INFORMATIQUE

2 heures

N.B. : Le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

Les calculatrices sont interdites

Remarques générales

Important : Nous informons les candidats que chaque partie de ce sujet peut être traitée séparément. Vous pouvez utiliser toutes les fonctions des questions précédentes même si vous ne les avez pas implémentées.
Si la réponse attendue est spécifique à un langage de programmation, seul le langage **Python** est permis.
Les structures algorithmiques doivent être clairement identifiables par des indentations visibles ou par des barres droites entre le début et la fin de la structure comme l'exemple ci-dessous :

```
si (Condition)
    alors
        | Instructions
    sinon
        | Instructions
fin si
```

Dans les questions où l'on demandera d'écrire une fonction, on donnera systématiquement sa signature, c'est-à-dire le nom de la fonction avec les types des paramètres ainsi que le type du résultat retourné par cette fonction.

Par exemple, la fonction `foo` qui prend en entrée deux paramètres de type entier et retourne une liste, aura comme description :

Signature de la fonction `foo` :
`foo(int, int) -> list`

Sécurisation de l'entrée du personnel d'une entreprise

Partie I - Présentation

Une entreprise possède 5 sites de production. Le Directeur Général veut améliorer la sécurité. Il souhaite attribuer à chaque employé une carte personnelle et infalsifiable lui permettant l'accès à son site de production et peut-être à d'autres sites de l'entreprise. Chaque employé est affecté uniquement à un site que l'on appellera son site d'origine.

Chaque site de production ne pourra compter plus de 100 employés. La carte attribuée à l'employé comportera sa photo d'identité ainsi qu'une puce. Le code représentant le nom de la personne sera intégré dans la photo mais pas dans la puce.

Un tel code a été placé dans la photo de droite de la figure 1. Comme on peut le remarquer, il y a très peu de différences entre les deux photos, l'une sans code, l'autre avec un code intégré.

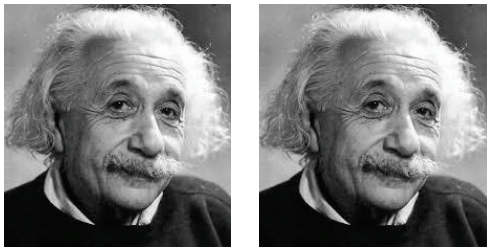

FIGURE 1 – Photos d'Einstein

Chaque employé sera référencé par un code composé d'un entier codé en machine sur 3 bits (indiquant le site de référence où il travaille) ainsi qu'une séquence de 7 caractères (pris parmi les 7 premiers caractères de l'alphabet en majuscules : `'A'... 'G'`).

Par exemple, pour un employé travaillant dans le site n° 3, le code pourrait être `3DABGEFC`.

Ce code n'affiche pas le nom de l'employé mais un automate permet, à partir des 7 caractères, de le retrouver.

Q1 Donner le nombre maximal d'employés par site ainsi que le nombre maximal de sites que pourrait gérer cette entreprise avec ce type de codage. La réponse peut être donnée par une expression numérique sans chercher à la calculer.
Indiquer si la technique de codage est suffisante pour gérer le personnel de cette entreprise.

Partie II - Distance de Levenshtein

La problématique est de savoir si les codes créés pour gérer les employés sont suffisamment disjoints les uns des autres, c'est-à-dire si la mesure de la différence entre deux codes est suffisamment importante. Ceci est possible grâce à la *distance de Levenshtein* qui peut être utilisée pour analyser tous les codes. L'entreprise veut tester tous les codes associés aux employés et changer les codes de tous ceux dont la distance de Levenshtein est inférieure à la valeur 3.

Dans cette partie, nous allons développer les fonctions permettant de répondre à cette problématique.

La distance de Levenshtein est une valeur donnant une mesure de la différence entre deux chaînes de caractères. Elle est égale au nombre minimal de caractères qu'il faut supprimer, insérer ou remplacer pour passer d'une chaîne A à une chaîne B.

On considère que chaque opération élémentaire mise en œuvre (supprimer, insérer ou remplacer) a un coût de 1. Chaque opération élémentaire ne porte que sur un seul caractère. La distance de Levenshtein, au sens mathématique du terme, est la somme de ces coûts.

