TP noté

PTSI Lycée Eiffel

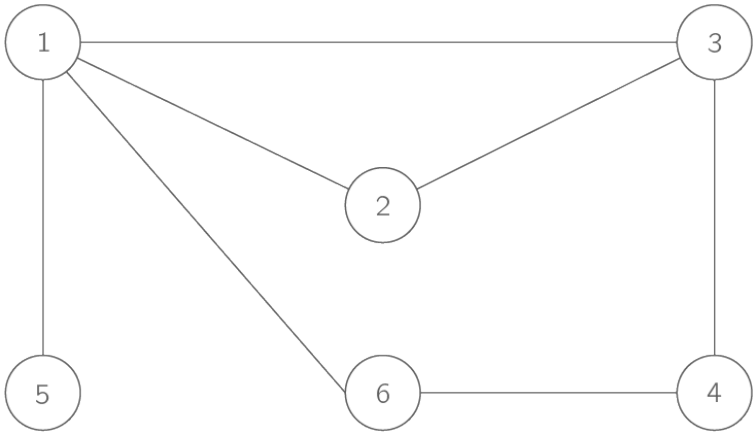
15 décembre 2016

Quelques remarques avant de commencer :

- **Les notes de cours sont autorisées (mais PAS le recours à des programmes tout faits trouvés sur Internet).**
- **Les seules fonctions Python sur les listes que vous avez le droit d'utiliser sont *len* et la méthode *append*. En particulier, PAS de *sort* ou de *sum*.**
- **Les étudiants rendront à la fin du TP un fichier Python (extension .py) contenant leurs programmes que l'enseignant récuperera directement sur une clé USB. Il est bien entendu autorisé et même conseillé de commenter les programmes écrits sur ce fichier (les commentaires en Python s'introduisent avec un #).**
- **On ne se préoccupera pas de l'efficacité des algorithmes programmés, et on aura bien entendu le droit de réutiliser certains des programmes écrits précédemment dans des questions ultérieures du sujet.**

I. Quelques algorithmes élémentaires sur les graphes.

Le thème de ce TP noté est l'étude des graphes. Dans cette première partie, nous nous concentrerons sur des graphes non orientés, qui sont donc simplement constitués d'une liste de n sommets (numérotés de 1 à n), et d'un certain nombre d'arêtes reliant deux des sommets du graphe. Un exemple d'un tel graphe :



Un tel graphe sera représenté dans un premier temps en Python par la liste de ses arêtes, chaque arête étant elle-même définie par une liste de deux nombres entiers contenant les numéros des deux sommets reliés par l'arête (l'ordre n'ayant ici aucune importance). Ainsi, le graphe précédent sera représenté par la liste suivante : **graphe**= $[[1,2],[1,3],[1,5],[3,2],[1,6],[6,4],[4,3]]$ (l'ordre des arêtes dans la liste n'a aucune importance non plus). Cet exemple sera celui repris tout au long de cette première partie. On supposera que chaque sommet du graphe est relié à au moins un autre sommet, et apparait donc dans une des arêtes de la liste représentant le graphe.

Question 1 : Écrire une fonction Python **taille(g)** renvoyant la taille du graphe représenté par la liste g, c'est-à-dire son nombre d'arêtes.

Exemple : taille(graphe) doit retourner la valeur 7.

Question 2 : Écrire une fonction Python **ordre(g)** qui retourne l'ordre du graphe représenté par la liste g, c'est-à-dire son nombre de sommets (on rappelle que les sommets seront toujours numérotés de 1 à n, il suffit donc de déterminer le sommet ayant le plus grand numéro !).

Exemple : ordre(graphe) doit retourner la valeur 6.

Question 3 : Écrire une fonction Python **degre(g,n)** qui renvoie le nombre de voisins du sommet n dans le graphe représenté par g (c'est-à-dire le nombre de sommets reliés à n par une arête), aussi appelé degré du sommet n dans le graphe g.

Exemple : degre(graphe,1) doit retourner la valeur 4.

