

Algorithmique

Sandrine Vial
`sandrine.vial@uvsq.fr`

Octobre 2010

Introduction

3 niveaux d'abstraction

- Les **problèmes** décrits en langage naturel
- Les **algorithmes** décrits dans un pseudo-langage de programmation, proche du langage naturel
- Les **programmes** décrits dans un langage de programmation (C, Caml, Pascal, C++, Java, ...)

Niveaux de difficulté

Conceptuel

- Difficulté du problème ?
- Comment le résoudre ?
- Quelle démarche utiliser ?

Résolution du problème => Algorithme

Technique

- Comment mettre en œuvre mon algorithme sur une machine ?
- Quelles sont les ressources à ma disposition ?
- Quel est le langage le plus adapté ?

Questions relatives aux algorithmes

- Les sorties correspondent-elles à la solution de mon problème ?

Preuve de l'algorithme

- Combien de calculs élémentaires doit-on faire pour produire la sortie ?

Complexité en temps de l'algorithme

Complexité d'un algorithme

- **Mesure intrinsèque** de la complexité de l'algorithme indépendamment de l'implémentation.
- Permet la **comparaison** entre différents algorithmes pour un même problème.

Complexité d'un algorithme (2)

Recherche de l'algorithme avec le moins d'étapes élémentaires.

Somme des nombres de 1 à n

Idée 1 : Utilisation d'une boucle

Algorithme 1

$i \leftarrow 1$

$som \leftarrow 0$

Tant que $i \leq n$ Faire

$som \leftarrow som + i$

$i \leftarrow i + 1$

Fin Tant que

Coût : $2 \times n$ additions

Complexité d'un algorithme (3)

Somme des nombres de 1 à n

Idée 2 : Utilisation des mathématiques

$$\sum_{i=1}^n i = \frac{n \times (n + 1)}{2}$$

Algorithme 2

som ← *n* + 1

som ← *som* * *n*

som ← *som*/2

Coût : 1 addition, 1 multiplication et 1 division.

Complexité d'un algorithme (4)

Différentes Mesures

- Complexité en temps
- Complexité en espace

But

« *Sur toute machine, et quel que soit le langage utilisé, l'algorithme α est meilleur que l'algorithme β pour des données de grande taille.* »

Complexité d'un algorithme (5)

- Mesure élémentaire :
 - nombre de comparaisons
 - nombre d'affectations
 - nombre d'opérations arithmétiques
 - ...
- On cherche la complexité d'un algorithme \mathcal{A} en fonction d'un paramètre représentatif des entrées (taille).
- Attention : pas de système complet de règles.

Quelques règles

$\text{cout}(x)$: nbre d'op. élémentaires de l'ens. d'instructions x .

- **Séquence d'instructions** : $x_1; x_2$;

$$\text{cout}(x_1; x_2) = \text{cout}(x_1) + \text{cout}(x_2)$$

Exemple

Mesure : nombre d'opérations arithmétiques.

Algorithme \mathcal{A}

Début

$\text{som} \leftarrow n + 1$

$\text{som} \leftarrow \text{som} * n$

$\text{som} \leftarrow \text{som} / 2$

Fin

$$\text{cout}(\mathcal{A}) = \text{cout}(\text{som} \leftarrow n + 1) + \text{cout}(\text{som} \leftarrow \text{som} * n) + \text{cout}(\text{som} \leftarrow \text{som} / 2) = 3$$

Quelques règles

- Les boucles simples : *tant que condition faire* x_i ;

$$\text{cout}(\text{boucle}) = \sum_{i=1}^n (\text{cout}(x_i) + \text{cout}(\text{condition}))$$

Exemple

Mesure : nombre de comparaisons.

