Calcul de coût d'algorithme

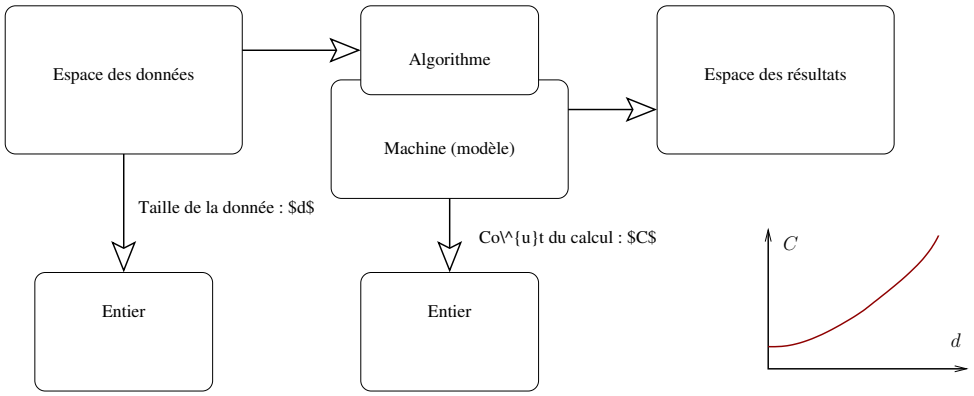
Concepts : Analyse de coût
Méthodes : Décomposition du coût, ordres de grandeur

Présentation

Le concept de **coût** associé à un algorithme $\mathcal{A}$ formalise la notion intuitive d'**efficacité** d'un programme, c'est à dire sa capacité à fournir le résultat escompté dans un temps minimal, on parle également de performances du programme. Du point de vue pratique, l'analyse du coût permet de déterminer quelles seront les ressources en temps de calcul, en mémoire et éventuellement en entrées/sorties nécessaires pour exécuter le calcul associé à l'algorithme.

Pour définir le coût, on se donne un **modèle de machine** avec une mémoire que l'on suppose en général infinie et munie d'opérations dont le temps d'exécution (coût élémentaire) est constant (ou majoré par une constante). L'exécution de l'algorithme sur une donnée d est modélisé par une séquence finie d'opérations de la machine ; le coût de l'exécution $C(\mathcal{A}, d)$ sur la donnée d est donc le nombre d'exécutions de chaque opération au cours de l'exécution. Éventuellement on restreindra le nombre d'opérations étudiées en choisissant celles dont l'interprétation sera la plus intéressante (opération la plus coûteuse).

Comme l'algorithme va être exécuté sur un grand ensemble de données, il est nécessaire de spécifier l'espace des données $\mathcal{D}$ et d'en fournir une description algébrique. Pour analyser le coût d'un algorithme, on cherche alors à donner des facteurs expliquant le coût en fonction de paramètres de la donnée. En particulier, on s'intéresse au lien entre la **taille** de la donnée et le coût de l'algorithme.



Définition 1 (Coût au pire)
Le coût au pire de l'algorithme $\mathcal{A}$ est défini par la fonction de n :

$$Coût_{\mathcal{A}}(n) = \max_{d \in \mathcal{D}_n} C(\mathcal{A}, d).$$

On peut également définir le coût au mieux en fonction de n en prenant le minimum des coûts.

Comportement asymptotique

Ce qui est intéressant dans l'analyse du coût est le comportement asymptotique en fonction de n (notion de passage à l'échelle). Pour cela on compare la fonction de coût à des fonctions de référence appelées **fonctions d'échelle**. Typiquement les fonctions puissance x^n, exponentielles 2^n ou logarithmiques $\log^k(n)$ sont des fonctions d'échelle. Pour exprimer la comparaison asymptotique on utilise la notation de Landau :

Coût d'algorithme

Définition 2 (Borne asymptotique approchée : fonction Θ)
Pour une fonction donnée g on note $\Theta(g)$ l'ensemble de fonctions :

$$\Theta(g) = \{f \text{ telles que } \exists c_1, c_2 \geqslant 0, \exists n_0 \geqslant 0, \forall n \geqslant n_0, c_1.g(n) \leqslant f(n) \leqslant c_2.g(n)\}\,;$$

on écrit $f = \Theta(g)$ pour $f \in \Theta(g)$ et on dit que g est une borne asymptotique approchée pour f.

Définition 3 (Borne supérieure asymptotique : fonction $\mathcal{O}$)
Pour une fonction donnée g on note $\mathcal{O}(g)$ l'ensemble de fonctions :

$$\mathcal{O}(g) = \{f \text{ telles que } \exists c \geqslant 0, \exists n_0 \geqslant 0, \forall n \geqslant n_0, f(n) \leqslant c.g(n)\}\,;$$

on écrit $f = \mathcal{O}(g)$ pour $f \in \mathcal{O}(g)$ et on dit que g est une borne supérieure asymptotique pour f.

Définition 4 (Borne inférieure asymptotique : fonction Ω)
Pour une fonction donnée g on note $\Omega(g)$ l'ensemble de fonctions :

$$\mathcal{O}mega(g) = \{f \text{ telles que } \exists c \geqslant 0, \exists n_0 \geqslant 0, \forall n \geqslant n_0, c.g(n) \leqslant f(n)\}\,;$$

on écrit $f = \Omega(g)$ pour $f \in \Omega(g)$ et on dit que g est une borne inférieure asymptotique pour f.

Règles de calcul du coût

Pour calculer le coût d'un algorithme on utilise les règles de composition suivantes :

Le coût d'une **itération** est égal à la **somme** des coûts de chaque terme de l'itération ;

Le coût d'une **instruction conditionnelle** est **majoré** par le **maximum** des coûts de chacune des branches de l'instruction conditionnelle ;

Le coût d'un **appel de procédure** est égal à la **somme** du coût de l'appel (évaluation des paramètres) et du coût du corps de la procédure.

