

INF 431



B. WERNER



Un exemple

$$\begin{array}{ll} \text{Minimiser} & c^T x \\ \text{Pourvu que} & Ax = B \\ \text{et} & l \leq x \leq u \end{array}$$

Programmation linéaire ; nombreuses applications.

Entre 1988 et 2004 :

- algorithme : 2360 × plus vite
- machines : 800 ×

total : $\approx 2.000.000 \times$

Traité en troisième année : Programmation par contraintes

source : Gilles Kahn et Bob Bixby ; interstices.info

Complexité

10 février 2010

Complexité = temps de calcul

classe de problèmes : temps en fonction de la taille de l'entrée

- Trier un tableau de taille n
- Multiplier deux matrices $m \times n$
- Trouver un mot (taille m) dans un dictionnaire (taille d)
- Ajouter un mot dans un dictionnaire
- Décoder un message secret (taille de clé RSA n)

Temps de calcul en fonction de n, m, d, \dots

Complexité de l'algorithme, complexité du problème

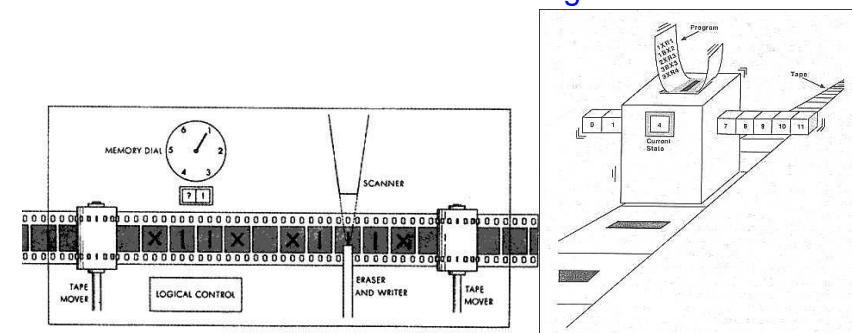
Temps pratique, temps théorique

En pratique, ça dépend :

Ordinateur/CPU, compilateur, vitesse du disque, etc. . .

Lorsqu'on veut être rigoureux :

On se ramène à une **machine de Turing**



On va approcher la réalité : le nombre d'opérations

Quelles opérations ? Ça dépend...

Nombres d'additions, multiplications, affectations dans un tableau...

Pas si facile

Maximum dans un tableau

```
int mint;
mint = t[0]; // 1 affectation
// n-1 affectations et comparaisons
for(int i = 1; i < t.length; i++)
    if(t[i] > mint) // n-1 comparaisons
        mint = t[i]; // F(t) affectations
```

La quantité $F(t)$ dépend fortement de t :

$$0 \leq F(t) \leq n - 1$$

et on peut montrer que sa valeur moyenne (sur tous les tableaux de taille n) vaut $\log n$

Pas si dur : traité en PC

- On s'intéresse à l'algorithme,
- pas à **tous** les détails du programme

On ne décrit pas l'algo/le programme en java, mais en pseudocode

```
rechercheMinimum(A)
    imin <- 0;
    pour i <- 1 à longueur(A)-1 faire
        si A[i] < A[imin] alors
            // on a trouvé un minimum local
            imin <- i;
    retourner A[imin];
```

Voir l'intro du poly.

Boucles emboîtées

Tri par sélection

```
final int l : t.length;
for (int i = 1; i < l; i++){
    j = selectMin(i, l);
    // l - i comparaisons
    exchange(i, j);
}
```

En tout : $l + (l - 1) + (l - 2) + \dots + 1 = \frac{n \times (n + 1)}{2}$ comparaisons
Ce qui est important : $t \simeq k.n^2$

Comportement asymptotique

$r_n = g(n)/f(n)$	$r_n \rightarrow 0$	$r_n < C$	$c < r_n < C$
Landau	$g(n) = o(f(n))$	$g(n) = O(f(n))$	$g(n) = \Theta(f(n))$

$c < r_n$	$r_n \rightarrow +\infty$
$g(n) = \Omega(f(n))$	$g(n) = \omega(f(n))$

- Complexité d'un algorithme :
 - ▶ Dans le pire cas
 - ▶ En moyenne
- Complexité d'un problème : On ne peut pas faire mieux que..

