Cours Algorithmique Avancée

Master 1 Informatique
RN- Options : RS & SIA

Prof. BOUALLOUCHE Louiza

Chap. 3. Exemple typique sur l'efficacité de la méthode de résolution. Calcul de la puissance n de X

Position du problème et principe

En entrée : On a un entier *n* et un réel x.
En sortie : On souhaite obtenir x^n.

Principe général de calcul

L'idée consiste à construire une suite y_i , $i \in [2,m]$ en partant de $y_1 = x$, telle que la valeur y_k soit obtenue par multiplication de deux puissances de x précédemment calculées : $y_k = y_u \times y_v$, avec $1 \leq u, v < k$, $k \in [2,m]$.

Le résultat attendu est alors : $y_m = x^n$

Complexité en nombre de multiplications. La complexité ou le coût du calcul de x^n en nombre de multiplications est alors *m-1* (dépend de *m*. Plus *m* est petit, meilleure est la méthode utilisée).

1. Algorithme naïf (trivial)

La méthode classique naïve permet de multiplier x, *n-1* fois.
En partant de $y_1 = x$, $y_i = y_{i-1} * y_1$, $i \in [2,n]$. Résultat : $y_n = x^n$. Coût = $n-1$ multiplications.

```
Algorithme Puissance (x : réel, n : entier)
Début
 y[1] ← x;
 Pour i  allant de 2 à n faire
    y[i] ← y[i−1] × y[1];
 Fin pour
 retourner y[n];
Fin.
```

Complexité. Cet algorithme génère *n-1* multiplications.

2. Amélioration de l'algorithme naïf.

La première amélioration de cet algo. est de s'appuyer sur la propriété d'associativité de l'opération de multiplication (*), d'où la relation $x^{2n} = (x^n)^2$. Ainsi, on calcule d'abord x^n puis on multiplie le résultat par lui-même, à l'aide d'une seule multiplication. D'où $(X^n)^2 = X^{2n}$

Il en découle un algorithme récursif défini par

$$x^n = \begin{cases} x & Si \quad n = 1 \\ x^{n/2} * x^{n/2} & Sinon \ \ Si \quad n \ est \ \ pair \\ x^{n/2} * x^{n/2} * x & Sinon \ \ (n \neq 1 \ \ et \ \ \ n \ est \ impair) \end{cases}$$

a. Ecrire l'algorithme correspondant
b. Montrer qu'il est valide
c. Calculer sa complexité en nombre de multiplications

a. l'Algorithme récursif correspondant est décrit par:

```
Fonction puissance (x: réel, n: entier): réel
Debut
  Si n = 1 alors
      Retourner (x) ;
    Sinon
      si n mod 2 = 0 alors
          y ← puissance(x , n/2);      /* / : division entière
          Retourner (y*y) ;
        Sinon
          y ← puissance(x , n/2);      /* / : y ← puissance(x , (n − 1)/2);
          Retourner (y*y*x) ;
      Finsi
  Finsi
Fin
```

b. L'algorithme récursif est valide, car :

Démonstration par récurrence

a- On suppose que l'algorithme est valide jusqu'à n/2 (il fournit $x^{n/2}$,/ division entière), on peut aisément montrer qu'il est valide jusqu'à n (il fournit x^n). En effet, si n est pair alors

$x^n = x^{n/2} * x^{n/2}$ et si n est impair $x^n = x^{n/2} * x^{n/2} * x$

b- Il reste à montrer que l'algorithme est valide (fournit x^n) pour la valeur de départ. En effet, Si $n = 1$ alors $x^n = x$

c. La complexité T(n) en nombre de multiplications en utilisant sa relation de récurrence

T(1) = 0 (en effet, pour calculer x^n si n = 1, on n'a besoin d'aucune multiplication),

Si n > 1, on a

T(n) = T(n/2)+1 si n est pair sinon T(n) = T(n/2)+2
On a alors T(n) <= T(n/2)+2 <= $T(n/2^2)+2*2$ <= $T(n/2^3)+2*3$ … <= …. <= $T(n/2^k)+2*k$

On s'arrête lorsque $n/2^k = 1$. D'où k = log(n)/log2 = $\log_2(n)$
On obtient T(n) <= 2*k = 2* $\mathbf{[\log_2 n]}$

Exemple, pour x^{15} $on\ a\ 2^*\ [\log_2 15] = 6$ multiplications ou lieu de 14 multiplications avec l'algorithme naïf

Remarque. En particulier si n est une puissance de 2, $T(n) = \log_2(n)$ (en effet, à chaque niveau de la récurrence, la relation $T(n) = T(n/2)+1$ sera utilisée)

Exemple, pour n = 16, $T(n) = \log_2(16) = \log_2(2^4) = 4\log_2(2) = 4$ multiplications

3. Algorithme binaire

Algorithme

1 -- Écrire *n* sous forme binaire
2 -- Eliminer le 1 le plus à gauche
3 --Remplacer chaque :
– « 1 » par la paire de lettres « SX »; – « 0 » par la lettre « S ».
(S signifie « élever au carré » (*squaring*); – X signifie « multiplier par x »).

