Preuve et Notations asymptotiques [pf]
Exercices de cours

Karine Zampieri, Stéphane Rivière, Béatrice Amerein-Soltner

Unisciel algoprog Version 21 mai 2018

Table des matières

alg - Exercices de cours (Solution)

1 Appréhender le cours

1.1 Relations d'inclusions entre O

Mots-Clés Preuve et Notations asymptotiques ■
Utilise Notations asymptotiques ■

Objectif
Cet exercice manipule le grand-Oh.

Donnez les relations d'inclusion entre les ensembles suivants :
$O(n \log n)$, $O(2^n)$, $O(\log n)$, $O(1)$, $O(n^2)$, $O(n^3)$ et $O(n)$.

Solution simple
Trivialement $O(1) \subset O(n) \subset O(n^2) \subset O(n^3)$ puisque

$$\forall n \geq 1 : 1 \leq n \leq n^2 \leq n^3$$

On a également $O(1) \subset O(\log n)$ et $O(n) \subset O(n \log n)$ puisque

$$\forall n \geq 3 : 1 \leq \log n \quad (\text{donc } n \leq n \log n)$$

Ainsi que $O(\log n) \subset O(n)$ et $O(n \log n) \subset O(n^2)$ puisque

$$\forall n \geq 1 : \log n \leq n \quad (\text{donc } n \log n \leq n^2)$$

Enfin $O(n^3) \subset O(2^n)$ puisque

$$\begin{aligned} n^3 \leq 2^n &\Leftrightarrow \log n^3 \leq \log 2^n \\ &\Leftrightarrow 3 \log n \leq n \log 2 \end{aligned}$$

qui est vrai à partir d'une certaine valeur de n puisque $\log n \in O(n)$.

L'ordre d'inclusion est donc :

$$O(1) \subset O(\log n) \subset O(n) \subset O(n \log n) \subset O(n^2) \subset O(n^3) \subset O(2^n)$$

Que pouvez-vous en conclure ?

Solution simple
Un algorithme ayant une complexité logarithmique sera meilleur (en terme de complexité) qu'un algorithme linéaire, lui-même meilleur qu'un algorithme polynomial, à son tour meilleur qu'un algorithme exponentiel.

1.2 Durée d'exécution d'un algorithme

Mots-Clés Preuve et Notations asymptotiques ■
Utilise Notations asymptotiques ■

Objectif
Soit un ordinateur pour lequel toute instruction possède une durée de 10^{-6} secondes. On exécute un algorithme qui utilise $f(n)$ instructions avec $f(n)$ l'une des fonctions suivantes : $\log n$, $n \log n$, n, n^2, n^3 ou 2^n.

Sous un tableur,

Calculez le tableau suivant qui donne la durée d'exécution de l'algorithme en fonction de la taille $n = 10, 20, 30, 60$ et de la fonction $f(n)$.

f(n)	10	20	30	60
$\log n$				
n				
$n \log n$				
n^2				
n^3				
2^n				

Solution simple
Le tableau est calculé en utilisant un logarithme népérien.

Estimez approximativement le temps de $f(2^{60})$ en terme d'années.

Solution simple
On a 1.15×10^{12}s ≈ 36558 ans.

Tracez les courbes des fonctions.

2 Appliquer le cours

2.1 Validité de la fonction gerase

Mots-Clés Preuve et Notations asymptotiques ■
Utilise Validité d'un algorithme ■

Objectif
Soit l'algorithme suivant qui calcule le PGCD par la méthode de De Gérase de deux entiers positifs.

Fonction gerase

```
Fonction gerase ( a : Entier ; b : Entier ) : Entier
Variable x , y : Entier
Début
  |  x <- a
  |  y <- b
  |  TantQue ( x <> y ) Faire
  |    |  Si ( x > y ) Alors
  |    |    |  x <- x - y
  |    |  Sinon
  |    |    |  y <- y - x
  |    |  FinSi
  |  FinTantQue
  |  Retourner ( x )
Fin
```

Prouvez que la fonction calcule effectivement le PGCD de deux entiers a et b.

