SAÉ S1.02 : Community detection

Ce document correspond au sujet de la SAÉ S1.02 de l'année universitaire 2022/2023. Ce travail est à faire par binôme en autonomie. Il correspond à la suite de la SAÉ S01.01.

La SAÉ devra être rendue avant le 13 janvier.

Évaluation

La SAÉ sera notée de la manière suivante :
- Évaluation du rendu : 8 points répartis de la manière suivante
    - Correction automatique du code via des tests unitaires : 2 points
    - Tests unitaires rendus : 2 points
    - Qualité du code (clarté, efficacité, commentaires, docstring) : 2 points
    - Comparaison expérimentale et théorique (fichier notebook) : 2 points
- Contrôle du 13 janvier : 12 points

Attention : Pour le contrôle du 13 janvier, il faut connaître le sujet et le code (les structures de données utilisées et les diffférentes fonctions définies) dans les SAÉ S01.01 et S01.02.

Sujet

Le sujet est la suite de la SAÉ S01.01. La correction de cette SAÉ est donnée dans le fichier `community_detection.py`. Lorsqu'une comparaison avec une fonction de la SAÉ S01.01 est demandée, celle-ci devra se faire avec la définition donnée en correction.

Remarque : Les noms des fonctions à définir sont en anglais pour faire la différence avec les fonctions définies dans la SAÉ S01.01.

Important : Dans ce projet
- une personne fait référence à une chaîne de caractères qui est le prénom de cette personne ; par exemple `"Alice"` est une personne.
- un groupe de personnes est un tableau de personnes, c'est-à-dire un tableau de chaînes de caractères qui sont les prénoms des personnes. Par exemple, `["Alice", "Bob"]` est un groupe de deux personnes.
- le terme réseau (network en anglais) correspond à un dictionnaire dont les clés sont les prénoms des personnes et les valeurs des tableaux contenant la liste des amis de la personne, comme dans la SAÉ S01.01. Par exemple,
  ```python
  {
    "Alice" : ["Bob", "Dominique"],
    "Bob" : ["Alice", "Charlie", "Dominique"],
    "Charlie" : ["Bob"],
    "Dominique" : ["Alice", "Bob"]
  }
  ```
  est un réseau de 4 personnes : Alice, Bob, Charlie et Dominique. Bob est ami avec tout le monde et Alice est également amie avec Dominique.

  Remarque : L'ordre des amis d'une personne n'a pas d'importance. Le réseau aurait pu être représenté par ce dictionnaire :

```python
{
  "Alice" : ["Bob", "Dominique"],
  "Bob" : ["Alice", "Dominique", "Charlie"],
  "Charlie" : ["Bob"],
  "Dominique" : ["Bob", "Alice"]
}
```

- une communauté est un groupe de personnes qui sont toutes amies entre elles, c'est-à-dire que lorsque l'on prend n'importe quelles deux personnes du groupe, elles sont amies. Pour le réseau donné en exemple, `["Alice", "Bob", "Dominique"]` est une communauté alors que `["Alice", "Bob", "Charlie"]` n'en est pas une.

Question 1

On souhaite faire une fonction similaire à la fonction `dico_reseau` de la SAÉ S01.01 mais avec un autre algorithme.
Définir la fonction `create_network` prenant en paramètre un tableau de couples d'amis et retournant le réseau associé.

Important : L'appel de la fonction `create_network` doit retourner le même dictionnaire (seul l'ordre des amis peut varier) que l'appel de la fonction `dico_reseau` si le tableau de couples d'amis est le même.

La construction du dictionnaire se fera en parcourrant une seule fois le tableau de couples d'amis. Pour chaque couple `(p1, p2)`, on ajoutera directement `p2` à la liste des amis de `p1` si ce dernier est une clé du dictionnaire. Sinon, on ajoutera dans le dictionnaire la clé `p1` avec `p2` comme seul ami. On procèdera de même avec `p2`.

Question 2

Comparer théoriquement et expériementalement les fonctions `dico_reseau` et `create_network`.

Question 3

Définir la fonction `get_people` prenant en paramètre un réseau et retournant la liste des personnes de ce réseau dans un tableau.

L'appel de cette fonction avec le réseau donné en exemple doit retourner par exemple le tableau `["Alice", "Bob", "Charlie", "Dominique"]`. L'ordre n'a pas d'importance.

Question 4

Définir la fonction `are_friends` prenant en paramètre un réseau et deux personnes. La fonction devra retourner `True` si les deux personnes sont amies, et `False` sinon.

L'appel de cette fonction avec le résau donné en exemple et les personnes `"Alice"` et `"Bob"` doit retourner `True`. Par contre, avec les personnes `"Alice"` et `"Charlie"`, la fonction doit retourner `False`.

Question 5

Définir la fonction `all_his_friends` prenant en paramètre réseau, une personne et un groupe de personnes et retournant `True` si la personne est amie avec toutes les personnes du groupe, et False sinon.

