Cours Info - 10

Complexité algorithmique

D.Malka

MPSI 2015-2016

Sommaire

Sommaire

Temps d'exécution d'un programme

Exemple – Tri d'une liste

Tri d'une liste

On cherche à trier une liste.

On dispose de deux programmes fondés chacun sur deux algorithmes exacts :

- **Le tri par insertion,**
- **Le tri fusion.**

Voir programme de 2ème année pour le détails de ces algorithmes.

Temps d'exécution d'un programme

Exemple – Tri d'une liste

Tri d'une liste

Les deux programmes sont :

- **écrits en langage Python**
- **exécutés sur la machine suivante :**
  - `Processeur Intel Core i5-4210U CPU 1,70GHz x 4`
  - `Mémoire vive 8Go (7,7 Go utilisable)`
  - `OS : Linux Ubuntu 14.04 LTS 64bits`

Temps d'exécution d'un programme

Exemple – Tri d'une liste

Expérience

Avec chacun des deux programmes, trions par ordre croissant une liste aléatoire de N éléments pour N allant de :

- 100 **à...,**
- **...** 10^8**.**

Temps d'exécution d'un programme

Exemple – Tri d'une liste

Expérience

Résultats :

N	10^2	10^3	10^4	10^5	10^6	10^7	10^8
Tri-fusion	$0,3ms$	$0,9ms$	$5,5ms$	$20ms$	$215ms$	$2,3s$	$25,8s$
Tri-sélection	$1,8ms$	$67ms$	$5,7s$	$11min$	$19h$	$66j$	$21ans$

Temps d'exécution d'un programme

Facteurs d'influence

Le temps d'exécution d'un programme dépend :

- **de la machine qui l'exécute,**
- **du langage dans lequel il est écrit,**
- **de l'algorithme sous-jacent.**

Prépondérance de l'algorithme

Pour des entrées de *grandes tailles*, l'algorithme est la cause déterminante du temps d'exécution d'un programme $\rightarrow$ notion de *complexité algorithmique en temps*.

Sommaire

Complexité (en temps) d'un algorithme

Que mesure la complexité ?

Complexité en temps d'un algorithme

La complexité en temps d'un algorithme :

- **est une mesure intrinsèque de sa rapidité d'exécution,**
- **permet de comparer deux algorithmes résolvant le même problème.**

Coût d'un algorithme

Exemple - Somme des n premiers entiers

Calcul de la somme des *n* premier entiers à l'aide d'une boucle :

Entrées: entier naturel n
Sorties: entier naturel somme
somme=0
i=1
tant que *i<=n* **faire**
    somme=somme+i
    i=i+1
retourner *somme*

Algorithm 1: Algorithme $\mathcal{A}_1$

Coût($\mathcal{A}_1$) : $2 \times n$ *additions*

Coût d'un algorithme

Exemple - Somme des n premiers entiers

Calcul de la somme des n premiers entiers à l'aide de la formule mathématique : $\sum_1^n = \frac{n(n+1)}{2}$.

Entrées: entier naturel n
Sorties: entier naturel somme
somme=n*(n+1)/2
retourner *somme*

Algorithm 2: Algorithme $\mathcal{A}_2$

Coût($\mathcal{A}_2$) : 1 addition + 1 multiplication + 1 division

Coût d'un algorithme

Exemple - Somme des n premiers entiers

Comparaison du coût des deux algorithmes en terme d'opérations arithmétiques :

- **Coût($\mathcal{A}_1$) : 2$\times$n opérations arithmétiques.**
- **Coût($\mathcal{A}_2$) : 3 opérations arithmétiques.**

Pour $n > 1$, Coût($\mathcal{A}_2$) < Coût($\mathcal{A}_1$).

L'algorithme $\mathcal{A}_2$ est plus efficace que l'algorithme $\mathcal{A}_1$.