Solution simple
On dispose des propriétés mathématiques suivantes :

$$\begin{cases} \gcd(0,0) = 0 \\ \gcd(x,y) = \gcd(y,x) \\ \gcd(x,y) = \gcd(-x,y) \\ \gcd(x,0) = |x| \end{cases}$$

On peut donc se limiter au cas où a et b sont positifs. On sait aussi que :

$$\gcd(x,y) = \gcd(x-y,x)$$

En effet :

- Si p divise $x = p\,k$ et $y = p\,k'$, il divise $x - y = p(k - k')$ et y.
- Réciproquement, tout diviseur p de $x - y$ et de y divise x et y car :

$$x - y = p\,k'' \Leftrightarrow x = p\,k'' + p\,k' = p(\,k'' + \,k')$$

La méthode de De Gérase consiste à utiliser cette dernière propriété associée à la relation $\gcd(x, y) = \gcd(-x, y)$ pour construire deux suites récurrentes (x_i) et (y_i) telles que l'on ait toujours $p = \gcd(x_i, y_i)$. Il vient :

$$\begin{cases} x_{i+1} = x_i - y_i & \text{si } x_i > y_i \\ y_{i+1} = y_i - x_i & \text{si } y_i > x_i \end{cases}$$

Comme $x_0 = a$ et $y_0 = b$, on a aussi $p = \gcd(a, b)$.

2.2 Somme des éléments d'un tableau

Objectif
Soit l'algorithme :

Fonction sommeTab

```
Fonction sommeTab ( DR t : Entier [ TMAX ] ; n : Entier ) : Entier
Variable somme : Entier
Variable ix : Entier
Début
  |  somme <- 0
  |  Pour ix <- 1 à n Faire
  |    |  somme <- somme + t [ ix ]
  |  FinPour
  |  Retourner ( somme )
Fin
```

Montrez la terminaison de l'algorithme.

Solution simple
La terminaison est triviale puisque la boucle est exécutée `n` fois et que le corps de la boucle n'est constitué que d'une somme et d'une affectation, opérations qui s'exécutent en un temps fini.

Soit la propriété suivante : « A la fin de l'itération k, la variable `s` contient la somme des `k` premiers éléments du tableau `t` ». Prouvez que l'algorithme calcule la somme des `n` éléments du tableau `t`.

Solution simple
Notons s_k la valeur de la variable `s` à la fin de l'itération k, s_0 sa valeur initiale et effectuons un raisonnement par récurrence sur k.

Base La propriété est trivialement vraie pour $k = 1$ puisqu'à l'issue de la première itération, la variable `s` (initialisée à 0) contient la valeur $t[1]$:

$$s_1 = s_0 + t[1] = t[1] = \sum_{i=1}^{1} t[i]$$

Induction Supposons la propriété vraie jusqu'à l'itération k :

$$s_k = \sum_{i=1}^{k} t[i]$$

A la fin de l'itération $k+1$, la variable `s` contiendra la valeur qu'elle contenait à la fin de l'itération k (donc la somme des k premiers éléments) plus $t[k+1]$ qui lui a été ajouté à l'itération $k+1$ donc :

$$s_{k+1} = s_k + t[k+1] = \left(\sum_{i=1}^{k} t[i]\right) + t[k+1] = \sum_{i=1}^{k+1} t[i]$$

Initialement vraie, la propriété reste vraie à chaque nouvelle itération. Cette propriété est donc un invariant de l'algorithme. L'algorithme se terminant, à la fin de l'itération n, il aura donc bien calculé la somme des n éléments du tableau t.

Déterminez la complexité de l'algorithme.

Solution simple

On s'intéresse au nombre $C(n)$ d'additions effectuées par la fonction. De façon évidente, $C(n) = n$. On en déduit que l'algorithme est en $\Theta(n)$.

2.3 Produit matriciel

Propriété
Soient $A = (a_{ij})$ et $B = (b_{ij})$ deux matrices carrées de taille $n \times n$.

La multiplication de A et B est définie par

$$C = A \times B$$

avec :

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Écrivez une procédure `produitMatN(A,B,C,n)` qui calcule le produit matriciel d'ordre `n` d'une matrice `A` par une matrice `B` dans une matrice `C`, les matrices étant représentées sous forme d'un tableau de réels à deux dimensions de taille maximale `TMAX`.

