# Option Informatique

## TP $n°3$ : récursivité, itération

Note : même lorsque ce n'est pas demandé, écrire des fonctions auxilliaires n'est jamais une mauvaise idée.

**Exercice -1.** Binomiaux (bis).

On rappelle que les coefficients binomiaux $\begin{pmatrix} n \\ k \end{pmatrix}$ vérifient les propriétés suivantes :

- $\forall n \in \mathbb{N}, \begin{pmatrix} n \\ 0 \end{pmatrix} = 1$ ;
- $\forall k > 0, \begin{pmatrix} 0 \\ k \end{pmatrix} = 0$ ;
- $\forall (n,k) \in \mathbb{N}^2, \begin{pmatrix} n+1 \\ k+1 \end{pmatrix} = \begin{pmatrix} n \\ k+1 \end{pmatrix} + \begin{pmatrix} n \\ k \end{pmatrix}$.

On a déjà écrit une fonction récursive permettant de calculer les binomiaux à l'aide de ces relations.

On souhaite écrire une fonction **itérative** qui prend en entrée deux arguments $n$ et $k$ et rend en sortie $\begin{pmatrix} n \\ k \end{pmatrix}$.

On va donc utiliser la relation de récurrence rappelée ci-dessus et une matrice (= tableau de tableau) d'entiers permettant de stocker successivement les valeurs des $\binom{j}{i}$ avec $(i,j) \leqslant_{lex} (n,k)$ où $\leqslant_{lex}$ est l'ordre lexicographique.

1. Pourquoi faut-il ne surtout pas utiliser la syntaxe `Array.make k (Array.make n 0)` pour initialiser la matrice ?
2. Écrire la version itérative.
3. Quelle fonction est la plus rapide ? La récursive ou l'itérative ? Utiliser Sys.time et proposer une explication.

**Exercice 0.** Exponentiation rapide.

Traitons cet exercice si (ou plutôt car) on n'a pas eu le temps de le corriger en cours. On considère la fonction suivante.

```
let rec f x n = match n with
        | 0 -> 1
        | n when n mod 2 = 0 -> let a = f x (n/2) in a*a
        | n -> let a = f x (n/2) in x*a*a;;
```

1. Que permet-elle de calculer ?
2. Le démontrer.
3. Montrer qu'elle termine.
4. Quel est son intérêt par rapport à celle vue en DS ?

**Exercice 1.** Fibonacci en récursivité terminale.

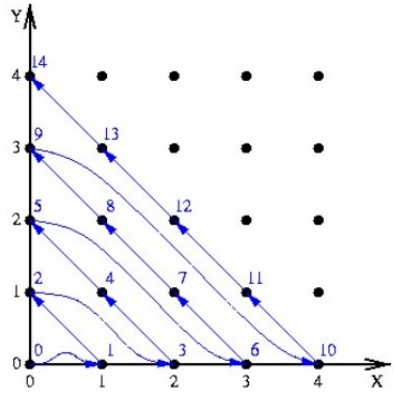
Rappel : on peut proposer la version suivante de la factorielle, en récursivité terminale :

```
let factorielle n =
        let rec aux n accu =
                match n with
                |0 -> accu
                |_ -> aux (n-1) (n*accu) in
        aux n 1;;
```

1. Proposer une fonction calculant le $n^{\text{ième}}$ terme de la suite de Fibonacci en utilisant la récursivité terminale.
2. Comparer la vitesse d'exécution de votre fonction avec celle de la version enveloppée. Utiliser la fonction Sys.time.
3. Les différences de vitesse d'exécution ne s'expliquent **pas** ici par la pile d'exécution. Comment s'expliquent-elles ?

**Exercice 2.** Bijection $\mathbb{N}^2 \simeq \mathbb{N}$.

On sumérote chaque point du plan de coordonnées $(x, y) \in \mathbb{N}^2$ par le procédé suggéré sur la figure ci-dessous :




1. Écrire une fonction récursive `numero x y` qui retourne le numéro du point de coordonnées $(x, y)$.
2. Justifier sa terminaison.

*HS : on peut trouver une formule explicite pour* `numero x y`*. Sauriez-vous le faire ?*

**Exercice bonus** seulement pour les plus rapides (s'ils existent).

Dans cet exercice, on appelle *mot binaire de longueur* $n$ une chaine de $n$ caractères composée exclusivement de '0' et de '1'. Il y en a donc $2^n$ (pourquoi ?).

1. Écrire une fonction récursive qui prend comme argument un entier $n$ et rend comme résultat la liste de tous les mots binaires de longueur $n$.
2. Écrire une fonction itérative qui prend comme argument un entier $n$ et rend comme résultat la liste de tous les mots binaires de longueur $n$. La structure de donnée liste étant plus adaptée aux traitements récursifs, on pourra si on préfère retourner le tableau de tous les mots binaires de longueur n.