Équations de récurrence
(Résumé de l'annexe B du manuel)

Table des matières

B Résolution d'équations de récurrence et applications à l'analyse des algorithmes

L'utilisation de la statégie diviser-pour-régner avec récursivité conduit souvent à des fonctions définissant le temps d'exécution ayant l'allure suivante :

$$T(n) = aT(n/b) + f(n)$$

Plus précisément, une telle équation serait associée à une application de la stratégie diviser-pour-régner ayant les caractéristiques suivantes :

1. Le problème initial est décomposé en a sous-problèmes.
2. Chaque sous-problème est de taille n/b.
3. Le coût pour décomposer le problème de taille n en ses sous-problèmes *puis* combiner les solutions des sous-problèmes est donné par la fonction $f(n)$.

En d'autres mots, la structure de la récursion possède l'allure indiquée à l'algorithme 1, en supposant une décomposition récursive jusqu'au cas de base trivial, où `n` indique la taille du problème.

```
PROCEDURE proc( p: Probleme, n: Nat ) sol: Solution
  SI n = 1 ALORS
    sol <- on résout le problème simple p
  SINON
     (p_1, ..., p_a) <- on décompose p en a sous-problèmes de taille n/b
     POUR i <- 1 A a FAIRE
       sol_i <- proc( p_i, n/b )
     FIN
     sol <- on combine les solutions sol_1, ..., sol_a
  FIN
FIN
```

Algorithme 1: Algorithme diviser-pour-régner illustrant les paramètres d'une équation de récurrence générique

Quelques exemples vus précédemment illustrant les constantes a, b et la fonction $f(n)$:

- Fouille binaire :
  - $b = 2$: on divise l'intervalle en deux moitiés.
  - $a = 1$: on fouille *une* seule des deux moitiés.
  - $f(n) = \Theta(1)$: pour la comparaison.
- Tri par fusion :
  - $a = 2$: deux parties à trier.
  - $b = 2$: chaque partie compte exactement la moitié des éléments.
  - $f(n) = \Theta(n)$: coût pour division ($\Theta(1)$) et coût de la fusion ($\Theta(n)$).
- Algorithme de Strassen (en comptant toutes les opérations arithmétiques et en supposant une matrice carrée $n \times n$, n donnant donc la taille du problème) :
  - $a = 7$: sept produits de sous-matrices.

– $b = 2$: matrice carrée de taille $\frac{n}{2} \times \frac{n}{2}$.
– $f(n) = \Theta(n^2)$

On verra à la section B.4 (p. 4) une méthode simple et assez générale permettant de résoudre une grande partie des équations de cette forme. Toutefois, nous examinerons tout d'abord d'autres méthodes qui peuvent aussi s'appliquer même dans le cas où l'équation de récurrence *n'est pas* de la forme précédente, par exemple, si la taille du sous-problème est $n - 1$, conduisant à une équation de la forme $T(n) = T(n-1) + f(n)$.

B.1 Résolution à l'aide de l'induction mathématique

Une première façon de trouver une solution à une équation de récurrence est de *deviner* la solution (*sic*) puis de prouver, à l'aide de l'induction mathématique, que cette solution est valable (méthode dite "*guess-and-test*").

Exemple :

- $T(n) = T(n/2) + 1$, pour $n > 1$
- $T(1) = 1$

Quelques calculs nous indiquent une solution possible (on utilise des n puissance de 2 à cause de l'expression $n/2$) :

- $T(1) = 1$
- $T(2) = T(1) + 1 = 2$
- $T(4) = T(2) + 1 = 3$
- $T(8) = T(4) + 1 = 4$
- ...

Montrons que $T(n) = \lg n + 1$, donc que $T(n) \in \Theta(\lg n)$, pour $n = 2^k$:

- Cas de base :
  $T(1) = 1 = 0 + 1 = \lg 1 + 1$.
- Cas inductif (induction généralisée) :
  Supposons $T(m) = \lg m + 1$ pour $m < n$.

$$\begin{aligned} T(n) &= T(n/2) + 1 \\ &= (\lg (n/2) + 1) + 1 \\ &= \lg n - \lg 2 + 1 + 1 \\ &= \lg n - 1 + 1 + 1 \\ &= \lg n + 1 \end{aligned}$$

On peut donc en conclure que pour tout $n \geq 1$, $T(n) = \lg n + 1$.

B.2 Résolution à l'aide de l'équation caractéristique

Section omise.

