SDA2 - TD2

Exemples: complexité et récursivité

Jean-Michel Dischler, Pascal Schreck

Complexité

Pour les fonctions f et g suivantes, vérifier les affirmations : $f = O(g)$, $f = \Omega(g)$ et $f = \Theta(g)$.

1. $f = n^2$, $g = n$

   (a) *Pour le cas :* $f = O(g)$
   c'est faut, en effet il faudrait que pour un n_0 *donné, on ait* $n^2 \leq c \times n$*. Ceci implique que* c *est plus grand que* n*, or* c *doit être une constante quelque soit* n*.*

   (b) *Pour le cas :* $f = \Omega(g)$
   c'est vrai, il suffit de prendre $n_0 = 1$, $c = 1$

   (c) *Pour le cas :* $f = \Theta(g)$
   c'est faux d'après ce qui précède

2. $f = \sqrt{n}$, $g = n$

   (a) *Pour le cas :* $f = O(g)$
   c'est vrai, il suffit de prendre $n_0 = 1$, $c = 1$

   (b) *Pour le cas :* $f = \Omega(g)$
   c'est faut, en effet il faudrait que pour un n_0 *donné, on ait* $n \leq c \times \sqrt{n}$*. Ceci implique que* c *est plus grand que* $\sqrt{n}$*, or* c *doit être une constante quelque soit* n*.*

   (c) *Pour le cas :* $f = \Theta(g)$
   c'est faux d'après ce qui précède

3. $f = 3n^2 + \sqrt{n}$, $g = n^2$

   (a) *Pour le cas :* $f = O(g)$
   c'est vrai, il suffit de prendre $n_0 = 1$, $c = 4$

   (b) *Pour le cas :* $f = \Omega(g)$
   c'est vrai, il suffit de prendre $n_0 = 1$, $c = 1$

   (c) *Pour le cas :* $f = \Theta(g)$
   c'est vrai d'après ce qui précède

Nombres parfaits

Un nombre est dit **parfait** s'il est égal à la somme de ses diviseurs (lui-même exclus). Exemple : $6 = 3 + 2 + 1$, c'est un nombre parfait.

1. Écrivez un algorithme itératif qui permet d'afficher tous les nombres parfaits entre 1 et n.

   PARFAIT*(n)*
     pour $j = 1$ ***à*** n ***faire***
       $s \leftarrow 1$
       pour $i = 2$ ***à*** $\frac{j}{2}$ ***faire***
         si $j \% i = 0$ ***alors*** $s \leftarrow s + i$
       si $s = j$ ***alors*** ***afficher*** j

2. Calculer le coût T(n) en nombre de divisions. En déduire que la complexité de cet algorithme en nombre de divisions est en $\Theta(n^2)$.
   Le coût est trivialement : $T(n) = \sum_{j=1}^{n} \frac{j}{2} - 1$, *soit* $T(n) = \frac{1}{4}n^2 - \frac{3}{4}n$.

Suite de Fibonacci

La suite de Fibonacci est définie comme suit :

$$\mathrm{Fib}(n) = \begin{cases} 1 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ \mathrm{Fib}(n-1) + \mathrm{Fib}(n-2) & \text{sinon.} \end{cases}$$

1. Écrivez un algorithme récursif calculant Fib(n).

   FIBONACCI(n)
   si $n = 0$ ou $n = 1$ **alors renvoyer** 1
   sinon renvoyer FIBONACCI($n-1$) + FIBONACCI($n-2$)

2. Compter le nombre d'appels à la fonction. Montrez que la complexité (en nombre d'additions) de cet algorithme est en $\Omega(2^{\frac{n}{2}})$.
   Le nombre d'appels correspond à la suite de Fibonacci elle-même. Pour évaluer la complexité, on procède par récurrence. On veut montrer qu'il existe une constante c strictement positive telle que $T(n) \geq c.2^{\frac{n}{2}}$*, pour des valeurs de n supérieures à une certaine borne* n_0 *(à déterminer). Supposons le résultat démontré jusqu'au rang* $n-1$*. Alors :*

   $$T(n) = T(n-1) + T(n-2) + 1 \geq c.2^{\frac{n-1}{2}} + c2^{\frac{n-2}{2}} + 1 \geq c.2^{\frac{n-2}{2}} + c.2^{\frac{n-2}{2}} + 1 \geq 2 \times c.2^{\frac{n-2}{2}} = c.2^{\frac{n}{2}}$$

