Travaux Pratiques d'algorithmique n°3

Cours d'Informatique de Deuxième Année

—DEUG Sciences S3-MI—

Récursivité

L'objet de cette séance est d'introduire la notion de récursivité. Dans un premier temps, on s'intéressera à plusieurs exemples basiques qui permettent de se familiariser avec cette technique. Ensuite, nous appliquerons cette méthode à des problèmes plus complexes. Vous devrez présenter clairement les algorithmes, ainsi que **le calcul de leur complexité**.

▸ **Exercice 1.** ***(Algorithmes classiques)*** *Écrire les fonctions récursives suivantes :*

- *calcul de la factorielle d'un entier passé en paramètre. Calculer sa complexité.*

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

- *calcul du n^{ieme} nombre de Fibonacci. Compter le nombre d'appels de fonction pour* `n` $=$ 4. *Que vous permet de conclure ce calcul ?*

$$\begin{cases} F_0 &= 0 \\ F_1 &= 1 \\ F_n &= F_{n-1} + F_{n-2} \end{cases}$$

- *calcul du coefficient du binôme pour deux entiers passés en paramètres.*

$$C_n^p = \begin{cases} 0 & \text{si } n < p \\ 1 & \text{si } p = 0 \text{ ou } p = n \\ C_{n-1}^{p-1} + C_{n-1}^p & \text{si } 0 < p < n \end{cases}$$

- *calcul du PGCD de deux entiers par la méthode d'Euclide.*

$$Si\ a \geq b\ > 0\ \ PGCD(a,b) = \begin{cases} (\text{Soient q, r tels que} & a = qb + r) \\ b & \text{si } r = 0 \\ PGCD(b,r) & \text{sinon} \end{cases}$$

▸ **Exercice 2.**

Écrire une fonction de prototype `void AffDecomp(int n, int b)` *qui affiche la décomposition de l'entier* `n` *dans la base* `b`. *Par exemple, pour* `n`$=$ 21 *et* `b`$=$ 8, *l'affichage sera* 25. *Pour cela, vous utiliserez un algorithme* ***récursif*** *dont vous estimerez la complexité.*

▸ **Exercice 3.**

Écrire une fonction `Puissance` *en appliquant cette définition :* $b^0 = 1$, $b^n = b\, b^{n-1}$.

Écrire une fonction `PuissanceBis` *qui tient compte des relations suivantes :*

$$b^n = \begin{cases} 1 & \text{si } n = 0 \\ (b^{n/2})^2 & \text{si } n \text{ est pair} \\ b\,(b^{(n-1)/2})^2 & \text{si } n \text{ est impair} \end{cases}$$

Évaluer la complexité dans les deux cas.

▸ **Exercice 4.** ***(Recherche d'un élément dans un tableau trié)***

Écrire une fonction récursive `RechercheRec` *qui prend en argument un tableau* `tab` *de réels, sa taille* `n` *et un élément* `x` *et qui renvoie la position de* `x` *dans* `tab`*, ou* -1 *si* `x` *n'est pas présent dans le tableau. La fonction devra utiliser la méthode* ***dichotomique*** *et être* ***récursive****. Évaluer sa complexité.*

▸ **Exercice 5.** ***(Somme des chiffres d'un entier)***

- *On veut calculer la somme des chiffres d'un entier. Par exemple, la somme des chiffres formant* 3281 *est* 14.

  Écrire une fonction `SommeChiffre` *prenant en argument un entier* `n` *et qui calcule récursivement la somme des chiffres qui le forment.*

- *Le critère de divisibilité suivant est connu de tous : un entier est divisible par* 3 *si la somme des chiffres qui le forment est un multiple de* 3*. Autrement dit :*

$$n \equiv 0\ [3] \Leftrightarrow \sum_{i=0}^{k} c_i \equiv 0\ [3]$$

  On remarquera que la somme des chiffres d'un entier supérieur à 10 *est toujours strictement plus petit que cet entier. Cette simplification (passer d'un entier à la somme de ses chiffres) fait naître un algorithme récursif.*

  Un entier `n` *est-il divisible par* 3 *? Si l'entier est strictement plus petit que* 10 *: facile à déterminer ; sinon on se pose la même question sur la somme de ses chiffres.*

  Effectuer le calcul complet pour 3281 *et* 1847394*.*

  Écrire une fonction récursive réalisant cet algorithme.

- *Effectuer le même travail pour la divisibilité par* 11 *:*

$$n \equiv 0\ [11] \Leftrightarrow \sum_{i=0}^{k} (-1)^i c_i \equiv 0\ [11]$$

  On utilisera une nouvelle fonction `SommeAlternée`*.*