Complexité

Philippe Lac
(philippe.lac@ac-clermont.fr)
Malika More
(malika.more@u-clermont1.fr)

IREM Clermont-Ferrand

Stage Algorithmique

Année 2010-2011

Trois questions

Face à un algorithme

- Est-ce-qu'il donne un résultat ?
  ou bien est-ce-qu'il ne s'arrête jamais ?

  (Terminaison)
- Est-ce-qu'il donne le résultat attendu ?
  ou bien est-ce-qu'il calcule n'importe quoi ?

  (Correction)
- Est-ce-qu'il donne le résultat en un temps raisonnable ?
  ou bien est-ce-qu'il faut attendre plusieurs siècles ?

  Complexité

Résumé des épisodes précédents

On a démontré que des algorithmes

- ... donnaient le résultat attendu $\longrightarrow$ Correction
- ... en un nombre fini d'étapes $\longrightarrow$ Terminaison

Mathématiquement c'est satisfaisant

- Mais pratiquement c'est insuffisant
- Comment prévoir si le calcul effectif prendra un temps raisonnable ? $\longrightarrow$ Complexité (en temps)
- (Comment prévoir si le calcul effectif ne dépassera pas la mémoire disponible ? $\longrightarrow$ Complexité (en espace))

Retour sur l'exponentielle rapide

Fonction `ExpRap(`*a*,*n*`)`

début
- Donner à p la valeur 1
- Donner à b la valeur a
- Donner à m la valeur n
- **tant que** $m > 0$ **faire**
  - **si** m *impair* **alors**
    - Donner à p la valeur $p \times b$
  - **fin**
  - Donner à b la valeur b^2
  - Donner à m la valeur $\mathrm{quo}(m, 2)$
- **fin**
- **retourner** : p

fin

Principe de l'exponentielle rapide

$$5^{13} = \underbrace{5 \times 5 \times \ldots \times 5}_{13 \text{ fois}}$$

$$5^{13} = 5^{2^0+2^2+2^3} \qquad \text{i.e. } 13_{dec} = 1101_{bin}$$

$$= 5^{2^0} \times 5^{2^2} \times 5^{2^3}$$

$$= 5^{2^0} \times \left(\left(5^{2^0}\right)^2\right)^2 \times \left(\left(\left(5^{2^0}\right)^2\right)^2\right)^2 \qquad \text{car } 5^{2^{i+1}} = \left(5^{2^i}\right)^2$$

Exponentielle naïve

Fonction `ExpNaive(`*a*,*n*`)`

début
  Donner à p la valeur 1
  Donner à b la valeur a
  Donner à m la valeur n
  tant que $m > 0$ **faire**
    Donner à p la valeur $p \times b$
    Donner à m la valeur $m - 1$
    retourner : p
  fin
fin

Exponentielle naïve

Fonction `ExpNaive(`*a*,*n*`)`

début
  Donner à p la valeur 1
  Donner à b la valeur a
  Donner à m la valeur n
  tant que $m > 0$ **faire**
    Donner à p la valeur $p \times b$
    Donner à m la valeur $m - 1$
    retourner : p
  fin
fin

Nombre d'opérations arithmétiques effectuées par cet algorithme pour calculer a^n :

- Dans la boucle
  - 1 multiplication
  - 1 soustraction
- Nombre de passages dans la boucle
  - n passages pour calculer a^n
- Total :
  - n multiplications
  - n soustractions

Exponentielle rapide

Fonction `ExpRap(`*a,n*`)`

début
- Donner à p la valeur 1
- Donner à b la valeur a
- Donner à m la valeur n
- **tant que** $m > 0$ **faire**
  - **si** *m impair* **alors**
    - Donner à p la valeur $p \times b$
  - **fin**
  - Donner à b la valeur b^2
  - Donner à m la valeur $\mathrm{quo}(m, 2)$
- **fin**
- **retourner** : p

fin

Exponentielle rapide

Fonction `ExpRap(a,n)`

début
- Donner à p la valeur 1
- Donner à b la valeur a
- Donner à m la valeur n
- **tant que** $m > 0$ **faire**
  - **si** *m impair* **alors**
    - Donner à p la valeur $p \times b$
  - **fin**
  - Donner à b la valeur b^2
  - Donner à m la valeur $\mathrm{quo}(m, 2)$
- **fin**
- **retourner** : p

fin

Nombre d'opérations arithmétiques pour calculer a^n :

