Devoir en temps libre d'informatique n° 01

Vous numéroterez vos copies et ferez apparaître clairement sur la première page le nombre de copies. Vous prêterez une attention particulière **au soin** de vos copies.
Si vous ave accès à un ordinateur muni de Python (environnement de base ou plus évolué), n'hésitez pas à tester vos fonctions.

Exercice 1 : Tableaux binaires

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 considérer un tel tableau comme une liste de listes .
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 de la colonne j s'obtient par la syntaxe : $T[i][j]$.

1. . Soit $N \in \mathbb{N}^*$. Combien existe-t-il de tableaux binaires distincts de taille $N \times N$? En considérant que les entiers sont codés sur 32 bits, donner l'ordre de grandeur (en Go) de l'espace mémoire nécessaire pour stocker tous les tableaux de taille 6×6.

   On dit qu'un tableau binaire est *équilibré* lorsqu'il y a autant de 0 que de 1. On dit qu'un tableau binaire est *totalement déséquilibré* lorsqu'aucun sous-tableau carré que l'on peut former de façon contigue en partant du coin supérieur gauche n'est *équilibré*.

   Exemple. Parmi les tableaux suivants, T_e est équilibré, T_d est totalement déséquilibré et le tableau T_r n'est ni l'un ni l'autre :

$$T_e = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \quad , \quad T_d = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{et} \quad T_r = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

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

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

   Pour simplifier, on supposera désormais que le nombre de lignes (et donc de colonnes) est pair.

2. Définir une fonction Python **compt0(T)** dont l'argument est un tableau binaire T et qui retourne le nombre de 0 dans le tableau.

3. Définir une fonction Python **equilibre(T)** dont l'argument est un tableau binaire T et qui retourne le booléen **True** si le tableau T est équilibré et **False** sinon.

   Pour un tableau binaire T, on définit la mesure d'équilibre **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 équilibré ; si un tel entier n'existe pas, on pose mes(T) $= 0$

   Exemple. $mes(T_e) = 2, mes(T_d) = 0 et mes(T_r) = 1$.

4. Définir une fonction Python **mes(T)** qui réalise le travail demandé.

5. Définir une fonction Python **desequilibre(T)** dont l'argument est un tableau binaire T et qui retourne le booléen **True** si le tableau T est totalement déséquilibré et **False** sinon.

6. **Simulations et estimations de fréquences**.

(a) A l'aide de la fonction **randrange** du module **random** et qui donne un nombre entier aléatoire dans l'intervalle entier $[a, b[$, écrire une fonction **alea(N)** qui retourne un tableau binaire $N \times N$ dont les coefficients 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 tableaux pris au hasard ; les estimations seront réalisś à l'aide d'un échantillon suffisamment grand (au moins 1000 tableaux). Quelle est la fréquence théorique d'apparition d'un tableau binaire équilibré de taille $N \times N$?

Problême : Factorielle et coeffciients binomiaux

0.1 Partie I : Factorielle

On considère l'algorithme suivant

```
>>> f = 1
>>> for i in range(1, n+1):    # n \'etant un entier naturel donné
>>>         f = f * i
```

1. Suivre l'état des variables i et f dans cet algorithme pour l'entrée $n = 6$.
2. Déterminer un *invariant de boucle* permettant de justifier que l'algorithme précédent retourne bien $n!$ lorsque n est un entier strictement positif
3. Calculer le *nombre de multiplications* d'entiers pour une entrée n donnée
4. Écrire une fonction Python **factorielle(n)** qui reprend l'algorithme ci-dessus. On veillera cependant à ce que l'appel factorielle(0) retourne 1.

0.2 Partie II : Calcul d'un coefficient binomial

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

```
>>> def binomial1(n, k):
>>>         return (factorielle(n) // (factorielle(k) * factorielle(n-k))
```

   Calculer le nombre de multiplications d'entiers de l'appel de la fonction **binomial1(n,k)**. En déduire que la complexité en nombre de multiplications est en $O(n)$

2. (a) Question mathématique Montrer les identités suivantes :

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

   (b) 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.

   (c) Implémenter les fonctions **binomial1(n,k)** et **binomial2(n,k)** en Python et comparer leur vitesse à l'aide de la fonction **time** du module **time** vue en TP. On pourra réaliser un tableau comparatif pour plusieurs grandes valeurs de n et k afin de mettre en évidence une différence significative entre les fonctions.

3. Remarquant que $\binom{n}{k} = \prod_{i=0}^{k-1} \frac{n-i}{i+1}$, un élève définit alors la fonction **binomial3(n,k)** 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) Question mathématique Montrer que si n est un entier inférieur á 124 ($= 5^3 - 1$), alors la valuation 5-adique de $n!$ est égal à : $\lfloor \frac{n}{5} \rfloor + \lfloor \frac{n}{25} \rfloor$

(b) En déduire que le résultat de binomial3(59, 22) est faux. Comment expliquer cette erreur Corriger la section de la page Wikipedia à l'adresse[1] :

`en.wikipedia.org/wiki/Binomial_coefficient`

0.3 Partie III : Calcul de tous les coefficients binomiaux

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

(a) On fixe un entier N. Par un calcul direct utilisant une des deux fonctions **binomial** de la partie précédente, donner le nombre de calculs nécessaires à l'obtention de tous les coefficients binomiaux $\binom{N}{k}$ pour k allant de 0 à N. Quelle est la complexité d'une telle méthode ?

Dans ce genre de situation, il est plus sage de faire appel à la 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]]

(b) Ecrire la fonction **tableauBinomial(N)**. Vous pouvez suivre la démarche indiquée ci-dessous :

label=☞ On initialise une variable locale tableau = [[0 for j in range(0,N+1)]]

label=☞ On remplit la *première colonne* par les affectations tableau[i][0] = 1

label=☞ On remplit itérativement le reste du tableau en utilisant la relation de Pascal et les affectations :

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

Tester la fonction pour plusieurs valeurs de n

(c) 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.

1. Erreur encore visible le 15/02/2015

```
>>> def affiche(T):
>>>         tab =' '
>>>         for ligne in T:
>>>                         for coeff in ligne :
>>>                                 tab = tab + ' ' + str(coeff).center(5) + ' 
 '
>>>                         tab = tab + '\n'
>>>         print(tab)
```

(d) Modifier la procédure affiche(T) pour qu'elle affiche le caractère * si le coefficient binomial est impair et le caractère *espace* si le coefficient binomial est pair.