# Devoir Maison 2

Pour le 6 janvier 2015

**Consignes.** *Testez vos fonctions en machine. Vous restituez une copie manuscrite où les fonctions sont rédigées proprement à la main.*

## Exercices. Tableaux binaires

À un niveau plus ou moins fondamental l'information est traitée sous forme binaire. L'étude des structures de données constituées uniquement de `0` et de `1` joue donc un rôle important en algorithmique.

On a l'exemple des *QR codes* dont la structure s'apparente à un tableau constitué de 0 (carrés blancs) et de 1 (carrés noirs).

Dans cet exercice les structures de données étudiées sont des tableaux carrés dont les coefficients sont uniquement des 0 et des 1 (on parlera de tableaux binaires). En Python on pourra implanter un tel tableau comme une liste de listes (liste des lignes).

**Par exemple.** L'objet `T = [[1,0,1,1],[0,0,0,0],[1,1,0,0],[0,0,0,1]]` représente le tableau :

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

L'accès à l'élément de la ligne $i$ et colonne $j$ s'obtient avec la syntaxe `T[i][j]`.

1. Fixons $N \in \mathbb{N}^*$. Combien existe-t-il de tableaux binaires distincts de taille $N \times N$ ? Donner l'ordre de grandeur (en Go) de l'espace mémoire nécessaire pour stocker tous les tableaux binaires de taille $6 \times 6$.

On dit qu'un tableau binaire est *équilibrée* lorsqu'il y a autant de 0 que de 1. On dit qu'un tableau binaire est *totalement déséquilibrée* lorsqu'aucun des sous-tableaux carrés que l'on peut former contiguement en partant du coin supérieur gauche n'est *équilibré*.

**Exemple.** Le tableau $T_e$ est équilibré ; le tableau $T_d$ est *totalement déséquilibré* ; le tableau $T_r$ n'est ni l'un ni l'autre :

$$T_e = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \;;\; T_d = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \;;\; T_r = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Pour $T_d$ on vérifie en effet qu'aucun des sous-tableaux suivants n'est équilibré :

$$(1) \;;\; \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \;;\; \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \;;\; \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Pour simplifier le nombre de lignes (et donc de colonnes) des tableaux binaires est désormais supposé pair.

1. Définir une fonction Python `compt0(T)` dont l'argument d'entrée est un tableau binaire `T`. La fonction retourne le nombre de 0 dans le tableau.
2. Définir une fonction Python `equilibre(T)` dont l'argument d'entrée est un tableau binaire `T`. La fonction retourne `True` si la liste est *équilibrée* et `False` sinon. Majorer le nombre de tests d'égalité et d'opérations arithmétiques pour une entrée de taille $N \times N$.

Pour un tableau binaire $T$ on définit la mesure d'équilibre $\texttt{mes}(T)$ comme étant le plus grand entier $k$, s'il existe, tel que le sous-tableau carré de $T$ de taille $2k \times 2k$ extrait en partant du coin supérieur gauche est *équilibrée* ; si un tel entier n'existe pas on pose $\texttt{mes}(T) = 0$.

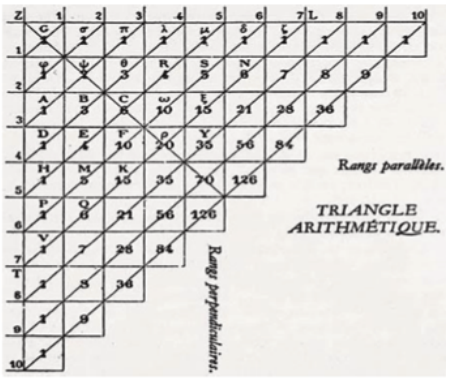
**Exemples.** $\texttt{mes}(T_e) = 2$, $\texttt{mes}(T_d) = 0$ et $\texttt{mes}(T_r) = 1$.