Question 4 : Écrire une fonction Python **voisins(g,n)** qui va renvoyer cette fois-ci la liste des voisins du sommet n dans le graphe g.

Exemple : voisins(graphe,3) doit retourner la liste [1,2,4] (l'ordre des éléments dans la liste n'a pas d'importance).

Question 5 : Écrire une fonction Python **degremax(g)** qui calcule le degré maximal du graphe g.

Exemple : degremax(graphe) doit retourner la valeur 4.

Question 6 : Écrire une fonction Python **regulier(g)** qui déterminer si le graphe g est régulier, c'est-à-dire si tous ses sommets ont le même nombre de voisins (la fonction renverra True si c'est le cas, False sinon).

Exemple : regulier(g) doit retourner la valeur False.

II. Quelques algorithmes sur les matrices.

Une matrice est un tableau de nombres constitué de n lignes et p colonnes, et habituellement représenté en mathématiques de la façon suivante : $M = \begin{pmatrix} 1 & 2 & 4 \\ -1 & 1 & 3 \end{pmatrix}$ est par exemple une matrice à 2 lignes et 3 colonnes. Une telle matrice pourra être représentée en Python par une liste de n éléments, chacun de ces éléments étant lui-même une liste constituée de p éléments. Ainsi, la matrice M précédente correspondra en Python à la liste $l = [[1, 2, 4], [-1, 1, 3]]$. L'élément situé sur la i-ème ligne, j-ème colonne de la matrice M sera ainsi accessible dans la version Python via la commande $l[i][j]$ (attention tout de même au fait que les lignes et colonnes seront numérotées à partir de 0 et non à partir de 1 comme on le fait habituellement en maths).

Question 7 : On suppose que la variable l est une liste de listes. Écrire une fonction Python **testmatrice(l)** qui détermine si l est une matrice ou non, c'est-à-dire si chacune de ses lignes contient le même nombre d'éléments. Cette fonction renverra True ou False selon le résultat du test.

Question 8 : Écrire une fonction Python **matcarree(l)** qui détermine si une liste de listes l est une matrice carrée (nombre de lignes égal au nombre de colonnes).

Question 9 : Dans cette question et les suivantes, on suppose que m est une liste de listes correspondant à une matrice carrée. Écrire une fonction Python **zeros(m)** comptant le nombre de zéros contenus dans la matrice.

Question 10 : Le produit mathématique de deux matrices carrées (contenant chacune n lignes et n colonnes) se calcule de la façon suivante : en notant $a_{i,j}$ le coefficient situé dans la i-ème ligne et j-ème colonne de la matrice A, les coefficients de la matrice $C = A \times B$ sont obtenus par le calcul $c_{i,j} = \sum_{k=0}^{n-1} a_{i,k} \times b_{k,j}$. Écrire une fonction Python **produit(a,b)** effectuant le produit de deux matrices carrées a et b (attention, le calcul de somme permet de calculer un seul élément de la matrice C, votre programme devra donc faire intervenir une triple boucle pour faire varier les indices i et j, puis l'indice k pour le calcul de somme).

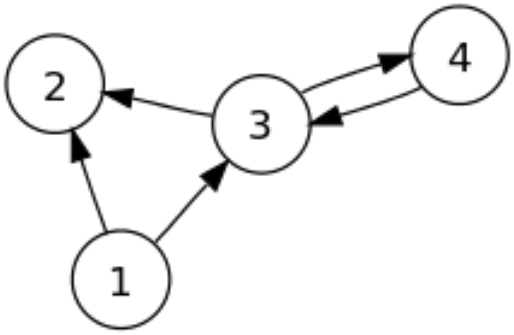
Exemple : produit([[1,2],[3,1]],[[2,0],[1,1]]) doit retourner la matrice $[[4, 2], [7, 1]]$, mais **produit([[2,0],[1,1]],[[1,2],[3,1]])** doit renvoyer la matrice $[[2, 4], [4, 3]]$ (le produit de matrices n'est pas commutatif).

