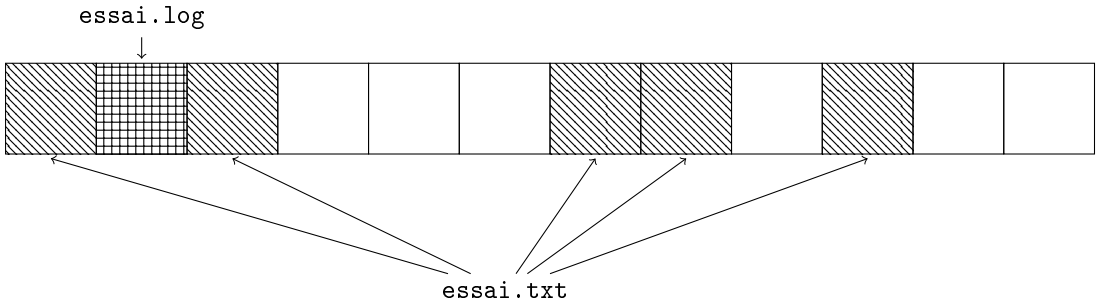
INFORMATIQUE MP CONCOURS BLANC

03 MARS 2017

PROBLÈME - ORGANISATION DE FICHIERS

Nous décrivons ici une version simplifiée de l'organisation d'un disque selon le système FAT (file allocation table).
Un disque est divisé en secteurs de taille fixe, par exemple chaque secteur occupe 512 octets. Quand on enregistre un fichier sur le disque il occupe un nombre entiers de secteurs : un fichier de 12540 octets occupera donc 25 secteurs soit un espace total de 12800 octets. Un fichier de 10 octets occupera un secteur entier. Voici un exemple d'occupation d'un disque très petit (6144 octets) ; chaque rectangle représente un secteur.



La fin du disque est occupée par deux tableaux de taille fixe. L'un décrit l'occupation de chaque secteur, l'autre indique les noms des fichiers présents sur le disque ainsi que leur emplacement.
Dans le deuxième tableau chaque fichier est décrit par son nom, la position du premier secteur et d'autres renseignements.

Partie A : Modélisation

Les secteurs sont indicés par un entier `i` : le premier secteur est indicé par 0.
Le tableau des fichiers sera représenté dans ce problème par une liste `TF` dont les éléments sont des couples formés du nom du fichier sous forme d'une liste de caractères et de l'entier indiquant le premier secteur où est stocké le fichier.
La première composante du couple, le nom du fichier, est accessible par `TF[i][0]` et le second, le secteur de départ, par `TF[i][1]`.
Dans l'exemple ci-dessus on pourra avoir

```
TF = [("essai.log", 1), ("essai.txt", 0)]
```

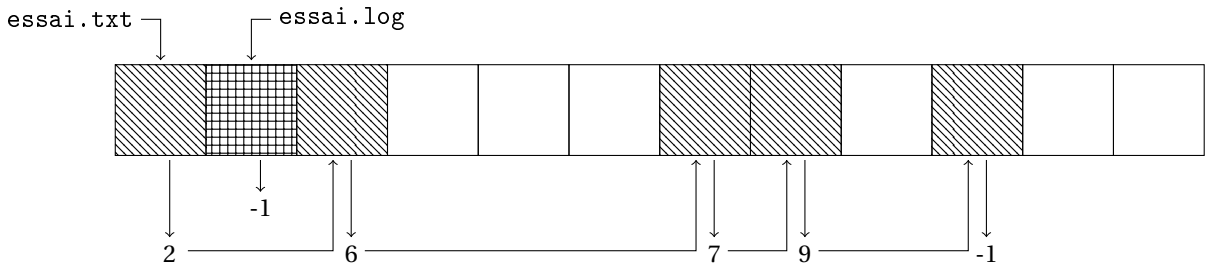
ainsi `TF[0][0]` renvoie "essai.log" et `TF[1][1]` renvoie 0.

Le tableau des secteurs sera représenté dans ce problème par une liste d'entiers `TS`.

