DS INFORMATIQUE N°2 (le 11/02/2021)

PROBLÈME 1

I. Préliminaires

```
#################
# Question I.1 #
#################

def creerPile():
    return []

def estVide(p):
    return p == []
# ou bien: return len(p) == 0

def empiler(p, x):
    p.append(x)

def depiler(p):
    # on suppose p non viode
    return p.pop()

def sommet(p):
    # on suppose p non vide
    return p[-1]

###################
# Question I.2.a #
###################

def retourne(p):
    q = creerPile()
    while not estVide(p):
        empiler(q, depiler(p))
    return q

###################
# Question I.2.b #
###################

def copie(p):
    # ce n'est pas si facile de faire une copie sans utiliser le copy de Python!
    q1 = creerPile()
    q2 = creerPile()
    while not estVide(p):
        empiler(q1, sommet(p))
        empiler(q2, depiler(p))
    # q1 et q2 contiennet donc deux copies de p, à l'envers
    while not estVide(q1):
        empiler(p, depiler(q1))
    # on ne peut pas faire directement:  p=retourne(q1)
    # car cela crée une NOUVELLE variable p!
    return retourne(q2)
```

II. Représentation d'entiers naturels par des piles de booléens

1. Voir question suivante.

2.

```
from DefPiles import *

##################
# Question II.2 #
##################

def pileVersEntier(p):
    # calcule la valeur de l'entier n représenté par la pile de booléens p
    # p est supposée non vide
    q = copie(p) # pour ne pas modifier p
    puiss = 1 # variable contenant les puisances sucessives de 2
    # il serait très maladroit de recalculer les 2^i à chaque fois
    x = 0
    while not estVide(q):
        if depiler(q): # cad si le sommet de q vaut True
            x += puiss
        puiss *= 2
    return x

# il y a 2 opérations et 1 test par boucle, la complexité est donc de
# l'ordre de len(p).

def entierVersPile(n):
    # c'est l'écriture de n en base 2
    p = creerPile()
    if n == 0:
        empiler(p, False)
    while n != 0:
        empiler(p, n%2 != 0) # si le reste vaut 1 on empile True sinon False
        n = n // 2
        # on peut aussi utiliser divmod(n,2) pour faire les deux opérations en une seule fois
    return retourne(p)

##################
# Question II.1 #
##################

p = creerPile()
empiler(p, True)
empiler(p, False)
empiler(p, False)
empiler(p, True)
empiler(p, False)
empiler(p, True)

#Réponse:
print('Q II.1: ', p, '--->', pileVersEntier(p))
```

```
Q II.1:  [True, False, False, True, False, True] ---> 37
```

3. **a)** `empiler(p, False)` revient à ajouter un 0 la fin de l'écriture en base 2 de n donc à multiplier n par 2.

b) `empiler(p, True)` revient à ajouter un 1 la fin de l'écriture en base 2 de n donc à multiplier n par 2 et lui ajouter 1.

c) `depiler(p)` revient à supprimer le dernier chiffre de l'écriture de n en base 2, donc à remplacer n par le quotient de la division euclidienne de n par 2.

d) Un entier n est une puissance de 2 si et seulement si son chiffre le plus à gauche est un 1 et ses autres chiffres des 0.

```
from DefPiles2 import *

def testPuissance2(p):
    test = False
    # test est False tant que tous les chiffres de droite sont des zéros
    while not estVide(p) and not test:
        x = depiler(p)
        if not estVide(p):
            test = test or x
        else:
            return x
    return False

#Essais:
n = 32
print('Q II.3.d: test pour',n,':', testPuissance2(entierVersPile(n)))
n = 33
print('Q II.3.d: test pour',n,':', testPuissance2(entierVersPile(n)))
n = 2048
print('Q II.3.d: test pour',n,':', testPuissance2(entierVersPile(n)))
n = 2021
print('Q II.3.d: test pour',n,':', testPuissance2(entierVersPile(n)))
Q II.3.d: test pour 32 : True
Q II.3.d: test pour 33 : False
Q II.3.d: test pour 2048 : True
Q II.3.d: test pour 2021 : False
```