Par exemple, la distance de Levenshtein entre les chaînes `Soveil` et `Soleil` est de coût égal à 1 car il a fallu réaliser une seule transformation, une substitution pour remplacer 'v' par 'l' dans la chaîne `Soveil`.

Nous allons maintenant déterminer quelle est la distance de Levenshtein entre les chaînes de caractères `Auttame` et `Automne`.

Q2 Écrire, dans le langage Python, les transformations (insertion, substitution, suppression) appliquées à la chaîne `Auttame` afin d'obtenir la chaîne `Automne`.
On déterminera systématiquement la valeur de la chaîne, notée `s`, avant et après chaque transformation ainsi que le nom de la transformation appliquée.
Calculer le coût de la distance de Levenshtein.

L'algorithme suivant décrit le calcul de la distance de Levenshtein entre deux chaînes de caractères. Cet algorithme retourne un entier (éventuellement nul) donnant la distance entre deux chaînes de caractères au sens de Levenshtein.

```
 1 levenshtein(chaine1, chaine2) =
 2 entree :
 3     chaine1[], chaine de caracteres d'indice 1 a n avec n egal a longueurChaine1 ;
 4     chaine2[], chaine de caracteres d'indice 1 a m avec m egal a longueurChaine2 ;
 5 sortie :
 6     entier : la valeur de la distance ;
 7 declarer :
 8     d[] : matrice de nombres entiers initialisee avec des 0, indice de 0 a n et de 0 a m ;
 9     i, j, coutSubstitution : entier ;
10 debut :
11     pour i de 0 a n
12         d[i,0] := i
13     pour j de 0 a m
14         d[0,j] := j
15     pour i de 1 a n
16         pour j de 1 a m
17             si (chaine1[i] = chaine2[j])
18                 alors
19                     coutSubstitution := 0
20                 sinon
21                     coutSubstitution := 1
22             fin si
23             d[i,j] := minimum(
24                           d[i-1,j] + 1,   # pour une suppression
25                           d[i,j-1] + 1,   # pour une insertion
26                           d[i-1,j-1] + coutSubstitution) # pour une substitution
27     retourner d[n,m]
28 fin
```

Q3 Déterminer la matrice `d` de l'algorithme de Levenshtein après l'instruction de la ligne 14 et après celle de la ligne 26, ainsi que la distance de Levenshtein lors de son exécution entre les chaînes `'GAZ'` et `'LA'`.

Q4 Déterminer la complexité de l'algorithme précédent.

Tous les codes ont été rassemblés dans une liste nommée `table_code`.

Q5 Écrire une fonction `code_bon_lvs` qui supprime tous les codes de la liste `table_code` dont la mesure de Levenshtein est inférieure ou égale à 3. Les codes à supprimer seront remplacés dans la liste `table_code` par le code nul, c'est-à-dire la chaîne de caractères `'0000000'`.

Signature de la fonction `code_bon_lvs` :
`code_bon_lvs(list) -> NoneType`

Q6 Écrire une fonction `pourcentage_lvs` qui détermine le pourcentage de codes nuls dans la liste `table_code`. Le résultat sera une chaîne de caractères constituée d'un nombre entier et du caractère %. On permet l'arrondi d'un calcul à l'entier inférieur ou à l'entier supérieur.

Signature de la fonction `pourcentage_lvs` :
`pourcentage_lvs(list) -> str`

Exemple :
```
>>> pourcentage_lvs(table_code)
'17%'
```

Partie IV - Codage des couleurs

La photo, une image couleur, est décrite informatiquement comme un tableau (une matrice) de pixels. Chaque pixel est représenté par une couleur au format RGB [1].

Par exemple, la couleur d'un pixel pourrait être au format RGB `(64,78,191)`. Si l'on écrit les trois composantes RGB en code binaire, on obtient le triplet (0100000**0**, 0100111**0**, 1011111**1**). Le bit de poids faible de chacune des composantes RGB est représenté en gras. On va se servir de ce triplet de valeurs définies en gras pour coder une information dans l'image.

La modification du bit de poids faible a très peu de conséquences sur la représentation de l'image.

Q7 Donner le code RGB en décimal et hexadécimal des éléments suivants :

— un pixel de couleur `bleu` ;

— un pixel de couleur `blanc`.

