MASTER 1 MEEF : CAPES MATHS OPTION INFORMATIQUE

ECRIT BLANC - EPREUVE DE 3 HEURES

Les résultats des questions pourront être réutilisés ainsi que les différentes fonctions demandées non traitées au cours de l'épreuve. L'usage de la calculatrice est interdit. Le mémento Python est autorisé.

Partition de réseau

Nous souhaitons regrouper des personnes par affinité dans un réseau social. Pour cela, nous cherchons à répartir les personnes en deux groupes de sorte à minimiser le nombre de liens d'amitié entre les deux groupes. Une telle partition s'appelle une coupe minimale de réseau.

Notations. On désignera par $[\![n]\!]$ l'ensemble des entiers de 0 à n-1 : $[\![n]\!] = \{0, \ldots, n-1\}$.

Complexité. La complexité, ou le temps d'exécution, d'un programme P (fonction ou procédure) est le nombre d'opérations élémentaires (addition, multiplication, affectation, test, etc...) nécessaires à l'exécution de P. Lorsqu'il est demandé de donner une complexité, vous justifierez brièvement cette dernière si elle ne se déduit pas directement de la lecture du code. La complexité devra être donnée dans le pire des cas (notation O).

Implémentation. On rappelle qu'en Python, on dispose des opérations suivantes, qui ont toutes une complexité constante (en Python, les listes sont des tableaux de taille dynamique) :

- `[]` crée une liste vide (c.-à-d. ne contenant aucun élément).
- `[x]*n` crée une liste à n éléments contenant tous la valeur contenue dans x. Par exemple, `[1]*3` renvoie la liste `[1,1,1]` à 3 cases contenant toutes la même valeur 1.
- `len(liste)` renvoie la longueur de la liste `liste`
- `liste[i]` désigne le $(i + 1)^{\text{ème}}$ élément de la liste `liste` s'il existe et produit une erreur sinon (noter que le premier élément de la liste est `liste[0]`).
- `liste.append(x)` ajoute le contenu de x à la fin de la liste `liste` qui s'allonge ainsi d'un élément.
- `liste.pop()` renvoie la valeur du dernier élément de la liste `liste` et l'élimine de la liste.
- `random.randint(a,b)` renvoie un entier tiré (pseudo-)aléatoirement et uniformément dans l'ensemble $\{a, a+1, \ldots b-1, b\}$.

Important : L'usage de toute autre fonction sur les listes telle que `liste.insert(i,x)`, `liste.remove(x)`, `liste.index (x)`, ou encore `liste.sort(x)` est interdit (le code de ces fonctions devra être programmé explicitement si nécessaire).

Dans la suite, nous distinguerons fonction et procédure : les fonctions renvoient une valeur (un entier, une liste,...) tandis que les procédures ne renvoient aucune valeur. Vous êtes encouragé à introduire des procédures ou fonctions intermédiaires lorsque cela simplifie l'écriture. Ces procédures et fonctions devront alors être expliquées (ex. documentées).

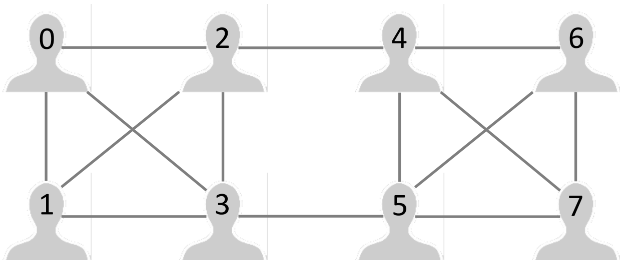
Partie A : Réseaux sociaux

Structure de données. Nous supposerons que des individus sont numérotés de 0 à n-1 où n est le nombre total d'individus. Nous représenterons chaque lien d'amitié entre deux individus i et j par une liste contenant leurs deux numéros dans un ordre quelconque, c.-à-d. par la liste [i,j] ou par la liste [j,i] indifféremment.

Un réseau social R entre n individus sera représenté par une liste `reseau` à deux éléments où :

- `reseau[0]` = n, c'est-à-dire le nombre d'individus appartenant au réseau
- `reseau[1]` = la liste non-ordonnée (et potentiellement vide) des liens d'amitié entre les individus

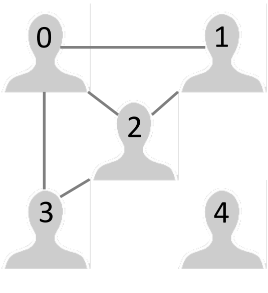
La figure 1 donne l'exemple d'un réseau social et d'une représentation possible sous la forme de liste. Chaque lien d'amitié entre deux personnes est représenté par un trait entre elles.



