

Algorithmique et Complexité

2. Quelques éléments pour l'analyse de la complexité

Nicole Bidoit

Université Paris XI, Orsay

Année Universitaire 2008–2009

Généralités

Objectifs : Evaluer (dans l'absolu) l'efficacité d'un algorithme, comparer des algorithmes, déterminer si un algorithme est le meilleur possible.

Rappel : modèle RAM

Mesure en temps : (Time complexity)

comptage de certaines opérations élémentaires/fondamentales (+, -, =, ...)

simplification du coût d'une opération : coût réel \rightarrow coût constant \rightarrow 1

\hookrightarrow pour toute opération op, coût(op)=1

... c'est un peu comme si on supposait que la terre était plate ... et c'est paradoxalement très utile

autre simplification : on ne considère pas toutes les opérations

Problème	opération "fondamentale"
recherche d'un élément e parmi n éléments	comparaison de e avec autres éléments
tri de n éléments	comparaison des éléments
multiplication de matrices $n \times n$	multiplication, addition

choix des opérations à considérer ? parfois délicat, jamais très compliqué

Généralités

Exemple : Algo voisin proche

Algo NNT

entrée : ensemble P de n points et leurs distances 2 à 2;

sortie : une suite $(p_i)_{i=0..(n-1)}$

```
choisir  $p$  dans  $P$ ; % initialisation
 $i \leftarrow 0$ ;  $p_i \leftarrow p$ ;  $P \leftarrow P - \{p_i\}$ 
while  $P$  non vide do % itération
    soit  $p$  dans  $P$  tel que  $p$  est le plus proche possible de  $p_i$  %  $\Leftarrow$  non élémentaire
     $i \leftarrow i+1$ ;  $p_i \leftarrow p$ ;  $P \leftarrow P - \{p_i\}$ 
```

L'exécution de **soit p dans P tel que p est le plus proche possible de p_i** dépend de la taille de P

Si P contient n points et est représenté par un tableau d'entiers ($T[1..n]$ of integer) de taille n ,

l'"instruction" s'écrit: choisir p dans $\{p \mid dist(p, T[i]) = \min\{dist(T[j], T[i]) \mid i+1 \leq j \leq n\}\}$.

Généralités

Exemple : Test de primalité (Théorème de Wilson)

Algo Wilson

entrée : entier n ;

sortie : booléen Prim (true si n est premier, false sinon)

```
if  $n$  divise  $(n - 1)! + 1$  sans reste then Prim  $\leftarrow$  true else Prim  $\leftarrow$  false;
```

les calculs de factorielle et de la divisibilité ne sont évidemment pas des opérations élémentaires!

détails de l'algorithme pour la correction

\neq

détails de l'algorithme pour l'analyse de complexité

Généralités

Exemple : Multiplication "Comme à l'école" et à la russe

Peut-on considérer l'addition et la multiplication comme des opérations élémentaires ?

oui si les opérandes sont de taille raisonnable

non si les opérandes sont "grands" (codage nécessitant plus de 32 bits ou 64 bits)

↔ la taille d'un "grand" entier est le nombre de mots nécessaire à son codage

Exercice : Quel est le coût des algorithmes de multiplication "Comme à l'école" et à la russe dans les cas suivants:

1. les opérandes sont de taille inférieure à un mot
2. les opérandes sont de grands entiers

Quel est le meilleur algorithme dans le cas 2. ?

Généralités

Mesure en espace :

(Space complexity)

Espace utilisé en plus de celui nécessaire pour coder l'instance (et le résultat).

simplification du coût en espace : codage variable $\leftarrow 1$ (espace "logique")

\hookrightarrow pour toute variable élémentaire V , $\text{coût}(V)=1$

... et on ne considère pas toutes les variables auxiliaires !