- Dans la boucle
  - Si m impair
    - 2 multiplications
    - 1 division
  - Si m pair
    - 1 multiplication
    - 1 division
  - Au plus :
    - 2 multiplications
    - 1 division

Exponentielle rapide

Fonction `ExpRap(a,n)`

début
- Donner à p la valeur 1
- Donner à b la valeur a
- Donner à m la valeur n
- **tant que** $m > 0$ **faire**
  - **si** *m impair* **alors**
    - Donner à p la valeur $p \times b$
  - **fin**
  - Donner à b la valeur b^2
  - Donner à m la valeur $\mathrm{quo}(m, 2)$
- **fin**
- **retourner** : p

fin

Nombre d'opérations arithmétiques pour calculer a^n :

- Nombre de passages dans la boucle
  - À chaque passage m est divisé par 2
  - $8 \to 4 \to 2 \to 1 \to 0$
  - $9 \to 4 \to 2 \to 1 \to 0$
  - $15 \to 7 \to 3 \to 1 \to 0$
  - 4 passages pour $2^3 \leq n \leq 2^4 - 1$
  - c'est-à-dire

$$\lfloor \log_2 n \rfloor + 1$$

passages

Exponentielle rapide

Fonction `ExpRap(a,n)`

début
  Donner à p la valeur 1
  Donner à b la valeur a
  Donner à m la valeur n
  tant que $m > 0$ **faire**
    si *m impair* **alors**
      Donner à p la valeur $p \times b$
    fin
    Donner à b la valeur b^2
    Donner à m la valeur $\mathrm{quo}(m, 2)$
  fin
  retourner : p
fin

Nombre d'opérations arithmétiques pour calculer a^n :

- Total : Au plus
  - $2|n|$ multiplications
  - $|n|$ divisions

  en notant $|n| = \lfloor \log_2 n \rfloor + 1$
- *Remarque* : $|n|$ est le nombre de chiffres de n en base 2, c'est-à-dire la taille de n écrit en binaire

Vérification expérimentale

Conséquence

Pour a fixé, le temps de calcul de a^n doit être

- proportionnel à n avec l'algorithme naïf
- proportionnel à $\log n$ avec l'algorithme rapide

$\longrightarrow$ `Scilab`

Remarque

L'exposant n doit être entier, mais on n'a fait aucune hypothèse sur a, qui peut donc être un entier, un flottant, ou même une matrice...

Complexité algorithmique

Objectif

- Estimer *a priori* le temps de calcul d'un algorithme

Complexité algorithmique

Objectif

- Estimer *a priori* le temps de calcul d'un algorithme

Remarque

- Indépendant des facteurs matériels
  - On ne va pas obtenir un résultat en secondes !
  - $\longrightarrow$ Compter les opérations élémentaires

Complexité algorithmique

Objectif

- Estimer *a priori* le temps de calcul d'un algorithme

Remarque

- Indépendant des données exactes
  - On a trouvé 5 multiplications pour calculer 5^{13}, mais pour 4^{14} ?
  - On cherche un résultat général
  - $\longrightarrow$ Majoration « dans le pire des cas »

Complexité algorithmique

Objectif

- Estimer *a priori* le temps de calcul d'un algorithme

Remarque

- Dépendant de la taille des données
  - Calculer 5^{13} et $5^{13000000}$, c'est différent
  - $\longrightarrow$ Le résultat est fonction de la taille des données

Complexité algorithmique

Objectif

- Estimer *a priori* le temps de calcul d'un algorithme

Remarque

- Indépendant des détails de l'implémentation
  - Il y a plusieurs façons de programmer le même algorithme
  - $\longrightarrow$ Se limiter à un ordre de grandeur

Complexité algorithmique

Objectif

- Estimer *a priori* le temps de calcul d'un algorithme

Résultat attendu

- Indépendant des facteurs matériels
  - Compter les opérations élémentaires
- Indépendant des données exactes
  - Majoration dans le pire des cas
- Dépendant de la taille des données
  - Fonction de la taille des données
- Indépendant des détails de l'implémentation
  - Se limiter à un ordre de grandeur

Complexité algorithmique

On recherche

Ordre de grandeur du nombre d'opérations élémentaires dans le pire des cas en fonction de la taille des données

