```
# -*- coding: utf-8 -*-
"""
Created on Tue Jun  7 11:22:38 2016

@author: linux12
"""
#   Dans '8fois4 Version 3' on a presque automatisé la recherche
# de couverture du 4-cube par des "quadruplets sans // sans bout commun"
# (QSPSBC). On sait qu'il y a 76 QSPSBC d'arête //x égale à [1,2]
#  Quitte à renommer les sommets il suffit de considérer cette [1,2]
#
# Dans cette Version 6 :
#   on a trouvé les 76  QSPSBC commençant par chaque arête par_x
#   on veut les groupes de 8 qui recouvrent
#   pour cela on va les commençant
#
# par_x : les arêtes parallèlles à la direction x, idem pour y,z,t
# import time

par_x = [[1, 2], [3, 8], [4, 7], [5, 6], [9, 14], [10, 13], [11, 12], [15, 16]]

par_y = [[1, 8], [2, 3], [4, 5], [6, 7], [9, 16], [10, 11], [12, 13], [14, 15]]

par_z = [[1, 6], [2, 5], [3, 4], [7, 8], [9, 10], [11, 16], [12, 15], [13, 14]]

par_t = [[1, 16], [2, 15], [3, 14], [4, 13], [5, 12], [6, 11], [7, 10], [8, 9]]


# en x,y,z,t on mémorise les arêtes déja vues (niveaux 0 à 6)
# erreur ici : j'avai écrit DV= [[], [], [], [], [], [], []]
# puis DVx, DVy, DVz, DVt = DV,DV,DV,DV .. ils étaient toujours égaux

# DVx = [[], [], [], [], [], [], []]
# DVy = [[], [], [], [], [], [], []]
# DVz = [[], [], [], [], [], [], []]
# DVt = [[], [], [], [], [], [], []]

# Bons[0] contiendra les quadruplets non // sans bout commun
# de la première ligne de l'octuplet cherché
# preBons : préparation du calcul pour gagner du temps
# Reste_par_u : celles non encore utilisées
# Bons = [[], [], [], [], [], [], [], []]
# Reste_par_x = par_x
# Reste_par_y = par_y
# Reste_par_z = par_z
# Reste_par_t = par_t

# preBons = [[], [], [], [], [], [], [], []]


# SansBoutCommun traite de deux arêtes
def SansBoutCommun(a, b):
    return ((a[0] != b[0]) and (a[0] != b[1]) and (a[1] != b[0]) and (a[1] !=
b[1]))


# 'injectif' teste l'injectivité d'une liste que l'on sait déjà triée
def injectif(l):
    if len(l) <= 1:
        return True
    else:
        return ((l[0] != l[1]) and injectif(l[1:]))

# 'flat' met à plat [[a],[b],[c],[d]] devient [a,b,c,d]
```

```
def flat(x):
    return x[0]+x[1]+x[2]+x[3]


#isf(x) dit si x est injectif (après sortage du flattage)
def isf(x):
    return injectif(sorted(flat(x)))


# pour une arête // x, on cherche toutes  celles // à y,z,t sans bout commun
def CherchePourUn(x):
    yPoss = [u for u in par_y if SansBoutCommun(u, x)]
    zPoss = [u for u in par_z if SansBoutCommun(u, x)]
    tPoss = [u for u in par_t if SansBoutCommun(u, x)]
    XYZT = [[x, y, z, t] for y in yPoss for z in zPoss for t in tPoss]
    return XYZT

#L[0..7] : ligne 0..7 du futur tableau qui va tout contenir
prepareL = [CherchePourUn(par_x[k]) for k in range(8)]

# on ne garde que les quadruplets injectifs :
# les y,z,t doivent aussi être disjoints
L = [[x for x in prepareL[k] if isf(x)] for k in range(8)]
# il y a 76 QSPSBC d'arête //x égale à [1,2]
#
# L[0] contient ceux ayant par_x[0] en premier (idem L[k])
# le but est de trouver huit QSPSBC 'recouvrant le 4-cube'
# L'expérience avec l'explorateur de variables montre qu'on n'y voit plus rien
# on va donc re-coder les arêtes par simple numérotation :
# [1,2] devient 0 ... [8,9] devient 31


def codeParx(m):
    return par_x.index(m)