Attention : un tableau d'entiers $T[1..n]$ of integer n n'est pas une variable élémentaire

Problème	auxiliaire "fondamental"	espace (logique)
échange de valeurs	1 variable auxiliaire du "bon" type	1
parcourt d'un arbre	pile	profondeur de l'arbre

\hookrightarrow le compromis temps / espace

Taille de l'instance :

même hypothèse que ci-dessus : espace "logique"

Problème	Taille de l'instance
recherche d'un élément e parmi n éléments	Nbre n d'éléments [†]
tri de n éléments	Nbre n d'éléments à trier
multiplication de matrices $n \times n$	dimension n

[†] peu importe le type de l'élément

Complexité dans le meilleur, le pire des cas et en moyenne

Complexité(s) (en temps) d'un algorithme A pour le problème P ?

fonction numérique

Soit n un entier; Soit D_n l'ens. des instances I de P telles que $|I| = n$

- complexité de **A** pour une instance I de taille n :

$coût_A(I) =$ nbre d'opérations fondamentales exécutées par **A** avec l'entrée I .

- complexité de **A** dans le meilleur des cas

(Best-Case complexity)

fonction définie pour chaque n comme étant le plus petit nombre d'opérations fondamentales

exécutées par **A** pour une instance de taille n

$$Min_A(n) = \min\{coût_A(I) \mid I \in D_n\}$$

↔ minorant de la complexité

- complexité de **A** dans le pire des cas

(Worst-Case complexity)

fonction définie pour chaque n comme étant le plus grand nombre d'opérations fondamentales

exécutées par **A** pour une instance de taille n

$$Max_A(n) = \max\{coût_A(I) \mid I \in D_n\}$$

↔ majorant de la complexité

l'algo finit toujours avant $Max_A(n)$

cas pire \neq cas usuel

Complexité dans le meilleur, le pire des cas et en moyenne

- complexité de **A** en moyenne (Average-Case complexity)

fonction définie pour chaque n comme étant, le nombre moyen d'opérations fondamentales exécutées par **A** pour une instance de taille n

$$Moy_A(n) = \frac{\sum_{I \in D_n} \text{coût}_A(I)}{|D_n|}$$

↔ comportement général de l'algorithme ?

oui si les cas extrêmes sont rares

oui si la complexité varie "peu" en fonction des données

↔ prendre en compte la probabilité $Pr[I]$ d'apparition de chacune des instances de taille n

$$Moybis_A(n) = \sum_{I \in D_n} Pr[I] \cdot \text{coût}_A(I)$$

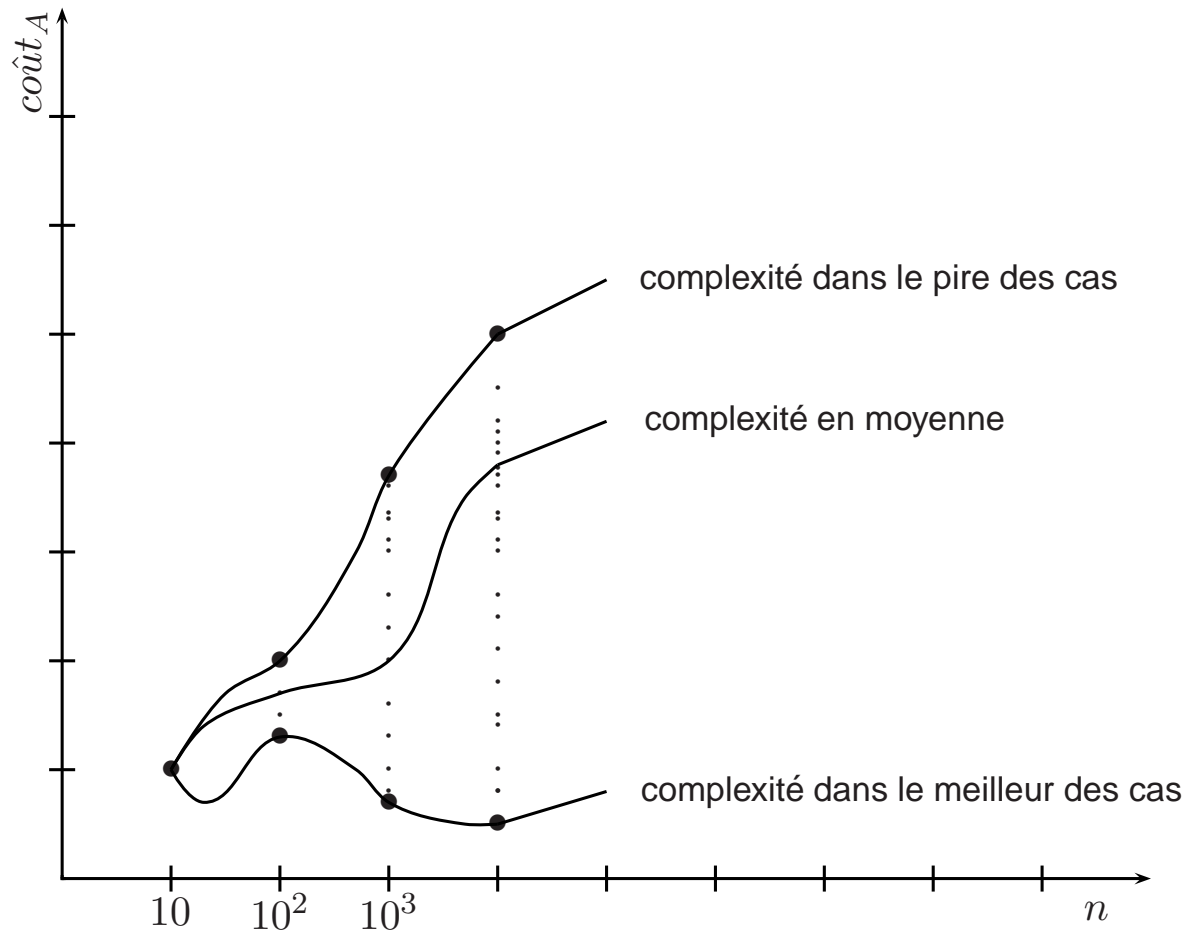
$Pr[I]$ vraiment très difficile à établir
et parfois difficilement exploitable

