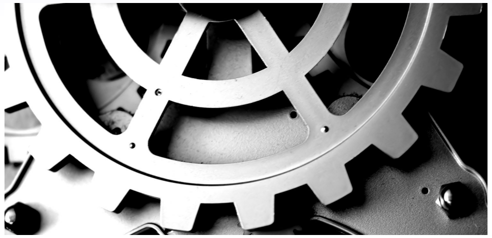


Algorithmique et structures de données
Complexité d'un algorithme récursif : rappels de méthode

Gaël Mahé

Université Paris Descartes

Licence 2

Étape 1 : écrire récursivement la cplxité

Algorithme 1 : Forme générale d'un algo Divide and Conquer

début

/* ENTRÉE : x, SORTIE : $f(x)$ */

opérations sur x

appel $f\big(T_1(x)\big)$

...

appel $f\big(T_p(x)\big)$

fusion, opérations sur x

fin

Hypothèses

- donnée x de taille n
- complexité des opérations hors appels récursifs : $C_{op}(n)$
- chaque $T_i(x)$ a une taille n/q

▶ $$C(n) = pC(n/q) + C_{op}(n)$$

Étapes 2 à 4,via les séries génératrices

2. Donner une forme plus classique à l'équation de récurrence :
   - $C(n) = pC(n/q) + C_{op}(n)$
   - On pose $n = 2^k$, $q = 2^\ell$: $C(2^k) = pC(2^{k-\ell}) + C_{op}(2^k)$
   - On pose $x(k) = C(2^k)$ et $u(k) = C_{op}(2^k)$: $x(k) = px(k-\ell) + u(k)$
3. Calculer la série génératrice de la suite x :
   - $x = px_\ell + u$
   - $X(z) = pz^\ell X(z) + U(z)$
   - $$X(z) = \frac{U(z)}{1 - pz^\ell}$$
4. En déduire $x(k)$, puis $C(n)$ $(k = \log_2(n))$

Étape 4 : exemple

Si par exemple $u(k) = \lambda 2^k$, $\ell = 1$ et $p \neq 2$ (NB : cas $p = 2$ déjà traité : voir tri fusion)

$$
\begin{aligned}
U(z) &= \lambda \sum_{k \geq 0} 2^k z^k = \frac{\lambda}{1 - 2z} \\
X(z) &= \frac{\lambda}{(1-2z)(1-pz)} = \frac{\alpha}{1-2z} + \frac{\beta}{1-pz} \\
&= \alpha \sum_{k \geq 0} 2^k z^k + \beta \sum_{k \geq 0} p^k z^k \\
&= \sum_{k \geq 0} (\alpha 2^k + \beta p^k) z^k
\end{aligned}
$$

Donc $x(k) = \alpha 2^k + \beta p^k$

Étape 4 : exemple (suite)

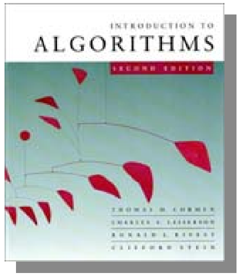
- $x(k) = \alpha 2^k + \beta p^k$
- $C(n) = x(k)$ pour $k = \log_2(n)$

Pour n grand,

- si $p = 1$, $x(k) \simeq \alpha 2^k$ ▶ $C(n) = \alpha n$
- si $p > 2$, $x(k) \simeq \beta p^k$ ▶ $C(n) = \beta p^{\log_2(k)} = \beta n^{\log_2(p)}$

Bilan

- On avait p appels récursifs sur des sous-ensemble de taille n/q, plus quelques opérations de complexité totale $C_{op}(n)$.
- Dans l'exemple, $C_{op}(n)$ n'affecte $C(n)$ que *via* α et β
- Souvent (comme ici), l'ordre de grandeur asymptotique de $C(n)$ ne dépend que de p/q

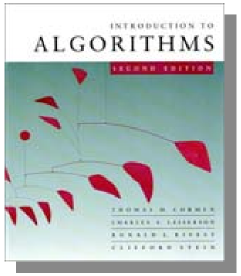


Matrix multiplication

$$\left.\begin{array}{ll}\textbf{Input:} & A = [a_{ij}],\ B = [b_{ij}]. \\ \textbf{Output:} & C = [c_{ij}] = A \cdot B.\end{array}\right\}\ i, j = 1, 2, \ldots, n.$$

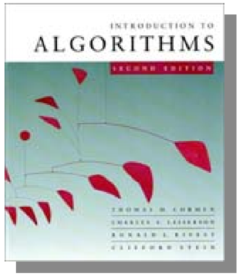
$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

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



Standard algorithm

for $i \leftarrow 1$ **to** n
    do for $j \leftarrow 1$ **to** n
        do $c_{ij} \leftarrow 0$
            for $k \leftarrow 1$ **to** n
                do $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$



Standard algorithm

for $i \leftarrow 1$ **to** n
  do for $j \leftarrow 1$ **to** n
    do $c_{ij} \leftarrow 0$
      for $k \leftarrow 1$ **to** n
        do $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$

Running time $= \Theta(n^3)$

Divide-and-conquer algorithm

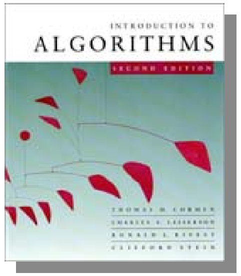
IDEA:

$n \times n$ matrix = 2×2 matrix of $(n/2) \times (n/2)$ submatrices:

$$\left[\begin{array}{c|c} r & s \\ \hline t & u \end{array}\right] = \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \cdot \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right]$$

$$C \quad = \quad A \quad \cdot \quad B$$

$$\left.\begin{aligned} r &= ae + bg \\ s &= af + bh \\ t &= ce + dg \\ u &= cf + dh \end{aligned}\right\} \begin{array}{l} 8 \text{ mults of } (n/2)\times(n/2) \text{ submatrices} \\ 4 \text{ adds of } (n/2)\times(n/2) \text{ submatrices} \end{array}$$



Divide-and-conquer algorithm

IDEA:

$n \times n$ matrix $= 2\times 2$ matrix of $(n/2)\times(n/2)$ submatrices:

$$\left[\begin{array}{c|c} r & s \\ \hline t & u \end{array}\right] = \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \cdot \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right]$$

$$C \quad = \quad A \quad \cdot \quad B$$

$$\left.\begin{aligned} r &= ae + bg \\ s &= af + bh \\ t &= ce + dh \\ u &= cf + dg \end{aligned}\right\}$$

recursive

8 mults of $(n/2)\times(n/2)$ submatrices

4 adds of $(n/2)\times(n/2)$ submatrices

Multiplication de matrices

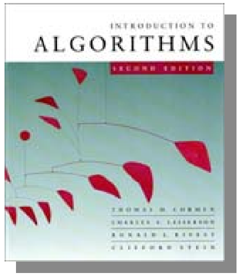
Algorithme 2 : Multiplication récursive de 2 matrices

début

/* ENTRÉES : 2 matrices A et B, de dimensions $n \times n$ */
/* SORTIE : produit matriciel AB */
$a \leftarrow \mathrm{hautgauche}(A)$
$b \leftarrow \mathrm{hautdroite}(A)$
$\ldots$
$P_1 \leftarrow \mathrm{mult}(a, c)$
$\ldots$
$P_8 \leftarrow \mathrm{mult}(d, h)$
$r \leftarrow \mathrm{add}(P_1, P_2)$
$s \leftarrow \mathrm{add}(P_3, P_4)$
$t \leftarrow \mathrm{add}(P_5, P_6)$
$u \leftarrow \mathrm{add}(P_7, P_8)$

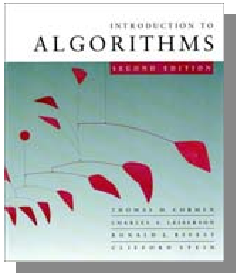
fin

Complexité : $C(n) = \Theta(n^3)$ (démonstration en TD)
Approche "diviser pour régner" sans intérêt ici.



Strassen's idea

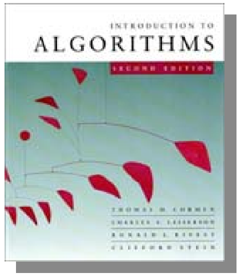
- Multiply 2×2 matrices with only 7 recursive mults.



Strassen's idea

- Multiply 2×2 matrices with only 7 recursive mults.

$$P_1 = a \cdot (f - h)$$
$$P_2 = (a + b) \cdot h$$
$$P_3 = (c + d) \cdot e$$
$$P_4 = d \cdot (g - e)$$
$$P_5 = (a + d) \cdot (e + h)$$
$$P_6 = (b - d) \cdot (g + h)$$
$$P_7 = (a - c) \cdot (e + f)$$

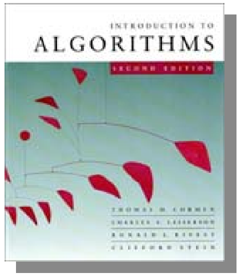


Strassen's idea

- Multiply 2×2 matrices with only 7 recursive mults.

$$
\begin{aligned}
P_1 &= a \cdot (f - h) \\
P_2 &= (a + b) \cdot h \\
P_3 &= (c + d) \cdot e \\
P_4 &= d \cdot (g - e) \\
P_5 &= (a + d) \cdot (e + h) \\
P_6 &= (b - d) \cdot (g + h) \\
P_7 &= (a - c) \cdot (e + f)
\end{aligned}
$$

$$
\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
s &= P_1 + P_2 \\
t &= P_3 + P_4 \\
u &= P_5 + P_1 - P_3 - P_7
\end{aligned}
$$



Strassen's idea

- Multiply 2×2 matrices with only 7 recursive mults.

$$
\begin{aligned}
P_1 &= a \cdot (f - h) \\
P_2 &= (a + b) \cdot h \\
P_3 &= (c + d) \cdot e \\
P_4 &= d \cdot (g - e) \\
P_5 &= (a + d) \cdot (e + h) \\
P_6 &= (b - d) \cdot (g + h) \\
P_7 &= (a - c) \cdot (e + f)
\end{aligned}
$$

$$
\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
s &= P_1 + P_2 \\
t &= P_3 + P_4 \\
u &= P_5 + P_1 - P_3 - P_7
\end{aligned}
$$

7 mults, 18 adds/subs.
Note: No reliance on commutativity of mult!