Chapitre 7

Récurrences

Plan

Lectures conseillées :

- MCS, chapitre 20.
- Rosen, 8.1, 8.2, 8.3
- Introduction to algorithms, Cormen et al. Chapitre 4.

Introduction

Rappel : La notation asymptotique vue dans le chapitre 6 permet d'approximer la complexité des algorithmes.

But de ce chapitre : Étudier des méthodes permettant de résoudre des *équations récurrentes.*

Motivation :

- La complexité des *algorithmes récursifs* est souvent calculable à partir d'équations récurrentes.
- Les récurrences permettent de résoudre des problèmes de *dénombrement*

Complexité d'algorithme récursif : tris

Tri rapide (*Quicksort*) : Nombre de comparaisons en moyenne (voir chapitre 6) :

$$\begin{aligned} C_1 &= 2 \\ C_N &= N+1+\sum_{k=1}^{N}\frac{1}{N}(C_{k-1}+C_{N-k}) \text{ pour } N>1 \end{aligned}$$

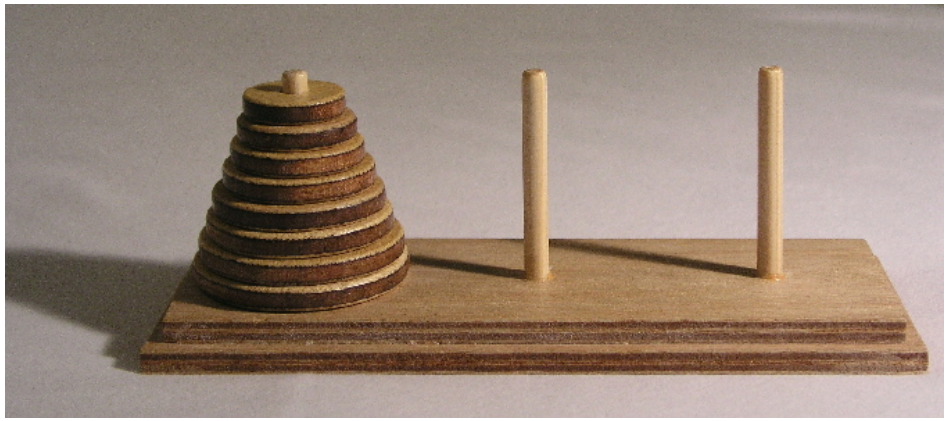
Tri par fusion (*merge sort*) : Pour trier un tableau de n éléments :

1. Diviser la liste en deux ;
2. Trier récursivement les deux sous-listes ;
3. Fusionner les deux listes triées.

Nombre de comparaison (dans le pire cas) :

$$\begin{aligned} C_1 &= 0 \\ C_N &= C_{\lceil N/2 \rceil}+C_{\lfloor N/2 \rfloor}+N-1 \text{ pour } N>1 \end{aligned}$$

Complexité d'algorithmes récursif : Tours de Hanoï



Source : `http ://fr.wikipedia.org/wiki/Tours_de_Hanoï`

But : Déplacer la tour complète de la première tige vers une des deux autres tiges.

Contraintes :

- On ne peut déplacer qu'un seul disque à la fois.
- Un disque ne peut jamais être déposé sur un disque de diamètre inférieur.

Solution récursive :

- Cas de base : Déplacer une tour d'un seul disque est immédiat.
- Cas récursif : Pour déplacer une tour de $n+1$ disques de la première vers la troisième tige en connaissant une solution pour le déplacement d'une tour de n disques :
  1. Par récursion, déplacer n disques vers la deuxième tige ;
  2. Déplacer le disque restant vers la troisième tige ;
  3. Par récursion, déplacer les n disques de la deuxième tige vers la troisième tige.

Notation : Soit T_n le nombre minimum d'étapes nécessaires au déplacement d'une tour de n disques d'une tige vers une autre.

Propriété (borne supérieure) : On a $T_n \leq 2T_{n-1} + 1$.

Remarques :

- Pour déplacer une tour, il faut obligatoirement déplacer le disque du bas.
- Accéder au disque du bas nécessite de déplacer tous les autres disques vers une tige libre (au moins T_{n-1} étapes).
- Ensuite, il faut remettre en place le reste de la tour (au moins T_{n-1} étapes).

On a donc la propriété suivante :
Propriété (borne inférieure) : On a $T_n \geq 2T_{n-1} + 1$.

D'où on déduit la Récurrence :

$$\begin{aligned} T_1 &= 1 \\ T_n &= 2T_{n-1} + 1 \text{ pour } n > 1 \end{aligned}$$

Dénombrement : exemple 1

Problème : Supposons le mécanisme suivant d'engagement des professeurs à Montefiore :

- Chaque professeur :
  - est nommé à vie ;
  - est supposé immortel
  - forme chaque année exactement un étudiant qui deviendra professeur l'année suivante (exception : lors de leur première année d'enseignement, les professeurs sont trop occupés pour former un étudiant).
- Année 0 : il n'y a aucun professeur.
- Année 1 : le premier professeur (autodidacte) est formé.

Combien y aura-t-il de professeurs à la fin de la n-ème année ?

Solution : Soit F_n le nombre de professeurs à la fin de la n-ème année :

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_n &= F_{n-1} + F_{n-2} \text{ pour } n > 1 \end{aligned}$$

Exercice : montrez que le nombre de séquences de n bits qui ne contiennent pas deux 0 consécutifs suit la même récurrence, avec des conditions initiales différentes

Dénombrement : exemple 2

Problème :

- Un programme informatique considère un chaîne constituée de chiffres décimaux comme un mot de passe valide si elle contient un nombre pair de chiffres 0.
- Par exemple “1230407869” et “12345” sont des mots de passe valide mais “120987045608” et “012879” ne le sont pas.
- Calculez T_n, le nombre de mots de passe valides de longueur n.

Solution : Une chaîne valide de n chiffres peut être obtenue de deux façons :

- en ajoutant un chiffre parmi $\{1, \ldots, 9\}$ au début d'une chaîne valide de longueur $n-1$
- en ajoutant un 0 au début d'une chaîne non valide de longueur $n-1$

On a donc :

$$T_n = 9\,T_{n-1} + (10^{n-1} - T_{n-1}) = 8\,T_{n-1} + 10^{n-1}$$

avec $T_0 = 1$.

Dénombrement : exemple 3

Définition : L'ensemble $\mathcal{B}$ des *arbres binaires* est défini comme suit :

- Cas de base : L'arbre vide $\emptyset$ appartient à $\mathcal{B}$.
- Cas récursif : Soit $B_1, B_2 \in \mathcal{B}$, alors $\mathbf{branch}(B_1, B_2) \in bTree$.

