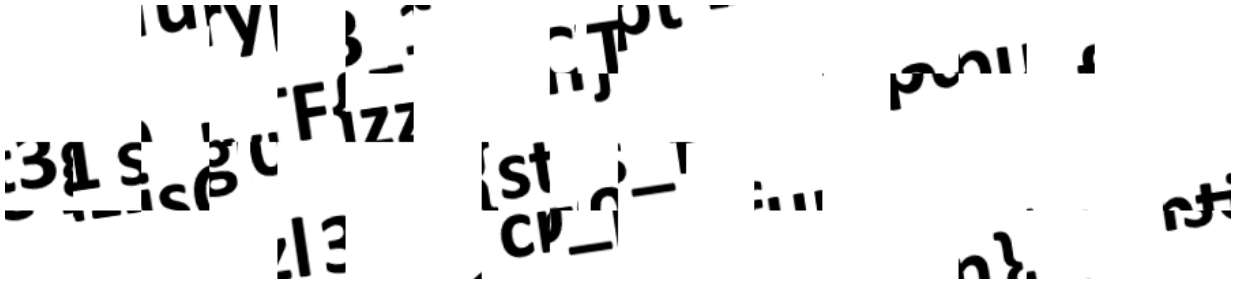
Stéganographie - Puzzle - 100 points

Kévin DUVERGER

Table des matières

1 Résolution Puzzle :

Voici l'image qui nous est donnée pour ce challenge :



Cette fois-ci, la consigne est plutôt simple : il faut résoudre le puzzle qui nous est donné.

Pour le résoudre, vous pouvez utiliser un logiciel de manipulation d'images (comme `paint.net` sous Windows) qui vous permettra de déplacer les pièces et de tenter de résoudre ce challenge.

Vous pouvez également tenter de le résoudre de façon automatique, l'idée étant de procéder en plusieurs étapes :

- 1. Ouverture de l'image et récupération des pièces (en 50 x 50 pixels).
- 2. On supprime toutes les pièces qui n'ont que des pixels blancs.
- 3. On trie les pièces par ordre décroissant du nombre de pixels noirs sur le contour.
- 4. On place la première pièce au milieu.
- 5. On prend le côté qui a le plus de pixels noirs et on prend la pièce qui matche le plus.
- 6. On place la pièce sur ce côté, puis on répète jusqu'à résolution.

L'idée de prendre les pièces avec le plus de pixels noir sur le contour puis de prendre les côtés avec le plus de pixels noir pour placer les pièces suivantes nous permet de diminuer au maximum la probabiliét de se tromper de pièce.

