Complexité des algorithmes
La O-Notation
(Partie 2)

1. Introduction
2. Efficacité en temps et en espace
3. Notation de Landau (O-Notation)
4. Classes de complexité
5. Règles de calcul de la complexité d'un algorithme itératif
6. Analyse des algorithmes récursifs

6. Analyse des algorithmes récursifs

Pour mesurer un algorithme récursif, il faut d'abord déterminer son équation de récurrence, puis résoudre des équations de récurrence.

Il existe 3 méthodes :

a) éliminer la récurrence par substitution de proche en proche (dilatation)

b) deviner une solution et la démontrer par récurrence

c) utiliser la solution de certaines équations connues

6. Analyse des algorithmes récursifs

6.1. Dilatation

Exemple 1 : Factorielle

Soit la fonction récursive factorielle suivante :

Posons T(n) le temps d'exécution nécessaire pour un appel à Fact(n)

$$T(n) = \begin{cases} a & \text{si } n<=1 \\ b+T(n-1) & \text{sinon (si } n>1) \end{cases}$$

```
Fact(n:entier) : entier
  Si n <= 1
    Fact = 1
  Sinon
    Fact = n * Fact(n-1)
  Fsi
```

a: est une constante

a = le temps du test (n<=1)
+ le temps de l'affectation (Fact = 1)

b: est une constante

b = le temps du test (n>1) + le temps de l'opération produit de n par Fact(n-1) + le temps de l'affectation finale

T(n-1): le temps nécessaire pour le calcul de Fact(n-1)
Il sera calculé (récursivement) avec la même décomposition.

6. Analyse des algorithmes récursifs

6.1. Dilatation

Exemple 1 : Factorielle

$$T(n) = \begin{cases} a & \text{si } n<=1 \\ b+T(n-1) & \text{sinon (si } n>1) \end{cases}$$

Pour calculer la solution générale de cette équation, on peut procéder par substitution:

T(n) = b + T(n-1) = b + [b+T(n-2)]
= 2b + T(n-2) = 2b + [b+T(n-3)]
= ...
= ib + T(n-i) = ib + [b+T(n-i+1)]
= ...
= (n-1)b + T(n-n+1) = nb - b + T(1) = nb - b + a

T(n) = nb - b + a (avec b une constante positive)

Donc Fact est en $\mathcal{O}$ (n)

6. Analyse des algorithmes récursifs

<u>Exemple 2</u> : Tour de Hanoi

```
Hanoi(A, B, C : caractère, n: entier)
  SI n = 1
    Déplacer un disque de A vers B
  SINON
    Hanoi(A, C, B, n - 1)
    Déplacer un disque de A vers B
    Hanoi(C, B, A, n-1)
  FSI
```

➔ approche « Diviser pour régner »

$$\begin{cases} T(n) = a & \text{si } n=0 \\ T(n) = 1+2T(n-1) & \text{sinon (si } n>=1) \end{cases}$$

Pour calculer la solution générale de cette équation :

$T(n) = 1 + 2T(n-1) \qquad = 1 + 2[1+2T(n-2)]$

$= 1 + 2 + 4T(n-2) \qquad = 1 + 2+4[1+2T(n-3)]$

...

$= 1 + 2 + 2^2 + \ldots + 2^{i-1} + 2^i\, T(n-i)$

...

$= 1 + 2 + 2^2 + \ldots + 2^{n-2} + 2^{n-1}\, T(1)$

$= 1 + 2 + 2^2 + \ldots + 2^{n-1} + 2^n\, T(0)$

$= 1 + 2 + 2^2 + \ldots + 2^{n-1} \quad = 2^n - 1$

Donc Hanoi est en $\Theta(2^n)$

6. Analyse des algorithmes récursifs

6.2. <u>Devinette</u>

<u>Exemple 1</u> :Tri par fusion

Soit la fonction récursive tri par fusion d'un tableau entre la borne inférieure B_{inf} et la borne supérieure B_{sup} suivante:

```
Tri_Fusion(Tab, B_inf, B_sup)
  var milieu : entier
  Si  B_sup - B_inf  > 1 Alors
     milieu = (B_inf +B_sup)/2
     Tri_Fusion(Tab,B_inf,milieu)
     Tri_Fusion(Tab,milieu,B_sup)
     Fusion(Tab,B_inf,milieu,B_sup)
  Fsi
```

Posons T(n) le temps d'exécution nécessaire pour l'appel à Tri_Fusion

$T(n) = 2T(n/2) + n$

Posons $n = 2^k \Rightarrow k = \log_2 n$

$T(2^k) = 2T(2^k/2) + 2^k = 2T(2^{k-1}) + 2^k$

$= 2^k * k + 2^k$

à démontrer par récurrence

on a deux appels à Tri_Fusion avec une taille de n/2 et un appel sur la fusion dans un tableau de taille n

➔ approche « diviser pour régner »

6. Analyse des algorithmes récursifs

6.2. <u>Devinette</u>

<u>Exemple 1</u> :Tri par fusion

Démontrons par récurrence que :

$T(2^k) = 2^k + 2T(2^{k-1}) \Rightarrow T(2^k) = 2^k + 2^k * k$

- <u>Initialisation</u>

T(0) = 0 (pas de donnée...)