Le nombre de nœuds $n(B)$ est défini récursivement par :

- Cas de base : $n(\emptyset) = 0$
- Cas inductif : $n(\mathbf{branch}(B_1, B_2)) = 1 + n(B_1) + n(B_2)$

Théorème : Le nombre b_n d'arbres binaires ayant n nœuds est donné par :

$$\begin{aligned} b_0 &= 1 \\ b_n &= \sum_{k=0}^{n-1} b_k \, b_{n-1-k} \text{ pour } n > 0 \end{aligned}$$

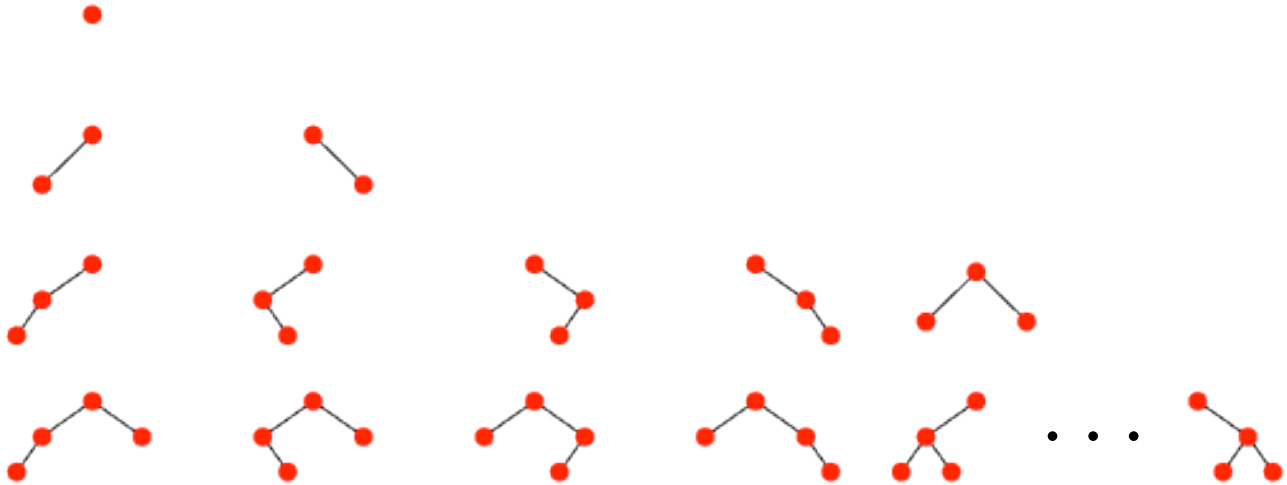
$b_0 = 1$

$b_1 = 1$

$b_2 = 2$

$b_3 = 5$

$b_4 = 14$



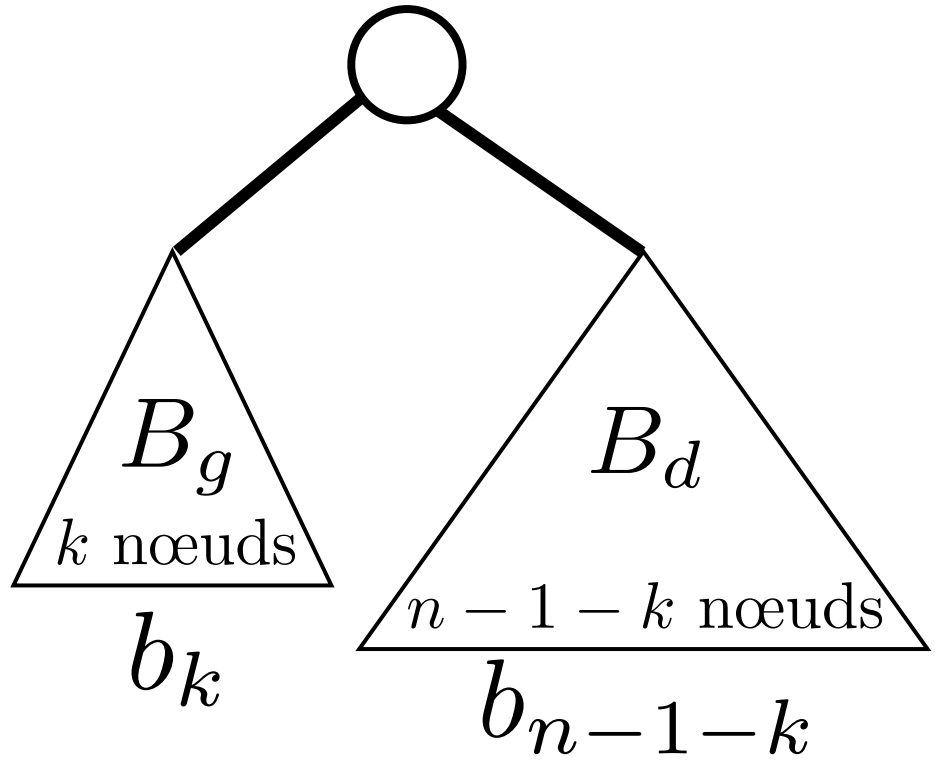
Démonstration : Par induction sur n.

Cas de base : $b_0 = 1$

Cas inductif :

- Un arbre binaire ayant $n > 0$ nœuds ne peut s'obtenir qu'en considérant tous les arbres binaires possibles de la forme **branch**(B_g, B_d) où B_g possède k nœuds et B_d possède $n - k - 1$ nœuds (avec $0 \leq k \leq n - 1$).
- Pour un k donné, il existe donc b_k arbres B_g possibles et b_{n-1-k} arbres B_d possibles.
- On a donc finalement :

$$b_n = \sum_{k=0}^{n-1} b_k \, b_{n-1-k}.$$


B_g
k nœuds
b_k
B_d
$n-1-k$ nœuds
b_{n-1-k}

Nombres de Catalan

Les nombres :

$$\begin{aligned} b_0 &= 1 \\ b_n &= \sum_{k=0}^{n-1} b_k b_{n-1-k} \text{ pour } n > 0 \end{aligned}$$

sont appelés les *nombres de Catalan* (du nom d'un mathématicien belge).

Ils apparaissent dans de nombreux problèmes de dénombrement.

Exercice : Montrez que le nombre b_n de façons de parenthéser le produit de n nombres $x_0 \cdot x_1 \cdots x_n$ satisfait la même récurrence.
Exemple : $b_2 = 2 \rightarrow (x_0 \cdot x_1) \cdot x_2, x_0 \cdot (x_1 \cdot x_2)$

