Cours: Algorithmes et Structures des Données

Chapitre 10: La complexité des algorithmes

Réalisé par:
Dr. Sakka Rouis Taoufik

1

I. Introduction

La classe des algorithmes qui satisfont une spécification donnée, si elle n'est pas vide, contient en général une infinité d'éléments.

En effet, un problème ayant une spécification donnée peut avoir peut avoir plusieurs algorithmes : A_1, A_2, A_3, …, A_n.

Par exemple, on cite : problème de tri et le problème de recherche dans une table.

Questions posées :

- Sur quelle base peut-on comparer ces algorithmes deux à deux ?
- Quels critères est-il pertinent d'utiliser dans la pratique ?

2

Chapitre 10 : La Complexité des algorithmes

I. Introduction

Typiquement, on distingue des **critères statiques** (c'est-à-dire indépendants de l'exécution de l'algorithme) et des critères dynamiques. En ce qui concerne les critères statiques, nous pouvons citer :

-le temps de développement d'une solution (d'un algorithme) à partir d'une spécification : (la lisibilité du programme ; sa maintenabilité ; la qualité de son ergonomie ; sa sécurité d'utilisation ; sa longueur ; etc)

➔ Les quatre premiers points ne doivent être négligés, mais ils sont **trop subjectifs**. Ils dépendent autant du programmeur et /ou du lecteur que de l'algorithme lui-même pour permettre une comparaison **impartiale.**

3

Chapitre 10 : La Complexité des algorithmes

I. Introduction

L'objectif de sécurité est, du point de vue des outils qui permettent d'évaluer l'algorithme, très lié à **la notion de correction** du programme par rapport à sa spécification.

Illustration : Démontrer qu'une propriété est satisfaite pour le programme comme par exemple le programme ne modifie que des fichiers locaux.

Le critère longueur est à la base de la notion de complexité de **Kolmogorov**.

En ce qui concerne les **critères dynamiques**, ils portent soit sur la place mémoire (la complexité spatiale) utilisée pour une exécution soit sur le temps d'exécution (la complexité temporelle).

4

Chapitre 10 : La Complexité des algorithmes

I. Introduction

La complexité algorithmique permet de mesurer les performances d'un algorithme et de le comparer avec d'autres algorithmes réalisant les même fonctionnalités.

La complexité d'un algorithmique consiste en l'étude de la quantité de ressources (de temps ou espace) nécessaire à l'exécution de cet algorithme

La complexité algorithmique est un concept fondamental pour tout informaticien, elle permet de déterminer si un algorithme a et meilleur qu'un algorithme b et s'il est optimal ou s'il ne doit pas être utilisé. . .

5

Chapitre 10 : La Complexité des algorithmes

II. Complexité spatiale versus complexité temporelle

Plusieurs arguments militent en faveur du critère temporel plutôt que du critère spatial :

-pour occuper de la place il faut consommer du temps : l'inverse n'est pas vrai

-la place occupée à l'issue de l'exécution, en supposant que l'exécution termine, et est à nouveau disponible. La mémoire est une ressource recyclable.

-la complexité temporelle établit une ligne de partage consensuelle entre les solutions praticables et les autres. Penser aux algorithmes de complexité exponentielle et supérieurs.

6

III. Notion de croissance asymptotique

Comparaison de fonctions

Considérons le problème du tri d'un tableau sans doublon. Ce problème est spécifié informellement de la manière suivante :

«la fonction tri(t) délivre un tableau représentant la permutation triée par ordre croissant des valeurs de t.» Supposons que nous cherchions à comparer la complexité de deux fonctions déterministes $tri_1(t)$ et $tri_2(t)$ qui satisfont cette spécification.

Supposons par ailleurs que la complexité de la fonction tri_1 (respectivement tri_2) soit donnée par la fonction $f_1(n)$ (respectivement $f_2(n)$), n étant la taille de t.

Chapitre 10 : La Complexité des algorithmes

III. Notion de croissance asymptotique

Nous intéressons aux fonctions de profil $N \rightarrow N$ ou $N \rightarrow R+$.

Ce choix est justifié par le fait que la complexité d'un algorithme est une fonction à valeurs entières ou réelles non négatives définie sur une grandeur scalaire discrète, typiquement **la taille de la donnée.**

Le caractère total de ces fonctions tient au fait qu'en général toutes **les tailles sont significatives**.

III. Notion de croissance asymptotique

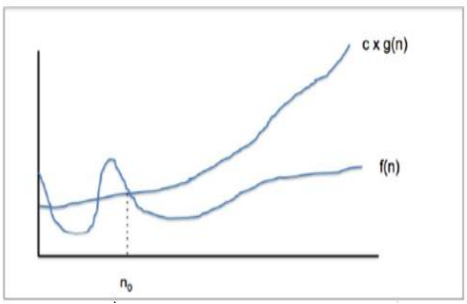
Définition 1 (Notation O omicron)

Soit g(n) une fonction de $N \rightarrow R^+$ On définit:

$$O(g(n)) = \{f(n) \mid \exists\, c > 0, \exists\, n_0 \geq 0, \forall n \geq n_0, f(n) \leq c\, g(n)\}$$

➔ Cela signifie qu'a partir certain rang, la fonction f est majorée par une constante fois la fonction g. Il s'agit donc d'une situation de **domination** de la fonction f par la fonction g.