Algorithme \mathcal{A}

Début

❶ $i \leftarrow 1$

❷ $\text{som} \leftarrow 0$

Tant que $i \leq n$ *Faire*

❸ $\text{som} \leftarrow \text{som} + i$

❹ $i \leftarrow i + 1$

Fin Tant que

Fin

$$\text{cout}(\mathcal{A}) = \text{cout}(\text{❶}; \text{❷}) + \sum_{i=1}^n (\text{cout}(i \leq n) + \text{cout}(\text{❸}; \text{❹})) = n$$

Quelques règles

- **Conditionnelle** : *Si condition alors x_{vrai} ; sinon x_{faux} ;*

$$\text{cout}(\text{conditionnelle}) \leq \text{cout}(\text{condition}) + \max(\text{cout}(x_{vrai}); \text{cout}(x_{faux}))$$

Exemple

Mesure : nombre d'affectations.

Algorithme \mathcal{A}

Début

```
❶       $u \leftarrow 0$   
      Si  $i \bmod 2 = 0$  Alors  
❷           $u \leftarrow i/2$   
      Sinon  
❸           $u \leftarrow i - 1$   
❹           $u \leftarrow u/2$   
      Fin Si
```

Fin

$$\text{cout}(\mathcal{A}) \leq \text{cout}(\text{❶}) + \text{cout}(i \bmod 2 = 0) + \max(\text{cout}(\text{❷}); \text{cout}(\text{❸}; \text{❹})) = 3$$

Un exemple : produit de deux matrices carrées

$$C = A \times B$$

Deux matrices $A = (a_{ij})$ et $B = (b_{ij})$ de dimensions $n \times n$.

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

Algorithme 1 Multiplication de deux matrices carrées

Mult (A : matrice $n \times n$, B : matrice $n \times n$) :
matrice $n \times n$

Début

pour i de 1 à n faire

pour j de 1 à n faire

$C[i][j] \leftarrow 0$

pour k de 1 à n faire

$C[i][j] \leftarrow C[i][j] + A[i][k] * B[k][j]$

fin pour

fin pour

fin pour

retourner C

Fin

- Opération élémentaire : la multiplication.
- Coût de l'algorithme :

$$\sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n \text{cout}(C[i][j] \leftarrow \dots) = n^3$$

Grandeurs

- Caractérisation du comportement d'un algorithme \mathcal{A} sur l'ensemble des données D_n de taille n .
- $Cout_{\mathcal{A}}(d)$: coût de l'algorithme \mathcal{A} sur la donnée d .
- **Complexité dans le meilleur cas :**

$$Min_{\mathcal{A}}(n) = \min\{Cout_{\mathcal{A}}(d), d \in D_n\}$$

Grandeurs

- Caractérisation du comportement d'un algorithme \mathcal{A} sur l'ensemble des données D_n de taille n .
- $Cout_{\mathcal{A}}(d)$: coût de l'algorithme \mathcal{A} sur la donnée d .

- **Complexité dans le meilleur cas :**

$$Min_{\mathcal{A}}(n) = \min\{Cout_{\mathcal{A}}(d), d \in D_n\}$$

- **Complexité dans le pire cas :**

$$Max_{\mathcal{A}}(n) = \max\{Cout_{\mathcal{A}}(d), d \in D_n\}$$

Grandeurs

- Caractérisation du comportement d'un algorithme \mathcal{A} sur l'ensemble des données D_n de taille n .
- $Cout_{\mathcal{A}}(d)$: coût de l'algorithme \mathcal{A} sur la donnée d .

