Université Bordeaux 1 LST Mention MI Semestre 2 2012-2013

J1MI2013: Algorithmes et Programmes

Devoir surveillé

Aucun document autorisé. Mercredi 24 avril 2013

Durée : 1h30

Le barème est donné à titre indicatif.

Exercice 1. (2.5 pts)

1. Écrire une fonction `procedureScalaireTableau(t, k)` où `t` est un tableau de nombres, `k` un nombre, et qui remplace dans chaque case `i` du tableau `t`, `t[i]` par `k` $\times$ `t[i]`.
2. Quelle est la complexité de votre fonction ?
3. Écrire une fonction `scalaireTableau(t, k)` où `t` est un tableau de nombres, `k` un nombre et qui retourne un nouveau tableau `s` de même taille que le tableau `t` et tel que pour chaque case `i` du tableau on a `s[i]` = `k` $\times$ `t[i]`.

Exemples :

```
>>> t = [1, 2, 3, 4]
>>> procedureScalaireTableau(t, 10)
>>> print(t)
[10, 20, 30, 40]
>>> scalaireTableau(t, 2)
[20, 40, 60, 80]
>>> print(t)
[10, 20, 30, 40]
```

Exercice 2. (9 pts)

On appelle *suite de Thue-Morse* la suite définie récursivement par :

$$\begin{cases} t_0 = 0 & \\ t_n = t_{n/2} & \text{si n est pair et strictement positif} \\ t_n = 1 - t_{(n-1)/2} & \text{si n est impair} \end{cases}$$

On peut montrer que cette suite ne contient que les valeurs 0 et 1. Les premières valeurs de la suite sont : 0,1,1,0,1,0,0,1,1...

1. Écrire en Python une fonction `estPair(n)` qui retourne `True` ou `False` suivant que l'entier `n` est pair ou non.
2. Donner la complexité en temps de votre fonction `estPair(n)`.
3. Écrire en Python une fonction itérative `ThueMorseTableau(n)` qui retourne un tableau contenant les chiffres d'indice inférieur ou égal à `n` de la suite de Thue-Morse. Exemple : l'appel `ThueMorseTableau(6)` retournera le tableau [0, 1, 1, 0, 1, 0, 0].
4. Donner la complexité en temps de votre fonction `ThueMorseTableau(n)`.
5. Écrire en Python une fonction récursive `chiffreThueMorseRecursive(n)` qui retourne $t_n (n \geq 0)$. Exemple : `chiffreThueMorseRecursive(9)` retourne 0.
6. Donner la liste des appels récursifs lors de l'exécution de `chiffreThueMorseRecursive(9)`.
7. Quelle est la complexité en temps de votre fonction `chiffreThueMorseRecursive(n)` ? Justifier votre réponse.
8. Étant donné un tableau `s` contenant une suite binaire, on veut savoir si cette suite est une sous-suite des `n` premiers chiffres de la suite de Thue-Morse. Pour cela, écrire en Python une fonction `estSousSuiteThueMorse(n, s)` qui d'abord construit un tableau représentant les `n` premiers chiffres de la suite de Thue-Morse, ensuite compare le contenu de ce tableau avec le contenu du tableau `s`. La fonction renvoie `True` si `s` correspond à une sous-suite des `n` premiers chiffres de la suite de Thue-Morse et renvoie `False` sinon. Par exemple, cette fonction renvoie `True` pour `s` = [0, 1, 0, 0, 1, 1] et `n` = 10 car les `n` premiers chiffres sont 011<u>0100110</u>0 ; elle renvoie `False` si `s` = [1,0,1] et `n` = 4.
9. Donner et justifier la complexité de votre fonction `estSousSuiteThueMorse(n, s)`.

Exercice 3. (4 pts)

Etant donnée la fonction `mystere` définie comme :

```
def mystere(t, i):
  if i >= len(t) - 1:
      return True
  t[i] = ( t[i + 1] and mystere(t, i + 1))
  return t[i]
```

Qu'affichent les codes suivants à l'écran ?

