Numéro d'inscription : 13753

Table des matières :

main.py

```
01| import os
02| os.chdir("C:/Users//OneDrive/Documents/TIPE/")
03|
04| import numpy as np
05| from PIL import Image
06| import matplotlib.pyplot as plt
07| from squelettisation import squelettisation
08| from extraction_mdp import extraction_mdp
09| from robustesse_mdp import robustesse
10|
11| ##Dimension d'une matrice
12|
13| def dimension(mat : list) -> (int, int):
14|     '''dimension(mat) renvoie les dimensions de la matrice mat : (lignes,
colonnes).'''
15|     assert type(mat) == np.ndarray or type(mat) == list
16|     return len(mat), len(mat[0])
17|
18| ##Ouverture des images
19|
20| def lit_valeurs_nb(nom_de_fichier : str) -> list:
21|     '''lit_valeurs_nb(nom_de_fichier) ouvre l'image nom_de_fichier et la renvoie,
en noir et blanc.'''
22|     assert type(nom_de_fichier) == str
23|     print("Ouverture de l'image : " + nom_de_fichier)
24|     im = Image.open("C:/Users/ffore/OneDrive/Documents/TIPE/Base_donnees/" +
nom_de_fichier)
25|     print("Taille de l'image : ", im.size)
26|     print("Mode : ", im.mode)
27|     print("Format : ", im.format)
28|     return formatage(np.array(im.convert('L')))
29|
30| def formatage(l : np.array) -> list:
31|     '''formatage(l) transforme l'array numpy l en une liste où chaque pixel est
codé sur 3 nombres identiques (nuances de gris, RGB).'''
32|     assert type(l) == np.ndarray
33|     n, p = dimension(l)
34|     val = [[[] for _ in range(1, p - 1)] for _ in range(1, n - 1)]
35|     for i in range(1, n - 1):
36|         for j in range(1, p - 1):
37|             temp = int(l[i][j])
38|             val[i - 1][j - 1] = [temp] * 3
39|     return val
40|
41| ##Traitement
42|
43| def traitement(nom : str) -> None:
44|     assert type(nom) == str
45|     val = lit_valeurs_nb(nom)
46|     squelettisation(val)
47|     mdp = extraction_mdp(val)
48|     print("Robustesse : ", robustesse(mdp))
49|     plt.clf()
50|     plt.imshow(val)
51|     plt.show()
52|     return mdp
```

squelettisation.py

```
001| import os
002| os.chdir("C:/Users//OneDrive/Documents/TIPE/")
003|
004| from math import sqrt, erf, sqrt, pi
005| from copy import deepcopy
006| from pile_priorite import *
007| from binarisation import binarisation
008|
009| ##Dimension d'une matrice
010|
011| def dimension(mat : list) -> (int, int):
012|     '''dimension(mat) renvoie les dimensions de la matrice mat : (lignes,
colonnes).'''
013|     return len(mat), len(mat[0])
014|
015| ##Algorithmes de calculs
016|
017| def moyenne(l : list) -> float:
018|     '''moyenne(l) renvoie la moyenne de l (tous les poids sont égaux à 1).'''
019|     assert type(l) == list
020|     n = len(l)
021|     m = 0
022|     for i in range(n):
023|         m = m + l[i]
024|     return m / n
025|
026| def ecart_type(l : list, moy : float) -> float:
027|     '''ecart_type(l, moy) renvoie l'écart type des éléments de l à la moyenne
moy.'''
028|     assert type(l) == list and type(moy) == float
029|     e = 0
030|     n = len(l)
031|     for p in l:
032|         e = e + (p - moy) ** 2
033|     return sqrt(e / n)
034|
035| def tau(alpha : float, sigma : float, mu : float, x : int, n : int) -> float:
036|     '''tau(alpha, sigma, mu, x, n) effectue le calcul de tau, nécessaire à
l'évaluation du paramètre local de la squelettisation.'''
037|     assert type(alpha) == float and type(sigma == float) and type(mu) == float
and type(x) == int and type(n) == int and n > 1
038|     return fncr_rec((1 - (1 - alpha) ** (1/(n - 1))) * fncr((x - mu) / sigma))
039|
040| def fncr(x : float) -> float:
041|     '''f est la fonction de répartition de la loi normale centrée réduite.'''
042|     assert type(x) == float
043|     return 1 / 2 + erf(x / sqrt(2)) / 2
044|
045| def fncr_rec(x : float) -> float:
046|     '''fncr_rec est la réciproque de fncr.'''
047|     assert type(x) == float
048|     return sqrt(2) * erf_rec(2 * x - 1)
049|
050| def erf_rec(x : float) -> float:
051|     '''erf_rec est la réciproque de la fonction erf (erreur).'''
052|     assert type(x) == float
053|     return 1 / 2 * sqrt(pi) * (x + pi / 12  * (x ** 3) + 7 * (pi ** 2) * (x ** 5)
/ 480  + 127 * (pi ** 3) * (x ** 7) / 40320 + 4369 * (pi ** 4) * (x ** 9) / 5806080 +
34807 * (pi ** 5) / 182476800 * (x ** 11))
054|
055| ##Algorithme utile
056|
057| def coordonnees_voisins(i : int, j : int) -> list:
058|     '''coordonnees_voisins(i, j) renvoie la liste des coordonnées des voisins de
(i,j), en partant du pixel en haut à gauche, et en tournant dans le sans anti-
horaire.'''
```

```
059|     assert type(i) == int and type(j) == int
060|     return [(i-1, j-1), (i, j-1), (i+1, j-1), (i+1, j), (i+1, j+1), (i, j+1),
(i-1, j+1), (i-1, j)]
061|
062| ##Points particuliers
063|
064| def coupe_sombre(l : list, i : int, j : int) -> (list, list, list):
065|     '''coupe_sombre(l, i, j) renvoie la coupe sombre du pixel de coordonnées i,j,
les coordonnées de ses voisins les plus sombres et une liste de booléens, représentant
les composantes sombres qui sont 4-connexes.'''
