Calculs de complexité

I Définitions et premières propriétés

I.1 Complexité en temps

Définition .

• La complexité **en temps** d'un algorithme **dans le pire des cas** est le plus grand nombre possible d'opérations élémentaires effectuées pour obtenir le résultat en fonction de la taille de la donnée entrée.

• La complexité **en temps** d'un algorithme **dans le meilleur des cas** est le plus petit nombre possible d'opérations élémentaires effectuées pour obtenir le résultat en fonction de la taille de la donnée entrée.

• La complexité **en temps** d'un algorithme **en moyenne** est le nombre **moyen** d'opérations élémentaires effectuées pour obtenir le résultat en fonction de la taille de la donnée entrée (on considère équiprobables toutes les données de taille n).

• Les notions d'*opération élémentaire* et de *taille des données* dépend du contexte où elle doit être précisée. On dit qu'on a différents *modèles de calcul*.

Sauf mention explicite du contraire, on s'intéresse uniquement à la complexité dans le pire des cas.

Remarque 1

Le nombre de chiffres en base 2 de l'entier n est $\lfloor \log_2(n) \rfloor$.

Remarque 2

On peut donner une définition plus formelle et totalement explicite, mais elle n'est ni praticable ni utile.

I.2 Complexité en espace

Définition .

• La complexité **en espace** d'un algorithme **dans le pire des cas** est la plus grande taille possible de la mémoire utilisée pour obtenir le résultat en fonction de la taille de la donnée entrée.

• La complexité **en espace** d'un algorithme **dans le meilleur des cas** est la plus petite taille possible de la mémoire utilisée pour obtenir le résultat en fonction de la taille de la donnée entrée.

• La complexité **en espace** d'un algorithme **en moyenne** est le taille **moyenne** de la mémoire utilisée pour obtenir le résultat en fonction de la taille de la donnée entrée (on considère équiprobables toutes les données de taille n).

• Le sens du mot "taille" est précisé par le contexte.

Sauf mention explicite du contraire, on s'intéresse uniquement à la complexité dans le pire des cas.

Remarque 3

Si le résultat est de taille $f(n)$, la complexité en espace est **au moins** de $f(n)$, mais elle peut être plus importante.

Exemple 1

Pour calcul itératif des binomiaux vu en TP, le résultat est un entier.

On considère ici les entiers comme étant de taille $O(1)$; la complexité en espace est $O(nk)$.

I.3 Expression d'une complexité

Définition (Rappel).

• On dit que $(c_n)_n$ est dominée par $(u_n)_n$ et on note $c_n = O(u_n)$ lorsqu'il existe une constante $K > 0$ telle que $c_n \leqslant K u_n$ APCR.

• On dit que $(c_n)_n$ est de même ordre de grandeur que $(u_n)_n$ et on note $c_n = \Theta(u_n)$ lorsqu'il existe des constantes $k_1, k_2 > 0$ telles que $k_1 u_n \leqslant c_n \leqslant k_2 u_n$ APCR.

En pratique, on essaie d'exprimer une complexité c_n par son ordre de grandeur : $c_n = \Theta(\cdots)$.

Lorsque ce n'est pas possible, on se contente d'une domination : $c_n = O(\cdots)$.

Remarque 4 $\log_b(n) = \Theta(\lg(n))$.

I.4 Complexités usuelles

On s'intéressera principalement aux algorithmes dont la complexité c_n est :

1. Bornée : $c_n = O(1)$.
2. Logarithmique : $c_n = \Theta(\lg(n))$.
3. Linéaire : $c_n = \Theta(n)$.
4. Quasi-linéaire : $c_n = \Theta(n \lg n)$.
5. Quadratique : $c_n = \Theta(n^2)$.
6. Polynomiale : $c_n = \Theta(n^k)$.
7. Exponentielle : $\lambda^n = O(c_n), \lambda > 1$.

II Exemples

II.1 Somme et produit de deux entiers

Exemple 2 Déterminons le nombre $c_{a,b}$ d'**additions bit-à-bit** dans le calcul de $a + b$.

