Séance 6: Listes d'entiers aléatoires; compression de textes

Université Paris-Diderot

Objectifs:

- Manipuler des listes unidimensionnels d'entiers et de chaînes de caractères.
- Trier une liste d'entiers par comptage.

Exercice 1 (Listes d'entiers aléatoires, ⋆)

Le but de cet exercice est de mettre en œuvre des opérations simples sur des listes d'entiers, aboutissant à une méthode de tri particulièrement simple et efficace pour des listes contenant des entiers "pas trop grands". Vous mettrez vos solutions pour cet exercice dans un fichier `sortRandomList.py` *et vos tests seront à placer à la fin de ce fichier. On rappelle que pour tester si deux listes* `l1` *et* `l2` *sont égales (c'est-à-dire elles ont la même taille et le même contenu), vous pouvez utiliser l'opérateur* `==`*. Ensuite pour afficher le contenu d'une liste* `l`*, vous pouvez faire* `print (l)`*.*

1. *Écrire une fonction* `createRandomList (n)` *qui renvoie une liste d'entiers de taille* `n`*, dont les cases sont remplies par des entiers aléatoires compris entre* `0` *et* `(n-1)`*. (**Rappel :** Vous pouvez faire appel à la fonction* `randrange(a,b)` *de la bibliothèque* `random` *qui renvoie un entier compris entre* `a` *inclus et* `b` *exclus).*
2. *Écrire une fonction* `minMaxAverage (a)` *qui renvoie une liste de taille 3 contenant au premier indice le minimum, au deuxième le maximum et dans le troisième la partie entière de la moyenne des éléments de la liste d'entiers* `a` *donnée en paramètre.*

   Contrat:
   L'appel
   ```
   print (minMaxAverage (createRandomList (100)))
   ```
   peut afficher, par exemple,
   ```
   [0, 96, 46]
   ```
   Les valeurs exactes sont imprévisibles pour un tel appel, à cause du remplissage aléatoire.
3. *Écrire une fonction* `occurrences (a)` *qui renvoie une liste contenant, à l'indice* `i`*, le nombre d'occurrences de* `i` *dans la liste* `a`*. On supposera que les entiers contenus dans* `a` *sont positifs ou nuls.* ***Remarque :*** *La taille de la liste renvoyée par* `occurrences(a)` *est* `1+(minMaxAverage (a))[1]`*.*

   Contrat:
   Si `a` *est la liste* `[1,3,0,0,0,1]`*,* `occurrences (a)` *renvoie* `[3,2,0,1]`*,*
4. *Une fois obtenue la liste* `occurrences(a)`*, il est possible de trier*[1] `a` *très simplement : en commençant à l'indice* `0`*, on insère autant de* `0` *qu'indiqué dans la première case de liste des occurrences, puis autant*

1. Trier une lsite d'entiers a consiste en la construction d'une liste qui contient les mêmes éléments que a, disposés en ordre croissant : l'entier contenu dans la case d'indice i n'est pas plus grand que l'entier contenu dans la case d'indice (i+1), pour tout indice.

de `1` *qu'indiqué dans sa deuxième case et ainsi de suite. Appliquée à l'exemple précédente, cette méthode produit la liste* `[0, 0, 0, 1, 1, 3]` *(d'abord trois* `0`*, puis deux* `1`*, puis zéro* `2`*, puis un* `3`*), qui est bien ce qu'on voulait.*

(a) *Écrire une fonction* `countingSort( a)`*, qui utilise* `occurrences`*, et renvoie une liste triée contenant les mêmes éléments que* `a` *en utilisant l'algorithme décrit ci-dessus*[2]*.*

(b) *Écrire une procedure* `countingSort2 (a)`*, qui utilse le même procédé que* `countingSort` *pour trier* `a` sur place *: le contenu de* `a` *change suite à l'appel* `countingSort2(a)`*, c'est-à-dire que l'on ne créée pas de nouvelles listes mais on change la liste* `a` *directement.*