4.

```
def ouexclusif(b1,b2):
    return (b1 and not b2) or (b2 and not b1)

def f(x, y, r):
    z = ouexclusif(ouexclusif(x, y), r)
    t =(x and y) or (x and r) or (y and r)
    return z, t

#Résultat de ces fonctions:
print('ouexclusif(b1,b2) =', end=' ')
for b1 in [True, False]:
    for b2 in [True, False]:
        print(ouexclusif(b1, b2), end=' ')
print('\n')

print('z =', end=' ')
for x in [True, False]:
    for y in [True, False]:
        for r in [True, False]:
            print(f(x, y, r)[0], end=' ')
print('\n')
print('t =', end=' ')
for x in [True, False]:
    for y in [True, False]:
        for r in [True, False]:
            print(f(x, y, r)[1], end=' ')
print('\n')
```

```
ouexclusif(b1,b2) = False True True False

z = True False False True False True True False

t = True True True False True False False False
```

5. a) La fonction f réalise l'addition de x et y sachant qu'il y a déjà une retenue r, puis renvoie dans z le chiffre obtenu et dans t la nouvelle retenue éventuelle.

b)

```
from DefPiles3 import *

def addition(p1, p2):
    q = creerPile()
    r = False # retenue initiale
    while not estVide(p1) or not estVide(p2):
        if not estVide(p1):
            x = depiler(p1)
        else:
            x = False
        if not estVide(p2):
            y = depiler(p2)
        else:
            y = False
        z, t = f(x, y, r)
        empiler(q, z)
        r = t
    if r: # dernière retenue éventuelle
        empiler(q, r)
    return retourne(q)

# essai
x, y = 23, 5
p1 = entierVersPile(x)
p2 = entierVersPile(y)
q = addition(p1, p2)
print('Q II.5.b:', x,'+',y,'=', pileVersEntier(q))
```

```
Q II.5.b: 23 + 5 = 28
```

6.

```
from DefPiles4 import *

def successeurCollatz(p):
    if not sommet(p): # si nombre pair
        depiler(p) # n//2
        return p
    q = copie(p)
    empiler(q, True) # 2n + 1
    r = addition(p, q) # 3n+1
    depiler(r) # (3n+1)//2
    return r

# Essai suite de Collatz
n = 23
p = entierVersPile(n)
print('Q II.6 Suite de Collatz de', n,':', end=' ')
while p != [True]: # correspond à  1
    p = successeurCollatz(p)
    print(pileVersEntier(p), end=' ')
print('\n')
```

```
Q II.6 Suite de Collatz de 23 : 35 53 80 40 20 10 5 8 4 2 1
```

III. Fractran

1.

```
from DefPiles import *

def fractran(p, n):
    result = creerPile()
    empiler(result, n)
    q = creerPile()
    while not estVide(p):
        (a, b) = sommet(p)
        if n % b == 0:
            n = n*a//b
            empiler(result, n)
            while not estVide(q):
                empiler(p, depiler(q))
        else:
            empiler(q, depiler(p))
    while not estVide(q):
        empiler(p, depiler(q))
    return result

p = creerPile()
empiler(p, (1,2))
empiler(p, (1,3))
empiler(p, (5,6))
for n in [12, 36, 432, 41]:
    print('Q III.1: fractran(p,',n,')=', fractran(p, n), ' phi_p(',n,') = ', sommet(fractran(p
```

```
Q III.1: fractran(p, 12 )= [12, 10, 5]  phi_p( 12 ) =  5
Q III.1: fractran(p, 36 )= [36, 30, 25]  phi_p( 36 ) =  25
Q III.1: fractran(p, 432 )= [432, 360, 300, 250, 125]  phi_p( 432 ) =  125
Q III.1: fractran(p, 41 )= [41]  phi_p( 41 ) =  41
```

2. Soit p une pile de couples d'entiers (a,b) représentant des fractions irréductibles $\frac{a}{b}$ et n un entier.
On parcourt la pile en commençant par son sommet ; soit $\frac{a}{b}$ ce sommet.