Exemples:
$n^2 \in O(2^n+2n)$
$20n^2 \in O(3n^3+2n+1)$
$20n^2 \in O(20n^2+50n+5)$
$20n^2 \in O(19n^2)$
$n^2 \notin O(200n)$



III. Notion de croissance asymptotique

Exercices sur la notation O omicron

Exercice 1: Prouver que :

1) Si f(n)=4 alors f(n)= **O**(1),
➔prendre par exemple c=5 et $n_0 = 1$

2) Si f(n)=3n+2 alors f(n)= **O**(n),
➔prendre par exemple c=4 et n0 = 2

3) Si $f(n)=2n^2+3$ alors f(n)= **O**(n^2),
➔ prendre par exemple c=3 et n0 = 2

III. Notion de croissance asymptotique

Exercices sur la notation O omicron

Exercice 2:

Pour chacune des fonctions T_i (n) suivantes déterminer sa complexité asymptotique dans la notation O.

Exemple

0) $T_0(n) = 3n$ $\in O(n)$

1) $T_1(n) = 6\,n^3 + 10n^2 + 5n + 2$

2) $T_2(n) = 3 \log n + 4$

3) $T_3(n) = 2^n + 6n^2 + 7n$

4) $T_4(n) = 7k + 2$

5) $T_5(n) = 4 \log n + n$

6) $T_6(n) = 6\,n^3 + 10n^2 + 5n + 2$

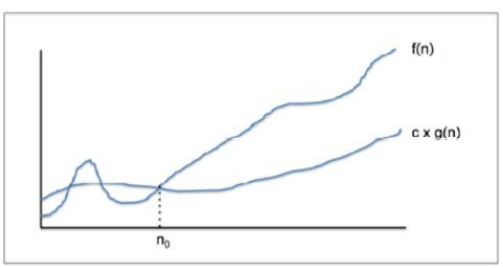
III. Notion de croissance asymptotique

Définition 2 (Notation Ω oméga)

Soit g(n) une fonction de $N \to R^+$ On définit

$$\Omega(g(n)) = \{f(n) \mid \exists\, c > 0, \exists\, n_0 \geq 0, \forall n \geq n_0,\quad c\,g(n) \leq f(n)\}$$

➔ Cela signifie qu'a partir certain rang, la fonction f est minorée par une constante fois la fonction g. Il s'agit donc d'une situation de **domination** de la fonction g par la fonction f.



III. Notion de croissance asymptotique

Exercice sur la notation Ω oméga

Prouver que :

1) Si f(n)=4 alors f(n)= **Ω**(1),

2) Si f(n)=4n+2 alors f(n)= **Ω**(n)

3) Si f(n)=4n²+1 alors f(n)= **Ω**(n)

III. Notion de croissance asymptotique

Définition 3 (Notation θ théta)

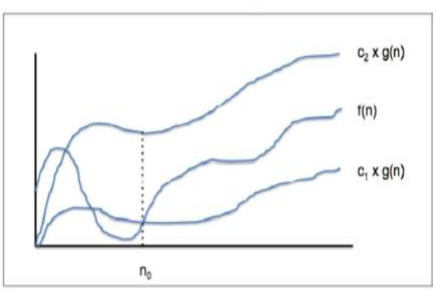
Soit g(n) une fonction de $N \rightarrow R^+$ On définit:

$$\Theta(g(n)) = \{f(n) \mid \exists c_1 > 0, \exists c_2 > 0, \exists n_0 \geq 0, \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

➔ g est équivalent (ou comparable) asymptotiquement à f

- A partir d'un certain rang n0
  la fonction f(n) peut être bornée inférieurement par c1 ***g(n);**
  la fonction f(n) peut être bornée supérieurement par c2* ***g(n);***

Exemples
$n^2 \notin \theta(2^n+2n)$
$20\ n^2 \notin \theta(3n^3+2n+1)$
$20\ n^2 \in \theta\ (20n^2+50n+5)$
$20\ n^2 \in \theta(19n^2)$
$n^2 \notin \theta(200n)$



III. Notion de croissance asymptotique

Exercice sur la notation θ théta: Prouver que :

1) Si f(n)=4 alors f(n) ∈ θ(1),

2) Si f(n)=4n+2 alors f(n) ∈ θ(n),

3) Si f(n)=4n²+1 alors f(n) ∈ θ(n²)

IV. Règles de calcul de la complexité en O

Pour calculer la complexité grand **O** d'un algorithme il faut compter le nombre **d'opérations de base** qui effectue comme :

- Opération arithmétique ou logique
- Opération d'affectation
- Vérification d'une condition
- Opération d'entrée/Sortie

➔ La complexité de chaque opération de base est **constante** ou **O(1)**