

EA4 – Éléments d’algorithmique

TD n° 1 : opérations arithmétiques

Exercice 1 : des limites du progrès...

Il y a 25 ans, un ordinateur faisait dix millions d’opérations par seconde et implémentait un algorithme de test de primalité demandant $1200\sqrt{n} + 100$ opérations pour un tableau d’entrée de taille n . On souhaite le comparer à un ordinateur 100 fois plus rapide, mais sur lequel tourne un (moins bon) algorithme de test de primalité demandant $\frac{n}{2}$ opérations. Quels sont les temps de calcul de chacun pour une entrée de taille $n = 10^8$? et $n = 10^{12}$? Quelle devrait être l’accélération pour avoir le même temps de calcul dans chacune des deux configurations?

Exercice 2 : exponentiation binaire, ou *exponentiation rapide*

1. Décrire un algorithme récursif `exponentiation_binaire_rec(m, n)` permettant de calculer m^n , similaire à l’algorithme vu en cours, mais basé sur l’identité $m^{2k} = (m^2)^k$ plutôt que $m^{2k} = (m^k)^2$.
2. Démarrer cet algorithme pour $m = 2$, $n = 13$.
3. Combien d’appels récursifs sont effectués en fonction de n ? Et combien de multiplications dans le pire des cas?
4. Proposer une version itérative de cet algorithme.

Exercice 3 : évaluation de polynômes

1. Proposer un algorithme le plus naïf possible qui, étant donné un polynôme P et une valeur x , calcule $P(x)$. On supposera que P est décrit par un tableau contenant, en case d’indice i , le coefficient du monôme de degré i .
2. Appliquer votre algorithme au polynôme $P(x) = x^4 + x^3 + x^2 + x + 1$ avec $x = 3$. Que constatez-vous?
3. Quel est le nombre d’additions et de multiplications effectuées lors de l’exécution de cet algorithme sur un polynôme de degré n ?
4. Comment peut-on améliorer la complexité de cet algorithme en utilisant l’exponentiation binaire?
5. Comment peut-on améliorer la complexité de cet algorithme en conservant dans une variable entière les puissances successives de la valeur de x ?
6. Montrer que l’algorithme suivant résout le même problème :

```
def horner(P, x) :  
    res = 0  
    for coeff in P[::-1] :      # effectue un parcours du tableau à l’envers  
        res = res * x + coeff  
    return res
```

Quelle est le nombre d’additions et de multiplications d’entiers effectuées lors de l’exécution de cet algorithme sur un polynôme de degré n ?

Exercice 4 : racine carrée

Pour cet exercice, nous considérons les deux algorithmes suivants :

```
def racine_1(n) :
    i = 0
    carre = i * i
    while carre <= n :
        if carre == n :
            return i
        i = i + 1
        carre = i * i
    return i - 1
```

```
def racine_2(n) :
    i, j = 0, n
    if n < 2 : return n
    while j > i + 1 :
        m = (i + j) // 2
        carre = m * m
        if carre == n : return m
        elif carre < n : i = m
        else : j = m
    return i
```

1. Tester `racine_1` pour les valeurs $n = 9$ et $n = 14$.
2. Montrer que `racine_1` calcule $\lfloor \sqrt{n} \rfloor$.
3. Calculer le nombre d'additions et de multiplications au pire lors de l'exécution de cet algorithme en fonction de n .
4. Montrer que `racine_2` calcule aussi \sqrt{n} , et calculer le nombre d'additions et de multiplications au pire en fonction de n .
5. Pour quelles valeurs de n est-il préférable d'utiliser chacun de ces algorithmes ?

Exercice 5 : nombres premiers

1. Proposer un algorithme `est_premier(p)` le plus naïf possible permettant de déterminer si un entier p est premier. Quelle est sa complexité ? Comment peut-on l'améliorer ?

On considère maintenant l'algorithme suivant :

```
def eratosthene(n) :
    tab = [False, False] + [True]*(n-1) # tab = [False, False, True, True, ..., True]
    for i in range(2, n+1) :
        if tab[i] :
            for k in range(2*i, n+1, i) : # depuis 2i, par pas de i, sans dépasser n
                tab[k] = False
    return tab
```

2. Exécuter l'algorithme pour $n = 10$.
3. Que représente le tableau calculé par `eratosthene(n)` ? Justifier.
4. Calculer un majorant du nombre d'additions et de multiplications d'entiers effectuées par l'algorithme pour une entrée n .
5. Pour vous convaincre de l'impact pratique des différences de complexité, comparer les temps de calcul de `[p for p in range(10**6) if est_premier(p)]` et de `[p for p,b in enumerate(eratosthene(10**6)) if b]`.

Calcul complexité :

- complexité
- par rapport à des opérations de base
 - en fonction d'un paramètre (généralement n)

ex:

```
def max(f):  
    n = len(f) → length tableau = null  
    maxi = -∞ → -∞ au None  
    for i = 1 to n  
        if maxi ≤ f[i]  
            maxi = f[i]  
    return maxi
```

Ω comparaisons
et $n+2$ affectations

Ex1

$$1200 \text{ Vn} + 100$$

Machin 1

vitesse 10^7 op/s

n	10^8	10^{12}
opération	$1,2 = 10^7 + 100$ $12 \times 10^8 = 100$ $1200 \times 10^9 + 100$	$1,2 \times 10^9 + 100$
temps	$1,2 + 10^{-7} \text{ sec}$ dans $1,2 \text{ sec}$	1200 s

$$f_p = \frac{\text{op}}{10^7 \text{ op/s}}$$

	10^8	10^{12}	
n			
OP	5×10^4	5×10^{11}	$\frac{n}{g} \text{ OP}$

temps	$0,05\text{s}$	500s	100×10^4
		$5 \times 10^3 : 10^9$	OP/s

$= 5 \times 10^{-2}$

$= 10^5 \text{ op/s}$

vitesse

Acceleration

$$\frac{\text{OP ordi 1}}{\text{vitesse ordi 1}} = \frac{\text{OP ordi 2}}{\text{vitesse ordi 2}} \rightarrow \text{temps}$$

$\hookrightarrow = a \times \text{vitesse ordi 1}$

$$\frac{n_1}{\text{OP ordi 1}} = \frac{n_2}{\text{OP ordi 2}}$$

$$\frac{n_1}{\text{vitesse ordi 1}} = \frac{n_2}{a \times \text{vitesse ordi 1}}$$

Quel doit être a pour prendre le même temps ?

