Algorithmique et Programmation 1

TD6 : Récusivité

1 Pour commencer

1. Écrivez une fonction récursive `puissance(x,n)`, qui calcule $x^n = \underbrace{x \times x \times \ldots \times x}_{n}$, en n'utilisant que des multiplications (sans utiliser le symbole $**$).
2. Écrivez une fonction récursive `produit(x,n)`, qui calcule $x * n = \underbrace{x + x + \ldots + x}_{n}$, en n'utilisant que des additions (sans utiliser le symbole $*$).
3. Écrivez une fonction récursive `factoriel(n)` qui calcule $n! = 1 \times 2 \times \ldots \times n$.
4. Écrivez une fonction récursive qui calcule le pgcd (plus grand diviseur commun) de deux nombres positifs a et b, en appliquant la formule :
$$pgcd(a,b) = \begin{cases} a & \text{si } b = 0 \\ pgcd(b, a \bmod b) & \text{sinon} \end{cases}$$
Déroulez l'exécution de cette fonction pour $a = 60$ et $b = 100$.
5. Écrivez une fonction récursive `binomial(n,p)` qui calcule $\binom{n}{p}$ pour tout p, n entiers positifs, avec $p \leq n$.
Rappel : $\binom{n}{p}$ est le nombre de combinaisons différentes de p éléments parmi n. Ce sont les nombres qui se trouvent dans le triangle de Pascal. La définition mathématique récursice (qui utilise la propriété du triangle de Pascal : chaque entier est la sommes des deux entiers adjacents de la ligne du dessus) est la suivante :
$$\binom{n}{p} = \begin{cases} 1 & \text{si } p = 0 \text{ ou } p = n \\ \binom{n-1}{p} + \binom{n-1}{p-1} & \text{sinon} \end{cases}$$

2 Amélioration de l'efficacité

1. Quel est le nombre de multiplications qu'effectue votre fonction `puissance(x,n)` définie à l'exercice précédent ? Réécrivez cette fonction, en utilisant le fait que
$$x^n = \begin{cases} \left(x^{n/2}\right)^2 & \text{si } n \text{ est pair} \\ x * \left(x^{\lfloor n/2 \rfloor}\right)^2 & \text{si } n \text{ est impair} \end{cases}$$
Quel est le nombre de multiplications qu'effectue maintenant `puissance(x,n)` quand $n = 2^k$?
2. Faites de même pour la fonction `produit`.

3 Fibonacci

Les nombres de Fibonacci F_n sont définis par la formule de récurrence d'ordre 2 suivante :

$$F_0 = 1, \quad F_1 = 1, \quad \forall n \geq 2, \quad F_n = F_{n-1} + F_{n-2}$$

1. Écrivez une fonction récursive `fibonacci(n)` qui calcule (naïvement) la valeur de F_n.
L'implémentation naïve de la fonction `fibonacci(n)` fait un appel récursif à `fibonacci(n-1)`, et un autre à `fibonacci(n-2)`. Mais `fibonacci(n-1)` fait elle-même aussi un appel récursif à `fibonacci(n-2)` (et un autre à `fibonacci(n-3)`). Un certain nombre de calculs sont effectués plusieurs fois, ce qui fait rapidement exploser le temps de calcul de `fibonacci(n)` : il est exponentiel en n.
Pour pallier cela, on regroupe les nombres de fibonacci par paires (F_n, F_{n+1}).
2. Écrivez une fonction récursive `fibonacciCouple(n)` qui renvoie le couple (F_n, F_{n+1}). On ne veut qu'un seul appel récursif au sein du corps de la fonction `fibonacciCouple(n)`.
3. Écrivez une fonction wraper `fibonacci2(n)` qui utilise la fonction récursive `fibonacciCouple` pour calculer la valeur de F_n.