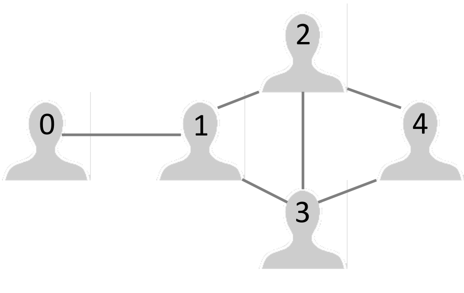

```
reseau = [ 8,
           [ [0,1], [1,3], [3,2], [2,0], [0,3], [2,1], [4,5],
             [5,7], [7,6], [6,4], [7,4], [6,5], [2,4], [5,3] ]
         ]
```

Figure 1: Un réseau à 8 individus ayant 14 liens d'amitié déclarés et une de ses représentations possibles en mémoire sous forme d'une liste [nombre d'individus, liste des liens d'amitié].

1. Donner une représentation sous forme de liste pour chacun des deux réseaux sociaux ci-dessous :



Réseau A



Réseau B

2. Ecrire une fonction `creerReseauVide(n)` qui crée, initialise et renvoie la représentation sous forme de liste du réseau à n individus n'ayant aucun lien d'amitié déclaré entre eux.

3. Ecrire une fonction `estUnLienEntre(paire,i,j)` où `paire` est une liste à deux éléments et i et j sont deux entiers, et qui renvoie `True` si les deux éléments contenus dans `paire` sont i et j dans un ordre quelconque; et renvoie `False` sinon.

4. Ecrire une fonction `sontAmis(reseau,i,j)` qui renvoie `True` s'il existe un lien d'amitié entre les individus i et j dans le réseau `reseau`; et renvoie `False` sinon. Quelle est la complexité en temps de votre fonction dans le pire cas en fonction du nombre m de liens d'amitié déclarés dans le réseau?

5. Ecrire une procédure `declareAmis(reseau,i,j)` qui modifie le réseau `reseau` pour y ajouter le lien d'amitié entre les individus i et j si ce lien n'y figure pas déjà. Quelle est la complexité en temps de votre procédure dans le pire cas en fonction du nombre m de liens d'amitié déclarés dans le réseau?

6. Ecrire une fonction `listeDesAmisDe(reseau,i)` qui renvoie la liste des amis de i dans le réseau `reseau`. Quelle est la complexité en temps de votre fonction dans le pire cas en fonction du nombre m de liens d'amitié déclarés dans le réseau?

Partie B : Partitions

Une partition en k groupes d'un ensemble A à n éléments consiste en k sous-ensembles disjoints non vides $A_1, \dots, A_k$ de A dont l'union est A, c.-à-d. tels que $A_1 \cup \dots \cup A_k = A$ et pour tout $i \neq j$, $A_i \cap A_j = \emptyset$. Par exemple $A_1 = \{1,3\}, A_2 = \{0,4,5\}, A_3 = \{2\}$ est une partition en trois groupes de $A = [\![6]\!]$. Dans cette partie, nous implémenterons une structure de données très efficace pour coder des partitions de $[\![n]\!]$.

Le principe de cette structure de données est que les éléments de chaque groupe sont structurés par une relation filiale : chaque élément a un unique parent choisi dans le groupe. L'unique élément du groupe qui est son propre parent est appelé le représentant du groupe. On s'assure par construction que chaque élément i du groupe a bien pour ancêtre le représentant du groupe, c.-à-d. que le représentant du groupe est bien le parent du parent du parent etc. (autant de fois que nécessaire) de l'élément i. La relation filiale est symbolisée par une flèche allant de l'enfant au parent dans la figure 2 qui présente un exemple de cette structure de données. Dans l'exemple de cette figure, 14 a pour parent 11 qui a pour parent 1 qui a pour parent 9 qui est son propre parent. Ainsi, 9 est le représentant du groupe auquel appartiennent 14, 11, 1 et 9. Notons que ce groupe contient également 8, 13 et 15.

Notez que la représentation n'est pas unique (si l'on choisit un autre représentant pour un groupe et une autre relation filiale, on pourra toujours avoir une représentation du même groupe).

