Complexité des algorithmes (2)

Nour-Eddine Oussous, Éric Wegrzynowski

Licence ST-A, USTL - API2

5 octobre 2009

Coût d'une boucle pour

Dans la boucle

```
pour i variant de a à b faire
  ACTION(i)
fin pour
```

si $f(i)$ désigne le coût de l'exécution de ACTION(i), alors le coût de la boucle est

$$c = \sum_{i=a}^{b} f(i)$$

Tri par insertion (algo)

Algo

Données : un tableau $A[1..n]$ d'entiers
But : trier le tableau A par ordre croissant
Var. locales : i

```
pour i variant de 2 à n faire
  inserer(A,i)
fin pour
```

Tri par insertion (coût)

Le coût de l'algorithme dépend

1. de la taille n du tableau
2. du contenu du tableau
3. du coût des opérations élémentaires (échanges, comparaisons, accès)

Nous nous intéressons

1. au nombre d'échanges d'éléments du tableau $e(A, n)$,
2. et au nombre de comparaisons d'éléments du tableau $c(A, n)$

Tri par insertion (coût)

En désignant par $e'(A, i)$ et $c'(A, i)$ les nombres d'échanges et de comparaisons dans l'action inserer(A,i), on a

$$\begin{aligned} e(A, n) &= \sum_{i=2}^{n} e'(A, i) \\ c(A, n) &= \sum_{i=2}^{n} c'(A, i) \end{aligned}$$

Tri par insertion (coût)

Comme on a vu que

$$0 \leq e'(A, i) \leq i - 1$$

et

$$1 \leq c'(A, i) \leq i - 1$$

on a

$$0 = \sum_{i=2}^{n} 0 \leq e(A, n) \leq \sum_{i=2}^{n} i - 1 = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$$

et

$$n - 1 = \sum_{i=2}^{n} 1 \leq c(A, n) \leq \sum_{i=2}^{n} i - 1 = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$$

Tri par insertion (conclusion)

Encadrements obtenus :

$$\boxed{0 \leq e(A, n) \leq \frac{n(n-1)}{2} = \Theta(n^2)}$$

et

$$\boxed{\Theta(n) = n - 1 \leq c(A, n) \leq \frac{n(n-1)}{2} = \Theta(n^2)}$$

Bornes des encadrements atteintes

- borne inférieure atteinte pour un tableau déjà trié
- borne supérieure atteinte pour un tableau trié dans l'ordre inverse

Exemples de sommes

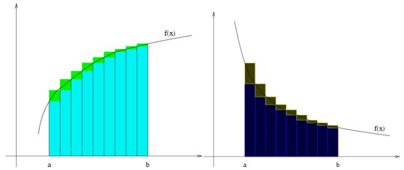
- $$\sum_{i=1}^{n} 1 = n$$
- $$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \Theta(n^2)$$
- $$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \Theta(n^3)$$
- Plus généralement pour $k \in \mathbb{N}$,
$$\sum_{i=1}^{n} i^k = \Theta(n^{k+1})$$
- Si $q \neq 1$
$$\sum_{i=0}^{n} q^i = \frac{q^{n+1}-1}{q-1} = \Theta(q^n)$$

Lien entre sommes et intégrales

Soient $a \leq b$ deux entiers. Soit $f : [a,b] \rightarrow \mathbb{R}$ une fonction croissante (resp. décroissante) et continue sur $[a,b]$. Alors

$$\sum_{i=a}^{b-1} f(i) \leqslant \int_a^b f(x)dx \leqslant \sum_{i=a+1}^{b} f(i) \qquad (1)$$

$$\left(\text{resp.} \sum_{i=a+1}^{b} f(i) \leqslant \int_a^b f(x)dx \leqslant \sum_{i=a}^{b-1} f(i)\right) \qquad (2)$$



Coût d'une boucle tant que

Dans la boucle

```
tant que  C(x)  faire
   ACTION(x)
fin tant que
```

en notant

- x_0 la valeur initiale de la donnée x, et $x_1, x_2, \ldots, x_k$ les valeurs qu'elle prend successivement à chaque étape, x_{k+1} la première valeur de la donnée pour laquelle $C(x_{k+1})$ n'est pas satisfaite,
- $g(x_i)$ le coût de la condition $C(x_i)$,
- et $f(x_i)$ le coût de l'action ACTION(x_i),

le coût de la boucle est

$$c = \sum_{i=0}^{k} f(x_i) + \sum_{i=0}^{k+1} g(x_i)$$

Multiplication de deux entiers

Le problème

Données : a et b deux entiers naturels
But : calculer $a \times b$

Plusieurs algorithmes variant selon les opérations élémentaires disponibles :

1. multiplication disponible : solution triviale
2. seule l'addition des entiers est disponible : solution avec boucle pour (exercice)
3. addition et division par deux des entiers disponibles.

