1 Correction du TD 5

Exercice 1

$$\left|m'_{uv} - m'_{ij}\right| = C\left(\|H_v - H_j\| + \|W_u - W_i\|\right) + \|H_v - H_j\|\,\|W_u - W_i\| \tag{1}$$

```
class CarreMagique :
    def __init__(self, coef) :
        self.mat = [ [ coef[i+j*3] for i in range(3) ] for j in range(3) ]
    def __str__(self) :
        return "\n".join ( [ ",".join( [ str(n) for n in row ] ) for row in self.mat ] )
    def __add__ (self, carre) :
        coef = []
        for i in range(3) :
            for j in range(3) :
                coef.append ( self.mat[i][j] + carre.mat[i][j])
        return CarreMagique(coef)

c = CarreMagique ( [ 1,3,4, 2,6,9, 8,7,5 ] )
print (c)
print (c + c)
```

```
Fonction \codes{est\_magique}:
class CarreMagique :
    def __init__(self, coef) :
        self.mat = [ [ coef[i+j*3] for i in range(3) ] for j in range(3) ]
    def __str__(self) :
        return "\n".join ( [ ",".join( [ str(n) for n in row ] ) for row in self.mat ] )
    def __add__ (self, carre) :
        coef = []
        for i in range(3) :
            for j in range(3) :
                coef.append ( self.mat[i][j] + carre.mat[i][j])
        return CarreMagique(coef)

    def somme_ligne_colonne_diagonale(self):
        tout = [ sum ( ligne ) for ligne in self.mat ] + \
               [ sum ( [ self.mat[i][j] for j in range(3) ] ) for i in range(3) ] + \
               [ sum ( [ self.mat[i][i] for i in range(3) ] ) ] + \
               [ sum ( [ self.mat[2-i][i] for i in range(3) ] ) ]
        return tout

    def coefficient_unique(self):
        d = { }
        for ligne in self.mat :
            for c in ligne :
                d [c] = d.get(c,0) + 1
        return len(d) == 9

    def est_magique(self):
        unique = self.coefficient_unique()
        if not unique : return False
        somme = self.somme_ligne_colonne_diagonale()
        return min(somme) == max(somme)


c = CarreMagique ( [ 1,1,1, 1,1,1, 1,1,1 ] )
print (c.est_magique())
c = CarreMagique ( [ 1,4,8, 5,2,6, 7,9,3 ] )
print (c.est_magique())
```

Trouver tous les carrés, solution naïve et très lente :

```
def tous_les_carre_naif() :
    res = []
    for a1 in range(9) :
        for a2 in range(9) :
            for a3 in range(9) :
                for b1 in range(9) :
                    for b2 in range(9) :
                        for b3 in range(9) :
                            for c1 in range(9) :
                                for c2 in range(9) :
                                    for c3 in range(9) :
                                        carre = CarreMagique( [a1,a2,a3, b1,b2,b3, c1,c2,c3 ])
                                        if carre.est_magique() :
                                            res.append (carre)
                                            print (carre)
    return res
```

Tous les carrés magique, même version naïve avec deux boucles et toujours très lente :

```
def tous_les_carre_naif2() :
    # on choisit l'ensemble de tous les tableaux de 9 chiffres compris entre 1 et 9
    coef = [ 1 ] * 9
    res = [ ]
    while coef [0] < 10 :
        carre = CarreMagique(coef)
        if carre.est_magique() :
            res.append (carre)
            print (carre)
        coef[-1] += 1
        if coef[-1] >= 10 :
            i = len(coef)-1
            while coef[i] >= 10 and i > 0 :
                coef[i] = 1
                coef[i-1] += 1
                i -= 1
```

On parcourt toutes les permutations, on trouve 112 carrés magiques dans un temps très raisonnables (une quinzaine de secondes) :

```
#coding:latin-1
def tous_les_carres_permutation( permut = None, pos = 0):
    if pos == 9 :
        carre = CarreMagique (permut)
        if carre.est_magique() :
            print (carre)
            print ()
            return [ carre ]
        else :
            return []
    else :
        res = [ ]
        if permut == None :
            permut = [ i+1 for i in range(9) ]
        for i in range (pos,9) :
            # on permute les éléments i et pos
            a = permut[i]
            permut[i] = permut[pos]
            permut[pos] = a
```

```
            res += tous_les_carres_permutation(permut, pos+1)

            # on effectue la permutation inverse
            a = permut[i]
            permut[i] = permut[pos]
            permut[pos] = a
        return res


res = tous_les_carres_permutation()
print ("nombre de carrés", len(res))
```

Pour aller plus loin ou pour ceux qui ont fini plus tôt .