066|     assert type(l) == list and type(i) == int and type(j) == int
067|     coupe, coord_sombre, t, est_4_conn = [], [], [], []
068|     verite = False
069|     co_voisins = coordonnees_voisins(i, j)
070|     for x in range(len(co_voisins)):
071|         a, b = co_voisins[x]
072|         if l[a][b][0] < l[i][j][0]:
073|             t.append(l[a][b][0])
074|             coord_sombre.append((a,b))
075|             if x % 2 == 1: #la composante est 4-connexe
076|                 verite = True
077|         elif t != []:
078|             coupe.append(t)
079|             t = []
080|             est_4_conn.append(verite)
081|             verite = False
082|     if t != []:
083|         assert len(coupe) == len(est_4_conn)
084|         if l[i - 1][j - 1][0] < l[i][j][0] and len(coupe) >= 1:
085|             for k in t:
086|                 coupe[0].append(k)
087|             est_4_conn[0] = est_4_conn[0] or verite
088|         else:
089|             coupe.append(t)
090|             est_4_conn.append(verite)
091|     return coupe, coord_sombre, est_4_conn
092|
093| def extremite(l : list, i : int, j : int) -> bool:
094|     '''extremite(l, i, j) renvoie True si le pixel (i,j) est extrémité.'''
095|     assert type(l) == list and type(i) == int and type(j) == int
096|     coupe, _, _ = coupe_sombre(l, i, j)
097|     return len(coupe) == 1 and len(coupe[0]) == 7
098|
099| def pic(l : list, i : int, j : int) -> bool:
100|     '''pic(l, i, j) renvoie True si le pixel (i, j) est pic.'''
101|     assert type(l) == list and type(i) == int and type(j) == int
102|     coupe, _, _ = coupe_sombre(l, i, j)
103|     return len(coupe) == 1 and len(coupe[0]) == 8
104|
105| def simple(l : list, i : int, j : int) -> bool:
106|     '''simple(l, i, j) renvoie True si le pixel (i,j) est simple.'''
107|     assert type(l) == list and type(i) == int and type(j) == int
108|     coupe, _, est_4_conn = coupe_sombre(l, i, j)
109|     return len(coupe) == 1 and True in est_4_conn and len(coupe[0]) != 8
110|
111| ##Algorithmes de traitements des impuretés
112|
113| def nettoie(l : list) -> None:
114|     '''nettoie(l) nettoie le squelette de la liste l.'''
115|     assert type(l) == list
116|     print("Nettoyage des impuretés.")
117|     enleve_simple(l)
118|     enleve_isole(l)
119|     enleve_aberrations(l)
120|     enleve_simple(l)
121|     enleve_isole(l)
122|     return None
123|
```

```
124| def enleve_isole(l : list) -> None:
125|     '''enleve_isole(l) enlève les points isolés (blanc et que des voisins noirs)
de la liste.'''
126|     assert type(l) == list
127|     n, p = dimension(l)
128|     for i in range(1, n - 1):
129|         for j in range(1, p - 1):
130|             co = coordonnees_voisins(i, j)
131|             if l[i][j][0] == 255 and [l[a][b][0] for (a,b) in co].count(255) ==
0:
132|                 l[i][j] = [0,0,0]
133|     return None
134|
135| def enleve_aberrations(l : list) -> None:
136|     '''enleve_aberrations(l) enlève les aberrations du squelette de la liste
l.'''
137|     assert type(l) == list
138|     n, p = dimension(l)
139|     for i in range(1, n - 1):
140|         for j in range(1, p - 1):
141|             if l[i][j][0] == 255 and extremite(l, i, j):
142|                 remonte_aberrations(l, i, j, [])
143|     return None
144|
145| def remonte_aberrations(l : list, i : int, j : int, ab: list) -> None:
146|     '''remonte_aberrations(l, i, j, ab) parcours l'aberration du pixel i,j,
remplit la liste de l'aberration ab et si elle a une longueur plus petite que c
lorsque l'algorithme arrive à une bifurcation ou une terminaison, elle met à 0 tous
les pixels de l'aberration.'''
147|     assert type(l) == list and type(i) == int and type(j) == int and type(ab) ==
list
148|     n, p = dimension(l)
149|     if len(ab) >= 20:
150|         return None
151|     else:
152|         if i == n - 1 or i == 0 or j == p - 1 or j == 0:
153|                 for a,b in ab:
154|                     l[a][b] = [0,0,0]
155|         else:
156|             co = [(i - 1, j), (i, j - 1), (i, j + 1), (i + 1, j)]
157|             ent = [l[x][y][0] for (x,y) in co]
158|             if (ent.count(255) == 1 and len(ab) != 0) or (ent.count(255) != 2 and
ent.count(255) != 1):
159|                 for a,b in ab:
160|                     l[a][b] = [0,0,0]
161|             else:
162|                 for a,b in co:
163|                     if l[a][b][0] == 255 and not ((a,b) in ab):
164|                         ab.append((i, j))
165|                         remonte_aberrations(l, a, b, ab)
166|     return None
167|
168| def enleve_simple(l : list) -> None:
169|     '''enleve_simple(l) enlève les points simples et non extrémités de l.'''
170|     assert type(l) == list
171|     n, p = dimension(l)
172|     for i in range(1, n - 1):
173|         for j in range(1, p - 1):
174|             if simple(l, i, j) and not extremite(l, i, j):
175|                 l[i][j] = [0,0,0]
176|     return None
177|
178| ##Squelettisation
179|
180| def abaissable(l : list, initiale : list, i : int, j : int, alpha : float) ->
bool:
181|     '''abaissable(l, initiale, i, j, alpha) renvoie true si le pixel (i,j) est
abaissable (le paramètre lambda est calculé localement), avec l la liste évoluant,
```

```
initiale la liste des valeurs initiales, et alpha le paramètre de précision, fixé par
défaut à 10 ** -3.'''