Méthode égyptienne

Calcul du produit de $a = 67$ par $b = 21$.

t	u	v
67	21	0
134	10	67
268	5	67
536	2	335
1072	1	335
1407		1407

opération en cours

Les nombres à multiplier $u \neq 1 \Rightarrow t := t + t$, $u := u/2$ $u = 1 \Rightarrow$ terminé
suppression des lignes où u est pair
somme des valeurs restantes dans la colonne t utilisation d'une variable v pour le calcul de la somme
u impair $\Rightarrow v := v + t$

Conclusion

$\boxed{67 \times 21 = 1407}$

Seules opérations utilisées :$+$ et $\div 2$

Multiplication égyptienne : algo

Algo

Données : a et b deux entiers naturels, $b > 0$
But : calculer $a \times b$
Variables locales : t, u, v

```
mult(a,b):
  t := a; u := b; v := 0;
  {t × u + v = a × b}
  tant que u > 1 faire
    si u impair alors
      v := v + t;
    fin si;
    t := t + t;
    u := u ÷ 2;
    {t × u + v = a × b}
  fin tant que
  {t × u + v = a × b, u = 1}
  retouner t + v;
```


Multiplication égyptienne : analyse

- on s'intéresse au nombre d'additions effectuées
- ce coût ne dépend que de b

$$c(b) = \text{nombre d'additions pour multiplier par } b$$

Multiplication égyptienne : analyse

- À chaque étape du tant que une ou deux additions selon la parité de u
- meilleur des cas : 1 addition à chaque étape. C'est le cas si b est une puissance de 2 : $b = 2^p$. Dans ce cas

$$c(b) = p + 1$$

- pire des cas : 2 additions à chaque étape. C'est le cas si b est une puissance de 2 moins un : $b = 2^p - 1$. Dans ce cas

$$c(b) = 2(p-1) + 1 = 2p - 1$$

Multiplication égyptienne : analyse

Dans tous les cas, si $2^{p-1} \leq b \leq 2^p - 1$ on a

$$\boxed{p \leq c(b) \leq 2p - 1}$$

En tenant compte du fait que $p = \Theta(\log_2(b))$, on a

$$\boxed{c(b) = \Theta(p) = \Theta(\log_2(b))}$$

Conclusion

- cet algorithme est logarithmique en fonction de la valeur de b
- ou bien linéaire en fonction de la taille p de b

Algo

```
fact(n) :
  si n=0 alors
    fact := 1
  sinon
    fact := n×fact(n−1)
  fin si
```

- coût recherché = nbre de multiplications
- dépend de n

$c(n)$ = nbre de mult pour calculer $n!$

- Cas de base

$$c(0) = 0$$

- Cas récursif

$$\forall n \geq 1 \quad c(n) = 1 + c(n-1)$$

- $\Rightarrow$

$$\boxed{c(n) = n}$$

Conclusion

- Algorithme linéaire en la valeur de n
- Algorithme exponentiel en la taille de n

Algo

```
H(n,D,A,I):
  si n=1 alors
    deplacer de D vers A
  sinon
    H(n−1,D,I,A);
    deplacer de D vers A;
    H(n−1,I,A,D);
  fin si
```

- coût recherché = nbre de déplacements
- dépend de n

$c(n)$ = nbre de déplacements

- Cas de base

$$c(1) = 1$$

- Cas récursif

$$\forall n \geq 1 \quad c(n) = 1 + 2c(n-1)$$

- $\Rightarrow$

$$\boxed{c(n) = 2^n - 1}$$

Conclusion

- Algorithme exponentiel en la valeur de n

Schéma d'analyse récursive

Le coût d'un algorithme récursif peut toujours s'exprimer sous forme d'une équation de récurrence.

- la résolution des équations de récurrence peut s'avérer parfois délicate
- mais peut toujours être programmée