En partant de x, x^n est obtenu en effectuant, de gauche à droite les opérations, d'élévation au carré ou de multiplication par x.

Illustration avec *n* = 23

1. *n* = 10111
2. 0 1 1 1
3. S SX SX SX
4. En partant de $y_1 = x$, on obtient successivement :

$Y_2 = y_1 * y_1 = x^2$ $Y_3 = y_2 * y_2 = x^4$ $Y_4 = y_3 * y_1 = x^5$ $Y_5 = y_4 * y_4 = x^{10}$ $Y_6 = y_5 * y_1 = x^{11}$
$Y_7 = y_6 * y_6 = x^{22}$ $Y_8 = y_7 * y_1 = x^{23}$

On a donc réussi à évaluer x^{23} en 7 multiplications au lieu de 22!

Complexité.

Notons que les nombres dont la représentation binaire a exactement p chiffres, forment l'intervalle $[2^{p-1}, 2^p - 1]$.

Le nombre de chiffres dans l'écriture binaire de n : $1 + \lfloor \log_2 n \rfloor$
Notons $v(n)$ le nombre de «1».

Le nombre d'opérations effectuées :

- $(1+\lfloor \log_2 n \rfloor)-1$ élévations au carré (multiplications);
- $v(n)-1$ multiplications par x.

Soit en tout $T(n) = \lfloor \log_2 n \rfloor + v(n) - 1$ multiplications. La complexité ne peut être calculée de manière exacte du fait que l'on ne peut connaitre le nombre exact de «1»,

Trivialement, $1 \leq v(n) \leq \lfloor \log_2 n \rfloor$ *et* $\lfloor \log_2 n \rfloor \leq T(n) \leq 2\lfloor \log_2 n \rfloor$.

Pour $n = 1000$, l'algorithme trivial effectue 999 multiplications, et la méthode binaire au plus 18.

L'algorithme binaire est nettement plus efficace que le trivial, mais est-il optimal?

Appliquons la méthode binaire pour $n = 15$.

$n = 1111$

1 1 1

SX SX SX

En partant de x et on obtient successivement : x^2, x^3, x^6, x^7, x^{14}, x^{15}. Le calcul de x^{15} s'effectue alors en 6 multiplications.

Un autre schéma de calcul est : x^2, x^3, x^6, x^{12}, $x^{15} = x^{12} * x^3$. On obtient ainsi x^{15} en 5 multiplications.

La méthode binaire n'est donc pas *optimale* (c'est-à-dire que l'on peut mieux faire).

4. Algorithme des facteurs

$$x^n = \begin{cases} x^n & Si \quad n = 1 \\ x^{n-1} * x & \text{Si n} > 1 \text{ et } n \text{ est premier} \\ \left(x^p\right)^{n'} & \text{Si n} > 1 \text{ et } n \text{ est non premier} \\ & \text{et } n = p * n' \text{ avec } p : \text{plus petit diviseur premier de n} \end{cases}$$

Illustration pour *n=15*

15 = 3*5, 3 étant le plus petit diviseur premier de 15. Donc $x^{15} = (x^3)^5$.

Nous ré-exécutons l'algorithme pour évaluer $y = x^3$ *puis* y^5.

Evaluation de x^3 :

3 est premier. Donc $x^3 = x^2 * x$. Nous ré-exécutons l'algorithme pour évaluer x^2.

2 est premier. Donc $x^2 = x*x$.

Finalement, x^3 est évalué comme suit : $x^3 = x*x*x$, soit en deux multiplications.

Evaluation de y^5 :

5 est premier. Donc $y^5 = y^4 * y$. Nous ré-exécutons l'algorithme pour calculer y^4.

4 = 2*2, où 2 est le plus petit facteur premier de 4. Donc $y^4 = (y^2)^2$.

Finalement y^5 est évalué comme suit : $z = y*y$, $u = z*z$, $y^5 = u*y$, soit en 3 multiplications.

Finalement, x^{15} est évalué en 5 multiplications.

Cet algorithme est efficace mais on ne peut dire qu'il est optimal. Il est certain que l'on peut trouver des contre-exemples.