

Algorithmes récursifs

Diviser pour régner

Michel Van Caneghem

Novembre 2004

M02 - Complexité #9b - ©2004 - Michel Van Caneghem

Multiplication de matrices

Problème classique : $C = A \times B$ avec A et B matrices carrés d'ordre n .

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

```
for i:=1 to N do
  for j:=1 to N do begin
    C[i,j] := 0;
    for k:=1 to N do
      C[i,j] := C[i,j] + A[i,k]*B[k,j];
    end;
  end;
```

Complexité : $O(N^3)$

M02 - Complexité #9b - ©2004 - Michel Van Caneghem

1

Multiplication de matrices (2)

On pensait ne pas pouvoir faire mieux. En 1968 une autre méthode est proposée par Strassen. On regarde d'abord la multiplication de matrices d'ordre 2 :

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{pmatrix}$$

$$= \begin{pmatrix} x_1+x_2-x_4+x_6 & x_4+x_5 \\ x_6+x_7 & x_2-x_3+x_5-x_7 \end{pmatrix}$$

$$\begin{aligned} x_1 &= (b-d)(g+h) & x_5 &= a(f-h) \\ x_2 &= (a+d)(e+h) & x_6 &= d(g-e) \\ x_3 &= (a-c)(e+f) & x_7 &= (c+d)e \\ x_4 &= (a+b)h \end{aligned}$$

M02 - Complexité #9b - ©2004 - Michel Van Caneghem

2

Multiplication de matrices (3)

Il faut donc 7 multiplications et 18 additions au lieu de 8 multiplications et 4 additions.

C'est intéressant si le temps d'une multiplication est beaucoup plus grand que celui d'une addition.

Mais si nous cherchons à appliquer cette remarque sur les matrices alors il est évident que l'addition de 2 matrices est beaucoup plus simple et rapide ($O(N^2)$) que la multiplication de 2 matrices ($O(N^3)$).

M02 - Complexité #9b - ©2004 - Michel Van Caneghem

3

Multiplication de matrices (4)

Supposons que la taille des matrices soit une puissance de 2 : $N = 2^n$, alors on peut toujours diviser récursivement les matrices en 4 de la manière suivante :

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \text{ avec } A_{ij} \text{ de taille } \frac{N}{2} = 2^{n-1}$$

On peut alors écrire un programme de multiplication de 2 matrices qui utilisera les règles précédentes. Il utilisera les deux procédures suivantes :

$$C = \text{ProdMat}(A, B, N), \quad C = \text{AddMat}(A, B, N)$$

M02 - Complexité #9b - ©2004 - Michel Van Caneghem

4

Multiplication de matrices (5)

```
fonction ProdMat(A, B, N)
begin
  if N = 1 then ProdMat := A * B
  else begin Découper_en_4(A); Découper_en_4(B); end;
  X1 := ProdMat(SubMat(A12, A22, N/2),
                AddMat(B21, B22, N/2), N/2);
  ...
  X7 :=
  ...
  C22 := AddMat(SubMat(X2, X3, N/2), SubMat(X5, X7, N/2), N/2);
  Recompose(C);
end;
ProdMat(..., N) fait donc 7 appels à ProdMat(..., N/2)
et 18 appels à AddMat(..., N/2).
```

M02 - Complexité #9b - ©2004 - Michel Van Caneghem

5

Multiplication de matrices (6)

Calcul du nombre de multiplications : Avec $N = 2^n$ on a $\ln N = n \times \ln 2$. On trouve alors :

$$n_{mult} = 7^n = 7^{\frac{\ln N}{\ln 2}} = e^{\ln 7 \frac{\ln N}{\ln 2}} = N^{\frac{\ln 7}{\ln 2}} = N^{2.807}$$

Calcul du nombre d'additions : C'est un peu plus compliqué :

$$\begin{aligned} a_n &= 7a_{n-1} + 18 \times (N/2)^2 \\ &= 7a_{n-1} + \frac{9}{2} \times 4^n \end{aligned}$$

On pose alors $a_n = 7^n y_n$.

M02 - Complexité #9b - ©2004 - Michel Van Caneghem

6

Multiplication de matrices (7)

$$\begin{aligned} y_n &= y_{n-1} + \frac{9}{2} \times \left(\frac{4}{7}\right)^n \\ &= \frac{9}{2} \sum_{k=1}^n \left(\frac{4}{7}\right)^k \\ &\leq \frac{9}{2} \sum_{k=1}^{\infty} \left(\frac{4}{7}\right)^k = \frac{9}{2} \times \frac{1}{1 - 4/7} = \frac{21}{2} \\ y_n &\leq \frac{21}{2} \end{aligned}$$

et on trouve donc $n_{add} = O(7^n) = O(N^{2.807})$

M02 - Complexité #9b - ©2004 - Michel Van Caneghem

7

Multiplication de matrices (8)

En conclusion, le produit de matrice (et la résolution d'un système linéaire) peut être fait en $O(N^{2.807})$ opérations.