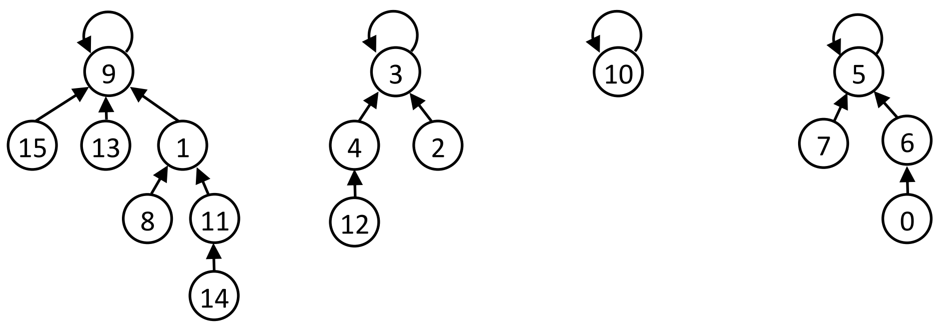
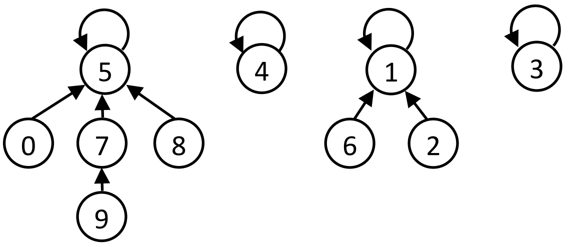


Figure 2 : Une représentation filiale de la partition suivante de $[\![16]\!]$ en quatre groupes : {1,8,9,11,13,14,15}, {2,3,4,12}, {10} et {0,5,6,7} dont les représentants respectifs sont 9, 3, 10 et 5.

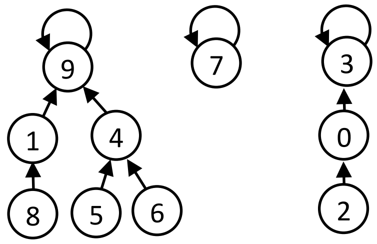
Pour coder cette structure de données, on utilise un tableau `parent` à n éléments où la case `parent[i]` contient le numéro du parent de i. Par exemple, les valeurs du tableau `parent` encodant la représentation filiale donnée dans la figure 2 sont :

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
`parent[i]`	6	9	3	3	3	5	5	5	1	9	10	1	4	9	11	9

7. Donner les valeurs du tableau `parent` encodant les représentations filiales des deux partitions de $\llbracket 10 \rrbracket$ ci-dessous, et préciser les représentants de chaque groupe :



Représentation A



Représentation B

Initialement, chaque élément de $\llbracket n \rrbracket$ est son propre représentant et la partition initiale consiste en n groupes contenant chacun un individu. Ainsi, initialement, `parent`[i] = i pour tout $i \in \llbracket n \rrbracket$.

8. Ecrire une fonction `creerPartitionEnSingletons(n)` qui crée et renvoie un tableau à n éléments dont les valeurs sont initialisées de sorte à encoder la partition de $\llbracket n \rrbracket$ en n groupes d'un seul élément.

Nous sommes intéressés par deux opérations sur les partitions :

- Déterminer si deux éléments appartiennent au même groupe dans la partition.
- Fusionner deux groupes pour n'en faire plus qu'un. Par exemple, la fusion des groupes $A_1 = \{1,3\}$ et $A_3 = \{2\}$ dans la partition de $\llbracket 6 \rrbracket$ donnée en exemple en début de partie donnera la partition en deux groupes $A_2 = \{0,4,5\}$ et $A_4 = A_1 \cup A_3 = \{1,2,3\}$.

9. Ecrire une fonction `representant(parent,i)` qui utilise le tableau `parent` pour trouver et renvoyer la valeur du représentant du groupe auquel appartient i dans la partition encodée par le tableau `parent`. Quelle est la complexité dans le pire cas de votre fonction en fonction du nombre total n d'éléments? Indiquer également quel est ce pire cas.

Pour réaliser la fusion de deux groupes désignés par l'un de leurs éléments i et j respectivement, on applique l'algorithme suivant :

1. Calculer les représentants p et q des deux groupes contenant i et j respectivement

2. Affecter q comme parent de p

La figure 3 présente la structure filiale obtenue après la fusion des groupes contenant respectivement 6 et 14 de la figure 2.