Un tri rapide : Quicksort

Algorithme historique : 1962

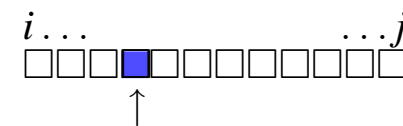
C.A.R. Hoare



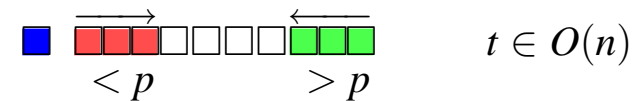
Diviser pour régner : Quicksort

On veut trier une portion de tableau, indices i à j

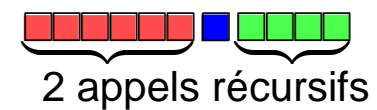
1. Sélection d'un pivot p :



2. Séparer :



3. Rappeler :



Complexité de Quicksort

Ca dépend...

1) Meilleur cas : on coupe au milieu à chaque fois

$$t(n) = k.n + 2.t(n/2) \Rightarrow t(n) = k.n.\log_2(n)$$

2) En moyenne :

$$t(n) = \Theta(n.\log_2(n)) \quad \text{également}$$

3) Pire cas, on choisit le max/min comme pivot :

$$t(n) = k.n + t(n-1) \Rightarrow t(n) = \Theta(n^2)$$

Que faire ?

En pratique :

- Changer d'algo (tri fusion...)
- Choisir le pivot aléatoirement (cf. PC)

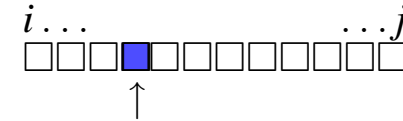
En théorie : peut-on trouver le meilleur pivot assez vite ?

Quickselect

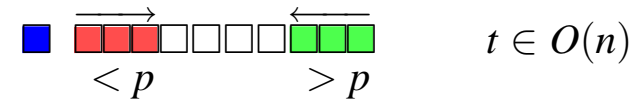
On veut juste trouver le k -ème élément du tableau trié

Cas particulier : trouver la médiane

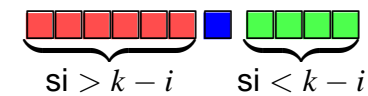
1. Sélection d'un pivot p :



2. Séparer :



3. Rappeler :

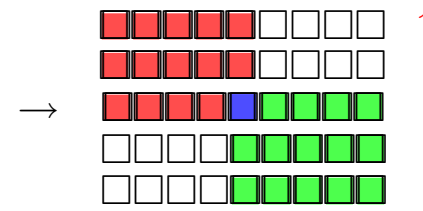


Au plus un appel récursif

Un peu d'astuce : médiane en temps linéaire (1973)

On cherche toujours le k -ème élément.

1. On range par paquets de 5



2. On trie chaque paquet

3. On calcule récursivement la médiane des médianes

4. On sépare suivant ce pivot

5. On fait un appel récursif ; on a éliminé au moins $3n/10$

$$t(n) \leq k.n + t(n/5) + t(7n/10)$$

miracle : $1/5 + 7/10 = 9/10 < 1$

$$t(n) = \Theta(n)$$

Complexité de Quickselect :

$$\text{En moyenne : } n + n/2 + n/4 + \dots = \Theta(n)$$

Au pire encore $O(n^2)$

Cela permet de faire un quicksort qui est toujours $n \cdot \log(n)$:

- On calcule la médiane en temps n
- On la choisit comme pivot
 - ▶ Séparation suivant la médiane en temps n
 - ▶ Deux appels récursifs sur des tableaux de taille $n/2$

En pratique, il vaut mieux choisir le pivot au hasard. . .