Est-il possible d'aller plus vite que de parcourir l'ensemble des permutations ? La réponse est oui. En parcourant les permutations, la fonction qui teste si les chiffres sont uniques est devenu inutile. Pour vérifier qu'on va plus vite, on peut mesurer le temps que met la fonction pour trouver tous les carrés :

```
import time
d = time.clock()
res = tous_les_carres_permutation()
d = time.clock() - d
print ("nombre de carrés", len(res), " en ", d)
```

Pour aller plus vite, il faut utiliser la contrainte des sommes. Comment ? Lorsqu'on permute les nombres, on peut simplement vérifier que les deux premières lignes ont la même somme. L'utilisation de cette contrainte nous permet de d'aller 15 fois plus vite et d'obtenir le résultat en moins d'une seconde :

```
def tous_les_carres_permutation_ligne12_meme_somme( permut = None, pos = 0):
    if pos == 9 :
        carre = CarreMagique (permut)
        if carre.est_magique() :
            #print (carre)
            #print ()
            return [ carre ]
        else :
            return []
    else :
        if pos >= 6 :                                       # ajout
            if sum ( permut[:3]) != sum(permut[3:6]) :      # ajout
                return [ ]                                  # ajout

        res = [ ]
        if permut == None :
            permut = [ i+1 for i in range(9) ]
        for i in range (pos,9) :
            # on permute les éléments i et pos
            a = permut[i]
            permut[i] = permut[pos]
            permut[pos] = a

            res += tous_les_carres_permutation_ligne12_meme_somme(permut, pos+1)  # changé

            # on effectue la permutation inverse
            a = permut[i]
```

```
            permut[i] = permut[pos]
            permut[pos] = a
        return res
```

Le programme final au complet :

```
#coding:latin-1
class CarreMagique :
    def __init__(self, coef) :
        self.mat = [ [ coef[i+j*3] for i in range(3) ] for j in range(3) ]
    def __str__(self) :
        return "\n".join ( [ ",".join( [ str(n) for n in row ] ) for row in self.mat ] )
    def __add__ (self, carre) :
        coef = []
        for i in range(3) :
            for j in range(3) :
                coef.append ( self.mat[i][j] + carre.mat[i][j])
        return CarreMagique(coef)

    def somme_ligne_colonne_diagonale(self):
        tout = [ sum ( ligne ) for ligne in self.mat ] + \
               [ sum ( [ self.mat[i][j] for j in range(3) ] ) for i in range(3) ] + \
               [ sum ( [ self.mat[i][i] for i in range(3) ] ) ] + \
               [ sum ( [ self.mat[2-i][i] for i in range(3) ] ) ]
        return tout

    def coefficient_unique(self):
        d = { }
        for ligne in self.mat :
            for c in ligne :
                d [c] = d.get(c,0) + 1
        return len(d) == 9

    def est_magique(self):
        unique = self.coefficient_unique()
        if not unique : return False
        somme = self.somme_ligne_colonne_diagonale()
        return min(somme) == max(somme)

def tous_les_carres_permutation_ligne12_meme_somme( permut = None, pos = 0):
    if pos == 9 :
        carre = CarreMagique (permut)
        if carre.est_magique() :
            #print (carre)
            #print ()
            return [ carre ]
        else :
            return []
    else :
        if pos >= 6 :                                       # ajout
            if sum ( permut[:3]) != sum(permut[3:6]) :      # ajout
                return [ ]                                  # ajout

        res = [ ]
        if permut == None :
            permut = [ i+1 for i in range(9) ]
        for i in range (pos,9) :
            # on permute les éléments i et pos
```

```
            a = permut[i]
            permut[i] = permut[pos]
            permut[pos] = a

            res += tous_les_carres_permutation_ligne12_meme_somme(permut, pos+1)  # changé

            # on effectue la permutation inverse
            a = permut[i]
            permut[i] = permut[pos]
            permut[pos] = a
        return res

import time
d = time.clock()
res = tous_les_carres_permutation_ligne12_meme_somme()
d = time.clock() - d
print ("nombre de carrés", len(res), " en ", d)
```

fin correction TD ?? □