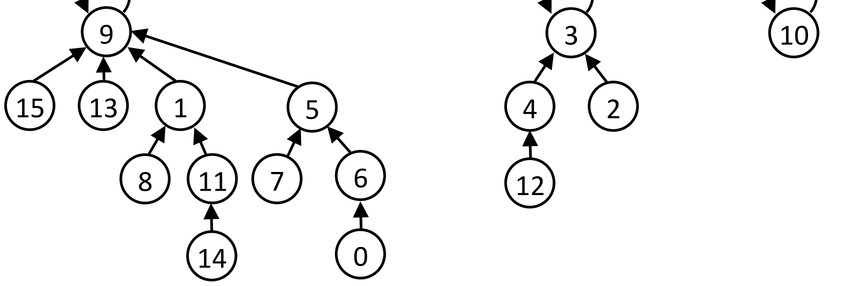


Figure 3 : Représentation filiale obtenue après la fusion des groupes contenant respectivement 6 et 14 de la figure 2.

10. Ecrire une procédure `fusion(parent,i,j)` qui modifie le tableau `parent` pour fusionner les deux groupes contenant i et j respectivement.

11. Quelle est la complexité de la fusion de n singletons de $[\![n]\!]$?

Pour remédier à cette mauvaise performance, une astuce consiste à compresser la relation filiale à chaque appel à la fonction `representant(parent,i)`. L'opération de compression est la suivante : si p est le résultat de l'appel à la fonction `representant(parent,i)`, modifier le tableau `parent` de façon à ce que chaque ancêtre (c.-à-d. parent de parent ... de parent) de i (inclus), ait pour parent direct p. Noter bien que un appel à `representant(parent,i)` renvoie le représentant de i, mais modifie donc également le tableau `parent`. Si l'on reprend l'exemple de la figure 2, le résultat de l'appel `representant(parent,14)` est 9, que l'on a calculé en remontant les ancêtres successifs de 14 : 11, 1 puis 9. L'opération de compression consiste alors à donner la valeur 9 aux cases d'indices 14, 11, et 1 du tableau `parent`. La structure filiale obtenue après l'opération de compression menée depuis 14 est illustrée dans la figure 4.

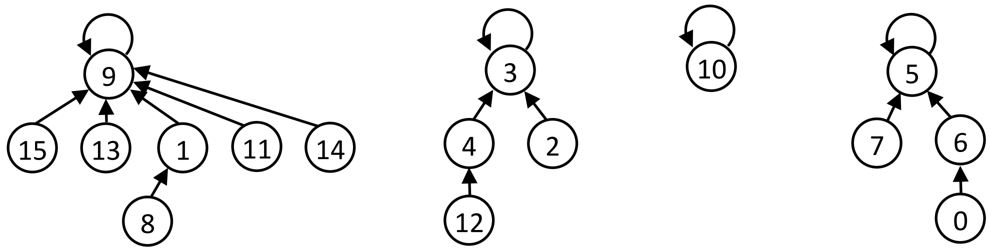


Figure 4 : Résultat de la compression depuis 14 dans la représentation filiale de la figure 2

12. Modifier votre fonction `representant(parent,i)` pour qu'elle modifie également le tableau `parent` pour faire pointer directement tous les ancêtres de i vers le représentant de i. En quoi cette optimisation de la structure filiale peut-elle être considérée comme « gratuite » du point de vue de la complexité de la fonction `representant`?

Afin d'afficher de manière lisible la partition codée par un tableau `parent`, on souhaite calculer à partir du tableau `parent` la liste des listes des éléments des différents groupes. Une sortie possible pour le tableau `parent` correspondant à la figure 2 serait :

```
[ [ 9, 1, 8, 11, 13, 14, 15],
  [ 3, 2, 4, 12],
  [ 10 ],
  [ 5, 0, 6, 7] ]
```

13 (difficile). Ecrire une fonction `listeDesGroupes(parent)` qui renvoie la liste des différents groupes codés par le tableau `parent` sous la forme d'une liste des listes des éléments des différents groupes.

<u>Indication</u> : On itérera sur les individus dont on récupère les représentants. Si un des groupes en constitution admet déjà le représentant, on ajoute l'individu à la liste existante, sinon on crée une nouvelle liste avec le représentant et l'individu.