def codePary(m):
    return par_y.index(m)+8


def codeParz(m):
    return par_z.index(m)+16


def codePart(m):
    return par_t.index(m)+24

# ParOfCode(x) prend un entier de 0..23 et donne l'arête en [ini,fin]
# exemple ParOfCode(0) vaut [1,2]

def ParOfCode(x):
    if x < 8:
        return par_x[x]
    elif x < 16:
        return par_y[x-8]
    elif x < 24:
        return par_z[x-16]
    else:
        return par_t[x-24]


def CodeOfPar(m):
    if m in par_x:
        return codeParx(m)
```

```
    elif m in par_y:
        return codePary(m)
    elif m in par_z:
        return codeParz(m)
    elif m in par_t:
        return codePart(m)
    else:
        print("non arête")

# codeParQ utilise les codePar pour les composantes d'un quadruplet
def codeParQ(q):
    return [codeParx(q[0]), codePary(q[1]), codeParz(q[2]), codePart(q[3])]

codeL = [[codeParQ(q) for q in x] for x in L]


# On choisit un QSPSBC dans codeL[0] et
# on retire des codeL[1..7] ceux ayant le même y ou z ou t


def retire(choisi, autres):
    pasy = [x for x in autres if not (choisi[1] == x[1])]
    pasz = [x for x in pasy if not (choisi[2] == x[2])]
    past = [x for x in pasz if not (choisi[3] == x[3])]
    return past



# retirePartout0 enlève 'choisi' de codeL[1] ...  codeL[7]
# def retirePartout0(choisi):
#     return [retire(choisi, codeL[x]) for x in range(1, 8)]

# pp0 = codeL

#pp1 = retirePartout0(codeL[0][0])


# En prenant à chaque fois le premier (pour pp6 : le 3-ième de pp5)
# on va jusqu'à un pp6 non vide


def retirePartout(m, pere, choisi):
    return [retire(choisi, pere[x]) for x in range(1, 8-m)]

# pp1 = retirePartout(0, pp0, pp0[0][0])

# pp2 = retirePartout(1, pp1, pp1[0][2])

# pp3 = retirePartout(2, pp2, pp2[0][0])

# pp4 = retirePartout(3, pp3, pp3[0][0])

# pp5 = retirePartout(4, pp4, pp4[0][0])

# pp6 = retirePartout(5, pp5, pp5[0][2])

# pp7 = retirePartout(6, pp6, pp6[0][0])


# Il faut maintenant automatiser tout cela :
#   on choisira à la main le QSPSBC de codeL[0][0] puis tout
# sera automatique (avec enregistement des ppè non vides)
# QR : QuadrupletsRestants
```

```
# global QRm codeL est fix2 ;qis AR50- doit pouvoir chqnger 5sqns chqnger codeL-
QR = [[], [], [], [], [], [], [], []]
QR[0] = codeL
QR[0] =[[x for x in codeL[k]] for k in range(len(codeL))]

#    deb = time.time()#    print(time.time()-deb)

def RP(niveau, Liste):
    global QR
    if niveau == 7: #   print("victoite", Liste)
        return Liste
    else:
        if (Liste[niveau] <= len(QR[niveau][0])-1): #            print("ICI 2",
niveau, QR[niveau][0][Liste[niveau]])
            QR[niveau+1] = retirePartout(niveau, QR[niveau], QR[niveau][0]
[Liste[niveau]])
            RP(niveau+1, Liste)
#        else:   #  print("je coince au niveau", niveau, Liste)

RP(0, [14, 0, 0, 0, 0, 0, 0, 0])

Res = [RP(0, [k, 0, 0, 0, 0, 0, 0, 0]) for k in range(76)]

"""
 On obtient avec RP(0, [14, 0, 0, 0, 0, 0, 0, 0]) les QSPSBC :

 0,11,18,28
 1,10,16,25
 2,8,17,26
 3,9,19,24
 4,13,22,27
 5,15,21,31
 6,12,23,30
 7,14,20,29

     il faut maintenant les exclure de la suite

     il faut maintenant que les Listes de départ s'enchaînent seules

"""

# les PairesInterdites par [a,b,c,d] sont les six paires [a,b] ... [c,d]
def Pinterdits(q):