Classification des récurrences

Forme générale : $u_n = f(\{u_0, u_1, \ldots, u_{n-1}\}, n)$

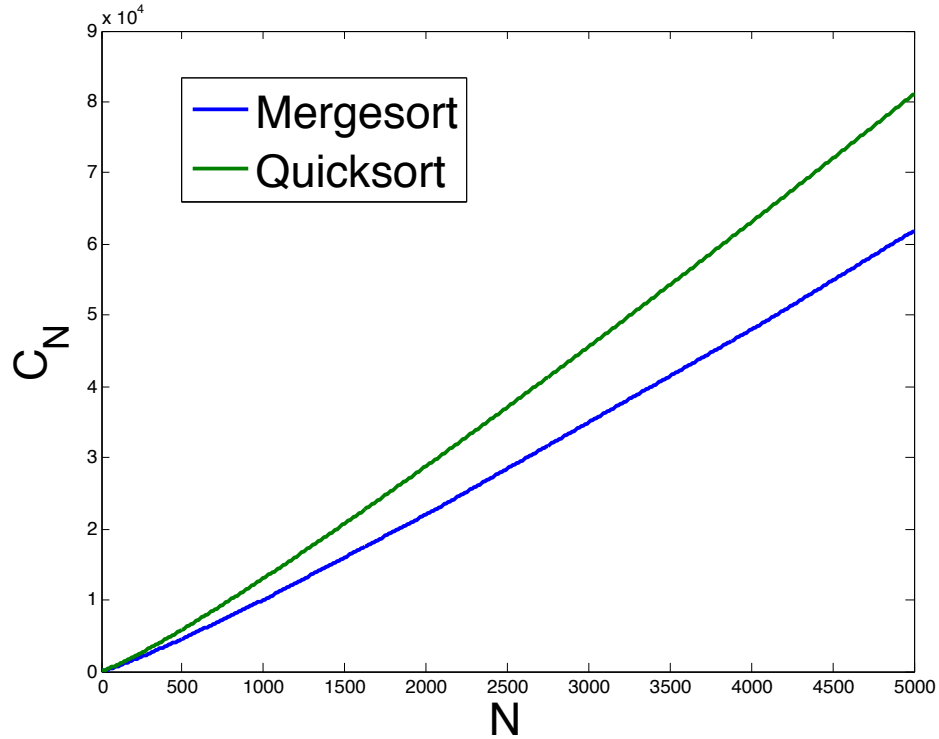
Une récurrence peut être :

- *linéaire* si f est une combinaison linéaire à coefficients *constants* ou *variables*.
  - $u_n = 3u_{n-1} + 4u_{n-2}$, $u_n = n * u_{n-2} + 1$
- *polynomiale* si f est un polynôme.
  - $b_n = \sum_{k=0}^{n-1} b_k b_{n-1-k}$
- *d'ordre k* si f dépend de $u_{n-1}, \ldots, u_{n-k}$.
- *complète* si f dépend de $u_{n-1}, \ldots, u_0$.
- *de type "diviser pour régner"* si f dépend de $u_{n/a}$, avec $a \in \mathbb{N}$ constant.
  - $u_n = 2u_{n/2}$
- *homogène* si f ne dépend que des u_i.
  - $u_n = 2u_{n/2} + u_{n/4}$
- *non homogène* si elle est de la forme $u_n = f(\{u_p | p < n\}) + g(n)$.
  - $u_n = 2u_{n-1} + n$

Calculer les valeurs d'une récurrence

On peut utiliser l'ordinateur pour calculer les valeurs d'une récurrence

Quicksort versus mergesort :



Avantages : ça s'applique à toutes les récurrences
Limitations : temps de calcul peut être élevé, comparaisons hasardeuses.

Remarque sur l'implémentation d'une récurrence

La manière d'implémenter la récurrence est importante

$$C_N = N + 1 + \frac{2}{N} \sum_{k=1}^{N} C_{k-1}$$

Implémentation récursive naïve

```
CQS(N)
1  if N == 0
2        return 0
3  else val = 0
4        for k = 1 to N
5              val = val + CQS(k − 1)
6        return N + 1 + 2 ∗ val/N
```

Nombre d'additions :

$$\begin{aligned} & C'_N = N + 2 + \textstyle\sum_{k=1}^{N} C'_{k-1} \\ \Leftrightarrow \quad & C'_N = 2C'_{N-1} + 1 \\ \Leftrightarrow \quad & C'_N = 2^N - 1 \end{aligned}$$

(avec $C'_0 = 0$)

Implémentation efficace
(par programmation dynamique)

```
CQS(N)
1  if N == 0
2        return 0
3  else sum = 0
4        for k = 1 to N
5              val = k + 1 + 2 ∗ sum/k
6              sum = sum + val
7        return val
```

Nombre d'additions :

$$C'_N = 3N$$

Techniques de résolution de récurrences

On souhaite obtenir une solution analytique à une récurrence.

Plusieurs approches possibles :

- "Deviner-et-vérifier"
- "Plug-and-chug" (telescopage)
- Arbres de récursion
- Equations caractéristique (linéaire)
- Master theorem et Akra-Bazzi (diviser-pour-régner)
- Changement de variable
- Fonctions génératrices (Chapitre 8)

Méthode “Deviner-et-Vérifier”

Principes :

1. Calculer les quelques premières valeurs de T_n ;
2. Deviner une solution analytique ;
3. Démontrer qu'elle est correcte, par exemple par induction.

Application :

- $T_n = 2T_{n-1} - 1$ (tours de Hanoï) :

n	1	2	3	4	5	6	$\cdots$
T_n	1	3	7	15	31	63	$\cdots$

  $\Rightarrow$ On devine $T_n = 2^n - 1$

- On doit démontrer (par induction) que la solution est correcte.

Preuve d'une solution par induction

Théorème : $T_n = 2^n - 1$ satisfait la récurrence :

$$\begin{aligned} T_1 &= 1 \\ T_n &= 2T_{n-1} + 1 \text{ pour } n \geq 2. \end{aligned}$$

Démonstration : Par induction sur n. $P(n) =$ " $T_n = 2^n - 1$".
Cas de base : $P(1)$ est vrai car $T_1 = 1 = 2^1 - 1$.
Cas inductif : Montrons que $T_n = 2^n - 1$ (pour $n \geq 2$) est vrai dès que $T_{n-1} = 2^{n-1} - 1$ est vrai :

$$\begin{aligned} T_n &= 2T_{n-1} + 1 \\ &= 2(2^{n-1} - 1) + 1 \\ &= 2^n - 1. \end{aligned}$$

□

Piège de l'induction

Si une solution exacte ne peut pas être trouvée, on peut vouloir trouver une borne supérieure sur la solution.

Essayons de montrer que $T_n \leq 2^n$ pour la récurrence $T_n = 2T_{n-1} + 1$.

Tentative de démonstration : Par induction sur n. $P(n) =$ " $T_n \leq 2^n$".
Cas de base : $P(1)$ est vrai car $T_1 = 1 \leq 2^1$.
Cas inductif : Supposons $T_{n-1} \leq 2^{n-1}$ et montrons que $T_n \leq 2^n$ $(n \geq 2)$:

$$\begin{aligned} T_n &= 2T_{n-1} + 1 \\ &\leq 2(2^{n-1}) + 1 \\ &\not\leq 2^n \end{aligned}$$

□

La démonstration ne fonctionne pas si on n'utilise pas la borne exacte plus forte (voir aussi le transparent 367).

Méthode “Plug-and-Chug” (force brute)

(aussi appelée téléscopage ou méthode des facteurs sommants)

1. “Plug” (appliquer l’équation récurrente) et “Chug” (simplifier)

$$\begin{aligned} T_n &= 1 + 2T_{n-1} \\ &= 1 + 2(1 + 2T_{n-2}) \\ &= 1 + 2 + 4T_{n-2} \\ &= 1 + 2 + 4(1 + 2T_{n-3}) \\ &= 1 + 2 + 4 + 8T_{n-3} \\ &= \dots \end{aligned}$$

Remarque : Il faut simplifier *avec modération*.

