

Coût et Complexité

Le Master Theorem

Jean-Marc.Vincent@univ-grenoble-alpes.fr

Université de Grenoble-Alpes, UFR IM²AG
L3 INFO



RÉCURRENCE ET COMPLEXITÉ

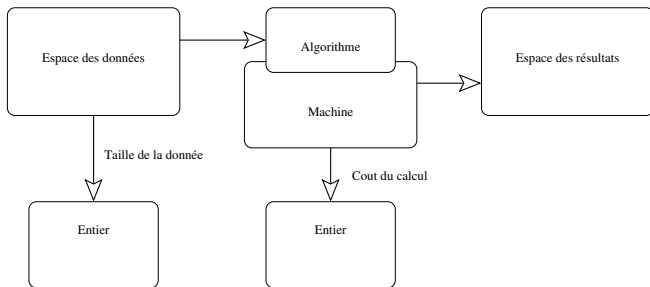
1 **LE PROBLÈME : évaluer le coût d'un algorithme**

2 ÉQUATIONS LINÉAIRES

3 ÉQUATION DE PARTITION

4 VADE MECUM

ÉVALUATION DU COÛT



Coût associé à un algorithme en fonction de la taille n des données en entrée

- ▶ **Choix des opérateurs** : donné par le modèle de machine
- ▶ **Ordre de grandeur** : échelles logarithmique $\log n$, polynomiale n^k , exponentielle α^n
 $\mathcal{O}(\cdot)$ (majore asymptotiquement) ou $\Omega(\cdot)$ (minore) ou $\Theta(\cdot)$ (du même ordre).
- ▶ **Expression de l'algorithme** : forme itérative ou récursive

Résoudre des équations de récurrence.

RÉCURRENCE ET COMPLEXITÉ

1 LE PROBLÈME : évaluer le coût d'un algorithme

2 ÉQUATIONS LINÉAIRES

3 ÉQUATION DE PARTITION

4 VADE MECUM

ÉQUATIONS SIMPLES

$$\begin{cases} T(0) = 0; \\ T(n) = T(n-1) + c(n), \text{ pour } n \geq 1. \end{cases}$$

où c est une fonction (en général positive).

$$T(n) = T(0) + c(0) + c(1) + \dots + c(n) = \sum_{i=0}^n c(i).$$

Sommes classiques

$$\sum_{i=1}^n 1 = n = \Theta(n);$$

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

$$\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6} = \Theta(n^3);$$

$$\sum_{i=1}^n i^3 = \frac{n^2(n+1)^2}{4} = \Theta(n^4);$$

et de manière plus générale

$$\sum_{i=1}^n i^k = \Theta(n^{k+1});$$

Exemples Calcul du coût de l'algorithme de tri par insertion, par sélection,...

ÉQUATIONS DE RÉCURRENCE MULTIPLE

$$\begin{cases} T(0) = t_0, T(1) = t_1 \cdots T(k-1) = t_{k-1}; \\ T(n) = a_1 T(n-1) + a_2 T(n-2) + \cdots + a_k T(n-k), \text{ pour } n \geq k. \end{cases}$$

Ces équations de récurrence linéaires se développent à partir d'une base de solutions de la forme :

$$T(n) = \sum_{i=1}^p \left(\sum_{j=0}^{m_i-1} c_{i,j} n^j \right) \lambda_i^n; \text{ avec } \{\lambda_1, \lambda_2, \dots, \lambda_p\} \text{ les } p \text{ racines du polynôme}$$

$$P(x) = x^k - a_1 x^{k-1} - a_2 x^{k-2} - \cdots - a_0; \text{ de multiplicité respectives } \{m_1, \dots, m_p\}.$$

Les constantes $c_{i,j}$ sont fixées par les conditions initiales.

$$T(n) = \mathcal{O}(n^{m-1} |\lambda_{\max}|^n) \text{ avec } \lambda_{\max} \text{ la racine de module maximal et de multiplicité } m.$$

C'est à dire que la fonction de coût croit de manière très rapide, au moins exponentiellement.

Exercice : Calculer la valeur de $T(n)$ pour l'équation de récurrence

$$T(0) = T(1) = 1;$$

$$T(n) = T(n-1) + T(n-2); \text{ pour } n \geq 2.$$

AUTRES EXPRESSIONS UTILES

Croissance logarithmique

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{n} = \log n + \gamma + o\left(\frac{1}{n}\right)$$

$n^{\text{ième}}$ nombre harmonique, $\gamma \simeq 0,5772156649$ constante d'Euler-Mascheroni

$$H_n = \Theta(\log n)$$

Croissance exponentielle

$$1 + a + a^2 + \dots + a^n = \frac{a^{n+1} - 1}{a - 1} = \Theta(a^n) \text{ avec } a \neq 1$$

RÉCURRENCE ET COMPLEXITÉ

- 1 LE PROBLÈME : évaluer le coût d'un algorithme
- 2 ÉQUATIONS LINÉAIRES
- 3 ÉQUATION DE PARTITION
- 4 VADE MECUM

ÉQUATIONS DE PARTITION

$$\begin{cases} T(1) = 1; \\ T(n) = aT(\frac{n}{b}) + c(n), \text{ pour } n \geq 1, \end{cases}$$