— $\overline{43}^{10} = \overline{101011}^2$

— $\overline{21}^{10} = \overline{010101}^2$

— $\overline{64}^{10} = \overline{1000000}^2$

On fait $\Theta(1)$ opérations bit à bit pour chaque chiffre, il y a

$$\begin{cases} \lg(a) & \text{chiffres pour } a \\ \lg(b) & \text{chiffres pour } b \end{cases}$$

On a donc $c_{a,b} = \Theta(\max\{\lg a, \lg b\})$

Exemple 3 Déterminons le nombre $c_{a,b}$ d'**additions bit-à-bit** dans le calcul de $a \times b$.
Cela dépend évidemment de l'algorithme choisi !
On fait

$$\underbrace{43 + 43 + \cdots + 43}_{21 \text{ fois}}$$

$$\begin{array}{ll} 43 & \text{taille } \log(a) \\ +43 & \text{taille } \log(a+1) \\ \vdots & \\ +43 & \text{taille } \log(a+1) \end{array}$$

D'où $c_{a,b} = \Theta(b \lg a)$

Exemple moins naïf

$$\begin{array}{rl} & \overline{43}^{10} = \overline{101011}^2 \\ \times & \\ & \overline{21}^{10} = \overline{010101}^2 \\ = & \overline{1110000111}^2 \end{array}$$

On a :

$$\begin{aligned} c_{a,b} &= \Theta((\log a + 1) + \cdots + (\log a + \log b)) \\ &= \Theta(\log a \log b) \end{aligned}$$

On verra en TD **l'algorithme de Karastuba** qui permet de calculer plus efficacement un tel produit à l'aide d'une stratégie de type "diviser pour régner".

II.2 Factorielle

Exemple 4
On a vu plusieurs versions d'un algorithme de calcul de $n!$ mais ce sont différentes versions **d'un même algorithme**.

Notons c_n le nombre de **multiplications** dans le calcul de $n!$.

$$f(n) := \begin{cases} 1 & \text{si } n = 0 \\ n \cdot f(n-1) & \text{sinon} \end{cases}$$

$$\begin{aligned} c_0 &= 0 \\ c_n &= 1 + c_{n-1} \\ &= 1 + 1 + c_{n-2} && = 2 + c_{n-2} \\ &= 2 + 1 + c_{n-3} && = 3 + c_{n-3} \\ &\vdots \\ &= n + c_0 && \boxed{c_n = n} \end{aligned}$$

$$\begin{aligned} c_n &= \Theta(1) + c_{n-1} \\ &= \Theta(1) + \Theta(n) + c_{n-1} \\ &= \underbrace{\Theta(1) + \cdots + \Theta(1)}_{n \text{ fois}} + c_0 \\ &= n\Theta(1) \\ &= \Theta(n) \end{aligned}$$

Complexité en espace

$$s_n = \Theta(1)$$

II.3 ExponentiationS

Exemple 5

On a vu plusieurs algorithmes de calcul de x^n et c'étaient vraiment **des algorithmes différents**.

$$x^n = \begin{cases} 1 & \text{si } n = 0 \\ x \cdot x^{n-1} & \text{sinon} \end{cases}$$

D'où

$$c_n = \begin{cases} 0 & \text{si } n = 0 \\ 1 + c_{n-1} & \text{sinon} \end{cases}$$

On a $c_n = c_{n-1} + \Theta(1)$ d'où

$$c_n = \Theta(n)$$

Exponentiation rapide

$$x^n := \begin{cases} 1 & \text{si } n = 0 \\ x & \text{si } n = 1 \\ a \cdot a & \text{si } a \mod 2 = 0 \text{ avec } a = x^{\frac{n}{2}} \\ x \cdot a \cdot a & \text{si } a \mod 2 = 1, \text{ avec } a = x^{\lfloor \frac{n}{2} \rfloor} \end{cases}$$

Ainsi :

$$\begin{aligned} c_n &:= \begin{cases} 0 & \text{si } n = 0 \\ 0 & \text{si } n = 1 \\ 1 + c_{\frac{n}{2}} & \text{si } a \mod 2 = 0 \text{ avec } a = x^{\frac{n}{2}} \\ 2 + c_{\lfloor \frac{n}{2} \rfloor} & \text{si } a \mod 2 = 1, \text{ avec } a = x^{\lfloor \frac{n}{2} \rfloor} \end{cases} \\ &= \begin{cases} 0 & \text{si } n \in \{0, 1\} \\ 2 + c_{\lfloor \frac{n}{2} \rfloor} & \text{sinon} \end{cases} \\ \implies c_n &= c_{\lfloor \frac{n}{2} \rfloor} + \Theta(1) \\ &= \Theta(1) + \Theta(1) + c_{\frac{n}{4}} && \text{à partir d'ici, on note } \frac{a}{b} := \left\lfloor \frac{a}{b} \right\rfloor \\ &= \underbrace{\Theta(1) + \cdots + \Theta(1)}_{k \text{ fois}} + c_{\frac{n}{2^k}} \\ &= \underbrace{\Theta(1) + \cdots + \Theta(1)}_{k \text{ fois}} + c_{0 \text{ ou } 1} && \text{pour } k = \lceil \lg n \rceil \\ &= \Theta(k) \\ &= \Theta(\lceil \lg n \rceil) \\ &= \Theta(\lg n) \end{aligned}$$