Exemples

Complexité algorithmique

On recherche

Ordre de grandeur du nombre d'opérations élémentaires dans le pire des cas en fonction de la taille des données

Exemples

- Le nombre d'opérations arithmétiques de l'algorithme d'exponentielle rapide est proportionnel à la taille de l'exposant en binaire

  $\longrightarrow$ L'algorithme d'exponentielle rapide est en $\mathcal{O}(|n|)$, si n est l'exposant

Complexité algorithmique

On recherche

Ordre de grandeur du nombre d'opérations élémentaires dans le pire des cas en fonction de la taille des données

Exemples

- Le nombre de comparaisons et d'échanges de l'algorithme du tri par insertion est majoré par une constante fois le carré du nombre d'éléments à trier

  $\longrightarrow$ L'algorithme du tri par insertion est en $\mathcal{O}(n^2)$

Ordres de grandeur

Notation $\mathcal{O}$

$f(n) = \underset{n \to +\infty}{\mathcal{O}}(g(n))$ ssi $\exists N, C$ t.q. $\forall n > N \ |f(n)| \leq C\,|g(n)|$

Notation	Croissance
$\mathcal{O}(1)$	constante
$\mathcal{O}(\log(n))$	logarithmique
$\mathcal{O}((\log(n))c)$	polylogarithmique
$\mathcal{O}(n)$	linéaire
$\mathcal{O}(n\log(n))$	heu...
$\mathcal{O}(n^2)$	quadratique
$\mathcal{O}(n^c)$	polynomiale
$\mathcal{O}(c^n)$	exponentielle
$\mathcal{O}(n!)$	factorielle

Pire des cas

Signification

Il s'agit d'une majoration

Remarque

- On s'aperçoit parfois que les instances *difficiles*, c.-à-d. pour lesquelles le pire des cas se produit, sont rares
- Pour le tri par insertion, le pire des cas se produit quand le tableau d'entrée est trié dans l'ordre inverse
- En pratique, une notion de complexité « en moyenne » semblerait plus adaptée

Pire des cas

Exemple

L'algorithme de tri rapide est en $\mathcal{O}(n^2)$ dans le pire des cas (tableau déjà presque trié), mais en $\mathcal{O}(n \log n)$ en moyenne, ce qui en fait l'algorithme de tri le plus utilisé en pratique

Difficultés

- Quelle distribution de probabilité sur les entrées ?
- Analyse mathématique difficile
- $\longrightarrow$ Peu de résultats théoriques

Opérations élémentaires

Qu'est-ce-que c'est ?

Une notion à moduler en fonction du problème

Exemples

- Exponentielle $\longrightarrow$ opération arithmétique $+, \times$
- Tri $\longrightarrow$ comparaison, affectation
- Mais aussi $\longrightarrow$ appel à une fonction, etc.

Opérations élémentaires

En résumé

Choisir des opérations pertinentes vis-à-vis de l'algorithme

Remarque

En général c'est évident

- Pour l'algorithme d'exponentielle rapide, la multiplication de deux nombres est une opération élémentaire
- Pour l'algorithme de multiplication rapide de Karatsuba, ce n'est évidemment plus le cas

Opérations élémentaires

Hypothèse fondamentale

Chaque opération élémentaire prend un temps constant

Remarque

Par certains côtés c'est une approximation

- En théorie, multiplier deux grands nombres est plus long que multiplier deux petits nombres
- En pratique, les entiers et les flottants ont une taille fixe en machine
- Mais en calcul formel, on travaille en précision infinie...

Taille des données

Qu'est-ce-que c'est ?

Encore une notion qui dépend du contexte

Exemple

- Exponentielle $\longrightarrow$ nombre de chiffres de l'exposant
- Tri $\longrightarrow$ nombre d'éléments à trier

En résumé

Choisir des opérations pertinentes vis-à-vis de l'algorithme

Taille des données

Dans le cas des entiers

- En théorie, la taille d'un entier est son nombre de chiffres
- En pratique, les entiers informatiques ont une taille fixe (souvent 32 bits), et on peut considérer raisonablement que la taille d'un entier est... l'entier lui-même

Conséquence

Il faut préciser quelle convention on adopte :

- Dans le premier cas, on dira que l'algorithme d'exponentielle rapide est linéaire
- Dans le second cas, qu'il est logarithmique

À SUIVRE ...