- Si le secteur d'indice i n'est pas occupé alors `TS[i]` contient la valeur 0.
- Si le secteur d'indice i est occupé par un fichier alors deux cas peuvent se produire
  - s'il reste des données à stocker lorsque le fichier a rempli le secteur i alors `TS[i]` contient la valeur du secteur suivant,
  - si le fichier est complètement stocké, c'est-à-dire si le secteur d'indice i est le dernier secteur contenant le fichier, alors `TS[i]` contient la valeur -1.

Dans l'exemple ci-dessus on aura

```
TS = [2, -1, 6, 0, 0, 0, 7, 9, 0, -1, 0, 0 ]
```



Ainsi "essai.txt" commence à être stocké au secteur 0, la suite est stockée dans le secteur `2 = TS[0]`, puis dans le secteur `6 = TS[2]`, puis dans les secteurs 7 et 9 où stockage est terminé car `TS[9]` vaut -1.
"essai.log" est stocké dans le secteur 1 et uniquement celui-ci car `TS[1]` vaut -1.
Dans le problème on supposera définie une variable globale `taille_secteur = 512`.

1. Que fait la fonction suivante ?

```
def inconnue(TS):
    """ Entrée : une table des secteurs
        Sortie : ???.
    n = len(TS)
    compteur = 0
    for i in range(n):
        if TS[i] == 0:
            compteur = compteur +1
    return compteur*taille_secteur
```

Dans la mesure du possible, pour améliorer la vitesse d'accès aux secteurs, le système essaie d'écrire un fichier dans des secteurs contigus c'est-à-dire dans une suite de secteurs libres placés les uns après les autres. On souhaite donc connaître la plus grande suite de secteurs libres, c'est-à-dire la plus longue suite de 0 dans la liste `TS`.

2. Construire une fonction `nombreZerosDroite(i, liste)` qui prenne comme paramètres un entier `i` et une liste `TS` et renvoie le nombre de cases contiguës du tableau contenant 0 à droite de la case d'indice i, cette case étant comprise.

   Dans l'exemple précédent `nombreZerosDroite(1, TS)` renverra 0, `nombreZerosDroite(3, TS)` renverra 3 et `nombreZerosDroite(10, TS)` renverra 2.

3. On s'intéresse à la complexité du nombre de comparaisons. Déterminer le pire cas de `liste` et le nombre de comparaisons dans ce cas pour la fonction `nombreZerosDroite(i, liste)`.

On remarque que le nombre maximal de zéros consécutifs est le résultat de `nombreZerosDroite(i, TS)` quand i est le premier indice de la portion de la liste qui contient le maximum de zéros.
On en déduit une fonction du calcul de la taille maximale en secteurs contigus.

```
def libreMax(liste):
    """ Entrée : une table des secteurs
        Sortie : la taille maximale libre en un seul bloc"""
    n  = len(liste)
    nbMax = 0
    for i in range(n):
        l = nombreZerosDroite(i, liste)
        if l > nbMax:
            nbMax = l
    return nbMax*taille_secteur
```

4. Modifier la fonction précédente en `libreMaxInd(TS)` qui renvoie, en plus de la taille maximale, l'indice du premier secteur libre dans une portion maximale. Dans le cas où tous les secteurs sont occupés cet indice peut être quelconque.
5. Déterminer la compléxité en comparaisons dans le pire des cas pour `libreMaxInd(TS)`.
6. Proposer une fonction `libreMaxInd1(TS)` qui renvoie la taille maximale et l'indice du premier secteur libre dans la portion maximale dont la complexité maximale soit de $O(n)$ tests pour une liste de taille 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. Écrire une fonction qui reçoit comme paramètre une table des secteurs et renvoie le nombre de fichiers indiqués comme occupant au moins un secteur.
2. En déduire une fonction `coherence(TS,TF)` qui renvoie `True` ou `False` selon que les les deux tables (des secteurs et des fichiers) indiquent le même nombre de fichiers.
3. Écrire une fonction `depart(nom, TF)` qui renvoie le secteur de départ d'un fichier donné par son nom (une chaîne de caractères) en explorant la liste `TF`.
   La fonction devra renvoyer -1 si le fichier n'est pas présent.
   Dans l'exemple `depart("essai.log", TF)` renverra 1 et `depart("perdu.txt", TF)` renverra -1
4. Écrire une fonction `places(nom, TS, TF)` qui renvoie la liste des secteurs occupés par un fichier donné par son nom.
   La fonction devra renvoyer la liste vide si le fichier n'est pas présent.

