TP noté : corrigé

PTSI Lycée Eiffel

15 décembre 2016

I. Quelques algorithmes élémentaires sur les graphes.

Question 1. On peut faire extrêmement rapide ici, puisque la taille du graphe sera simplement le nombre d'éléments de la liste g :

```
def taille(g) :
    return len(g)
```

Question 2. On peut utiliser ici un algorithme similaire à celui vu en cours pour la recherche du maximum dans une liste. Il faut toutefois faire attention au fait que chaque élément de la liste g est ici un couple d'entier, il faut donc tester chacun des deux éléments de ce couple :

```
def ordre(g) :
    a=0
    for i in g :
        if i[0]>a :
            a=i[0]
        if i[1]>a :
            a=i[1]
    return a
```

Question 3. Il faut ici parcourir toutes les arêtes du graphe (donc tous les éléments de la liste g), et compter cette qui contiennent la valeur n, soit en première position, soit en deuxième position.

```
def degre(g,n) :
    a=0
    for i in g :
        if i[0]==n or i[1]==n :
            a=a+1
    return a
```

Question 4. Le programme est similaire à celui de la question précédente, mais cette fois-ci, au lieu d'incrémenter un compteur à chaque fois qu'on croise le sommet n dans une arête, on va créer une liste à laquelle on ajoutera, quand on croise le sommet n, le sommet auquel il est relié par l'arête.

```
def voisins(g,n) :
    l=[]
    for i in g :
        if i[0]==n :
            l.append(i[1])
        if i[1]==n :
            l.append(i[0])
return l
```

Question 5. Ce n'est pas optimal du tout mais on peut procéder de la façon suivante : on calcule successivement les degrés de chaque sommet (à l'aide de la fonction écrite précédemment) et on conserve le plus grand (par une méthode su style recherche d'un maximum dans une liste).

```
def degremax(g) :
    a=0
    n=ordre(g)
    for i in range(1,n+1) :
        b=degre(g,i)
        if b>a :
            a=b
return a
```

Pour ceux qui ont envie de faire un peu plus efficace, on peut se contenter de parcourir une seule fois la liste du graphe, en comptant le nombre de voisins de tous les sommets simultanément (qu'on met dans une liste), puis de chercher le maximum de toutes ces valeurs :

```
def degremaxbis(g) :
    n=ordre(g)
    l=[0 for i in range(n)]
    for i in g :
        l[i[0]-1]+=1
        l[i[1]-1]+=1
    d=0
    for i in l :
        if i>d :
            d=i
    return d
```

Question 6. Pour cette question, on va vérifier que tous les sommets (à partir du numéro 2) ont un ordre degré égal à celui du sommet numéro 1. On peut exploiter les spécifités du return de Python pour raccourcir légèrement le temps d'exécution : dès qu'on croise un sommet dont le degré est différent du de celui du sommet 1, on retourne False, ce qui arrête l'exécution de la fonction. Si par contre on arrive jusqu'au bout de la boucle, c'est que le graphe est régulier, on peut donc retourner True (cette valeur ne sera pas retournée s'il y a eu un False précédemment).

```
def regulier(g) :
    n=ordre(g)
    a=degre(g,1)
    for i in range(2,n+1) :
        if degre(g,i) !=a :
            return False
return True
```

II. Quelques algorithmes sur les matrices.

Question 7. Il faut ici vérifier que tous éléments de l (éléments qui sont eux-même des listes !) ont la même longueur, ce qu'on peut faire de la façon suivante :

```
def testmatrice(l) :
    a=len(l[0])
    for i in l[1 :] :
        if len(i) !=a :
```

```
                return False
        return True
```

Question 8. Si on suppose déjà que la matrice est carrée, il suffit de vérifier que le nombre de lignes (c'est-à-dire la longueur de la liste de listes) est le même que le nombre d'éléments de la première ligne. Si on veut tester au préalable que la matrice est carrée, on fait comme ceci :

```
def matcarree(l) :
    return (testmatrice(l) and len(l)==len(l[0]))
```

