrivez une fonction* `popular(R)` *qui renvoie l'utilisateur le plus populaire du réseau : c'est l'utilisateur avec le plus grand nombre d'amis dans le réseau. S'il y a plusieurs utilisateurs qui sont le plus populaire il suffit d'en renvoyer un de votre choix. Sur le réseau* `R` *donné en exemple ci-dessus, l'appel* `popular(R)` *renvoie* `3`*.*
6. *Écrivez une fonction* `nb_common_friends(R,a,b)` *qui renvoie pour un réseau* `R` *et deux utilisateurs* `a` *et* `b` *le nombre d'amis qu'ils ont en commun. Par exemple, si* `R` *est le réseau donné en exemple ci-dessus,* `nb_common_friends(R,0,1)` *renvoie* `1`*, alors que* `nb_common_friends(R,0,3)` *renvoie* `0`*.*
7. *Écrivez une fonction* `ls_common_friends(R,a,b)` *qui renvoie pour un réseau* `R` *et deux utilisateurs* `a` *et* `b` *une liste contenant les amis qu'ils ont en commun. Par exemple, si* `R` *est le réseau donné en exemple ci-dessus,* `ls_common_friends(R,0,1)` *renvoie* `[3]`*, alors que* `ls_common_friends(R,0,3)` *renvoie la liste vide.*
8. *Écrivez une fonction* `add_user(R)` *qui renvoie un nouveau réseau qui est comme* R*, sauf qu'il y a un utilisateur en plus qui n'a pas d'amis.*
9. *Écrivez une procédure* `register_friends(R,a,frs)` *qui modifie un réseau* `R` *en mettant l'information que tous les utilisateurs de la liste* `frs` *sont des amis de l'utilisateur* `a`*. Veillez à l'invariant expliqué en haut.*
10. *Considérons une liste* `noms` *qui contient des chaînes de caractères. Pour chaque indice* `i` *la case* `noms[i]` *contient le nom de l'utilisateur* `i`*. Par exemple, si* `noms` *vaut :*

```
noms = ['Evan Spiegel', 'Jack Dorsey', 'Mark Zuckerberg', '
Kevin Systrom']

```

Alors le nom de l'utilisateur numéro 2 *est* `noms[2]` *et donc* `'Mark Zuckerberg'`*. Quelle est la longueur de cette liste ? Écrire un fonction* `popular_name` *qui, etant donné un réseau* `R` *et une liste de noms* `noms` *pour les utilisateurs de* `R`*, renvoie le nom de l'utilisateur le plus populaire d'un réseau.*

□

Exercice 2 (Événement, ⋆⋆⋆)

Dans cette partie on souhaite utiliser notre réseau social afin d'améliorer l'alchimie d'une pendaison de crémaillère à venir.

On va tester les exercices suivants avec le réseau social suivant :

```
R2 = [[0, 1, 0, 1, 0],
      [1, 0, 1, 0, 0],
      [0, 1, 0, 1, 1],
      [1, 0, 1, 0, 0],
      [0, 0, 1, 0, 0]]

```

1. *Une possibilité est d'inviter nos amis avec leurs amis à eux. Programmer une fonction* `invite(R,a)` *qui, pour un utilisateur* `a`*, renvoie tous ses amis sur le réseau mais aussi les amis de ses amis. La liste renvoyée ne doit pas contenir* `a` *et doit être sans répétitions.*

   Pour cet exercice il peut être utile d'écrire une fonction `est_element_de(el, l)` *qui renvoie* `True` *si la liste* `l` *contient l'élément* `el`*, et* `False` *sinon.*

   Test : `invite(R2, 0)` *doit renvoyer* `[1, 3, 2]`*.*

2. *Vous êtes prêt à inviter des utilisateurs du réseau même s'ils ne sont pas vos amis. Pour assurer que chacun aura suffisamment matière à conversation vous demandez que chacun des invités ait au moins deux amis communs avec au moins un autre invité.*

   Pour cela, écrivez une fonction `nb_common_friends_matrix(R)` *qui calcule une matrice (une liste de listes d'entiers) qui contient pour chaque paire d'utilisateurs* `i` *et* `j` *le nombre de leurs amis communs. Cette matrice se calcule par*

   $$\texttt{P[i][j]} = \Sigma_{k=0}^{n-1} \ \texttt{R[i][k] * R[k][j]}$$

   NB : Sur la diagonale, la matrice contiendra le nombre d'amis d'un utilisateur (amis communs avec lui-même).

   Écrire une fonction `well_connected` *qui prend un réseau* `R` *en argument, et qui renvoie la liste des utilisateurs du réseau qui ont au moins deux amis communs avec au moins un* autre *utilisateur du réseaux. Cette fonction utilisera* `nb_common_friends_matrix`*.*

   Test : `well_connected(R2)` *doit renvoyer* `[0, 1, 2, 3]`*.*

□