INFORMATIQUE MP CONCOURS BLANC

03 MARS 2017

PROBLÈME - ORGANISATION DE FICHIERS

Partie A : Modélisation

Petit flou d'énoncé pour les fonctions : liste ou TS en argument ?

1. La fonction compte le nombre de secteurs marqués 0, les secteurs libres.
   Elle renvoie alors ce nombre multiplié par la taille d'un secteur : c'est la taille libre dans le disque.
2. ```
   def nombreZerosDroite(i, liste):
       """ Entrée : un indice et une table des secteurs
           Sortie : le nombre de secteurs libres à droite de i"""
       n  = len(liste)
       nombre = 0
       position = i
       while position position < n and liste[position] == 0 :
           nombre += 1
           position += 1
       return nombre
   ```
3. Dans le pire cas la liste ne contient que des 0, on fait un test pour chaque entier de i à $n-1$: il y en a $n-i$.
4. ```
   def libreMax(liste):
       """ Entrée : une table des secteurs
           Sortie : la taille maximale libre en un seul bloc
                    et l'indice du début"""
       n  = len(liste)
       nbMax = 0
       ind = 0
       for i in range(n):
           l = nombreZerosDroite(i, liste)
           if l > nbMax:
               nbMax = l
               ind = i
       return nbMax*taille_secteur, ind
   ```
5. Pour i variant de 0 à $n-1$ on fait $n-i$ tests. Le nombre total est donc $\sum_{i=0}^{n-1}(n-i) = \sum_{k=1}^{n} k = \frac{n(n+1)}{2} = O(n^2)$.
6. ```
   def libreMaxIndMieux(liste):
       """ Entrée : une table des secteurs
           Sortie : la taille maximale libre en un seul bloc
                    et l'indice du début"""
       n  = len(liste)
       nbMax = 0
       ind = 0
       position = 0
       while position < n:
           l = nombreZerosDroite(position, liste)
           if l > nbMax:
               nbMax = l
               ind = position
           position  = position + l + 1
       return nbMax*taille_secteur, ind
   ```
   On avance la position à droite du dernier indice testé. On ne test chaque case de `liste` une seule fois, on a donc un complexité de $O(n)$.
   ```

Partie B : Position des fichiers

On peut remarquer que chaque fichier est écrit dans plusieurs secteurs mais qu'il existe un, et un seul, pour lequel la table des secteurs renvoie -1, le dernier.

1. ```
   def nombreFichiers(TS):
       """ Entrée : une table des secteurs
           Sortie : le nombre de fichiers inclus."""
       n = len(TS)
       nbFichiers = 0
       for i in range(n):
           if T[i] == -1:
               nbFichiers = nbFichiers +1
       return nbFichiers
   ```
2. ```
   def coherence(TS,TF):
       return nombreFichiers(TS) == len(TF)
   ```
3. ```
   def depart(nom, TF):
       """ Entrée : un nom et une table des fichiers
           Sortie : l'adresse du début du fichier"""
       n = len(TF)
       for i in range(n):
           couple = TF[i]
           if couple[0] == nom:
               return couple[1]
       return -1
   ```
4. ```
   def places(nom, TS, TF):
       position = depart(nom, TF)
       liste = []
       while position != -1:
           liste.append(position)
           position  = TS[position]
       return liste
   ```

Partie C : Tri des fichiers

Dans l'interface utilisateur il est important de pouvoir trier les fichiers selon leur taille ou selon leur nom. On propose d'abord de construire un tri fusion.

1. Écrire une fonction `fusion(L1,L2)` prenant en entrée deux listes d'éléments comparables avec `<=`, supposées triées dans l'ordre croissant, et **renvoyant** une nouvelle liste composée des mêmes éléments, triés. La fonction doit avoir une complexité linéaire en le nombre d'éléments présents dans les deux listes.

   ```
   def fusion(L1,L2):
       L_fusion=[]
       i,j=0,0
       while (i<len(L1)) and (j<len(L2)):
           if L1[i]<L2[j]:
               L_fusion.append(L1[i])
               i+=1
           else:
               L_fusion.append(L2[i])
               j+=1
       if len(L1)==i:
           L_fusion+=L2[j:]
       else:
           L_fusion+=L1[i:]
       return L_fusion
   ```

2.
```
def tri_fusion(L):
      n=len(L)
      if n==1:
          return L
      if n>1:
          L1=L[0:n//2]
          L2=L[n//2:n]
          L1=tri_fusion(L1)
          L2=tri_fusion(L2)
          return fusion(L1,L2)
```
3. Pour une liste de taille p la complexité est $O(p\log(p))$
4.
```
FC = []
n = len(TF)
for i in range(n) :
    nom = TF[i][0]
    taille = len(places(nom,TS,TF))*taille_secteur
    FC.append( (nom,taille))
\end{enumerate}
```
5.
```
def ordre(a,b,typ=0):
          return( a[typ] <= b[typ]) # Hahaha
```

Partie D : Base de données

On considère une base de données contenant les tables suivantes dont des extraits sont donnés :

fichiers		
iden	nom	taille
A0EF	"essai.txt"	2560
C2DC	"essai.log"	512
016D	"correction.pdf"	4096

etat		
idfic	**type**	createur
A0EF	libre	Goku
C2DC	libre	Gohan
016D	restreint	Goku

1. Écrire une requête SQL renvoyant les noms de fichiers créés par Goku.

   ```
   SELECT nom FROM fichiers JOIN etat ON iden = idfic WHERE createur = 'Goku'
   ```

2. Écrire une requête SQL donnant le nom du fichier de taille maximale.

   ```
   SELECT nom , MAX(taille) FROM fichiers
   ```

   Ne réponds pas exactement à la question, contrairement à

   ```
   SELECT nom  FROM fichiers
   WHERE taille $ (SELECT MAX(taille) FROM fichier)
   ```

3. Écrire une requête SQL renvoyant la somme des tailles de fichiers de type restreint.

   ```
   SELECT SUM(taille) FROM fichiers JOIN etat ON iden = idfic WHERE type = 'restreint'
   ```

EXERCICE - UN PEU DE CHIMIE

1. Préciser un vecteur X et une fonction mathématique f tel que le système différentiel s'écrive sous la forme $\frac{dX}{dt} = f(X)$.
   On prends le vecteur $X = (A, B, C)$ de $\mathbb{R}^3$ et on considère $f : \mathbb{R}^3 \to \mathbb{R}^3$ définie par

$$f(x, y, z) = (-dx, dx - ey, ey)$$

2.
```
import numpy as np

def f(X):
    """ Fonction definissant l'equation differentielle"""
    global d,e
    A,B,C = X
    return( np.array([-d*A**2 , d*A**2-e*B**2,e*B**2]))


# Paramètres
tmax = 200
d = 0.03
e = 0.02
X0 = np.array([10,0,0])

h = 0.01
N = int(tmax/h)

t = 0
X = X0
tt = [t]
XX = [X]

# Euler
for i in range(N):
    t = t + h
    X = X + h*f(X)
    tt.append(t)
    XX.append(X)
```

3. Il suffit de mettre le `while` suivant à la place du `for`

```
while X[2] < 0.9*X0[0] :
```

4.
```
import matplotlib.pyplot as pl

XX= np.array(XX)
pl.plot(tt,XX[:,0])
pl.plot(tt,XX[:,1])
pl.plot(tt,XX[:,2])
pl.show()
```