    [m, n, o, p] = q
    return [[m, n], [m, o], [m, p], [n, o], [n, p], [o, p]]

# initiaqlisation de la listeliste d'interdits
Interdits = [[] for k in range(32)]

# quand [u,v] a été vu on ajoute v dans Interdits[u]
def enregistreI(p):
    Interdits[p[0]] = Interdits[p[0]] + [p[1]]

# on enregistre les 6 paires d'une ListePaires
def enregistreLP(lp):
    for k in range(6):
        enregistreI(lp[k])

# pour pouvoir tester
Q0 = [[0,11,18,28],[1,10,16,25],
      [2,8,17,26], [3,9,19,24],
      [4,13,22,27],[5,15,21,31],
      [6,12,23,30],[7,14,20,29]]
```

```
# Dans la version 8 : je cherchais comment interdire des  QSPSBC
"""
    Autre idée : inspiré par l'exemple de la dimension 3 :
        On cherche toutes les isométries de K4
            Pour chaque isométrie on prend l'image de Q0
            On jette celles des images ayant un côté commun avec Q0

                Il reste à jouer au puzzle

    Les isométries de K4 sont 24 x 16 : si on parle du cube de centre [0,0,0,0]
        On a [c1,c2,c3,c4] une permutation de [0,1,2,3]
        On a quatre signe [s0,s1,s2,s3]
        On va appliquer cela à un quadruplet [x,y,z,t] (formés de 0 ou 1)

"""
"""  LP01 = ListePermutations de 0 et 1
    InserePartout([2,5,8],36) = [[36, 2, 5, 8], [2, 36, 5, 8], [2, 5, 36, 8],
[2, 5, 8, 36]]
    suivant ferait ce qu'on veut avec *InserePartout ... c'est interdit (?)
        on utilise donc deliste pour le faire après coup
         deliste([[7],[4],[[3],[9]]]) =  [7, 4, [3], [9]]

    On utilise tout cela pour fabriquer les LP02 et LP03

        On fabrique alors les seize quadruplets de 'p' et 'm'
        emboite([1,4,7],[2,5,8]) =  [[1, 2], [4, 5], [7, 8]]
        enfin seizifie emboite un quadruplet avec chacun des 16 quadruplets de
'p' ou 'm'

"""
LP01 = [[1,0],[0,1]]

def InserePartout(l,x):
    long = len(l)
    return [l[:k]+[x]+l[k:] for k in range(long+1)]

def suivant(l,n):
    return [InserePartout(l[k],n) for k in range(len(l))]

def deliste(ll):
    if len(ll)==1:
        return ll[0]
    else:
        return ll[0]+deliste(ll[1:])

LP02 = deliste(suivant(LP01,2))

LP03 =  deliste(suivant(LP02,3))

deux = [['p'],['m']]

quatre = deliste([[x+['p'],x+['m']] for x in deux])

huit = deliste([[x+['p'],x+['m']] for x in quatre])

seize = deliste([[x+['p'],x+['m']] for x in huit])

def emboite(l1,l2):
   return  [[l1[k],l2[k]] for k in range(len(l1))]

def seizifie(q):
    return [emboite(q,seize[k]) for k in range(len(seize))]
```

```
"""
    LIK4 est la liste des isométries du cube de dimension 4 : l'idée est d'en
coder les matrices
    LIK4[123] = [[3, 'm'], [0, 'm'], [2, 'p'], [1, 'm']]
        cela signifie que
            e0 a pour image -e3
            e1 a pour image -e0
            e2 a pour image  e2
            e4 a pour image -e1

    après il faut tenir compte de ce que mes coordonnées étaient 0 ou 1 (au lieu
de -1 ou 1)
        mes arêtes étaient codées [2,5] il faut

"""

enversLIK4 = deliste([seizifie(x) for x in LP03])

LIK4 = [list(reversed(enversLIK4[u])) for u in range(len(enversLIK4))]

def evalue(f,q):
    [x,y,z,t] = q
    res = [0,0,0,0]
    if f[0][1] == 'p':
        res[f[0][0]] = x
    else:
         res[f[0][0]] = 1-x
