Devoir Surveillé n° 1

MP Lycée Camille Jullian

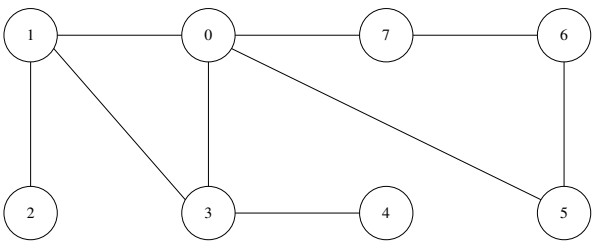
22 octobre 2021

Ce sujet, fortement inspiré d'un sujet de concours (X-ENS 2016 filière MP), est consacré à l'étude de certaines propriétés des réseaux sociaux et des graphes par lesquels ils peuvent être modélisés. Quelques précisions concernant ce qui est attendu dans les questions de programmation qui constituent la quasi-totalité du sujet :

- toutes les fonctions doivent évidemment être programmées dans le langage Python.
- concernant les manipulations de liste, la fonction **len**, les méthodes **append** et **pop**, ainsi que toutes les utilisations du *slicing* (accès à une « tranche » de la liste via la commande l[i :j]) sont autorisés dans toutes les questions. Les méthodes **insert**, **remove** ou **sort** sont par contre interdites, sauf mention explicite du contraire dans l'énoncé de la question.
- si l'énoncé demande de programmer une *fonction*, le programme doit renvoyer une valeur (matérialisée par un **return**), s'il s'agit d'une *procédure*, aucun retour n'est attendu.
- on rappelle que, si l'énoncé demande qu'une certaine fonction renvoie « un booléen », la sortie du programme doit être True ou False.
- on rappelle que le module **random** contient une fonction **randint(a,b)**, qui retourne un entier aléatoire compris entre a et b (tous les deux inclus).
- pour les questions portant sur la complexité de certains algorithmes, une réponse rapidement justifiée (un argument du type « on effectue n boucles qui contiennent chacune un nombre constant d'opérations » suffit) sous la forme « complexité en $O(n)$ » (où n est un paramètre pertinent de la fonction étudiée) est attendue.

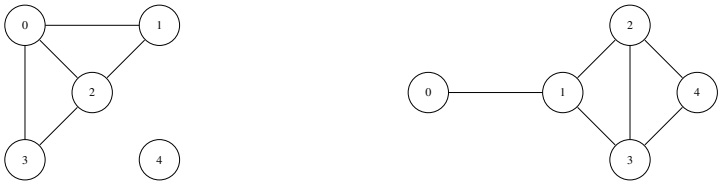
I. Modélisation de graphes et algorithmes élémentaires.

Les graphes étudiés dans ce problème seront des graphes non-orientés, c'est-à-dire des objets mathématiques consitués d'une liste de n sommets et d'arêtes reliant deux de ces sommets de manière symétrique (l'arête reliant les sommets 1 et 3 dans l'exemple suivant n'est pas orientée, 1 est relié à 3 et 3 est relié à 1 de façon indifférente). Un exemple de tel graphe est constitué par l'ensemble des usagers d'un réseau social, deux usagers étant reliés par une arête si et seulement si ils se sont déclarés comme « amis » sur ledit réseau. Ci-dessous, un exemple de représentation graphique élémentaire d'un graphe non-orienté :



Un tel graphe sera réprésenté en Python par une liste dont le premier élément sera l'entier n représentant le nombre de sommets du graphe (qui seront toujours numérotés de 0 à $n-1$ dans les exemples donnés dans le sujet pour rester cohérents avec les notations usuelles en Python), et dont le deuxième élément sera lui-même une liste contenant toutes les paires (chacune sous forme de liste à deux éléments !) d'éléments reliés entre eux dans le graphe. L'ordre des éléments de cette liste n'a aucune importance, et on supposera par ailleurs que la liste de liens n'est jamais redondante (si le couple $[1, 3]$ apparait dans liste, il n'y apparait qu'une seule fois, et le couple $[3, 1]$ ne peut pas être également dans la liste). Ainsi, notre exemple précédent serait représenté par exemple par la liste **graphe**= $[8, [[1, 2], [3, 1], [0, 3], [1, 0], [3, 4], [0, 5], [5, 6], [7, 6], [7, 0]]]$.