D'où $c_n = \Theta(\lg n)$

Exponentiation qui se croit rapide

$$x^n = \begin{cases} 1 & \text{si } n \in \{0,1\} \\ x^{\frac{n}{2}} \cdot x^{\frac{n}{2}} & \text{si } n \text{ est pair} \\ x \cdot x^{\frac{n}{2}} \cdot x^{\frac{n}{2}} & \text{si } n \text{ est impair} \end{cases}$$

On a :

$$\begin{aligned} c_n &= \mathbf{2}c_{\frac{n}{2}} + \Theta(1) \\ &= \Theta(1) + 2\Theta(1) + 4c_{\frac{n}{4}} \\ &= \Theta(1) + 2\Theta(1) + 4\Theta(1) + 8c_{\frac{n}{8}} \\ &\vdots \\ &= \Theta(1) + \cdots + 2^k\Theta(1) + 2^k c_{\frac{n}{2^k}} \\ &= \Theta(1 + 2 + 4 + \cdots + 2^k) + c_0 && \text{pour } k = \lceil \lg n \rceil \\ &= \Theta(1)(2^{k+1} - 1) + 0 \\ &= \Theta(2^k 2 - 1) \\ &= \Theta(2^k) \\ &= \Theta(n) \end{aligned}$$

Complexité spatiale

$$\Theta(1).$$

Youpie.

II.4 Tris

Exemple 6
On a vu plusieurs algorithmes de tris et c'étaient vraiment **des algorithmes différents**.

Tri par insertion On a

$$c_n = c_{n-1} + \Theta(1)$$

D'où

$$c_n = \Theta(n^2)$$

Tri par sélection C'est pareil, juste la sélection est couteuse au lieu de l'insertion

$$c_n = \Theta(n^2)$$

Tri par fusion Trois étapes :

1. scission (coûte n)
2. les deux appels récursifs (coûte deux fos $c_{\frac{n}{2}}$)
3. la fusion (coûte n)

D'où

$$\begin{aligned}
c_n &= 2c_{\lceil \frac{n}{2} \rceil} + \Theta(n) \\
&= \Theta(n) + 2c_{\lceil \frac{n}{2} \rceil} \\
&= \Theta(n) + 2\Theta(\frac{n}{2}) + 4c_{\lceil \frac{n}{4} \rceil} \\
&= \Theta(n) + 2\Theta(\frac{n}{2}) + 4\Theta(\frac{n}{4}) + 8c_{\lceil \frac{n}{8} \rceil} \\
&\vdots \\
&= \underbrace{\Theta(n) + \cdots + 2^k\Theta(\frac{n}{2^k})}_{k \text{ fois}} + 2^{k+1}c_0 \\
&= \underbrace{\Theta(n) + \cdots + \Theta(n)}_{k \text{ fois}} \\
&= \Theta(nk) \\
&= \Theta(n \lg n)
\end{aligned}$$

Tri par pivot Dans le cas `liste = []`

1. scission : n
2. deux appels récursifs : le pivot est le plus grand ou le plus petit, donc c_{n-1} et 1
3. fusion

On a $c_n = c_{n-1} + \Theta(1) = \Theta(n^2)$.
Dans le meilleurs cas (autant d'éléments supérieurs au pivot que inférieur)
Alors, les appels récursifs sont sur $floor\frac{n}{2}$.
On a ainsi

$$c_n = 2c_{\lfloor \frac{n}{2} \rfloor} + \Theta(n) = \Theta(n \lg n)$$

II.5 Fibonacci

Exemple 7
On a vu plusieurs algorithmes de calcul de F_n et c'étaient vraiment **des algorithmes différents**.
On a $c_0 = c_1 = 1$ et pour $n \neq 0$, $c_n = c_{n-1} + c_{n-2} + 1$.
Même relation de récurrence que pour Fibonacci. Si on a le temps, on montre qu'on a $c_n = \Theta(F_n)$.
Un peu de maths : $F_n = \frac{1}{\sqrt{5}}\left(\phi^n + \left(\frac{-1}{\phi}\right)^n\right)$.
Conclusion, $c_n = \Theta(\phi^n)$

III Méthodes de calcul générales

III.1 Itération

Remarque 5
Complexité d'une composition : un algorithme obtenu en appliquant successivement des algorithmes de complexités $c_{1,n}$, $c_{2,n}$, …, $c_{r,n}$ a pour complexité $c_n = c_{1,n_1} + c_{2,n_2} + \cdots + c_{r,n_r}$.