Lorsque le programme est récursif, on exprime le coût de l'algorithme en fonction du coût des appels récursifs. On obtient ainsi une équation de **récurrence** sur la fonction de coût.

Coût en moyenne

Lorsque, pour une taille de donnée fixée, le coût peut prendre plusieurs valeurs entre les bornes coût maximum et coût minimum. Il est interessant de savoir si, en général, pour une donnée arbitraire on est plutôt près du coût au mieux ou au pire coût. Pour formaliser le raisonnement, on munit l'espace des données de taille n d'une probabilité (usuellement la loi uniforme) et on calcule la distribution du coût, c'est à dire on calcule la probabilité que le coût de l'algorithme prenne la valeur $1, 2, \cdots, n, \cdots$. Comme il est souvent très difficile de calculer la probabilité de chaque valeur de coût on se contente de sa valeur moyenne.

Définition 5 (Coût moyen (cas uniforme))
Le coût moyen de l'algorithme $\mathcal{A}$ est défini par la fonction de n :

$$Coût - moyen_{\mathcal{A}}(n)) = \frac{1}{Card(\mathcal{D}_n)} \sum_{d \in \mathcal{D}_n} C(\mathcal{A}, d).$$

Coût d'algorithmes récursifs

Dans de nombreuses situations il est possible d'exprimer le coût (au mieux, au pire, en moyenne) d'un algorithme sur des données de taille n comme une fonction du coût de l'exécution du même algorithme sur des des données de tailles plus petites que n.

La fonction de coût vérifie ainsi une équation de **récurrence**. On peut classer les équations de récurrence en fonction de leur forme (linéaire, de partition, ...) et donner les asymptotiques. On notera $T(n)$ le coût de l'algorithme pour une donnée de taille n.

Coût d'algorithme

Équations linéaires simples

$$T(0) = 0;$$

$$T(n) = T(n-1) + f(n), \text{ pour } n \geqslant 1,$$

où f est une fonction (en général positive). En itérant la relation à partir de $n = 0$ on obtient facilement :

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

On rappelle que

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

$$\sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4} = \Theta(n^4) \text{ et de manière plus générale } \sum_{i=1}^{n} i^k = \mathcal{O}(n^{k+1}).$$

Exercice : faire la preuve en comparant $\sum_{i=1}^{n} i^k$ et $\int_1^n x^k$

Exemples

Étudier le tri par sélection, le tri par insertion (cf TD) avec cette méthode.

Équations linéaires multiples

$$T(0) = t_0, T(1) = t_1 \cdots T(k-1) = t_{k-1};$$

$$T(n) = a_1 T(n-1) + a_2 T(n-2) + \cdots a_k T(n-k), \text{ pour } n \geqslant k.$$

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

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

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

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

De manière générale si λ_{max} est la racine de module maximal et de multiplicité m alors

$$T(n) = \mathcal{O}(n^{m-1} |\lambda_{max}|^n).$$

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

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

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

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

On appelle la suite $T(n)$, la suite de Fibonacci (Léonardo de Pise mathématicien Italien du XII-XIII$^{\text{ième}}$ siècle).

Coût d'algorithme

Équations de partition

On suppose que la fonction de coût suit l'équation suivante

$$T(1) = 1;$$

$$T(n) = aT(\frac{n}{b}) + c(n), \text{ pour } n \geqslant 1,$$

avec $a \geqslant 1$, $b > 1$ 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)

Ce type d'équation apparaît naturellement lorsque la résolution d'un problème de taille n se fait en résolvant a problèmes de taille $\frac{n}{b}$. Pour simplifier on suppose que $n = b^k$ et on note $t_k = T(b^k)$. On a facilement l'équation de récurrence sur les t_k en remarquant que

$$\begin{array}{rcrcr}
t_k & = & at_{k-1} & + & c(b^k) \\
at_{k-1} & = & a^2t_{k-2} & + & ac(b^{k-1}) \\
a^2t_{k-2} & = & a^3t_{k-3} & + & a^2c(b^{k-2}) \\
\vdots & = & \vdots & \vdots & \vdots \\
a^{k-1}t_1 & = & a^kt_0 & + & a^{k-1}c(b^0) \\
\hline
t_k & = & a^kt_0 & + & \sum_{i=0}^{k-1} a^ic(b^{k-i})
\end{array}$$

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

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

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.

Calcul de coût

1. Spécifier les opérateurs de base de votre modèle de machine (coût constant) ;
2. Décrire l'espace des données et sa segmentation (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 $\mathcal{O}mega$) 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) ;

Lectures suggérées

Algorithmique, Cormen, Leiserson, Rivest et Stein Dunod 2011 (pour la version française) chapitres 2 et 3 **faire les exercices du chapitre 3**

Coût d'algorithme

Pour les aspects mathématiques discrètes et application au calcul de coût on consultera le livre : Concrete Mathematics, Second Edition by Ronald L. Graham, Donald E. Knuth, and Oren Patashnik ou sa version française Mathématiques concrètes.

Références

[1] Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein. *Algorithmique - 3ème édition - Cours avec 957 exercices et 158 problèmes*. Dunod, 3e édition edition, 6 2010.

[2] Ronald Graham, Donald E. Knuth, and Oren Patashnik. *Mathématiques concrètes*. International Thomson publishing France, 2e éd edition, 1998.

[3] Robert Sedgewick and Kevin Wayne. *Algorithms (4th Edition)*. Addison-Wesley Professional, 2011.