    assert type(l) == list and type(i) == int and type(j) == int and type(alpha)
== float
    coupe, coord_sombre, est_4_conn = coupe_sombre(l, i, j)

    #nombre de composantes connexes
    k = len(coupe)

    #liste de tous les pixels plus sombres avec le pixel i,j
    coupe_sombre_complete = [l[i][j][0]]
    for a in coupe:
        for b in a:
            coupe_sombre_complete.append(b)

    #nombre de pixels plus sombres
    n = len(coupe_sombre_complete)

    if k == 1:

        #liste des valeurs initiales des pixels plus sombres et du pixel i,j
        coupe_initiale = [initiale[z][r][0] for z,r in coord_sombre]
        coupe_initiale.append(initiale[i][j][0])

        mu = moyenne(coupe_initiale)
        sigma = ecart_type(coupe_initiale, mu)

        if n == 9:
            #pic
            if sigma == 0:
                return min(coupe_initiale) >= mu
            return min(coupe_initiale) >= mu + sigma * tau(alpha, sigma, mu,
initiale[i][j][0], n)
        elif n == 8:
            #extrémité
            if sigma == 0:
                return min(coupe_initiale) >= mu
            return min(coupe_initiale) >= mu + sigma * tau(alpha, sigma, mu,
initiale[i][j][0], n)
        else:
            return False
    elif k >= 2 and est_4_conn.count(True) >= 2:
        #crête
        co_voisins = coordonnees_voisins(i, j)
        #liste des composantes 4_connexes
        comp_4_conn = [coupe[a] if est_4_conn else _ for a in range(k)]

        #k représente désormais le nombre de composantes 4_connexes
        k = len(comp_4_conn)

        #valeurs du voisinage en prenant les valeurs courantes
        voisinage_courant = [l[a][b][0] for (a,b) in co_voisins]
        voisinage_courant.append(l[i][j][0])

        #valeurs du voisinage en prenant les valeurs initiales
        voisinage_initial = [initiale[a][b][0] for (a,b) in co_voisins]
        voisinage_initial.append(initiale[i][j][0])

        mu = moyenne(voisinage_courant)
        sigma = ecart_type(voisinage_initial, moyenne(voisinage_initial))

        if k == 2:
            alpha0 = 0.0316
        elif k == 3:
            alpha0 = 0.0184
        elif k == 4:
            alpha0 = 0.0130
        o = 0
```

```
245|         for x in range(len(comp_4_conn)):
246|             if sigma == 0:
247|                 if min(comp_4_conn[x]) >= mu:
248|                     o = o + 1
249|             elif min(comp_4_conn[x]) >= mu + sigma * tau(alpha0, sigma, mu, l[i]
[j][0], len(comp_4_conn[x]) + 1):
250|                 o = o + 1
251|         return o >= k - 1
252|     else:
253|         return False
254|
255| def squelettisation(l : list, alpha : int = 10**-3) -> None:
256|     '''squelettisation(l, alpha) effectue la squelettisation paramétrée de la
liste l avec la précision alpha, valant à défaut 10**-3 (modifie l), et binarise
l'image obtenue.'''
257|     assert type(l) == list and type(alpha) == float
258|     print("Squelettisation de l'image.")
259|     initiale = deepcopy(l)
260|     pile_prio = pile_priorite_vide()
261|     n, p = dimension(l)
262|     for i in range(1, n - 1):
263|         for j in range(1, p - 1):
264|             if abaissable(l, initiale, i, j, alpha) or (simple(l, i, j) and not
extremite(l, i, j)):
265|                 empile(pile_prio, (i,j), l[i][j][0])
266|     while not est_vide(pile_prio):
267|         (i, j) = depile(pile_prio)
268|         coordonnees = coordonnees_voisins(i, j)
269|         if abaissable(l, initiale, i, j, alpha) or (simple(l, i, j) and not
extremite(l, i, j)):
270|             l[i][j] = [max([l[a][b][0] for a,b in coordonnees if l[a][b][0] <
l[i][j][0]])] * 3
271|         for a,b in coordonnees:
272|             if (a != 0 and a != n - 1 and b != 0 and b != p - 1) and
(abaissable(l, initiale, a, b, alpha) or (simple(l, a, b) and not extremite(l, a,
b))):
273|                 empile(pile_prio, (a, b), l[a][b][0])
274|     binarisation(l)
275|     nettoie(l)
276|     return None
```

```
# pile_priorite.py

01| ##Une pile de priorité est une liste de deux éléments : une liste de 256 listes,
représentant les différentes priorités de 0 à 255, et un entier représentant l'indice
de la liste de plus petite priorité. L'entier est None si la pile est vide.
02|
03| def pile_priorite_vide():
04|     '''Créer une pile de priorité vide : [[[], ..., []], None].'''
05|     return [[[] for _ in range(256)], None]
06|
07| def empile(pile_prio, pixel : tuple, p : int):
08|     '''Empile pixel sur la pile de priorité, à la priorité p.'''
09|     assert p >= 0 and p < 256
10|     if pixel in pile_prio[0][p]:
11|         return None
12|     pile_prio[0][p].append(pixel)
13|     if pile_prio[1] == None:
14|         pile_prio[1] = p
15|     else:
16|         pile_prio[1] = min(p, pile_prio[1])
17|     return None
18|
19| def est_vide(pile_prio):
20|     '''Renvoie True si la pile de priorité est vide, False sinon.'''
21|     return pile_prio[1] == None
22|
23| def depile(pile_prio):
24|     '''Dépile l'élément de plus petite priorité.'''
25|     temp = pile_prio[1]
26|     if len(pile_prio[0][pile_prio[1]]) == 1:
27|         i = pile_prio[1] + 1
28|         while i < 256 and pile_prio[0][i] == []:
29|             i = i + 1
30|         if i == 256:
31|             pile_prio[1] = None
32|         else:
33|             pile_prio[1] = i
34|     return pile_prio[0][temp].pop()
```

binarisation.py

```
001| ##Dimension d'une matrice
002|
003| def dimension(mat : list) -> (int, int):
004|     '''dimension(mat) renvoie les dimensions de la matrice mat : (lignes,
colonnes).'''
005|     return len(mat), len(mat[0])
006|
007| ##Algorithmes de calcul
008|
009| def somme(l : list) -> float:
010|     '''somme(l) renvoie la somme des éléments de l.'''
011|     assert type(l) == list
012|     s = 0
013|     for k in l:
014|         s = s + k
015|     return s
016|
017| def moy_pond(l : list, p : int) -> float:
018|     '''moy_pond(l , p) effectue une moyenne pondérée de la liste m, sachant que
le premier élément de la liste l est le pixel p.'''
019|     assert type(l) == list and type(p) == int
020|     s = somme(l)
021|     if len(l) == 0 or s == 0:
022|         return 0
023|     moy = 0
024|     for k in range(len(l)):
025|         moy += (k + p) * l[k]
026|     return moy / s
027|
028| def indice_max(l : list) -> int:
029|     '''indice_max(l) renvoie l'indice de la première occurence du maximum de la
liste l.'''