3. Définir en Python la fonction $\texttt{mes}(T)$. Expliquer de manière concise votre démarche. Tester la fonction sur les tableaux $T_e$, $T_d$ et $T_r$. De quelle méthode dispose-t-on désormais pour tester si un tableau binaire est *totalement déséquilibré*.

4. **Simulations et estimations de fréquences.** *Dans cette question on n'exige pas de transcription manuscrite du code Python.*
   (a) À l'aide de la fonction `randrange` du module `random` réaliser une fonction `alea(N)` qui retourne un tableau binaire de taille $N \times N$ dont les coefficients 0 ou 1 sont pris aléatoirement.
   (b) Pour $N = 4, 6, 8 \cdots$ estimer les proportions de tableaux *équilibrés* et *totalement déséquilibrés* en comptant le nombre d'occurrences de tels tableaux sur un grand nombre de tableau pris au hasard ; les estimations seront réalisés à l'aide un échantillon suffisamment grand (au moins 10000 tableaux). L'étude statistique pourra être complétée par les estimations des fréquences d'apparition de chaque mesure d'équilibre ainsi qu'une discussion les valeurs théorique attendues.

# Problème. Factorielle et coefficients binomiaux

La fonction factorielle et les coefficients binomiaux jouent un rôle majeur en analyse combinatoire, en statistique ainsi que dans de nombreuses autres parties des mathématiques ou de l'informatique théorique.
Ce problème aborde des questions relatives aux calculs effectifs de ces nombres.




## Partie 1. Factorielle et récursivité terminale

1. Rappeler la définition récursive vue en cours pour la fonction calculant la factorielle d'un entier.

On propose la fonction ci-dessous :

```
def F(acc,n):
    if n==0:
        return 1
    elif n==1:
        return acc
    else:
        return F(acc*n,n-1)
```

2. Donner l'ensemble des appels récursifs lors de l'appel de `F(1,6)`. Que retourne la fonction ?

La fonction ci-dessus est *récursive terminale* car la dernière instruction effectuée avant de quitter la fonction est un appel récursif seul qui pas pris en argument d'une fonction ou d'une opération. Ce mode de rédaction permet une économie des ressources mémoires.

## Partie 2. Calcul d'un coefficient binomial

Le calcul récursif des coefficients binomiaux a été étudié en TP. On se concentre dans la suite du problème sur des calculs itératifs.

3. On considère l'algorithme suivant :

| **Algorithme**. |
|---|
| **Entrée :** un entier $n$ |
| **Sortie :** l'entier $n!$ |
| $f = 1$ |
| **Pour** $i = 1$ **à** $n$ : |
| $\quad f = f * i$ |
| **Retourner** $f$ |

(a) Suivre l'état des variables `i` et `f` pour l'entrée $n = 6$.

(b) Déterminer un *invariant de boucle* permettant de justifier l'algorithme ci-dessus.

(c) Calculer le *nombre de multiplications* d'entiers pour une entrée $n$.

(d) Définir une fonction Pyhton `factorielle(n)` qui reprend l'algorithme ci-dessus. Compléter votre fonction pour que `factorielle(0)` retourne 1.

4. Voici une fonction `binomial1(n,k)` pour le calcul des coefficients binomiaux :

| |
|---|
| **Fonction :** binomial1$(n, k)$. |
| **Entrées :** deux entiers $n$ et $k$ avec $n \geq k \geq 0$ |
| **Sortie :** l'entier $\binom{n}{k}$ |
| **Retourner** factorielle$(n)$//(factorielle$(k)$ $*$ factorielle$(n-k)$) |

Calculer le nombre de multiplications d'entiers de la fonction `binomial` pour une entrée $n, k$. En déduire que la complexité en nombre de multiplications est en $\mathcal{O}(n)$.