- si le produit $n \times \frac{a}{b}$ est un entier m, on le stocke dans la variable `result` ; les empilages/dépilages à l'aide de la pile q permettent de remettre la pile p dans son état initial, et l'on recommence le processus avec m à la place de n.
- sinon, on dépile p en prenant soin de stocker le sommet ainsi dépilé dans la pile q, et on recommence avec le nouveau sommet (c'est-à-dire l'élément suivant de la pile).
- si le produit $n \times \frac{a}{b}$ n'est un entier pour aucune valeur de $\frac{a}{b}$ dans p, la procédure s'arrête.

Exemple : la pile de la question précédente est : $\left[\frac{5}{6}, \frac{1}{3}, \frac{1}{2}\right]$ (le sommet étant $\frac{5}{6}$).

Supposons que $n = 2^a 3^b$ avec $a, b \in \mathbb{N}$.

- On teste si le produit $2^a 3^b \times \frac{5}{6}$ est un entier.
  Si oui, c'est-à-dire si a et b sont supérieurs à 1, on remplace n par $5 \times 2^{a-1} 3^{b-1}$ et on recommence au début.
- Sinon, on teste si $2^a 3^b$ est divisible par 3 c'est-à-dire si $b \geqslant 1$ (dans ce cas il ne peut pas l'être par 2) et alors on remplace n par $2^a 3^{b-1}$, ou s'il est divisible par 2 (donc pas par 3) et alors on le remplace par $2^{a-1} 3^b$.

À la fin, $\varphi_p(2^a 3^b)$ vaut $5^{\min(a,b)}$.

3. Pour obtenir 5^{a+b} à partir de $2^a 3^b$, il suffit de considérer la pile

$$p = \left[\frac{5}{2}, \frac{5}{3}\right].$$

4. $p = [(2,2)]$ répond à la question...

5. La pile considérée ici peut s'écrire :

$$p = \left[\frac{1}{11}, \frac{136}{15}, \frac{5}{17}, \frac{4}{5}, \frac{26}{21}, \frac{7}{13}, \frac{1}{7}, \frac{33}{4}, \frac{5}{2}, \frac{1}{7}\right]$$

(le sommet étant $\frac{1}{11}$.)

Supposons que l'on parte de $N = 2^n$ avec $n \geqslant 2$.

La multiplication par les fractions $\frac{33}{4}$ puis au second passage par $\frac{1}{11}$ (seules multiplications dans la liste qui sont faites), donnent des nombres de la forme $3^k \times 2^{n-2k}$, donc on arrivera à un nombre de la forme 3^p si n est pair égal à $2p$ et $3^p \times 2$ si n impair égal à $2p+1$.

- Si $n$est pair, la multiplication par 7 ensuite donnera $3^p \times 7$ puis la multiplication par les fractions $\frac{26}{21}$ et $\frac{7}{13}$ vont donner $2^p \times 7$, et enfin la multiplication par $\frac{1}{7}$ donnera 2^p. On est donc arrivé à $2^{n//2}$.
- Si n est impair, la multiplication par $\frac{5}{2}$ transforme $3^p \times 2$ en $3^p \times 5$, puis les multiplications par les fractions $\frac{136}{15}$ et $\frac{5}{17}$ finissent par donner 5×2^{3p}. Enfin la multiplication par $\frac{4}{5}$ donnera 2^{3p+2}, et justement, $3p+2$ est égal à $(3n+1)//2$.

La prochaine puissance de 2 dans la liste sera donc bien $2^{s(n)}$ où s est la fonction définie en **II.6**.