030|     assert type(l) == list
031|     m = 0
032|     for k in range(1, len(l)):
033|         if l[k] > l[m]:
034|             m = k
035|     return m
036|
037| def decoupe(l :list, taille : int) -> list:
038|     '''decoupe(l, taille) découpe la liste l, représentant une image, en bloc de
taille * taille pixels.'''
039|     ligne = []
040|     n, p = dimension(l)
041|     for i in range(n//taille + 1):
042|         temp = l[taille * i: taille * (i + 1)]
043|         bloc = []
044|         for j in range(p//taille + 1):
045|             att = []
046|             for k in range(len(temp)):
047|                 att.append(temp[k][taille * j: taille * (j + 1)])
048|             bloc.append(att)
049|         ligne.append(bloc)
050|     return ligne
051|
052| ##Algorithmes de binarisation
053|
054| def histogramme(l : list) -> list:
055|     '''histogramme(l) renvoie l'histogramme de la liste l.'''
056|     assert type(l) == list
057|     h = [0 for _ in range(256)]
058|     for i in l:
059|         for j in i:
060|             h[j[0]] += 1
061|     return h
062|
```

```
063| def seuil(l : list, s : int) -> None:
064|     '''seuil(l, s) effectue un seuillage de la liste l au niveau s (modifie l en
place).'''
065|     assert type(l) == list and type(s) == int
066|     n, p = dimension(l)
067|     for i in range(n):
068|         for j in range(p):
069|             if l[i][j][0] <= s:
070|                 l[i][j] = [0, 0, 0]
071|             else:
072|                 l[i][j] = [255, 255, 255]
073|     return None
074|
075| def otsu(l : list) -> int:
076|     '''otsu(l) renvoie le seuil détecté avec la méthode d'Otsu.'''
077|     assert type(l) == list
078|     histo = histogramme(l)
079|     w1, w2, mu1, mu2 = 0, 1, 0, moy_pond(histo, 0)
080|     sigma = [w1 * w2 * (mu1 - mu2) ** 2]
081|     total = somme(histo)
082|     for i in range(1,256):
083|         temp = somme(histo[i:])
084|         w1 = (total - temp)/ total
085|         w2 = temp / total
086|         mu1 = moy_pond(histo[:i],0)
087|         mu2 = moy_pond(histo[i:],i)
088|         sigma.append(w1 * w2 * (mu1 - mu2) ** 2)
089|     return indice_max(sigma)
090|
091| def binarisation(l : list, taille : int = 20) -> None:
092|     '''binarisation(l, taille) effectue la binarisation de l avec la méthode
d'Otsu (modifie l en place) en découpant la liste l en sous listes représentants des
images de taille * taille pixels.'''
093|     assert type(l) == list and type(taille) == int
094|     print("Binarisation du squelette.")
095|     ligne = decoupe(l, taille)
096|     c,d = dimension(ligne)
097|     for a in range(c):
098|         for b in range(d):
099|             seuil(ligne[a][b], otsu(ligne[a][b]))
100|     n, p = dimension(l)
101|     for i in range(n):
102|         for j in range(p):
103|             l[i][j] = ligne[i//taille][j//taille][i%taille][j%taille]
104|     return None
```

extraction_mdp.py

```
001| import os
002| os.chdir("C:/Users//OneDrive/Documents/TIPE/")
003|
004| from math import sqrt, acos, pi
005| from hachage import md5
006| from code_ascii import tables_ascii
007|
008| ##Algorithmes de calculs
009|
010| def barycentre(ter : list, bif : list) -> ((float, float), (float, float)):
011|     '''barycentre(ter, bif) renvoie les coordonnées des barycentres des
terminaisons et des bifurcations.'''
012|     assert type(ter) == list and type(bif) == list
013|     xb, yb = 0, 0
014|     xt, yt = 0, 0
015|     nb, nt = len(bif), len(ter)
016|     assert nb != 0 and nt != 0
017|     for i,j in bif:
018|         xb += i
019|         yb += j
020|     for i,j in ter:
021|         xt += i
022|         yt += j
023|     return (xt / nt, yt / nt), (xb / nb, yb / nb)
024|
025| def crossing_number(l : list, i : int, j : int) -> int:
026|     '''crossing_number(l, i, j) renvoie le crossing number (le nombre de
transitions noir/blanc et blanc/noir) voisinage du pixel (i,j) de l.'''
027|     assert type(l) == list and type(i) == int and type(j) == int
028|     voisins = []
029|     coordonnees = [(i-1, j-1), (i, j-1), (i+1, j-1), (i+1, j), (i+1, j+1), (i,
j+1), (i-1, j+1), (i-1, j)]
030|     for m, p in coordonnees:
031|         voisins.append(l[m][p][0])
032|     temp = voisins[-1]
033|     a = 0
034|     for k in voisins:
035|         if k != temp:
036|             a += 1
037|             temp = k
038|     return a
039|
040| def somme(l : list) -> float:
041|     '''somme(l) renvoie la somme des éléments de l.'''
042|     assert type(l) == list
043|     s = 0
044|     for k in l:
045|         s = s + k
046|     return s
047|
048| ##Algorithmes de tris
049|
050| def decoupe(l : list) -> (list, list):
051|     '''decoupe(l) coupe la liste l en 2 listes.'''
052|     assert type(l) == list
053|     n = len(l)
054|     return l[:n//2], l[n//2:]
055|
056| def fusion(l1 : list, l2 : list) -> list:
057|     '''fusion(l1, l2) fusionne les listes l1 et l2 supposées triées par ordre
croissant en une liste elle aussi triée par ordre croissant.'''
058|     assert type(l1) == list and type(l2) == list
059|     if l1 == []:
060|         return l2
061|     elif l2 == []:
062|         return l1
```

```
063|     else:
064|         l = []
065|         while l1 != [] and l2 != []:
066|             if l1[0][0] <= l2[0][0]:
067|                 l.append(l1[0])
068|                 l1 = l1[1:]
069|             else:
070|                 l.append(l2[0])
071|                 l2 = l2[1:]
072|         if l1 != []:
073|             for k in l1:
074|                 l.append(k)
075|         elif l2 != []:
076|             for k in l2:
077|                 l.append(k)
078|         return l
079|
080| def tri_fusion(l : list) -> (list, list):
081|     '''tri_fusion(l) effectue le tri de la liste l avec la méthode de tri fusion,
où l est une liste de couple, le tri étant basé sur la première composante de chaque
couple.'''