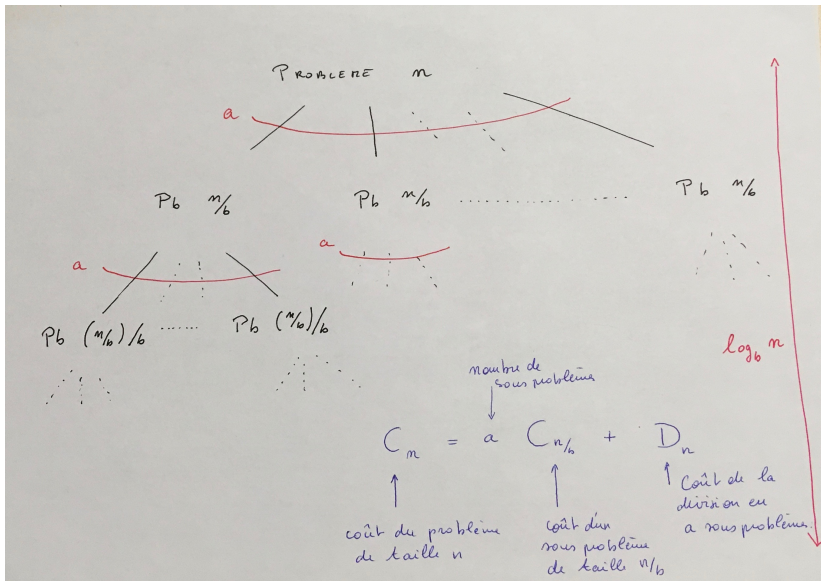
avec $a \geq 1$, $b > 1$ et $c(n)$ une fonction positive de n .

- ▶ Attention cette formulation contient déjà une approximation : la fraction $\frac{n}{b}$ n'a pas de raison d'être entière.
- ▶ Expression récurrente avec un facteur de division constant
- ▶ souvent on a un \leq au lieu du $=$

Exemples

- ▶ Recherche dichotomique, recherche dans un ABR
- ▶ Parcours d'un arbre binaire
- ▶ tri rapide, tri partition fusion
- ▶ fusion en place
- ▶ ...

UN DESSIN



SOLUTION DE L'ÉQUATION DE PARTITION

Pour simplifier on suppose que $n = b^k$ et on note $t_k = T(b^k)$ (changement de variable).

$$\begin{array}{rclcl} t_k & = & a & t_{k-1} & + & c(b^k) \\ t_{k-1} & = & a & t_{k-2} & + & c(b^{k-1}) \\ t_{k-2} & = & a & t_{k-3} & + & c(b^{k-2}) \\ \vdots & = & \vdots & \vdots & & \\ t_1 & = & a & t_0 & + & c(b^1) \end{array}$$

SOLUTION DE L'ÉQUATION DE PARTITION

Pour simplifier on suppose que $n = b^k$ et on note $t_k = T(b^k)$ (changement de variable).

$$\begin{array}{rclclcl}
 & t_k & = & a & t_{k-1} & + & c(b^k) \\
 a & t_{k-1} & = & a^2 & t_{k-2} & + & a c(b^{k-1}) \\
 a^2 & t_{k-2} & = & a^3 & t_{k-3} & + & a^2 c(b^{k-2}) \\
 & \vdots & = & \vdots & \vdots & + & \vdots \\
 a^{k-1} & t_1 & = & a^k & t_0 & + & a^{k-1} c(b^1)
 \end{array}$$

$$t_k = a^k t_0 + \sum_{i=0}^{k-1} a^i c(b^{k-i})$$

En utilisant le fait que $t_0 = 1$ et que $a^k = n^{\log_b a}$, le résultat se met sous la forme

$$T(n) = n^{\log_b a} + \sum_{i=0}^{k-1} a^i c(b^{k-i}).$$

En fonction du problème étudié, soit le premier terme l'emporte "le coût de partition" n'est pas prépondérant, soit c'est le deuxième terme qui l'emporte et toute la complexité du calcul est dans le partitionnement.

SOLUTION DE L'ÉQUATION DE PARTITION (DÉTAILS)

$$T(n) = \Theta(n^{\log_b a}) \quad \text{si } c(n) = O(n^{\log_b a - \varepsilon}) \text{ pour un } \varepsilon > 0;$$

$$T(n) = \Theta(n^{\log_b a} \log n) \quad \text{si } c(n) = \Theta(n^{\log_b a})$$

$$T(n) = \Theta(c(n)) \quad \text{si } c(n) = \Omega(n^{\log_b a + \varepsilon}) \\ \text{pour un } \varepsilon > 0 \text{ et si } c(n) \geq adc(n/b) \text{ pour } d \geq 1.$$

En pratique on recalcule pour bien comprendre le poids de la partition et le poids du calcul de fusion.

RÉCURRENCE ET COMPLEXITÉ

1 LE PROBLÈME : évaluer le coût d'un algorithme

2 ÉQUATIONS LINÉAIRES

3 ÉQUATION DE PARTITION

4 VADE MECUM

VADE MECUM

Calcul de coût et complexité

- 1 Spécifier les **opérateurs** de base de votre modèle de machine (coût constant) ;
- 2 Décrire l'espace des données et son partitionnement (définir la **taille** d'une donnée) ;
- 3 Écrire l'**expression du coût** à l'aide des règles de composition des coûts élémentaires des opérateurs de base ;
- 4 Pour une donnée de taille n calculer le coût **maximal** (coût au pire) et le coût minimal (coût au mieux) de l'algorithme ;
- 5 Étudier l'ensemble des données conduisant au pire ou au meilleur coût (**exemples**) ;
- 6 Donner l'**ordre de grandeur** du coût (avec les fonctions \mathcal{O} ou Ω) pour le coût au pire d'une donnée de taille n sur une échelle permettant de comparer les algorithmes (classiquement on utilise l'échelle déduite de l'échelle polynomiale, des logarithmes (en base 2) et des exponentielles (puissances de 2) ;