Algorithmique et Structures de Données

—

Corrigé de l'examen écrit

—

G1: pascal.monasse*(at)*enpc.fr G2: nicolas.audebert*(at)*lecnam.net
G3: julien.hauret*(at)*lecnam.net

28/03/2022

Les exercices sont indépendants. Il n'est pas interdit d'utiliser votre portable pour tester vos algorithmes, mais évidemment pas le wifi.

1 Suites de type Fibonacci

On s'intéresse aux suites du type $f_{n+2} = a\,f_{n+1} + b\,f_n$, $f_0 = f_1 = 1$, avec $a, b \in \mathbb{R}$, $b \neq 0$. On rappelle que pour la suite de Fibonacci ($a = b = 1$), on a $f_n \sim \varphi^n$ avec le nombre d'or $\varphi = (1+\sqrt{5})/2$. Quand il s'agira de complexité, on considère qu'une opération élémentaire est l'une des quatre opérations sur les nombres, ainsi qu'une fonction de la librairie mathématique, comme la fonction puissance `pow`.[1]

1. Écrire une fonction récursive C++ calculant le terme f_n.

```
float fib(int n, float a, float b) {
  if(n<2) return 1;
  return a*fib(n-1,a,b)+b*fib(n-2,a,b);
}
```

2. Quelle est sa complexité en fonction de n ? On a deux appels récursifs, avec en plus les 2 multiplications et 1 addition, soit $C_n = C_{n-1} + C_{n-2} + 3$, en notant $D_n = C_n + 3$, on trouve que D_n est la suite de Fibonacci, soit $D_n = O(\varphi^n)$, tout comme C_n, complexité exponentielle.
3. Écrire une fonction C++ analogue mais de complexité linéaire $O(n)$.

```
float fib(int n, float a, float b) {
  float fn_1=1, fn_2=1; // termes n-1 et n-2.
  for(int i=2; i<=n; i++) {
    fn_2 = a*fn_1 + b*fn_2;
    swap(fn_1,fn_2);
  }
  return fn_1;
}
```

4. On considère le vecteur $F_n = \begin{pmatrix} f_n \\ f_{n-1} \end{pmatrix}$ pour $n \geq 1$.
   (a) Écrire une relation de récurrence sous forme matricielle liant F_n et F_{n-1}.

$$F_n = \begin{pmatrix} a & b \\ 1 & 0 \end{pmatrix} F_{n-1}.$$

   (b) En déduire une formule exprimant F_n en fonction de n. $F_n = A^{n-1}F_1$ avec A la matrice ci-dessus.
5. On se base sur la formule précédente pour calculer plus vite.

1. Les trois premières questions sont un classique des entretiens techniques d'embauche, la suite est plus originale et permet de se démarquer.

(a) On décompose n en base 2 : $n = \sum_{i=0}^{m} \epsilon_i 2^i$ avec $\epsilon_m = 1$, les autres étant 0 ou 1. A désignant une matrice réelle 2×2, notant $A_i = A^{2^i}$, écrire A^n en fonction des A_i et ϵ_i.

$$A^n = A^{\sum_i \epsilon_i 2^i} = \Pi_i A^{\epsilon_i 2^i} = \Pi_i (A^{2^i})^{\epsilon_i} = \Pi_i A_i^{\epsilon_i}$$

(b) Écrire une relation de récurrence liant A_{i+1} à A_i. $A_{i+1} = A^{2^{i+1}} = A^{2^i 2} = (A^{2^i})^2 = A_i^2$.

(c) Utilisant les types `Imagine::Vector<Float>` et `Imagine::Matrix<float>`, écrire une fonction de complexité $O(\log n)$ calculant f_n. (Opérateurs utiles : `V[i]`, `M(i,j)`, `M*V` et `M1*M2`)

```
float fib(int n, float a, float b) {
  Vector<float> F(2); F[0]=F[1]=1;
  Matrix<float> Ai(2,2); Ai(0,0)=a; Ai(0,1)=b; Ai(1,0)=1; Ai(1,1)=0;
  n /= 2;
  while(n!=0) {
    if(n%2==1) // n%2 est le epsilon_i. Un pro ecrirait plutot if(n&1)
      F = Ai*F;
    Ai = Ai*Ai; // Formule de recurrence pour Ai
    n /=2; // Un pro ecrirait plutot n >>= 1
  }
  return F[0];
}
```

6. On suppose $a^2 + 4b > 0$. Expliquer comment procéder pour calculer f_n en $O(1)$ (sans faire le détail des calculs). Bonus : et si $a^2 + 4b < 0$? On peut dans ce cas diagonaliser A : son polynôme caractéristique est $X^2 - aX - b$ dont le discriminant est $a^2 + 4b$; étant positif, on a deux racines réelles distinctes et on peut écrire

$$A = P^{-1} \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} P \quad \text{et} \quad A^n = P^{-1} \begin{pmatrix} \lambda_1^n & 0 \\ 0 & \lambda_2^n \end{pmatrix} P.$$

Le A^{n-1} de la formule se calcule donc en diagonalisant A et en élevant les valeurs propres à la puissance n, tout est en $O(1)$.

Bonus : On fait de façon analogue, mais alors les valeurs propres sont complexes conjuguées λ et $\bar{\lambda}$, on diagonalise donc avec des matrices complexes. Pour calculer λ^n, on le met sous forme polaire $\lambda = r\cos\theta + i\, r \sin\theta$ et on a $\lambda^n = r^n cos(n\theta) + i\, r^n \sin(n\theta)$.

