Numéro d'anonymat :

Partiel LU2IN003

Mercredi 17 Mars 2021, 1.5 heures
Aucun document autorisé

Exercice 1 – Algorithme itératif (8.5 points)

On considère la fonction suivante, où a est un réel non nul et n un entier naturel :

```
def f(a, n):
    x = a ; k = n; r = 1
    print("x␣=␣",x, "␣k␣=␣", k, "et␣r␣=␣",r)
    while k > 0:
        if (k % 2) == 1:
            r = r * x
        k = k // 2
        x = x * x
        print("x␣=␣",x, "␣k␣=␣", k, "et␣r␣=␣",r)
    return r
```

Les opérations $k\%2$ et $k//2$ renvoient respectivement le reste et le quotient de la division de k par 2.

Question 1
Exécuter l'appel de `f(2,10)`, en ne donnant que les affichages.

Solution:

```
x =  2  k =  10 et r =  1
x =  4  k =  5 et r =  1
x =  16  k =  2 et r =  4
x =  256  k =  1 et r =  4
x =  65536  k =  0 et r =  1024
1024
```

Question 2
Montrer que l'algorithme se termine.

Solution:
La valeur de k décroît strictement à chaque passage dans la boucle, elle atteindra donc la valeur 0 au bout d'un nombre fini d'itérations et chaque itération est constituée d'un nombre fini d'instructions donc l'algorithme se termine.

Question 3
On note x_i, resp. k_i, resp. r_i, la valeur de x, resp. k, resp. r, à la fin de la i-ème itération. Initialement, $x_0 = a$, $k_0 = n$, et $r_0 = 1$.

1. Montrer, par récurrence sur i, que l'égalité $x_i^{k_i} * r_i = a^n$ est vérifiée pour tout i tel qu'il existe une i-ème itération.
2. En déduire que `f(a,n)` calcule a^n.

Solution:

1. **Base** Pour $i = 0$, on a $x_0 = a$, $k_0 = n$, et $r_0 = 1$ donc $x_0^{k_0} * r_0 = a^n * 1 = a^n$.
   Induction Soit $i \geq 0$ Supposons que l'égalité $x_i^{k_i} * r_i = a^n$ soit vérifiée et qu'il y ait une $(i+1)$-ème itération.
   Il y a deux cas à étudier : le cas où k_i est pair et le cas où k_i est impair.
   Dans le cas où k_i est pair, $x_{i+1} = x_i^2$, $k_i = 2k_{i+1}$ et $r_{i+1} = r_i$ donc $x_{i+1}^{k_{i+1}} * r_{i+1} = (x_i^2)^{k_{i+1}} * r_i = x_i^{2k_{i+1}} * r_i = x_i^{k_i} * r_i = a^n$.
   Dans le cas où k_i est pair, $x_{i+1} = x_i^2$, $k_i = 2k_{i+1} + 1$ et $r_{i+1} = r_i * x_i$ donc $x_{i+1}^{k_{i+1}} * r_{i+1} = (x_i^2)^{k_{i+1}} * r_i * x_i = x_i^{2k_{i+1}+1} * r_i = x_i^{k_i} * r_i = a^n$.
   Conclusion L'égalité $x_i^{k_i} * r_i = a^n$ est vérifiée pour tout i tel qu'il existe une i-ème itération.
2. À la fin du dernier tour de boucle, l'égalité $x^k * r = a^n$ est vérifiée et $k = 0$ donc $r = a^n$.
   La valeur retournée est r, donc a^n.

Question 4
Dans cette question, on calcule la complexité de `f(a,n)`, comptée en nombre de multiplications, pour $n \geq 1$.
Dans toute cette question, on suppose $n \geq 1$.
On pose $p = \lfloor \log_2(n) \rfloor$, on a donc $2^p \leq n < 2^{p+1}$.

