Initiation à la Programmation en C (L1 CPEI)*

TP 5

20/02/2019

§1. Complexité

À noter : La complexité d'un algorithme (ou d'un programme) est une mesure de son efficacité. Il s'agit du nombre d'opérations de bases réalisées en fonction de la taille de l'entrée.

Exercice 1 (La bonne et la mauvaise exponentiation)**.** Dans cet exercice, l'opération de base qu'on utilisera est la multiplication de deux entiers. Le but est de trouver le meilleur algorithme pour calculer a^b à partir de $a, b \in \mathbb{N}$. On rappelle que $\log_2(n)$, le *logarithme binaire entier* de $n \in \mathbb{N}$, est le plus petit nombre $x \in \mathbb{N}$ tel que $2^x > n$. Cette fonction possède quelques propriétés intéressantes :
- il faut $\log_2(n)$ bits pour pouvoir écrire le nombre n en binaire ;
- $\log_2(n)$ est aussi le plus petit nombre x tel que diviser x fois n par 2 donne 0.

Voici maintenant les consignes de cet exercice :
(1) Écrire un algorithme qui calcule a^b en utilisant la relation de récurrence suivante

$$a^0 = 1, \qquad a^1 = a, \qquad a^{b+1} = a \times a^b.$$

Quelle est sa complexité ?
(2) Combien de multiplication fait l'algorithme pour calculer a^{16} ? Peut-on faire mieux? Trouvez une méthode pour calculer a^{16} en utilisant au plus 4 multiplications.
(3) Généralisez la méthode de la question précédente pour calculer a^b en utilisant $O(\log_2(b))$ multiplications, pour n'importe quels entiers $a, b \in \mathbb{N}$.

*Cours donné par prof. Roberto Amadio. Moniteur 2019 : Cédric Ho Thanh. TPs/TDs basés sur ceux des précédents moniteurs : Florien Bourse (2017), Antoine Dallon (2018). Autres contributeurs : Juliusz Chroboczek, Gabriel Radanne.

Exercice 2. Dans le TP2, exercice 3, nous avons écrit un programme qui devine l'age de l'utilisateur, entre 1 et 100. Considérons la généralisation du programme dans laquelle l'utilisateur pense à un nombre entre min et max inclus, pour des entiers $\mathsf{min} \leq \mathsf{max}$ donnés. Nous allons calculer la complexité de la recherche par dichotomie *dans le pire des cas.*

(1) Combien d'étapes faut-il pour trouver le nombre quand $\mathsf{min} = \mathsf{max}$, dans le pire des cas ?
(2) Combien d'étapes faut-il pour trouver le nombre quand $\mathsf{max} = \mathsf{min} + 1$, dans le pire des cas ?
(3) Si min et max sont fixés, combien de possibilités y-a-t-il ?
(4) À chaque étape de la recherche par dichotomie, de combien le nombre de possibilités réduit-il ?
(5) Déduire la complexité de la recherche par dichotomie dans le pire des cas. La comparer avec l'algorithme naïf qui demande tous les nombres 1 par 1.

Exercice 3 (Fibonacci par produit matriciel)**.** Soit la matrice

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}.$$

(1) Si $(f_n)_{n\in\mathbb{N}}$ est la suite de Fibonacci, montrez que pour tout $n \geq 2$ on a

$$\begin{pmatrix} f_n \\ f_{n-1} \end{pmatrix} = A \begin{pmatrix} f_{n-1} \\ f_{n-2} \end{pmatrix}.$$

(2) Quelle est la complexité du produit matriciel pour des matrices carrées de taille 2 ?
(3) Donnez un algorithme qui calcule A^n en $O(\log_2(n))$ opérations.
(4) En déduire un algorithme qui calcule f_n en $O(\log_2(n))$ opérations.

§2. Révisions pour le TP noté

Exercice 4. Refaites tous les exercices de tous les TPs et TDs, à l'exception du TP0. Vous pouvez utiliser tous les outils de programmation que vous avez vu jusqu'à maintenant, même pour les exercices des premières semaines.