Module EA4 – Éléments d'Algorithmique II

Dominique Poulalhon
dominique.poulalhon@liafa.univ-paris-diderot.fr

Université Paris Diderot
L2 Informatique
Année universitaire 2014-2015

Points à retenir du cours précédent

algorithme = méthode systématique pour résoudre un problème

Points à retenir du cours précédent

algorithme = méthode systématique pour résoudre un problème

nécessite une preuve de correction : pour chaque entrée, l'algorithme doit terminer en produisant la bonne sortie

Points à retenir du cours précédent

algorithme = méthode systématique pour résoudre un problème

nécessite une preuve de correction : pour chaque entrée, l'algorithme doit terminer en produisant la bonne sortie

il peut exister plusieurs algorithmes pour le même problème

Points à retenir du cours précédent

algorithme = méthode systématique pour résoudre un problème

nécessite une preuve de correction : pour chaque entrée, l'algorithme doit terminer en produisant la bonne sortie

il peut exister plusieurs algorithmes pour le même problème

pour les comparer, il faut étudier leur complexité en temps et en espace

Complexité en espace

= quantité de mémoire nécessaire pour effectuer le calcul

Complexité en espace

= quantité de mémoire nécessaire pour effectuer le calcul

en général, on ne tient compte que de la mémoire auxiliaire

i.e. on ne tient pas compte de la mémoire incompressible nécessaire pour stocker les données et le résultat

Complexité en espace – exemple

```
def fibo_2(n) :
  if n <= 0 : return 0
  liste = [0, 1]
  for i in range(1, n) :
    liste.append(liste[i-1] + liste[i])
  return liste[n]
```

utilise un tableau de n (grands) entiers pour en calculer un seul (plus un (petit) entier comme indice de boucle)

Complexité en espace – exemple

```
def fibo_3(n) :
  if n <= 0 : return 0
  previous, last = 0, 1
  for i in range(1, n) :
    previous, last = last, previous + last
  return last
```

utilise seulement une variable auxiliaire de type (grand) entier (plus un (petit) entier comme indice de boucle)

Complexité en temps

= temps nécessaire pour mener le calcul à son terme

Complexité en temps

= temps nécessaire pour mener le calcul à son terme

plus difficile à quantifier précisément, essentiellement car ce temps dépend de la machine utilisée

Complexité en temps

= temps nécessaire pour mener le calcul à son terme

plus difficile à quantifier précisément, essentiellement car ce temps dépend de la machine utilisée

convention : on exprime ce temps en nombre d'opérations élémentaires effectuées

opération élémentaire = opération dont le temps d'exécution peut être considéré comme constant

exemple : affectation, comparaison, opération arithmétique sur des nombres de taille bornée...

Complexité et ordres de grandeur

Nombre d'opérations effectuées par un ordinateur à 1 Ghz :

en 1 seconde	10^9
en 1 heure	$3 \cdot 10^{12}$
en 1 jour	$9 \cdot 10^{13}$
en 1 an	$3 \cdot 10^{16}$

Complexité et ordres de grandeur

n	10	100	10^3	10^6	10^9	10^{12}
$\log_2 n$	4	7	10	20	30	40
$10n$	100	10^3	10^4	10^7	10^{10}	10^{13}
$n \log_2 n$	34	665	10^4	$2 \cdot 10^7$	$3 \cdot 10^{10}$	$4 \cdot 10^{13}$
n^2	100	10^4	10^6	10^{12}	10^{18}	10^{24}
n^3	10^3	10^6	10^9	10^{18}	10^{27}	10^{36}
2^n	10^3	10^{30}	10^{301}	...	...	...

Complexité et ordres de grandeur

n	10	100	10^3	10^6	10^9	10^{12}
$\log_2 n$	4	7	10	20	30	40
$10n$	100	10^3	10^4	10^7	10^{10}	10^{13}
$n \log_2 n$	34	665	10^4	$2 \cdot 10^7$	$3 \cdot 10^{10}$	$4 \cdot 10^{13}$
n^2	100	10^4	10^6	10^{12}	10^{18}	10^{24}
n^3	10^3	10^6	10^9	10^{18}	10^{27}	10^{36}
2^n	10^3	10^{30}	10^{301}	...	...	...

