L'algorithmique, outil ou fondement en mathématiques

Jacques-Arthur Weil - XLIM, Université de Limoges

12 Novembre 2009

L'algorithmique en classe de seconde ?

- Point de vue d'un enseignant-chercheur, incompétent sur la classe de seconde..
- .. mais expérience de l'enseignement de l'algorithmique, ou avec de l'algorithmique (à l'université).
- Point de vue d'un spécialiste en calcul formel (algorithmique mathématique exacte).

L'algorithmique dans l'enseignement des maths

Éléments d'analyse d'algorithmes

- Correction
- Complexité et évaluation de performances

Quelques principes d'algorithmique mathématique

- Un exemple : multiplication rapide des polynômes
- Principes algorithmiques :
- Diviser pour règner
- Évaluation-interpolation
- Récursivité

L'algorithmique pour un enseignant-chercheur en mathématiques (à l'université)

- Dans la recherche : `http://www.xlim.fr/dmi`
  - Objet de recherche : Calcul numérique (optimisation numérique), Calcul exact (calcul formel, codage, cryptographie).
  - Outil pour la recherche, expérimentation mathématique (papier, crayon, livres, Maple), aide au calcul.
- Dans l'enseignement (Licence, Master) :
  - Objet d'enseignement (surtout en Master).
  - Outil pour l'enseignement (surtout en Licence).

Bien séparer : enseigner un principe algorithmique, ou utiliser des aspects algorithmiques pour faire passer une notion (expérimentation, découverte, etc).

I. Compte-rendu d'expériences d'enseignement

avec des ordinateurs en mathématiques

Apparition d'outils informatiques dans un cours de mathématiques

- Ordinateur comme support à l'enseignement (expérimentation, travaux pratiques).
- Enseigner un algorithme mathématique.
- Enseigner/approfondir des fondements algorithmiques (typage, correction, complexité, etc).

Apparaissent souvent dans cet ordre.

L'algorithmique dans les mathématiques de licence

- L'algorithmique en tant que telle :
  - principalement enseignée dans les cours d'informatiques,
  - quelques cours spécifiques (analyse numérique, arithmétique, etc) d'algorithmique mathématique.
- Travaux pratiques en licence pour découvrir et manipuler des objets mathématiques.
  - L'aspect algorithmique devient `outil` , mais pas `finalité` .
  - Expérimentations et `mise en responsabilité` , travail en groupe. Petits projets.

Expérimentation mathématiques avec des ordinateurs

- L'ordinateur comme `objet transitionel` : se concentrer sur l'écran (plus que sur papier ?) et l'objectif à atteindre.
- Objectif facile à atteindre (satisfaction). Susciter une réflexion critique derrière.
- Aspect algorithmique facile : l'algo. n'est pas `l'objet` , mais `l'outil` (donc ne doit pas être une barrière).
- Construire la séance sur un problème à résoudre : l'ordinateur est un auxilliaire et un `facilitateur` .
- S'affranchir des calculs lourds (faits par la machine) pour mieux évaluer leur but.
- Occasion de séances individualisées.
- Importance de l'interface : nos séances se passent mieux avec Maple qu'avec Scilab parce que l'interface plait mieux aux étudiants. (à retenir dans le choix d'un langage). Familiarité avec le logiciel.

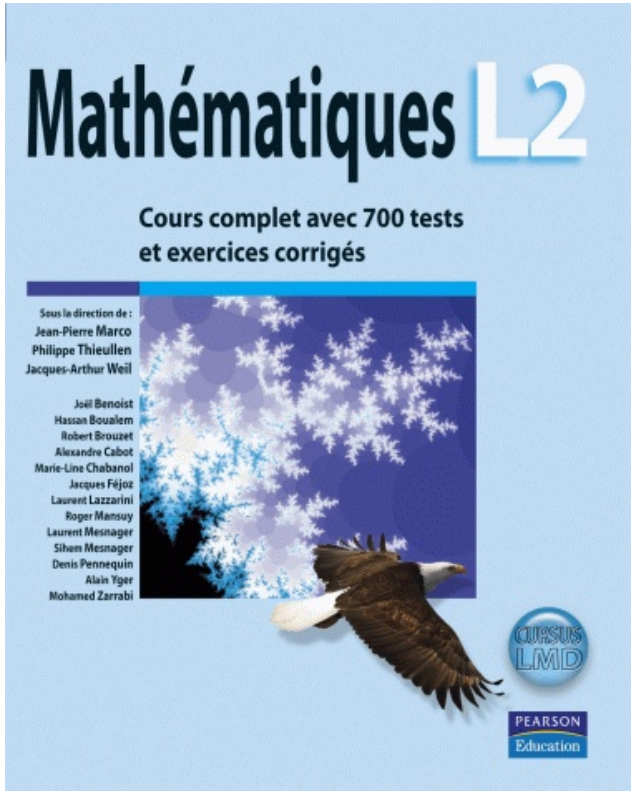
Expérimentation mathématiques : embuches courantes

Ingrédients d'une séance qui se passe mal.

