J1MI2013: Algorithmes et Programmes

Corrigé du devoir surveillé du Mercredi 24/04/2013

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 $k \times t[i]$.

```
def procedureScalaireTableau(t, k):
    for i in range(len(t)):
        t[i] *=k
```

2. Quelle est la complexité de votre fonction ?

   Si `t` est de taille n, la compléxité de la fonction est $\Theta(n)$.

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 $s[i] = k \times t[i]$.

```
def scalaireTableau(t, k):
    n = len(t)
    s = creerTableau(n)
    for i in range(n):
        s[i] = t[i] * k
    return s
```

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.

```
def estPair(n):
    return n%2==0
```

2. Donner la complexité en temps de votre fonction `estPair(n)`.

   La fonction est de complexité constante, elle est $\Theta(1)$.

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].

```
def ThueMorseTableau(n):
    t = creerTableau(n + 1)
    for i in range(1, n + 1):
        if estPair(i):
            t[i] = t[i // 2]
        else:
            t[i] = 1 - t[(i - 1) // 2]
    return t
```

4. Donner la complexité en temps de votre fonction `ThueMorseTableau(n)`.
   La complexité de la fonction ci-dessus est $\Theta(n)$.
5. Écrire en Python une fonction récursive `chiffreThueMorseRecursive(n)` qui retourne $t_n(n \geq 0)$. Exemple : `chiffreThueMorseRecursive(9)` retourne 0.

```
def chiffreThueMorseRecursive(n):
    if n == 0:
        return 0
    if estPair(n):
        return chiffreThueMorseRecursive(n // 2)
    return 1 - chiffreThueMorseRecursive((n - 1) // 2)
```

6. Donner la liste des appels récursifs lors de l'exécution de `chiffreThueMorseRecursive(9)`.

   `chiffreThueMorseRecursive(9)`, `chiffreThueMorseRecursive(4)`,
   `chiffreThueMorseRecursive(2)`, `chiffreThueMorseRecursive(1)`,
   `chiffreThueMorseRecursive(0)` $\rightarrow 0$

7. Quelle est la complexité en temps de votre fonction `chiffreThueMorseRecursive(n)` ? Justifier votre réponse.

   La complexité est proportionnelle au nombre d'appels récursifs. À chaque nouvel appel récursif le paramètre est calculé en divisant par deux (division entière) le paramètre de l'appel précédent, la récursivité s'arrête lorsque le paramètre vaut 0. Si n est la valeur initiale du paramètre, le nombre d'appels récursifs dépend donc du nombre de divisions entières par deux nécessaires pour atteindre 0 à partir de n. La complexité est $\Theta(\log_2(n))$.

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>1 ; elle renvoie `False` si `s` = [1, 0, 1] et `n` = 4.

```
def estSousSuiteThueMorse(s, n):
    ls = len(s)
    t = ThueMorseTableau(n)
    for i in range(n - ls + 1):
        j = 0
        while j < ls and t[i + j] == s[j]:
            j +=1
        if j == ls:
            return True
    return False
```

9. Donner et justifier la complexité de votre fonction `estSousSuiteThueMorse(n, s)`.

   Dans tous les cas il faut construire le tableau `t` (complexité : $\Theta(n)$).
   Meilleur des cas : $\Omega(n + len(s))$ (la suite dans `s` est sous-suite de la suite de Thue-Morse à partir du terme d'indice 0).
   Pire des cas : $\mathcal{O}(n * len(s))$ (la suite dans `s` n'est pas sous-suite de la suite de Thue-Morse).

Exercice 3. (4 pts)

Étant 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)
   ```

   Affiche :
   ```
   True
   [True, True]
   ```
2. ```
   >>> t = [True, False]
   >>> mystere(t, 0)
   >>> print(t)
   ```
   Affiche :
   ```
   False
   [False, False]
   ```
3. ```
   >>> t = [True, True, False, True, False, True]
   >>> mystere(t, 0)
   >>> print(t)
   ```
   Affiche :
   ```
   False
   [False, False, False, True, False, True]
   ```
4. ```
   >>> t = [True, True, False, True, False, True]
   >>> mystere(t, 1)
   >>> print(t)
   ```
   Affiche :
   ```
   False.
   [True, False, False, True, False, True]
   ```
5. Que fait la fonction `mystere` dans le cas général ?

   La fonction `mystere` est une fonction récursive à deux paramètres, `t` un tableau de booléens et `i` un entier qui désigne un indice de `t`. La fonction retourne un booléen et peut modifier le contenu du tableau. La valeur `True` est retournée si tous les éléments d'indice strictement supérieur à `i` valent `True` (sinon la valeur `False` est retournée). Quand la fonction retourne `True`, `t[i]` prend la valeur `True`. Quand la fonction retourne `False`, si `j` est le plus petit indice $>$ `i` tel que `t[j]` $==$ `False` alors tous les éléments d'indice compris entre `i` et `j` $-$ 1 prennent la valeur `False`.

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)`.
   La complexité de la fonction ci-dessus est $\Theta(\texttt{len}(\texttt{t}))$.
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` ($\texttt{k} > 0$).
   S'il y a moins de `k` occurrences de l'élément `x` dans `t`, toutes les occurrences sont supprimées.

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

3. Écrire une fonction `compacterOccurrencesElement(t, x)` qui remplace chaque suite d'occurrences de `x` dans `t` par une seule occurrence de `x`.

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

4. Écrire une fonction `compacter(t)` qui remplace dans `t` toute suite d'occurrences d'une valeur par une seule occurrence de cette valeur.

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

5. Donner et justifier la complexité de votre fonction `compacter(t)`.
   La complexité de la fonction ci-dessus est $\Theta(\texttt{len}(\texttt{t}))$.