Measuring recursive algorithms

D.E ZEGOUR

École Supérieure d'Informatique

ESI

Measuring recursive algorithms

Method

To measure a recursive algorithm

- Find the recurrence equation
- Solve it

Measuring recursive algorithms

Recurrence equation

```
Fact(n)
IF n ≤ 1
   Fact := 1
ELSE
   Fact := n * Fact(n-1)
ENDIF
```

⇓

$T(n) = a$ if $n \le 1$
$\quad = T(n-1) + b$ Otherwise

```
Sort (L, n) { Suppose n = 2^k }
IF n = 1
   Sort := L
ELSE
    L1 := L [1..n/2]
    L2 := L [n/2+1..n]
    Sort := Merge ( Tri(L1, n/2), tri(L2, n/2))
ENDIF
```

⇓

$T(n) = a$ if $n=1$
$\quad = 2\,T(n/2) + bn$ Otherwise

Measuring recursive algorithms

Resolution

In general, there are three methods to
solve recurrence equations:

a) By substitution (expand the recurrence).

b) Guess a solution and prove it by induction..

c) Use solutions from known recurrence equations

Measuring recursive algorithms

Resolution by substitution (Example 1)

```
Fact(n)
IF n ≤ 1
   Fact := 1
ELSE
   Fact := n * Fact(n-1)
ENDIF
```

$T(n) = a$ If $n \leq 1$
$T(n-1) + b$ Otherwise

Recurrence equation

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

It's $O(n)$

Measuring recursive algorithms

Resolution by substitution (Example 2)

```
Sort (L, n) { Suppose n = 2^k }
IF n = 1
   Sort := L
ELSE
    L1 := L [1..n/2]
    L2 := L [n/2+1..n]
    Sort := Merge ( Tri(L1, n/2), tri(L2, n/2))
ENDIF
```

$T(n) = a$ If $n=1$

$= 2\,T(n/2) + bn$ Otherwise

Recurrence equation

$T(n) = 2\,T(n/2) + bn$

$= 2\,[2\,T(n/4) + bn/2] + bn$

$= 4\,T[n/4] + 2\,bn$

$= 8\,T[n/8] + 3\,bn$

$= \ldots$

$= 2^i\,T[n/2^i] + ibn$

$= \ldots$

$= n\,T[1] + Log(n)\,bn$

$= an + Log(n)\,bn$

$= n(a + bLog(n))$

It's $O(n\,Log(n))$

Measuring recursive algorithms

Resolution by guessing (Example)

```
Sort (L, n) { Suppose n = 2^k }
IF n = 1
   Sort := L
ELSE
    L1 := L [1..n/2]
    L2 := L [n/2+1..n]
    Sort := Merge ( Tri(L1, n/2), tri(L2, n/2))
ENDIF
```

$T(n) = c_1$ if $n=1$

$= 2\ T(n/2) + c_2 n$ Otherwise **(1)**

Recurrence equation

To show that $T(n) = O(n \log2(n))$, which means $T(n) \leq a\ n \log2(n) + b$ for given a and b starting from a certain rang n:

- If $n = 1$, $T(1) \leq b$ (we can take $b = c_1$).

- We assume $T(k) \leq a\ k \log2(k) + b$ for all $k < n$ and try to establish that $t(n) \leq a\ n \log2(n) + b$.

Suppose $n \geq 2$,
then from (1) we get:
$T(n) \leq 2T(n/2) + c_2\ n$
$\leq 2(a(n/2) \log2(n/2) + b) + c_2\ n$
$\leq a\ n \log2(n) - a\ n \log2(2) + 2\ b + c_2\ n$
$\leq a\ n \log2(n) - a\ n + 2\ b + c_2 n$
$\leq a\ n \log2(n) + b + (b + c_2\ n - a\ n)$

Measuring recursive algorithms

Resolution by guessing (Example)

```
Sort (L, n) { Suppose n = 2^k }
IF n = 1
   Sort := L
ELSE
    L1 := L [1..n/2]
    L2 := L [n/2+1..n]
    Sort := Merge ( Tri(L1, n/2), tri(L2, n/2))
ENDIF
```

$T(n) = c_1$ si $n=1$

$= 2\,T(n/2) + c_2 n$ Otherwise **(1)**

Recurrence equation

For $T(n)$ to be $\leq a\,n\log2(n) + b$, we need:

$b + c_2\,n - a\,n \leq 0$

$a\,n \geq b + c_2\,n$

$a \geq (b + c_2\,n) / n$

For all $n \geq 1$ $\quad a \geq b + c_2$

Therefore, $T(n) \leq a\,n\log2(n) + b$ if the following two conditions are satisfied:

$b \geq c_1$

$a \geq b + c_2$

By choosing $b = c_1$ and $a = c_1 + c_2$, we conclude that for all $n > 1$:

$T(n) \leq (c_1 + c_2)\,n\mathrm{Log}(n) + c_1$

In other words, $T(n)$ is $O(\,n\,\mathrm{Log}(n)\,)$)

Measuring recursive algorithms

Resolution using known recurrence equations

1. Homogeneous equations

$a_0 t_n + a_1 t_{n-1} + ...a_k t_{n-k} = 0$ **(1)**

has as characteristic equation :

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

Solutions in R :

$r_1, r_2, ...r_k$.