- Difficultés avec la syntaxe : traduire son idée en algorithme puis en syntaxe. (utilité d'être familier avec le logiciel).
- Difficultés avec l'interface (se sentir à l'aise, limiter "l'hostilité de la machine").
- Séances où l'on mêle problème mathématique et difficulté algorithmique : bien séparer les deux (ou alors beaucoup guider).
- Objectif pas clair (but algorithmique ou but mathématique ?), ou trop dur.

II. Un peu d'algorithmique mathématique.

Principes et exemples.



Analyse d'un algorithme : l'algorithme d'Euclide étendu

Algorithme 1 Algorithme d'Euclide étendu

Entrées : $A, B \in \mathbb{N}$

Sorties : le pgcd D et $U, V \in \mathbb{Z}$ tels que $UA + VB = D$.

$R_0 := A$ et $R_1 := B$:

$U_0 := 1$, $U_1 := 0$; $V_0 := 0$: $V_1 := 1$.

$i := 1$;

tant que $R_i \neq 0$ **faire**

$Q_{i+1} := quo(R_{i-1}, R_i)$, quotient de la division euclidienne

$R_{i+1} := R_{i-1} - Q_{i+1} R_i$

$U_{i+1} := U_{i-1} - Q_{i+1} U_i$

$V_{i+1} := V_{i-1} - Q_{i+1} V_i$

$i := i + 1$

fin tant que

$i := i - 1$;

retourner R_i, U_i, V_i.

Definition

On dit qu'un algorithme est *correct* si les trois conditions suivantes sont remplies :

1. chaque étape est bien définie ;
2. l'algorithme se termine en un nombre fini d'étapes ;
3. le résultat est toujours celui qu'on attend.

Exemple : Euclide étendu est correct.

((difficile de trouver les erreurs : enseignement en soit))

Complexité et évaluation de performances

Mesures de performance :
temps d'exécution et quantité de mémoire nécessaire.

1. Coût en temps ,
2. Coût en mémoire ,
3. Coût en nombre d'opérations .

Évaluation de la *complexité a priori*
vs
évaluation de performances

Exemple : méthode de Horner pour évaluer un polynôme.

Méthode de Horner pour évaluer un polynôme.

Soit un polynôme $P = \sum_{i=0}^{n} p_i X^i \in \mathbb{K}[X]$ de degré n.
Nombre d'opérations nécessaires au calcul de $P(a)$?

`Méthode naïve` : $3n - 1$ opérations (et n cases mémoire).

`Méthode de Horner` :

$$P(a) = p_0 + a\,(p_1 + a\,(p_2 + a\,(\cdots + a\,(p_{n-1} + ap_n)\cdots))).$$

$f := p_n$ puis, pour i allant de 1 à n, on pose $f := af + p_{n-i}$.
Utilise $2n$ opérations (et une case mémoire).

Dans les deux cas : `coût linéaire` (ordre de grandeur du nombre d'opérations).

Le pivot de Gauss

Méthode du pivot de Gauss pour mettre un système de n équations linéaires à n inconnues sous forme triangulaire.

On ne peut pas calculer le nombre *exact* d'opérations nécessaires.
Mais on peut estimer son `ordre de grandeur :` $\mathcal{O}(n^3)$
(car $\sum_{i=1}^{n} i^2 = \frac{1}{6}\, n\,(n+1)\,(2\,n+1) = \mathcal{O}(n^3)$)

Remarque : le calcul d'un déterminant par la définition est en $\mathcal{O}(n.n!)$, calcul par Gauss en $\mathcal{O}(n^3)$.

Distinguer : complexité *en moyenne* et complexité *dans le pire des cas* .

Complexité d'Euclide (d'après Lamé, 1845)

Suite de Fibonacci définie par

$$F_0 = 0,\ F_1 = 1, \qquad F_{n+2} = F_{n+1} + F_n, n \geq 0.$$

Complexité de l'algorithme d'Euclide *dans le pire des cas* : suite des divisions $n = kq_1 + r_2$, $k = r_2 q_2 + r_3$, $\ldots, r_{i-1} = r_i q_{i+1} + r_{i+1}$. Elle se termine par $r_{m+2} = 0$. Majorer le nombre m d'additions à effectuer.

1. Le pire des cas se présente lorsque les quotients sont tous égaux à 1. Voir que r_i est alors le $(m+2-i)$-ième nombre de Fibonacci F_i défini ci-dessus.
2. Alors $n \geq F_{m+2} = (\phi^{m+2} - (1-\phi)^{m+2})/\sqrt{5}$ (où ϕ nombre d'or).
3. On en déduit $m + 2 \leq \log_\phi(n)$ (donc que l'algorithme d'Euclide est, dans le pire des cas, polynomial en le nombre de chiffres de n).

Quelques principes d'algorithmique mathématique.

Un exemple :
multiplication rapide des polynômes (Karatsuba, 1962)

Algorithme de Karatsuba (1962).

$f, g \in \mathbb{K}[X]$ deux polynômes de degré n ; de combien d'opérations (additions et multiplications) aurons-nous besoin pour calculer le produit $f.g$?

