Université Toulouse III – Paul sabatier
L2 Informatique

Complexité des algorithmes

Semestre 3

Toulouse, le 2 décembre 2012

Modalité de Contrôle de connaissance

1. Devoir Maison à rendre avant 10h le 21 décembre. (10%)
2. Contrôle de continue sous forme de QCM lundi 3 décembre de 8h15 à 9h45 (20%)
3. Contrôle terminal vendredi 21 décembre de 10h à 12h (70%)

Table des matières

Chapitre 1

Introduction

Sommaire

1.1 Complexité

On cherche à estimer le temps de calcul d'un algorithme A en fonction d'un paramètre n. Pour avoir une mesure indépendante de la machine, on identifie le temps de calcul avec le nombre d'instructions exécutées.

Ex Le paramètre n pourrait être la taille d'un tableau, par exemple.

Soit D_i l'ensemble des données possibles telle que $n = i$. Pour $d \in D_i$ on notera $T(A, d)$ le nombre d'instructions exécutée pendant l'exécution de $A(d)$.
On notera $\text{prob}(d|i)$ la probabilité que les données soit d étant donné qu'elles sont de taille i.

1.1.1 La complexité temporelle maximale

La complexité temporelle maximale[1] d'un algorithme A :

$$T_{\max}(i) = \max_{d \in D_i}\{T(A, d)\}$$

1.1.2 La complexité temporelle moyenne

La complexité temporelle moyenne[2] d'un algorithme A :

$$T_{\text{moy}} = \sum_{d \in D_i} \text{prob}(d|i) \times T(A, d)$$

1. Complexité dans le pire des cas
2. Complexité dans le cas moyen

R Pour pouvoir calculer T_{moy}, il faut connaître la distribution des données, ce qui n'est pas toujours évident (par exemple en traitement d'image)

1.1.3 La complexité temporelle minimale

La complexité temporelle minimale [3] d'un algorithme A :

$$T_{\min}(i) = \min_{d \in D_i}\{T(A, d)\}$$

R Peu utilisé, sauf pour prouver qu'un algorithme est mauvais. Si la complexité temporelle minimale est mauvaise même dans le meilleur des cas, alors l'algorithme n'est pas bon.

1.1.4 Comparaison de complexités en fonction de la machine

Complexité	**Nombre d'instructions pouvant executer la machine**	
	1 000 000	1 000 000 000 000
n	1 000 000	1 000 000 000 000
$n \log_2 n$	64 000	32 000 000 000
n^2	1 000	1 000 000
n^3	100	10 000
2^n	20	40

1.2 Complexité asymptotique

Pour comparer des algorithmes, on ne s'intéresse qu'à leur comportements pour n grand. On cherche une mesure de complexité qui soit indépendante du langage de programmation et de la vitesse de la machine.
⇒ On ne doit pas perdre en compte des facteurs constants.
⇒ Ordre de grandeur

1.2.1 La complexité asymptotique

La complexité asymptotique [4] est l'ordre de grandeur de sa limite lorsque $n \to \infty$

1.2.2 Notation

Soient T, f des fonctions positives ou nulles. Rotations de grandeur de fonction asymptotiques.

3. Complexité dans le meilleur des cas
4. Que ce soit maximale, moyenne ou minimale

Grand O $T = O(f)$ si $\exists c \in \mathbb{R}^{>0}$ et $n_0 \in \mathbb{N}$ tels que $\forall n \geq n_0$, $T(n) \leq cf(n)$.

Grand Oméga $T = \Omega(f)$ si $\exists c \in \mathbb{R}^{>0}$ et $n_0 \in \mathbb{N}$ tels que $i\forall n \geq n_0$, $T(n) \geq cf(n)$

Petit O $T = o(f)$ si $\frac{T(n)}{f(n)} \to O$ lorsque $n \to \infty$.

R T est négligeable devant f

Ex

1. $2n^2 + 5n + 10 = O(n^2)$
   Dans la définition $n_0 = 5$,$c = 4$:
   $\forall n \geq 5,\ 2n^2 + 5n + 10 \leq 4n^2$
2. $2n^2 + 5n + 10 = \Omega(n^2)$
   Dans la définition, $n_0 = 1$, $c = 2$
   $\forall n \geq 1,\ 2n^2 + 5n + 10 \geq 2n^2 \cdots$
   Donc $2n^2 + 5n + 10 = \Theta(n^2)$
3. $\frac{1}{5} + n = O(n \log_2 n)$ $(n_0 = 2,\ c = 2)$
4. $\frac{1}{5} n \log_2 n + n = \Omega(n \log n)$ $(n_0 = 1, c = \frac{1}{5})$
5. $\forall k \geq 0$, $n^k = O(n^{k+1})$ mais $n^k \neq \Omega(n^{k+1})$
6. $\forall a, b > 1, \log_a n = \Theta(\log_b n)$ car $\log_a n = \frac{\log_b n}{\log_b a}$ et $\log_b a$ est une constante.
   $\Rightarrow$ On a pas besoin de préciser la base de logarithme dans une complexité asymptotique
7. $2n^2 + 5n + 10 = 2n^2 + 0(n^2)$
8. Pour toute constante $c > 0$, $C = \Theta(1)$
9. $2^n = o(3^n)$