If all the solutions ri are distinct, then solution of (1) is $t_n = c_1(r_1)^n + c_2(r_2)^n + ...c_k(r_k)^n$

If rj is a repeated solution (with multiplicity m), then solution of **(1)** is

$t_n = c_1(r_1)^n + c_2(r_2)^n + ... + \mathbf{(c_{j1}(r_j)^n + c_{j2}n\,(r_j)^n + ...\ c_{jm}n^{m-1}(r_j)^n\ )} + ...+ c_k(r_k)^n$

Remark : constants are determined by the initial conditions

Reminder : solution

$t_n = c_1(r_1)^n + c_2(r_2)^n + ...c_k(r_k)^n$

Measuring recursive algorithms

Resolution using known recurrence equations

Homogeneous equation (Example)

$t_n - 3t_{n-1} - 4t_{n-2} = 0$ for $n \geq 2$

$t_0 = 0, t_1 = 1$

Characteristic equation : $x^2 - 3x - 4 = 0$

Solutions : -1 et 4.

$t_n = c_1(-1)^n + c_2 4^n$

Initial Conditions

$0 = c_1 + c_2$

$1 = - c_1 + 4c_2$

$c_1 = -1/5$ et $c_2 = +1/5$

$t_n = -(1/5)(-1)^n + (1/5)4^n$

It's $O(4^n)$.

Measuring recursive algorithms

Resolution using known recurrence equations

2. Non-homogeneous equations

$a_0t^n + a_1t^{n-1} + ...a_kt^{n-k} = b_1^nP1(n) + b_2^nP2(n)+....$

b_i : constants

P_i : polynoms of degree d_i

Characteristic equation:

$(a_0x^k + a_1x^{k-1}+...a_k)\ (x-b_1)^{d1+1}\ (x-b_2)^{d2+1} .. = 0$

Solutions in R :

$r_1, r_2, ...r_k.$

If all the solutions ri are distinct, then solution of (1) is $t_n = c_1(r_1)^n + c_2(r_2)^n + ...c_k(r_k)^n$

If rj is a repeated solution (with multiplicity m), then solution of (1) is

$t_n = c_1(r_1)^n + c_2(r_2)^n + ... + (c_{j1}(r_j)^n + c_{j2}n\ (r_j)^n + ...\ c_{jm}n^{m-1}(r_j)^n\ ) + ...+ c_k(r_k)^n$

Remark : constants are determined by the initial conditions

Reminder : $a_0 t^n + a_1 t^{n-1} + ... a_k t^{n-k} = b_1^n P1(n) + b_2^n P2(n) + ...$

Measuring recursive algorithms

Reminder $t_n = c_1(r_1)^n + c_2(r_2)^n + ...c_k(r_k)^n$

Resolution using known recurrence equations

Non-homogeneous equation (Example 1)
Fibonacci sequence
Fib(n) := Fib(n-1) + Fib(n-2) if n > 1
Fib(0) = 0, Fib(1) = 1

Recurrence Equation :
T(n) = T(n-1) + T(n-2) + b if n > 1
T(n) = a otherwise

Non-homogeneous equation:
$t_n - t_{n-1} - t_{n-2} = b = 1^n b$
$t_0 = 0, t_1 = 1,$

$b_1 = 1, P_1 = b$, b is a polynom of degree 0

Characteristic equation : $(x^2 - x - 1)(x-1) = 0$

$(x-r1)(x-r2)(x-1) = 0$

$r1 = (1 + \sqrt{5})/2$
$r2 = (1 - \sqrt{5})/2$
$r3 = 1$

Therefore $t_n = c_1 ((1 + \sqrt{5})/2)^n + c_2 ((1 - \sqrt{5})/2)^n + c_3 1^n$

Measuring recursive algorithms

Resolution using known recurrence equations

$t_n = c_1 ((1 + \sqrt{5})/2)^n + c_2 ((1 - \sqrt{5})/2)^n + c_3 1^n$

Initial conditions

$0 = c_1 + c_2 + c_3$

$1= c_1(1 + \sqrt{5})/2 + c_2(1 - \sqrt{5})/2 + c_3$

$0 = c_1 + c_2$

$1= c_1(1 + \sqrt{5})/2 + c_2(1 - \sqrt{5})/2$

$c_2 = - c_1$

$c_1(1 + \sqrt{5})/2 - c_1(1 - \sqrt{5})/2 = 1$

$c_1\sqrt{5} = 1$

Initial conditions give

$c_1= 1/\sqrt{5}$, $c_2 = -1/\sqrt{5}$ et $c3 = 0$

$t_n = 1/\sqrt{5} ((1 + \sqrt{5})/2)^n - 1/\sqrt{5} ((1 - \sqrt{5})/2)^n$

It's $\mathbf{O((1 + \sqrt{5})/2)^n)}$

or $O(1.6180339^n)$

Measuring recursive algorithms

Resolution using known recurrence equations

Non-homogeneous equation (example 2)

$t_n - 2t_{n-1} = n + 2^n$

$t_0 = 0$

$b_1 = 1, P_1 = n; b_2 = 2, P_2 = 1$

Reminder : r_j repeated solution (multiplicity m)

$t_n = c_1(r_1)^n + c_2(r_2)^n + \ldots + \mathbf{(c_{j1}(r_j)^n + c_{j2} n (r_j)^n + \ldots c_{jm} n^{m-1} (r_j)^n)} + \ldots + c_k(r_k)^n$

Characteristic equation : $(x-2)\,(x-1)^2\,(x-2) = 0$

Therefore $t_n = c_1\, 1^n + c_2\, n\, 1^n + c_3\, 2^n + c_4\, n\, 2^n$

Initial conditions give $c_1 = -c_3$, c_2 and c_4 are arbitrary .

It's $O(n\, 2^n)$.