Validez votre procédure avec la solution.

Solution alg @[UtilsMT.alg]

```
Action produitMat ( DR A , B : Matrice ; m , n , p : Entier ; R C : Matrice )
Variable ix , jx , kx : Entier
Variable somme : Réel
Début
  |  Pour ix <- 1 à m Faire
  |    |  Pour jx <- 1 à p Faire
  |    |    |  somme <- 0.0
  |    |    |  Pour kx <- 1 à n Faire
  |    |    |    |  somme <- somme + A [ ix , kx ] * B [ kx , jx ]
  |    |    |  FinPour
  |    |    |  C [ ix , jx ] <- somme
  |    |  FinPour
  |  FinPour
Fin
```

Montrez la terminaison de l'algorithme.

Solution simple
La terminaison est triviale puisque les boucles `Pour` sont exécutées chacune `n` fois et que les corps des boucles internes ne sont constitués que d'une somme et d'une affectation, opérations qui s'exécutent en un temps fini.

Calculez la complexité de cet algorithme.
Doit-on préciser dans quels cas (pire cas, meilleur cas, cas moyen) cette complexité est obtenue ?

Solution simple

Si l'on s'intéresse au nombre $C(n)$ d'additions et de multiplications effectuées par l'algorithme, on obtient de façon immédiate que $C(n) = n^3$. On en déduit que l'algorithme est en $\Theta(n^3)$. Il n'y a donc ici ni meilleur ni pire des cas.

Dupliquez puis modifiez la procédure `produitMatN` en la procédure modifiée `produitMat(A,B,m,n,p,C)` lorsque la matrice `A` est de dimension `(m,n)` et la matrice `B` de dimension `(n,p)`.

Validez votre procédure avec la solution.

Solution alg @[UtilsMT.alg]

```
Action produitMatN ( DR A , B : Matrice ; n : Entier ; R C : Matrice )
Variable ix , jx , kx : Entier
Variable somme : Réel
Début
  |  Pour ix <- 1 à n Faire
  |     |  Pour jx <- 1 à n Faire
  |     |     |  somme <- 0.0
  |     |     |  Pour kx <- 1 à n Faire
  |     |     |     |  somme <- somme + A [ ix , kx ] * B [ kx , jx ]
  |     |     |  FinPour
  |     |     |  C [ ix , jx ] <- somme
  |     |  FinPour
  |  FinPour
Fin
```

Quelle est alors la complexité de cet algorithme ?

Solution simple

Le nombre d'additions et de multiplications effectuées est cette fois-ci $C(n, m, p) = npm$. On en déduit que l'algorithme est en $\Theta(n\,p\,m)$.

3 Approfondir le cours

3.1 Preuve de la relation R2

Mots-Clés Preuve et Notations asymptotiques ■
Utilise Notations asymptotiques ■

Soient $f, g : \mathbb{N} \to \mathbb{N}$.
Prouvez que si $f \in O(g)$ alors $(f + g) \in O(g)$.

Solution simple
Si $f \in O(g)$ alors il existe un réel $c > 0$ et un entier $n_0 \geq 0$ tels que :

$$\forall n \geq n_0 : 0 \leq f(n) \leq c\, g(n)$$

On déduit de l'expression précédente que :

$$\forall n \geq n_0 : f(n) + g(n) \leq (c + 1)g(n)$$

Donc $f(n) + g(n)$ est en $O(g(n))$.

De même, prouvez que $O(f + g) = O(\max(f, g))$.

Solution simple
Pour montrer l'égalité de deux ensembles $E_1 := O(f + g)$ et $E_2 := O(\max(f, g))$, il faut montrer la double inclusion.

- Inclusion $E_1 \subset E_2$: Pour toute fonction $h \in E_1$, il existe $c > 0$ et $n_0 \geq 0$ tels que :

  $$\forall n \geq n_0 : h(n) \leq c\,(f(n) + g(n)) \leq 2c\ \max(f(n), g(n))$$

  Donc $h \in E_2$.
- Inclusion $E_2 \subset E_1$: Pour toute fonction $h \in E_2$, il existe $c > 0$ et $n_0 \geq 0$ tels que :

  $$\forall n \geq n_0 : h(n) \leq c\ \max(f(n), g(n)) \leq c\,(f(n) + g(n))$$

  Donc $h \in E_1$.