R

1. O et Ω sont des pré-ordres [a] :
   $f = O(f)$ et $f = O(g)$ et $g = O(h)) \Rightarrow f = O(h)$
2. Θ est une relation d'équivalence [b] : $f = \Theta(g) \Leftrightarrow g = \Theta(f)$

a. Relations réflexives et transitives
b. relation réflexives, symétrique et transitive

Proposition

$$\text{Si } \lim_{n\to\infty} \frac{f(n)}{g(n)} = a > 0 \text{ Alors } f = \Theta(g)$$

R La réciproque est fausse

Notation

$$f \sim g \Rightarrow \lim_{n \to \infty} \frac{f(n)}{g(n)} = 1$$

Ex $(3n+1)^3 \sim 27n^3$

1.3 Exemple de complexités d'algorithmes

1.3.1 Le tri à bulles

$$\begin{aligned} T_{\text{min}}(n) &= \Theta(n) \text{ Si le tableau est est déjà trié} \\ T_{\text{max}}(n) &= \Theta(n^2) \text{ Si le tableau est trié en ordre décroissant} \\ T_{\text{moy}}(n) &= T_{\text{max}}(n) = \Theta(n^2) \end{aligned}$$

1.3.2 Tri par fusion

$$T_{\text{min}}(n) = T_{\text{max}}(n) = T_{\text{moy}}(n) = \Theta(n \log n)$$

1.3.3 Tri rapide

$$\begin{aligned} T_{\text{min}}(n) = T_{\text{moy}}(n) &= \Theta(n \log n) \\ T_{\text{max}}(n) &= \Theta(n^2) \end{aligned}$$

1.4 Comportement symptotique de fonctions usuelles

Il y a quatre groupes importants de fonction positives croissantes.

Logarithmiques $(\log n)^\sigma$ (où $\sigma > 0$), $\log \log n, \ldots$

Polynomiales n^γ (où $\gamma > 0$), $n^\gamma (\log n)^\gamma$(où $\gamma > 0$)

Exponentielles $2^{\alpha n^\beta}$ (où $\alpha > 0$ et $0 < \beta \leq 1$), par exemple 2^n, 4^n, $2^{\sqrt{n}}$

Supra exponentielles $n!$, n^n, 2^{n^2}, $\ldots$

R Il existe des fonctions intermédiaires(par exemple $n^{\log_2 n}$) mais ces fonctions se rencontrent très rarement dans l'analyse de complexité d'algorithmes

$$\begin{aligned}\lim_{n\to\infty}\frac{n^b}{a^n} = O \text{ Pour toutes constantes} a, b \text{ avec } a > 1)\\ n^b = o(a^n)\\ \lim_{n\to\infty}\frac{(\log_2 n)^\sigma}{n^\sigma} = O\\ \Rightarrow (\log_2 n)^\sigma = o(n^\sigma)\end{aligned}$$

1.4.1 La formule de Stisling

$$\begin{aligned}n! \sim \sqrt{2\pi n}(\frac{n}{e})^n\\ \Rightarrow n! = o(n^n) \text{ et } n! = \Omega(2^n)\end{aligned}$$

On peut aussi en déduire :

$$\log(n!) \sim n\log n$$

Chapitre 2

Complexité des boucles

Sommaire

2.1 Complexité de boucles “pour”

```
pour i:= 1 a n faire
  -- Corps de la boucle
fin pour;
```

Notions I_i la i^e itération (les instructions executées lors du i^e passage dans la boucle) et $T(I_i)$ sa complexité temporelle. :

Par exemple, $T_{\text{moy}}(n) = T\max(n) = \Theta(n)$ si $T(I_i)$ constant et $= \Theta(n^2)$ si $T(I_i) = an + b$ (boucle imbriquée.

2.1.1 Exemple

Calculer $A = BC$, le produit de 2 matrics. Rappel :

$$a_{ik} = \sum_{j=1}^{n} = b_{ij}C_{ji}$$

```
pour i = 1 a n faire
  pour k = 1 a n faire
      aik  0
      pour j = 1 a n faire
        aik = aik + bij * cjk;
      fin pour;
    fin pour;
  fin pour;
fin pour;
```

$$T_{\text{moy}}(n) = T_{\max}(n) = \sum_{i=1}^{n}\sum_{k=1}^{n}(1+n) = \Theta(n^3)$$

2.2 Complexité de boucles "tant que"

```
tantque C faire
  -- Corps de la boucle
fin tantque;
```

$$T_{\text{moy}} = 1 + \sum_{i=1}^{\infty} \text{Prob}$$

On ajoute 1 pour le test de la condition C lorsque C = faux.