Le programme Python suivant illustre cela ; j'ai juste modiifé la procédure `fractran` afin de ne stocker que les puissances de 2.

```
from DefPiles5 import *

def s(n):
    if n%2 == 0:
        return n//2
    else:
        return (3*n+1)//2

def suiteSyracuse(n):
    result = creerPile()
    empiler(result, n)
    while n!= 1:
        n = s(n)
        empiler(result, n)
    return result

p = creerPile()
empiler(p, (7,1))
empiler(p, (5,2))
empiler(p, (33,4))
empiler(p, (1,7))
empiler(p, (7,13))
empiler(p, (26,21))
empiler(p, (4,5))
empiler(p, (5,17))
empiler(p, (136,15))
empiler(p, (1,11))

def fractranSyracuse(p, n):
    # ici on ne retient dans result que les puissances de 2
    # et on s'arrête quand l'exposant vaut 1
    result = creerPile()
    empiler(result, n)
    q = creerPile()
    while not estVide(p):
        (a, b) = sommet(p)
        if n % b == 0:
            n = n*a//b
            if testPuissance2(entierVersPile(n)):
                empiler(result, n)
                if n == 2:
                    return result
```

```
            while not estVide(q):
                empiler(p, depiler(q))
        else:
            empiler(q, depiler(p))
    while not estVide(q):
        empiler(p, depiler(q))

n = 7
print('Q III.5: n=',n,'donc suiteSyracuse(n)=',suiteSyracuse(n),'\n', 'et fractranSyracuse(p,2
```

```
Q III.5: n= 7 donc suiteSyracuse(n)= [7, 11, 17, 26, 13, 20, 10, 5, 8, 4, 2, 1]
 et fractranSyracuse(p,2^n)= [128, 2048, 131072, 67108864, 8192, 1048576, 1024, 32, 256, 16, 4
```

6. Pour écrire une fonction `fractran` simplifiée, j'ai simplement suivi le principe exposé à la question **III.2**.

```
from DefPiles import *
def fractranSimplifie(L, n):
    # L est ici la liste des fractions dans l'ordre où elles seront examinées
    # donc l'ordre inverse de celle de la pile de l'énoncé.
    result = creerPile()
    empiler(result, n)
    fini = False
    while not fini:
        for frac in L:
            (a, b) = frac
            if n % b == 0:
                n = n*a//b
                empiler(result, n)
                fini = False
                break # on sort ici de la boucle for intérieure, pas de la boucle while!
            else:
                fini = True
    return result

L = [ (5,6), (1,3), (1,2) ] # la même pile que dans l'énoncé, mais à l'envers
print('Q III.6: fractranSimplifie(p, 12)=', fractranSimplifie(L, 12))
print('Q III.6: fractranSimplifie(p, 36)=', fractranSimplifie(L, 36))
```

```
Q III.6: fractranSimplifie(p, 12)= [12, 10, 5]
Q III.6: fractranSimplifie(p, 36)= [36, 30, 25]
```

PROBLÈME 2

I. Question préliminaire.

0. On peut écrire :

```
def coordDistinctes(a, b):
    for i in range(len(a)):
        for j in range(len(b)):
            if a[i] == b[j]:
                return False
    return True
```

mais aussi, de façon plus synthétique :

```
def coordDistinctes(a, b):
    for x in a:
        if x in b:
            return False
    return True
```

II. Confection d'un tournoi où les matchs sont constitués en fonction du classement des joueurs, modèle "têtes de série"

1. Je donne ci-dessus un programme inspiré du tri par insertion vu en cours, adapté au fait que le tri est ici par ordre décroissant, et que les éléments de la liste sont des couples dont le deuxième élément sert de critère de comparaison pour le tri.

D'après le cours, la complexité de ce programme est un $\mathcal{O}(p^2)$ où p est la taille de la liste.

```
def tri(liste_inscr):
    # on fait un tri par insertion, après avoir copié la liste
    L = liste_inscr.copy()
    n = len(L)
    for i in range(1, n):
        # ici L[:i] contient les i premiers éléments triés par ordre décroissant de cote
        # et on y insère L[i]
        x = L[i]
        j = i-1
        while (j >= 0) and (L[j][1] < x[1]):
            L[j+1] = L[j]
            j -= 1
        L[j+1] = x
    return L

# Exemple de l'énoncé:
liste_inscr = [(1028,0.25),(757,-0.85),(2045,0.65),(534,0.72)]
print(tri(liste_inscr))
```

```
[(534, 0.72), (2045, 0.65), (1028, 0.25), (757, -0.85)]
```