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.
2. En déduire une fonction `tri_fusion(L)` prenant en entrée une liste d'éléments et **renvoyant** une nouvelle liste, composée des mêmes éléments, triés dans l'ordre croissant, suivant le principe :
   — si `L` possède au plus 1 élément, on renvoie la même liste
   — sinon, on découpe `L` en deux morceaux, on trie récursivement les deux morceaux, et on fusionne le résultat, qu'on renvoie.
3. Donner sans démonstration la complexité du tri pour une liste de taille p.

On souhaite construire à partir des listes `TF` et `TS` une liste `FC = [ ("essai.log",512),("essai.txt",2560),...]` qui contienne les couples (nom du fichier, poids du fichier).

4. Construire cette liste `FC`

Pour généraliser le tri fusion, il suffit de remplacer dans le code le symbole `<=` par une fonction `ordre(a,b)` que l'on donnera en paramètre. C'est cette fonction qui va permettre de trier selon les noms ou les tailles.

5. Construire une fonction `ordre(a,b,type=0)` prenant en argument des couples de type `(chr,int)` renvoyant :
   - Si `type` vaut 0 : `True` si le premier élément de `a` est plus petit que le premier élément de `b` pour l'ordre lexicographique.
   - Si `type` vaut 1 : `True` si le deuxième élément de `a` est plus petit que le deuxième élément de `b` pour l'ordre sur les entiers.

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	etat	createur
A0EF	libre	Goku
C2DC	libre	Gohan
016D	restreint	Goku

La table `fichiers` contient les colonnes :
- iden : l'indentifiant du fichier : un nombre de 4 chiffres en hexadécimal
- nom : le nom du fichier

- taille : le poids en octets du fichier

La table `etat` contient les colonnes :
- idfic : l'identification du fichier
- etat : si le fichier est consultable
- fichier : le nom du créateur du fichier

1. Écrire une requête SQL renvoyant les noms de fichiers créés par Goku.
2. Écrire une requête SQL donnant le nom du fichier de taille maximale.
3. Écrire une requête SQL renvoyant la somme des tailles de fichiers de type restreint.

EXERCICE - UN PEU DE CHIMIE

Pour arrondir leurs fins de mois, deux professeurs de Physique de CPGE d'un lycée de centre ville de Blois décident d'utiliser le laboratoire de chimie pour synthétiser des substances interdites. Le sujet étant sensible et afin de ne pas donner des idées, les noms des composants chimiques seront remplacés par des lettres. La production se fait à partir d'une solution contenant un réactif A. Sous l'action de la chaleur, une réaction chimique se met en place, transformant successivement le réactif A en un réactif B, puis en un réactif C (B et C ne sont pas présents au temps $t = 0$ du début de la réaction). Les équations régissant les concentrations des réactifs sont les suivantes :

$$\begin{cases} \frac{d[A]}{dt} &= -d[A]^2 \\ \frac{d[B]}{dt} &= d[A]^2 - e[B]^2 \\ \frac{d[C]}{dt} &= e[B]^2 \end{cases}$$

où d et e sont deux coefficients qui dépendent notamment de la température et de la circulation d'air. On souhaite évaluer l'évolution des concentrations des réactifs A, B et C en fonction du temps. Pour cela, on va utiliser la méthode d'Euler.

1. Préciser un vecteur X et une fonction mathématique f tel que le système différentiel s'écrive sous la forme $\frac{dX}{dt} = f(X)$.
2. Le programme suivant a été écrit en Python. Compléter la fonction `f(X)` et `# Euler`.

```
import numpy as np

def f(X):
    """ Fonction définissant l'équation différentielle"""
    global d,e
    # A compléter

# 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 = # A compléter
    tt.append(t)
    # A compléter
```

3. La procédure de fabrication se termine lorsque la concentration du réactif C dépasse de 90 % la concentration initiale du réactif A. Modifier le programme précédent afin d'obtenir le temps total de la réaction.
4. Afin de construire un graphique avec les courbes des concentrations de A, B et C en fonction du temps, compléter le code suivant :

```
import matplotlib.pyplot as pl

XX= np.array(XX) # afin de pourvoir extraire les colonnes
#  A compléter
pl.show()
```