↔ partitionner D_n en sous-ensembles D_n^1, \dots, D_n^k d'instances partageant la même complexité i.e.

$$\forall i \in [1..k], \forall I, J \in D_n^i, \quad \text{coût}_A(I) = \text{coût}_A(J) = \text{coût}_A(D_n^i)$$

$$Moyter_A(n) = \sum_{i \in [1..k]} Pr[D_n^i] \cdot \text{coût}_A(D_n^i)$$

Complexité dans le meilleur, le pire des cas et en moyenne



↪ Les complexités dans le meilleur, le pire des cas et en moyenne sont parfois difficiles à établir exactement

Complexité(s) : exemple

Algorithme : recherche séquentielle d'un élément e dans un ensemble

Algo RechSeq

entrée : $T[1..n]$ of elt

e : elt

sortie : In boolean

auxiliaire : i integer; ok boolean

% une liste d'éléments

% élément recherché

% vrai si e dans T

% indice tableau

$i \leftarrow 1$; $ok \leftarrow false$;

% initialisation indice

while $i \leq n$ and $\neg ok$ **do**

% itération

if $(T[i] \neq e)$ **then** $i \leftarrow i+1$ **else** $ok \leftarrow true$;

% étape de parcours

if $i > n$ **then** $In \leftarrow false$ **else** $In \leftarrow true$

% résultat

Opération fondamentale = comparaison de e et d'un élément de la liste

Soit la liste du tableau $T =$

15	27	10	76	65	21	65	14
----	----	----	----	----	----	----	----

Considérons les instances suivantes (toute de taille $n = 8$) du problème de recherche:

$$I_1 = (T, e = 15) \quad \hookrightarrow \text{coût}_{\text{RechSeq}}(I_1) = 1$$

$$I_2 = (T, e = 50) \quad \hookrightarrow \text{coût}_{\text{RechSeq}}(I_2) = 8$$

$$I_3 = (T, e = 14) \quad \hookrightarrow \text{coût}_{\text{RechSeq}}(I_3) = 8$$

Complexité(s) : exemple

Algorithme : recherche séquentielle d'un élément e dans un ensemble.

- complexité dans le meilleur cas $Min_{RechSeq}(n) = 1$
- complexité dans le pire cas $Max_{RechSeq}(n) = n$
- complexité en moyenne ?