L'appel de cette fonction avec le réseau donné en exemple, la personne `"Alice"`, et le groupe `["Bob", "Dominique"]` doit retourner `True` alors qu'elle doit retourner `False` pour le groupe `["Bob", "Charlie"]`.

Question 6

Définir la fonction `is_a_community` prenant en paramètre un dictionnaire modélisant le réseau et un groupe de personnes (tableau de personnes) et retourne `True` si ce groupe est une communauté, et False sinon.

L'appel de cette fonction avec le réseau donné en exemple doit retourner `True` pour le groupe `["Alice", "Bob", "Dominique"]`, et `False` pour le groupe `["Alice", "Bob", "Charlie"]`.

Question 7

Pour trouver une communauté maximale (au sens où il n'existe personne qui puisse être ajoutée dans cette communauté) dans un réseau, une heuristique est la suivante :
- On part d'une communauté vide.
- On considère les personnes les unes après les autres. Pour chacune des personnes, si celle-ci est amie avec tous les membres de la communauté déjà créée, alors on l'ajoute à la communauté.

Définir la fonction `find_community` prenant en paramètre un réseau et un groupe de personnes et retournant une communauté en fonction de l'heuristique décrite juste au-dessus. L'ordre des personnes sera donné par l'ordre de celles-ci dans le tableau correspondant au groupe.

L'appel de cette fonction avec le réseau donné en exemple doit retourner :
- `["Alice", "Bob", "Dominique"]` pour le groupe `["Alice", "Bob", "Charlie", "Dominique"]`,
- `["Charlie", "Bob"]` pour le groupe `["Charlie", "Alice", "Bob", "Dominique"]`,
- `["Charlie"]` pour le groupe `["Charlie", "Alice", "Dominique"]`.

Question 8

Définir la fonction `order_by_decreasing_popularity` prenant en paramètre un réseau et un groupe de personnes et triant le groupe de personnes selon la popularité (nombre d'amis) décroissante.

L'appel de cette fonction avec le réseau donné en exemple et le groupe `["Alice", "Bob", "Charlie"]` doit trier le groupe dans l'ordre `["Bob", "Alice", "Charlie"]`.

Question 9

Définir la fonction `find_community_by_decreasing_popularity` prenant en paramètre un réseau. La fonction doit trier l'ensemble des personnes du réseau selon l'ordre décroissant de popularité puis retourner la communauté trouvée en appliquant l'heuristique de construction de communauté

maximale expliquée plus haut.

L'appel de cette fonction avec le réseau donné en exemple doit retourner la communauté `["Bob", "Alice", "Dominique"]` (l'ordre dans le tableau n'a pas d'importance).

Question 10

Par définition, si une personne appartient à cette communauté, alors la communauté est un sous-ensemble des amis de cette personne plus elle-même.

On peut donc modifier l'heuristique précédente pour constuire une communauté maximale :
- on choisit une personne du réseau,
- on crée une communauté contenant juste cette personne,
- on considère les amis de cette personne par ordre de popularité décroissante. Pour chacune de ces personnes, si celle-ci est amie avec tous les membres de la communauté déjà créée, alors on l'ajoute à la communauté.

Si l'on part d'une personne la plus populaire, alors cette heuristique est la même que la précédente (attention cependant, les communautés retournées peuvent ne pas être les mêmes si plusieurs personnes ont le même nombre d'amis).

Définir la fonction `find_community_from_person` prenant en paramètre un réseau et une personne, et retournant une communauté maximale contenant cette personne selon l'heuristique décrite ci-dessus.

L'appel de cette fonction avec le réseau donné en exemple et la personne `"Alice"` doit retourner la communauté `["Alice", "Bob", "Dominique"]`. Avec la personne `"Charlie"`, la fonction doit retourner `["Charlie", "Bob"]`.

Question 11

Comparer théoriquement et expérimentalement les deux heuristiques de construction, celle donnée par la fonction `find_community_by_decreasing_popularity` et celles donnée par la fonction `find_community_from_person` appliquée à une personne la plus populaire (la recherche de la personne la plus populaire sera prise en compte dans la complexité).

Question 12

Définir la fonction `find_max_community` prenant en paramètre un réseau et appliquant l'heuristique de recherche de communauté maximale donnée par `find_community_from_person` pour toutes les personnes du réseau. La fonction doit retourner la plus grande communauté trouvée.

L'appel de cette fonction avec le réseau donné en exemple doit retourner la communauté `["Alice", "Bob", "Dominique"]` (peu importe l'ordre).

Développement et rendu

La SAÉ devra être faite en binôme. Les fonctions doivent être implémentées dans un module appelé `community_detection.py` et les tests unitaires des fonctions dans le module `test_community_detection.py`. Les comparaisons théoriques et expérimentales demandées devront être données dans un notebook appelé `using_community_detection.ipynb`. Le répertoire du projet

devra également contenir un fichier `etudiants.txt` contenant les codes INE de deux étudiants du projet (un par ligne).

Un squelette de rendu de projet se trouve sur le dépôt gitlab de l'université.