2. Identifier et vérifier un “pattern”
   - Identification :

$$T_n = 1 + 2 + 4 + \cdots + 2^{i-1} + 2^i T_{n-i}$$

   - Vérification en développant une étape supplémentaire :

$$\begin{aligned} T_n &= 1 + 2 + 4 + \cdots + 2^{i-1} + 2^i(1 + 2T_{n-(i+1)}) \\ &= 1 + 2 + 4 + \cdots + 2^{i-1} + 2^i + 2^{i+1} T_{n-(i+1)} \end{aligned}$$

3. Exprimer le $n^{\text{ème}}$ terme en fonction des termes précédents
   En posant $i = n - 1$, on obtient

$$\begin{aligned} T_n &= 1 + 2 + 4 + \cdots + 2^{n-2} + 2^{n-1} T_1 \\ &= 1 + 2 + 4 + \cdots + 2^{n-2} + 2^{n-1} \end{aligned}$$

4. Trouver une solution analytique pour le $n^{\text{ème}}$ terme

$$
\begin{aligned}
T_n &= 1 + 2 + 4 + \cdots + 2^{n-2} + 2^{n-1} \\
&= \sum_{i=0}^{n-1} 2^i \\
&= \frac{1 - 2^n}{1 - 2} \\
&= 2^n - 1
\end{aligned}
$$

Tri par fusion

Appliquons le “plug-and-chug” à la récurrence du tri par fusion *dans le cas où* $N = 2^n$:

$$\begin{aligned} C_1 &= 0 \\ C_N &= C_{\lceil N/2 \rceil} + C_{\lfloor N/2 \rfloor} + N - 1 = 2C_{N/2} + N - 1 \text{ pour } N > 1 \end{aligned}$$

- Pattern :

$$\begin{aligned} C_N &= 2^i C_{N/2^i} + (n - 2^{i-1}) + (n - 2^{i-2}) + \ldots + (n - 2^0) \\ &= 2^i C_{N/2^i} + i \cdot N - 2^i + 1 \end{aligned}$$

- En posant $i = n$ et en utilisant $n = \log_2 N$:

$$\begin{aligned} C_N &= 2^n C_{N/2^n} + n \cdot N - 2^n + 1 \\ &= NC_1 + N \log_2 N - N + 1 \\ &= N \log_2 N - N + 1 \end{aligned}$$

Récurrence linéaire d'ordre 1

Le plug-and-chug appliqué à une récurrence du type :

$$T_n = T_{n-1} + f(n)$$

donne directement :

$$T_n = T_0 + \sum_{k=1}^{n} f(n).$$

La récurrence n'est rien d'autre que la réécriture d'une somme (voir chapitre 6).

Si le coefficient de T_{n-1} n'est pas 1 :

$$T_n = g(n)\,T_{n-1} + f(n),$$

diviser gauche et droite par $h(n) = g(n) \cdot g(n-1) \cdot g(n-2) \cdots g(1)$ permet de se ramener à une récurrence de la forme :

$$\frac{T_n}{h(n)} = \frac{T_{n-1}}{h(n-1)} + \frac{f(n)}{h(n)} \Rightarrow T_n = h(n)\left(\frac{T_0}{h(0)} + \sum_{k=1}^{n} \frac{f(k)}{h(k)}\right)$$

Intéressant si le produit $h(n)$ a une forme simple

Récurrence linéaire d'ordre 1 : Exemples

Exemple 1 : $T_0 = 0,\ T_n = 2T_{n-1} + 2^n$ (pour $n > 0$)
En divisant gauche et droite par $h(n) = 2 \cdot 2 \cdots 2 = 2^n$, on obtient :

$$\frac{T_n}{2^n} = \frac{T_{n-1}}{2^{n-1}} + 1 \Rightarrow T_n = 2^n(0 + \sum_{k=1}^{n} 1) = n2^n$$

Exemple 2 : $T_0 = 0,\ T_n = (1 + \frac{1}{n})T_{n-1} + 2$ (pour $n > 0$) *(Quicksort)*
En divisant gauche et droite par :

$$h(n) = \frac{n+1}{n}\frac{n}{n-1}\cdots\frac{3}{2}\frac{2}{1} = n+1,$$

on obtient

$$\frac{T_n}{n+1} = \frac{T_{n-1}}{n} + \frac{2}{n+1} = \sum_{k=1}^{n}\frac{2}{n+1} \Rightarrow T_n = 2(n+1)H_n - 2n$$

Arbres de récursion

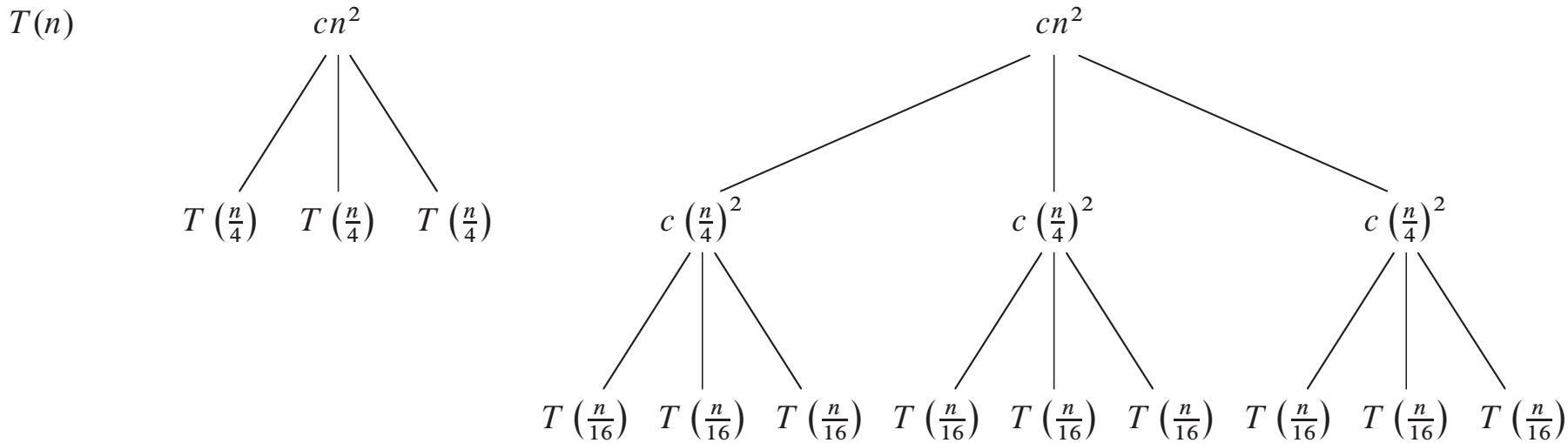
Approche graphique pour *deviner* une solution analytique (ou une borne asymptotique) à une récurrence.