Supposer que toutes les instances de taille n sont équiprobables ne correspond pas toujours à la réalité : la recherche négative est un cas extrême fréquent dans une base de données ... mais pas sur le Web (-!)

Supposons que (1) tous les éléments de T sont distincts

(2) la probabilité que e soit dans T est $Pr[e \in T] = q$, et

(3) si e dans T alors toutes les places sont équiprobables.

$$\begin{aligned} \text{Pour } i = 1..n, \quad D_n^i &= \{(T, e) \mid T[i]=e\} & \text{coût}_{RechSeq}(D_n^i) &= i & Pr[D_n^i] &= \frac{q}{n} \\ \text{et } D_n^0 &= \{(T, e) \mid \forall i \in [1..n] T[i] \neq e\} & \text{coût}_{RechSeq}(D_n^0) &= n & Pr[D_n^0] &= 1-q \end{aligned}$$

$$\begin{aligned} Moyter_{RechSeq}(n) &= \sum_{i=0..n} Pr[D_n^i] \cdot \text{coût}_A(D_n^i) \\ &= (1-q) \cdot n + \sum_{i=1..n} \frac{q}{n} \cdot i = (1-q) \cdot n + \frac{q}{n} \sum_{i=1..n} i \\ &= (1-q) \cdot n + \frac{q}{n} \frac{n(n+1)}{2}, \\ &= (1-q) \cdot n + \left(\frac{n+1}{2}\right) \cdot q \end{aligned}$$

Complexité et (retour) correction : exemple

Algorithme : recherche séquentielle d'un élément e dans un ensemble.

Comment montrer que cet algorithme est correct pour la recherche ?

1. identifier des propriétés de l'algorithme (itération)

Notation : $i(k)$ = valeur de i juste avant l'exécution de la k ème itération.

propriété à (juste avant) itération k (invariant) : $i(k)=k$ et $\forall j \in [1..k[, T[j] \neq e$

sortie après k itérations (arrêt) : $(k \leq n \wedge T[k] = e)$ ou $k=n+1$

2. prouver ces propriétés

propriété à itération k :

cas de base ($k=1$) : $i(1) = k = 1$ et $[1..1[$ est vide

induction : on suppose que l'invariant est vrai pour k et on montre qu'il est vrai pour $k+1$

par hypothèse, on a $i(k) = k$ et $\forall j \in [1..k[, T[j] \neq e$; on a aussi $k \leq n$;

l'exécution de $i \leftarrow i+1$ implique donc que $i(k+1) = k+1$;

elle a été faite parce que $T[i(k)] = T[k] \neq e$ donc $\forall j \in [1..k+1[, T[j] \neq e$;

Réfléchir à quelques exercices

Exercice : on peut faire celui de la multiplication maintenant ...

Exercice : Ecrire un algorithme permettant de déterminer si deux éléments consécutifs d'une suite sont identiques.

Quelle est la complexité au pire ...

Exercice : Recherche bis

Refaire l'analyse de complexité de l'algorithme RechSeq en considérant toutes les opérations (affectation, incrémentation, ...) et en associant un coût distinct à chacune.

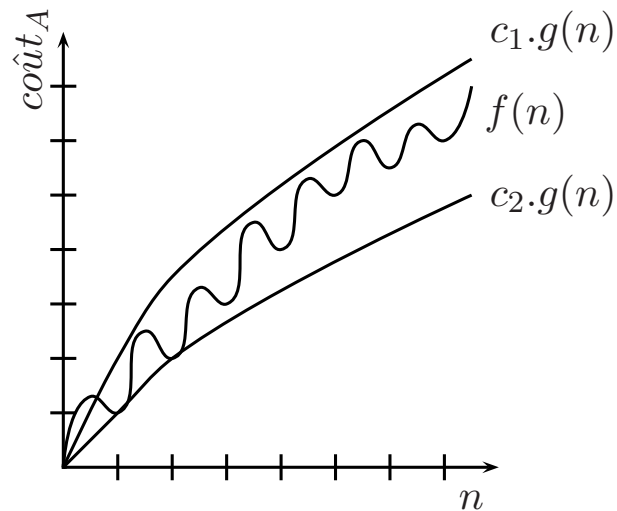
Complexité et Ordres de grandeur asymptotique

↪ La complexité $f(n)$ d'un algorithme est difficile à établir exactement

↪ Pour n grand, peu importe si $f(n) = n$ ou si $f(n) = n+75$

↪ Fréquemment, on cherche à comparer les complexités $f_A(n)$ et $f_B(n)$ de deux algorithmes A et B.