Evaluation de la complexité d'un algorithme

Choix de la mesure élémentaire

- **Pour évaluer la complexité en temps d'un algorithme, il faut choisir une mesure :**
  - nombre de comparaisons,
  - nombre d'affectations,
  - nombre d'opérations arithmétiques,
  - ...
- **La complexité algorithmique est *une évaluation asymptotique* du nombre d'opérations élémentaires.**
- **Elle est exprimée en fonction de la taille n des entrées.**
- **Il n'existe pas un ensemble de règles standardisées pour évaluer la complexité d'un algorithme.**

Evaluation de la complexité d'un algorithme

Quelques règles

La séquence d'instructions : `x_1,x_2... x_n`

$$\text{Coût}(x_1, x_2, ..., x_n) = \sum_k \text{Coût}(x_k)$$

Exemple :

somme=n+1
somme=somme*n
somme=somme/2

Algorithm 3: Algorithme $\mathcal{A}$

$$\text{Coût}(\mathcal{A}) = \text{Coût}(1) + \text{Coût}(2) + \text{Coût}(3)$$

Evaluation de la complexité d'un algorithme

Quelques règles

La boucle simple : `Tant que` $i < n$ `faire` x_i

$$\text{Coût}(\textit{boucle}) = \sum_i (\text{Coût}(\textit{comparaison}) + \text{Coût}(x_i))$$

Exemple :

tant que *i<=n* **faire**
    somme=somme+i
    i=i+1

Algorithm 4: Algorithme $\mathcal{A}$

$$\text{Coût}(\mathcal{A}) = \text{Coût}(i <= n) + \text{Coût}(\textit{somme} + i) + \text{Coût}(i = i + 1) = 3n$$

Evaluation de la complexité d'un algorithme

Quelques règles

Les instructions conditionnelles : `Si` *condition* `faire` x_{vrai} `sinon faire` x_{faux}

$$\text{Coût}(\textit{conditionnelle}) <= \text{Coût}(\textit{test}) + \text{Max}(\text{Coût}(x_{vrai}), \text{Coût}(x_{faux}))$$

Exemple :

si *i%2==0* **alors**
    n=i/2
sinon
    i=i+1
    n=i/2

Algorithm 5: Algorithme $\mathcal{A}$

$$\text{Coût}(\mathcal{A}) <= \text{Coût}(i\%2 == 0) + \text{Max}(\text{Coût}(i = i+1), \text{Coût}(i = i+1, n = i/2))$$
$$\text{Coût}(\mathcal{A}) <= 3$$

Complexité d'un algorithme

Les types de complexité

On note :

- d : une entrée de taille n (par exemple, un tableau d'entiers de longueur n),
- $\mathcal{D}_n$: l'ensemble des entrées d possibles, **de même taille n** (par exemple l'ensemble des listes de taille n=3 d'entiers égaux à 0 ou 1, les possibilités sont : $[0,0,0]$, $[0,0,1]$,$[0,1,0]$, ... $[1,1,0]$ ... $[1,1,1]$)
- $\text{Coût}_{\mathcal{A}}(d)$: le coût de l'algorithme $\mathcal{A}$ pour l'entrée d.

On peut alors calculer trois complexités différentes :

- **la complexité dans le pire des cas,**
- **la complexité dans le meilleur des cas,**
- **la complexité moyenne.**

Les pire cas, meilleur cas et cas moyen s'entendent à n fixé et n grand !

Complexité d'un algorithme

Complexité au pire

Complexité dans le pire des cas

$$\text{Max}_{\mathcal{A}} = \text{Max}(\text{Coût}_{\mathcal{A}}(d)|d \in \mathcal{D}_n)$$

C'est cette complexité qu'on utilise le plus souvent pour comparer deux algorithmes.

Complexité d'un algorithme

Complexité au mieux

Complexité dans le meilleur des cas

$$\text{Min}_{\mathcal{A}} = \text{Min}(\text{Coût}_{\mathcal{A}}(d) | d \in \mathcal{D}_n)$$

Cette complexité est rarement intéressante.

