université de BORDEAUX

Année Universitaire 2014/2015
DS 1

Parcours : Licence LIMI201 & LIMI211
Code UE : J1MI2013
Épreuve : Algorithmes et Programmes
Date : Lundi 20 avril 2015, 8h30–10h
Durée : 1 heure 30
Documents : non autorisés

Collège Sciences et Technologies

SUJET + CORRIGE

Avertissement

— La plupart des questions sont indépendantes.
— À chaque question, vous pouvez répondre, au choix, par un algorithme ou par un programme Python.
— Les indentations des fonctions écrites en `Python` doivent être **respectées**.
— Répondez directement sur cette feuille.

Question	Points	Score
Compréhension de fonction	3	
Somme des inverses	2	
Multiples de 9	7	
Tableaux	8	
Total:	20	

Exercice 1 : Compréhension de fonction (3 points)

La fonction `mystere(t,s)` donnée ci-dessous prend en paramètre deux tableaux `t` et `s`.

```
def mystere(t,s):
    lt = len(t)
    ls = len(s)
    if lt != ls:
        return False
    i = 0
    while i < lt and  t[i] == s[ls-i-1]:
        i+=1
    return i==lt
```

1. (1 point) Que retournent les appels suivants ?
   a. `mystere([],[])`

   Solution: L'appel renvoie `True`.

   b. `mystere([1,2,3,4],[4,3,2,1])`

   Solution: L'appel renvoie `True`.

c. `mystere([1,2,5,4],[4,3,2,1])`

Solution: L'appel renvoie `False`.

d. En général, quel est le résultat d'un appel de la fonction `mystere` ? Justifier brièvement.

Solution: La fonction prend en paramètres deux tableaux, `t` et `s`, et vérifie si la suite des éléments dans `t`, lue du plus petit au plus grand indice, est la même que la suite dans `s`, lue en sens inverse.

2. (1 point) Évaluer la complexité en temps (meilleur des cas et pire des cas) de la fonction `mystere`. Justifier brièvement.

Solution: Considérons le nombre de comparaisons entre éléments des deux tableaux. Le meilleur des cas se produit lorsque les longueurs des deux tableaux diffèrent ou lorsque le premier élément du premier tableau et le dernier élément du deuxième tableau diffèrent : $\Omega(1)$. Le pire des cas se produit lorsque tous les éléments des deux tableaux sont comparés deux à deux : $\mathcal{O}(\texttt{len}(\texttt{t}))$.

3. (1 point) Lors de l'appel `mystere([1,2,3,4],[4,3,2,1])`

a. pour quelles valeurs de la variable `i` la comparaison `i < lt` est-elle effectuée ?

b. pour quelles valeurs de la variable `i` la comparaison `t[i] == s[ls-i-1]` est-elle effectuée ?

Justifier ces réponses.

Solution: La comparaison `i < lt` est effectuée pour $0 \leq i \leq 4$.
La comparaison `t[i] == s[ls-i-1]` est effectuée pour $0 \leq i \leq 3$.
Lorsque `i` vaut 4 l'expression `i < lt` a valeur `False`. En raison de l'évaluation paresseuse de l'opérateur `and` la deuxième comparaison n'est pas éffectuée.

Exercice 2 : Somme des inverses (2 points)

La suite u_n est définie pour $n \geqslant 1$ par la formule suivante :

$$u_n = \sum_{k=1}^{n} \frac{1}{k} = 1 + \frac{1}{2} + \cdots + \frac{1}{n}.$$

1. (½ point) Écrire une définition par récurrence de cette suite.

Solution:

$$\begin{cases} u_1 = 1 \\ \forall n > 1, u_n = u_{n-1} + \frac{1}{n} \end{cases}$$

2. (1½ points) Écrire une fonction **récursive**

`sommeInverses(n)`

qui calcule u_n lorsque $n \geqslant 1$. Pour $n \leqslant 0$, la fonction retournera 0.

Solution:

```
def sommeInverses(n):
    if n <= 0:
        return 0
    if n == 1:
        return 1
    return 1/n + sommeInverses(n-1)
```

Exercice 3 : Multiples de 9 (7 points)

1. (½ point) Écrire une fonction `sommeChiffres(n)` qui calcule la somme des chiffres du nombre entier `n`.

Solution:

```
def sommeChiffres (n):
    s = 0
    while n > 0:
        s += n % 10
        n //= 10
    return s
```

On veut écrire une fonction qui teste si un nombre est multiple de 9 en utilisant la propriété suivante :

Un nombre est multiple de 9 si et seulement si la somme de ses chiffres est multiple de 9.

Le principe est de répéter le calcul de la somme des chiffres jusqu'à obtenir un nombre d'un seul chiffre (rappel : 0 aussi est multiple de 9).

Exemple : on teste si 9565938 est multiple de 9 en calculant la somme de ses chiffres, qui est 45. On recommence : la somme des chiffres de 45 est 9. Le nombre 9565938 est donc multiple de 9.

2. (1 point) Écrire une fonction **non récursive** `estMultipleDe9(n)` qui renvoie `True` si l'entier `n` est multiple de 9 et `False` sinon, en utilisant la propriété indiquée. En particulier, elle devra utiliser la fonction `sommeChiffres` de la question 1, et ne pas utiliser les opérateurs (`/`, `//`, `%`, `*`).