Pour n grand, quel est le meilleur algorithme si $f_A(n) = n^2$ et $f_B(n) = 3 \cdot 10^4 \cdot n$?



$O(g(n))$ est l'ensemble des fonctions "**majorées asymptotiquement**" par $g(n)$

La fonction $f(n)$ est dans $O(g(n))$ ssi

$$\exists c_1 \in \mathbb{R}^+, \exists n_1 \in \mathbb{N}, \text{ tels que: } \forall n > n_1, \quad f(n) \leq c_1 \cdot g(n)$$

$\Omega(g(n))$ est l'ensemble des fonctions "**minorés asymptotiquement**" par $g(n)$

La fonction $f(n)$ est dans $\Omega(g(n))$ ssi

$$\exists c_2 \in \mathbb{R}^{*+}, \exists n_2 \in \mathbb{N}, \text{ tels que: } \forall n > n_2, \quad c_2 \cdot g(n) \leq f(n)$$

$\Theta(g(n))$ est l'ensemble des fonctions "**de même ordre**" que $g(n)$

La fonction $f(n)$ est dans $\Theta(g(n))$ ssi

$$\exists c_1, c_2 \in \mathbb{R}^{*+}, \exists n_0 \in \mathbb{N}, \text{ tels que: } \forall n > n_0, \quad c_2 \cdot g(n) \leq f(n) \leq c_1 \cdot g(n)$$

Complexité et Ordres de grandeur asymptotique

Quelques exemples ... pour Grand Oh

$3n^2 - 100n + 6$ est dans $O(n^2)$, $3n^2 - 100n + 6$ est "majorée asympt." par $3.n^2$

$3n^2 - 100n + 6$ est dans $O(n^3)$, $3n^2 - 100n + 6$ est "majorée asympt." par $.01.n^3$

$3n^2 - 100n + 6$ n'est pas dans $O(n)$, en effet $c.n < 3.n^2$ pour $n > c$.

Quelques exemples ... pour Grand Omega

$3n^2 - 100n + 6$ est dans $\Omega(n^2)$, $3n^2 - 100n + 6$ est "minorée asympt." par $2.99.n^2$.

$3n^2 - 100n + 6$ n'est pas dans $\Omega(n^3)$, en effet $3n^2 - 100n + 6 < n^3$ pour $n > ?$.

$3n^2 - 100n + 6$ est dans $\Omega(n)$, $3n^2 - 100n + 6$ est "minorée asympt." par $10^{10^{10}}n$.

Quelques exemples ... pour Grand Theta

$3n^2 - 100n + 6$ est dans $\Theta(n^2)$, voir ci-dessus !

Exemple : Recherche Séquentielle

La complexité au mieux $Min_{RechSeq}(n)$ est dans $\Theta(1)$;

La complexité au pire $Max_{RechSeq}(n)$ est dans $\Theta(n)$;

La complexité en moyenne $Moyter_{RechSeq}(n)$ est dans $\Theta(n)$;

Exercice : Recherche Bis

Qu'en est-il ?

Complexité, Ordres de grandeur asymptotique, Dominance

Comparer les complexités $f_A(n)$ et $f_B(n)$ de deux algorithmes ?

1. Essayer de déterminer les ordres de grandeurs $\Theta(f(n))$ et $\Theta(g(n))$ de $f_A(n)$ et $f_B(n)$
2. Comparer les ordres de grandeurs comment ça?

↔ si $f_A(n)$ et $f_B(n)$ sont de même ordre ... le travail n'est pas fini

- regarder les algorithmes un peu plus en détails et
- comparaison hybride (programmer)

↔ par contre si $f_A(n)$ et $f_B(n)$ ne sont pas de même ordre, on peut en déduire le meilleur des deux.