$T(1) = 1 + 2*T(0) = 1 = 2^0 + 2^0*0$

$T(2) = 2 + 2*T(1) = 4 = 2^1 + 2^1*1$

$T(4) = 4 + 2*T(2) = 12 = 2^2 + 2^2*2$

...

- <u>Récurrence</u>

$T(2^{k+1}) = 2^{k+1} + 2*T(2^k)$

$= 2^{k+1} + 2*(2^k + 2^k*k)$

$= 2^{k+1} + 2^{k+1} + 2^{k+1}*k$

$= 2^{k+1} + 2^{k+1}*(1 + k)$

$= 2^{k+1} + 2^{k+1}*(k + 1)$

Donc:

$T(n) = n + 2T(n/2)$

$\Rightarrow T(n) = n + n\mathrm{Log}(n)$

Donc Tri par fusion est en $\Theta(n\mathrm{Log}(n))$

6. Analyse des algorithmes récursifs

6.3. Equations de récurrences connues

6.3.1. Equations homogènes

Une équation de récurrence est dite homogène si elle a la forme suivante:

$$a_0 t_n + a_1 t_{n-1} + \ldots + a_k t_{n-k} = 0 \qquad (1)$$

pour trouver sa solution générale, on procède comme suit:

a) Etablir son équation caractéristique (un polynôme de degré k) :

$$a_0 x^k + a_1 x^{k-1} + \ldots a_k = 0$$

calculer ses racines: $r_1, r_2, \ldots r_k$

b) Si toutes les racine r_i sont distinctes, alors la solution générale de (1) est donnée par :

$$t_n = C_1 r_1^n + C_2 r_2^n + \ldots C_k r_k^n \qquad (2)$$

où les C_i sont des constantes que l'on peut déterminer avec les conditions initiales de l'équation de récurrence.

6. Analyse des algorithmes récursifs
6.3. Equations de récurrences connues
6.3.1. Equations homogènes

c) Si r_j est une solution multiple (de multiplicité m : apparaît m fois comme solution de l'équation caractéristique), alors la solution générale de (1) est donnée en remplaçant dans (2) le terme $C_j r_j^n$ par la somme: $C_{j1} r_j^n + C_{j2} n r_j^n + C_{j3} n^2 r_j^n \dots C_{jm} n^{m-1} r_j^n$

Exemples:

1) Soit : $t_n - 3t_{n-1} - 4t_{n-2} = 0$ *(forme générale pour n>=2)*
$t_0 = 0$, $t_1 = 1$ *(conditions initiales)*

son équation caractéristique est : $x^2 – 3x – 4 = 0$

➔ 2 racines distinctes : $r_1 = -1$ et $r_2 = 4$
donc la solution générale est : $t_n = C_1(-1)^n + C_2\, 4^n$

➔ ∅ (4^n) car C_2 est positive
en appliquant les conditions initiales on trouve:
$C_1 = -1/5$ et $C_2 = 1/5$

6. Analyse des algorithmes récursifs
6.3. Equations de récurrences connues
6.3.1. Equations homogènes

Exemples:

2) Soit : $t_n - 5t_{n-1} + 8t_{n-2} - 4t_{n-3} = 0$ *(forme générale pour n>=3)*
$t_0 = 0$, $t_1 = 1$, $t_2 = 2$ *(conditions initiales)*

son équation caractéristique est : $x^3 – 5x^2 + 8x – 4 = 0$
ou alors $(x-1)(x-2)^2 = 0$

➔ 3 racines: 1, 2 et 2 (2 est une solution multiple de multiplicité = 2)
donc la solution générale est: $t_n = C_1(1)^n + C_2\, 2^n + C_3 n\, 2^n$

➔ en appliquant les conditions initiales on trouve
$C_1 = -2$, $C_2 = 2$ et $C_3 = -1/2$

6. Analyse des algorithmes récursifs
6.3. Equations de récurrences connues
6.3.2. Equations non homogènes

Une équation de récurrence est dite non homogène si elle a la forme suivante:

$a_0 t_n + a_1 t_{n-1} + \dots + a_k t_{n-k} = b_1^n P_1(n) + b_2^n P_2(n) + b_3^n P_3(n) + \dots$ (3)

où les b_i sont des constantes distinctes et les $P_i(n)$ des polynômes en n de degré d_i

La résolution d'une telle équation suit le même schéma que celui des équations homogènes en partant de l'équation caractéristique suivante :

$(a_0 x^k + a_1 x^{k-1} + \dots a_k)\,(x-b_1)^{d_1+1}\,(x-b_2)^{d_2+1}\,(x-b_3)^{d_3+1} \dots = 0$

6. Analyse des algorithmes récursifs
6.3. Equations de récurrences connues
6.3.2. Equations non homogènes

Exemple:

Soit : $t_n - 2t_{n-1} = n - 2^n$ *(forme générale pour n>=1)*
$t_0 = 0$, *(conditions initiales)*

C'est une équation non homogène avec:
$b_1 = 1$, $P_1(n) = n$, $d_1 = 1$
$b_2 = 2$, $P_2(n) = 1$, $d_2 = 0$

son équation caractéristique est donc: $(x-2)(x-1)^2(x-2) = 0$

➔ 4 racines: 2, 1, 1 et 2 (1 et 2 sont, chacune, de multiplicité 2)
donc la solution générale est:
$t_n = C_1(1)^n + C_2 n(1)^n + C_3 2^n + C_4 n 2^n$