Question 11 : Écrire une fonction Python **puissance(m,k)** qui calcule la matrice m^k pour une matrice carrée m donnée (cette matrice est simplement définie par $M^k = m \times \cdots \times m$ (k produits de m par elle-même)).

III. Matrice d'adjacence d'un graphe.

Revenons désormais à nos histoires de graphes. Une autre représentation possible d'un graphe est d'utiliser une matrice appelée matrice d'adjacence du graphe, qui ne contient que des 0 et des 1, le coefficient situé sur la i-ème ligne et la j-ème colonne de la matrice valant 1 si et seulement si les sommets i et j sont reliés dans le graphe (attention, ici, il est sous-entendu que les lignes et colonnes de la matrice sont numérotés à partir de 1, ce qui n'était pas le cas dans la partie précédente). Ainsi, le graphe défini au début de l'énoncé aurait pour matrice d'adjacence $M = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$.

Cette représentation sous forme de matrice d'adjacence est également adaptée à la représentation de graphes orientés, c'est-à-dire de graphes où les arêtes peuvent être à sens unique (on les représente alors sous forme de flèches dirigées vers le sommet d'arrivée) : si $M[i][j] = 0$, mais $M[j][i] = 1$, cela signifie qu'il existe dans le graphe une arête allant de j vers i, mais pas d'arête allant de i vers j. Le graphe orienté représenté ci-dessous aurait pour matrice d'adjacence $\begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$



Question 12 : Écrire une fonction Python **nonoriente(m)** qui prend comme argument une matrice carrée m (qu'on suppose remplie uniquement de 0 et de 1), et qui détermine s'il s'agit de la matrice d'un graphe non orienté (ce sera le cas si chacune des arêtes peut être parcourue dans les deux sens, donc si $m[i][j]$ est toujours égal à $m[j][i]$).

Question 13 : Écrire des fonctions Python **taillebis(m)** et **degrebis(m,n)** qui renvoient respectivement la taille et le degré du sommet n dans un graphe de matrice d'adjacence m (cf questions 1 et 3 pour la définition de la taille et du degré).

Question 14 : Écrire une fonction Python **conversiongraphematrice(g)** qui prend comme argument la liste des arêtes d'un graphe (comme on l'a définie dans la partie I.) et qui renvoit la matrice d'adjacence correspondante.

Question 15 : Écrire une fonction Python **conversionmatricegraphe(m)** qui part cette fois de la matrice d'adjacence et qui retourne la liste des arêtes du graphe correspondant (si on a le temps, on en fera une version adaptée aux graphes orientés, où une arête qui peut être parcourue dans le deux sens devra donc apparaitre deux fois dans la liste, avec l'ordre des sommets inversé ; et une version pour les graphes non orientés, où une même arête ne devra surtout pas apparaitre deux fois).

On admet le résultat suivant : si M est la matrice d'adjacence d'un graphe g, le coefficient situé sur la i-ème ligne et j-ème colonne de la matrice M^k représente le nombre de chemins possibles dans le graphe permettant d'aller du sommet i jusqu'au sommet j en k étapes. En particulier, ce coefficient est nul si et seulement si on ne peut pas atteindre le sommet j depuis le sommet i en k étapes (mais on peut peut-être l'atteindre en moins de k étapes !). La **distance** entre deux sommets i et j d'un graphe est le nombre minimal d'arêtes que l'on doit parcourir pour atteindre le sommet j depuis le sommet i. Le **diamètre** d'un graphe est la distance maximale entre deux sommets de ce graphe (on suppose ici que nos graphes sont connexes, c'est-à-dire qu'on peut toujours atteindre un sommet du graphe en partant d'un autre quelconque de ses sommets).

Question 16 : Écrire une fonction Python **distance(m,i,j)** qui calcule la distance entre les sommets i et j dans le graphe dont la matrice d'adjacence est m, puis une fonction Python **diametre(m)** qui calcule le diamètre du graphe de matrice d'adjacence m.

Exemple : Notre graphe initial (celui qui n'était pas orienté) a un diamètre égal à 3, et vérifie par exemple **distance(4,2)=2**.