- ✓ On connaît des méthodes où l'exposant est 2,5 et on pense que la valeur optimale est $2 + \epsilon$.
- ✓ De toute façon la complexité minimale est en $O(N^2)$ car il faut pouvoir traiter $2N^2$ coefficients.
- ✓ Ce résultat présente surtout un intérêt théorique, car on a négligé le surcoût des opérations secondaires (Appels récursifs, découpages de matrices). Cette méthode n'est probablement intéressante que pour de très grosses matrices

La résolution des systèmes linéaires

Les résultats précédents supposaient que le temps d'une multiplication était constant. Ce n'est pas le cas : cela dépend de la taille des nombres manipulés. Nous avons alors les résultats suivants :

- Gauss en flottant : $O(n^3)$.
- Gauss en précision parfaite : $O(n^5 \log^2 n)$.
- Méthode modulaire : $O(n^4 \log n)$.
- Méthode P-Adique $O(n^3 \log^2 n)$.

Multiplication de polynômes

$$P = \sum_{i=0}^m a_i x^i \quad Q = \sum_{i=0}^n b_i x^i$$
$$PQ = \sum_{i=0}^{m+n} c_i x^i \quad \text{avec} \quad c_k = \sum_{i+j=k} a_i b_j$$

d'où le programme

```
for (i = 0; i <= m + n; i++) C[i] = 0;
for (i = 0; i <= m; i++)
  for (j = 0; j <= n; j++) C[i+j] += A[i]*B[j];
```

et sa complexité en $O(mn)$.

Multiplication de polynômes (2)

Même remarque que pour les matrices : On multiplie 2 polynômes de degré $N = 2^n$.

$$P = P_1 + x^{N/2} P_2 \quad Q = Q_1 + x^{N/2} Q_2$$
$$R_1 = P_1 \times Q_1 \quad R_2 = P_2 \times Q_2 \quad R_3 = (P_1 + P_2)(Q_1 + Q_2)$$
$$P \times Q = R_1 + (R_3 - R_2 - R_1)x^{N/2} + R_2 x^N$$

Il faut donc 3 multiplications au lieu de 4 (et 6 additions). On en déduit que :

$$C(N) = 3^n = N^{\frac{\log 3}{\log 2}} = N^{1.58}$$

Intéressant pour la multiplication de grands nombres.

Multiplication de polynômes (3)

Ce résultat est intéressant, mais on sait faire mieux !

Remarquons qu'un polynôme de degré n est entièrement défini par sa valeur en $n + 1$ points.

Donc pour calculer $R = P \times Q$ il suffit de calculer :

$$P(x_i) \quad i \in [0..2n] \quad Q(x_i) \quad i \in [0..2n]$$

$$R(x_i) = P(x_i) \times Q(x_i) \quad i \in [0..2n]$$

Mais il faut encore $O(n^2)$ multiplications !

Schéma de Horner

On peut évaluer la valeur d'un polynôme de degré N en un point avec N multiplications et N additions.

```
double value(C,n,x) {
  value = C[n];
  for (i=n-1; i>=0; i--) value=value*x+C[i];
  return value;
}
```

Ex : $x^3 + 2x^2 + 3x + 4$ au point 2

BIEN : $2 \times (2 \times (2 \times 1 + 2) + 3) + 4$

MAL : $2 \times 2 \times 2 + 2 \times 2 \times 2 + 3 \times 2 + 4$

Multiplication de polynômes (4)

Il est facile de passer des coefficients aux valeurs en $n + 1$ points. Pour passer des valeurs aux coefficients, il faut résoudre un système linéaire de dimension $n + 1$.

Il faut donc choisir un bon ensemble de points. Une brillante idée consiste à utiliser les racines nièmes de l'unité :

Ex : Racine quatrième :

$$z^4 = 1 \quad \{1, i, -1, -i\}$$

De manière générale :

$$\omega_k = e^{\frac{2\pi i k}{n}} \quad k \in [0..n-1]$$

Transformée de Fourier

Pour calculer la transformée de Fourier de la séquence :

$$a_0, a_1, \dots, a_{n-1}$$

On forme le polynôme de coefficients a_i et on calcule ses valeurs pour les racines nièmes de l'unité. Cette suite de valeurs est La Transformée de Fourier de la séquence :

$$f(t) = a_0 + a_1 t + \dots + a_{n-1} t^{n-1}$$

$$f(\omega_j) = \sum_{k=0}^{n-1} a_k \omega_j^k = \sum_{k=0}^{n-1} a_k e^{\frac{2\pi i j k}{n}}$$