`Cas des polynômes de degré 1` :

si $f = f_0 + f_1 X$ et $g = g_0 + g_1 X$, alors

$$h = f_1\, g_1\, X^2 + (f_0\, g_1 + f_1\, g_0)\, X + f_0\, g_0;$$

`Coût` : une addition et quatre multiplications.

L'astuce de Karatsuba : calculer les *trois* produits $v_0 = f_0 g_0$, $v_1 = (f_0 + f_1)(g_0 + g_1)$, $v_2 = f_1 g_1$.

On a alors $h_0 = v_0$, $h_2 = v_2$, puis $h_1 = v_1 - v_0 - v_2$. On obtient donc le polynôme hg avec trois produits et quatre additions (dont deux soustractions).

Intérêt : le produit est plus coûteux que l'addition !

Algorithme de Karatsuba (suite).

On suppose $n = 2^k$. $f = \sum_{i=0}^{n} f_i X^i$ et $g = \sum_{i=0}^{n} g_i X^i$
$h := fg = \sum_{i=0}^{2n} h_i X^i$

`Produit naïf` : $\mathcal{O}(n^2)$ opérations.

$$f = F_0 + F_1 X^{\frac{n}{2}} \quad \text{et} \quad g = G_0 + G_1 X^{\frac{n}{2}},$$

F_i, G_i sont des polynômes de degré (au plus) $n/2$

$$h = (F_0 G_0) + X^{\frac{n}{2}} ((F_0 + F_1)(G_0 + G_1) - F_0 G_0 - F_1 G_1) + X^n (F_1 G_1).$$

Ainsi : produit de deux polynômes de degré n en trois produits de polynômes de degré $n/2$, deux additions en degré n et deux additions en degré $n/2$.
`Coût par Karatsuba :` $\mathcal{O}(n^{1.59})$ (mieux que n^2).

Coût de l'algorithme de Karatsuba.

Proposition

Notons $K(n)$ le nombre de multiplications dans $\mathbb{K}$ nécessaire à la multiplication de deux polynômes par le procédé de Karatsuba. Alors, $K(n) = \mathcal{O}(n^{1.59})$.

Supposons, pour simplifier l'écriture, que $n = 2^k$.
Nous avons vu que $K(n) = 3\mathbb{K}(n/2) = 3\mathbb{K}(2^{k-1})$
d'où, de proche en proche, $K(n) = 3^k\mathbb{K}(1) = 3^{k+1}$.
Or $n = 2^k$, donc $k = \frac{\ln(n)}{\ln(2)}$ et

$$3^k = e^{k\ln(3)} = e^{\left(ln(n)\frac{\ln(3)}{\ln(2)}\right)} = n^{\left(\frac{\ln(3)}{\ln(2)}\right)}.$$

Or $\frac{\ln(3)}{\ln(2)} \simeq 1.59$.

Principes algorithmiques sous-jacents

L'algorithme de Karatsuba met en œuvre trois principes algorithmiques fondamentaux :

1. Diviser pour régner (autre exemple : algorithmes de tri)
2. Évaluation-interpolation
   - évaluer les polynômes f et g en trois « points » (0, 1, et « l'infini »),
   - en déduire l'évaluation du produit fg en ces trois points,
   - puis reconstruire le produit fg à partir de ces trois valeurs

   (autre exemple : calculer modulo des nombres premiers et reconstruire par le théorème des restes chinois).
3. Récursivité.

Un autre exemple d'algorithme récursif : l'exponentiation binaire

Exponentiation binaire : calcul de g^n

Calcul de g^n.
Naïvement, n opérations $(g.g.\ldots.g.g)$. Faisons mieux.

Décomposons n en base 2 : $n = n_0 + 2n_1 + 4n_2 + \cdots + 2^k n_k$ où les $n_i \in \{0, 1\}$. Ainsi, g^n peut se réécrire

$$\begin{aligned} g^n &= g^{n_0} g^{2n_1} g^{4n_2} \ldots g^{2^k n_k} \\ &= g^{n_0} (g^2)^{n_1} (g^4)^{n_2} \ldots (g^{2^k})^{n_k} \end{aligned} .$$

L'élément g^n peut donc être obtenu en multipliant les g^{2^i} tels que $n_i = 1$. Le nombre total d'opérations est donc inférieur à deux fois le nombre de chiffres de n en base 2

Programme récursif pour l'exponentiation binaire.

`ExponentiationBinaire`(n,g) qui calcule g^n pour $n \in \mathbb{N}$

Algorithme 2 ExponentiationBinaire

Entrées : $n \in \mathbb{N}, g$
Sorties : g^n.

```
si n=0 alors
    renvoyer(1)
fin si
q, r := div(n, 2) {division euclidienne : n = 2q + r}
h := ExponentiationBinaire(q, g)^2
si  r=0 alors
    renvoyer(h)
sinon
    renvoyer( g.h)
fin si
```

Algorithme correct. Peut se programmer séquentiellement.

En guise de conclusion ouverte

Finalement, l'algorithmique est (pour moi) parrallèlement un outil et un fondement en mathématiques

pour la recherche et pour l'enseignement.

Outil très fructueux et champs de recherche passionant.