Devoir sur table du 03/11/2020 - Algorithmique et Programmation Python
Durée approximative 2H

Aucun document autorisé

Instructions **à lire attentivement!**

1. Réponses en Python. Dans la suite j'appelle procédure une fonction Python dont le flot termine par return (sans valeur de retour).
2. Les codes écrits seront expliqués (au besoin avec des exemples), *commentés*, indentés correctement. Lorsque la signature d'un algorithme n'est pas fournie, vous veillerez à bien justifier vos choix.
3. Les réponses aux questions seront numérotées, et on évitera le grapillage.
   Dans toutes les questions, vous avez le droit de faire appel aux fonctions des questions précédentes, même si vous ne les avez pas écrites.

Partie Algo – Problème (Compression de fichiers BZIP)

d'après Concours d'entrée Polytechnique 2007 pour les non-optants info et TP de Caml de K. Huguenin et C. Avenel ; et un sujet posé en 2012 à Lille.

Nous allons implémenter en C l'algorithme `bzip` lui-même. Dans la deuxième partie nous implémenterons un autre algorithme utile pour la compression de données.

Les deux parties sont indépendantes

Partie I - BZip (compression par redondance)

Dans cette partie, on considère un texte (numérique) contenu dans un tableau d'entiers `t`, de taille N **constante globale** (pour les exemples, $N = 12$). Le tableau est considéré rempli (toutes les cases de 0 à $N-1$ contiennent donc un entier) par des entiers de 0 à 255. Dans cette partie nous allons compresser ce texte par la méthode Bzip.

i	0	1	2	3	4	5	6	7	8	9	10	11 = N-1
$t[i]$	2	3	3	200	10	50	7	12	50	50	2	2

FIGURE 1 – Un texte à compresser

Question 1

Écrire une **fonction Python** de signature

```
def nb_occ (t, el)
"""
t table, el entier
"""
```

qui calcule et retourne le nombre d'occurrences de l'entier *el* dans le tableau t. Sur l'exemple, l'appel `nb_occ(t,2)` retournera donc 3. Quelle est la complexité de votre fonction en nombre de tests ?

Question 2

Écrire une **procédure Python** de signature :

```
def occurences (t)
"""
t table de taille N
"""
r=256*[0] #init à 0
```

qui prend en argument un tableau d'entiers t de taille N, qui retourne un tableau r de façon à ce que $r[i]$ contienne le nombre d'occurences de i dans t. Sur l'exemple de la figure 1 , comme t comporte 3 occurrences du nombre 2, la case d'indice 2 du tableau r contiendra donc 3. *Il n'est pas obligatoire d'utiliser la fonction précédente.* Quelle est la complexité en nombre de tests de votre action ?

Question 3

Écrire une **fonction Python** `minichar` qui étant donné un tableau d'entiers t de taille N, calcule et retourne le plus petit nombre positif ou nul qui n'apparaît pas dans le tableau t. *On utilisera la fonction précédente.* Sur le tableau de l'exemple, l'appel `minichar(t)` retournera donc 0 puisque 0 n'apparaît pas dans le texte.

Question 4

Écrire une **fonction Python** qui prend en paramètre deux tableaux de taille N et qui retourne `true` si et seulement si les deux tableaux sont égaux case par case. On veillera à faire le moins de tests possibles.

Le principe de la compression par redondance est d'observer les répétitions de lettres consécutives et de les remplacer par le nombre de ces répétitions. L'algorithme codé par vos collègues de l'an dernier avait pour objectif de transformer le texte de façon à faire apparaître les redondances. Supposons donc que le texte à compresser comporte des redondances et qu'*il existe un nombre qui n'apparaît pas dans le texte.*

i	0	1	2	3	4	5	6	7	8	9	10	11 = N-1
$t[i]$	0	0	3	2	3	3	3	3	3	3	5	5

FIGURE 2 – Un texte à compresser avec redondances

Le texte compressé se construit de la façon suivante :
— L'entier `minichar(t)` sert de marqueur.
— On lit le texte de gauche à droite, et le texte compressé commencera par le marqueur.
— Si le nombre lu k n'apparaît qu'une seule fois à la suite, alors son codage est lui-même (k). Dans la figure 2, c'est le cas de 2 par exemple.
— Si le nombre lu k apparaît plusieurs fois de suite, mettons j fois, alors la suite des j occurrences de k est codée par 3 chiffres successifs : le marqueur, puis $j-1$, puis k.

Ainsi, sur l'exemple 2, le marqueur est 1. Le texte compressé sera donc :

$$\underbrace{1}_{marqueur}\ ,\underbrace{1,1,0}_{2fois0},\ \underbrace{3}_{1fois3}\ ,\ \underbrace{2}_{1fois2}\ ,\underbrace{1,5,3}_{6fois3},\underbrace{1,1,5}_{2fois5}$$

Dans ce cas, le texte compressé est de taille 12, tout comme le texte initial. Néanmoins, la taille du texte compressé peut-être plus petite (ou plus grande !) que le texte initial.

Question 5

Considérons le tableau suivant :

i	0	1	2	3	4	5	6	7	8	9	10	11 = N-1
$t2[i]$	1	1	1	2	1	5	3	3	3	2	2	2

Quel est le marqueur ? Quel texte obtient-on après compression de ce tableau ?