5. On donne les identités suivantes :

$$\binom{n}{k} = \frac{\prod_{i=0}^{k-1} n-i}{\prod_{i=0}^{k-1} i+1} \quad \text{et} \quad \binom{n}{k} = \binom{n}{n-k}$$

(a) En déduire une fonction `binomial2(n,k)` pour le calcul des coefficients binomiaux nécessitant *moins* de multiplications que l'algorithme précédent. Majorer le nombre de multiplications d'entiers.

(b) Implanter les fonctions `binomial1(n,k)` et `binomial2(n,k)` en Python et comparer leur vitesse à l'aide de la fonction `time` étudiée en cours. On réalisera un tableau comparatif pour plusieurs grandes valeurs de $n$ et $k$ afin de mettre en évidence une différence significative entre les fonctions.

6. Remarquant que $\binom{n}{k} = \prod_{i=0}^{k-1} \frac{n-i}{i+1}$ un élève définit alors la fonction `binomial3` suivante :

```
def : binomial3(n,k) :
prod=1
for i in range(0,k) :
    prod=prod*(n-i)/(i+1)
return int(prod)
```

Il obtient alors les résultats contradictoires :

```
>>> binomial2(59,22)
8964377427999630
>>> binomial3(59,22)
8964377427999631
```

(a) En remarquant que 59 est un nombre premier trouver un argument permettant d'affirmer que le résultat de `binomial3` est faux. Comment expliquer cette erreur ?

(b) Corriger la section 11 de la page Wikipédia à l'adresse [1] :

`en.wikipedia.org/wiki/Binomial_coefficient`

## Partie 3. Calcul de tous les coefficients binomiaux

De nombreux problèmes nécessitent d'avoir accès à tous les coefficients binomiaux (ou au moins ceux d'une ligne du triangle de Pascal).

7. On fixe un entier $N$. Par un calcul direct utilisant une des deux fonctions `binomial` de la partie précédente donner le nombre de calcul nécessaires à l'obtention de tous les coefficients binomiaux $\binom{N}{k}$ pour $k$ allant de 0 à $N$. Comment qualifier la complexité d'une telle méthode ?

1. À l'heure de la rédaction de ce DM il y a bien une erreur.

Dans ce genre de situation il est plus sage de faire appel à la relation récurrence (ou relation de Pascal) :

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

On veut concevoir une fonction Python `tableauBinomial(N)` retournant la liste des listes $\left[\binom{n}{k} \text{ pour } k = 0 \cdots N\right]$ avec $n$ variant entre 0 et $N$. Ainsi si `T=tableauBinomial(N)` alors `T[n,k]` $=\binom{n}{k}$.
Par exemple pour $N = 3$ on doit trouver la liste :

```
[[1, 0, 0, 0], [1, 1, 0, 0], [1, 2, 1, 0], [1, 3, 3, 1]]
```

8. Rédiger la fonction `tableauBinomial(N)`. Vous pouvez (ou pas) suivre la démarche indiquée ci-dessous :
   - On initialise une variable locale `tableau = [[0 for j in range(0,N+1)] for i in range(0,N+1) ]`.
   - On remplit la *première colonne* par les affectations `tableau[i][0]=1`.
   - On remplit itérativement le reste du tableau en utilisant la relation de Pascal et les affectations

     `tableau[i][j]= tableau[i-1][j] + tableau[i-1][j-1].`

   Tester la fonction pour plusieurs valeurs de $n$.
9. Calculer le nombre d'opérations arithmétiques (additions) pour l'entrée `N`.

Pour un affichage plus sympathique, passer la liste des coefficients binomiaux en argument de la procédure donnée ci-dessous.

```
def affiche(T):
    tab=''
    for ligne in T:
        for coeff in ligne:
            tab=tab+' '+str(coeff)+' '
        tab=tab+'\n'
    print(tab)
```

10. Modifier la procédure `affiche(T)` pour qu'elle affiche le caractère `*` si le coefficient binomial est impair et le caractère *espace* sinon. *Dans cette question on n'exige pas de transcription manuscrite du code Python.*