Vous vous demandez peut-être comment faire lorsque $a^2 + 4b = 0$. On peut traiter le cas en rusant. On a $f_{n+2} = a\, f_{n+1} - \frac{a^2}{4} f_n$. On considère

$$e_n = \left(\frac{2}{a}\right)^n f_n \quad \text{et il vient} \quad e_{n+2} = 2\, e_{n+1} - e_n.$$

On peut réécrire $e_{n+2} - e_{n+1} = e_{n+1} - e_n$ et on trouve donc $e_i - e_{i-1} = e_1 - e_0$. En sommant ces égalités pour $i = 1 \ldots n$, $e_n = e_0 + n(e_1 - e_0)$. En injectant les valeurs $e_0 = 1$ et $e_1 = 2/a$ on trouve

$$f_n = (1 - n)\left(\frac{a}{2}\right)^n + n\left(\frac{a}{2}\right)^{n-1}.$$

2 Multiplications

(algorithme de Karatsuba) Lors d'un séminaire, Kolmogorov affirma : "On ne peut pas multiplier deux nombres de n chiffres en moins de $O(n^2)$ opérations, sinon on aurait bien trouvé comment en 6000 ans." Insensible à cet argument d'autorité, un jeune étudiant vint le voir trois semaines plus tard pour lui montrer ce qu'il avait découvert et qu'on détaille dans cet exercice... Les opérations élémentaires sont ici les additions, soustractions et multiplications de nombres à un seul chiffre.

1. Montrer que les méthodes apprises à l'école primaire pour l'addition et la soustraction sont en $O(n)$. On ajoute chaque chiffre de b à celui correspondant de a (avec les restes) dans $a + b$, donc $O(n)$. De même pour la soustraction (avec les retenues).
2. Montrer que la complexité est $O(n^2)$ pour la multiplication. On multiplie d'abord chaque chiffre de a par un chiffre de b : $O(n)$ opérations (et on ajoute les restes). On doit le faire pour chaque chiffre de b ; donc $O(n^2)$. On a ensuite n nombres de n ou $n + 1$ chiffres à additionner, encore $O(n^2)$.

3. On suppose a et b de longueur $2n$. On décompose en $a = a_1 \times 10^n + a_2$ et $b = b_1 \times 10^n + b_2$ avec les a_i, b_i ayant chacun n chiffres. Notant M_n la complexité de la multiplication $a \times b$, montrer qu'on a besoin de mutliplier les a_i par les b_j et que donc $M_{2n} = 4\,M_n + O(n)$. A-t-on diminué la complexité ?

   $$a \times b = (a_1 \times 10^n + a_2)(b_1 \times 10^n + b_2) = a_1 b_1 10^{2n} + (a_1 b_2 + a_2 b_1)10^n + a_2 b_2.$$

   On a 4 multiplications (tous les $a_i b_j$) de nombres à n chiffres et des additions, donc $M_{2n} = 4\,M_n + Cn$ avec C constante. En déroulant,

   $$\begin{aligned} M_n &= 4M_{n/2} + Cn/2 = 4(4M_{n/4} + Cn/4) + Cn/2 = 4^2 M_{n/4} + C(1+2)n/2 \\ &= 4^3 M_{n/8} + Cn/2(1+2+2^2) = 4^4 M_{n/16} + Cn/2(1+2+2^2+2^3) = \cdots = 4^{\log_2 n} + Cn2^{\log_2 n} = O(n^2). \end{aligned}$$

4. Montrer qu'au prix de quelques additions ou soustractions en plus, on peut se contenter de 3 multiplications de nombres à n ou $n+1$ chiffres (on peut considérer $a_1 + a_2$ et $b_1 + b_2$). On peut calculer $\alpha = a_1 b_1$ et $\gamma = a_2 b_2$, ainsi que $\beta = (a_1 + a_2)(b_1 + b_2)$. On a alors $a_1 b_2 + a_2 b_1 = \beta - \alpha - \gamma$ et ne requiert pas de multiplication supplémentaire.
5. Montrer qu'on a alors $M_{2n} \leq 3\,M_n + C\,n$ avec C une constante. On a $2M_n + M_{n+1} + C_1 n$ opérations, car le calcul de β implique des nombres à $n+1$ chiffres potentiellement. Mais $M_{n+1} \leq M_n + C_2 n$, d'où le résultat avec $C = C_1 + C_2$.
6. En déduire que $M_n = O(n^{\log 3/\log 2}) = O(n^{1.6})$.

   $$\begin{aligned} M_n &\leq 3M_{n/2} + Cn/2 \leq 3(3M_{n/4} + Cn/4) + Cn/2 = 3^2 M_{n/4} + C(1+3/2)n/2 \\ &\leq 3^3 M_{n/8} + C(1+3/2+3^2/2^2)n/2 \leq \ldots \leq 3^{\log_2 n} + C(3/2)^{\log_2 n} 2^{\log_2 n} = O(3^{\log_2 n}). \end{aligned}$$

   On peut écrire $3 = 2^{\log_2 3}$ et donc $3^{\log_2 n} = 2^{\log_2 n \log_2 3} = (2^{\log_2 n})^{\log_2 3} = n^{\log_2 3} = n^{\log 3/\log 2}$.