—

III. Établissement d'un système de cotation des joueurs et prise en compte des résultats pour l'évolution des cotes.

III.1. Préliminaires mathématiques et premières fonctions Python

1. Soit $y \in]-1\,;1[$, fixé, et f l'application qui à tout $x \in]-1\,;1[$ associe $\dfrac{x+y}{1+xy}$.

f est bien définie sur $]-1\,;1[$ car, pour $(x,y)\in]-1\,;1[^2$, on a $|xy|<1$ donc xy ne peut être égal à -1.
f est de classe $\mathscr{C}^1$ comme quotient de telles fonctions ; on calcule : $f'(x)=\dfrac{1-y^2}{(1+xy)^2}$; ainsi $f'(x)>0$ sur $]-1\,;1[$, donc f est strictement croissante, et réalise une bijection de $]-1\,;1[$ sur l'intervalle image, qui est aussi égal à $]-1\,;1[$ car $\lim\limits_{x\to\pm1} f(x)=\pm1$.

On a montré en particulier que, si $(x,y)\in]-1\,;1[$, alors $x\star y$ appartient encore à $]-1\,;1[$, c'est-à-dire que la loi $\star$ est interne.

2. Ce sont de simples calculs. Pour tous $x,y,z\in]-1\,;1[$ on a :

- $(x\star y)\star z=\left(\dfrac{x+y}{1-xy}\right)\star z=\dfrac{\frac{x+y}{1+xy}+z}{1+\frac{x+y}{1+xy}z}=\dfrac{x+y+z+xyz}{1+xy+yz+zx}$, et l'égalité $(x\star y)\star z=x\star(y\star z)$ se vérifie facilement : la loi $\star$ est associative ;
- $x\star y=y\star x$ de façon immédiate : la loi est *commutative* ;
- $x\star 0=x$: 0 est *élément neutre* ;
- $x\star(-x)=0$: tout élément $x\in]-1\,;1[$ admet un symétrique pour la loi $\star$.

3. Voir question **6.**

4. On peut remarquer que la fonction f est tout simplement la fonction th, mais cela n'est guère utile.
Pour montrer que f est une bijection de $\mathbb{R}$ sur $]-1\,;1[$ il suffit de montrer que, pour tout $y\in]-1\,;1[$, l'équation $y=f(x)$ possède une et une seule solution dans $\mathbb{R}$. Or :

$$\begin{aligned} y=f(x)&\iff y=\frac{\mathrm{e}^x-\mathrm{e}^{-x}}{\mathrm{e}^x+\mathrm{e}^{-x}}\iff y=\frac{\mathrm{e}^{2x}-1}{\mathrm{e}^{2x}+1}\qquad\text{(en multipliant num. et dénom. par }\mathrm{e}^x)\\ &\iff y\left(\mathrm{e}^{2x}+1\right)=\mathrm{e}^{2x}-1\iff \mathrm{e}^{2x}(1-y)=1+y.\end{aligned}$$

Or y appartient à $]-1\,;1[$ donc $1-y$ est non nul et $\dfrac{1+y}{1-y}$ est strictement positif, donc finalement :

$$y=f(x)\iff 2x=\ln\left(\frac{1-y}{1+y}\right)$$

ce qui montre que f est bijective de $\mathbb{R}$ sur $]-1\,;1[$, et que sa bijection réciproque est l'applciation $y\mapsto\dfrac{1}{2}\ln\left(\dfrac{1-y}{1+y}\right)$.