082|     assert type(l) == list
083|     if l == [] or len(l) == 1:
084|         return l
085|     else:
086|         l1, l2 = decoupe(l)
087|         return fusion(tri_fusion(l1), tri_fusion(l2))
088|
089| ##Changement d'écritures
090|
091| def hex_dec(mess : str) -> int:
092|     ''''hex_dec(mess) transforme le mess de l'héxadécimal vers la base 10.'''
093|     assert type(mess) == str
094|     dico = {'0' : 0, '1' : 1, '2' : 2, '3' : 3, '4' : 4, '5' : 5, '6' : 6, '7' :
7, '8' : 8, '9' : 9, 'a' : 10, 'b' : 11, 'c' : 12, 'd' : 13, 'e' : 14, 'f' : 15}
095|     s = 0
096|     n = len(mess)
097|     for k in range(n):
098|         s = s + dico[mess[len(mess) - 1 - k]] * 16 ** k
099|     return s
100|
101| def base_mdp(nbi : int) -> str:
102|     '''base_mdp(nbi) transforme l'entier nbi en un mot de passe en utilisant la
base 95.'''
103|     assert type(nbi) == int
104|     dic_ascii, _ = tables_ascii()
105|     mdp = ""
106|     temp = nbi
107|     while temp != 0:
108|         a = str(temp % 95 + 32)
109|         while len(a) < 3:
110|             a = '0' + a
111|         mdp = dic_ascii[a] + mdp
112|         temp = temp // 95
113|     return mdp
114|
115| ##Algorithmes principaux
116|
117| def terminaison_bifurcation(l : list) -> (list, list):
118|     '''termminaison_bifurcation(l) renvoie les listes des coordonnées des pixels
terminaisons et bifurcations de l.'''
119|     assert type(l) == list
120|     mat_cro_num, ter, bif = [], [], []
121|     for i in range(1, len(l) - 1):
122|         temp = []
123|         for j in range(1, len(l[0]) - 1):
124|             if l[i][j] == [255,255,255]:
125|                 temp.append(crossing_number(l, i, j))
```

```
126|             else:
127|                 temp.append(0)
128|         mat_cro_num.append(temp)
129|     for i in range(1, len(l) - 1):
130|         for j in range(1, len(l[0]) - 1):
131|             if mat_cro_num[i - 1][j - 1] == 2:
132|                 ter.append((i, j))
133|             elif mat_cro_num[i - 1][j - 1] == 6 or mat_cro_num[i - 1][j - 1] ==
8:
134|                 bif.append((i,j))
135|     return ter, bif
136|
137| def extraction_mdp(l: list) -> str:
138|     '''extraction_mdp(l) renvoie le mot de passe extrait de l, où l est une liste
représentant une image binarisée.'''
139|     print("Recherche des minuties et extraction du mot de passe.")
140|     ter, bif = terminaison_bifurcation(l)
141|     print(len(bif), " bifurcations trouvées.")
142|     print(len(ter), " terminaisons trouvées.")
143|     dter, dbif = [], []
144|     (xbt, ybt), (xbb, ybb) = barycentre(ter, bif)
145|
146|     abif = []
147|     for i,j in bif:
148|         dx, dy = i - xbb, j - ybb
149|         r = sqrt(dx ** 2 + dy ** 2)
150|         assert r != 0
151|         dbif.append(int(r))
152|         teta = abs(acos(dy / r))
153|         if dx < 0:
154|             teta = (- teta) % (2 * pi)
155|         abif.append(teta)
156|
157|     assert len(dbif) == len(abif)
158|     assert len(dbif) >= 6
159|
160|     pour_tri = [(dbif[i], abif[i]) for i in range(len(dbif))]
161|     l_coupleb = tri_fusion(pour_tri)[:6]
162|     dbif = [x for (x, y) in l_coupleb]
163|     encore_pour_tri = [(y,x) for (x,y) in l_coupleb]
164|     l_coupleb2 = tri_fusion(encore_pour_tri)
165|     abif = [x for (x,_) in l_coupleb2]
166|
167|     angle_bif = [int((2 * pi - abif[len(abif) - 1] + abif[0]) * 10) / (10)]
168|     for k in range(len(abif) - 1):
169|         angle_bif.append(int((abif[k + 1] - abif[k]) * 10) / (10))
170|     assert len(angle_bif) == 6
171|     assert somme(angle_bif) >= 5.8 and somme(angle_bif) <= 6.6
172|
173|     encore_tri = [(angle_bif[i], None) for i in range(len(angle_bif))]
174|     l_coupleb2 = tri_fusion(encore_tri)
175|     angle_bif = [l_coupleb2[i][0] for i in range(len(l_coupleb2))]
176|
177|     ater = []
178|     for i,j in ter:
179|         dx, dy = i - xbb, j - ybb
180|         r = sqrt(dx ** 2 + dy ** 2)
181|         assert r != 0
182|         dter.append(int(r))
183|         teta = abs(acos(dy / r))
184|         if dx < 0:
185|             teta = (- teta) % (2 * pi)
186|         ater.append(teta)
187|
188|     assert len(dter) == len(ater)
189|     assert len(dter) >= 6
190|
191|     pour_tri = [(dter[i], ater[i]) for i in range(len(dter))]
```

```
    l_couplet = tri_fusion(pour_tri)[:6]
    dter = [x for (x,_) in l_couplet]
    encore_pour_tri = [(y,x) for (x,y) in l_couplet]
    l_couplet2 = tri_fusion(encore_pour_tri)
    ater = [x for (x,_) in l_couplet2]

    angle_ter = [int((2 * pi - ater[len(abif) - 1] + ater[0]) * 10) / (10)]
    for k in range(len(ater) - 1):
        angle_ter.append(int((ater[k + 1] - ater[k]) * 10) / (10))
    assert len(angle_ter) == 6
    assert somme(angle_ter) >= 5.8 and somme(angle_ter) <= 6.6

    encore_tri = [(angle_ter[i], None) for i in range(len(angle_ter))]
    l_couplet2 = tri_fusion(encore_tri)
    angle_ter = [l_couplet2[i][0] for i in range(len(l_couplet2))]

    chaine = ""
    for liste in [dter, angle_ter, dbif, angle_bif]:
        for i in liste:
            car = str(int(i))
            while len(car) != 3:
                car = "0" + car
            chaine = chaine + car

    return base_mdp(hex_dec(md5(chaine)))
```

robustesse_mdp.py

```
01| ##Evaluation de la robustesse d'un mot de passe
02|
03| minuscule = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n',
'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
04| majuscule = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N',
'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']
05| chiffre =  ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
06| caracteres_speciaux = [' ', '!', '"', '#', '$', '%', '&', "'", '(', ')', '*', '+',
',', '-', '.', '/', '[', '\\', ']', '^', '_', '`', '{', '|', '}', '~', '@']
07|
08| def robustesse(mdp : str):
09|     """robustesse(mdp) renvoie la robustesse de mdp, c'est-à-dire une note sur 5
représentant la difficulté que le pirate rencontrera lors de la tentative de cassage
de mdp (5 étant la difficulté maximale).