- **Complexité dans le meilleur cas :**

$$Min_{\mathcal{A}}(n) = \min\{Cout_{\mathcal{A}}(d), d \in D_n\}$$

- **Complexité dans le pire cas :**

$$Max_{\mathcal{A}}(n) = \max\{Cout_{\mathcal{A}}(d), d \in D_n\}$$

- **Complexité en moyenne :**

$$Moy_{\mathcal{A}}(n) = \sum_{d \in D_n} p(d) \times Cout_{\mathcal{A}}(d)$$

Remarques

- Comportement à l'extrême pour les complexités dans le meilleur et le pire cas.
- Complexité en moyenne : comportement de l'algorithme en général avec un modèle probabiliste sur les données.

$$Min_{\mathcal{A}}(n) \leq Moy_{\mathcal{A}}(n) \leq Max_{\mathcal{A}}(n)$$

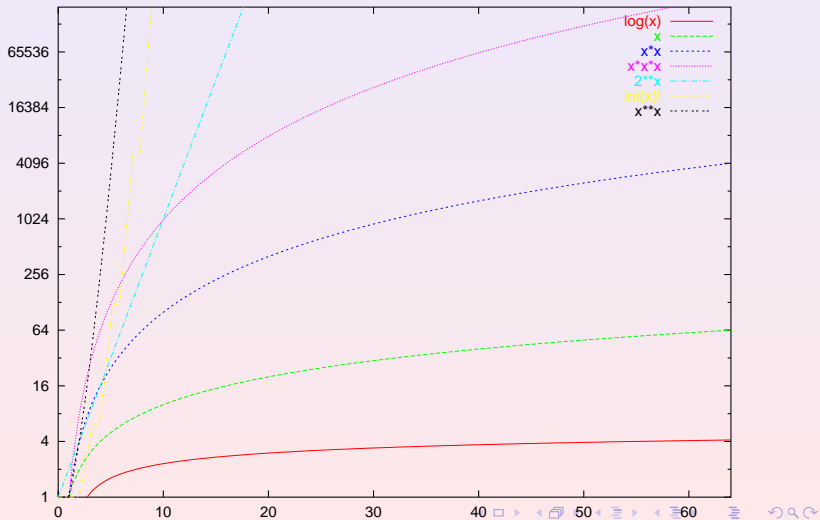
- exemple des matrices :

$$Min_{\mathcal{A}}(n) = Moy_{\mathcal{A}}(n) = Max_{\mathcal{A}}(n) = n^3$$

Ordres de grandeurs

- Une approximation de la fonction de complexité est suffisante.
- Utilisation d'une échelle de comparaison avec les fonctions n^n , $n!$, 2^n , n^3 , n^2 , $n \log n$, n , $\log n$

Ordres de grandeurs



Notations O

O « Borne Supérieure »

Soient f et $g : \mathbb{N} \rightarrow \mathbb{R}_+ : f = O(g)$ ssi $\exists c \in \mathbb{R}_+, \exists n_0 \in \mathbb{N}$ tels que :

$$\forall n > n_0, f(n) \leq c \times g(n)$$

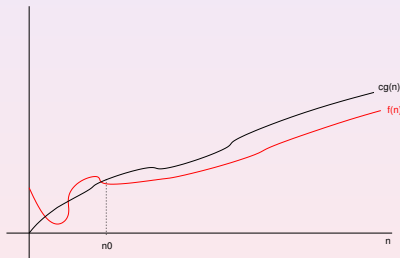


Figure: $f(n) = O(g(n))$

Notations Ω

Ω « Borne Inférieure »

Soient f et $g : \mathbb{N} \rightarrow \mathbb{R}_+ : f = \Omega(g)$ ssi $\exists c \in \mathbb{R}_+, \exists n_0 \in \mathbb{N}$ tels que :

$$\forall n > n_0, 0 \leq c \times g(n) \leq f(n)$$

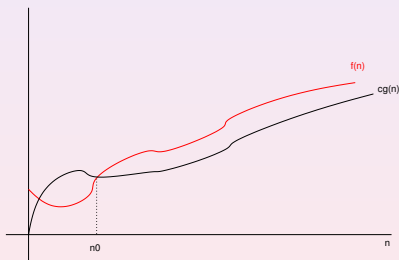


Figure: $f(n) = \Omega(g(n))$

Notation Θ

Θ

$$f = \Theta(g) \text{ ssi } f = O(g) \text{ et } f = \Omega(g)$$

$\exists c, d \in \mathbb{R}_+, \exists n_0 \in \mathbb{N}$ tels que :