5. Soient $s,t\in\mathbb{R}$; alors :

$$\begin{aligned} f(s)\star f(t)&=\frac{f(s)+f(t)}{1+f(s)f(t)}=\frac{\frac{\mathrm{e}^s-\mathrm{e}^{-s}}{\mathrm{e}^s+\mathrm{e}^{-s}}+\frac{\mathrm{e}^t-\mathrm{e}^{-t}}{\mathrm{e}^t+\mathrm{e}^{-t}}}{1+\frac{\mathrm{e}^s-\mathrm{e}^{-s}}{\mathrm{e}^s+\mathrm{e}^{-s}}\frac{\mathrm{e}^t-\mathrm{e}^{-t}}{\mathrm{e}^t+\mathrm{e}^{-t}}}=\frac{(\mathrm{e}^s-\mathrm{e}^{-s})(\mathrm{e}^t+\mathrm{e}^{-t})+(\mathrm{e}^t-\mathrm{e}^{-t})(\mathrm{e}^s+\mathrm{e}^{-s})}{(\mathrm{e}^s+\mathrm{e}^{-s})(\mathrm{e}^t+\mathrm{e}^{-t})+(\mathrm{e}^s-\mathrm{e}^{-s})(\mathrm{e}^t-\mathrm{e}^{-t})}\\ &=\frac{\mathrm{e}^{s+t}-\mathrm{e}^{-s-t}}{\mathrm{e}^{s+t}+\mathrm{e}^{-s-t}}=f(s+t).\end{aligned}$$

Soient $x,y,z\in]-1\,;1[$; f étant bijective de $\mathbb{R}$ sur $]-1\,;1[$, et g étant sa bijection réciproque, en posant $x'=g(x)$, $y'=g(y)$ et $z'=g(z)$, on a $x=f(x')$, $y=g(y')$ et $z=f(z')$ d'où

$$x\star y=z\iff f(x')\star f(y')=f(z')\iff f(x'+y')=f(z')\iff x'+y'=z'\iff g(x)+g(y)=g(z).$$

6.

```python
from math import exp, log as ln

def etoile(x, y):
    return (x+y)/(1+x*y)

def f(t):
    return (exp(t)-exp(-t))/(exp(t)+exp(-t))

def g(x):
    return 0.5*ln((1+x)/(1-x))

def resout(z, d):
    x = f(0.5*(g(z)+g(d)))
    y = f(0.5*(g(z)-g(d)))
```

```
    return (x,y)
```

III.2. Évolution des cotes par prise en compte des résultats

1. Essayons d'expliquer...

- Au début de la fonction, les expériences des deux joueurs sont augmentées de 1.
- `diff12` représente le déséquilibre effectif sur le match (vaut -1 si score 0-21 et 1 si score 21-0).
- Si on imagine que la cote `c2` du joueur 2 est sa cote objective, le joueur 1 devrait avoir pour cote `nouvpos1 = etoile(c2,diff12)`, et de manière analogue pour `nouvpos2`; `delta1` et `delta2` représentent alors les différences entre les cotes `c1` et `c2` et ces nouvelles cotes théoriques calculées.
- On déplace alors les cotes des deux joueurs dans la direction de `delta1` et `delta2`, en multipliant par un facteur multiplicatif pour ne pas qu'un seul set modifie trop la cote d'un joueur, et en multipliant par le quotient entre l'expérience de l'adversaire (plus elle est grande, plus la valeur de `c2`- est fiable) divisée par la somme des deux expériences (plus les joueurs sont expérimentés, et moins un seul set doit les modifier).