Soit E_i l'événement C = Vrai au début de i_i
Si $\forall i, j E_i, E_j$ sont indépendantes et prob$(E_i) = p < 1$, où p est une constante, alors prob(on exécute I_i) = prob$(E_1 \cdots E_i) = p^i$ d'où

$$T_{\text{moy}}(n) = 1 + \sum_{i=1}^{\infty} p^i * T(I_i)$$

Si $T(I_i)$ est constante, alors

$$T_{\text{moy}}(n) = \Theta(1 + \frac{p}{1-p}) = \Theta(\frac{1}{1-p}) = \Theta(1)$$

2.2.1 Exemple

Comparaison de 2 suites $\{A_i\}, \{b_i\}$.

```
i := 1;
tantque (ai = bi et i <= n) faire
  i := i + 1;
fin tantque;
```

$T_{\text{moy}}(n) = \Theta(1)$ si les suites sont indépendantes et aléatoires.

2.3 Approximation asymptotique de sommes partielles

Exemples de sommes partielles

$$\sum_{i=1}^{n} \frac{1}{i} \quad \sum_{i=1}^{n} i^k \quad \sum_{i=1}^{n} \log_2 i$$

2.3.1 Principe de la méthode

Pour calculer une approximation asymptotique de $\sum_{i=1}^{n} f(i)$ où f est une fonction monotone on l'encadre par $\int f(n)du$.

Proposition Si f est décroissante, alors

$$\int_{p}^{n+1} f(u)du \leq \sum_{i=p}^{n} f(i) \leq \int_{p-1}^{n} f(u)du$$

Ex f(u) $= \frac{1}{u}.H_n = \sum_{i=1}^{n} \frac{1}{i}$ est la série harmonique. On ne peut intégrer $\frac{1}{u}$ qu'à partir de 1 donc on choisit $p = 2$.

$$\int_{2}^{n+1} \frac{1}{u} du \leq H_n - 1 \leq \int_{1}^{n} \frac{1}{u} du$$

$$\begin{array}{rcl} [\log_e u]_2^{n+1} & \leq & H_n - 1 \leq [\log_e u]_1^n \\ \log_e(n+1) - \log_e 2 & \leq & H_n - 1 \leq \log_e n - \log_e 1 \\ \log_e n - \log_e 2 + 1 & < & H_n \leq (\log_e n) + 1 \end{array}$$

Donc $H_n = \Theta(\log n)$

2.3.2 Application

Étude de complexité d'un algorithme de génération d'une permutation aléatoire des entiers $1, 2, \cdots, n$ dans un tableau `perm`

R Il existe un algorithme de complexité $\Theta(N)$ pour ce problème : pour chaque $i \in \{1, 2, \cdots, n\}$, échanger `perm`[i] et `perm`[`random(i)`].

```
pour i = 1 a n faire
  vu[i] = faux;
fin pour;
pour i = 1 a n faire
  x = random(n);
  tantque vu[x] faire
    x = random(n);
  fin tantque;
  perm[i] = x;
  vu[x] = vrai;
fin pour;
```

Listing 2.1 – Génération d'une permutation aléatoire

R $T_{\max} = \infty$ car il n'y a aucune garantie de terminaison. C'est un exemple d'algorithme de type *Las Vegas* la probabilité de non terminaison est nulle.

On suppose que la complexité de `perm`(n) est $\Theta(1)$.

Pour i, n fixe, à chaque itération de la boucle "tantque", la probabilité de rentrer dans la boucle est une constante pour $p = \frac{i-1}{n}$ et $p < 1$ pour $1 \leq i \leq n$.

Par l'analyse de la complexité d'une boucle "tantque" (section 2.2), la complexité moyenne de la boucle "tantque" est $\Theta(\frac{1}{1-p}$ donc

$$\begin{aligned} T_{\text{moy}} &= \Theta(\sum_{i=1}^{n} \frac{1}{1-\frac{i-1}{n}}) \\ &= \Theta(\sum_{i=1}^{n} \frac{n}{n-(i-1)}) \\ &= \Theta(n \sum_{k=1}^{n} \frac{1}{k}) \\ &= \Theta(nH_n) = \Theta(n \log n) \text{ car } H_n = \Theta(\log n) \end{aligned}$$

2.4 Analyse de cas particuliers de boucles

Parfois, il est possible de trouver un majorant de la complexité d'un algorithme en identifiant une variable monotone croissante dont la valeur est majorée.

2.4.1 Algorithme gourmand pour trouver une semengation optimale

2.4.1.1 Problème

Décomposer une suite d'entiers $A_1 A_2 \cdots A_n$ en un nombre minimum de segments tels que les valeurs dans un même segment ne différent que par au plus k.