10|     La note est nulle si mdp a une longueur inférieure ou égale à 8, ou s'il
apparaît dans un dictionnaire, et est sinon calculée comme suit: on ajoute 0.5 à
chaque fois que la longueur augmmente de 1 à partir de 8, et on ajoute 0.125 pour
chaque type de caractères différents, avec un maximum de 2 * 0.125 = 0.25 par type
(correspondant à deux caractères par type)."""
11|     n = len(mdp)
12|     if n >= 16:
13|         n = 16
14|     if n <= 8 or dico(mdp):
15|         return 0.0
16|     else:
17|         nbtype = [0, 0, 0, 0]
18|         for k in mdp:
19|             if k in minuscule:
20|                 nbtype[0] = min([nbtype[0] + 1,2])
21|             elif k in majuscule:
22|                 nbtype[1] = min([nbtype[1] + 1,2])
23|             elif k in chiffre:
24|                 nbtype[2] = min([nbtype[2] + 1,2])
25|             else:
26|                 nbtype[3] = min([nbtype[3] + 1,2])
27|         return 0.5 * (n  - 8) + (nbtype[0] + nbtype[1] + nbtype[2] + nbtype[3]) /
8
28|
29| def dico(mdp : str):
30|     """dico(mdp) renvoie True si mdp appartient à l'un des dictionnaires de la
base de donnée, False sinon."""
31|     fid1 = open("C:/Users//Desktop/TIPE/Base_donnee/Dictionnaires/français.txt",
'r', encoding = 'UTF-8')
32|     lignes1 = fid1.readlines()
33|     fid1.close()
34|     fid2 = open("C:/Users//Desktop/TIPE/Base_donnee/Dictionnaires/
mot_de_passe.txt", 'r', encoding = 'UTF-8')
35|     lignes2 = fid2.readlines()
36|     fid2.close()
37|     return not (mdp in lignes1 or mdp in lignes2)
```

```
# code_ascii.py

01| def tables_ascii():
02|     '''Renvoie les dictionnaires des principaux caractères associés à leur code
ascii et inversement.'''
03|     fid = open("C:/Users/ffore/OneDrive/Documents/TIPE/Base_donnees/ascii.txt",
'r')
04|     lignes = fid.readlines()
05|     dic_char = {}
06|     dic_code = {}
07|     for k in range(len(lignes) // 2): #il y a un nombre paire de lignes
08|         dic_char[lignes[2 * k + 1].strip("\n")] = lignes[2 * k].strip("\n")
09|         dic_code[lignes[2 * k].strip("\n")] = lignes[2 * k + 1].strip("\n")
10|     fid.close()
11|     return dic_code, dic_char
```

hachage.py

```
001| import os
002| os.chdir("C:/Users//OneDrive/Documents/TIPE/")
003|
004| from code_ascii import tables_ascii
005| from math import sin, floor
006|
007| dic_code, dic_char = tables_ascii()
008|
009| ##Opérateurs sur les nombres binaires
010|
011| def et(a : str, b : str) -> str:
012|     '''et(a, b) est l'opérateur "et logique termes à termes" entre a et b.
013|     Conditions : a et b sont deux chaînes de caractères de même longueur,
représentant des nombres binaires.'''
014|     assert type(a) == str and type(b) == str and len(a) == len(b)
015|     mot = ""
016|     for i in range(len(a)):
017|         if a[i] == "1" and b[i] == "1":
018|             mot = mot + "1"
019|         else:
020|             mot = mot + "0"
021|     return mot
022|
023| def ou(a : str, b : str) -> str:
024|     '''ou(a, b) est l'opérateur "ou logique termes à termes" entre a et b.
025|     Conditions : a et b sont deux chaînes de caractères de même longueur,
représentant des nombres binaires.'''
026|     assert type(a) == str and type(b) == str and len(a) == len(b)
027|     mot = ""
028|     for i in range(len(a)):
029|         if a[i] == "0" and b[i] == "0":
030|             mot = mot + "0"
031|         else:
032|             mot = mot + "1"
033|     return mot
034|
035| def non(a : str) -> str:
036|     '''non(a) est l'opérateur "négation termes à termes" de a.
037|     Conditions : a est un chaîne de caractères représentant un nombre binaire.'''
038|     assert type(a) == str
039|     mot = ""
040|     for i in range(len(a)):
041|         mot = mot + str((int(a[i]) + 1) % 2)
042|     return mot
043|
044| def xou(a : str, b : str) -> str:
045|     '''xou(a, b) est l'opérateur "ou exclusif logique termes à termes" entre a et
b.
046|     Conditions : a et b sont deux chaînes de caractères de même longueur,
représentant des nombres binaires.'''
047|     assert type(a) == str and type(b) == str and len(a) == len(b)
048|     mot = ""
049|     for i in range(len(a)):
050|         mot = mot + str((int(a[i]) + int(b[i])) % 2)
051|     return mot
052|
053| def rotationg(mess : str, n : int) -> str:
054|     '''rotationg(mess, n) effectue une rotation de n bits vers la gauche sur le
message mess.
055|     Conditions : mess est une chaîne de caractères, représentant un nombre
binaire et n un entier.'''
056|     assert type(n) == int and type(mess) == str
057|     n = n % len(mess)
058|     code = mess[n:] + mess[:n]
059|     return code
060|
```

```
061| ##Changements d'écritures
062|
063| def str_ascii(mdpi : str) -> str:
064|     '''str_ascii(mdpi) convertit mdpi, une chaine de caractères, en convention
ASCII.'''