$$\forall n > n_0, d \times g(n) \leq f(n) \leq c \times g(n)$$

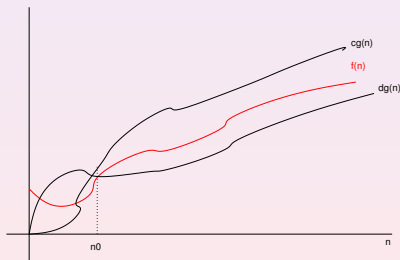


Figure: $f(n) = \Theta(g(n))$

Exemples

$$2n = O(n^2)$$

$$2n = O(n)$$

$$2n = \Theta(n)$$

$$2n \neq \Theta(n^2)$$

Un algorithme ...

- de complexité $O(1)$ effectue un nombre constant d'opérations
- de complexité $O(n)$ est un algorithme linéaire
- de complexité $O(n^k)$ est un algorithme polynomial

Quelques chiffres ...

		Complexité						
		1	$\log n$	n	$n \log n$	n^2	n^3	2^n
Taille des données	$n = 10^2$	1 μ s	6.6 μ s	0.1ms	0.6ms	10ms	1s	4.10 ¹⁶ a
	$n = 10^3$	1ns	9.9 μ s	1ms	9.9ms	1s	16.6min	∞
	$n = 10^4$	1 μ s	13.2 μ s	10ms	0.1s	100s	11.5j	∞
	$n = 10^5$	1 μ s	16.6 μ s	0.1s	1.66s	2.7h	31.7a	∞
	$n = 10^6$	1 μ s	19.9 μ s	1s	19.9s	11.5j	31.7 * 10 ³ a	∞

Temps d'exécution en fonction de la complexité d'un algorithme et de la taille des données.

$\infty = \ll > 10^{25}$ années »

Nombre d'opérations par seconde = 10^6

Quelques chiffres ...

		Complexité						
		1	$\log n$	n	$n \log n$	n^2	n^3	2^n
Taille des données	$n = 10^2$	1ns	6.6ns	0.1 μ s	6.6 μ s	10 μ s	0.001s	4.10 ¹³ a
	$n = 10^3$	1ns	9.9ns	1 μ s	9.9 μ s	0.001s	1s	∞
	$n = 10^4$	1ns	13.2ns	10 μ s	1.32ms	0.1s	1min 40s	∞
	$n = 10^5$	1ns	16.6ns	1ms	1.66ms	10s	11j 13h 46min 40s	∞
	$n = 10^6$	1ns	19.9ns	0.001s	0.019s	1min 40s	> 31a	∞

Temps d'exécution en fonction de la complexité d'un algorithme et de la taille des données.

$\infty = \ll > 10^{25}$ années »

Nombre d'opérations par seconde = 10^9

Quelques chiffres ...

		Complexité						
		1	$\log n$	n	$n \log n$	n^2	n^3	2^n
Taille des données	$n = 10^2$	1ps	6.64 ps	0.1 ns	0.66 ns	0.01 μs	1 μs	3.99 $10^{10} a$
	$n = 10^3$	1ps	9.96 ps	1 ns	9.96 ns	1 μs	0.001s	∞
	$n = 10^4$	1ps	13.28 ps	10 ns	132.87 ns	10 ms	1 s	∞
	$n = 10^5$	1ps	16.6 ps	0.1 μs	1.6 μs	0.01 s	> 16 min	∞
	$n = 10^6$	1ps	19.93 ps	1 μs	19.93 μs	1s	> 11 jours	∞

Temps d'exécution en fonction de la complexité d'un algorithme et de la taille des données.

$\infty = \ll > 10^{25}$ années »

Nombre d'opérations par seconde = 10^{12}