Structures de données et algorithmes

Béatrice Duval & Igor Stéphan

UFR Sciences Angers

2012-2013

Plan du cours de «Structures de données et algorithmique»

Complexité des algorithmes

L'exécution d'un programme mobilise les ressources de l'ordinateur.
La difficulté de l'exécution peut être caractérisée

- le temps de calcul pour exécuter les opérations et
- la place mémoire nécessaire (pour stocker les données et le programme en cours d'exécution).

L'analyse de la complexité consiste à mesurer ces 2 grandeurs pour comparer entre eux différents algorithmes, afin de choisir le mieux adapté (le plus rapide, le moins gourmand en place mémoire, ...)

On ne considère ici que **la complexité en temps**.

Complexité des algorithmes

La vitesse d'exécution dépend de la vitesse du microprocesseur mais cet aspect ne nous intéresse pas ici. On veut pouvoir dire :

Sur toute machine, quelque soit le langage de programmation, l'algorithme A1 est meilleur que A2 pour des données de grande taille

On va donc étudier la complexité des algorithmes et s'intéresser aussi aux données soumises à l'algorithme

Complexité des algorithmes

La complexité d'un algorithme dépend de la taille des données.

Examples

- Recherche d'un élément E dans une liste L
  $\rightarrow$ taille de la liste L dans laquelle on cherche E
- Tri d'une liste
  $\rightarrow$ taille de la liste à trier
- Calculer le produit d'une matrice par elle-même
  $\rightarrow$ dimension de la matrice dont on calcule le carré
- Recherche d'un motif de taille m dans un texte de n caractères
  $\rightarrow$ 2 tailles m et n à prendre en compte

La taille de données est importante pour caractériser la complexité : on la note généralement n et le temps d'exécution est alors noté $T(n)$. $T(n)$ est donc une fonction de $\mathbb{N}$ dans $\mathbb{R}^{*+}$

Complexité des algorithmes

On cherche ici une évaluation du temps d'exécution, pas une mesure précise.

- Mettre en évidence les opérations fondamentales pour le problème traité : le temps est proportionnel au nombre de ces opérations

Examples

- Recherche d'un élément E dans une liste L
  $\rightarrow$ nombre de comparaisons entre E et les éléments de L
- Recherche d'un élément sur disque
  $\rightarrow$ nombre d'accès à la mémoire secondaire
- Multiplications de 2 matrice de réels
  $\rightarrow$ nombre de multiplications et d'additions de réels

Complexité des algorithmes

Pour certains problèmes, le temps ne dépend pas seulement de la taille de la donnée mais aussi de la donnée elle-même.

Example

La recherche de E dans une liste dépend de la position de E dans cette liste

Definition

- Complexité au mieux
  $T_{min}(n) = min(Temps(d)|\text{d donnée de taille n})$
- Complexité au pire
  $T_{max}(n) = max(Temps(d)|\text{d donnée de taille n})$
- Complexité en moyenne $T_{moy}(n) = \sum p(d) * Temps(d)$
  où $p(d)$ est la probabilité de la donnée d de taille n.

Temps d'exécution d'un algorithme

On a bien sûr $T_{min}(n) \leq T_{moy}(n) \leq T_{max}(n)$

Example

Recherche séquentielle de E dans un tableau L de taille n.

- $T_{min}(n) = 1$ et $T_{max}(n) = n$
- $T_{moy}(n)$?
  - q la probabilité que E soit présent
  - D_{ni}, $1 \leq i \leq n$, ensemble de liste où E est à la i^{eme} place
  - D_{n0}, ensemble de liste où E n'est pas présent
  - $p(D_{ni}) = q/n$, $T(D_{ni}) = i$, $p(D_{n0}) = 1 - q$ et $T(D_{n0}) = n$
  - $T_{moy}(n) = \Sigma_{i=0}^{n} p(D_{ni}) * T(D_{ni}) = (1-q)n + \Sigma_{i=1}^{n} i * q/n = (1-q)n + (q/n) * ((n*(n+1))/2) = (1-q)n + (q*(n+1))/2$
  - si $q = 1$ alors $T_{moy}(n) = (n+1)/2$
  - si $q = 1/2$ alors $T_{moy}(n) = (3n+1)/4$

Ordre de grandeur asymptotique : la notation $\mathcal{O}$

On s'intéresse à $T(n)$ pour n grand et ce qui nous intéresse, c'est la rapidité de croissance de cette fonction T(n) lorsque n devient très grand : **ordre de grandeur asymptotique**
Le décompte exact du nombre d'instructions n'est pas le plus important. De même les constantes multiplicatives sont peu importantes.

Example

Si un algorithme A1 a un temps d'exécution $T1(n) = 25n$ et A2 a un temps d'exécution $T2(n) = 3n^2$, alors A1 est meilleur que A2 pour tout $n > 8$

La notation $\mathcal{O}$ permet d'étudier le comportement asymptotique de $T(n)$ sans tenir compte des constantes additives ou multiplicatives.