1. Exprimer le nombre de tours de boucle effectués par `f(a,n)` en fonction de p. Justifier la réponse.
2. En déduire le nombre de multiplications effectuées par `f(a,n)` dans le pire cas et dans le meilleur cas.
3. En déduire que la complexité de `f(a,n)` est en $\Theta(log(n))$.

<u>**Solution**</u>:

1. Initialement, la valeur de k est n, avec $2^p \leq n < 2^{p+1}$. La valeur de k est divisée par 2 à chaque tour de boucle, elle vaut donc 0 au bout de $p + 1$ tours de boucle.
   Le nombre de tours de boucle effectués par `f(a,n)` est égal à $p + 1$, pour $2^p \leq n < 2^{p+1}$.
2. Dans chaque tour de boucle, il y a 1 multiplication (quand k est pair) ou 2 multiplications (quand k est impair). Il y a donc $p + 1$ multiplications dans le meilleur cas et $2(p + 1)$ multiplications dans le pire cas.
3. Soit c la complexité de `f(a,n)` alors $p + 1 \leq c \leq 2(p + 1)$. Donc c est en $\Theta(p)$. Puisque $p = \lfloor \log_2(n) \rfloor$, on a $\Theta(p) = \Theta(log_2(n)) = \Theta(log(n))$. Par conséquent, la complexité de `f(a,n)` est en $\Theta(log(n))$.

Exercice 2 – Etude d'une fonction récursive (6.5 points)

On considère la fonction suivante. `L` est une liste d'entiers et `x` un entier.

```
def H(L,x):
    print("Appel de H avec L=", L, "et x=",x)
    if (len(L)>0):
        res=L[0]+x*H(L[1:],x)
    else:
        res=0
    print("Retourne", res,"pour L=", L, "et x=",x)
    return res
```

L'appel `len(L)` retourne le nombre d'éléments de `L`. La notation `L[1:]` retourne la liste `L` sans son premier élément `L[0]`.

Question 1
Exécuter l'appel de `H(L,2)` pour `L=[3,2,0,1]` en ne donnant que les affichages et la valeur finale.
Solution:

```
Appel de H avec L= [3, 2, 0, 1] et x= 2
Appel de H avec L= [2, 0, 1] et x= 2
Appel de H avec L= [0, 1] et x= 2
Appel de H avec L= [1] et x= 2
Appel de H avec L= [] et x= 2
Retourne 0 pour L= [] et x= 2
Retourne 1 pour L= [1] et x= 2
Retourne 2 pour L= [0, 1] et x= 2
Retourne 6 pour L= [2, 0, 1] et x= 2
Retourne 15 pour L= [3, 2, 0, 1] et x= 2
15
```

Question 2
Montrez par récurrence sur la taille n de la liste `L` que l'appel `H(L,x)` se termine pour tout entier `x` et retourne $\sum_{i=0}^{n-1} L[i] * x^i$.
Solution:
Soit la propriété $\mathcal{P}$ définie par :
$\mathcal{P}(n)$: pour toute liste `L` de n éléments et tout entier `x` , l'appel `H(L,x)` se termine et retourne $\sum_{i=0}^{n-1} L[i] * x^i$. On démontre que pour tout $n \geq 0$, la propriété est vérifiée par récurrence faible.

Base : Pour $n = 0$, la liste est vide et la fonction retourne 0. $\mathcal{P}(0)$ est donc vérifiée.

Induction : Supposons que la propriété est vérifiée pour une valeur $n - 1 \geq 0$. On démontre $\mathcal{P}(n)$.
Soit L une liste de n éléments et un entier x. L est non vide, donc la sous-liste $L^\star = L[1:]$ existe, et par hypothèse de récurrence, l'appel `H(L[1:],x)` se termine et retourne $\sum_{i=0}^{n-2} L^\star[i] * x^i$.
Ainsi, l'appel `H(L,x)` se termine. De plus, $S = \sum_{i=0}^{n-1} L[i] * x^i = L[0] + \sum_{i=1}^{n-1} L[i] * x^i$. Pour tout $i \in \{0, \ldots, n-2\}$, $L^\star[i] = L[i+1]$, et donc $S = L[0]+x. \sum_{i=0}^{n-2} L[i+1]*x^i = L[0]+x. \sum_{i=0}^{n-2} L^\star[i]*x^i$.
On en déduit que $\mathcal{P}(n)$ est vérifiée par r.