Question 9. On se contente de parcourir la matrice en comptant les zéros, en faisant juste attention au fait que c'est une liste de listes, ce qui nécessite une double boucle :

```
def zeros(m) :
    a=0
    for i in m :
        for j in i :
            if j==0 :
                a=a+1
    return a
```

Question 10. Ce n'est pas si terrible, surtout qu'on a déjà supposé les matrices carrées et de même taille. On crée une matrice de n lignes et n colonnes ne contenant que des 0, puis on remplit cette matrice case par case en utilisant la formule de l'énoncé :

```
def produit(a,b) :
    n=len(a)
    l=[[0 for i in range(n)] for j in range(n)]
    for i in range(n) :
        for j in range(n) :
            c=0
            for k in range(n) :
                c=c+a[i][k]*b[k][j]
            l[i][j]=c
    return l
```

Question 11. Il suffit de multiplier k fois de suite la matrice m par elle-même (là encore ce n'est pas optimal mais ça marche) :

```
def puissance(m,k) :
    l=m[ :]
    for i in range(k-1) :
        l=produit(l,m)
    return l
```

III. Matrice d'adjacence d'un graphe.

Question 12. Pour chaque numéro de ligne et chaque numéro de colonne, on vérifie que les coefficients situés sur la ligne i colonne j et sur la lisgne j colonne i sont les mêmes. Pour ne pas faire de vérification inutile, on ne vérifie que si le numéro de colonne est strictement supérieur au numéro de ligne (on supposera pour toutes les questions restantes qu'il y a des 0 sur la diagonale de la matrice) :

```
def nonoriente(m) :
    n=len(m)
```

```
    for i in range(n) :
        for j in range(i,n) :
            if m[j][i] !=m[i][j] :
                return False
    return True
```

Question 13. Pour déterminer la taille du graphe, on compte le nombre de coefficients de la matrice égaux à 1 (encore une fois en ne gardant que les numéros de colonne strictement supérieurs aux numéros de ligne pour ne pas compter deux fois chaque arête). Pour le nombre de voisins d'un sommet, il suffit de compter le nombre de 1 situés sur la ligne correspondant à ce sommet :

```
def taillebis(m) :
    a=0
    n=len(m)
    for i in range(n) :
        for j in range(i,n) :
            if m[i][j]==1 :
                a=a+1
    return a

def degrebis(m,n) :
    a=0
    for i in m[n-1] :
        if i==1 :
            a=a+1
    return a
```

Question 14. On crée une matrice à n lignes et n colonnes (n étant l'ordre du graphe) remplie initialement de zéros, puis on parcourt la liste du graphe, et à chaque arête croisée, on remplace l'élément correspondant de la matrice par un 1 :

```
def conversiongraphematrice(g) :
    n=ordre(g)
    m=[[0 for i in range(n)] for j in range(n)]
    for i in g :
        a,b=i[0],i[1]
        m[a-1][b-1]=1
        m[b-1][a-1]=1
    return m
```

Question 15. C'est presque plus facile dans l'autre sens : on crée une liste vide, et on parcourt la matrice, ajoutant la liste une arête à chaque fois qu'on croise un nombre 1 dans la matrice (en décalant pour tenir compte des conventions de l'énoncé concernant la numérotation des sommets d'un graphe).

```
def conversionmatricegraphe(m) :
    l=[]
    n=len(m)
    for i in range(n) :
        for j in range(i,n) :
            if m[i][j]==1 :
                l.append([i+1,j+1])
    return l
```

Question 16. On calcule les puissances de la matrice d'adjacence jusqu'à obtenir un coefficient différent de 0 à l'endroit souhaité :

```
def distance(m,i,j) :
    d=1
    a=m
    while a[i-1][j-1]==0 :
        d=d+1
        a=produit(a,m)
    return d
```

Pour calculer le diamètre, une méthode extrêmement brutale (et particulièrement mauvaise si on cherche à optimiser l'algorithme) consiste tout simplement à calculer toutes les distances possibles entre sommets du graphe et à conserver la plus grande (les plus motivés chercheront à faire mieux).

```
def diametre(m) :
    n=len(m)
    a=0
    for i in range(n) :
        for j in range(i,n) :
            d=distance(m,i,j)
            if d>a :
                a=d
    return a
```