Complexité d'un algorithme

Complexité moyenne

Complexité moyenne

$$\text{Moy}_{\mathcal{A}} = \sum_{d \in \mathcal{D}_n} P(d)\text{Coût}_{\mathcal{A}}(d)$$

avec $P(d)$ la probabilité d'occurrence de l'entrée d.

La plus satisfaisante mais très difficile à calculer par méconnaissance de la distribution de probabilités suivie par le entrées.

Complexité d'un algorithme

Complexité asymptotique

Les algorithmes s'exécutent souvent sur des entrées de grande taille n.

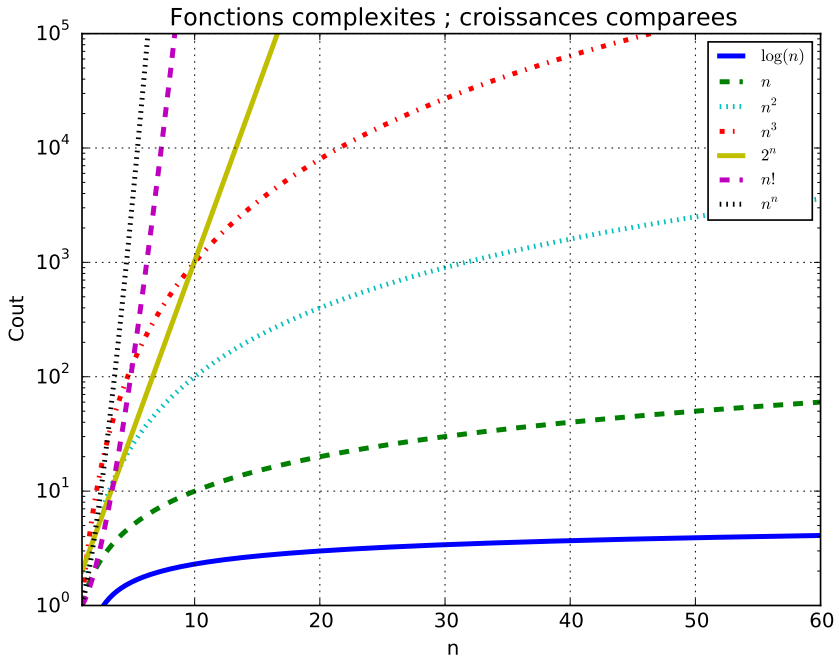
Conséquences :

- **le coût exact d'un algorithme n'est pas intéressant,**
- **une approximation asymptotique de la complexité est suffisante.**
- **échelle de fonction de complexité :**

$$\log(n) < n < n\log(n) < n^2 < n^3 < \ldots < k^n < n! < n^n$$

Complexité d'un algorithme

Complexité asymptotique



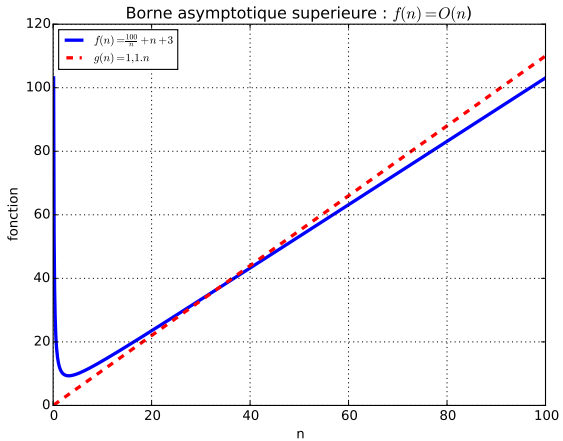
Complexité d'un algorithme

Complexité asymptotique : notation O

Notation O

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

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



On majore la complexité algorithmique.

