Travaux Dirigés d'algorithmique n°1

Cours d'informatique de Deuxième Année

—DEUG Sciences S3-MI—

Récursivité, Complexité

Nous introduisons dans ce TD, au travers d'exemples simples, les notions de récursivité, de complexité (en temps et en espace) d'un algorithme. Nous parlerons également de preuves d'algorithmes.

▸ **Exercice 1.** *La fonction* factorielle *d'un entier positif ou nul n, notée $n!$, est définie par*

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

Écrire une fonction C récursive, et une autre itérative, qui retourne la factorielle de son argument. Calculer le temps nécessaire à chacune des deux fonctions pour calculer $n!$. Calculer la place mémoire nécessaire à chacune des deux fonctions pour calculer $n!$. Prouver que les fonctions écrites retournent bien $n!$.

▸ **Exercice 2.** *Soient a 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. Évaluer la complexité en temps de chacune de ces fonctions.

▸ **Exercice 3.** *On définit les nombres de Fibonacci par :*

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

1. *Calculer F(5) ;*
2. *Donner une fonction récursive,* `FiboR`*, qui retourne le nième nombre de Fibonacci et évaluer sa complexité en temps et en espace ;*
3. *quel est le défaut de cette méthode ?*
4. *donner une nouvelle fonction,* `FiboIt`*, qui calcule le nième nombre de Fibonacci de façon itérative. Évaluer la complexité de cette fonction et montrer qu'elle est correcte.*

▸ **Exercice 4.** *Que fait le programme suivant ?*

```
#include <stdio.h>

void bizarre(void)
{
  char c;

  c=getchar();
  if (c=='\n')
      printf("\n");
  else
      { bizarre(); printf("%c", c); }
}

int main(void)
{
  bizarre(); printf("\n");
  return 0;
}
```