Ordre de grandeur asymptotique : la notation $\mathcal{O}$

Definition

Soit f et g deux fonctions de $\mathbb{N}$ dans $\mathbb{R}^{*+}$

$$f = \mathcal{O}(g)$$

ssi

$$\exists c \in \mathbb{R}^{*+}, \exists n_0 \in \mathbb{N} \text{ tq } \forall n > n_0, f(n) \leq cg(n)$$

$f = \mathcal{O}(g)$ doit se lire "f est en grand O de g". On trouve aussi parfois la notation $f \in \mathcal{O}(g)$ qui exprime que f appartient à la classe des fonctions majorées asymptotiquement par g

Example

Si $f(n) = 10n$ et $g(n) = n^2$ on a bien $f = \mathcal{O}(g)$.

Equivalent asymptotique : la notation Θ

Definition

Soit f et g deux fonctions de $\mathbb{N}$ dans $\mathbb{R}^{*+}$

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

c'est-à-dire

$$\exists c \in \mathbb{R}^{*+}, \exists d \in \mathbb{R}^{*+}, \exists n_0 \in \mathbb{N} \text{ tq } \forall n > n_0, dg(n) \leq f(n) \leq cg(n)$$

Example

$10n = \Theta(n)$ mais $10n$ n'est pas en $\Theta(n^2)$

En pratique

- En pratique on va chercher un majorant du temps d'exécution et on utilisera la notation $\mathcal{O}$ pour cela.
- Il faudra veiller à chercher le plus petit majorant possible pour bien évaluer les performances de l'algorithme.
- La notation Θ est plus précise mais nous obligerait à des évaluations plus fines.