Question 1 : Donner la représentation sous forme de liste Python des deux graphes représentés dans la figure ci-dessous.



Question 2 : Écrire une fonction Python **taille(g)** renvoyant la taille du graphe représenté par la liste g, c'est-à-dire son nombre d'arêtes.

Question 3 : Écrire une fonction Python **degre(g,i)** qui renvoie le nombre de voisins du sommet numéro i dans le graphe représenté par g, c'est-à-dire le nombre de sommets reliés à i par une arête. On fera attention à renvoyer un message d'erreur si i n'est pas un numéro de sommet valide pour le graphe.

Question 4 : Écrire une fonction Python **voisins(g,i)** qui renvoie la liste des voisins du sommet numéro i dans le graphe représenté par la liste g. Quelle est la complexité de votre fonction (on essaiera de l'exprimer en fonction de la taille m du graphe manipulé) ?

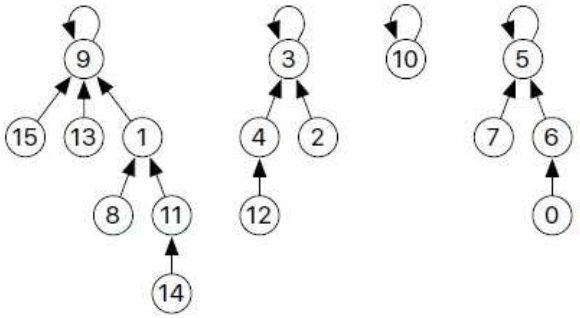
Question 5 : Écrire une fonction Python **degremax(g)** qui renvoie le nombre maximal de voisins d'un sommet du graphe représenté par g (tout appel à une méthode calculant immédiatement le maximum d'une liste est interdit pour cette question).

Question 6 : Écrire une fonction Python **lien(g,i,j)** qui renvoie True si les sommets i et j sont reliés par une arête du graphe, False sinon. Quelle est la complexité de cette fonction en fonction de la taille du graphe ?

Question 7 : Écrire une procédure Python **creelien(g,i,j)** qui modifie la liste g pour y ajouter une arête entre les sommets i et j (bien sûr, on ne fera cet ajout que s'il n'existe pas encore d'arête reliant les deux sommets). Estimer la complexité de votre fonction en fonction de la taille du graphe.

II. Partitions d'un ensemble.

Une *partition* d'un ensemble E en k sous-ensembles est consitué de sous-ensembles disjoints et non-vides dont l'union est égale à E. Ainsi, une partition de l'ensemble $E = \{1, 2, 3, 4, 5, 6\}$ en trois sous-ensembles peut être donnée par la répartition suivante : $A_1 = \{2, 4\}$, $A_2 = \{1, 3, 6\}$ et $A_3 = \{5\}$. Pour représenter facilement une partition sur un ensemble donné, on considère que celui-ci est muni d'une relation *filiale* : chaque élément de l'ensemble est associé à un unique élément de l'ensemble appelé son *parent* (qui a le droit d'être l'élément lui-même) et qui appartient au même sous-ensemble de la partition de lui. Un élément qui est son propre parent est appelé *représentant* d'un sous-ensemble de la partition, et on s'assure que tout élément de l'ensemble soit relié au représentant de son sous-ensemble par une chaîne de parents (le représentant doit être le parent du parent du parent etc. de l'élément choisi).



Dans le schéma ci-dessus, chaque flèche relie un élément de l'ensemble $\{0, 1, \ldots, 15\}$ à son parent. Le schéma représente ainsi une partition de cette ensemble en quatre sous-ensembles : $A_1 = \{1, 8, 9, 11, 13, 14, 15\}$, $A_2 = \{2, 3, 4, 12\}$, $A_3 = \{10\}$ et $A_4 = \{0, 5, 6, 7\}$, dont les représentants respectifs sont 9, 3, 10 et 5. Pour coder ce genre de schéma en Python, on crée simplement une liste **parent**, dont l'élément parent[i] contient le numéro du parent de l'élément i. Ainsi, le schéma précédent serait codé par le tableau $parent = [6, 9, 3, 3, 3, 5, 5, 5, 1, 9, 10, 1, 4, 9, 11, 9]$.