Conclusion : Pour tout $n \geq 0$, $\mathcal{P}(n)$ est vérifiée par récurrence faible.

Question 3
Soit u_n, $n \geq 0$, le nombre de multiplications effectuées pour une liste de n éléments.

1. Que vaut u_0 ? Quelle est la relation de récurrence vérifiée par u_n ?
2. En déduire l'ordre de grandeur de la complexité de `H` .

Solution:

1. $u_0 = 0$ et $u_n = u_{n-1} + 1$.
2. $u_n = n$ et donc la complexité de la fonction est $\Theta(n)$.

Partiel du 17 mars 2021 - QCM

Un seul choix est possible. Une bonne réponse = 0.5 point. Une mauvaise réponse = -0.25 points

1. (0.5 points) On considère la fonction suivante, où n est un entier naturel.

```
def f(n):
    a = 1 ; k = 0
    while a <= n:
        a = a * 2 ; k = k + 1
    return k - 1
```

Dire laquelle des égalités suivantes est un invariant de boucle (vérifié à la fin du tour de boucle) :

- ○ $a = 2^{k-1}$
- √ $a = 2^k$
- ○ $a = 2^{k+1}$
- ○ $a = 2 * k$

2. (0.5 points) On considère la fonction suivante, où s est une chaîne de caractères. On rappelle que `len(s)` est la longueur de la chaîne s et que les positions des caractères dans s sont numérotées de 0 à `len(s)`.

```
def g(s):
    n = len(s) ; i = 0 ; res = ''
    while i < n:
        res = s[n-1-i] + res + s[i]
        i = i + 1
    return res
```

Pour `s = 'algorithmique'`, donner les valeurs de res et i après 3 tours de boucle :

- ○ `res = 'euqalg'` et $i = 3$
- ○ `res = 'quealg'` et $i = 4$
- √ **`res = 'quealg'` et** $i = 3$
- ○ `res = 'euqalg'` et $i = 4$

3. (0.5 points) Quelle est la valeur retournée par la fonction suivante pour $n = 6$ et $p = 4$?

```
def g(n,p):
    if (p == 0) or (p == n):
        return -1
    return g(n - 1, p) + g(n - 1, p - 1) + 1
```

- √ **-1**
- ○ 16
- ○ 9
- ○ 5

4. (0.5 points) Dire laquelle des inclusions suivantes est correcte :
   - ◯ $\Theta(\log(n)) \subseteq \Theta(n)$
   - ◯ $\Omega(\log(n)) \subseteq \Omega(n)$
   - ◯ $\Omega(\log(n)) \subseteq O(n)$
   - √ $O(\log(n)) \subseteq O(n)$
5. (0.5 points) On rappelle que la suite de Fibonacci est définie par $F_0 = 0$, $F_1 = 1$ et $F_n = F_{n-1} + F_{n-2}$ si $n \geq 2$. On considère la fonction suivante, où n est un entier naturel.

```
def f(n):
    if n == 0: return 1
    if n == 1: return 3
    return f(n-1) + f(n-2) - 1
```