    if f[1][1] == 'p':
        res[f[1][0]] = y
    else:
         res[f[1][0]] = 1-y
    if f[2][1] == 'p':
        res[f[2][0]] = z
    else:
         res[f[2][0]] = 1-z
    if f[3][1] == 'p':
        res[f[3][0]] = t
    else:
         res[f[3][0]] = 1-t
    return res


S1 = [0,0,0,0]
S2 = [1,0,0,0]
S3 = [1,1,0,0]
S4 = [1,1,1,0]
S5 = [1,0,1,0]
S6 = [0,0,1,0]
S7 = [0,1,1,0]
S8 = [0,1,0,0]
S9 = [0,1,0,1]
S10 = [0,1,1,1]
S11 = [0,0,1,1]
S12 = [1,0,1,1]
S13 = [1,1,1,1]
S14 = [1,1,0,1]
S15 = [1,0,0,1]
S16 = [0,0,0,1]

SommetsK4 = [S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,S13,S14,S15,S16]

evalue(LIK4[123],S15)

def evalue2(f,pq):
```

```
    return [evalue(f,w) for w in pq]

def N_to_Q(n):
    return SommetsK4[n-1]

def PaireN_to_PaireQ(pn):
   return [N_to_Q(a) for a in pn]

"""
    par_x_coord : les coordonnées des arçtes parallélles à Ox

"""


Par_x_coord = [PaireN_to_PaireQ(w) for w in par_x]
Par_y_coord = [PaireN_to_PaireQ(w) for w in par_y]
Par_z_coord = [PaireN_to_PaireQ(w) for w in par_z]
Par_t_coord = [PaireN_to_PaireQ(w) for w in par_t]

"""
   On revient des 0 et 1 aux sommets numérotés

"""

def Q_to_N(w):
    return 1+SommetsK4.index(w)

def PaireQ_to_PaireN(pq):
   return [Q_to_N(a) for a in pq]

def evalueN(f,pn):
        return PaireQ_to_PaireN(evalue2(f,PaireN_to_PaireQ(pn)))

"""

evalueN(LIK4[123],[1,2]) = [14, 3] (on trouve pareil à la main)

On veut maintenant appliquer evalueN(f,?) à un QSPSBC énoncé en codes-arêtes de
0 à 23

"""
def evalueNna(f,na):
    m = ParOfCode(na)
    return CodeOfPar(sorted(evalueN(f,m)))

def evalueQ(f,q):
    return sorted([evalueNna(f,a) for a in q])

#collision2 dit si les quadruplets (triés) q1 et q2 ont (au moins) 2 termes
communs
def collision2(q1,q2):
    if len(q1)<=1:
        return 0
    elif q1[0:2]==q2[0:2]:
        return 1
    else:
       return collision2(q1[1:],q2[1:])

"""
   Le test de evalueQ(LIK4[1],Q0[0]) = [0, 10, 19, 29] (Q0[0] était [0, 11, 18,
28])
    fait comprendre: les isométries de K4 ayant autre chose que 0 sur la
diagonale sont à éliminer
```

```
 .... non, zut il en faut deux

   Les isométries de LIK4 restantes vont former BIK4
"""

Q1 = [[0, 12, 22, 26],
      [1, 15, 20, 24],
      [2, 13, 23, 28],
      [3, 14, 18, 30],
      [4, 8, 21, 25],
      [5, 10, 19, 29],
      [6, 11, 17, 27],
      [7, 9, 16, 31]]


""" quand une isométrie laisse deux directions fixes elle ne transformera pas QO
en un autre utile"""

def bonneIK4(i):
    nbfixes =((i[0][0] == 0) + (i[1][0] == 1) + (i[2][0] == 2) + (i[3][0] == 3))
    return (nbfixes<=1)


BIK4 = [w for w in LIK4 if bonneIK4(w)] # il en reste 272


def testBon(oq1,oq2):
    res = 1
    for a in range(7):
        for b in range(7):
  #          print('a = ',a,'b = ',b)
            c2 = collision2(oq1[a],oq2[b])
  #         if c2:
  #              print('oq1(a) =',oq1[a],' oq2(b) =',oq2[b])
            res = res and (not c2)
    return res

premimQ0 = [evalueQ(BIK4[5],Q0[w]) for w in range(8)]

preQ2 = premimQ0

def essaye(k):
    im =  [evalueQ(BIK4[k],Q0[w]) for w in range(8)]
    if testBon(Q0,im) and  testBon(Q1,im):
        print(k)
        return im
    else:
        print(k,' rate')

#[essaye(w) for w in range(20)]