Les deux questions suivantes seront justifiées avec soin sur des exemples, elles sont plus complexes que les autres.

Question 6

Écrire une **fonction Python** `compresse` qui étant donné un texte de taille N (toutes les cases sont remplies), calcule le marqueur, puis la compression du texte et stocke le texte compressé dans le tableau $resu$ de taille M supposé suffisamment grand. La fonction retournera la taille du texte après compression : dans le cas du tableau de de la Figure 2, cette taille est 12.

Question 7

Écrire une **procédure Python** de signature

```
void decompresse (com, decomp, taille)
"""
comp table de taille M
decom table de taille N
taille un entier
"""
```

qui décompresse le tableau $comp$ de façon à retrouver le tableau initial, sachant que la taille après compression est l'entier `taille`. Quelle est la complexité ?

Question 8

Écrire un programme **Python complet** : main, fonctions (déclarations, et code avec des pointillés), etc :

— Déclaration des constantes symboliques : N vaut 12, M vaut 42.
— Déclaration et initialisation du tableau t de la Figure 2.
— Déclaration d'un tableau $comp$ de taille M et appel de la fonction `compresse` de façon à ce que $comp$ contienne le texte t compressé. La taille $\ell < m$ (nombre de caractères) du message compressé sera récupérée.
— Décompression du texte $comp$ dans un tableau dec de taille N préalablement déclaré.
— Vérification de l'égalité des tableaux t et dec et impression du résultat.

Partie II - Algorithme "Move To Front"

Dans cette partie, on considère un texte de caractères minuscules (de 'a' à 'z') sous forme de tableau de caractères de taille N. On va transformer ce texte en un texte de même taille, en utilisant un tableau d'association de taille 26.

DÉFINITION 1. Un tableau d'association est un tableau de caractères de taille 26 indexé de 0 à 25 contenant toutes les lettres de l'alphabet (une et une seule fois chacune).

Ce tableau associe donc à chaque lettre de l'alphabet (entre 'a' et 'z') un unique entier (son indice d'apparition dans le tableau) , ce qui nous servira pour encoder le texte. L'originalité de la méthode est que le tableau d'association va être modifié au cours du codage.

Question 9

Écrire en **Python** un algorithme qui remplit un tableau de taille 26 avec les caractères 'a' (case 0) à 'z' (case 25). Le tableau calculé sera donc :

i	0	1	2	...	25
$assoc[i]$	'a'	'b'	'c'	...	'z'

On pourra (mais ce n'est pas obligatoire) utiliser le fait que le caractère 'a' a le code ascii 97 et qu'un caractère et son code ascii sont le même "objet". Ce tableau sera le *tableau d'association initial*.

Question 10

Écrire en **Python** un algorithme qui étant donné un caractère ch (entre 'a' et 'z') et un tableau d'association $assoc$:

— Va chercher l'indice i de ce caractère dans le tableau ($assoc[i] = ch$).
— Décale tous les caractères avant lui (indices 0 à $i-1$) vers la droite.
— Place ce caractère à la case 0 du tableau.
— Retourne i.

Sur le tableau d'association initial, et le caractère 'c', le tableau sera donc *modifié* de la façon suivante :

i	0	1	2	3	...	25
$assoc[i]$	'c'	'a'	'b'	'd'	...	'z'

et le nombre retourné sera 3.

(fin question 10)

L'algorithme d'encodage est alors le suivant :
— On génère le tableau d'association initial.
— Le texte à encoder est lu de gauche à droite lettre après lettre : chaque lettre lue est encodée par un nombre (entre 0 et 25) qui correspond à son indice dans le tableau d'association courant.
— Après chaque lecture de lettre ch et écriture de l'encodage, le tableau d'association est décalé à l'aide de l'algorithme de la question 10 avec la lettre ch.

Pour illustrer l'encodage, prenons la chaîne ima (supposée stockée dans un tableau de taille supérieure à 4). Le tableau d'association initial est le tableau de la question 9. Voici les étapes :
— On lit $'i'$, qui est à la case numéro 8 du tableau, donc le premier nombre du codage est 8, et le tableau d'assoc devient :

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	...	25
$assoc[i]$	'i'	'a'	'b'	'c'	'd'	'e'	'f'	'g'	'h'	'j'	'k'	'l'	'm'	'n'	...	'z'

— On lit $'m'$, dont l'encodage est 12, et on décale

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	...	25
$assoc[i]$	'm'	'i'	'a'	'b'	'c'	'd'	'e'	'f'	'g'	'h'	'j'	'k'	'l'	'n'	...	'z'

— On lit $'a'$ dont l'encodage est 2.
Finalement l'encodage de la chaîne ima est $8, 12, 2$.

Question 11

Quel est l'encodage de la chaîne $cool$?

L'algorithme suivant sera justifié avec soin.

Question 12

Écrire l'algorithme d'encodage en **Python**, et analyser sa complexité.

Remarques :

— L'étudiant·e curieux pourra se reporter aux pages Wikipédia sur la compression de données. en particulier, l'algorithme de la partie II est utilisé dans certains algorithmes de compression pour faire apparaître des 0 en début de texte à compresser. Il peut être utilisé notamment après la transformation de Burrows-Wheeler, et avant l'algorithme de la partie I.