   Une seule affirmation est vérifiée, laquelle ?
   - ◯ Pour tout entier $n \geq 0$, l'appel $f(n)$ se termine et retourne $2F_n - 1$
   - ◯ La fonction ne se termine pas pour certains entiers $n \geq 0$.
   - √ **Pour tout entier $n \geq 0$, l'appel $f(n)$ se termine et retourne $2F_n + 1$.**
   - ◯ Pour tout entier $n \geq 0$, l'appel $f(n)$ se termine et retourne $F_n + 2$.
6. (0.5 points) On considère la fonction suivante, où n est un entier naturel.

```
def f(n):
    a = 1 ; k = 0
    while a <= n:
        a = a * 2 ; k = k + 1
    return k - 1
```

   La complexité de cette fonction est en :
   - ◯ $\Theta(n \log(n))$
   - ◯ $\Theta(n)$
   - ◯ $\Theta(\sqrt{n})$
   - √ $\Theta(\log(n))$
7. (0.5 points) La complexité du QuickSort appliqué à un tableau de taille n est en :
   - √ $\Omega(n \log(n))$ **et** $O(n^2)$
   - ◯ $\Theta(n \log(n))$
   - ◯ $\Theta(n^2)$
   - ◯ $\Omega(n)$ et $O(n \log(n))$
8. (0.5 points) On considère la fonction suivante, où n est un entier naturel.

```
def dixVersDeux(n):
    if (n==0)or(n==1):
        return n
    return n%2 + 10*dixVersDeux(n//2)
```

   Les opérations $n\%2$ et $n//2$ renvoient respectivement le reste et le quotient de la division de n par 2. Soit u_n le nombre de multiplications réalisées par la fonction pour un entier n. On a $u_0 = u_1 = 0$.
   Une seule affirmation est vérifiée, laquelle ?
   - ◯ $u_n = u_{\frac{n}{2}} + 1$ et la complexité est en $\Theta(n)$.
   - √ $u_n = u_{\frac{n}{2}} + 1$ **et la complexité est en $\Theta(\log n)$.**
   - ◯ $u_n = 2u_{n-1} + 1$ et la complexité est en $\Theta(\log n)$.
   - ◯ $u_n = 2u_{n-1} + 1$ et la complexité est en $\Theta(2^n)$.

9. (0.5 points) On appelle la fonction du tri rapide (Quicksort) vue en cours sur la liste $L = (7, 8, 3, 2, 10, 12, 3)$. Les premiers affichages obtenus sont les suivants :

```
Appel QS avec L= (7, 8, 3, 2, 10, 12, 3)
Appel QS avec L= [3, 2, 3]
Appel QS avec L= [2]
Valeur de retour L= [2]
Appel QS avec L= [3]
```

Quels sont les 6 lignes affichées qui suivent ?

√ **Valeur de retour L= [3]**
Valeur de retour L= [2, 3, 3]
Appel QS avec L= [8, 10, 12]
Appel QS avec L= []
Valeur de retour L= []
Appel QS avec L= [10, 12]

◯ Valeur de retour L= [3]
Appel QS avec L= [8, 10, 12]
Appel QS avec L= []
Valeur de retour L= []
Appel QS avec L= [10, 12]
Appel QS avec L= []

◯ Valeur de retour L= [3]
Appel QS avec L= [8, 10, 12]
Appel QS avec L= []
Appel QS avec L= [10, 12]
Appel QS avec L= []
Appel QS avec L= [12]

◯ Valeur de retour L= [3]
Valeur de retour L= [2, 3, 3]
Appel QS avec L= [8, 10, 12]
Appel QS avec L= [10, 12]
Appel QS avec L= []
Appel QS avec L= [12]

10. (0.5 points) On considère la fonction de tri suivante :

```python
def Sort(tab):
    i=1
    n = len(tab)
    while i != n:
        tmp = tab[i]
        j = i-1
        while j > -1 and tab[j] > tmp:
            tab[j+1] = tab[j]
            j=j-1
        tab[j+1] = tmp
        i=i+1
```

Quel est ce tri ?

√ **Le tri par insertion.**

- ◯ Le Quicksort.
- ◯ Le tri à bulles.
- ◯ Le tri par sélection.