3.2 Preuve de la relation R3

Mots-Clés Preuve et Notations asymptotiques ■
Utilise Notations asymptotiques ■

Soient $f, g, S, T : \mathbb{N} \to \mathbb{N}$.
On suppose $S \in O(f)$ et $T \in O(g)$.

Prouvez que $S\,T \in O(fg)$.

Solution simple
Si $S \in O(f)$ et $T \in O(g)$ alors il existe $c > 0$, $c' > 0$, $n_0 \geq 0$ et $n'_0 \geq 0$ tels que :

$$\begin{aligned} \forall n \geq n_0 &\ : \ S(n) \leq cf(n) \\ \forall n \geq n'_0 &\ : \ T(n) \leq c'g(n) \end{aligned}$$

Si on définit $K := cc'$ et $m_0 := \max(n_0, n'_0)$, on a de façon évidente :

$$\forall n \geq m_0 : S(n)T(n) \leq K\ f(n)g(n)$$

Donc $S\,T$ est en $O(fg)$.

Concluez.

Solution simple
Si dans un algorithme, on a une première partie en $O(f(n))$ dans laquelle est imbriquée une seconde partie en $O(g(n))$, alors l'algorithme est globalement en $O(f(n)g(n))$.

3.3 Bornes asymptotiques

Mots-Clés Preuve et Notations asymptotiques ■
Utilise Notations asymptotiques ■

Soient $f, g : \mathbb{N} \to \mathbb{N}$.
Prouvez que si $f \in O(g)$ alors $g \in \Omega(f)$.

Solution simple
Si $f \in O(g)$ alors il existe $c > 0$ et $n_0 \geq 0$ tels que :

$$\forall n \geq n_0 : 0 \leq f(n) \leq c\, g(n)$$

On en déduit que :

$$\forall n \geq n_0 : \frac{1}{c} f(n) \leq g(n)$$

Donc g est en $\Omega(f)$.

Soient $f, g : \mathbb{N} \to \mathbb{N}$.
Montrez que $(f + g) \in \Theta(\max(f, g))$.

Solution simple
De façon évidente :

$$\forall n \geq 0 : \max(f(n), g(n)) \leq f(n) + g(n) \leq 2 \max(f(n), g(n))$$

Donc $(f + g)$ est en $\Theta(\max(f, g))$.

3.4 Relation O

Mots-Clés Preuve et Notations asymptotiques ■
Utilise Notations asymptotiques ■

Soient $f, g, S, T : \mathbb{N} \to \mathbb{N}$.
On suppose $S \in O(f)$ et $T \in O(g)$.

Prouvez que si $f \in O(g)$ alors $(S + T) \in O(g)$.

Solution simple
Si $S \in O(f)$ et $T \in O(g)$ alors il existe $c > 0$, $c' > 0$, $n_0 \geq 0$ et $n'_0 \geq 0$ tels que :

$$\begin{array}{rcl} \forall n \geq n_0 & : & S(n) \leq cf(n) \\ \forall n \geq n'_0 & : & T(n) \leq c'g(n) \end{array}$$

Si de plus $f \in O(g)$ alors il existe $c'' > 0$ et $n''_0 \geq 0$ tels que :

$$\forall n \geq n''_0 : f(n) \leq c''\, g(n)$$

On définit alors $K := c\, c'' + c'$ et $m_0 := \max(n_0, n'_0, n''_0)$. De façon évidente :

$$\forall n \geq m_0 : S(n) + T(n) \leq cf(n) + c'g(n) \leq (cc'' + c')g(n) = Kg(n)$$

Donc $(S + T)$ est en $O(g)$.

Concluez.

Solution simple
Si dans un algorithme, on a une première partie en $O(f(n))$ suivie séquentiellement d'une seconde partie en $O(g(n))$ et que $f(n) \in O(g(n))$, alors l'algorithme est globalement en $O(g(n))$.