B.3 Résolution par substitution

Une autre façon de résoudre une équation de récurrence est de substituer, de façon répétitive, la définition de la fonction dans le coté droit de son équation jusqu'à ce qu'une forme simple soit obtenue (un peu comme on l'a fait dans les exemples du Chapitre 2) — Goodrich et Tamassia (Data Structures and Algorithms in `Java`, John Wiley & Sons, 1998) appellent cette méthode "*plug-and-chug*".

Exemple :

- $T(n) = T(n-1) + n$, pour $n > 1$
- $T(1) = 1$

Solution par substitution répétitive :

$$\begin{aligned}
T(n) &= T(n-1) + n \\
&= [T(n-2) + (n-1)] + n \\
&= [[T(n-3) + (n-2)] + (n-1)] + n \\
&= \ldots \\
&= T(1) + 2 + \ldots + (n-2) + (n-1) + n \\
&= 1 + 2 + \ldots + (n-2) + (n-1) + n \\
&= \sum_{i=1}^{n} i \\
&= \frac{n(n+1)}{2} \\
&\in \Theta(n^2)
\end{aligned}$$

B.4 Comment étendre un résultat pour n défini comme une puissance de b à un n général

Il est généralement préférable (pcq. plus simple), dans le cas où l'équation est de la forme $T(n) = aT(n/b) + f(n)$, de travailler en supposant que $n = b^k$.

Supposons que $f(n)$ est une fonction *éventuellement non décroissante*, donc satisfaisant la condition suivante :

$$\text{il existe } N \text{ tel que si } N < n_1 < n_2 \text{ alors } f(n_1) \leq f(n_2)$$

En d'autres mots, à partir d'un certain point N, la fonction n'est pas décroissante. Exemples :

1. Figure B.1 (p. 574 du manuel)
2. Figure 1.

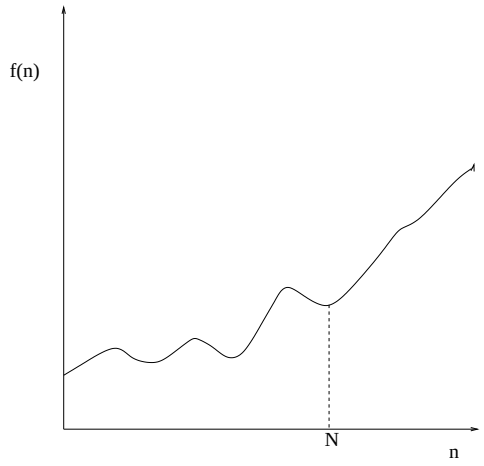

Figure 1: Fonction éventuellement non-décroissante

Supposons ensuite que $f(n)$ soit une fonction de complexité qui soit *lisse* (*smooth*) donc qui satisfaisait les conditions suivantes :

1. $f(n)$ est éventuellement non décroissante.
2. $f(2n) \in \Theta(f(n))$.

Note : $\lg n$, n, $n \lg n$ et n^k sont toutes des fonctions de complexité lisses. Par contre, 2^n ne l'est pas.

Théorème 1 *Soit $b \geq 2$ un entier, $f(n)$ une fonction de complexité lisse, et $T(n)$ une fonction de complexité éventuellement non décroissante.*

Si $T(n) \in \Theta(f(n))$ pour n une puissance de b, alors $T(n) \in \Theta(f(n))$ pour un n arbitraire.

Note : La même propriété s'applique pour O, Ω et o.

Résolution à l'aide du théorème général (version Neapolitan and Naimipour)

Théorème 2 (Théorème général) *Soit $T(n)$ une fonction de complexité éventuellement non décroissante définie par les équations de récurrence suivantes, où $b \geq 2$, $k \geq 0$, $a > 0$, $c > 0$ et $d > 0$:*

- $T(n) = aT(\frac{n}{b}) + cn^k$, *pour $n > 1$, n une puissance de b.*
- $T(1) = d$.

Alors, la relation entre a, b et k détermine la fonction $T(n)$ comme suit :

1. *Si $a > b^k$, alors $T(n) \in \Theta(n^{log_b\ a})$.*
2. *Si $a = b^k$, alors $T(n) \in \Theta(n^k \lg n)$.*
3. *Si $a < b^k$, alors $T(n) \in \Theta(n^k)$.*

Note : On remarque que c et d n'ont aucune influence sur la solution. La constante c est simplement la constante de proportionalité indiquant que le travail de décomposition et recombinaison des sous-solutions est $\Theta(n^k)$. Quant au d, il indique simplement que le travail dans le cas du problème trivial de taille 1 est $\Theta(1)$.

