```python
# prog.py

## MAIN

'''
découpage de l'algorithme en quatres étapes, détermination du seuillage, de la composante connexe, de l'histogramme et enfin des pics
il y a donc 4 parties qui suivent, chacune implémentant une étape en une ou plusieurs fonctions
'''

def main(img):
    imageFiltree = seuillageGris(img, 100)
    imageComp, composante = main_composantes(imageFiltree)
    histogramme = histo(img, imageComp, composante)
    l1, l2 = determinationLABEL(histogramme)
    return (l1,l2)

## FILTRE IMAGE

'''
algorithme de seuillage de l'image
'''

import cv2
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np



def seuillageGris(img, seuil):
    image = Image.open(img)
    ImageTab = np.array(image)
    imgHSI = cv2.cvtColor(ImageTab, cv2.COLOR_RGB2HSV)
    dim = imgHSI.shape
    res = np.zeros((dim[0],dim[1]))
    for i in range(dim[0]):
        for j in range(dim[1]):
            if imgHSI[i][j][2] < seuil:
                res[i,j] =  1
    return res

## COMPOSANTES CONNEXES

'''
Algo de reconaissance des composantes connexes d'un tableau composé de cases ayant soit 0 soit 1 pour valeur
image : tableau de 0 et 1
'''

import numpy as np

# Main

'''
découpage de cette étape en sous étape, car cette étape est la plus longue et la plus compliquée
On inverse d'abord l'image, puis on détermine les composantes connexes de l'image, puis celle qui nous intéresse
'''

def main_composantes(img):
    (a,b) = np.shape(img)
    inversionImage(img)
    determinationComposantes(img)
    interieur = determineInterieurCadre(img)
    return (img,interieur)

# renvoie le numéro de la plus grande composante connexe de l'inverse de l'image qui ne touche pas le bord, donc l'intérieur du cadre
def determineInterieurCadre(image):
    (a,b) = np.shape(image)
    composantes = [0]
    for i in range(a):
        for j in range(b):
            if image[i,j] not in composantes:
                composantes.append(image[i,j])
    composantesBord = [0]
    for i in range(a):
        if image[i,0] not in composantesBord:
            composantesBord.append(image[i,0])
        if image[i,b-1] not in composantesBord:
            composantesBord.append(image[i,b-1])
    for j in range(b):
        if image[0,j] not in composantesBord:
            composantesBord.append(image[0,j])
        if image[a-1,j] not in composantesBord:
```

```
081|             composantesBord.append(image[a-1,j])
082|     tailleComposantesPasBord = []
083|     for elem in composantes:
084|         if elem not in composantesBord:
085|             tailleComposantesPasBord.append([elem, 0])
086|     for i in range(a):
087|         for j in range(b):
088|             pixel = image[i,j]
089|             if Existe(tailleComposantesPasBord, pixel):
090|                 newVal = Recup(tailleComposantesPasBord, pixel) + 1
091|                 Modif(tailleComposantesPasBord, pixel, newVal)
092|     max = 0
093|     plusGrandeComposante = -1
094|     for couple in tailleComposantesPasBord:
095|         if couple[1] > max:
096|             max = couple[1]
097|             plusGrandeComposante = couple[0]
098|     return plusGrandeComposante
099|
100| def Recup(tab, cle):
101|     for elem in tab:
102|         if elem[0] == cle:
103|             return elem[1]
104|
105| def Modif(tab, cle, val):
106|     for elem in tab:
107|         if elem[0] == cle:
108|             elem[1] = val
109|
110| def Existe(tab, cle):
111|     for elem in tab:
112|         if elem[0] == cle:
113|             return True
114|     else:
115|         return False
116|
117| # Fonctions
118|
119| def triCorrespondances(tab):
120|     for k in range(len(tab)):
121|         temp = tab[k]
122|         tab[k] = tab[temp]
123|     k = 1
124|     while k < len(tab):
125|         continu = True
126|         val = tab[k]
127|         valinf = tab[k-1]
128|         if val > valinf:
129|             tab[k] = valinf + 1
130|         if k == len(tab) - 1:
131|             break
132|         while continu and k<len(tab):
133|             k += 1
134|             if tab[k] == val:
135|                 tab[k] = valinf + 1
136|             else:
137|                 continu = False
138|
139| def numsPrecedent(table, pixel, image):
140|     (i, j) = pixel
141|     res = []
142|     if i > 0 and image[i-1,j] == 1:
143|         res.append(HashRecup(table, (i-1,j)))
144|     if j > 0 and image[i,j-1] == 1:
145|         res.append(HashRecup(table, (i, j-1)))
146|     return res
147|
148| def inversionImage(image):
149|     (a,b) = np.shape(image)
150|     for i in range(a):
151|         for j in range(b):
152|             image[i,j] = (image[i,j] + 1)%2
153|
154| def determinationComposantes(image):
155|     (a,b) = np.shape(image)
156|     composantes = HashCreation(a,b)
157|     etiquette = 0
158|     correspondances = [0]
159|     for i in range(a):
160|         for j in range(b):
161|             if image[i,j] == 1:
162|                 etiquettesPrec = numsPrecedent(composantes, (i,j),image)
163|                 long = len(etiquettesPrec)
164|                 if long == 0:
165|                     HashAjout(composantes, (i,j), etiquette)
166|                     etiquette += 1
```

```
                    correspondances.append(etiquette)
                elif long == 1:
                    HashAjout(composantes, (i,j), etiquettesPrec[0])
                else:
                    num1 = int(etiquettesPrec[0])
                    num2 = int(etiquettesPrec[1])
                    HashAjout(composantes, (i,j), num1)
                    if num1 > num2:
                        correspondances[num1] = num2
                    elif num1 == num2:
                        None
                    else:
                        correspondances[num2] = num1
    triCorrespondances(correspondances)
    for i in range(a):
        for j in range(b):
            composantes[i,j] = correspondances[int(composantes[i,j])]
    for i in range(a):
        for j in range(b):
            if image[i,j] == 1:
                image[i,j] = HashRecup(composantes, (i,j)) + 1

# Dictionnaire sous forme de tableau (complexité en temps bien meilleure)

def HashCreation(a,b):
    res = np.zeros((a,b))
    for i in range(a):
        for j in range(b):
            res[i,j] = -1
    return res

def HashAjout(table, cle, elem):
    table[cle] = elem

def HashRecup(table, cle):
    return table[cle]

def HashExiste(table, cle):
    return table[cle] != -1

def HashModif(table, cle, newVal):
    table[cle] = newVal

def HashSuppr(table, cle):
    table[cle] = -1

## CONSTRUCTION DE L'HISTOGRAMME ET DETERMINATION DES PICS

import cv2
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np

def convertToHSI(img):
    image = Image.open(img)
    ImageTab = np.array(image)
    return cv2.cvtColor(ImageTab, cv2.COLOR_RGB2HSV)

def histo(img, imgConnex, n):
    ImHSV = convertToHSI(img)
    histogramme = np.zeros((180,2))
    for i in range(180):
        histogramme[i,0] = i
    dim = ImHSV.shape
    for i in range(dim[0]):
        for j in range(dim[1]):
            if imgConnex[i][j] == n :
                histogramme[ImHSV[i][j][0],1]+= 1
    return histogramme

import math

def tableauCorresp():
    tab = []
    for i in range(15):
        tab.append("ROUGE")
    for i in range(30):
        tab.append("JAUNE")
    for i in range(30):
        tab.append("VERT")
    for i in range(30):
        tab.append("TURQUOISE")
    for i in range(30):
        tab.append("BLEU")
    for i in range(30):
        tab.append("MAGENTA")
```

```
253|     for i in range(15):
254|         tab.append("ROUGE")
255|     return np.array(tab)
256|
257| def determinationPics(histogramme):
258|     premierPic = max(histogramme)
259|     deuxiemePic = max2(histogramme, premierPic)
260|     return (premierPic,deuxiemePic)
261|
262| def determinationLABEL(histogramme):
263|     p1,p2 = determinationPics(histogramme)
264|     t = tableauCorresp()
265|     print(p1,p2)
266|     return (t[int(p1)], t[int(p2)])
267|
268| def max(tab):
269|     max = -math.inf
270|     indice_max = -1
271|     for i in range(len(tab)):
272|         if tab[i,1] > max:
273|             max = tab[i,1]
274|             indice_max = tab[i,0]
275|     return indice_max
276|
277| def max2(tab,max1):
278|     max = -math.inf
279|     indice_max = -1
280|     for i in range(len(tab)):
281|         if (i < (max1 - 45) or i > (max1 + 45)) and (i < (max1 + 180 - 45) or i > (max1 - 180 + 45)) and
tab[i,1] > max:
282|             max = tab[i,1]
283|             indice_max = tab[i,0]
284|     return indice_max
```