Travaux Dirigés d'algorithmique n°3

Cours d'informatique de deuxième année

—DEUG Sciences S3-MI—

Récursivité, preuves d'algorithmes, complexité

Nous introduisons dans ce TD la notion de récursivité, nous prouverons la correction d'algorithmes récursifs et nous en calculerons la complexité (en temps).

▸ **Exercice 1.** *On souhaite ecrire en C une fonction qui prend en argument un entier positif n et qui calcule la somme des n premiers cubes. On remarque que, si $n > 0$,*

$$\sum_{i=0}^{n} i^3 = n^3 + \sum_{i=0}^{n-1} i^3 \ ,$$

c'est-à-dire que la somme des n premiers cubes s'écrit en fonction de la somme des $n-1$ premiers cubes. Écrire deux fonctions, une itérative et l'autre récursive, qui calculent la somme des n premiers cubes. En faire la preuve. Calculez les complexités, en temps et en espace.

▸ **Exercice 2.** *Si a est un entier et n un entier positif ou nul on se propose de calculer a^n en utilisant les trois propriétés suivantes :*

1. $a^n = \begin{cases} a * a^{n-1} & \text{si } n > 0, \\ 1 & \text{si } n = 0. \end{cases}$

2. $a^n = \begin{cases} (a^2)^{\lfloor n/2 \rfloor} & \text{si } n > 0 \text{ est pair}, \\ a * (a^2)^{\lfloor n/2 \rfloor} & \text{si } n \text{ est impair}, \\ 1 & \text{si } n = 0. \end{cases}$

3. $a^n = \begin{cases} (a^{\lfloor n/2 \rfloor})^2 & \text{si } n > 0 \text{ est pair}, \\ a * (a^{\lfloor n/2 \rfloor})^2 & \text{si } n \text{ est impair}, \\ 1 & \text{si } n = 0. \end{cases}$

Écrire, pour chacune des trois propriétés susnommées une fonction C qui retourne a^n en utilisant cette propriété. Quelle est la plus efficace des trois en terme de temps de calcul ? Faire la preuve de la dernière.

Recherche dans un tableau trié

Dans les exercices suivants, t est un tableau d'entiers trié pour l'ordre croissant et x est un entier.

▸ **Exercice 3.** *Écrire une fonction itérative qui recherche si x est présent dans le tableau t. Évaluer la complexité de cette fonction en nombre de comparaisons. Quel est l'invariant de boucle ?*

▸ **Exercice 4.** *L'idée de la recherche dichotomique (qui ne peut s'appliquer qu'à un tableau trié) est la suivante : on compare x avec l'élément du milieu du tableau. Si x est présent dans le tableau alors, s'il est plus grand que cet élément du milieu, il se trouve nécessairement dans la deuxiéme moitié du tableau sinon il se trouve nécessairement dans la première moitié du tableau. Il suffit alors de recommencer la recherche sur une seule des deux moitiés.*

- *Quel est le critère d'arrêt dans la méthode décrite ci-dessus ? Écrire une fonction récursive qui recherche si x est présent dans le tableau en utilisant cette méthode.*
- *Évaluer la complexité de cette fonction en nombre de comparaisons et montrer qu'elle est correcte.*
- *Écrire une version itérative de cette fonction. Quel est l'invariant de boucle ? Quelle est la complexité ?*