1.
```
>>> t = [True, True]
>>> mystere(t, 0)
>>> print(t)
```

2.
```
>>> t = [True, False]
>>> mystere(t, 0)
>>> print(t)
```

3.
```
>>> t = [True, True, False, True, False, True]
>>> mystere(t, 0)
>>> print(t)
```

4.
```
>>> t = [True, True, False, True, False, True]
>>> mystere(t, 1)
>>> print(t)
```

5. Que fait la fonction `mystere` dans le cas général ?

Exercice 4. (5.5 pts)

La fonction `supprimerOccurrences(t, x)` supprime de la suite de nombres entiers rangés dans le tableau `t` toutes les occurrences de l'élément `x`. Voir un exemple ci-dessous.

```
def supprimerOccurrences(t, x):
    sup = 0
    for i in range(len(t)):
        if t[i] == x:
            sup += 1
        else:
            t[i-sup] = t[i]
    supprimerNcases(t, sup)
```

1. Donner et justifier la complexité de la fonction `supprimerOccurrences(t, x)`.
2. En suivant le modèle de la fonction `supprimerOccurrences`, écrire une fonction `supprimerKOccurrences(t, x, k)` qui supprime de la suite de nombres entiers rangés dans le tableau `t` les `k` premières occurrences de l'élément `x` ($k > 0$).
   S'il y a moins de `k` occurrences de l'élément `x` dans `t`, toutes les occurrences sont supprimées.
3. Écrire une fonction `compacterOccurrencesElement(t, x)` qui remplace chaque suite d'occurrences de `x` dans `t` par une seule occurrence de `x`.
4. Écrire une fonction `compacter(t)` qui remplace dans `t` toute suite d'occurrences d'une valeur par une seule occurrence de cette valeur.
5. Donner et justifier la complexité de votre fonction `compacter(t)`.

Exemples :

```
>>> t = [1, 9, 3, 6, 9, 9, 9, 9, 6, 9, 3, 3, 9, 9]
>>> supprimerOccurrences(t, 9)
>>> print(t)
[1, 3, 6, 6, 3, 3]

>>> t = [1, 9, 3, 6, 9, 9, 9, 9, 6, 9, 3, 3, 9, 9]
>>> supprimerKOccurrences(t, 9, 5)
>>> print(t)
[1, 3, 6, 6, 9, 3, 3, 9, 9]

>>> t = [1, 9, 3, 6, 9, 9, 9, 9, 6, 9, 3, 3, 9, 9]
>>> compacterOccurrencesElement(t, 9)
>>> print(t)
[1, 9, 3, 6, 9, 6, 9, 3, 3, 9]

>>> t = [1, 9, 3, 6, 9, 9, 9, 9, 6, 9, 3, 3, 9, 9]
>>> compacter(t)
>>> print(t)
[1, 9, 3, 6, 9, 6, 9, 3, 9]
```

Annexe : rappel des principales fonctions du module bibTableau

1. Création et initialisation de tableaux d'entiers
   (a) `creerTableau(taille)`
   La fonction retourne un tableau de taille éléments initialisés à 0.
   Exemple : creerTableau(4) retourne le tableau `[0, 0, 0, 0]`
   (b) `creerTableau(taille,valeurInitiale)`
   La fonction retourne un tableau de `taille` éléments initialisés à valeurInitiale.
   Exemple : creerTableau(5,-3) retourne le tableau `[-3, -3, -3, -3, -3]`
2. Modifications de tableaux
   (a) `ajouterNcases(t,n)`
   La fonction ajouterNcases ajoute n cases au tableau fourni en paramètre. Ces cases sont initialisées a None.
   (b) `supprimerNcases(t,n)`
   La fonction supprimerNcases supprime les n dernières cases du tableau fourni en paramètre.