   Il nous reste juste à montrer que cette équation est vraie « au départ ». Nous ne pouvons bien évidemment pas partir des cas $n = 0$ *et* $n = 1$*, puisque pour ces valeurs* $T(n) = 0$*. Nous partons donc des cas* $n = 2$ *et* $n = 3$ *(la récurrence nécessite* deux *valeurs de départ) :*
   - *Cas* $n = 2$ *:* FIBONACCI*(2)* = FIBONACCI*(1)* + FIBONACCI*(0), et* $T(2) = 1$*. Pour que la propriété désirée soit vraie, c doit donc vérifier :*

     $$1 \geq c.2^{\frac{2}{2}} = 2c \quad \Leftrightarrow \quad c \leq \frac{1}{2}$$

   - *Cas* $n = 3$ *:* FIBONACCI*(3)* = FIBONACCI*(2)* + FIBONACCI*(1), et* $T(3) = 2$*. Pour que la propriété désirée soit vraie, c doit donc vérifier :*

     $$2 \geq c.2^{\frac{3}{2}} = 2\sqrt{2}c \quad \Leftrightarrow c \leq \frac{\sqrt{2}}{2}$$

   Donc si $c = \frac{1}{2}$*, pour* $n \geq 2$*, on a* $T(n) \geq c2^{\frac{n}{2}}$ *et donc* $T(n) = \Omega(2^{\frac{n}{2}})$*.*

3. Écrire un algorithme récursif qui calcule, pour $n > 0$, le couple (FIBONACCI(n), FIBONACCI($n-1$)).

   FIB-PAIRE(n)
   si $n = 1$ **alors renvoyer** (1, 1)
   sinon (x, y) = FIB-PAIRE($n-1$)
   renvoyer $(x + y, x)$

4. Utilisez l'algorithme précédent pour écrire un nouvel algorithme calculant FIBONACCI(n).

   FIBONACCI(n)
   si $n = 0$ **alors renvoyer** 1
   sinon (x, y) = FIB-PAIRE(n)
   renvoyer x

5. Qu'elle est la complexité (en nombre d'additions) de cet algorithme ?
   La complexité de l'algorithme FIB-PAIRE*, en nombre d'additions, est donnée par la récurrence* $T(n) = 1 + T(n-1)$*. On a donc* $T(n) = n - 1$ *pour* FIB-PAIRE*, et par extension pour la nouvelle version de* FIBONACCI*.*

Nombre de combinaisons

Le nombre de combinaison C_n^p est défini comme suit :

$$\mathrm{C}(n,p) = \begin{cases} 1 & \text{si } n = 0 \ \vee \ n = p \\ \mathrm{C}(n-1,p) + \mathrm{C}(n-1,p-1) & \text{sinon.} \end{cases}$$

1. Écrivez un algorithme récursif calculant $\mathrm{C}(n,p)$.

   Combin(n, p)
   si $n = 0$ ou $n = p$ **alors renvoyer** 1
   sinon renvoyer Combin($n-1, p$) + Combin($n-1, p-1$)

2. Compter le nombre d'appels à la fonction. Montrez que la complexité dans le pire cas de cet algorithme est en $O(2^n)$.

   Faire un dessin de l'arbre binaire. On voit que dans le pire cas, il y a deux appels. Le nombre d'appels peut donc être borné par : $T(n) <= 2T(n-1) + 1$. *Donc, au pire,* $T(n) <= 1 + 2 + 4 + ... + 2^{n-1} = 2^n - 1$. *Notez que le calcul du nombre exact d'appels dépend non seulement de n mais aussi de p.*