Cas $n = 10^8$

$$\text{temp}_1 = \frac{1,2 \times 10^4}{10^4} = 1,2$$

$$\text{temp}_2 = \frac{5 \times 10^4}{a \times 10^4} = \frac{5}{a}$$

$$t_1 = t_2 \Leftrightarrow \frac{5}{a} = 1,2$$

$$a = \frac{5}{1,2}$$

$$a \approx 4,17$$

$$\text{Cas } n = 10^{12}$$

$$\text{temps}_1 = \frac{1,2 \times 10^9}{10^9} = 1,2 \text{ s}$$

$$\text{temps}_2 = \frac{5 \times 10^{11}}{a \times 10^9} = \frac{5 \times 10^2}{a}$$

$$t_1 = t_2 \Leftrightarrow \frac{5 \times 10^2}{a} = 1,2$$

$$\Leftrightarrow a = 417$$

417

Ex 2 : Calcul de m^n

res = 1

pour $i = 1 \text{ à } n$:

$$\text{res} = \text{res} \times m$$

Renvoyer res

Complexité n multiplications

Remarque : $m^{2k} = (m^2)^k$
 $m^{2k+1} = (m^2)^k \times m$

$$p = m \times m$$

$$m^a = p^k$$

l'op exponentiation-binaire-rec(m, a)

pour calculer m^n

$$\text{res} = 1$$

Si n est pair:

$$k = \frac{n}{2}$$

return exp($m \times m, k$)

(n est égal à $2k$)
pour un certain k

sinon :

$$k = \frac{n-1}{2}$$

n est impair $\exists k$

$$\begin{aligned} n &= 2k+1 \\ n-1 &= 2k \end{aligned}$$

$$\text{res} = \exp(m \times m, k)$$

$$\text{res} = \text{res} \times m$$

return res

$$n \text{ pair} \rightarrow m^n = p^k$$

$$n \text{ impair} \rightarrow m^n = p^k \times m$$

$$2 \times \exp(2, 13) \quad |$$

$$k = \frac{13-1}{2} = 6$$

$$\text{res} = \exp(2 \times 2, 6) \times 2$$

$$\exp(16, 3)$$

$$\hookrightarrow K = 1$$

$$\exp(m = 16 \times 16 = 256, n = 1)$$

$\times 16 \quad K = 0$

$\Rightarrow \underbrace{\exp(256, 0)}_1 \times 256$

$$\Rightarrow 256 \times 16 \times 2 = 8192$$

Le nombre d'appel :

\hookrightarrow en partant de n
quand est-ce que l'arrive à 0?

On a $K \leq \frac{n}{2}$ $n_0 \rightarrow n_1 \rightarrow n_2 \dots \rightarrow 0$

$$\Rightarrow \frac{1}{2} \frac{n_0}{2} \geq \frac{n_1}{2}$$

$$\frac{1}{2} \frac{n_1}{2} \geq \frac{n_2}{2}$$

$$\left. \begin{array}{l} \frac{n_0}{2} \geq n_1 \\ \frac{n_0}{4} \geq n_2 \end{array} \right\} \frac{n_0}{2^i} \geq n_i$$

$$\frac{n_0}{2^i} \geq n_i$$

On a $\frac{n_0}{2^i} \geq n_i$

Il existe i tel que $n_i = 0$

Qu'est ce que $\frac{n_0}{2^i} < 1$
 $n < 2^i$

$$\Leftrightarrow \log n < i$$

$$\log_2(2^j) = j$$

$$\log_2(8192) = 13$$

13 opérations

26 multiplications

Ex 3

$$P(x) = x^4 + 3x^2 + x^1 + \cancel{5x}^0$$

$$P(g) = g^4 + 3g^2 + g + g^0$$

$$t = [5, 1, 3, 0, 1]^T \text{ coef}$$

Opérations de base

addition
multiplication

Taille de l'entrée

eval(P, x)

res = 0

n = deg(P)

pour $i = 0$ à n
 calcule $p[i] \times \boxed{X}$

$$v = 1$$

pour $j = 1$ à i

$$v = v \times X$$

res += $p[i] \times X^i$; $\rightarrow n+1$ fois

$$v = X^i$$

Renewer res

$$m = 1 + 2 + 3 + \dots + n - 1 + n$$

$$m = n + n - 1 + \dots + 2 + 1$$

$$2m = (n+1) + (n+1) - \dots + (n-1) + (n+1)$$

$$m = \frac{(n+1)n}{2}$$

Nombre de multiplications $\frac{(n+1)n}{2}$

$\Theta(n^2)$

$$\frac{n^2 + 3n + 2}{2}$$

Amélioration 1

Utiliser : exponentiel en binaire (Prod)
pour calculer X'

$$\sum_{i=1}^n 2 \log r = 2 \sum_{i=1}^n \log_i = 2 \log(1+2+2+\dots+n) \\ = 2 \log(n!)$$

$$O(n \cdot \log n)$$

$$(\log a) + \log(b) = \log(a \times b)$$