Voici le code :

```
import random

def specialScalarProduct(a, b) :
    if a == None or b == None :
        return 0
    sumToReturn = 0
    for i in range(0, len(a)) :
        if a[i] == b[i] :
            sumToReturn += 256 * 256
        else :
            sumToReturn -= 256
    return sumToReturn

def getNeighborsLines(cellY, cellX, piecesPositions, pieces, pieceSize) :
    neighboursToCheck = { "bottom" : None, "top" : None, "left" : None, "right" : None }
    if piecesPositions[cellY + 1][cellX] != -1 :
        pieceBelow = piecesPositions[cellY + 1][cellX]
        neighboursToCheck["bottom"] = [ pieces[pieceBelow][0][i] for i in range(0, pieceSize) ]
    if piecesPositions[cellY - 1][cellX] != -1 :
        pieceTop = piecesPositions[cellY - 1][cellX]
        neighboursToCheck["top"] = [ pieces[pieceTop][pieceSize - 1][i] for i in range(0, pieceSize)
     ]
    if piecesPositions[cellY][cellX - 1] != -1 :
        pieceLeft = piecesPositions[cellY][cellX - 1]
        neighboursToCheck["left"] = [ pieces[pieceLeft][i][pieceSize - 1] for i in range(0,
    pieceSize) ]
    if piecesPositions[cellY][cellX + 1] != -1 :
        pieceRight = piecesPositions[cellY][cellX + 1]
        neighboursToCheck["right"] = [ pieces[pieceRight][i][0] for i in range(0, pieceSize) ]
    return neighboursToCheck

def sortSpacesAroundByMostBlackPixels(pieceSize, pieces, piecesPositions, spacesAroundToTry) :
    # First, getting the values
    neighborsBlackPixelCount = []
    for i in range(0, len(spacesAroundToTry)) :
        neighboursToCheck = getNeighborsLines(spacesAroundToTry[i][0], spacesAroundToTry[i][1],
    piecesPositions, pieces, pieceSize)
        blackPixelCount = 0
        if neighboursToCheck["bottom"] != None :
            for i in range(0, len(neighboursToCheck["bottom"])) : blackPixelCount += 1 if
    neighboursToCheck["bottom"][i] != 255 else 0
        if neighboursToCheck["top"] != None :
```

```
            for i in range(0, len(neighboursToCheck["top"])) : blackPixelCount += 1 if neighboursToCheck["top"][i] != 255 else 0
        if neighboursToCheck["left"] != None :
            for i in range(0, len(neighboursToCheck["left"])) : blackPixelCount += 1 if neighboursToCheck["left"][i] != 255 else 0
        if neighboursToCheck["right"] != None :
            for i in range(0, len(neighboursToCheck["right"])) : blackPixelCount += 1 if neighboursToCheck["right"][i] != 255 else 0
        neighborsBlackPixelCount.append(blackPixelCount)

    # Then, sorting them
    for i in range(0, len(spacesAroundToTry)) :
        maxIndex = i
        for j in range(i + 1, len(spacesAroundToTry)) :
            if neighborsBlackPixelCount[j] > neighborsBlackPixelCount[maxIndex] :
                maxIndex = j
        temp1 = spacesAroundToTry[i]
        spacesAroundToTry[i] = spacesAroundToTry[maxIndex]
        spacesAroundToTry[maxIndex] = temp1
        temp2 = neighborsBlackPixelCount[i]
        neighborsBlackPixelCount[i] = neighborsBlackPixelCount[maxIndex]
        neighborsBlackPixelCount[maxIndex] = temp2

# Loading image
pieceSize     = 50
scrambled     = Image.open("scrambled.png")
width, height = scrambled.size

# Getting the pieces
pieces = []
for yPiece in range(0, height // pieceSize) :
    for xPiece in range(0, width // pieceSize) :
        currentPiece = [ ]
        for y in range(0, pieceSize) :
            currentPiece.append([])
            for x in range(0, pieceSize) :
                pixelValue = scrambled.getpixel((xPiece * pieceSize + x, yPiece * pieceSize + y))
                temp = (pixelValue[0] + pixelValue[1] + pixelValue[2]) // 3
                currentPiece[-1].append(0 if temp < 255 else 255)
        pieces.append(currentPiece)

# Removing white pieces
i = 0
removedCount = 0
while i < len(pieces) :
    isWhite = True
    for j in range(0, pieceSize) :
        for k in range(0, pieceSize) :
            if pieces[i][j][k] != 255 :
                isWhite = False
                break
        if not isWhite :
            break
    if isWhite :
        print("Removing piece " + str(i + 1 + removedCount))
        pieces = pieces[:i] + pieces[i+1:]
        removedCount += 1
    else :
        i += 1

# Now, we sort the pieces by decreasing amount of black pixels on the contour
blackPixelsOnContourForEach = []
for i in range(0, len(pieces)) :
    nonWhiteCount = 0
    for j in range(0, pieceSize) :
        nonWhiteCount += 1 if pieces[i][0][j] != 255 else 0
        if j > 0 : nonWhiteCount += 1 if pieces[i][j][0] != 255 else 0
        if j > 0 : nonWhiteCount += 1 if pieces[i][pieceSize - 1][j] != 255 else 0
        if j > 0 and j < pieceSize - 1 : nonWhiteCount += 1 if pieces[i][j][pieceSize - 1] != 255 else 0
    blackPixelsOnContourForEach.append(nonWhiteCount)
# Sorting
for i in range(0, len(pieces)) :
    minIndex = i
    for j in range(i + 1, len(pieces)) :