def VaAvec(k):
    ik =  [evalueQ(BIK4[k],Q0[w]) for w in range(8)]
    return [w for w in range(272) if ((w != k) and
testBon([evalueQ(BIK4[w],Q0[m]) for m in range(8)],ik))]


"""
   Ci-dessous je teste 'tout' : tous ceux compatibles avec avec un octo-
quadruplet
     On ne trouve que deux familles :
[1, 2, 4, 7, 8, 11, 13, 14, 16, 19, 21, 22, 25, 26, 28, 31, 33, 34, 36, 39, 40,
43, 45, 46, 49, 50, 52, 55, 56, 59, 61, 62, 65, 66, 68, 71, 72, 75, 77, 78, 80,
```

```
83, 85, 86, 89, 90, 92, 95, 97, 98, 100, 103, 104, 107, 109, 110, 112, 115, 117,
118, 121, 122, 124, 127, 129, 130, 132, 135, 136, 139, 141, 142, 144, 147, 149,
150, 153, 154, 156, 159, 161, 162, 164, 167, 168, 171, 173, 174, 177, 178, 180,
183, 184, 187, 189, 190, 192, 195, 197, 198, 201, 202, 204, 207, 209, 210, 212,
215, 216, 219, 221, 222, 225, 226, 228, 231, 232, 235, 237, 238, 240, 243, 245,
246, 249, 250, 252, 255, 257, 258, 260, 263, 264, 267, 269, 270]
   et
[0, 3, 5, 6, 9, 10, 12, 15, 17, 18, 20, 23, 24, 27, 29, 30, 32, 35, 37, 38, 41,
42, 44, 47, 48, 51, 53, 54, 57, 58, 60, 63, 64, 67, 69, 70, 73, 74, 76, 79, 81,
82, 84, 87, 88, 91, 93, 94, 96, 99, 101, 102, 105, 106, 108, 111, 113, 114, 116,
119, 120, 123, 125, 126, 128, 131, 133, 134, 137, 138, 140, 143, 145, 146, 148,
151, 152, 155, 157, 158, 160, 163, 165, 166, 169, 170, 172, 175, 176, 179, 181,
182, 185, 186, 188, 191, 193, 194, 196, 199, 200, 203, 205, 206, 208, 211, 213,
214, 217, 218, 220, 223, 224, 227, 229, 230, 233, 234, 236, 239, 241, 242, 244,
247, 248, 251, 253, 254, 256, 259, 261, 262, 265, 266, 268, 271]

  ceux d'une des familles 'vont' avec ceux de l'autre
      et comme elles sont disjointes, je suis coincé

VaAv = [[]]*272

VaAv[0] = VaAvec(0)
VaAv[1] = VaAvec(1)
VaAv[2] = VaAvec(2)
VaAv[3] = VaAvec(3)


for k in range(272):
    VaAv[k] = VaAvec(k)

"""

""" les listes prises par inter sont présumées triées, le résultat est
l'intersection """
def inter(l1,l2):
    if (l1 == [] or l2 == []):
            return []
    elif l1[0]==l2[0]:
        return [l1[0]]+inter(l1[1:],l2[1:])
    elif l1[0]<l2[0]:
        return inter(l1[1:],l2)
    else:
        return inter(l1,l2[1:])

import numpy as np

""" PADV = Pairesd'ArêtesDéjàVues """
# PADV = np.zeros((32, 32), dtype='b')

PADV = np.zeros((32, 32), dtype=int)

" dans cette version 10, enregistre écrit un nombre : celui du futur dessin """

def enregistre(a,b,s):
    PADV[a,b] = s

"""   on enregistre les cas de parallélisme  """

[enregistre(i,j,"20") for i in range(8) for j in range(8) if i<j]
[enregistre(i+8,j+8,"21") for i in range(8) for j in range(8) if i<j]
[enregistre(i+16,j+16,"22") for i in range(8) for j in range(8) if i<j]
[enregistre(i+24,j+24,"23") for i in range(8) for j in range(8) if i<j]
```

```
"""   on enregistre les cas de 'même coin'  """

for i in range(32):
    for j in range(32):
        if ((i<j) and (inter(ParOfCode(i),ParOfCode(j)) != [])):
    #           print('i ',i,' j ',j,' bout ',inter(ParOfCode(i),ParOfCode(j)))
          enregistre(i,j,inter(ParOfCode(i),ParOfCode(j))[0])


"""   on enregistre les cas de 'QO'  """

for o in range(8):
    for i in range(4):
        for j in range(4):
            if i<j:
                enregistre(Q0[o][i],Q0[o][j],o+30)