Un pixel est codé dans le format RGB en (10,10,10). Indiquez la couleur de ce pixel.

Q8 Donner pour le codage RGB le nombre de couleurs possibles. La réponse peut être donnée par une expression numérique sans chercher à la calculer.

1. Le système RGB `(Red, Green, Blue)` ou en français `(Rouge, Vert, Bleu)`, permet de coder les couleurs en informatique. Un écran informatique est composé de pixels représentant une couleur au format RGB.

La composante R est codée sur 8 bits de 0 à 255 en décimal et de 00 à FF en hexadécimal. Il en va de même pour les autres composantes. Le codage des couleurs va du plus foncé au plus clair.

Par exemple, la couleur d'un pixel orange pourrait avoir comme valeur (255,100,100). Un pixel sera de couleur `gris` si les composantes R, G et B sont identiques.

Partie V - Codage de l'information

Cette partie sera consacrée à l'implémentation de fonctions qui serviront notamment dans la partie VI, réservée au décodage de l'information se trouvant dans la photo de l'employé(e).

Les informations constituant le code sont définies dans une trame composée de blocs non consécutifs. Chaque bloc sera associé à un pixel. Un premier bloc est constitué d'un pixel pour représenter le numéro du site et est suivi d'une séquence de 7 blocs représentant chacun un caractère. Le premier bloc sera considéré comme le bloc de référence.

Pour un pixel codant le numéro du site de l'employé, les bits de poids faible indiquent directement le numéro au format binaire.

Ainsi, pour l'exemple précédent, le codage correspondant au site n° 3 sera (01000000, 01001111, 10111111) puisque le triplet `011` correspond à la valeur 3 en code décimal. Par conséquent, on met une information dans l'image en modifiant uniquement les bits de poids faible, ce qui a peu de conséquences sur la représentation de cette image.

Pour un pixel codant une des lettres de l'identifiant de l'employé, on valide la convention suivante : le caractère `'A'` sera associé à `001`, le caractère `'B'` à `010`, jusqu'au caractère `'G'` qui aura comme valeur `111`.

Par exemple, un pixel codant le caractère `'B'` (`010`) pourrait avoir pour format RGB (... 0, ... 1, ... 0).

On ne traitera pas le caractère associé à `000` qui aura pour lettre `'W'`, réservée pour la gestion du personnel de service (gardiens, personnel de nettoyage, etc.) ayant l'autorisation d'entrer dans les sites de l'entreprise.

Chaque trame sera constituée de 8 blocs, c'est-à-dire 8 pixels (voir la figure 2).

— Un pixel de référence est défini par ses coordonnées que l'on nomme `posi` dans la suite.

— Le paramètre ou la variable `posi` est un couple de valeurs (colonne, ligne) qui est obtenu à l'aide d'une clé se trouvant dans la puce. Les valeurs des coordonnées `posi` sont supposées données.

— Les caractères définis dans le code sont, dans l'image, décalés d'une valeur `delta` qui est déterminée à partir du pixel de référence en calculant la somme des valeurs de ses composantes R, G et B. Autrement dit, si le pixel de référence est à la position (x, y) et si la valeur $x+\delta$ n'est pas supérieure à la largeur (`imx`) de l'image, le premier bloc sera à la position $(x + \delta, y)$, sinon le bloc se trouvera sur la ligne suivante, et ceci jusqu'au $7^{\text{ème}}$ caractère défini dans le code de l'employé.