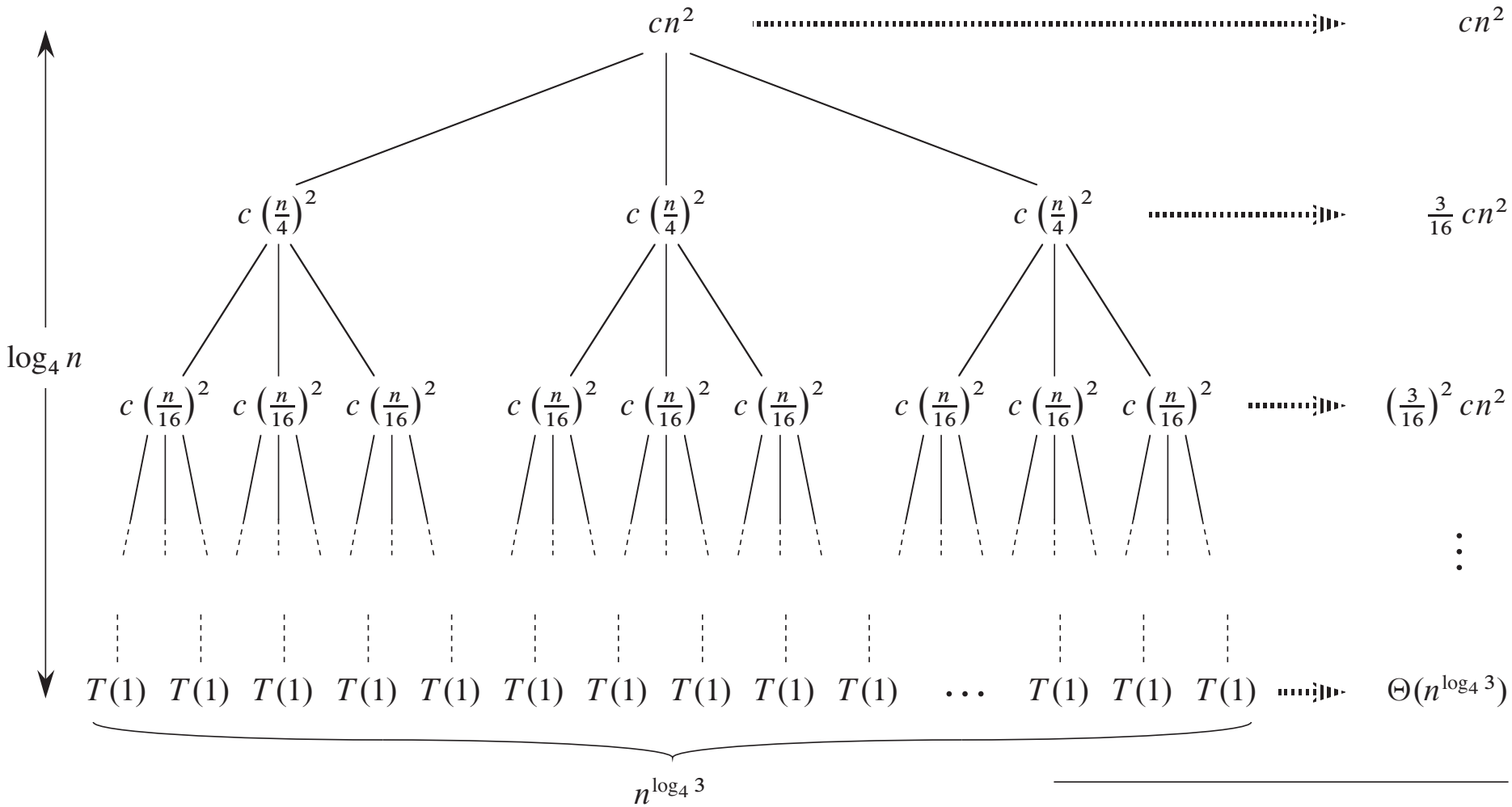
La solution devra toujours être démontrée ensuite (par induction).

Illustration sur la récurrence suivante :

- $T(1) = a$
- $T(n) = 3T(n/4) + cn^2$ (Pour $n > 1$)

(Introduction to algorithms, Cormen et al.)


$T(n)$
cn^2
$T\left(\frac{n}{4}\right)$ $T\left(\frac{n}{4}\right)$ $T\left(\frac{n}{4}\right)$
cn^2
$c\left(\frac{n}{4}\right)^2$ $c\left(\frac{n}{4}\right)^2$ $c\left(\frac{n}{4}\right)^2$
$T\left(\frac{n}{16}\right)$ $T\left(\frac{n}{16}\right)$ $T\left(\frac{n}{16}\right)$ $T\left(\frac{n}{16}\right)$ $T\left(\frac{n}{16}\right)$ $T\left(\frac{n}{16}\right)$ $T\left(\frac{n}{16}\right)$ $T\left(\frac{n}{16}\right)$ $T\left(\frac{n}{16}\right)$


cn^2
cn^2
$c\left(\frac{n}{4}\right)^2$
$c\left(\frac{n}{4}\right)^2$
$c\left(\frac{n}{4}\right)^2$
$\frac{3}{16}cn^2$
$\log_4 n$
$c\left(\frac{n}{16}\right)^2$
$\left(\frac{3}{16}\right)^2 cn^2$
$T(1)$
$\Theta(n^{\log_4 3})$
$n^{\log_4 3}$

- Le coût total est la somme du coût de chaque niveau de l'arbre :

$$\begin{aligned} T(n) &= cn^2 + \frac{3}{16}cn^2 + \ldots + \left(\frac{3}{16}\right)^{\log_4 n-1} cn^2 + an^{\log_4 3} \\ &= \sum_{i=0}^{\log_4 n-1} \left(\frac{3}{16}\right)^i cn^2 + an^{\log_4 3} \\ &< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + an^{\log_4 3} \\ &= \frac{1}{1-(3/16)} cn^2 + an^{\log_4 3} \\ &= O(n^2) \end{aligned}$$

(à vérifier par induction)

- Comme le coût de la racine est cn^2, on a aussi $T(n) = \Omega(n^2)$ et donc $T(n) = \Theta(n^2)$.

Preuve d'une borne supérieure

Théorème : Soit la récurrence :

- $T(1) = a$,
- $T(n) = 3T(n/4) + cn^2$ (Pour $n > 1$).

On a $T(n) = O(n^2)$.

Préparation de la démonstration : Soit $P(n) =$ "$T(n) \leq dn^2$". L'idée est de trouver un d tel que $P(n)$ peut se montrer par induction forte.
Cas de base : $P(1)$ est vrai si $a \leq d \cdot 1^2 = d$.
Cas inductif : Supposons $P(1), P(4), \ldots, P(n/4)$ vérifiés et trouvons une contrainte sur d telle que $P(n)$ le soit aussi :

$$\begin{aligned} T(n) &\leq 3T(n/4) + cn^2 \\ &\leq 3d(n/4)^2 + cn^2 \\ &= \frac{3}{16}dn^2 + cn^2 \\ &\leq dn^2, \end{aligned}$$

La dernière étape est valide dès que $d \geq (16/13)c$. Le théorème est donc vérifié pour autant que $d \geq \max\{(16/13)c, a\}$.

Preuve d'une borne supérieure

Ré-écriture de la preuve :

Démonstration : Soit une constante d telle que $d \geq \max\{(16/13)c, a\}$.
Montrons par induction forte que $P(n) =$ "$T(n) \leq dn^2$" est vrai pour tout $n \geq 1$ (qui implique $T(n) = O(n)$).
Cas de base : $P(1)$ est vrai puisque $d \geq \max\{(16/13)c, a\} \geq a$.
Cas inductif : Supposons $P(1), P(4), \ldots, P(n/4)$ vérifiés et montrons que $P(n)$ l'est aussi :

$$\begin{aligned} T(n) &\leq 3T(n/4) + cn^2 \\ &\leq 3d(n/4)^2 + cn^2 \\ &= \frac{3}{16}dn^2 + cn^2 \\ &\leq dn^2, \end{aligned}$$

où la dernière étape découle de $d \geq max((16/13)c, a)$.