Note : Si dans l'équation $T(n) = aT(\frac{n}{b}) + cn^k$ on substitue "=" par "$\leq$" (resp. "$\geq$"), alors on peut substituer "Θ" par "O" (resp. "Ω") dans le résultat. Par exemple :

- $T(n) \leq 4T(\frac{n}{2}) + cn^2$, pour $n > 1$, n une puissance de 2.
- $T(1) \leq d$.

Alors, $T(n) \in O(n^2 \lg n)$.

Note : Le théorème reste valide même si les équations s'appliquent uniquement à partir d'une certaine valeur $s > 1$. En d'autres mots, le théorème reste vrai même si les équations définissant $T(n)$ sont les suivantes (s étant une constante supérieur ou égale à 1) :

- $T(n) = aT(\frac{n}{b}) + cn^k$, pour $n > s$, n une puissance de b.
- $T(s) = d$.

– Quelques exemples d'application :

- Fouille binaire :
  - $a = 1$
  - $b = 2$
  - $f(n) = \Theta(1)$

  Ici, on a $k = 0$, donc $a = b^k$. La solution de l'équation associée est donc $\Theta(n^0 \lg n) = \Theta(\lg n)$.

- Tri par fusion :
  - $a = 2$
  - $b = 2$
  - $f(n) = \Theta(n)$

  On a $k = 1$, donc $a = b^k$. La solution de l'équation associée est donc $\Theta(n^1 \lg n) = \Theta(n \lg n)$.

- Algorithme de Strassen (en comptant toutes les opérations arithmétiques) :
  - $a = 7$
  - $b = 2$
  - $f(n) = \Theta(n^2)$

  On a $k = 2$, donc $a = 7 > 4 = b^k$. La solution est donc $\Theta(n^{log_b\ a}) = \Theta(n^{log_2 7}) \approx \Theta(n^{2.81})$.

Résolution à l'aide du théorème général (version Cormen, Leiserson et Rivest)

Le théorème précédent ne s'applique pas nécessairement de façon précise à tous les cas. Par exemple, soit les équations de récurrences suivantes :

- $T(n) = 3T(n/4) + n \lg n$, pour $n > 1$
- $T(1) = d$

Ici, la fonction $f(n) = n \lg n$ *n'est pas* de la forme cn^k. Par contre, on pourrait utiliser la version moins forte du théorème précédent, c'est-à-dire, la version utilisant "$\leq$" plutôt que "$=$" et utilisant O plutôt que Θ. Sachant que $n \lg n \in O(n^2)$ et en utilisant donc $k = 2$, on a que $3 < 4^2$, d'où on peut conclure que $T(n) \in O(n^2)$. Or, bien que cette ne solution pas fausse, la réponse plus précise est en fait $T(n) \in \Theta(n \lg n)$.

Cette dernière solution peut être obtenue à l'aide de la version plus générale du théorème général (*sic*) introduite par Cormen, Leiserson et Rivest.

Théorème 3 (Théorème général (version Cormen *et al.*)) *Soient $a \geq 1$ et $b > 1$ deux constantes, soit $f(n)$ une fonction, et soit $T(n)$ définie par la récurrence suivante :*

- $T(n) = aT(\frac{n}{b}) + f(n)$, *pour* $n > 1$, n *une puissance de* b.
- $T(1) = d$.

Alors, $T(n)$ peut être bornée asymptotiquement comme suit :

1. *Si $f(n) \in O(n^{log_b\ a-\epsilon})$ pour une certaine constante $\epsilon > 0$, alors $T(n) \in \Theta(n^{log_b\ a})$.*
2. *Si $f(n) \in \Theta(n^{log_b\ a})$, alors $T(n) \in \Theta(n^{log_b\ a} \lg n)$.*
3. *Si $f(n) \in \Omega(n^{log_b\ a+\epsilon})$ pour une constante $\epsilon > 0$, et si $af(n/b) \leq cf(n)$ pour une constante $c < 1$ et pour tous les n suffisamment grands, alors $T(n) \in \Theta(f(n))$.*

Exemple :

