Complexité

Objectifs

- Définir ce qu'est la complexité d'un algorithme.
- Comprendre les notions de complexité minimale, maximale et moyenne.
- Connaître et savoir utiliser les notations asymptotiques O, Ω et Θ.

Introduction

Définition

La complexité d'un programme notée $C(n)$ ou $T(n)$ est une fonction qui dépend de la taille n des données en entrée du programme et qui informe sur l'évolution du temps d'exécution de ce programme.

Le temps mis par un programme est donc dépendant du nombre d'opérations à faire.

Exemples

- **Recherche d'un élément parmi n :**
  Si $n = 5$, il faut au pire 5 tests et si $n = 10$, il en faut 10.
- **Problème du ramassage de plots (en prenant le plus court chemin) :**
  Si $n = 5$ il faut tester $5! = 120$ cas, et si $n = 10$, il faut en tester 3628800.

Complexité min, max et moy

Même avec une taille fixée, le nombre d'opérations n'est pas constant. On considère souvent trois cas : le pire, le meilleur et le cas moyen.

Exemple : Recherche d'un élément parmi n

- **Meilleur cas :** on le trouve du premier coup : 1 essai.
- **Pire cas :** on le trouve en dernier ou il n'y est pas : n essais.
- **Cas moyen :** on fait la moyenne de tous les cas possibles :
  Il y a n cas possibles de 1 essai à n essais, soit :

$$\frac{\sum_{i=1}^{n} i}{n} = \frac{\frac{n(n+1)}{2}}{n} = \frac{n+1}{2} \text{ essais}$$

Notation

1. La complexité dans le **meilleur cas** est notée : $\check{C}(n)$ ou $\check{T}(n)$
2. La complexité dans le **pire cas** : $\hat{C}(n)$ ou $\hat{T}(n)$
3. La complexité dans le **cas moyen** : $\bar{C}(n)$ ou $\bar{T}(n)$

Propriété

Pour tout n entier :

$$\check{T}(n) \leq \bar{T}(n) \leq \hat{T}(n)$$

Notations asymptotiques

Les fonctions de complexité sont souvent trop complexes et ce qui nous intéresse est de savoir si l'évolution de la complexité sera rapide ou non. C'est pourquoi nous allons essayer de borner notre fonction.

Borne supérieure

Définition : Grand O

Soient f et g deux fonctions de $\mathbb{N}$ dans $\mathbb{R}$.
S'il existe une constante positive c telle qu'il existe $n_0 \in \mathbb{N}$ et pour tout $n \geq n_0$ on ait $0 \leq f(n) \leq c \times g(n)$, alors on dit que g domine f.
On note $f = O(g)$ ou $f(n) = O(g(n))$ et cela se dit f est en *grand O* de g.

Interprétation graphique : La fonction f reste en dessous d'un multiple constant de g à partir d'un certain rang n_0.

Exemples

- $3n^2 - n + 6 = O(n^2)$ (avec $c = 3$ et $n_0 = 6$)
- $3n^2 - n + 6 = O(n^3)$ (avec $c = 1$ et $n_0 = 6$)
- Par contre $3n^2 - n + 6 \neq O(n)$ (car pour tout c entier positif, $3n^2 - n + 6 > cn$ pour $n > c + 1$)

Propriété

Soient f et g deux fonctions de $\mathbb{N}$ dans $\mathbb{R}$.
Si $\lim\limits_{n \to +\infty} \dfrac{f(n)}{g(n)} = 0$ alors $f = O(g)$.
(Attention, la réciproque est fausse).

Borne inférieure

Définition : Grand Omega

Soient f et g deux fonctions de $\mathbb{N}$ dans $\mathbb{R}$.
S'il existe une constante positive c telle qu'il existe $n_0 \in \mathbb{N}$ et pour tout $n \geq n_0$ on ait $0 \leq c \times g(n) \leq f(n)$.
On note $f = \Omega(g)$ ou $f(n) = \Omega(g(n))$, on dit que f est en *grand omega* de g.

Exemples

- $3n^2 - n + 6 = \Omega(n^2)$ (avec $c = 1$ et $n_0 = 0$)
- $3n^2 - n + 6 = \Omega(n)$ (avec $c = 1$ et $n_0 = 0$)
- $3n^2 - n + 6 \neq \Omega(n^3)$

Fonction bornée

Définition : Grand Theta

Soient f et g deux fonctions de $\mathbb{N}$ dans $\mathbb{R}$.
Lorsque $f = O(g)$ et $f = \Omega(g)$, alors on note $f = \Theta(g)$ ou $f(n) = \Theta(g(n))$, on dit que f est en *grand thêta* de g.
f est alors asymptotiquement équivalente à g à une constante près.

Exemple

$3n^2 - n + 6 = \Theta(n^2)$

Règles de calculs

Propriétés

Soient f et g deux fonctions de $\mathbb{N}$ dans $\mathbb{R}$.

— **Constante multiplicative :**

$$O(c\, f(n)) = O(f(n)) \qquad (c > 0)$$

Valable aussi pour Ω et Θ.

— **Terme dominant (addition) :**

$$O(f(n) + g(n)) = O(\max\{f(n), g(n)\})$$

Idem pour Ω et Θ.