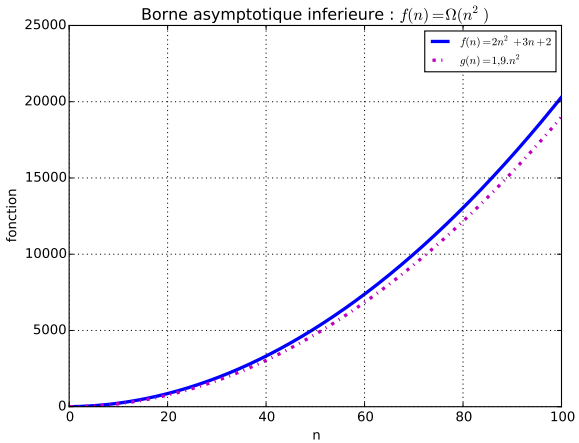
Complexité d'un algorithme

Complexité asymptotique : notation Ω

Notation Ω

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

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



On minore la complexité algorithmique.

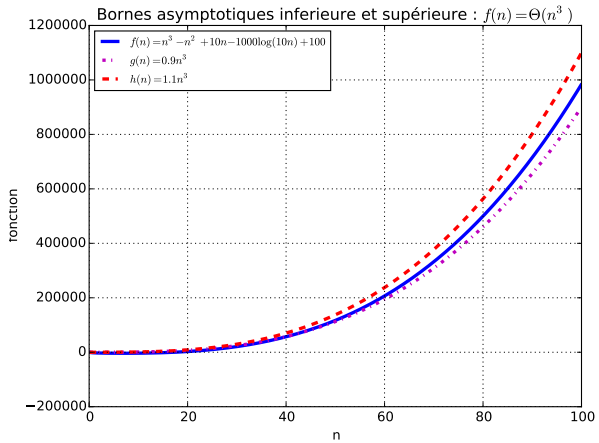
Complexité d'un algorithme

Complexité asymptotique : notation Θ

Notation Θ

Soient f et $g : \mathbb{N} \to \mathbb{R}^+$: $f = \Theta(g)$ ssi $\exists c_1, c_2 \in \mathbb{R}^+$; $\exists n_0 \in \mathbb{N}$ tels que :

$$\forall n > n_0, \quad c_1 \times g(n) \leq f(n) \leq c_2 \times g(n)$$



On borne la complexité algorithmique.

Complexité d'un algorithme

Exemple

Notations asymptotiques

- $2n = O(n^2)$
- $2n = O(n)$
- $2n = \Theta(n)$
- $2n \neq \Theta(n^2)$

Complexité d'un algorithme

Exemple - Factorielle de n

Entrées: entier naturel n
Sorties: entier naturel fact
c=n
fact=1
tant que *c>0* **faire**
    fact=fact*c
    c=c-1
retourner *fact*

Algorithm 6: Calcul de n !

Complexité en temps

Mesure : nombre d'affectations.

$\sim 2n$ affectations $\rightarrow$ La complexité (au pire) de l'algorithme est $O(n)$.

Complexité d'un algorithme

Vocabulaire

Un algorithme . . .

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

Sommaire

Complexité en espace d'un algorithme

En bref

Complexité en espace

Mesure l'espace mémoire occupé maximal durant l'exécution de l'algorithme. On définit l'unité d'espace mémoire comme la taille d'une structure de données particulière. On mesure la complexité en espace par le nombre n d'unités de mémoire occupées lors de l'exécution de l'algorithme.

Brièvement...

- Soit un tableau d'entiers de longueur n. Unité d'espace mémoire : taille d'un entier.
  L'espace mémoire occupé par ce tableau est $O(n)$.
- Soit un tableau d'entiers de longueur $5n$. Unité d'espace mémoire : taille d'un entier.
  L'espace mémoire occupé par ce tableau est $O(n)$.
- Soit une matrice $m \times n$ à coefficients réels . Unité d'espace mémoire : taille d'un flottant.
  L'espace mémoire occupé par cette matrice est $O(n \times m)$.