Université de Lille — Faculté des Sciences et Technologies — Département Informatique

L1 MASS/SESI/PEIP - 2017–2018

Algorithmes et Programmation 1

DS2 - 14/04/2018

durée 2h - documents non autorisés (sauf memento Python3)

Ce devoir contient trois exercices indépendants. Indiquez sur la copie votre parcours et le numéro de votre groupe. Sauf mention explicite du contraire, il n'est pas nécessaire de documenter les fonctions qui vous sont demandées.

Exercice 1 *Compréhension de code*

Voici le code d'une fonction nommée `mystere` :

```
def mystere(l, k):
    for i in range(1, len(l)):
        j = i - 1
        while j >= 0 and k[l[j]] > k[l[j+1]]:
            l[j], l[j+1] = l[j+1], l[j]
            j = j - 1
```

Question 1 Dans cette question uniquement, on considère que l'on a `l = ['A', 'T', 'G', 'G', 'C']` et `k = {'A':8, 'C':9, 'G':5, 'T':13}`.

Tracez l'appel `mystere(l, k)` sous forme d'un tableau donnant les valeurs des variables `i`, `j` et `l`.

Dans toute la suite, on suppose que `l` est une liste quelconque et `k` un dictionnaire.

Question 2 Quelles conditions doit respecter `k` pour que l'appel `mystere(l, k)` s'exécute sans déclencher d'exception ? Précisez les exceptions que l'on peut rencontrer si ces conditions ne sont pas satisfaites.

Question 3 Rédigez une documentation de la fonction `mystere`. (il n'est pas demandé de doctest).

Exercice 2 *Fichiers CSV*

Dans cet exercice on travaille avec des fichiers textes au format CSV dont les lignes sont de la forme

```
ETU001;TALON;Achille;PEIP15;12;15;17;18;11
ETU002;TALON;Eugénie;MASS1;14;13;11;;11
ETU003;NESTOR;Timoléon;PEIP15;11;10;10
ETU004;CALBUTH;Raymond;SESI11;17;14;
...
```

Le premier champ est un identifiant de la forme `ETUxxx`, les `xxx` étant une représentation sur trois chiffres d'un nombre entier. On suppose que le fichier ne contient pas deux lignes avec le même identifiant. (Il ne peut donc pas y avoir d'informations pour plus de 1000 étudiants dans le fichier).

Les deuxième et troisième champs donnent le nom et le prénom de l'étudiant, le quatrième son groupe de TD, et les champs suivants ses notes dont on pourra supposer qu'elles sont toutes entières. On peut remarquer sur l'exemple que tous les étudiants n'ont pas nécessairement le même nombre de notes.

Les champs sont tous séparés par un point-virgule (`;`).

Question 1 Réalisez une fonction `lire_etudiants`, paramétrée par le nom du fichier, et permettant de lire l'intégralité d'un tel fichier et renvoyant un dictionnaire dont les clés sont les identifiants, et les valeurs associées sont des quadruplets (tuple) de la forme `(nom, prenom, groupe, notes)`, `nom`, `prenom` et `groupe` étant des chaînes de caractères, et `notes` une liste de nombres entiers.

En supposant que le fichier `test.csv` contient les quatre lignes décrites ci-dessus et uniquement celles là on a

```
>>> etu = lire_etudiants('test.csv')
>>> len(etu)
4
>>> etu['ETU001']
('TALON', 'Achille', 'PEIP15', [12, 15, 17, 18, 11])
```

On suppose que tous les étudiants figurant dans le fichier de données possède au moins une note. La moyenne qui lui est alors associée est la moyenne arithmétique de l'ensemble des notes, c'est-à-dire la somme des notes divisées par le nombre de notes.

Question 2 Réalisez une fonction `moyenne_etudiant` paramétrée par un identificateur d'étudiant, et un dictionnaire du type précédent, qui renvoie la moyenne calculée à partir des notes de l'étudiant.

```
>>> moyenne_etudiant('ETU004', etu)
15.5
```

Question 3 Réalisez la fonction `classer_etudiants` paramétrée par un dictionnaire du type précédent, qui renvoie la liste de quadruplets de la forme `(id, nom, prenom, moyenne)` classée par ordre décroissant des moyennes.

Question 4 Réalisez une procédure `produire_resultats` paramétrée par deux noms de fichiers :
- le premier étant celui d'un fichier de données tel que celui décrit au départ
- le second étant celui du fichier de résultats à produire, ce fichier étant aussi au format CSV avec des lignes de la forme `id;nom;prenom;moyenne`, les lignes étant classées par ordre décroissant de moyenne.

Exercice 3 *Listes extraites croissantes de longueur maximale*

Dans cet exercice, on présente un algorithme qui permet de déterminer, la longueur maximale d'une liste extraite croissante. C'est–à–dire, la longueur de la plus grande liste triée qu'on peut réaliser à partir d'une liste donnée en ne choisissant de ne garder que certains éléments de cette liste dans le même ordre. Des exemples seront donnés plus loin dans le sujet.

Dans la suite, `li` désigne une liste homogène d'éléments comparables.