        if blackPixelsOnContourForEach[j] < blackPixelsOnContourForEach[minIndex] :
            minIndex = j
    temp1 = blackPixelsOnContourForEach[i]
    blackPixelsOnContourForEach[i] = blackPixelsOnContourForEach[j]
```

```
    blackPixelsOnContourForEach[j] = temp1
    temp2 = pieces[i]
    pieces[i] = pieces[j]
    pieces[j] = temp2
# Printing those pieces to an image
tempImage = Image.new("RGB", ((pieceSize + 1) * len(pieces), pieceSize))
for i in range(0, len(pieces)) :
    for y in range(0, pieceSize) :
        for x in range(0, pieceSize) :
            tempImage.putpixel((x + i * (pieceSize + 1), y), (pieces[i][y][x], pieces[i][y][x], pieces[i][y][x]))
    for j in range(0, pieceSize) :
        tempImage.putpixel((pieceSize + i * (pieceSize + 1), j), (255, 0, 0))
tempImage.save("blacksSorted.png")

# Now, we put this piece in the center and try to put all other ones around
step = 0
threshold = 100000
placedPieces = [ False for i in range(0, len(pieces)) ]
placedPieces[0] = True
piecesPositions = [ [ -1 for j in range(0, 50) ] for i in range(0, 10) ]
piecesPositions[5][25] = 0
spacesAroundToTry = [ [ 6, 25 ], [ 5, 26 ], [ 5, 24 ], [ 4, 25 ] ]
sortSpacesAroundByMostBlackPixels(pieceSize, pieces, piecesPositions, spacesAroundToTry)
while len(spacesAroundToTry) > 0 :
    # First, finding the intervals to try
    currentSpaceToTry = spacesAroundToTry[0]
    neighboursToCheck = getNeighborsLines(currentSpaceToTry[0], currentSpaceToTry[1], piecesPositions, pieces, pieceSize)
    spacesAroundToTry = spacesAroundToTry[1:]

    # Then, finding the scores for each piece on each side
    scoresPerPiece = []
    for i in range(0, len(pieces)) :
        if placedPieces[i] != True :
            scoreBottom = specialScalarProduct(neighboursToCheck["bottom"], [ pieces[i][pieceSize - 1][j] for j in range(0, pieceSize) ])
            scoreTop    = specialScalarProduct(neighboursToCheck["top"],    [ pieces[i][0][j] for j in range(0, pieceSize) ])
            scoreLeft   = specialScalarProduct(neighboursToCheck["left"],   [ pieces[i][j][0] for j in range(0, pieceSize) ])
            scoreRight  = specialScalarProduct(neighboursToCheck["right"],  [ pieces[i][j][pieceSize - 1] for j in range(0, pieceSize) ])
            scoresPerPiece.append(scoreBottom + scoreTop + scoreLeft + scoreRight)
        else :
            scoresPerPiece.append(0)

    # Getting the piece with the highest score and placing it here
    maxIndex = 0
    for i in range(0, len(pieces)) :
        if scoresPerPiece[i] > scoresPerPiece[maxIndex] :
            maxIndex = i
    if scoresPerPiece[maxIndex] > threshold :
        placedPieces[maxIndex] = True
        piecesPositions[currentSpaceToTry[0]][currentSpaceToTry[1]] = maxIndex
        if not neighboursToCheck["bottom"] and [ currentSpaceToTry[0] + 1, currentSpaceToTry[1] ] not in spacesAroundToTry :
            spacesAroundToTry.append([ currentSpaceToTry[0] + 1, currentSpaceToTry[1] ])
        if not neighboursToCheck["top"] and [ currentSpaceToTry[0] - 1, currentSpaceToTry[1] ] not in spacesAroundToTry :
            spacesAroundToTry.append([ currentSpaceToTry[0] - 1, currentSpaceToTry[1] ])
        if not neighboursToCheck["left"] and [ currentSpaceToTry[0], currentSpaceToTry[1] - 1 ] not in spacesAroundToTry :
            spacesAroundToTry.append([ currentSpaceToTry[0], currentSpaceToTry[1] - 1 ])
        if not neighboursToCheck["right"] and [ currentSpaceToTry[0], currentSpaceToTry[1] + 1 ] not in spacesAroundToTry :
            spacesAroundToTry.append([ currentSpaceToTry[0], currentSpaceToTry[1] + 1 ])
        sortSpacesAroundByMostBlackPixels(pieceSize, pieces, piecesPositions, spacesAroundToTry)

    # Increasing step count
    step += 1

# Saving the image, we first need to find the size
minXCell, maxXCell = 50, 0
minYCell, maxYCell = 10, 0
for i in range(0, 10) :
    for j in range(0, 50) :
        if piecesPositions[i][j] != -1 :
            if i < minYCell : minYCell = i
```

```
            if i > maxYCell : maxYCell = i
            if j < minXCell : minXCell = j
            if j > maxXCell : maxXCell = j
width  = (maxXCell - minXCell + 1) * pieceSize
height = (maxYCell - minYCell + 1) * pieceSize
# Then, we can write into it
tempImage = Image.new("RGB", (width, height), (255, 255, 255))
for yCell in range(minYCell, maxYCell + 1) :
    for xCell in range(minXCell, maxXCell + 1) :
        if piecesPositions[yCell][xCell] >= 0 :
            for y in range(0, pieceSize) :
                for x in range(0, pieceSize) :
                    color = (pieces[piecesPositions[yCell][xCell]][y][x], pieces[piecesPositions[
    yCell][xCell]][y][x], pieces[piecesPositions[yCell][xCell]][y][x])
                    tempImage.putpixel((x + (xCell - minXCell) * pieceSize, y + (yCell - minYCell) *
     pieceSize), color)
tempImage.save("solved.png")
```

La solution que l'on obtient est la suivante :