065|     assert type(mdpi) == str
066|     mdpc = ""
067|     for i in mdpi:
068|         mdpc = mdpc + dic_char[i]
069|     return mdpc
070|
071| def dec_binaire(a : int, n : int) -> str:
072|     '''dec_binaire(a, n) convertit a de la base 10 vers la base 2 sur n bits.'''
073|     assert type(a) == int and type(n) == int and 2 ** n > a
074|     (q, r) = (0, a)
075|     mdp = ''
076|     for i in range(n):
077|         q = r // 2 ** (n - 1 - i)
078|         mdp = mdp + str(q)
079|         r = r % 2 ** (n - 1 - i)
080|     return mdp
081|
082| def binaire_dec(b : str) -> int:
083|     '''binaire_dec(b) convertit b de la base 2 vers la base 10.'''
084|     assert type(b) == str
085|     somme = 0
086|     n = len(b)
087|     for l in range(n):
088|         somme = somme + int(b[n - 1 - l]) * 2 ** l
089|     return somme
090|
091| def endian(i : int, n : int) -> str:
092|     '''endian(i, n) convertit l'entier i de la base 10 vers la base 2 sur n bits
avec la convention little-endian'''
093|     assert type(i) == int and type(n) == int
094|     mdpi = dec_binaire(i, n)
095|     mdp = ""
096|     for k in range(0, len(mdpi) - 7, 8):
097|         mdp = mdpi[k:k + 8] + mdp
098|     return mdp
099|
100| ##Algorithme de découpe
101|
102| def decoupage(mot : str, n : int) -> list:
103|     '''decoupage(mot, n) découpe la chaîne de caractères mot en blocs de n
caractères.'''
104|     assert type(mot) == str and type(n) == int and n * (len(mot) // n) ==
len(mot)
105|     return [mot[r * n : n + r * n] for r in range(len(mot) // n)]
106|
107| ##Algorithmes d'initialisations
108|
109| def padding(mdpi : str, longueur : int) -> str:
110|     '''padding(mdpi, longueur) effectue l'étape de padding, et renvoie mdpi,
auquel on a rajouté le bit 1, et autant de bits 0 que nécessaire pour avoir une
longueur égale à 448 modulo 512, ainsi que la longueur du mot de passe initial en
binaire (chaque caractère est représenté sur 8 bits) et avec la convention little
endian.'''
111|     assert type(mdpi) == str and type(longueur) == int
112|     mdp = mdpi + "1"
113|     while (len(mdp) % 512) != 448:
114|         mdp = mdp + "0"
115|     long = endian(longueur * 8, 64)
116|     return mdp + long
117|
118| def initialisation(mdpi : str) -> tuple:
119|     '''initialisation(mdpi) initialise les différentes constantes de
l'algorithme, et effectue l'étape de padding.'''
```

```
120|     h0 = "01100111010001010010001100000001"
121|     h1 = "11101111110011011010101110001001"
122|     h2 = "10011000101110101101110011111110"
123|     h3 = "00010000001100100101010001110110"
124|     r = [7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22, 5,  9,
14, 20,  5, 9,
125|         14, 20,  5,  9, 14, 20,  5,  9, 14, 20, 4, 11, 16, 23,  4, 11, 16, 23,
4, 11, 16, 23,
126|         4, 11, 16, 23, 6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15,
21]
127|     k = []
128|     mdpbin = ""
129|     mdpascii = str_ascii(mdpi)
130|     list_car = [mdpascii[p:p + 3] for p in range(0, len(mdpascii) - 2, 3)]
131|     list_bin = [dec_binaire(int(u), 8) for u in list_car]
132|     for o in list_bin:
133|         mdpbin = mdpbin + o
134|     for i in range(64):
135|         k.append(floor(abs(sin(i + 1)) * 2 ** 32))
136|     return h0, h1, h2, h3, r, k, padding(mdpbin, len(mdpi))
137|
138| ##Algorithme prinicipal
139|
140| def md5(mdpi : str):
141|     '''md5(mdpi) renvoie l'empreinte numérique du mot de passe mdpi par la
fonction de hachage md5.'''
142|     h0, h1, h2, h3, r, k, mdpbin = initialisation(mdpi)
143|     bloc_512 = decoupage(mdpbin, 512)
144|     for p in bloc_512:
145|         bloc_32 = decoupage(p, 32)
146|         bloc_32 = [endian(binaire_dec(bloc_32[k]),32) for k in
range(len(bloc_32))]
147|         a = h0
148|         b = h1
149|         c = h2
150|         d = h3
151|         for i  in range(64):
152|             if i >= 0 and i <= 15:
153|                 f = ou(et(b, c), et(non(b), d))
154|                 g = i
155|             elif i >= 16 and i <= 31:
156|                 f = ou(et(d, b), et(non(d), c))
157|                 g = (5 * i + 1) % 16
158|             elif i >= 32 and i <= 47:
159|                 f = xou(xou(b, c), d)
160|                 g = (3 * i + 5) % 16
161|             elif i >= 48 and i <= 63:
162|                 f = xou(c, ou(b, non(d)))
163|                 g = (7 * i) % 16
164|             tempo = d
165|             d = c
166|             c = b
167|             z = rotationg(dec_binaire((binaire_dec(a) + binaire_dec(f) + k[i] +
binaire_dec(bloc_32[g])) % (2 **32),32), r[i])
168|             b = dec_binaire((binaire_dec(z) + binaire_dec(b)) % (2 ** 32) ,32)
169|             a = tempo
170|         h0 = dec_binaire((binaire_dec(h0) + binaire_dec(a)) % (2 ** 32), 32)
171|         h1 = dec_binaire((binaire_dec(h1) + binaire_dec(b)) % (2 ** 32), 32)
172|         h2 = dec_binaire((binaire_dec(h2) + binaire_dec(c)) % (2 ** 32), 32)
173|         h3 = dec_binaire((binaire_dec(h3) + binaire_dec(d)) % (2 ** 32), 32)
174|     h0 = endian(binaire_dec(h0), 32)
175|     h1 = endian(binaire_dec(h1), 32)
176|     h2 = endian(binaire_dec(h2), 32)
177|     h3 = endian(binaire_dec(h3), 32)
178|     mdp = (str(hex(binaire_dec(h0 + h1 + h2 + h3)))).strip("0x")
179|     while len(mdp) != 32:
180|         mdp = '0' + mdp
181|     return mdp
```

```
# table_arc_en_ciel.py

001| import os
002| os.chdir("C:/Users//OneDrive/Documents/TIPE/")
003|
004| from hachage import md5
005| from code_ascii import tables_ascii
006| from random import randint
007|
008| ##Algorithmes de calculs
009|
010| def hex_dec(mess : str) -> int:
011|     ''''hex_dec(mess) transforme le mess de l'héxadécimal vers la base 10.'''