Ceci permet d'estimer la complexité d'une boucle à l'aide du théorème suivant.

Théorème (Sommation de O et de Θ).

- $\sum_{k=1}^{n} O(k^\alpha) = O(n^{\alpha+1})$ et $\sum_{k=1}^{n} \Theta(k^\alpha) = \Theta(n^{\alpha+1})$.
- $\sum_{k=1}^{n} O(k^\alpha \lg^\beta(k)) = O(n^{\alpha+1} \lg^\beta(n))$ et $\sum_{k=1}^{n} \Theta(k^\alpha \lg^\beta(k)) = \Theta(n^{\alpha+1} \lg^\beta(n))$.
- Pour $\lambda > 1$, $\sum_{k=1}^{n} O(\lambda^k) = O(\lambda^{n+1}) = O(\lambda^n)$.

III.2 Récursivité

Remarque 6

Complexité d'une reconstruction : en notant c_n la complexité d'un algorithme récursif avec appels récursifs est de la forme $f(n) = g(f(m_1), \dots, f(m_r))$, on a $c_n = c_{m_1} + \cdots + c_{m_r} + d_m$ où d_m est le nombre d'opérations pour le calcul de $g(a_1, \dots, a_r)$ une fois calculés les $a_i = f(m_i)$.

Théorème (Suites récurrentes simples classiques).

- La récurrence $c_n = c_{n-1} + \Theta(1)$ conduit à $c_n = \Theta(n)$.
- Pour $\alpha > 0$, la récurrence $c_n = c_{n-1} + \Theta\big(n^\alpha\big)$ conduit à $c_n = \Theta\big(n^{\alpha+1}\big)$.
- Pour $\lambda > 1$, la récurrence $c_n = \lambda c_{n-1} + O\big(n^\alpha\big)$ conduit à $c_n = \Theta(\lambda^n)$.
- Si $X^2 - aX - b$ a deux racines $\lambda_1 < \lambda_2$ alors la récurrence $c_n = ac_{n-1} + bc_{n-2} + \gamma$ conduit à $c_n = \Theta(\lambda_2^n)$.

Exemples 8

Stratégie "diviser pour régner".
On divise le problème en sous-problèmes de taille $\lfloor \frac{n}{\lambda} \rfloor$, on traite récursivement les sous-problèmes, on reconstruit la solution globale à l'aide des solutions partielles.

— coût de la division en sous-problèmes : d(n)

— coût d'un sous-problème : $c_{\lfloor \frac{n}{\lambda} \rfloor}$

— coût de la reconstruction : r(n)

Le coût total est : $c_n = ac_{\lfloor \frac{n}{\lambda} \rfloor} + f(n)$ où a est le nombre de sous-problèmes et $f(n) = d(n) + r(n)$.

Exemples 9

Théorème (Stratégie "diviser pour régner").

- La récurrence $c_n = a\, c_{\lfloor \frac{n}{2} \rfloor} + \Theta(n^\beta)$ conduit à
  (a) $c_n = \Theta(n^\alpha) = \Theta(a^{\lg(n)})$ si $\beta < \alpha = \lg(a)$.
  (b) $c_n = \Theta(n^\alpha \lg(n)) = \Theta(\lg(n) a^{\lg(n)})$ si $\beta = \alpha = \lg(a)$.
  (c) $c_n = \Theta(n^\beta)$ si $\beta > \alpha = \lg(a)$.
- Trois cas particuliers utiles :
  (a) La récurrence $c_n = c_{\lfloor \frac{n}{2} \rfloor} + \Theta(1)$ conduit à $c_n = \Theta(\lg(n))$.
  (b) Les récurrences $c_n = 2c_{\lfloor \frac{n}{2} \rfloor} + \Theta(1)$ et $c_n = c_{\lfloor \frac{n}{2} \rfloor} + c_{\lceil \frac{n}{2} \rceil} + \Theta(1)$ conduisent à $c_n = \Theta(n)$.
  (c) Les récurrences $c_n = 2c_{\lfloor \frac{n}{2} \rfloor} + \Theta(n)$ et $c_n = c_{\lfloor \frac{n}{2} \rfloor} + c_{\lceil \frac{n}{2} \rceil} + \Theta(n)$ conduisent à $c_n = \Theta(n \lg(n))$.

On retrouve les résultats vus précédemment.