Diviser ne suffit pas

Autre présentation de l'algorithme naïf :

$$P = P_1 + P_2 \cdot X^{\frac{n}{2}}$$

$$Q = Q_1 + Q_2 \cdot X^{\frac{n}{2}}$$

$$\begin{array}{r}
 P_2 \quad Q_2 \\
 P_2 \quad Q_1 \\
 P_1 \quad Q_2 \\
 \hline
 P_1 \quad Q_1 \\
 \hline
 P \quad Q
 \end{array}$$

4 appels récursifs

Complexité : $t(n) = 4t(n/2) \Rightarrow t(n) = \Theta(n^2)$

Peut-on faire mieux ?

$$P = p_0 + p_1X + p_2X^2 + \dots + p_nX^n$$

$$Q = q_0 + q_1X + q_2X^2 + \dots + q_nX^n$$

Calculer :

$$P \cdot Q = r_0 + r_1X + r_2X^2 + \dots + r_{2n}X^{2n}$$

Naïvement :

$$r_i = \sum_{j=0}^i p_j q_{i-j}$$

n^2 multiplications élémentaires.

Anatolii Alekseïevitch Karatsuba 1937-2008



Algorithme de Karatsuba : 1960

On calcule :

$$A = P_1 Q_1$$

$$B = P_2 Q_2$$

$$C = (P_1 + P_2)(Q_1 + Q_2)$$

$$C = P_1 Q_1 + P_2 Q_2 + P_1 Q_2 + P_2 Q_1 = A + B + P_1 Q_2 + P_2 Q_1$$

$$R = B.X^n + (C - A - B).X^{\frac{n}{2}} + A$$

Trois appels récursifs seulement

$$t(n) = 3t(n/2) \Rightarrow t(n) = \Theta(n^{\log_2(3)}) \simeq n^{1,58}$$

Multiplication de Matrices : Strassen (1969)

Calculer $C = A \times B$

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \times \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} = \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

Naïvement :

$$C_{1,1} = A_{1,1} \times B_{1,1} + A_{1,2} \times B_{2,1}$$

$$C_{1,2} = A_{1,1} \times B_{1,2} + A_{1,2} \times B_{2,2}$$

$$C_{2,1} = A_{2,1} \times B_{1,1} + A_{2,2} \times B_{2,1}$$

$$C_{2,2} = A_{2,1} \times B_{1,2} + A_{2,2} \times B_{2,2}$$

8 appels récursifs

$$t(n) = 8t(n/2) \Rightarrow t(n) = k.n^3$$

C'est attendu : n multiplications pour obtenir chaque $c_{i,j}$

- C'est (presque) pareil pour multiplier des grands nombres
- On fait mieux depuis ; plus compliqué, mais même idée
- Des algorithmes "de Karatsuba" pour diviser des grands nombres, calculer la racine carrée, etc. . .
Parfois mathématiquement non-trivial
- On en trouve plein dans des bibliothèques comme GMP (GNU Multiprecision Library)

Strassen

$$\begin{aligned} M_1 &= (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2}) & M_4 &= A_{2,2}(B_{2,1} - B_{1,1}) \\ M_2 &= (A_{2,1} + A_{2,2})B_{1,1} & M_5 &= (A_{1,1} + A_{1,2})B_{2,2} \\ M_3 &= A_{1,1}(B_{1,2} - B_{2,2}) & M_6 &= (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2}) \\ M_7 &= (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2}) \end{aligned}$$

$$C_{1,1} = M_1 + M_4 - M_5 + M_7$$

$$C_{1,2} = M_3 + M_5$$

$$C_{2,1} = M_2 + M_4$$

$$C_{2,2} = M_1 - M_2 + M_3 + M_6$$

7 multiplications : $t(n) \in \Theta(n^{\log_2(7)}) \simeq \Theta(n^{2,807})$

Utilise seulement pour les très grandes matrices

Instabilité numérique

Conclusions

- Trouver le bon algorithme, c'est important
- Ca peut être subtil
- Connaître Karatsuba et Quicksort
- Savoir utiliser le hasard si nécessaire
- Une science assez jeune

La suite en PC