2.

```
def priseEnCompteDouble(listing_joueurs, j1, j1bis, j2, j2bis, scor1, scor2):
    listingjoueurs[j1][1] += 1
    listingjoueurs[j2][1] += 1
    listingjoueurs[j1bis][1] += 1
    listingjoueurs[j2bis][1] += 1
    diff12 = (scor1-scor2)/(scor1+scor2)
    c1, e1 = listing_joueurs[j1]
    c2, e2 = listing_joueurs[j2]
    c1bis, e1bis = listing_joueurs[j1bis]
    c2bis, e2bis = listing_joueurs[j2bis]
    cbinom1 = etoile(c1, c1bis)
    cbinom2 = etoile(c2, c2bis)
    nouvposbinom1 = etoile(cbinom2, diff12)
    nouvposbinom2 = etoile(cbinom1, -diff12)
    nouvpos1, nouvpos1bis = resout(nouvposbinom1, etoile(c1, -c1bis))
    nouvpos2, nouvpos2bis = resout(nouvposbinom2, etoile(c2, -c2bis))
    delta1 = etoile(nouvpos1, -c1)
    delta2 = etoile(nouvpos2, -c2)
    delta1bis = etoile(nouvpos1bis, -c1bis)
    delta2bis = etoile(nouvpos2bis, -c2bis)
    facteur = 0.1
    s = e1 + e1bis + e2 + e2bis
    listing_joueurs[j1][0] = etoile(c1, facteur*(s-e1)/s*delta1)
    listing_joueurs[j2][0] = etoile(c2, facteur*(s-e2)/s*delta2)
    listing_joueurs[j1bis][0] = etoile(c1, facteur*(s-e1bis)/s*delta1bis)
    listing_joueurs[j2bis][0] =etoile(c2, facteur*(s-e2bis)/s*delta2bis)
```

III.3. Vérification de la pertinence du système de mise à jour des cotations

1.

```
def simulPoint(alpha):
    r = random()
    if r < alpha:
        return 1
    else:
        return 2

def simulSet(alpha):
    fini = False
    scor1, scor2 = 0, 0
    while not fini:
        r = simulPoint(alpha)
```

```
            if r == 1:
                scor1 += 1
            else:
                scor2 += 1
            diff = abs(scor1 - scor2)
            scormax = max(scor1, scor2)
            if (scormax >= 21 and diff >= 2) or scormax == 30:
                fini = True
        return scor1, scor2
```

2.

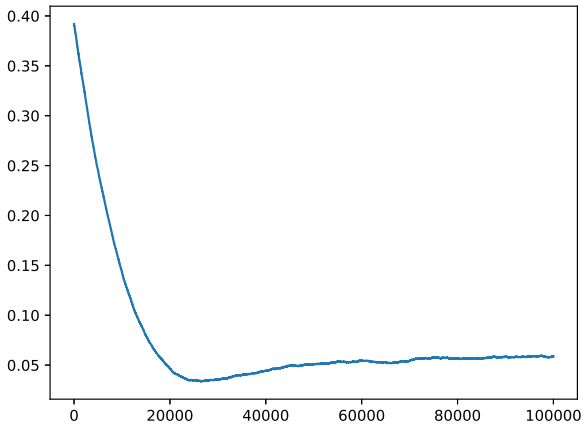
```
def joueMatchAleatoire(V, listing_joueurs):
        p = len(V)
        j1 = randint(0, p-1)
        encore = True
        while encore:
            j2 = randint(0,p-1)
            if j2 != j1:
                encore = False
        alpha = (1 + etoile(V[j1],-V[j2]))/2
        scor1, scor2 = simulSet(alpha)
        priseEnCompteSimple(listing_joueurs, j1, j2, scor1, scor2)
```

3.

```
def simul(V, N):
    p = len(V)
    listing_joueurs = [ [0,0] for _ in range(p) ]
    x = [0]
    s = 0
    for i in range(p):
        s += abs(V[i]-listing_joueurs[i][0])
    s /= p
    y = [s]
    for k in range(1, N+1):
        joueMatchAleatoire(V, listing_joueurs)
        s = 0
        for i in range(p):
            s += abs(V[i]-listing_joueurs[i][0])
        s /= p
        x.append(k)
        y.append(s)
    plt.plot(x,y)
    plt.show()
    plt.close('all')

# essai
p = 1000
N = 100000
# création d'une liste aléatoire de cotes entre -0.8 et 0.8
V = [ 1.6*random()-0.8 for _ in range(p) ]
simul(V, N)
```

Voici la figure obtenue :



Après 100000 simulations, on voit que l'erreur finale entre les cotes initiales et celles obtenues par la méthode de l'énoncé est de l'ordre de 5%, ce qui valide le modèle choisi.