Partie C : Algorithme aléatoire pour la coupe minimum

Revenons à présent à notre objectif principal : trouver une partition des individus d'un réseau social en deux groupes qui minimise le nombre de liens d'amitiés entre les deux groupes. Pour résoudre ce problème nous allons utiliser un algorithme aléatoire qui prend en entrée un réseau social à n individus :

1. Créer une partition P en n singletons de $[\![n]\!]$ où initialement aucun lien d'amitié n'est marqué
2. Tant que la partition P contient au moins trois groupes et qu'il reste des liens d'amitié non marqués dans le réseau faire :
    a. Choisir un lien au hasard parmi les liens non marqués du réseau, et notons le [i; j]
    b. Si i et j n'appartiennent pas au même groupe dans la partition P, fusionner les deux groupes correspondants
    c. Marquer le lien [i; j]
3. Si P contient encore $k \geq 3$ groupes, faire k - 1 fusions jusqu'à obtenir deux groupes
4. Renvoyer la partition P

La figure 5 présente une exécution possible de cet algorithme sur le réseau de la figure 1.

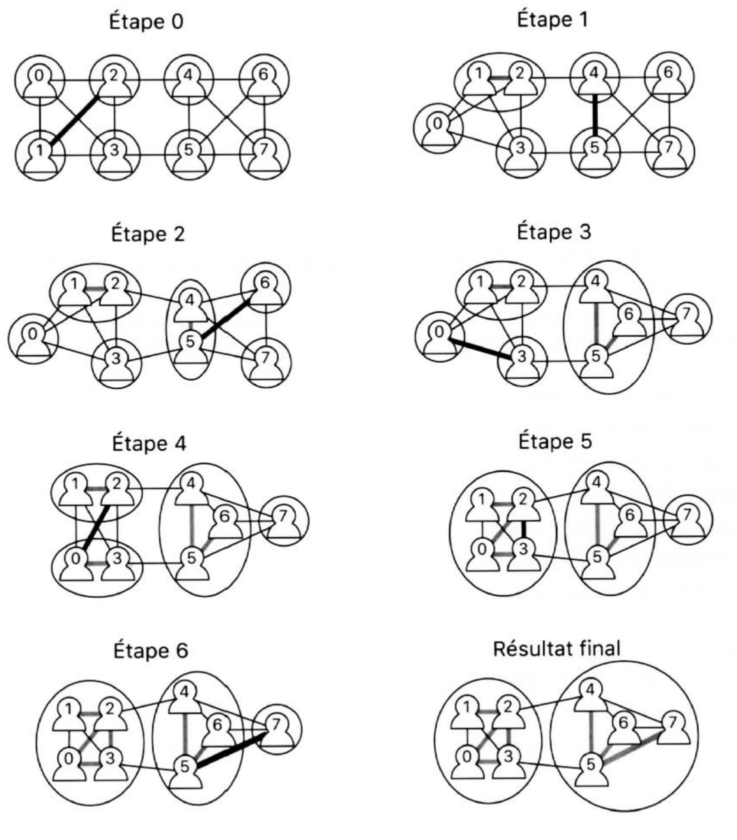


Figure 5 : Une exécution de l'algorithme aléatoire sur le réseau de la figure 1 où les liens sont sélectionnés dans l'ordre : [2, 1], [4, 5], [6, 5], [0, 3], [2, 0], [3, 2] et [5, 7]. Les liens représentés en noir épais sont les liens sélectionnés au hasard à l'étape courante: les liens en gris épais sont les liens marqués par l'algorithme; les ronds représentent la partition à l'étape courante.

14 (difficile). Ecrire une fonction `coupeMinimumAleatoire(reseau)` qui renvoie le tableau `parent` correspondant à la partition calculée par l'algorithme ci-dessus. Quelle est la complexité en fonction de n, m et $\alpha(n)$, où m est le nombre de liens d'amitié déclarés dans le réseau et où $\alpha(n)$ désigne la complexité d'un appel à la fonction `representant`?

<u>Indication</u> : au lieu de marquer explicitement les liens déjà vus, on pourra avantageusement les positionner à la fin de la liste non ordonnée des liens du réseau et ainsi pouvoir tirer simplement les liens au hasard parmi ceux placés au début de la liste. Ceci peut se faire en permutant le lien étudié avec le dernier des liens non marqués.

15. Ecrire une fonction `taillecoupe(reseau, parent)` qui calcule le nombre de liens entre les différents groupes de la partition représentée par `parent` dans le réseau `reseau`. Dans l'exemple de la figure 5, cette fonction renverrait 2, correspondant aux liens [2,4] et [3,5].