Contrat:

L'exécution de :

```
a = createRandomList (100)
b = countingSort (a)
countingSort2(a)
print (a == b)
```

affiche `true`*.*

□

Exercice 2 (Compression de textes, ⋆)

Dans cet exercice, vous allez mettre en œuvre un algorithme simple de compression de textes. Un texte peut être vu comme une liste de chaînes de caractères `lis`*, contenant un mot par case. La compression se fait en deux étapes : on construit d'abord une liste de chaînes de caracterès* `lex` *qui contient tous les mots du texte, sans répétitions. On appellera cette liste le* lexique *du texte. Ensuite, le texte est encodé par une liste d'entiers, de même longueur que* `lis`*, contenant à la case d'indice* `i` *l'indice du mot* `lis [i]` *dans liste* `lex`*. Par exemple, le texte* `["être", "ou", "ne", "pas", "être"]` *a comme lexique la liste* `["être", "ou", "ne", "pas"]`*, et comme code la liste* `[0, 1, 2, 3, 0]`*.*

Vous aller compléter le fichier `compression.py`*. Les tests seront à placer à la fin du fichier. Vous disposez aussi de la fonction* `loadText()` *qui renvoie une liste contenant le contenu du fichier* `text.txt` *avec un mot par indice. Vous utiliserez cette dernière fonction pour tester vos réponses.*

Construction du lexique

1. *Écrire une fonction* `isIn (s, lex)`*, qui vérifie si la chaîne de caractères* `s` *est dans la liste* `lex`*.*
2. *Écrire une fonction* `extendLexicon ( s, lex)`*, qui renvoie une liste de taille* `len (lex) + 1`*, qui contient* `s` *dans sa dernière case, et qui contient la chaîne* `lex[i]` *à l'indice* `i`*, pour* $0 \leq$ `i` $<$ `len (lex)`*.*
3. *Pour construire le lexique d'un texte* `lis`*, on procède comme suit :*
   — *On crée une liste* `lex` *de taille* 0 *(qui correspond donc à une liste vide).*
   — *Pour chaque case* `i` *de* `lis` *on vérifie si* `lis[i]` *est dans* `lex`*. Si ce n'est pas le cas, on étend* `lex`*, par une affectation* `lex = extendLexicon (lis[i], lex)`
   — *On renvoie* `lex`

   Écrire une fonction `buildLexicon (lis)`*, qui construit le lexique du texte contenu dans la liste de chaînes de caractères* `lis`*, en mettant en œuvre l'algorithme donné ci-dessus.*

Contrat:

L'exécution de :

```
text = loadText()
```

2. Pour les listes engendrées par `createRandomList (n)` ce procédé est efficace car elles ne contiennent que des entiers allant de `0` à `(n-1)`. Pour des listes quelconques, la taille potentiellement très grande de la liste des occurrences peut rendre cette méthode très inefficace.

```
print (len (text) + " " + len (buildLexicon (text))
```

affiche

```
1002 521
```

Codage et Décodage

1. *Écrire une fonction* `getCode (s, lex)`*, qui renvoie le premier indice de la chaîne de caractères* `s` *dans la liste* `lex`*, ou* `-1` *si* `s` *n'est pas dans* `lex`*.*
2. *Écrire une fonction* `code (lis, lex)`*, qui renvoie l'encodage du texte contenu dans* `lis` *relativement au lexique* `lex`*, c'est à dire une liste d'entiers de même taille que* `lis`*, contenant à la case d'indice* `i` *l'indice du mot* `lis[i]` *dans* `lex`*.*
3. *Écrire une fonction* `decode (code, lex)` *qui reconstruit un texte à partir de son code et du lexique.*

> ***Contrat:***
> *Encodez le résultat de la fonction* `loadText ()`*, puis décodez-le. Vérifiez que vous obtenez un texte identique à l'original en utilisant* `==` *pour tester si les deux listes sont les mêmes.*

□