4.

```
def memeOrdre(V, listing_joueurs):
    # teste si les deux listes sont rangées dans le même ordre
    p = len(V)
    k = 0
    for k in range(p-1):
        if (V[k+1]-V[k])*(listing_joueurs[k+1][0]-listing_joueurs[k][0])<0:
            return False
    return True

def tempsAttente(V):
    p = len(V)
    listing_joueurs = [ [1.6*random()-0.8, 0] for _ in range(p) ]
    nb = 0
    while not(memeOrdre(V, listing_joueurs)):
        nb += 1
        joueMatchAleatoire(V, listing_joueurs)
    return nb
```

Pour $p = 100$ le programme donne un temps d'attente entre 4000 et 10000, mais pour des valeurs plus grandes le temps d'attente semble extrêmement long...

IV. Constitution d'une journée de championnat de double la plus équilibrée possible

1.

```
def desequilibre(C):
    p=len(C)
    D=[]
    for i1 in range(p-2):
        for i2 in range(i1+1, p-1):
            for i2bis in range(i2+1, p):
                for i1bis in range(i1+1, p):
                    if i1bis!=i1 and i1bis!=i2 and i1bis!=i2bis:
                        d = abs(etoile(etoile(etoile(C[i1],C[i1bis]),-C[i2]),-C[i2bis]))
                        D.append((i1, i1bis, i2, i2bis, d))
    return D
```

À chaque étape de boucle, il y a 3 appels à la fonction `etoile`. Et on comme on a $\mathcal{O}(p^4)$ boucles, la complexité est en $\mathcal{O}(p^4)$.

2.

```
def coupeSeuil(D,s):
    R = []
    for match in D:
        if match[4] <= s:
            R.append(match)
    return R
```

3.

```
def creerMatchs(k, L):
    if k == 1:
        if L == []:
            return []
        else:
            return [L[randint(0, len(L)-1)]]
    else:
        while L != []:
            m = L[randint(0, len(L)-1)]
            Lprime = []
            for mprime in L:
                if coordDistinctes(m[0:4], mprime[0:4]):
                    Lprime.append(mprime)
            r = creerMatchs(k-1, Lprime)
            if r != []:
                return r + [m]
            else:
                L.remove(m)
        return []
```

4.

```
def journeeEquilibree(C):
    q = len(C)//4
    D = desequilibre(C)
    L = D.copy()
    M = creerMatchs(q, L)
    encore = True
    while encore:
        s = 0
        for m in M:
            s += m[4]
        s /= q # déséquilibre moyen
        L = coupeSeuil(D, s)
        nouvM = creerMatchs(q, L)
        if nouvM == []:
            encore = False
        else:
            M = nouvM
    return M

#pour tester si ca marche :
q = 8
C = [ 1.6*random()-0.8 for _ in range(4*q) ]
print(journeeEquilibree(C))
```

V. Utilisation d'une base de données fédérales

1.

```
SELECT ville,contact FROM clubs WHERE departement=94
```

2. Si l'on veut utiliser la syntaxe avec JOIN :

```
SELECT nomclub, COUNT(numlic) AS effectif, AVG(cote), MAX(cote)
FROM joueurs JOIN clubs ON clubaccueil=idclub
GROUP BY nomclub ORDER BY effectif DESC
```

mais il est plus clair à mon avis d'écrire :

```
SELECT nomclub, COUNT(numlic) AS effectif, AVG(cote), MAX(cote)
FROM joueurs, clubs WHERE clubaccueil=idclub
GROUP BY nomclub ORDER BY effectif DESC
```

3.

```
SELECT numlic, cote, COUNT(idset)
FROM joueurs, resultats
WHERE (numlic=j1 OR numlic=j2) AND (annee=2021 OR (annee=2020 AND mois>=2))
GROUP BY numlic
```

4.

```
SELECT jour, mois, annee, lieu, joueurs1.clubaccueil
FROM resultats, joueurs AS joueurs1, joueurs AS joueurs2
WHERE j1=joueurs1.numlic AND j2=joueurs2.numlic
      AND joueurs1.clubaccueil=joueurs2.clubaccueil
      AND joueurs1.stadecompet='Finale'
```