Ex $k = 1$, $A = 1\ 1\ 1\ 2\ 3\ 4\ 4\ 4\ 4\ 3\ 3\ 1\ 1\ 1\ 2\ 2$

Cet exemple possède au moins deux segmentations possibles

111	23	44433	11122	4 segments.
1112	344433	11122		3 segments.

Application Nettoyage de signal, compactage de données (avec perte d'informations)

Algorithme gourmand

1. Trouver le plus long préfixe $A_1 \cdots A_{i_1}$, de la suite $A_1 \cdots A_n$ telle que que $\forall i, j \in \{1, \cdots, i_1\}$, $|A_i - A_j| \leq k$
2. Appel récursif du même algorithme sur la suite $A_{i_1+1} \cdots A_n$

2.4.1.2 Démonstration que l'algorithme trouve toujours une segmentation optimale

Supposons que l'algorithme gourmand trouve une segmentation σ dont les segments se terminent aux positions $i_1, i_2, \cdots, i_\sigma$, mais qu'il existe une segmentation optimale σ_opt dont les segments se terminent aux positions $j_1, j_2, \cdots, j_t$ avec $t < r$.

Soient $i_0 = j_0 = 1$. Nous avons $i_t < i_r = n = j_t$.

Soit m le plus petit indice tel que $i_m < j_m$. Donc $i_{m-1} \geq j_{m-1}$. Un tel indice existe car $i_0 = j_0$ et $i_t < j_t$

Par définition de la segmentation «gourmande», σ, il y a une valeur j dans le segment S_m telle que $|y - x| > k$. Mais dans ce cas, σ_{opt} n'est pas une segmentation valide. Cette contradiction montre que la segmentation gourmande est toujours optimale.

```
i  = 1;
m  = 0;
i0 = 1;
tantque (i <= n) faire
  m = m + 1; --On cherche le segment Sm
  max = min = A[i] -- max et min sont les valeurs max et min de Sm
  i = i + 1;
  tantque (i <= n et A[i]-min <= k et max-A[I] <= k) faire
    si A[i] > max alors
      max = A[i];
    fin si;
    si A[i] < min alors
      min = A[i];
    fin si;

    i = i + 1;
  fin tantque
  im = i-1; --im = fin du segmetn de Sm
fin tantque;
```

Chaque itération des deux boucles `tantque` incrémente i. Puisque $i \leq n$, on peut en déduire $T_{\max}(n) = \Theta(n)$ malgré la présence de deux boucles imbriquées.

Chapitre 3

Complexité d'algorithmes définis par récurrence

3.1 Exemple introductif : Tri fusion

Étant donné un tableau T, on note T[i :j] le sous tableau de T qui va de la case i à la case j. L'algorithme de tri fusion utilise une procédure `fusion(T,i,j,k)`. On suppose que les deux sous tableaux T[i :j] et T[j+1 :k] sont déjà triés. En temps $\Theta(n)$, où $n = k - i + 1$, la procédure fusion produit le sous tableau T[i :k] trié à partir de la fusion de ces deux tableaux.

```
procedure triFusion(T,i,k) -- Tri du tableau T[i:k]
debut
  si k-i+1 > 1 alors
    j = (k+1)/2;
    triFusion(T,i,j);
    triFusion(T,j+1,k);
    fusion(T,i,j,k);
  fin si;
fin
```

Listing 3.1 – Algorithme du tri fusion

3.2 Méthode naïve d'analyse de complexité

Soit un temps maximal d'exécution de tri fusion sur un tableau de longueur n.

D'après l'algorithme, on a

$$U_n = U_{\frac{n}{2}} + U_{\frac{n}{2}}\Theta(n)$$

et $u_1 = 0$

Pour simplifier la récurrence on suppose que n est pair, et donc $U_n = 2U_{\frac{n}{2}} + \Theta(n)$

La méthode naïve consiste à deviner la solution, ici on devine $U_n \leq cn\log_2 n$. On suppose $U_{\frac{n}{2}} \leq C\frac{n}{2}\log_2\frac{n}{2}$ et on essaye d'en déduire $U_n \leq cn\log_2 n$

$$\begin{aligned} U_n = 2U_{\frac{n}{2}} + cn \quad &\leq \quad 2c\frac{n}{2}\log_2\frac{n}{2} + cn \\ &= cn(\log_2 n - 1) + cn = cn\log_2 n \end{aligned}$$

Puisque $u_1 = 0 \leq c1\log_2 1$, on en déduit $\forall n, i_n \leq cn\log_2 n$

3.2.1 Résumé de la méthode naïve

Pour une équation récurrente $u_n = f_n(U_{n-1}, \cdots, u_1)$ où f est une fonction monotone croissante

1. On devine une fonction g
2. On suppose que $\forall n < 1$ on a $U_n \leq g(m)$
3. On montre $U_n = f_n(U_{n-1}, \cdots, u_1 \leq f_n(g(n-1), \cdots, g(1)) \leq g(n)$
4. On conclut par récurrence que $\forall n$ on a $U_n \leq g(n)$

3.2.2 Exemples d'application

On commence par une **mauvaise** utilisation. Soit l'équation $U_n = 2U_{\frac{n}{2}}$. L'intuition $U_n \leq kn$ n'est pas correcte.

