université de BORDEAUX

Année Universitaire 2013/2014
DS 2

Parcours : Licence LIMI201 & LIMI211
Code UE : J1MI2013
Épreuve : Algorithmes et Programmes
Date : Vendredi 18 avril 2014 **Heure :** 8 heures 30
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 au choix répondre par un algorithme ou bien par un programme python.**
- Les indentations des fonctions écrites en `Python` doivent être respectées.
- L'espace laissé pour les réponses est suffisant (sauf si vous utilisez ces feuilles comme brouillon, ce qui est fortement déconseillé).

Question	Points	Score
Évaluation d'expressions	3	
Homogénéité	3	
Calculs d'une suite	6	
Tableaux	8	
Total:	20	

Exercice 1 : Évaluation d'expressions (3 points)

On définit les variables suivantes (en Python) :

```
t = [4, 8, 15, 16, 23, 42]
i = 3
x = t[1]
```

Donner le résultat de l'évaluation des expressions suivantes, en précisant le résultat de l'évaluation des sous-expressions qui sont utilisées.

(a) (1 point) `t[i + 1] - t[i - 1] == t[i - 2]`

Solution: '`True`' : '`t[4] - t[2] == t[1]`' donne '`23 - 15 == 8`' donne '`8 == 8`'

(b) (1 point) `x < len(t) and t[x] != 0`

Solution: '`False`' : '`8 < 6 and ...` ' donne '`False and ...` ', évalué paresseusement

(c) (1 point) `not t[i] < x or t[x] > 20`

Solution: 'True' : 'not t[3] < 8 or ... ' donne 'not False or ... ' donne 'True or ... ', évalué idem

Exercice 2 : Homogénéité (3 points)

(a) (2 points) Écrire une fonction `homogene (t)` prenant en paramètre un tableau `t` et qui retourne `True` si le tableau est homogène, c'est-à-dire si tous ses éléments sont identiques, et `False` sinon. Exemples :

```
>>> homogene([])
True
>>> homogene([2, 2, 2])
True
>>> homogene([2, 2, 1])
False
```

Solution:

```
def homogene (t):
    for i in range(1, len(t)):
        if t[i] != t[0]:
            return False
    return True
```

(b) (½ point) Donner la complexité dans le meilleur des cas ?

Solution: $\Omega(1)$. Le meilleur des cas se produit lorsque `t[0]` diffère de `t[1]`.

(c) (½ point) Donner la complexité dans le pire des cas ?

Solution: $O(len(t))$. Le pire de cas se produit lorsque le tableau est homogène.

Exercice 3 : Calculs d'une suite (6 points)

On considère la fonction :

```
def suiteIter (n):
    u = 1
    while n > 0:
        u = 2 * u + 1
        n -= 1
    return u
```

(a) (1 point) Calculer `suiteIter(3)`.

Solution: `suiteIter(3)` retourne 15.

(b) (2 points) Écrire une version récursive `suiteRec (n)` pour la fonction précédente.

Solution:

```
def suiteRec (n):
    if n == 0:
        return 1
    return 2 * suiteRec(n - 1) + 1
```

(c) (2 points) Pour tout entier naturel k, soit S_k la valeur retournée par `suiteIter(k)`. Sans utiliser les fonctions `suiteIter (n)` et `suiteRec (n)`, écrire une fonction `tableauSuite (n)` qui crée et retourne un tableau contenant les `n` premiers termes de la suite S.

Solution:

```
def tableauSuite (n):
    t = creerTableau(n,1)
    for i in range(1, n):
        t[i] = 2 * t[i - 1] + 1
    return t
```

(d) (1 point) Donner une version récursive terminale de la fonction `suiteRec (n)`.

Solution:

```
def suiteRecTerm (n, acc):
    if n == 0:
        return acc
    return suiteRecTerm(n - 1, 2 * acc + 1)
```

Exercice 4 : Tableaux (8 points)

Pour cet exercice, vous pouvez faire appel aux primitives de la bibliothèque `bibTableau.py` utilisée en cours.

(a) (2 points) Écrire une fonction `compterChangements (t)` qui prend en paramètre un tableau `t` et qui retourne le nombre d'apparitions d'une valeur différente de la précédente (en comptant la première valeur). Exemples :

```
>>> compterChangements([])
0
>>> compterChangements([2])
1
>>> compterChangements([2, 1, 1, 1, 2, 3, 2])
5
```

Solution:

```
def compterChangements (t): # Theta(len(t))
    l = len(t)
    if l == 0:
        return 0
    c = 1
    for i in range(1, l):
        if t[i] != t[i-1]:
            c = c + 1
    return c
```

(b) (1 point) Quelle est la complexité de cette fonction ?

Solution: $\Theta(len(t))$

Pour représenter une suite de n valeurs v identiques de manière compressée, on utilise une paire (k, v). En `Python`, on représentera une paire (k, v) par un tableau à deux cases `[k, v]`. Par exemple, la suite $2, 2, 2, 2$ est représentée par la paire $(4, 2)$ et en `Python` par le tableau `[4, 2]`.

(c) (1 point) Écrire une fonction `Python creerPaire (k, v)` qui retourne le tableau représentant la paire (k, v).

Solution:

```
def creerPaire (k, v):
```

```
    paire = creerTableau(2)
    paire[0] = k
    paire[1] = v
    return paire
```

(d) (½ point) Quelle est la complexité de la fonction `creerPaire` ?

Solution: $\Theta(1)$

Soit la fonction `mystere (t)` donnée Figure 1 qui prend en paramètre un tableau `t` **non vide**.

```
def mystere (t):
    s = creerTableau(compterChangements(t))
    v = t[0]
    k = 1
    j = 0
    for i in range(1, len(t)):
        if t[i] != v:
            s[j] = creerPaire(c, v)
            k = 1
            j = j + 1
            v = t[i]
        else:
            k = k + 1
    s[j] = creerPaire(k, v)
    return s
```

FIGURE 1 – Fonction `mystere`

(e) Que retournent les appels suivants ?

i. (½ point) `mystere([1])`

Solution: `[[1, 1]]`

ii. (½ point) `mystere([1, 1, 1])`

Solution: `[[3, 1]]`

iii. (½ point) `mystere([1, 1, 1, 4, 4, 1])`

Solution: `[[3, 1], [2, 4], [1, 1]]`

(f) (1 point) Quelle est la complexité de la fonction `mystere` ?

Solution: $\Theta(\mathit{len}(t))$

(g) (1 point) Expliquer en une ou deux phrases ce que retourne la fonction `mystere` en fonction de `t`.

Solution: La fonction retourne un tableau `s` de paires qui représente une forme compressée du tableau `t`. Si on concatène les séquences obtenues par décompression de chaque paire, on retrouve la suite des éléments de `t`.