"""   on enregistre les cas de 'Q1'  """

for o in range(8):
    for i in range(4):
        for j in range(4):
            if i<j:
                enregistre(Q1[o][i],Q1[o][j],o+40)


def verifie():
    compte = 0
    for i in range(32):
        for j in range(32):
            if PADV[i,j]:
                compte +=1
    return compte


""" NU (NombreUtiles) prend un quadruplet et dit combien de paires sont non déjà
vues """

def NumPADV(p):
    [x,y] = p
    if PADV[x,y]:
        return 0
    else:
        return 1

def NU(q):
    [m,n,o,p] = q
    paires =[[m,n],[m,o],[m,p],[n,o],[n,p],[o,p]]
    [p1,p2,p3,p4,p5,p6] = paires
    return (NumPADV(p1) + NumPADV(p2) + NumPADV(p3) + NumPADV(p4) + NumPADV(p5)
+ NumPADV(p6))


""" on montre les q de NU égal à voulu  """
def montreNU(q,voulu):
    if (NU(q) >= voulu):
        return [q," ",NU(q)]

def fouillecodeL(v,max):
     for k in range(76): print(montreNU(codeL[v][k],max))

def ListefouillecodeL(v,max):
     Res = []
```

```
     for k in range(76):
         if montreNU(codeL[v][k],max) != None:
             Res = Res + [montreNU(codeL[v][k],max)[0]]
     return Res

LF = ListefouillecodeL

# for k in range(len(codeL[0])): print(montreNU(codeL[2][k],6))
""" la nouvelle stratégie est gloutonne :
      -  on va chercher le premier des NU valant 6 (dans l'ordre')
      -  on l'enregistre
      -  on recommence
"""

"""
des utilitaires de listes  :
    insere(c,n) : c est une carte (liste déjà triée), n une nouverlle valeur à y
mettre
    map :
    VireDoubon : dédoublonne une liste
    VireZero   : retire les zéros en tête de liste
    inter      : intersection de listes (déjà triées)

"""

def insere(c,n):
    if c==[]:
        return [n]
    elif c[0]<n:
        return [c[0]]+insere(c[1:],n)
    elif c[0]==n:
         return ["erreur insère"]
    else:
        return [n]+c

def map(f,l):
   return  [f(x) for x in l]

def VireDoublon(l):
    if len(l) <= 1:
            return l
    else:
        if l[0]==l[1]:
            return VireDoublon(l[1:])
        else:
             return [l[0]]+VireDoublon(l[1:])

def VireZero(l):
    if l[0]==0:
        return VireZero(l[1:])
    else:
        return l

def inter(l1,l2):
    if ((l1==[]) or (l2==[])):
        return []
    else:
        if l1[0]==l2[0]:
            return [l1[0]]+inter(l1[1:],l2[1:])
        elif l1[0]<l2[0]:
             return inter(l1[1:],l2)
        else:
             return inter(l1,l2[1:])
```

```
""" Incompréhension : avec VireZero(VireDoublon(sorted(res))) ça marche
        avec VireDoublon(VireZero(sorted(res))) ça ne marche pas : le zéro du
début resrte  """

def carte(k):
    res = []
    for i in range(k):
        res = res+[PADV[i,k]] #    print(res)
    for i in range(k+1,32):
         res = res+[PADV[k,i]]
    return VireZero(VireDoublon(sorted(res)))

def Cartes():
    return [carte(k) for k in range(32)]

MC = Cartes


def enregistreL(l,n):
    enregistre(l[0],l[1],n)

""" desenregistre numérodeCarte,n de tous les emplacements de la carte numC, et
enregistre dans carte(numC) """
def desenregistreL(numC,n):
    for u in range(0,numC):
        if PADV[u,numC]==n:
            PADV[u,numC]=0
    for u in range(numC+1,32):