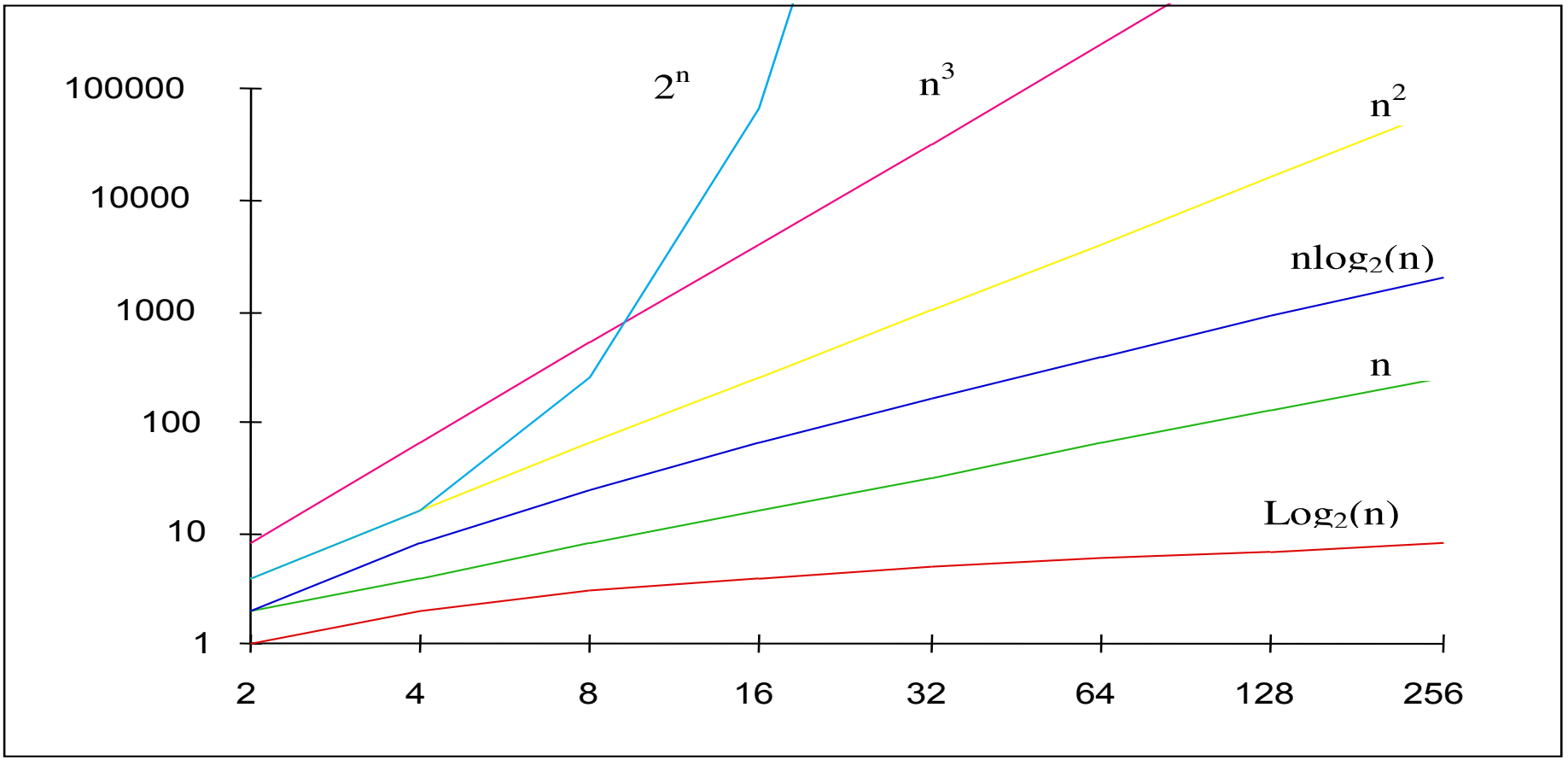
Propriétés de le notation $\mathcal{O}$

- Les constantes ne sont pas importantes
- Dans un polynome, les termes d'ordre inférieur sont négligeables
  Si $T(n) = a_k n^k + \cdots + a_1 n + a_0$ avec $a_k > 0$ alors $T(n) = \mathcal{O}(n^k)$
- Si $\lim_{n\to\infty} \frac{h(n)}{g(n)} = 0$ alors $g + h = \mathcal{O}(g)$
- Si $f = \mathcal{O}(g)$ et $g = \mathcal{O}(h)$ alors $f = \mathcal{O}(h)$
- Si $T_1(n) = \mathcal{O}(f_1)$ et $T_2(n) = \mathcal{O}(f_2)$ alors
  $T_1(n) + T_2(n) = \mathcal{O}(max(f_1, f_2))$
  Par abus de notation on notera $\mathcal{O}(max(f_1, f_2))$, l'ordre de grandeur qui est de plus forte croissance parmi les fonctions $f_1(n), f_2(n)$

Appellation des ordres de grandeur courant

- $\mathcal{O}(1)$: complexité constante
- $\mathcal{O}(\log(n))$: complexité logarithmique
- $\mathcal{O}(n)$: complexité linéaire
- $\mathcal{O}(n \log(n))$: complexité quasi-linéaire ou $n \log(n)$
- $\mathcal{O}(n^2)$: complexité quadratique
- $\mathcal{O}(n^3)$: complexité cubique
- $\mathcal{O}(2^n)$: complexité exponentielle

Croissance comparée des fonction usuelles



Rapport Temps/Taille des données

Complexité	$\log_2(n)$	n	n^2	n^3	2^n
Evolution du temps quand la taille des données est multipliée par 10	$t + 3.32$	$10 \times t$	$100 \times t$	$1000 \times t$	t^{10}
Evolution de la taille quand le temps est multiplié par 10	n^{10}	$10 \times n$	$3.16 \times n$	$2.15 \times n$	$n + 3.32$

Calcul de la complexité

- Tout lecture, écriture, affectation, sans appel de fonction, a une complexité en $\mathcal{O}(1)$
- La complexité d'une séquence d'instructions est la complexité de l'instruction de plus forte complexité
  Par abus de notation on notera
  $\mathcal{O}(max(f_1(n), f_2(n), \ldots, f_k(n)))$ l'ordre de grandeur qui est de plus forte croissance parmi les fonctions $f_1(n), f_2(n), \ldots, f_k(n)$
- Pour une structure conditionnelle

```
Si Cond
      alors Action1
      sinon Action2
```

  avec les complexités respectives $\mathcal{O}(C)$, $\mathcal{O}(f1)$, $\mathcal{O}(f2)$
  alors la structure a pour complexité $\mathcal{O}(max(C, max(f_1, f_2)))$

Calcul de la complexité

- Pour une structure répétitive

```
Pour i de A à B faire
     Iter_i
FinPour
```

1. si T(Iter_i) est indépendant de i et de complexité $\mathcal{O}(f)$, alors le temps de la structure répétitive est en $\mathcal{O}((B - A + 1) * f)$
2. si T(Iter_i) est dépendant de i et de complexité $\mathcal{O}(f_{iter}(n, i))$, alors le temps de la structure répétitive est en $\mathcal{O}(\sum_{i=A}^{B} f_{iter}(n, i))$

Calcul de la complexité

- Pour une structure répétitive

```
Tant que Cond faire
     Iter
FinTantque
```

- la difficulté est de déterminer une borne sup pour le nombre d'itérations Nb
  1. si T(Iter) est de complexité $\mathcal{O}(f)$ et l'évaluation de Cond de complexité $\mathcal{O}(C)$, alors le temps de la structure répétitive est en $\mathcal{O}(Nb * (C + f)))$
  2. si T(Iter) est dépendant de i et de complexité $\mathcal{O}(f_{iter}(n, i))$, alors le temps de la structure répétitive est en $\mathcal{O}(Nb * C + \sum f_{iter}(n, i)))$