— **Multiplication :**

$$O(f(n)) \times O(g(n)) = O(f(n)g(n))$$

Idem pour Ω et Θ.

Conséquences :

— On peut négliger les moindres croissances. Par exemple : $\Theta(n^2 + n) = \Theta(n^2)$

— On peut négliger les coefficients positifs constants. Par exemple : $\Theta(2n) = \Theta(n)$

Abus de notation :

$$\sum_{i=1}^{n} O(f(i)) = O\left(\sum_{i=1}^{n} f(i)\right)$$

Idem pour Ω et Θ.

Cela signifie que la somme de termes bornés par $f(i)$ l'est elle-même par la somme des $f(i)$ (à une constante près).

Exemple

Dans notre algorithme de recherche d'un élément parmi n :
- La complexité du pire cas est $\hat{C}(n) = \Theta(n)$
- Celle du meilleur cas est $\check{C}(n) = \Theta(1)$
- La complexité moyenne est $\bar{C}(n) = \Theta(\frac{n+1}{2}) = \Theta(n)$

Familles de complexité

Logarithmique

$$f(n) = \log_a(n) \text{ avec } a > 1$$

Il s'agit des complexités dans les cas de division des intervalles parcourus (par exemple dans les recherches dichotomiques).

Propriétés

Avec $a > 1$:
- Pour $n > 0$, $\log_a(n) = \frac{\ln(n)}{\ln(a)}$
- Pour $n > 0$, $n = a^{\log_a(n)}$
- Pour $x > 0$ et $y > 0$, $\log_a(xy) = \log_a(x) + \log_a(y)$
- Pour $b > 1$, $b^{\log_a(n)} = n^{\log_a(b)}$

Polynomiale

$$f(n) = a_k n^k + a_{k-1} n^{k-1} + ... + a_0 \text{ avec } a_k > 0$$

Il s'agit des complexités classiques de parcours de tableaux.

Si $k = 1$, on parle de complexité linéaire, si $k = 2$, c'est quadratique.

Propriétés

- $f(n) = \Theta(n^k)$
- On peut étendre aux valeurs réelles de k : $n\sqrt{n} = n^{1.5} = O(n^2)$

— Avec $a > 0$ et $b \in \mathbb{R}$:

$$\log(n)^b = O(n^a)$$

Exponentielle

$$f(n) = a^n \text{ avec } a > 1$$

Il s'agit de complexités où l'on veut tester par exemple toutes les combinaisons d'un code de taille et d'alphabet fixé.

Propriétés

— **Notations :** $a^0 = 1$, $a^1 = a$, $a^{-1} = \dfrac{1}{a}$
— **Multiplication :** $a^m \times a^n = a^{m+n}$
— **Composition d'exponentielle :** $(a^n)^m = a^{n \times m}$
— Avec $a > 1$ et $b \in \mathbb{R}$:

$$n^b = O(a^n)$$

Factorielle

$$f(n) = n!$$

Il s'agit de complexités où l'on veut tester par exemple tous les chemins possibles d'un graphe.

Propriétés

— **Convention :** $0! = 1$
— **Notation :** $n! = n \times (n-1) \times (n-2) \times ... \times 1$
— Avec $a > 1$:

$$a^n = O(n!)$$

Exercices d'analyse

Exercice : Analyser la complexité asymptotique

ALGO 1 :

```
DEBUT
  i ← 1
  TQ i   n FAIRE
    j ← 1
    TQ j   2^i FAIRE
      j ← j+1
    FTQ
    i ← i+1
  FTQ
FIN
```

ALGO 2 :

```
DEBUT
  i ← n
  TQ i   1 FAIRE
    i ←  i/2
  FTQ
FIN
```

Solution ALGO 1 :
Boucle intérieure : la variable j est initialisée à 1, elle est incrémentée de 1 et va jusqu'à 2^i inclus. Donc le nombre d'opérations faites est de $\sum_{j=1}^{2^i} 1 = 2^i$.
Boucle principale : la variable i est initialisée à 1, elle est incrémentée de 1 et va jusqu'à n inclus. Donc le nombre d'opérations faites est de :

$$\sum_{i=1}^{n}(2^i + 2) = \sum_{i=1}^{n} 2^i + \sum_{i=1}^{n} 2 = 2 \times \frac{2^n - 1}{2 - 1} + 2n = 2^{n+1} - 2 + 2n$$

$C(n)$ est donc en $\Theta(2^n)$.
Solution ALGO 2 :
Boucle principale : la variable i est initialisée à n, on va la diviser par 2 jusqu'à ce qu'elle soit inférieure à 1.
Soit (u_k) la suite des valeurs prises par i. Alors, pour tout $k \geq 0$, $\frac{n}{2^k} - 1 \leq u_k \leq \frac{n}{2^k}$.
La condition d'arrêt est vérifiée si $i < 1$, or cette condition est vérifiée dès que $\frac{n}{2^k} < 1 \Leftrightarrow k > \log_2(n)$ et cette condition n'est pas vérifiée tant que $\frac{n}{2^k} - 1 > 1 \Leftrightarrow k < \log_2(\frac{n}{2})$.
Donc le nombre d'itérations est en $\Theta(\log(n))$.