        if PADV[numC,u]==n:
            PADV[numC,u]=0
    carte(numC)

# TabInter crée l'énorme tableau des intersections
def TabInter():
    return [[inter(Cartes()[u],Cartes()[v]) for u in range(32) if u<v ] for v in
range(32)]

TI = TabInter


def Inter(u,v):
    return inter(carte(u),carte(v))

def ChercheDoublon():
    for u in range(32):
        for v in range(32):
            if ((u<v) and len(Inter(u,v))>1):
                print("doublon  ",u,"  ",v)

CD = ChercheDoublon

def KompteDoublon():
      K=0
      for u in range(32):
        for v in range(32):
            if ((u<v) and len(Inter(u,v))>1):
                K+=1
      return K

KD = KompteDoublon

""" CompteOccurence(numC,n) dit combien de fois n figure dans carte(numC) """

def CompteOccurence(numC,n):
```

```
    co = 0
    for u in range(0,numC):
        if PADV[u,numC]==n:
            co+=1
    for u in range(numC+1,32):
        if PADV[numC,u]==n:
            co+=1
    return co

CO = CompteOccurence

def bon1(x):
    return  (PADV[x[0],x[1]]==0)

def bon2(x):
    return  (inter(carte(x[0]),carte(x[1])) == [])


def bon(x):
   # print(x,"   ",bon1(x) and bon2(x))
    return  bon1(x) and bon2(x)


""" attention : avec [enregistreL(x,z) for x in paires if bon(x)] ne marche
pas :
      les enregistrements sont faits dans l'ordre et 'bon' n'est plus bon  """

def enregistreQ(q,z):
    [m,n,o,p] = q
    paires = [[m,n],[m,o],[m,p],[n,o],[n,p],[o,p]]   #    [print(x) for x in
paires if PADV[x[0],x[1]]==0]
    bonnespaires = [u for u in paires if bon(u)]
    [enregistreL(x,z) for x in bonnespaires]
    print(bonnespaires)
    [insere(carte(x[0]),x[1]) for x in bonnespaires]
    [insere(carte(x[1]),x[0]) for x in bonnespaires]


def desenregistreQ(q,z):
       [desenregistreL(x,z) for x in q]

"""
def enregistreQ(q,z):
    [m,n,o,p] = q
    paires = [[m,n],[m,o],[m,p],[n,o],[n,p],[o,p]]   #    [print(x) for x in
paires if PADV[x[0],x[1]]==0]
    bonnespaires = [u for u in paires if bon(u)]
    [enregistreL(x,z) for x in bonnespaires]

"""


""" on enregistre ainsi de taille 6 : avant verifie() dit 304 """

enregistreQ([0,10,21,30],50) # verifié : 310
enregistreQ([0,15,19,27],51) # verifié : 316
enregistreQ([1,12,17,29],52) # verifié : 322
enregistreQ([2,9,22,29],53)  # verifié : 328
enregistreQ([3,8,20,27],54)
enregistreQ([4,14,17,24],55)
enregistreQ([5,11,22,24],56)
enregistreQ([5,12,16,28],57)
enregistreQ([6,10,20,26],58)
enregistreQ([7,13,19,26],59) # vérifié : 364
```

```
"""   on enregistre ainsi de taille 'présumée' 5 :  """

enregistreQ([0,13,23,31],60) # vérifié 369 BON et conflit 42 et 60 dans carte 13
et carte 23 quand on
desenregistreL(13,60)        # verifie() passe à 367 (c'est bon)

enregistreQ([1,11,21,27],61) #   conflit 46 et 61 dans carte 11 et carte 27
verifie() = 370
desenregistreL(11,61)

enregistreQ([2,14,16,25],62)
desenregistreL(16,62)        #  ..... vérifié 373

enregistreQ([3,12,19,25],63)
desenregistreL(19,63)        #  ..... vérifié 376

enregistreQ([3,13,18,31],64)
desenregistreL(3,64)        #  ..... vérifié 379

enregistreQ([3,14,21,26],65)
desenregistreL(3,65)         #  ..... vérifié 382

enregistreQ([4,8,18,29],66)
desenregistreL(4,66)         #  ..... vérifié 385