L'idée géniale

Toujours la même : couper le problème en deux. Cette fois on va utiliser les termes de rangs pairs et ceux de rangs impairs. Si n pair :

$$f(\omega_j) = \sum_{k=0}^{n-1} a_k e^{\frac{2\pi i j k}{n}} \quad k = 2m \text{ ou } k = 2m + 1$$
$$f(\omega_j) = \sum_{m=0}^{n/2-1} a_{2m} e^{\frac{2\pi i j 2m}{n}} + \sum_{m=0}^{n/2-1} a_{2m+1} e^{\frac{2\pi i j (2m+1)}{n}}$$
$$f(\omega_j) = \sum_{m=0}^{n/2-1} a_{2m} e^{\frac{2\pi i j m}{n/2}} + e^{\frac{2\pi i j}{n}} \sum_{m=0}^{n/2-1} a_{2m+1} e^{\frac{2\pi i j m}{n/2}}$$

L'idée géniale (2)

Pour résumer pour calculer la transformée de Fourier de taille N on calcule :

- ✗ La transformée de Fourier ($N/2$) des coefficients pairs
- ✗ La transformée de Fourier ($N/2$) des coefficients impairs

Il reste un petit problème pour passer de $N/2$ valeurs à N valeurs. On profite de la périodicité de la transformée de Fourier. Si $J = j + N/2$ alors :

$$f(\omega_J) = \sum_{m=0}^{N/2-1} a_m e^{\frac{2\pi i (j+N/2)m}{N/2}} = e^{2\pi i m} \sum_{m=0}^{N/2-1} a_m e^{\frac{2\pi i j m}{N/2}}$$

Transformée de Fourier (2)

```
FFT(A,FA,N) {
  if (N == 1) {FA[0] = A[0]; return;}
  for (i = 0; i < N/2; i++)
    {Pair[i]=A[2*i]; Impair[i]=A[2*i+1];}
  FFT(Pair,FPair,N/2);
  FFT(Impair,FImpair,N/2);
  for (j = 0; j < N; j++) {
    omega = exp(2*Pi*I*j/N);
    FA[j]=FPair[j*N/2]+omega*FImpair[j*N/2];
  }
}
```

Transformée de Fourier (3)

Examinons maintenant la complexité :

$$T(N) = 2T(N/2) + N$$

$$T(N) = O(N \log N)$$

Il s'agit bien sur d'opération sur les nombres complexes.

Cet algorithme est appelé **Transformée de Fourier rapide** ou FFT (Fast Fourier Transform). Cet algorithme est d'une importance capitale pour le traitement du signal. Il a été inventé en **1965** par **Cooley et Tukey**

Multiplication de polynômes (5)

On a vu que la première phase : le calcul des valeurs, peut se faire avec une complexité de $O(N \log N)$

Pour Passer des valeurs aux coefficients on peut remarquer que :

$$f(\omega_j) = \sum_{k=0}^{n-1} a_k e^{\frac{2\pi i j k}{n}} \quad a_j = \frac{1}{N} \sum_{k=0}^{n-1} f(\omega_k) e^{\frac{-2\pi i j k}{n}}$$

- ① Prendre le conjugué de la suite des valeurs ;
- ② Appliquer la procédure FFT ;
- ③ Prendre le conjugué du résultat et le diviser par N

La complexité est encore de $O(N \log N)$

Multiplication de polynômes (6)

Pour multiplier deux polynômes P et Q de degré $N - 1$:

- ① Calculer les valeurs de P pour les racines $2N - 1$ ième de l'unité ;
- ② Calculer les valeurs de Q pour les racines $2N - 1$ ième de l'unité ;
- ③ Multiplier ces valeurs ;
- ④ Appliquer la FFT pour calculer les coefficients de PQ .

La complexité est bien de $O(N \log N)$

C'est la méthode de **Schönhage et Strassen (1971)** pour multiplier des grands nombres en un temps $O(N \log N)$.

Multiplication des grands nombres

- ✓ Multiplication normale : $O(n^2)$;
- ✓ En divisant par 2 (Karatsuba) : $O(n^{1.585})$;
- ✓ Algorithme de Toom - Cook : $O(n^{2.35\sqrt{\log n}})$;
- ✓ Méthode de Schönhage et Strassen : $O(n \log n \log \log n)$;
- ✓ Arithmétique modulaire (borné) : $O(n)$.

On peut remarquer que :

$$n^{2^{3.5\sqrt{\log n}}} = n^{1+3.5/\sqrt{\log n}}$$

qui bien sûr, croit plus vite que $\log n$!!!

Conclusion

- L'analyse des algorithmes est souvent complexe ;
- Mais les algorithmes obtenus sont très performants ;
- Utilisez des algorithmes simples et réguliers quand les performances ne sont pas nécessaires ;
- Méthode **diviser pour régner** :
 1. Diviser le problème en sous-problèmes ;
 2. Trouver une méthode pour résoudre le problème à partir des sous problèmes.