Question 8 : Écrire une fonction Python **creepartitionsingletons(n)** qui crée et renvoie le tableau parent à n éléments correspondant à une partition de l'ensemble $\{0, 1, \ldots, n-1\}$ en singletons, c'est-à-dire en n sous-ensembles contenant chacun exactement un élément.

Question 9 : Écrire une fonction Python **representant(parent,i)** qui prend comme arguments une liste parent et un entier i et qui renvoie le numéro du représentant du sous-ensemble auquel appartient i dans la partition représentée par le tableau parent.

L'opération de *fusion* de deux sous-ensembles d'une partition consiste à regrouper les deux sous-ensembles de la partition contenant deux éléments i et j en un seul sous-ensemble. Pour cela, on effectue simplement l'algorithme suivant : on calcule les représentants p et q des deux sous-ensembles

auxquels appartiennent i et j, puis on effectue le changement parent[p]=q dans le tableau parent représentant la partition. On admet que cette opération a l'effet souhaité, et qu'elle fonctionne également dans le cas où i et j appartenaient au même sous-ensemble de la partition (elle me modifie rien dans ce cas).

Question 10 : Écrire une fonction Python **fusion(parent,i,j)** qui modifie le tableau parent pour rendre compte de la fusion des sous-ensembles contenant les éléments i et j.

Question 11 : Écrire une fonction Python **partition(parent)** qui prend comme argument un tableau parent correspondant à une partition, et renvoie la liste des sous-ensembles de la partition, sous forme de liste de listes d'éléments. Ainsi, la partition schématisée plus haut sera renvoyée sous la forme (l'ordre des éléments dans chaque liste n'a pas d'importance) : $[[15, 8, 1, 9, 11, 13, 14], [4, 3, 2, 12], [7, 5, 6, 0], [10]]$.

III. Recherche d'une coupe minimale.

Le but de cette dernière partie est de mettre en place un algorithme permettant de trouver une coupe minimale, c'est-à-dire une partition des sommets d'un graphe en deux sous-ensembles ayant le moins possible de liens entre eux. Dans le cas d'un réseau social, il s'agit donc de découper la population des usagers en deux groupes ayant le moins possible de liens d'amitié entre eux. L'algorithme proposé est le suivant, qui est un algorithme randomisé (faisant appel au hasard) ne donnant pas toujours (loin de là même) un résultat optimal. En notant n le nombre de sommets du graphe, on procède comme suit :

- on crée une partition en singletons de l'ensemble des sommets du graphe, notée P pour la suite.
- initialement, aucune arête du graphe n'est marquée
- tant que la partition P contient au moins trois sous-ensembles et qu'il reste des arêtes du graphe non marquées, on choisit au hasard une arête non encore marquée, on fusionne les deux sous-ensembles contenant les éléments reliés par cette arête, puis on marque l'arête.
- si à l'issue de cette procédure, la partition P contient encore 3 sous-ensembles ou plus, on en fusionne le nombre nécessaire pour retomber à deux sous-ensembles
- on renvoie la partition P ainsi obtenue

Question 12 : Écrire une fonction Python **coupeminrandom(g)** qui effectue le calcul de la coupe minimle d'un graphe représenté par une liste g à l'aide de cet algorithme.

Pour simplifier l'écriture du programme, on pourra déplacer en fin de liste les arêtes marquées, pour ne tirer l'arête suivante que parmi les premières arêtes disponibles dans la liste.

Question 13 : Écrire une fonction Python **lienscoupe(g,parent)** qui calcule le nombre d'arêtes reliant deux éléments du graphe g appartenant à des sous-groupes différents de la partition représentée par le tableau parent (il n'est absolument pas nécessaire d'avoir réussi la question précédente pour aborder celle-ci).