□

Induction et notation asymptotique

Théorème faux : Soit la récurrence :

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

On a $T(n) = O(n)$.
(la solution correcte est $T(n) = \Theta(n \log(n))$)

Démonstration : (par induction forte)

- Soit $P(n) =$ "$T(n) = O(n)$".
- *Cas de base :* $P(1)$ est vrai car $T(1) = 1 = O(1)$.
- *Cas inductif :* Pour $n \geq 2$, supposons $P(1), P(2), \ldots, P(n-1)$. On a :

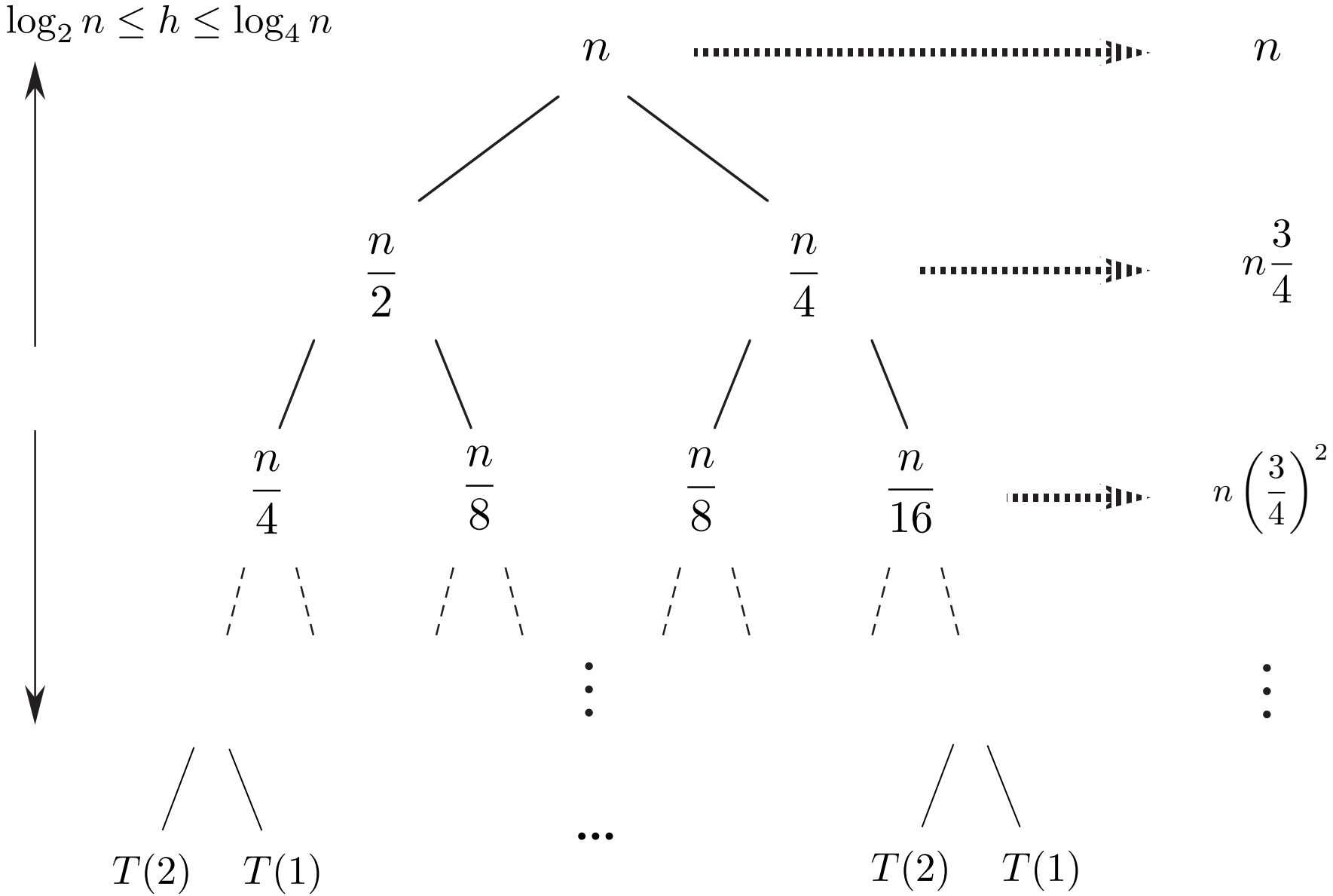
$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 2O(n/2) + n \\ &= O(n) \end{aligned}$$

Où est l'erreur ?

Autre exemple :

- $T(1) = 1$
- $T(2) = 2$
- $T(n) = T(n/2) + T(n/4) + n$ (pour $n > 2$)

(On suppose que n est toujours une puissance de 2)


$\log_2 n \le h \le \log_4 n$
n
n
$\frac{n}{2}$
$\frac{n}{4}$
$n\frac{3}{4}$
$\frac{n}{4}$
$\frac{n}{8}$
$\frac{n}{8}$
$\frac{n}{16}$
$n\left(\frac{3}{4}\right)^2$
$T(2)$
$T(1)$
$T(2)$
$T(1)$

- On déduit de l'arbre que

$$T(n) \leq \sum_{i=0}^{\infty} n \left(\frac{3}{4}\right)^i = n \frac{1}{1-\frac{3}{4}} = O(n)$$

- Vérification par induction forte qu'il existe un c tel que pour tout $n \geq n_0$, on a $T(n) \leq cn$

$$\begin{aligned} T(n) &= T(n/2) + T(n/4) + n \\ &\leq cn/2 + cn/4 + n \\ &= (c3/4 + 1)n \\ &\leq cn \end{aligned}$$

  $\Rightarrow$ ok pour tout $c > 4$

- Puisqu'on a aussi $T(n) = \Omega(n)$, on en déduit $T(n) = \Theta(n)$.

Synthèse

Trois approches empiriques pour trouver une solution analytique :

- “Deviner-et-vérifier”
- “Plug-and-Chug”
- Arbres de récursion

Ces approches sont génériques mais

- il n’est pas toujours aisé de trouver le pattern ;
- il faut pouvoir résoudre la somme obtenue ;
- la solution doit être vérifiée par induction.

On va voir des méthodes plus systématiques pour résoudre des récurrences particulières

- Récurrences linéaires d’ordre $k \geq 1$ à coefficients constants (solution analytique exacte)
- Récurrences “diviser-pour-régner” (borne asymptotique uniquement)

Récurrences linéaires

Définition : Une *récurrence linéaire homogène* est une récurrence de la forme

$$f(n) = a_1 f(n-1) + a_2 f(n-2) + \cdots + a_k f(n-k)$$

où $a_1, a_2, \ldots, a_k \in \mathbb{R}$ sont des constantes. La valeur $k \in \mathbb{N}_0$ est appelée l'*ordre* de la récurrence.

Définition : Une *récurrence linéaire* (générale) est une récurrence de la forme

$$f(n) = a_1 f(n-1) + a_2 f(n-2) + \cdots + a_k f(n-k) + g(n),$$

où g est une fonction ne dépendant pas de f.

