CORRIGÉ DU DM 1

À rendre le 07/11/2024

Exercice 1

D'après CCINP (maths) 2023

1. La matrice produit C de taille $n \times n$ où n est le nombre de lignes et de colonnes de A et de B a comme coefficient $c_{ij} = \sum_{k=0}^{n-1} a_{ik} b_{kj}$, d'où le programme :

```
def produit(A, B):
    n = len(A)
    C = [[0 for i in range(n)] for j in range(n)]
    for i in range(n):
        for j in range(n):
            s = 0
            for k in range(n):
                s += A[i][k] * B[k][j]
        C[i][j] = s
    return C
```

2. Si la matrice est symétrique, alors le graphe n'est pas orientée, on compare donc les termes a_{ij} et a_{ji} pour i et j dans $\{1, \dots, n-1\}$ (on peut diviser le nombre de calculs par 2 et ne faire varier j que dans $\{0, \dots, i-1\}$ avec $i \in \{1, \dots, n-1\}$

```
def oriente(A):
    n = len(A)
    for i in range(n):
        for j in range(n):
            if A[i][j] != A[j][i]:
                return True
    return False
```

3. Le coefficient de la ligne i et de la colonne j de A^p (noté a_{ij}^p) donne le nombre de chemins de longueur p qui joignent le sommet i au sommet j en p étapes.

   On cherche donc la première valeur de p telle que a_{ij}^p soit non nul :

```
def distance(A,i,j):
    p = 1
    B = A
    while B[i][j] == 0:
        p += 1
        B = produit(A,B)
    return p
```

4.
```
SELECT id FROM CLIENTS WHERE ville = "Toulouse"
```

5.
```
SELECT CLIENTS.email FROM CLIENTS
JOIN PARTENAIRES ON PARTENAIRES.id client = CLIENTS.id
WHERE PARTENAIRES.partenaire = "SCEI"
```

6. On garde les notations $A = (a_{ij})$ et $A^p = (a_{ij}^p)$.
On procède par récurrence sur $p \geq 1$.

   - Pour $p = 1$, la propriété est vrai car si un chemin (de longueur 1) relie i à j alors $a_{ij} = 1$ et sinon $a_{ij} = 0$.
   - Supposons que la propriété soit vraie pour les chemins de longueur p (c'est à dire : pour tout (i, j), le nombre de chemins de longueur p joignant i à j est égal à a_{ij}^p).
   Soit i et j deux sommets. Nous allons compter les sommets de longueurs $p+1$ joignant i à j.
   Pour cela, nous effectuons une partition de l'ensemble de ces chemins selon le premier sommet k visité après i : notons C_k l'ensemble des chemins de longueur $p+1$ joignant i à j tels que le premier sommet visité après i soit k. Le nombre total de chemins de longueur $p+1$ joignant i à j est donc la somme des nombres d'éléments des C_k.
   Déterminons le nombre d'éléments dans chaque C_k.
   Si $a_{ik} = 0$, ce nombre d'élements est nuls.
   Si $a_{ik} = 1$, ce nombre d'élements est égal au nombre de chemins de longueur p joignant k à j.
   Dans les deux cas, ce nombre d'élément est égal à :

   $$a_{ik} a_{kj}^p \ ;$$

   donc le nombre total de chemins de longueur $p+1$ joignant i à j est égal à

   $$\sum_{k=1}^{n} a_{ik} a_{kj}^p \ ;$$

   c'est à dire à

   $$a_{ij}^{p+1} \ .$$

   Ainsi la propriété est vrai au rang $p+1$.
   - On en conclut que pour tout $p \geq 1$, le nombre de chemins de longueur p joignant i à j est égal à a_{ij}^p.

Exercice 2

D'après Banque PT 2016

Question 1 – Une clé primaire est un attribut qui permet de faire référence sans ambiguité à une ligne (un enregistrement) de la table.
Dans la table `patients` , l'attribut `numero_secu` aurait pu également servir de clé primaire.

Question 2 –

```
SELECT datetime , pdias , psyst , pouls
FROM mesures
WHERE datetime>time1 AND datetime<time2 ;
```

Question 3 –

```
SELECT datetime , pdias , psyst , pouls
FROM mesures
WHERE datetime>time1 AND datetime<time2 AND pid=id_patient AND type='tension' ;
```

Question 4 –

```
def analyse(valeurs):
    n=len(valeurs)
    val_min=valeurs[0]
    val_max=valeurs[0]
    S=valeurs[0]
    for i in range(1,n):
        val=valeurs[i]
        if val<val_min:
            val_min=val
        if val>val_max:
            val_max=val
        S+=val
    moy=S/n
    return val_min,val_max,moy
```

Question 5 – A chaque passage dans la boucle de notre fonction, on effectue deux comparaisons et une addition (que ce soit dans le meilleur ou le pire des cas). De plus à la sortie de cette boucle on effectue une division. Ceci fait en tout $3(n-1)+1$ opérations élémentaires (on aurait pu compter également les affectations, mais leur nombre est du même ordre).
La complexité de notre fonction est donc linéaire.

Question 6 – On effectue un tri sélection.

```
def tri_select(valeurs):
    n=len(valeurs)
    for k in range(n-1):
        # on cherche le minimum de valeurs[k:]
        # et on le place en position k
        i_min=k
        val_min=valeurs[k]
```

```
        for i in range(k+1,n):
            if valeurs[i]<val_min:
                val_min=valeurs[i]
                i_min=i
        valeurs[k],valeurs[i_min]=valeurs[i_min],valeurs[k]
    return valeurs
```

On s'appuie ensuite sur cette fonction pour déterminer la médiane :

```
def mediane(valeurs):
    n=len(valeurs)
    valeurs=tri_select(valeurs)#ou juste tri_select(valeurs)
    med=valeurs[(n-1)//2]
    return med
```

Question 7 – Pour le tri, on effectue une boucle de longueur $n-1$ avec, au k-ème passage, $n-1-k$ comparaisons (que ce soit dans le meilleur ou le pire des cas). On a donc en tout un nombre de comparaisons égal à :

$$\sum_{k=0}^{n-2} n-1-k = \frac{n(n-1)}{2} .$$

La fonction médiane consiste ensuite en un nombre finis d'opérations. Donc la complexité globale est de l'ordre de $\mathcal{O}(n^2)$.

On peut obtenir un algorithme plus efficace en utilisant un algorithme de tri en $\mathcal{O}(n \ \ln n)$, comme le tri fusion.

Remarque : si l'on choisissait un tri insertion, le pire des cas était en $\mathcal{O}(n^2)$ et le meilleur en $\mathcal{O}(n)$.

Question 8 –

```
SELECT DISTINCT nom , prenom , telephone
FROM patients AS P JOIN mesures AS M
     ON P.id = M. pid
WHERE M.psyst>160 AND M.pdias>110 AND M.pouls>100 AND M.pouls<150 ;
```

Remarque : le `DISTINCT` n'est pas indispensable mais permet que l'on obtienne une seule fois le nom... de chaque patient dans cette situation.

Question 9 –

```
SELECT pid,MIN(pouls)
FROM mesures
GROUP BY pid ;
```