enregistreQ([4,9,16,30],67)
desenregistreL(16,67)        #  ..... vérifié 388

enregistreQ([6,15,18,24],68)
desenregistreL(15,68)        #  ..... vérifié 391

enregistreQ([7,8,23,28],69)
desenregistreL(28,69)        #  ..... vérifié 394

"""on enregistre ainsi de taille 'présumée' 4 :  """


enregistreQ([1,10,22,30],70)
desenregistreL(10,70)        # vérifié 397 collision, on retire 10,70

enregistreQ([1,11,23,25],71)
desenregistreL(1,71)         # ..... vérifié 400

enregistreQ([1,15,17,30],72) # vérifié 403  (pas de doublon)

enregistreQ([2,9,20,28],73)
desenregistreL(2,73)         #  ..... vérifié 406

enregistreQ([3,13,22,31],75)
desenregistreL(13,75)        # ..... vérifié 419

enregistreQ([5, 8, 18, 25],76)
desenregistreL(8,76)         # ..... vérifié 412

enregistreQ([6,12,19,25],77)
desenregistreL(12,77)        # ..... vérifié 415


"""on enregistre ainsi de taille  'présumée' 3 :  """

enregistreQ([1,10,23,24],80)
desenregistreL(1,80)        # ..... vérifié 415
```

```
enregistreQ([2,15,16,28],81)
desenregistreL(28,81)         # ..... vérifié 421

enregistreQ([4,9,19,28],82)  # vérifié 424

def tentative(q,n):
    enregistreQ(q,n)
    if (KD() != 0):
        desenregistreQ(q,n)
    else:
        return verifie()

def TraiteListeFouilleCode(lf,n):
    if lf != []:
        if tentative(lf[0],n)==None:
            TraiteListeFouilleCode(lf[1:],n)
        else:
            print(n)
            TraiteListeFouilleCode(lf[1:],n+1)
    else:
        print("fini")

TLFC = TraiteListeFouilleCode

"""on enregistre ainsi de taille 2 : il n'y en a aucun ....  """

"""
enregistreQ([1,11,20,25],90)
enregistreQ([1,15,16,28],91)  # vérifié 474
enregistreQ([2,9,21,31],92)   # vérifié 476
enregistreQ([2,9,23,24],93)   # vérifié 478
enregistreQ([2,12,16,26],94)  # vérifié 480
enregistreQ([2,13,17,24],95)  # vérifié 481
enregistreQ([5,8,18,25],96)   # vérifié 483
enregistreQ([5,15,18,29],97)  # vérifié 484
enregistreQ([6,10,16,31],98)  # vérifié 486
"""

def cherche():
    for i in range(32):
        for j in range(32):
            if ((i<j) and (PADV[i,j] == 0)):
                  print(i," ",j)

"""on enregistre enfin de taille 1 :  """

def ListeTrous():
    Z=[]
    for i in range(32):
        for j in range(32):
            if ((i<j) and (PADV[i,j] == 0)):
                  Z=Z+[[i,j]]
    return Z

def TraiteListeTrous(lt,n):
    if lt != []:
        enregistre(lt[0][0],lt[0][1],n)
        TraiteListeTrous(lt[1:],n+1)
    else:
        print("fini")

TraiteListeTrous(ListeTrous(),425)

TaillesCartes = [len(x) for x in MC()]
```

```
""" les tailles sont entre 11 et 14 """

def VD(l):
    if len(l)<=1:
        return l
    elif l[0]==l[1]:
        return VD(l[1:])
    else:
        return [l[0]]+VD(l[1:])

def colle(l):
    if len(l)<=1:
        return l[0]
    else:
        return l[0]+colle(l[1:])

NombreDessins = len(VD(sorted(colle(MC()))))

""" il y en a 138 """

#LF01 = LF(0,1)
#TLFC(LF01,90)

# avec LF01 on va jusqu'à 444, les codes vont jusqu'à 130

"""
enregistre(3,15,100)
enregistre(4,11,101)
enregistre(5,9,102)
enregistre(6,8,103)
enregistre(7,10,104)
enregistre(7,27,105)
enregistre(8,22,106)
enregistre(12,27,107)
enregistre(13,18,108)
enregistre(14,19,109)
"""
```