En effet, en remplaçant on obtient :

$$\begin{aligned} n_n &= 2U_{\frac{n}{2}} + 1 \\ &= 2k\frac{n}{2} + 1 \\ &= kn + 1 \end{aligned}$$

La bonne intuition est $u_n \leq kn - b$. En remplaçant on obtient :

$$\begin{aligned} u_n = 2U_{\frac{n}{2}} + 1 & \\ &\leq 2(k\frac{n}{2} - b) + 1 = kn - 2b + 1 \\ &\leq kn - b \text{ Si } b \geq 1 \end{aligned}$$

3.2.3 Réduction à des formes simples

Lors de l'analyse d'algorithmes récursifs, on rencontre souvent des équations récurrentes de la forme

$$u_n = aU_{\frac{n}{2}} + b,$$

où a et b sont des constantes. Par exemple le tri fusion.

Pour convertir ce type de récurrence en une forme affine $u'_n = a'u'_{n-1} + b'$, on pose

$$v_k = U_{2^k}$$

Autrement dit, on étudiera la suite $\{u_n\}_{n\geq 0}$ uniquement sur les puissances de 2.
Par exemple, pour le tri fusion, en remplaçant n par 2^k,

$$\begin{aligned} U_{2^k} &= 2U_{\frac{2^k}{2}} + C2^k \\ \text{donc } V_k &= 2v_{k-1} + c2^k \end{aligned}$$

3.3 Équation récurrentes linéaires

Définition Une équation récurrente linéaire à coefficients constants d'ordre k est une équation de la forme

$$\begin{cases} u_1 & = & C_i(O \leq i \leq k-1) & \text{CONDITIONS INITIALES (CI)} \\ u_n & = & \sum_i^k = 1 a_i u_{n-i} + g(n) & \text{EQUATION GÉNÉRALE} \end{cases}$$

Une équation est **homogène** si $\forall n g(n) = 0$. La solution générale est une suite satisfaisant uniquement l'équation générale. Une solution particulière est une solution générale satisfaisant aussi des conditions initiales.

3.3.1 Équations récurrentes linéaires homogènes d'ordre 1

Proposition La solution particulière de l'équation :

$$\begin{cases} u_0 & = & c \\ u_n & = & a u_{n-1} \end{cases}$$

est $u_n = Ca^n$ (c'est une suite géométrique)

3.3.2 Équations récurrentes linéaires non-homogènes d'ordre 1

On ne sait traiter facilement que les équations dans lesquelles le second membre $g(n)$ est un polynôme ou une exponentielle. Pour cela, on «dérive» l'équation pour faire baisser le degré du polynôme jusqu'à arriver à 0.

Ex Le tri fusion

On a une équation qui n'est pas homogène :

$$V_n = 2V_{n-1} + C2^n$$

Donc, au rang $n+1$, on a aussi

$$V_{n+1} = 2V_n + C \times 2^{n+1}$$

Pour éliminer la partie non-homogène, on enlève 2 fois la première équation à la seconde.

$$\begin{aligned} V_{n+1} - 2V_n &= 2V_n - 4V_{n-1} \\ V_{n+1} &= 4V_n - 4V_{n-1} \end{aligned}$$

3.3.3 Recherche d'une solution générale pour les équations récurrentes linéaires homogènes d'ordre 2

Une équation récurrente homogène d'ordre 2 est de la forme

$$\begin{cases} u_0 & = & C_0 \\ u_1 & = & C_1 \\ u_n & = & a_1 u_{n-1} + a_2 u_{n-2} \end{cases}$$

On peut obtenir ce type d'équation indirectement lorsque l'on a réduit une équation d'ordre 1 à une équation homogène d'ordre 2.

Ex L'équation récurrente linéaire homogène d'ordre 2 de Fibonacci

$$\begin{cases} U_0 & = & 1 \\ U_1 & = & 1 \\ U_n & = & U_{n-1} + U_{n-2} \end{cases}$$

On résout ces équations d'ordre 2 comme des équations d'ordre 1 : On cherche une solution générale de la forme λr^n. Une telle solution vérifie, pour le cas de la suite de Fibonacci : $\forall n \geq 2$, $\lambda r^n = \lambda r^{n-1} + \lambda r^{n-2}$
Soit, en divisant par λr^{n-2}

$$r^2 = r + 1$$

Autrement dit, r est une racine du polynôme $P(x) = x^3 - x - 1$.

