Méthodes Algorithmiques – TD

Sophie Pinchinat – L3 Info Rennes 1

2015 – 2016, S2

TD 0

Exercice 1

Q	1	2	3	4	5	6	7	8	9
R	n	$n+1$	n	$2n$	n^2	n^3	n^3+n^2+2n	$\frac{n(n+1)}{2}$	$+\infty$

Exercice 2

1. Soit f telle que $f \in O(g)$; alors $\exists c_1 > 0$ tq $f(n) \leqslant c_1 g(n)$ pour $n > n_0$ et de plus comme $g \in \Theta(h)$, $\exists c_2, c_3 > 0$ tq $c_2 g(n) \leqslant h(n) \leqslant c_3 g(n)$ pour $n > n_1$. Il vient alors pour tout $n > \max(n_0, n_1), f(n) \leqslant c_1 g(n) \leqslant \frac{c_1}{c_2} h(n)$ et donc on a bien $f \in O(h)$.

2. Démonstration analogue.

3. Posons $f(n) = 3n^2, g(n) = n^2 eth(n) = e^n$; alors on a bien $f \in \Theta(g)$ et $g \in O(h)$ mais pourtant pas $f \in \Theta(h)$ (quelque soit c, il existe un n_0 à partir duquel $3cn^2 < e^n$, c'est assez convaincant).

Exercice 3

1. Avant une étape de la boucle, $s = \sum_{j=0}^{i-1} A[j]$; la boucle donne $s' = \sum_{j=0}^{i-1} A[j] + A[i] = \sum_{j=0}^{i} A[j]$ et $i' = i+1$ d'où on a bien $s' = \sum_{j=0}^{i'-1} A[j]$.

2. Avant la boucle, $s = i = 0$ donc l'invariant est trivialement vérifié.

3. Comme de plus chaque étape de boucle maintient l'invariant, et que la boucle se termine avec $i = n$, on a bien à la fin $s = \sum_{j=0}^{n-1} A[j]$. L'algorithme est correct.

4. La boucle principale s'exécute n fois pour un tableau de taille n, d'où le résultat immédiat.

Exercice 4

1. Maximum d'un tableau A de taille n :

```
1. m = A[0]; k = 0; i = 1
2. tant que i < n faire
3.     si A[i] > m
4.         m = A[i]
5.         k = i
6. retourner (m, k)
```

2. $m = \max_{0 \leqslant j < n} A[j] \wedge A[k] = m$

3. Soit $m = \max_{0 \leqslant j < i} A[j] \wedge A[k] = m$ notre invariant. Avant la boucle, il est vérifié : on a bien $m = \max_{0 \leqslant j < 1} A[j] = A[0]$ et $A[k] = A[0] = m$. En le supposant vrai au début d'une itération :

— soit $A[i] \leqslant m$, les valeurs ne changent pas et l'invariant est toujours vrai (la valeur initiale de m est toujours le maximum courant) ;

— soit $A[i] > m$ (la valeur lue est supérieure au maximum connu) et alors les valeurs sont mises à jour : m est le nouveau maximum et $k = i \Rightarrow A[k] = m$, et l'invariant est de nouveau vrai après l'itération.

4. L'invariant étant vérifié, à la fin de la boucle on a bien $m = \max_{0 \leqslant j < n} A[j] \wedge A[k] = m$ comme demandé en question 2. L'algorithme fonctionne.

5. Complexité :

Op	Comparaison	Affectation	Les deux
$T(n)$	n	$O(n)$	$O(n)$

Exercice 5

1. On peut diviser successivement le nombre par 2 en notant les restes à l'envers. Ou alors sinon on fait autre chose.

(...)

Exercice 6

1. Calcul de la puissance n de a :

```
1. r = a; i = 1
2. tant que i < n faire
3.     r = r*a
4. retourner r
```

2. Un invariant : $r = a^i$ (vrai au début avec $r = a^1 = a$, dans la boucle avec $r' = ra = a^i a = a^{i+1} = a^{i'}$).

3. Cet algorithme a une complexité de n.

4. La boucle itère $\lceil \log_2 m \rceil$ fois.

5. Trivialement vrai avant la boucle, puis si m impair : $\text{res}' b'^{m'} = \text{res} \times b (b^2)^{\lfloor m/2 \rfloor} = \text{res} \times b b^{m-1} = a^n$ et si m pair, $\text{res}' b'^{m'} = \text{res} \times (b^2)^{\lfloor m/2 \rfloor} = \text{res} \times b^m = a^n$.

6. L'invariant étant vérifié, à la fin de la boucle (pour $m = 0$) il l'est toujours d'où $\text{res} = \text{res} \times b^0 = a^n$: l'algorithme est correct.

Exercice 7

1. Au départ de la boucle, $res = \prod_{i=n+1}^{n} i = 1$; au cours d'une itération,

$$res' = res \times m = m \prod_{i=m+1}^{n} i = \prod_{i=m}^{n} i = \prod_{i=m'+1}^{n} i$$

(puisque $m' = m - 1$).

2. En fin de calcul on a donc $res = \prod_{i=1}^{n} i = n!$; le programme est correct.

3. Complexité : $T(n) = n$.

TD 1

Exercice 1

Complexité : $O(\log_2 n)$

Exercice 2

1. Résolution récursive des tours de Hanoï :

```
HANOI n ori dest buf :
si n = 1
    déplacer ori vers dest
sinon
    HANOI n-1 ori buf dest
    déplacer ori vers dest
    HANOI n-1 buf dest ori
```

2. On a la relation $T(n) = 2T(n-1) + 1, T(1) = 1$ ce qui donne pour complexité $O(2^n)$.

Exercice 3

1. Pour $n = 2 (r = 1)$, on a bien $Pav(T)$: dans la table à trou de dimension 2, où que soit le trou, il est complémentaire à un des quatre triominos par définition.

Supposons maintenant $Pav(T)$ pour $n = 2^r$; alors, pour $n' = 2^{r+1}$, toute table a un trou dans un de ses quadrants de taille 2^r et ce quadrant peut être pavé. On observe en outre que les trois quadrants restants peuvent être considérés comme trois tables à trou disposées en L, avec leurs trous accolés au centre. Ces trois quadrants peuvent être pavés, et le trou au centre a également la forme d'un triomino : on a bien $Pav(T)$ pour $n' = 2^{r+1}$.

2. Algorithme de pavage :

```
PAV r i j :
si r = 1
    placer le bon triomino
sinon
    paver le quadrant troué
    paver les trois autres
    boucher le dernier trou
```

3. Il faut que $n^2 \equiv 1[3]$, et alors le nombre de triominos utilisé est $\sqrt{\frac{n^2-1}{3}}$.

Exercice 4

```
bool dans-triangle (Point A, Point B, Point C, Point p):
    renvoyer même-côté (A, B, C, p)
        && même-côté (B, C, A, p)
        && même-côté (C, A, B, p)
```