- $T(n) = 3T(n/4) + n \lg n$

  Ici, $log_b\ a = log_4\ 3 \approx 0.793$. Donc, pour $f(n) = n \lg n$, $f(n) \notin O(n^{0.793-\epsilon})$ et $f(n) \notin \Theta(n^{0.793})$. Par contre, on a bien que $f(n) \in \Omega(n^{0.793+\epsilon})$ (pour $\epsilon = 0.207$ et n suffisamment grand). De plus, $3f(n/4) = 3(n/4) \lg (n/4)) \leq 3(n/4) \lg n = \frac{3}{4} n \lg n$ — d'où $c = 0.75 < 1$ dans la clause 3. Donc, on peut conclure que $T(n) \in \Theta(n \lg n)$, ce qui est nettement plus précis que $T(n) \in O(n^2)$.

Remarque : Dans l'énoncé de leur théorème, Cormen, Leiserson et Rivest indiquent explicitement que $\frac{n}{b}$ peut être interprété aussi bien comme $\lfloor \frac{n}{b} \rfloor$ que comme $\lceil \frac{n}{b} \rceil$. En d'autres mots, le fait de définir et résoudre la récurrence en termes de n une puissance de b conduit au même résultat que de définir et résoudre la récurrence pour des n arbitraires.

Remarque : Le troisième cas dans le théorème général version Cormen, Leiserson et Rivest indique un cas où c'est le travail de décomposition puis de combinaison des solutions qui domine fortement l'ensemble du travail à effectuer (voir plus loin).

– Il est important de souligner que malgré sa plus grande généralité, la version de Cormen, Leiserson et Rivest ne couvre pas tous les cas possibles. Par exemple :

- $T(n) = 2T(n/2) + n \lg n$

À première vue, il pourrait sembler que le cas 3 s'applique, puisque $f(n) = n \lg n$ domine asymptotiquement $n^{log_b\ a} = n^{log_2 2} = n$ (c'est-à-dire, $n \lg n \in \Omega(n)$). Toutefois, cette domination asymptotique n'est pas *polynomialement* plus grande, c'est-à-dire, il n'existe aucun ϵ tel que $n \lg n \in \Omega(n^{1+\epsilon}) = \Omega(n \times n^{\epsilon})$. Plus précisément, en utilisant la définition de Ω, il n'existe aucun ϵ, c et N tels que pour $n > N$, on ait $n \times n^{\epsilon} \leq c\ n \lg n$, c'est-à-dire, $n^{\epsilon} \leq c \lg n$. Le théorème général, version Cormen *et al.*, ne peut donc pas être utilisé non plus pour cette équation.

Quelle est la signification de $log_b\ a$ dans le théoréme général (version simple de Neapolitant et Naimipour)?

On a vu précédemment qu'une équation de récurrence $T(n) = aT(n/b) + cn^k$ peut être inteprétée comme suit :

- $a =$ nombre de sous-problèmes.
- $b =$ facteur qui détermine la taille des sous-problèmes.
- $k =$ facteur qui détermine la complexité de la décomposition en sous-problèmes et de la recombinaison des solutions aux sous-problèmes pour obtenir la solution globale.

Quelle allure aura l'arbre des appels récursifs pour des a et b donnés?

- Nombre de niveaux de l'arbre : $log_b\ n$.
- Facteur de branchement de l'arbre : a.
- Nombre total de noeuds internes : $\sum_{j=0}^{log_b\ n-1} a^j$.
- Coûts associés à un noeud interne de niveau j : $a^j f(n/b^j)$.
- Coûts totaux associés à l'ensemble des noeuds internes :

$$\sum_{j=0}^{log_b\ n-1} a^j\ f(n/b^j)$$

- Nombre total de feuilles : $a^{log_b\ n} = n^{log_b\ a}$.
- Coûts totaux associées aux feuilles :

$$\Theta(n^{log_b\ a})$$

- Coûts totaux :

$$\Theta(n^{log_b\ a}) + \sum_{j=0}^{log_b\ n-1} a^j\ f(n/b^j)$$

Graphiquement, ceci peut être représenté tel qu'illustré à la figure 2.

Les trois cas du théoréme général correspondent alors aux trois cas suivants :

1. Le coût total est dominé par le coût des feuilles, c'est-à-dire, par la résolution des problèmes de base qui sont en très grand nombre.
2. Le coût est réparti régulièrement le long des divers niveaux de l'arbre.
3. Le coût total est dominé par le coût du travail fait au niveau des noeuds internes, c'est-à-dire, par la décomposition en sous-problèmes et combinaison des solutions.