012|     assert type(mess) == str
013|     dico = {'0' : 0, '1' : 1, '2' : 2, '3' : 3, '4' : 4, '5' : 5, '6' : 6, '7' :
7, '8' : 8, '9' : 9, 'a' : 10, 'b' : 11, 'c' : 12, 'd' : 13, 'e' : 14, 'f' : 15}
014|     s = 0
015|     n = len(mess)
016|     for k in range(n):
017|         s = s + dico[mess[len(mess) - 1 - k]] * 16 ** k
018|     return s
019|
020| def reduction(mess : str, x : str) -> str:
021|     '''reduction(mess, x) transforme l'empreinte mess en un nouveau mot de passe
commençant par x.'''
022|     assert type(mess) == str and type(x) == str
023|     while len(x) != 3:
024|         x = '0' + x
025|     mot = ""
026|     motd = hex_dec(mess)
027|     dico_ascii, _ = tables_ascii()
028|     while motd != 0:
029|         q = str(motd % 95 + 32)
030|         motd = motd // 95
031|         while len(q) != 3:
032|             q = '0' + q
033|         mot = dico_ascii[str(q)] + mot
034|     return dico_ascii[x] + mot
035|
036| def bout(mot : str) -> str:
037|     '''bout(mot) renvoie l'extrémité de la ligne de mot dans une table arc-en-
ciel, en effectuant 95 réductions et hachages successifs.'''
038|     assert type(mot) == str
039|     for k in range(95):
040|         mot = reduction(md5(mot), str(k + 32))
041|     return mot
042|
043| ##Algorithme de recherche
044|
045| def recherche(empr : str, table : str) -> bool or str:
046|     '''recherche(empr, table) recherche le mot de passe correspondant à
l'empreinte numérique empr dans la table table.'''
047|     assert type(empr) == str and type(table) == str
048|     nom = "C:/Users//OneDrive/Documents/TIPE/Base_donnees/" + table + ".txt"
049|     fid = open(nom, 'r')
050|     nbred = int(fid.readline().strip("\n"))
051|     lignes = fid.readlines()
052|     fid.close()
053|     l1 = []
054|     l2 = []
055|     h = 0
056|     verite = True
057|     for k in lignes:
058|         m = k.split("°")
059|         l1.append(m[1].strip("\n"))
060|         l2.append(m[0])
061|     i = nbred - 1
062|     while i != -1:
```

```
063|         mot = ''
064|         while not mot in l1 and i != -1:
065|             mot = reduction(empr, str(i + 32))
066|             for k in range(i + 1, nbred):
067|                 mot = reduction(md5(mot), str(k + 32))
068|             i -= 1
069|         if mot in l1:
070|             h = 0
071|             while verite and h < len(l1):
072|                 if l1[h] == mot:
073|                     mdp = l2[h]
074|                     for l in range(i):
075|                         mdp = reduction(md5(mdp), str(l + 32))
076|                     verite = not md5(mdp) == empr
077|                     if not verite:
078|                         i = -1
079|                 h += 1
080|         else:
081|             mdp = False
082|     return mdp
083|
084| ##Création de la tabla arc_en_ciel
085|
086| def creation(n : int) -> None:
087|     '''creation(n) construit la table arc-en-ciel avec la fonction md5 et des
fonctions de réductions qui sont des changements de base 95 en base 16, avec ajout
d'un caractère différent à chaque étape pour optimisation (ajoute n lignes si la table
existe déjà).'''
088|     nom = "C:/Users//OneDrive/Documents/TIPE/Base_donnees/table.txt"
089|     assert type(n) == int
090|     dico_ascii, _ = tables_ascii()
091|     dictio = {}
092|     fid = open(nom, "a")
093|     if fid.readlines() == []:
094|         fid.write("95\n")
095|     mdp = ""
096|     for k in range(n):
097|         o = randint(0,15)
098|         while len(mdp) != o:
099|            i = str(randint(32, len(dico_ascii) + 32 - 1))
100|            while len(i) != 3:
101|                i = '0' + i
102|            mdp = mdp + dico_ascii[i]
103|         if not mdp in dictio:
104|             print("aller")
105|             dictio[mdp] = k
106|             mdpi = mdp
107|             mdp = bout(mdp)
108|             fid.write(mdpi + "°" + mdp + "\n")
109|     fid.close()
110|     return None
```

collisions.py

```
01| import os
02| os.chdir("C:/Users//OneDrive/Documents/TIPE/")
03|
04| from hachage import md5
05|
06| def premiers_bits(mess : str, x : int, prefixe : str) -> str:
07|     '''premiers_bits(mess, x, prefixe) renvoie les 2x premiers bits de l'empreinte
numérique de la concaténation de préfixe et mess par md5.'''
08|     assert type(mess) == str and type(x) == int and type(prefixe) == str
09|     return md5(prefixe + mess)[:x * 2]
10|
11| def young(prefixe : str, nb : int) -> (str, str):
12|     '''young(prefixe, nb) renvoie deux mots de passe, ayant les nb premiers
caractères et les mêmes 2 * nb premiers bits une fois hachés par md5.'''
13|     assert type(prefixe) == str and type(nb) == int
14|     lent = premiers_bits(prefixe, nb, prefixe)
15|     rapide = premiers_bits(lent, nb, prefixe)
16|     while lent != rapide:
17|         lent = premiers_bits(lent, nb, prefixe)
18|         rapide = premiers_bits(premiers_bits(rapide, nb, prefixe), nb, prefixe)
19|     long_cycle = 0
20|     lent = prefixe
21|     while lent != rapide:
22|         m1 = rapide
23|         m0 = lent
24|         lent = premiers_bits(lent, nb, prefixe)
25|         rapide = premiers_bits(rapide, nb, prefixe)
26|         long_cycle += 1
27|     if long_cycle == 0:
28|         return "Aucune collision trouvée !\n"
29|     return prefixe + m0, prefixe + m1
```