Solution:

```
def estMultipleDe9(n):
    while n > 9:
        n = sommeChiffres(n)
    return n == 9 or n == 0
```

3. (1 point) Écrire une version **récursive** `estMultipleDe9Rec(n)` de la fonction écrite en question 2.

Solution:

```
def estMultipleDe9Rec(n):
    if n <= 9:
        return n==9 or n == 0
    return estMultipleDe9Rec(sommeChiffres(n))
```

4. (1½ points) Écrire une fonction `indiceDernierMultipleDe9(t)` qui, étant donné un tableau `t` d'entiers, retourne le dernier (le plus grand) indice de `t` où se trouve un multiple de 9, et retourne `None` si aucun multiple de 9 n'est présent dans `t`.

   On demande un algorithme qui minimise le temps d'exécution dans le cas le plus favorable (c'est-à-dire quand le dernier élément du tableau est un multiple de 9).

Solution:

```
def indiceDernierMultipleDe9(t):
    for i in range(len(t)-1,-1,-1):
        if estMultipleDe9(t[i]):
            return i
    return None
```

5. (3 points) Écrire une fonction `supprimerPremierMultipleDe9(t)` qui supprime du tableau `t` d'entiers le premier élément qui est un multiple de 9. Vous pouvez utiliser les primitives de la bibliothèque `bibTableau.py` du cours.

Solution:

```
def supprimerPremierMultipleDe9(t):
    n = len(t)
    i = 0
    while i < n and not estMultipleDe9(t[i]):
        i += 1
    if i < n:
        for i in range(i, n-1):
            t[i] = t[i+1]
        supprimerNcases(t,1)
```

Exercice 4 : Tableaux (8 points)

On veut représenter un tableau `t` de nombres par un autre tableau `c` appelé *codage* de `t`. Les suites consécutives de valeurs identiques de `t` sont représentées dans `c` par deux nombres `r,v` où `r` est le nombre de répétitions de la valeur `v` dans une telle suite. Une valeur `v` de `t` qui ne se répète pas est donc représentée par `1,v`. Une valeur `v` qui se répète deux fois consécutivement est représentée par `2,v`, et ainsi de suite.

Par exemple, le codage de `t = [0,0,0,0,5,-2,-2,-2,0,0,0,0,0]` est `c = [4,0,1,5,3,-2,5,0]`. Un entier à un indice pair dans `c` (dans l'exemple `4`, `1`, `3` et `5`) représente donc un nombre de répétitions consécutives d'une valeur de `t`.

1. (3 points) Écrire une fonction `decoder(c)` qui « décode » un tableau `c` de longueur paire, c'est-à-dire qui renvoie le tableau `t` dont `c` est le codage.

```
>>> decoder([4, 0, 1, 5, 3, -2, 5, 0])
[0, 0, 0, 0, 5, -2, -2, -2, 0, 0, 0, 0, 0]
```

Solution:

```
def decoder(c):
    n = 0
    for i in range(len(c)//2):
        n += c[i+i]
    t = creerTableau(n)
    j = 0 # indice d'ecriture dans t
    for i in range(0,len(c),2):
        for r in range(c[i]):
            t[j] = c[i+1]
            j += 1
    return t
```

ou bien :

Solution:

```
def decoder(c):
    t = creerTableau(0)
    j = 0 # indice d'ecriture dans t
    for i in range(0,len(c),2):
        ajouterNcases(t, c[i])
        for r in range(c[i]):
            t[j] = c[i+1]
            j += 1
    return t
```

2. (1 point) Quelle est la complexité de votre fonction `decoder` ? Justifiez la réponse.

Solution: La complexité est $\mathcal{O}(k)$ où k est la somme des valeurs aux positions paires de c.

On veut écrire la fonction de codage qui prend en entrée un tableau `t` et renvoie son codage `c`.

3. (2 points) Écrire une fonction `longueurBloc(t, pos)` qui renvoie la longueur du plus long bloc de positions consécutives du tableau `t` commençant en position `pos`, et composé de cases consécutives qui contiennent toutes la valeur `t[pos]`. Par exemple

```
>>> t = [0,0,0,0,5,-2,-2,-2,0,0,0,0,0]
>>> longueurBloc(t, 0)
4
>>> longueurBloc(t, 1)
3
>>> longueurBloc(t, 4)
1
>>> longueurBloc(t, 8)
5
```

Solution:

```
def longueurBloc(t, pos):
    longueur = len(t)
    depart = pos
    if pos >= longueur or pos < 0:
        return None
    while pos < longueur-1 and t[pos] == t[pos+1]:
        pos += 1
    return pos-depart+1
```

4. (2 points) En utilisant la fonction `longueurBloc`, écrire une fonction `codage(t)` qui renvoie le codage du tableau `t`. Vous pouvez utiliser les primitives de la bibliothèque `bibTableau.py` du cours.

Solution:

```
def codage(t):
  i = 0
  longueur = len(t)
  c = creerTableau(0)
  while i < longueur:
      ajouterNcases(c, 2)
      lb = longueurBloc(t, i)
      c[-2] = lb
      c[-1] = t[i]
      i += lb
  return c
```