Théorème : Si $f_1(n)$ et $f_2(n)$ sont solutions d'une récurrence linéaire homogène (sans tenir compte des conditions initiales), alors toute combinaison linéaire $c_1 f_1(n) + c_2 f_2(n)$ de $f_1(n)$ et $f_2(n)$ est également une solution pour tout $c_1, c_2 \in \mathbb{R}$.

Démonstration : On a $f_1(n) = \sum_{i=1}^{k} a_i f_1(n-i)$ et $f_2(n) = \sum_{i=1}^{k} a_i f_2(n-i)$.

Dès lors,

$$\begin{aligned} c_1 f_1(n) + c_2 f_2(n) &= c_1 \cdot \left(\sum_{i=1}^{k} a_i f_1(n-i) \right) + c_2 \cdot \left(\sum_{i=1}^{k} a_i f_2(n-i) \right) \\ &= \sum_{i=1}^{k} a_i (c_1 f_1(n-i) + c_2 f_2(n-i)). \end{aligned}$$

□

Exemple

Résolvons la récurrence du transparent 341 (Fibonacci) :

$$\begin{aligned} f(0) &= 0 \\ f(1) &= 1 \\ f(n) &= f(n-1) + f(n-2) \text{ pour } n > 1 \end{aligned}$$

- Une solution analytique pour une récurrence linéaire a souvent une forme exponentielle.
- On devine $f(n) = cx^n$ (c et x sont des paramètres à trouver).
-

$$\begin{aligned} & f(n) = f(n-1) + f(n-2) \\ \Rightarrow \quad & cx^n = cx^{n-1} + cx^{n-2} \\ \Rightarrow \quad & x^2 = x + 1 \\ \Rightarrow \quad & x = \frac{1 \pm \sqrt{5}}{2} \end{aligned}$$

- Les fonctions $c\left(\frac{1+\sqrt{5}}{2}\right)^n$ et $c\left(\frac{1-\sqrt{5}}{2}\right)^n$ sont des solutions de la récurrence (sans tenir compte des conditions initiales).
- Il en est de même pour toute combinaison linéaire de ces deux fonctions.
- On a donc $f(n) = c_1\left(\frac{1+\sqrt{5}}{2}\right)^n + c_2\left(\frac{1-\sqrt{5}}{2}\right)^n$.

- $f(0) = c_1 \left(\frac{1+\sqrt{5}}{2}\right)^0 + c_2 \left(\frac{1-\sqrt{5}}{2}\right)^0 = c_1 + c_2 = 0.$
- $f(1) = c_1 \left(\frac{1+\sqrt{5}}{2}\right)^1 + c_2 \left(\frac{1-\sqrt{5}}{2}\right)^1 = 1.$
- On obtient $c_1 = \frac{1}{\sqrt{5}}$ et $c_2 = \frac{-1}{\sqrt{5}}$.
- Finalement,

$$f(n) = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2}\right)^n.$$

Résolution des récurrences linéaires

Soit une récurrence de la forme

$$f(n) = a_1 f(n-1) + a_2 f(n-2) + \cdots + a_k f(n-k) + g(n)$$

et les conditions initiales $f(0) = b_0$, $f(1) = b_1$, etc.

Etape 1 : Trouver les racines de l'*équation caractéristique*

Définition : L'équation caractéristique est

$$x^k = a_1 x^{k-1} + a_2 x^{k-2} + \cdots + a_{k-1} x + a_k.$$

Remarque : Le terme $g(n)$ n'est pas pris en compte dans l'équation caractéristique.

Etape 2 : Trouver une *solution homogène*, sans tenir compte des conditions initiales

Il suffit d'additionner les termes suivants :

- Une racine non répétée r de l'équation caractéristique génère le terme

$$c_r r^n,$$

où c_r est une constante à déterminer plus tard.

- Une racine r avec multiplicité m de l'équation caractéristique génère les termes

$$c_{r_1} r^n, c_{r_2} n r^n, c_{r_3} n^2 r^n, \ldots, c_{r_m} n^{m-1} r^n,$$

où $c_{r_1}, c_{r_2}, \ldots, c_{r_m}$ sont des constantes à déterminer plus tard.

Etape 3 : Trouver une *solution particulière*, sans tenir compte des conditions initiales.

Une technique simple consiste à *deviner et vérifier* en essayant des solutions *ressemblant* à $g(n)$.

Exemples :

- Si $g(n)$ est un polynôme, essayer avec un polynôme de même degré, ensuite avec un polynôme de degré immédiatement supérieur, et ainsi de suite.
  Exemple : Si $g(n) = n$, essayer d'abord $f(n) = bn + c$, ensuite, $f(n) = an^2 + bn + c$, ...
- Si $g(n) = 3^n$, essayer d'abord $f(n) = c3^n$, ensuite $f(n) = bn3^n + c3^n$, $f(n) = an^2 3^n + bn3^n + c3^n$, ...

Remarque : On doit attribuer aux constantes $a, b, c, \ldots$ des valeurs satisfaisant l'équation récurrente.

Etape 4 : Former une *solution générale*, sans tenir compte des conditions initiales

Il suffit d'additionner la solution homogène et la solution particulière

Etape 5 : Déterminer les valeurs des constantes introduites à l'étape 2

- Pour chaque condition initiale, appliquer la solution générale à cette condition. On obtient une équation en fonction des constantes à déterminer.
- Résoudre le système formé par ces équations.

Exemple

Résolvons la récurrence du transparent 342 :

- $f(0) = 1$
- $f(n) = 8f(n-1) + 10^{n-1}$

Etape 1 : Trouver les racines de l'équation caractéristique

- L'équation caractéristique est $x = 8$.
- Sa seule racine est 8.

Etape 2 : Trouver une solution homogène, sans tenir compte des conditions initiales

La solution homogène est $f(n) = c8^n$.

Etape 3 : Trouver une solution particulière, sans tenir compte des conditions initiales.

- On devine que la solution est de la forme $d10^{n-1}$, où d est une constante.
- En substituant, on obtient

$$\begin{aligned} d \cdot 10^{n-1} &= 8 \cdot d \cdot 10^{n-2} + 10^{n-1} \\ 10 \cdot d &= 8 \cdot d + 10 \\ d &= 10/2 \end{aligned}$$

- On vérifie que $\frac{10}{2} \cdot 10^{n-1} = \frac{10^n}{2}$ est bien une solution particulière.

Etape 4 : Former une solution générale, sans tenir compte des conditions initiales

On obtient la solution générale

$$f(n) = c8^n + \frac{10^n}{2}.$$

Etape 5 : Déterminer les valeurs des constantes introduites à l'étape 2

$$\begin{aligned} f(0) = 1 &\Rightarrow c8^0 + \frac{10^0}{2} = 1 \\ &\Rightarrow c = \frac{1}{2}. \end{aligned}$$

Conclusion : $f(n) = \dfrac{8^n + 10^n}{2}$.

Récurrence générale “diviser-pour-régner”

Définition : Un récurrence “diviser-pour-régner” est une récurrence de la forme :

$$T_n = \sum_{i=1}^{k} a_i T(b_j n) + g(n),$$

où $a_1, \ldots, a_k$ sont des constantes positives, $b_1, \ldots, b_k$ sont des constantes comprises entre 0 et 1 et $g(n)$ est une fonction non négative.

