Travaux Dirigés d'algorithmique n°3

Cours d'Informatique de Deuxième Année

—DEUG Sciences S3-MI—

Récursivité

La récursivité est une méthode de programmation très puissante. Elle permet d'écrire des programmes faciles à comprendre et souvent efficaces bien que son principal inconvénient est d'obliger le compilateur à utiliser une pile pour mémoriser les calculs intermédiaires. Nous l'introduisons ici par des exemples simples.
Nous abordons également la notion de complexité d'algorithme, c'est-à-dire le temps (et parfois l'espace) nécessaire à son exécution.

▸ **Exercice 1. (fonction factorielle)** *Nous rappelons la définition récursive de la fonction* `factorielle` *:*

$$n\,! := \begin{cases} 1 & \text{si } n = 0, \\ n \times (n-1)\,! & \text{si } n > 0. \end{cases}$$

1. *Donner une fonction révursive* `Fact` *qui calcule la factorielle d'un entier* `n` *passé en argument ;*
2. *évaluer la complexité en temps de votre fonction.*

▸ **Exercice 2. (Plus Grand Commun Diviseur)** *Dans cet exercice, nous nous proposons de calculer le plus grand commun diviseur (**PGCD**) de deux nombres entiers (on suppose qu'ils ne sont pas tous les deux nuls) par l'algorithme d'Euclide :*

$$PGCD(a,b) = \begin{cases} a & \text{si } b = 0, \\ PGCD(b, a \bmod b) & \text{sinon.} \end{cases}$$

1. *calculer le PGCD de* 16 *et* 6 *;*
2. *écrire une fonction récursive qui calcule le **PGCD** de deux entiers (supposés non nul).*

▸ **Exercice 3. (une multiplication amusante)** *Dans cet exercice, nous cherchons à réaliser la multiplication de deux entiers* `n` *et* `m`*. Malheureusement, on ne dispose que de la division par deux et de l'addition... Cependant, on peut remarquer ceci :*

$$\begin{aligned} 0 * m &= 0 && \text{pour tout } m, \\ n * m &= (n/2) * (m+m) && \text{si } n \text{ est pair}, \\ n * m &= (n-1) * m + m && \text{si } n \text{ est impair.} \end{aligned} \tag{1}$$

- *Déduire des relations (**??**) un algorithme pour réaliser la multiplication de deux entiers ;*
- *en déduire une fonction* `Multiplication` *prenant en arguments deux entiers* `n` *et* `m` *et qui implémente l'algorithme précédant ;*
- *évaluer la complexité de la fonction* `Multiplication`*.*

▸ **Exercice 4.** *On définit les nombres de Fibonacci par :*

$$\begin{cases} F(0) = 0 \ , \qquad F(1) = 1 \ , \\ F(n) = F(n-1) + F(n-2) \quad \textit{si } n \geq 2 \ . \end{cases}$$

1. *Calculer F(5) ;*
2. *programmer de façon récursive, la fonction* `FiboR` *qui retourne le nième nombre de Fibonacci ;*
3. *quel est le défaut de cette méthode ?*
4. *donner une nouvelle fonction* `FiboIt` *qui calcule le nième nombre de Fibonnaci de façon itérative.*

▸ **Exercice 5.** *(on supposera que n est un puissance de 2.)*

1. *Que fait la fonction suivante ? (le démontrer),*

```
int f(int n)
{
  if (n == 0) return 1;
         else return(f(n/2)+n);
}
```

2. *évaluer sa complexité.*