info614 : Mathématiques pour l'informatique
TD 0 : les subtilités de la puissance et des polynômes...

Pierre Hyvernat
Laboratoire de mathématiques de l'université de Savoie
bâtiment Chablais, bureau 22, poste : 94 22
email : Pierre.Hyvernat@univ-savoie.fr
www : http://www.lama.univ-savoie.fr/~hyvernat/
wiki : http://www.lama.univ-savoie.fr/wiki

Partie 0 : calculs préliminaires instructifs

Question 1. À l'aide d'une calculatrice ou d'un ordinateur, remplissez le tableau suivant. La première colonne indique la complexité en microsecondes d'un algorithme pour une entrée de taille n. Pour chaque colonne, estimez le temps d'exécution de cet algorithme pour une entrée de la taille donnée.

	5	10	50	100	1000	10000
n	5×10^{-6} s	10^{-5} s	5×10^{-5} s	10^{-4} s	10^{-3} s	10^{-2} s
$n\log(n)$						
n^2						
n^3						
n^5						
2^n						
2^{2^n}						

Question 2. Quelles sont les classes de complexité "utilisables" ?

Si on estime que la loi de Moore est valide (la puissance de calcul double tous les deux ans), quelles sont les classes de complexité qui peuvent basculer du "infaisable" dans le "raisonnable" ?

Partie 1 : complexité de la puissance

Question 1. Programmez (en C par exemple) une fonction `puissance` qui prend en argument en entier `n` et un nombre `x` et qui calcule

$$\underbrace{\mathtt{x}\times\mathtt{x}\times\cdots\times\mathtt{x}}_{\mathtt{n}}$$

Question 2. En cryptographie, en a souvent besoin de calculer une puissance *modulo* un autre nombre. Modifiez votre fonction pour prendre un argument supplémentaire.

Question 3. Toujours en cryptographie, il arrive fréquemment que l'on ai besoin de calculer des puissances où `n` contient plusieurs centaines de chiffres.

Combien de multiplications sont utilisées par votre algorithme ? Qu'en pensez-vous ?

Question 4. La *méthode chinoise* pour calculer la puissance d'un nombre est basée sur les remarques suivantes :

$$\begin{array}{rcl} x^0 & = & 1 \\ x^{2n} & = & x^n \times x^n \\ x^{2n+1} & = & x^n \times x^n \times x \end{array}$$

Programmez une fonction `puissance_chinoise` récursive qui calcule $\texttt{x}^{\texttt{n}}$ mod `m`.

Question 5. Combien de multiplications sont utilisées par votre algorithme ? Qu'en pensez-vous ?

Question 6. La fonction `puissance_chinoise` est beaucoup plus efficace que la fonction `puissance`. Il reste le problème que les fonctions récursives consomment en général plus de mémoire que les fonctions purement itératives.

Pour transformer la fonction `puissance_chinoise` en fonction itérative, il faut utiliser un `accumulateur` spécial pour les multiplications par `x`.

Essayez de programmez la fonction correspondante.

Question 7. Combien de multiplications sont utilisées par notre algorithme pour le calcul de x^{15} ?

En utilisant le fait que $15 = 3 \times 5$, trouvez une méthode pour calculer x^{15} avec moins de multiplications que la méthode chinoise.

Est-ce que cette remarque vous permet de trouver une méthode plus efficace que la méthode chinoise ?

Partie 2 : subtilité des polynômes

Question 1. Combien de multiplications sont nécessaires pour évaluer un polynôme de degré n sur un entier ? (On suppose que tous les coefficients sont non-nul.)

Question 2. Quel est le nombre (exact) de multiplications et additions utilisées si on calcule les puissances "normalement", mais en réutilisant les calculs. (On suppose à nouveau que tous les coefficients sont non-nuls.)

Question 3. Même question si on utilise la règle de Horner :

$$c_0 + c_1 x + c_2 x^2 + \ldots + c_n x^n \quad = \quad \Big(\ldots \Big(\big((c_n x + c_{n-1}) x + c_{n-2} \big) x + c_{n-3} \Big) \ldots \Big) x + c_0$$

Question 4. Pensez-vous que l'on peut améliorer la fonction d'évaluation pour les polynômes *creux*. (Les polynômes où la plupart des coefficients sont nuls.)