Définition

Soit f et g deux fonctions de $\mathbb{N}$ dans $\mathbb{N}$. On dit que :

- $f \in O(g)$, ou $f(n) \in O(g(n))$, ou $f(n) = O(g(n))$ si :

$$\exists c > 0,\ \exists n_0 \in \mathbb{N},\quad \forall n \geqslant n_0,\ f(n) < c \cdot g(n)$$

- $f \in \Omega(g)$, ou $f(n) \in \Omega(g(n))$, si :

$$\exists c > 0,\ \exists n_0 \in \mathbb{N},\quad \forall n \geqslant n_0,\ f(n) > c \cdot g(n)$$

- $f \in \Theta(g)$, ou $f(n) \in \Theta(g(n))$, si :

$$f(n) \in O(g(n)) \quad \text{et} \quad f(n) \in \Omega(g(n))$$

Complexité de l'exponentiation binaire

```
def puissance(a, n) :
  if n == 0 : return 1
  tmp = puissance(a, n//2)
  carre = tmp * tmp               # une multiplication
  if n%2 == 0 : return carre
  else : return a * carre         # une multiplication
```

Complexité

$\Theta(\log_2 n)$ multiplications de la forme $a^k \cdot a^\ell$, $k \in \{1, \ell\}$

Complexité de l'exponentiation binaire

Complexité

$\Theta(\log_2 n)$ multiplications de la forme $a^k \cdot a^\ell$

Complexité de l'exponentiation binaire

Complexité

$\Theta(\log_2 n)$ multiplications de la forme $a^k \cdot a^\ell$

si ces multiplications ont un coût constant, *i.e.* si les opérandes ont une taille constante, complexité en $\Theta(\log_2 n)$

c'est le cas avec l'arithmétique modulaire ou l'arithmétique flottante utilisées usuellement : tous les nombres sont codés sur exactement 32 (ou 64) bits, donc le coût d'une multiplication est constant

Complexité de l'exponentiation binaire

Complexité

$\Theta(\log_2 n)$ multiplications de la forme $a^k \cdot a^\ell$

si ces multiplications ont un coût constant, *i.e.* si les opérandes ont une taille constante, complexité en $\Theta(\log_2 n)$

sinon il faut tenir compte du coût de ces multiplications ; par exemple en arithmétique exacte sur des entiers :

valeur	taille (en bits)	coût du calcul naïf du carré
a	$\log_2 a$	$\Theta((\log_2 a)^2)$
a^k	$k \cdot \log_2 a$	$\Theta(k^2 \cdot (\log_2 a)^2)$

Complexité des calculs de F_n

utilisation naïve de la récurrence $\Longrightarrow$ $\Theta(\varphi^n)$ additions

```
def fibo(n) :
  if n <= 2 : return 1
  return fibo(n-1) + fibo(n-2)
```

Complexité des calculs de F_n

utilisation naïve de la récurrence $\Longrightarrow$ $\Theta(\varphi^n)$ additions

calcul itératif des n premières valeurs $\Longrightarrow$ $\Theta(n)$ additions

```
def fibo(n) :
  previous, last = 1, 1
  for i in range(1, n) :
    previous, last = last, previous + last
  return last
```

Complexité des calculs de F_n

utilisation naïve de la récurrence $\Longrightarrow$ $\Theta(\varphi^n)$ additions

calcul itératif des n premières valeurs $\Longrightarrow$ $\Theta(n)$ additions

calcul de $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$ $\Longrightarrow$ $\Theta(\log_2 n)$ multiplications… de matrices 2×2

(chacune impliquant 4 additions et 8 multiplications d'entiers)

Complexité des calculs de F_n

utilisation naïve de la récurrence $\Longrightarrow$ $\Theta(\varphi^n)$ additions

calcul itératif des n premières valeurs $\Longrightarrow$ $\Theta(n)$ additions

calcul de $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$ $\Longrightarrow$ $\Theta(\log_2 n)$ multiplications… de matrices 2×2

(chacune impliquant 4 additions et 8 multiplications d'entiers)

comme $F_n \in \Theta(\varphi^n)$, les opérations arithmétiques se font sur des entiers de taille n

$\Longrightarrow$ additions en $O(n)$ opérations élémentaires, multiplications en $O(n^2)$ (mais en fait, on peut faire plus efficace)