On appelle *liste extraite* d'une liste `li`, la donnée d'un couple dont la première composante est la liste `li` et dont la seconde composante est une liste strictement croissante d'entiers compris au sens large entre `0` et `len(li) - 1`.

Par exemple, donnons à `li` la valeur `[3, 1, 4, 1, 5, 9]`.
- `(li, [0, 2, 4])` est une liste extraite de `li`,
- `(li, [])` est aussi une liste extraite de `li`,
- `(li, [0, 0, 1])` n'est pas une liste extraite de `li`,
- `(li, [0, 2, 1])` n'est pas une liste extraite de `li`.

Question 1 Réalisez un prédicat `est_liste_extraite` paramétré par une liste extraite `c` (donc un couple) et une liste `li` qui teste si le couple `c` est bien une liste extraite de la liste `li`.

La *longueur* d'une liste extraite est la longueur de la liste située en deuxième position du couple.

Question 2 Réalisez et documentez la fonction `longueur`.

Le i^e élément d'une liste extraite `(li, ind)` est `li[ind[i]]`.

Question 3 Réalisez une fonction `extraction` parametrée par une liste extraite et dont le résultat est la liste obtenue en prenant les i^e éléments de cette liste extraite pour i parcourant l'intervalle entier allant de 0 à la longueur de la liste extraite diminuée de 1 au sens large.

On dit qu'une liste extraite est croissante lorsque l'extraction de la liste extraite donne une liste triée.
- `([3, 1, 4, 1, 5, 9], [1, 2])` est une liste extraite croissante,
- `([3, 1, 4, 1, 5, 9], [1, 2, 3])` n'est pas une liste extraite croissante.

Question 4 Réalisez un prédicat nommé `est_liste_extraite_croissante` qui teste si une liste extraite est croissante, en utilisant `extraction` et le prédicat `est_trie` vu en cours.

Voici un théorème admis :

Pour toute liste `li`, il existe un nombre `lmax`, il existe une liste extraite de `li` croissante de longueur `lmax` et toute autre liste extraite de `li` croissante possède une longueur inférieure ou égale à `lmax`.

On appelle ce nombre `lmax` la *longueur maximale des listes extraites croissantes de `li`*.

Pour la liste `[3, 1, 4, 1, 5, 9]`, la longueur maximale est 4. En effet `([3, 1, 4, 1, 5, 9], [1, 3, 4, 5])` et `([3, 1, 4, 1, 5, 9], [0, 2, 4, 5])` sont deux listes extraites croissantes de longueur 4, et il n'en existe pas de plus longues.

Question 5 La longueur maximale d'une liste extraite croissante d'une liste non vide peut elle être nulle ?

Question 6 Que dire de la liste si la longueur maximale des listes extraites croissantes vaut 1 ? `len(li)` ?

Question 7 En utilisant la recherche dichotomique, réalisez une fonction `rech_dicho` paramétrée par
- un élément `x`,
- une liste `li` d'éléments comparable à `x` telle que la liste `[li[e] for e in u]` soit triée,
- une liste `u` d'indices compris entre 0 et `len(li)-1`

et renvoyant l'entier
- -1 si `u` est vide ou bien si `li[u[0]]` $>$ `x`,
- un indice `p` ≥ 0 tel que :
  - `li[u[p]]` $\leq$ `x`,
  - si `p` $<$ `len(u)` -1, `li[u[p+1]]` $>$ `x`.

Pour déterminer la longueur maximale des listes croissantes extraites d'une liste `li` on va déterminer une liste croissante extraite de longueur maximale, grâce à l'algorithme suivant :

Entrée : li une liste de longueur n.
Sortie : renvoie un couple (li, ind) où ind est de longueur maximale.

1: initialiser p, u et ind avec la liste vide
2: **pour** i variant de 0 à $n-1$ **faire**
3: $j \leftarrow rech\_dicho(li[i], li, u)$
4: **si** $j = -1$ **alors**
5: $v \leftarrow -1$
6: **sinon**
7: $v \leftarrow u[j]$
8: **fin si**
9: ajouter en fin de liste p la valeur v
10: **si** $(j+1) < len(u)$ **alors**
11: **si** $li[u[j+1]] > li[i]$ **alors**
12: $u[j+1] \leftarrow i$
13: **sinon**
14: ajouter en fin de liste u la valeur i
15: **fin si**
16: **fin si**
17: **fin pour**
18: $courant \leftarrow$ le dernier élément de u
19: **pour** i variant de 0 à $len(u)-1$ **faire**
20: ajouter l'élément $courant$ en tête de la liste ind
21: $courant \leftarrow p[courant]$
22: **fin pour**
23: **renvoyer** (li, ind)

Question 8 Réalisez la fonction `liste_extraite_croissante_maximale` qui implante cet algorithme.

Question 9 Justifiez que dans cet algorithme la longueur de la liste u ne dépasse jamais celle de li.

Question 10 Montrez que le nombre de comparaisons d'éléments effectuées dans le pire des cas lors de l'exécution de l'algorithme précédent est majoré par $n(\lceil \log_2(n) \rceil + 2)$, n désignant la longueur de la liste `li`.
