Stéganographie élémentaire

1 Décomposition des entiers en base 2

Question 1

Ecrire une fonction `int_to_8bits` qui prend en argument un entier n compris entre 0 et 255 et renvoie la liste $[\epsilon_0, \epsilon_1, \dots, \epsilon_7]$ des 8 premiers bits de sa décomposition en base 2.

```
def int_to_8bits(n):
    L=[]; r=n
    for i in range(8):
        L.append(r%2)
        r=r//2
    return L
```

Question 2

Ecrire la fonction réciproque `bits_to_int` qui recalcule n à partir de la liste de ses bits.

La fonction suivante fonctionne quelle que soit la longueur de la liste des bits de l'entier représenté.

```
def bits_to_int(L):
    p=len(L); r=0
    for i in range(p):
        r=2*r+L[p-1-i]
    return r
```

2 Stéganographie en deux dimensions

Question 3

Ecrire une fonction `cache_dans_int` qui prend en argument deux entiers x et y compris entre 0 et 255, et un entier $p \in [\![1;8]\!]$. Si l'on note $[\epsilon_0, \epsilon_1, \dots \epsilon_7]$ la liste des 8 premiers bits de x et $[\mu_0, \dots, \mu_7]$ celle de y, la fonction renvoie l'entier z dont l'écriture en base 2 est

$$[\epsilon_{8-p}, \dots, \epsilon_7, \mu_p, \dots \mu_7]$$

```
def cache_dans_int(x,y,p):
    Lx=int_to_8bits(x)
    Ly=int_to_8bits(y)
    for i in range(p):
        Ly[i]=Lx[8-p+i]
    return bits_to_int(Ly)
```

Question 4

Ecrire une fonction `extrait` qui prend en argument un entier $z \in [\![0;255]\!]$ et un entier $p \in [\![1;8]\!]$. Si z admet pour écriture $[z_0, \dots, z_7]$, alors la fonction renvoie l'entier $\widetilde{x}$ dont l'écriture en base 2 est

$$[0, \dots, 0, z_0, \dots z_{p-1}]$$

```
def extrait(n,s):
    return bits_to_int(([0]*(8-s))+(int_to_8bits(n)[:s]))
```

Question 5

En déduire une fonction `cache_dans_image` prenant en argument deux fichiers images `originale` et `secret` et un entier $p \in [\![1;8]\!]$. La fonction modifie chaque pixel de l'image `originale` en cachant les p premiers bits de chaque composante du triplet `secret[i,j]` dans la composante correspondante du triplet `originale[i,j]`.

```
def cache_dans_image(secret,fichier,p):
    tf=fichier.size
    ts=secret.size
    for i in range(min(tf[0],ts[0])):
        for j in range(min(tf[1],ts[1])):
            (a,b,c)=fichier.getpixel((i,j))
            (x,y,z)=secret.getpixel((i,j))
            r=cache_dans_int(x,a,p)
            s=cache_dans_int(y,b,p)
            t=cache_dans_int(z,c,p)
            fichier.putpixel((i,j),(r,s,t))
```

Question 6

Ecrire une fonction `extrait_image` qui prend cette fois deux fichiers `sortie` et `originale` et un entier p. Pour chaque pixel `originale[i,j]`, la fonction récupère l'entier caché dans les p premiers bits de chaque composante du triplet, et recopie le résultat dans le fichier `sortie`.
Utilisez cette fonction pour découvrir l'image cachée dans le fichier `originale.png`.

```
def extrait_image(fichier,sortie,p):
    tf=fichier.size
    for i in range(tf[0]):
        for j in range(tf[1]):
            (a,b,c)=fichier.getpixel((i,j))
            r=extrait(a,p)
            s=extrait(b,p)
            t=extrait(c,p)
            sortie.putpixel((i,j),(r,s,t))
```

A titre de complément, pour cacher l'image secrète (un fichier au format png nommé bravo), j'ai donc écrit :

```
fichier = Image.open("originale.png")
secret = Image.open("bravo.png")
fichier_secret = Image.open("originale.png") # une copie pour l'image modfiée
cache_dans_image(homer,fichier_secret,3)
fichier_secret.save("fichier_secret.png")
```

et pour le décodage

```
sortie = Image.open("originale.png"). # une copie pour la sortie
extrait_image(fichier_secret,sortie,3)
sortie.show()
```