Définition Le polynôme caractéristique d'une équation récurrente homogène d'ordre k

$$V_{n+k} + a_1 V_{n+k+1} + \cdots + a_k V_n = 0$$

est le polynôme $P(x) = x^k + a_1 x^{k-1} + \ldots + a_{k-1} + a_k$

Théorème Si r est une racine du polynôme caractéristique d'une équation récurrente linéaire homogène, alors pour toute constante λ, toute suite de la forme $\{\lambda r^n\}_{n\geq 0}$ est une solution générale de cette équation.

Dans le cas de la suite de Fibonacci, on calcule le discriminant $\Delta = 5$ et on trouve les deux racines $r_1 = \frac{1-\sqrt{5}}{2}$ et $r_2 = \frac{1+\sqrt{5}}{2}$

Cas des racines doubles Si le discriminant $\Delta = 0$, alors le polynôme n'a qu'une seule racine (de multiplicité 2). En remarquant que r racine double de P(x) implique que r est aussi une racine de $P'(x)$ on peut démontrer que $\{n\lambda r^n\}_{n\geq 0}$ est aussi une solution de l'équation récurrente.

Théorème Les solutions générales d'une équation récurrente linéaire homogène d'ordre 2 dont le polynôme caractéristique de deux racines r_1 et r_2 sont :
– Si $r_1 \neq r_2$: $\{\lambda_1 r_1^n + \lambda_2 r_2^n\}_{n\geq 0}$ pour toute constantes λ_1, λ_2
– Si $r_1 = r_2$: $\{(\lambda_1 + \lambda_2 \times n) r_1^n\}_{n\geq 0}$ pour toute constantes λ_1, λ_2

Preuve dans le cas d'une racine double Soit l'équation $u_{n+2} + aU_{n+1} + bu_n = 0$ et soit r une racine double du polynôme caractéristique.
$P(x) = x^2 + ax + b$, donc r est aussi une racine de $P'(x) = 2x + a$. La suite $\{nr^n\}_{n\geq 0}$ est une solution de l'équation car

$$\begin{aligned}(n+2)r^{n+2} + a(n+1)r^{n+1} + bnr^n &= n(r^{n+2} + ar^{n+1} + br^n) + 2r^{n+2} + ar^{n+1} \\ &= r^n[n(r^2 + ar + b) + r(2r + a)] \\ &= 0\end{aligned}$$

3.3.4 Recherche de solutions particulières pour les équations récurrentes linéaires homogènes d'ordre 2

Dans le cas de la suite de Fibonacci, on cherche une solution particulières satisfaisant les conditions initiales et qui est de la forme $\lambda_1 r_1^n + \lambda_2 r_2^n$ où $r_1 = \frac{1-\sqrt{5}}{2}$ et $r_2 = \frac{1+\sqrt{5}}{2}$

Donc on cherche λ_1, λ_2 tels que

$$\begin{aligned}u_0 &= 1 = \lambda_1 r_1^0 + \lambda_2 r_2^0 = \lambda_1 + \lambda_2 \\ u_1 &= 1 = \lambda_1 r_1^1 + \lambda r_2^1 = \frac{\lambda_1 + \lambda_2}{2} + \frac{\lambda_2 - \lambda_1}{2} \times \sqrt{5}\end{aligned}$$

$$\Rightarrow \begin{cases} 1 &= \lambda_1 + \lambda_2 \\ \frac{1}{2} &= \frac{\lambda_2 - \lambda_1}{2}\sqrt{5} \end{cases} \Rightarrow \begin{cases} 1 &= \lambda_1 + \lambda_2 \\ \frac{1}{\sqrt{5}} &= \lambda_2 - \lambda_1 \end{cases} \Rightarrow \begin{cases} \lambda_2 &= \frac{1+\frac{1}{\sqrt{5}}}{2} \\ \lambda_1 &= \frac{1-\frac{1}{\sqrt{5}}}{2} \end{cases}$$

Au final, on trouve la solution particulière :

$$U_n = \frac{\sqrt{5}-1}{2\sqrt{5}}(\frac{1-\sqrt{5}}{2}) + \frac{\sqrt{5}+1}{2\sqrt{5}}(\frac{1+\sqrt{5}}{2})^n$$

3.3.4.1 Résumé de la méthode pour les équations homogènes d'ordre 2

Pour résoudre l'équation $u_n = aU_{n-1} + bu_{n-2}$

1. On calcule le polynôme caractéristique $P(x) = x^2 - ax - b$
2. On calcul les racines (éventuellement complexes) r_1 et r_2 de P
3. On cherche les coefficients λ_1 λ_2 tels que $\lambda_1 r_1^n + \lambda_2 r_2^n$ satisfaisant les CI

3.3.5 Équations récurrentes d'ordre k

Pour les équations récurrentes homogènes d'ordre k, les considérations sur le polynôme caractéristique et ses racines restent valables. La difficulté est calculatoire car il faut trouver les racines d'un polynôme de degré k. Mais lorsque l'équation a été obtenu en éliminant la partie non-homogène, les coefficients utilisés sont des solutions.