Exemple : $k = 1$, $a_1 = 2$, $b_1 = 1/2$ et $g(n) = n - 1$ correspond au tri par fusion

Sous certaines conditions, il est possible de trouver des bornes asymptotiques sur les récurrences de ce type.

Un premier théorème

Théorème (Master theorem) : Soient deux constantes $a \geq 1$ et $b > 1$ et une fonction $f(n) = O(n^d)$ avec $d \geq 0$. La complexité asymptotique de la récurrence suivante :

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

est :

$$T(n) = \begin{cases} O(n^d) & \text{pour } d > \log_b a \\ O(n^d \log n) & \text{pour } d = \log_b a; \\ O(n^{\log_b a}) & \text{pour } d < \log_b a. \end{cases}$$

(Introduction to algorithms, Cormen et al.)

NB : les bornes sont valables quelles que soient les conditions initiales.

Exemple d'application

- Soit la récurrence suivante :

$$T(n) = 7T(n/2) + O(n^2).$$

(Méthode de Strassen pour la multiplication de matrice)

- $T(n)$ satisfait aux conditions du théorème avec $a = 7$, $b = 2$, et $d = 2$.
- $\log_b a = \log_2 7 = 2.807... \Rightarrow d = 2 < \log_b a = 2.807...$
- Par le troisième cas du théorème, on a :

$$T(n) = O(n^{\log_b a}) = O(n^{2.807...}).$$

Un second théorème plus général

Forme générale d'une récurrence "Diviser pour régner" :

$$T(x) = \begin{cases} \text{est défini} & \text{pour } 0 \leq x \leq x_0 \\ \displaystyle\sum_{i=1}^{k} a_i T(b_i x) + g(x) & \text{pour } x > x_0 \end{cases}$$

avec

- $a_1, a_2, \ldots, a_k > 0$,
- $b_1, b_2, \ldots, b_k \in [0, 1[$,
- x_0 suffisamment grand,
- $|g'(x)| = O(x^c)$ pour un $c \in \mathbb{N}$.

Théorème (Akra-Bazzi) :

$$T(x) = \Theta\left(x^p\left(1 + \int_1^x \frac{g(u)}{u^{p+1}}\,\mathrm{d}u\right)\right)$$

où

- p satisfait l'équation $\sum_{i=1}^{k} a_i b_i^p = 1$

Exemple d'application

- Soit la récurrence "diviser pour régner" suivante :

$$T(x) = 2T(x/2) + 8/9\,T(3x/4) + x^2$$

- On a bien $|g'(x)| = |2x| = O(x)$
- Trouvons p satisfaisant :

$$2(\frac{1}{2})^p + \frac{8}{9}(\frac{3}{4})^p = 1$$

  $\Rightarrow p = 2$
- Par application du théorème, on obtient :

$$\begin{aligned} T(x) &= \Theta\left(x^2\left(1 + \int_1^x \frac{u^2}{u^3}du\right)\right) \\ &= \Theta(x^2(1 + \log x)) \\ &= \Theta(x^2 \log x) \end{aligned}$$

Changement de variables

Un changement de variables permet parfois de transformer une récurrence “diviser pour régner” en une récurrence linéaire.

- Soit la récurrence du transparent 241 :

$$T(n) = 7T(n/2) + O(n^2)$$

- En posant : $n = 2^m$ et $S(m) = T(2^m)$, on obtient :

$$S(m) = T(2^m) = 7T(2^{m-1}) + O((2^m)^2) = 7S(m-1) + O(4^m)$$

Changement de variables

Un changement de variable permet de résoudre des récurrences qui semblent a priori complexes.

- Considérons la récurrence suivante :

$$T(n) = 2T(\sqrt{n}) + \log n$$

- Posons $m = \log n$. On a :

$$T(2^m) = 2T(2^{m/2}) + m.$$

- Soit $S(m) = T(2^m)$. On a :

$$S(m) = 2S(m/2) + m \Rightarrow S(m) = O(m \log m).$$

- Finalement :

$$T(n) = T(2^m) = S(m) = O(m \log m) = O(\log n \log \log n).$$

Résumé

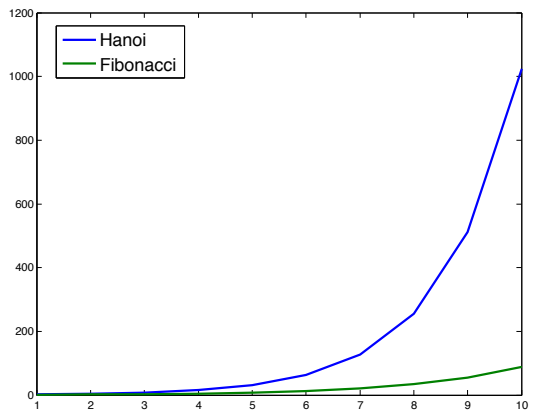
- Outils de résolution d'équations récurrentes :
  - Méthodes génériques : "Deviner-et-Vérifier", "Plug-and-Chug", arbres de récursion
  - Récurrences linéaires d'ordre k (à coefficients constants)
  - Récurrences "Diviser pour régner" : théorème "Master", théorème d'Akra-Bazzi

  Les deux dernières sont les plus systématiques.
- Le plus dur reste de traduire un problème réel en une équation récurrente.
- Exemple : Soit un type de plante qui vit éternellement, mais qui peut seulement se reproduire la première année. A quelle vitesse la population croît-elle ?

Comparaisons de récurrences : linéaires versus “d-p-r”

	Récurrence	Solution
Tours de Hanoï	$T_n = 2T_{n-1} + 1$	$T_n \sim 2^n$
Tours de Hanoï 2	$T_n = 2T_{n-1} + n$	$T_n \sim 2 \cdot 2^n$
Algo rapide	$T_n = 2T_{n/2} + 1$	$T_n \sim n$
Tri par fusion	$T_n = 2T_{n/2} + n - 1$	$T_n \sim n \log n$
Fibonacci	$T_n = T_{n-1} + T_{n-2}$	$T_n \sim (1.618\ldots)^{n+1}/\sqrt{5}$

- Récurrences “Diviser pour régner” généralement polynomiales
- Récurrences linéaires généralement exponentielles
- Générer des sous-problèmes petits est beaucoup plus important que de réduire la complexité du terme non homogène
  - Tri par fusion et Fibanocci sont exponentiellement plus rapides que les tours de Hanoï



Comparaisons de récurrences : nombre de sous-problèmes

Récurrences linéaires :

$$T_n = 2T_{n-1} + 1 \quad \Rightarrow \quad T_n = \Theta(2^n)$$
$$T_n = 3T_{n-1} + 1 \quad \Rightarrow \quad T_n = \Theta(3^n)$$

Augmentation exponentielle des temps de calcul quand on passe de 2 à 3 sous-problèmes.

Récurrence “diviser-pour-régner” :

$$\begin{aligned} T_1 &= 0 \\ T_n &= aT_{n/2} + n - 1 \end{aligned}$$

Par le master théorème, on a :

$$T_n = \begin{cases} \Theta(n) & \text{pour } a < 2 \\ \Theta(n \log_2 n) & \text{pour } a = 2 \\ \Theta(n^{\log_2 a}) & \text{pour } a > 2. \end{cases}$$

La solution est complètement différente entre $a = 1.99$ et $a = 2.01$.