Dominance et Quelques repères classiques

Admettre l'ordre des fonctions classiques suivantes :

$$1 < \log n < n < n \log n < n^2 < n^3 < 2^n < n!$$

< se lit "est négligeable"

Comprendre ce que ça veut dire ?

Comment établir que $g(n)$ est négligeable devant $f(n)$

$g(n)$ négligeable devant $f(n)$ si $g(n)$ croît strictement plus lentement que $f(n)$

autrement dit : $\lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = 0$

notation : $g(n) \in o(f(n))$

Complexité, Ordres de grandeur asymptotique, Dominance

Un tableau avec des valeurs de ces fonction pour **concrétiser** ... que :

les algorithmes exponentiels sont "sans fin" ... très vite

les algorithmes quadratiques (n^2) sont encore praticables jusqu'à $n = 10^6$

les algorithmes d'ordre $n \log n$ restent praticables jusqu'au billion 10^9

les algorithmes logarithmiques sont tout ce qu'on peut imaginer de mieux

n	$\lg n$	n	$n \lg n$	n^2	2^n	$n!$
10	0.003 μs	0.01 μs	0.033 μs	0.1 μs	1 μs	3.63 ms
20	0.004 μs	0.02 μs	0.086 μs	0.4 μs	1 ms	77.1 ans
30	0.005 μs	0.03 μs	0.147 μs	0.9 μs	1 sec	8.410 ¹⁵ ans
40	0.005 μs	0.04 μs	0.213 μs	1.6 μs	18.3 min	
50	0.006 μs	0.05 μs	0.282 μs	2.5 μs	13 $jours$	
10^2	0.007 μs	0.10 μs	0.644 μs	10 μs	4.10 ¹³ ans	
10^3	0.010 μs	1 μs	9.966 μs	1 ms		
10^4	0.013 μs	10 μs	130 μs	100 ms		
10^5	0.017 μs	0.10 ms	1.67 ms	10 sec		
10^6	0.020 μs	1 ms	19.93 ms	16.7 min		
10^7	0.023 μs	0.01 sec	0.23 sec	1.16 $jours$		
10^8	0.027 μs	0.10 sec	2.66 sec	115.7 $jours$		
10^9	0.030 μs	1 sec	29.90 sec	31.7 ans		

10^6 opérations par seconde ???

Détour par les Logarithmes

Un logarithme est la réciproque d'une fonction exponentielle

$$x = \log_b y \longleftrightarrow y = b^x$$

Intuitivement,

$\log_b y$ représente le *nombre de fois* qu'il est faut diviser y par b pour obtenir 1.

Plus concrètement,

Recherche dichotomique : recherche dans une liste triée

"Algorithme" : on se désintéresse de la moitié des éléments à chaque étape (après chaque comparaison)

Combien de fois peut on diviser n par 2 afin d'atteindre 1 ? $\lceil \log_2 n \rceil$!!!!!!!!!!!

Arbre binaire : Quelle est la hauteur nécessaire que doit atteindre un arbre binaire pour qu'il ait n feuilles ?

Combien de fois peut on diviser n par 2 afin d'atteindre 1 ? $\lceil \log_2 n \rceil$!!!!!!!!!!!

Représentation binaire : Combien de bits sont nécessaires pour coder les entiers entre 0 et $2^i - 1$?

chaque bit permet de doubler le nombre des entiers codables $\log_2(2^i) = i$!!!!!!!!!!!

Notation : $\log_2 y = \lg y$

Réfléchir à quelques Exercices

Addition, soustraction de fonctions et Grand Oh

Multiplication par une constante et Grand Oh

Trouver deux fonctions $f(n)$ et $g(n)$ (quand elles existent) qui satisfont les relations suivantes.

1. $f(n) \in \Theta(g(n))$ et $f(n) \in o(g(n))$
2. $f(n) \in \Theta(g(n))$ et $f(n) \notin O(g(n))$
3. $f(n) \in \Omega(g(n))$ et $f(n) \notin O(g(n))$

Montrer qu'il n'est pas utile de considérer la base d'un logarithme

$$\text{Rappel } \log_a n = \frac{\log_b n}{\log_b a}$$