Ex Pour l'algorithme de Strasser, on a obtenu l'équation en faisant

$$E_n - GE_{n-1}$$

où E_n désigne l'équation de rang n
$\Rightarrow$ 4 est une racine du polynôme caractéristique.

En cas de racine d'ordre m, on peut montrer par récurrence que $\{n^j\alpha^n\}_{n\geq 0}$ est une solution de l'équation récurrente homogène pour tout $j = 0, \cdots, m-1$. Ceci nous permet d'avoir k variables dans le système d'équation linéaires dérivées des CI.

Le théorème suivant généralise le théorème 2 au cas de récurrences homogènes d'ordre $k > 2$ et prend en compte directement le second membre.

Théorème 3 Supposons que le polynôme caractéristique de la récurrence homogène $u_n = au_{n-1} + \cdots + a_k u_{n-k}$ admet p racines $ri(i = 1, \cdots, p)$ de multiplicité $mi(i = 1, \cdots, p)$. Alors la solution de la récurrence

$$u_n = a_1 u_{n-1} + \cdots + a_k u_{n-k} + \sum_{i=1}^{t} b_i^n P_i(n)$$

où p_i est un polynôme de degré d_i) est donnée par

$$\sum_{i=1}^{t} b_i^n Q_i(n)\,^{1} + \sum_{i\in\{1,\ldots,p\}} r_i^n R_i(n)\,^{2}$$

tel que $r_i \notin \{b_1, \cdots, b_t\}$

Où

$$\deg(Q_i) = \begin{cases} d_i & si \quad b_i \notin \{r_i, \cdots, r_p\} \\ d_i + m_j & si \quad b_i = r_j \end{cases}$$

Et $\deg(R_i) = m_i - 1$

On obtient les polynômes Q_i et R_i à partir des CI et par identification des coefficients des termes $b_i^n n^j$ dans la récurrence.

R Dans le théorème 2, il n'y avait de second membre (t=0) et les polynômes R_i étaient de la forme λ_1 ou $\lambda_1 = \lambda_2 n$

$$\left.\begin{array}{lcl} u_n & = & u_{n-1} + 1n^3 \\ u_{n-1} & = & u_{n-2} + 1 \end{array}\right\} u_n - u_{n-1} = u_{n-1} - u_{n-2}$$

1. Partie de la solution qui prend en compte le second membre
2. Solution pour la récurrence homogène

Ex

$$T(n) = 2T(\frac{n}{2}) + n; T(1) = 1$$

Après changement de variable $n = 2^k$, $u_k = T(n)$, nous avons $u_k = 2u_{k-1} + 2^k$.
Ici le second membre

$$\sum_{i=1}^{t} b_i^k Pi(k) = 2^k$$

Donc $T = 1$, $P_i(k) = 1$, $b_i = 2$
Le polynôme caractéristique $P(x) = x - 2$. La seule racine est $r_i = 2$. Donc la solution particulière est de la forme $2^k(q_0 + q_i^k)$ car $\deg(Q_i) = \deg P_i +$ multiplicité $= 0+1$, et cette solution satisfait la CI et la récurrence $1 = T(1) = u_0$

$$2^k(q_0 + q_1 k) = 2 \times 2^{k-1}(q_0 + q_i(k-1))$$

D'où $q_0 = 1$, $q_1 = 1$ donc $u_n = 2^k(1+k)$ et $T(n) = u_k = n(1 + \log_2 n)$

3.3.6 Théorème pour les récurrences par divisions

Le théorème suivant nous donne directement l'ordre de grandeur de la solution en fonction des coefficients de l'équation récurrente.

Théorème 4 Soient $a \geq 1$, $b > 1$ deux constantes, $f(n)$ une fonction, et $\{t(n\}_{n\geq 0}$ une suite vérifiant l'équation $T(n) = aT(\frac{n}{b}) + f(n)$

On a pour $\epsilon > 0$
- Si $f(n) = 0(n^{\log_b a - \epsilon}$ alors $T(n) = \Theta(n^{\log_b a})$
- Si $f(n) = \Theta(n^{\log_b a})$ alors $T(n) = \Theta(n^{\log_b a . \log n})$
- Si $f(n) = \Omega(n^{\log_b a + \epsilon}$ et $af(\frac{n}{b}) \leq cf(n)$ pour une constante $c > 1$, alors $T(n) = \Theta(f(n))$

Annexe A

Exercices

A.1 TD 1

A.1.1 Lesquelles des affirmations suivantes sont vraies ?

1. $n^2.5 = \Theta(n^3)$: Faux
2. $n^2.5 = O(n^3)$: Vrai
3. $n^2.5 = \Omega(n^3)$: Faux
4. $log_2(2n) = \Theta(\log n)$: Vrai
5. Vrai
6. Faux

A.1.2 Une seule des afirmations suivantes est vraie. Laquelle ?