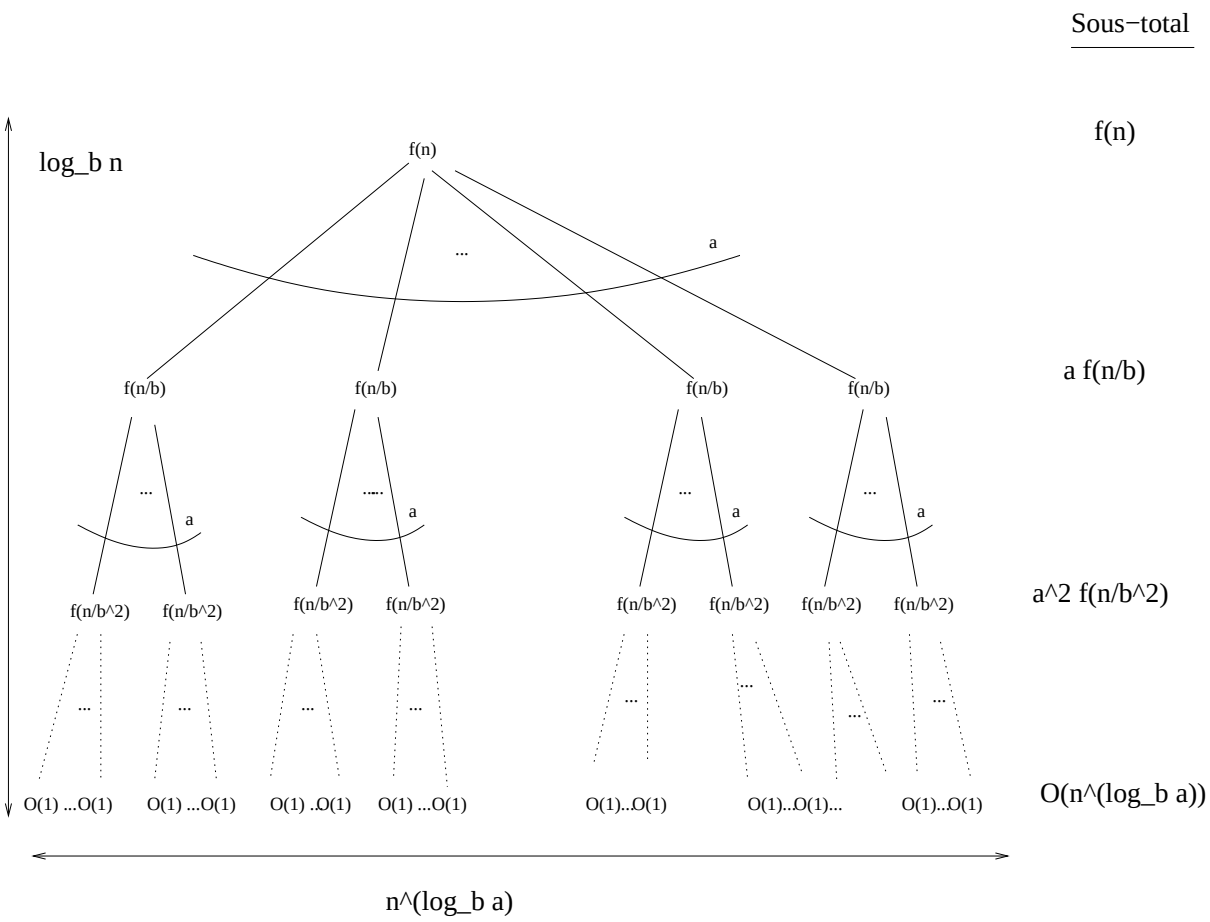

Figure 2: Arbre des appels dans le cas d'une fonction récursive générant a appels de taille $\frac{n}{b}$

Cas spécial intéressant non couvert par les théorèmes généraux

Soit une équation de récurrence de la forme générale (avec a, b et c des entiers non nuls) :

$T(n) = c$, pour $n \leq N$.

$T(n) = aT(n/b) + f(n)$, pour $n > N$.

Supposons que $f(n) \in \Theta(n^{\lg_b a} \lg^k n)$ pour un entier $k \geq 0$, où $\lg^k n$ signifie $(\lg n)^k$.
Alors $T(n) \in \Theta(n^{\lg_b a} \lg^{k+1} n)$.