Réponse D

A.1.3 Une seule des affirmations suivantes est vraie. Laquelle ?

Réponse C $n + n\log_2 n \leq 2n\log_2 n = \Theta(n\log n)$

R On ne s'occupe pas des facteurs constants

A.1.4

Réponse D

A.1.5 Laquelle des affirmations suivantes sont vraies

1. $\max(f(n), g(n)) = \Theta(f(n)+g(n))$ Vrai : $\max(f(n), g(n)) \leq f(n)+n(n) \leq 2\max(f(n), g(n))$
2. Vrai : $\frac{1}{c}f(n) \leq g(n)$ et $g(n) \leq \frac{1}{2}f(n)$

3. Vrai : $\forall n \geq n_0 : f(n) \leq cg(n)$
4. Faux
5. Vrai
6. Faux

$$g(n) = 2n, f(n) = ng(n) = O(f(n))2^{g(n)} = 2^{2n} = (2^n)^2$$

A.1.6 Lesquelles des affirmations suivantes sont vraies ?

Réponse D.

$$\begin{array}{ccc} f(n) \leq c_1 g(n) & , & g(n) \leq c_2 f(n) \\ \frac{1}{c_2} \leq \frac{f(n)}{g(n)} \leq c_1.1 & \Rightarrow & \frac{f(n)}{g(n)} = \Theta(1) \end{array}$$

A.1.7 Simplifiez les expressions suivantes

1. $O(4n^2 + 3n^2 + 7\log_2(n^n)) = O(n^3)$
2. $\Theta(n\log_2 n + 17n + 2n^3 = \Theta(n^2)$
3. $\Omega(4n^2 + 3n^3) = \Omega(n^3)$
4. $O(2^{n\log_3} n + 3\log_2 n!) = O(n^2)$
5. $O(2\log_3 n + 3\log_2 n + 6) = O(\log n)$

A.1.8 Classez les fonctions suivantes dans l'ordre croissant d'ordre de grandeur

1. $4n\log_2 n + 4n$
2. $2n\log_2 n + 4n$
3. $n^2 \log_e n$

A.1.9

$$\Theta(\frac{1}{1-p})$$

$$p = 1 - (\frac{1}{6})^{n-1}$$

$$\Theta(\frac{1}{1-(1-\frac{1}{6^{n-1}})}) = \Theta(\frac{1}{\frac{1}{6^{n-1}}}) = \Theta(6^{n-1}) = \Theta(6^n)$$

Donc réponse D.

A.1.10

a $\Theta(1)$
b $\Theta(1)$
c $\Theta(\log n)$
d $\Theta(n \log n)$
e $\Theta(n^3)$
f $\Theta(n^4)$

A.1.11 16.

A.1.11.1 a

$$\left.\begin{array}{rcr} V_n &=& V_{n-1}+1 \\ V_{n+1} &=& V_n+1 \end{array}\right\} \Rightarrow \begin{array}{rcr} V_{n+1}-V_n &=& V_n-V_{n-1} \\ V_{n+1} &=& 2V_n-V_{n-1} \end{array}$$

A.1.11.2 b

$$\left.\begin{array}{rcr} T(n) &=& 7T(n-1)+4^n \\ T(n+1) &=& 7T(n)+4^{n+1} \end{array}\right\} T(n+1)-4T(n)=7T(n)-28T(n-1)$$

A.1.12 17.

A.1.12.1 a

$$\begin{array}{rcl} V_{k+1}-4V_k+4V_{k-1} &=& 0 \\ x^2-4x+4 &=& 0 \end{array}$$

$$\Delta = 16-16=0$$

$$r=\frac{4}{2}=0$$

$$(\lambda_1+\lambda_2 n)2^n$$

A.1.12.2 b

$$\begin{array}{rcl} V_{k+2}-2V_{k+1}+V_k &=& 0 \\ x^2-2x+1 \end{array}$$

$$\Delta = 0$$
$$r = 1$$
$$S = \lambda_1 + \lambda_2 n$$

A.1.12.3 c

A.1.13 18

A.1.13.1 a

$$U_n = U_{n-1} + 2U_{n-2}; U_0 = u_1 = 1$$

Polynôme caractéristique $p(x) = x^2 - x - 2 = (x-2)(x+1)$
Solution générale $U_n = \lambda_1 2^n + \lambda_1(-1)^n$
CI

$$\left.\begin{array}{rcl} 1 & = & \lambda_1 + \lambda_2 \\ 1 & = & 2\lambda_1 - \lambda_2 \end{array}\right\} 2 = 3\lambda_1; y_1 = \frac{2}{3}; \lambda_2 = \frac{1}{3}$$

D'où la solution particulière $u_n = \frac{2}{3}2^n + \frac{1}{3}^n$