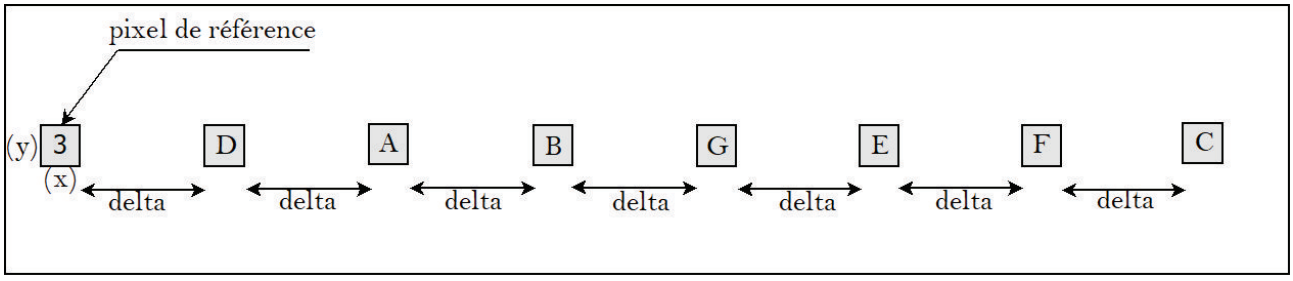

FIGURE 2 – Trame

Q9 Étant donné un entier `n`, justifier que le dernier chiffre du code binaire de `n` est égal à `n%2`. À quoi correspond `n//2` dans le code binaire de `n` ?
En déduire une fonction **récursive** `dec_to_bin()` qui, à partir d'un nombre entier non nul, retourne le code binaire de ce nombre. Cette fonction convertit un nombre entier en une chaîne de chiffres binaires.

Signature de la fonction `dec_to_bin` :

```
dec_to_bin(int) -> str
```

Exemples :
```
>>> dec_to_bin(45)
'101101'
>>> dec_to_bin(1)
'1'
```

Q10 À l'aide de la fonction `dec_to_bin()`, écrire une fonction `dec_to_bin2()` qui, à partir d'un nombre entier, retourne le code binaire de ce nombre, écrit sous la forme d'une séquence d'au moins 3 digits de type `string`.

Signature de la fonction `dec_to_bin2` :
```
dec_to_bin2(int) -> str
```

Exemples :
```
>>> dec_to_bin2(45)
'101101'
>>> dec_to_bin2(1)
'001'
```

Q11 Écrire la fonction `num()` qui calcule, à partir d'une chaîne en chiffres binaires, par exemple `'010'`, sa valeur en décimal.
Si la chaîne est vide, la fonction retournera comme valeur `-1`.

Signature de la fonction `num` :
```
num(str) -> int
```

Exemples :
```
>>> num('101')
5
>>> num('')
-1
```

Partie VI - Décodage de l'information

Cette partie sera consacrée à l'implémentation de fonctions pour le décodage de l'information.
Remarque : vous pouvez utiliser les fonctions des questions précédentes même si elles n'ont pas été traitées.

On donne le code suivant :

```
from PIL import Image
import os
os.chdir("C:\photos")         # Positionnement sur le bon répertoire.
im = Image.open("a123.bmp")   # Lecture de l'image de l'employé dont la
                              # photo est a123.
imx, imy = im.size            # La taille de l'image (largeur, hauteur).
                              # Résultat : (277, 250).
posi = (30,20)                # Position du pixel (x,y).
px = im.getpixel (posi)       # Récupère la valeur au format RGB du pixel
                              # situé aux coordonnées (30,20).
                              # Résultat : (2, 57, 139).
                              # Les indices lignes et colonnes débutent à 0.
```

Lorsque l'employé passe sa carte sur le lecteur pour ouvrir une porte, les données de la puce et le code se trouvant dans la photo sont lus par le lecteur qui génère 2 valeurs envoyées au processus d'analyse (dont vous allez implémenter certaines fonctions). Ces 2 valeurs sont le pixel de référence (avec sa valeur `posi`) et le code associé à l'employé (un entier suivi de 7 caractères). Le pixel de référence est différent pour chacun des employés.

Les caractères du code dans l'image ne sont pas placés d'une manière consécutive mais séparés les uns des autres par une même valeur (`delta`) calculée à l'aide des valeurs RGB du pixel de référence (voir la figure 2) en réalisant la somme des 3 composantes R, G et B.

Q12 Écrire la fonction `lettre()` qui, à partir d'une chaîne de 3 bits, détermine le caractère. La codification des 7 caractères a été réalisée sur seulement 3 bits. On rappelle que le caractère `'A'` est codé `'001'`, le caractère `'B'` est codé `'010'` et ainsi de suite jusqu'au caractère `'G'` qui est codé `'111'`, ceci en suivant l'ordre des 7 premières lettres de l'alphabet.

Signature de la fonction `lettre` :
```
lettre(str) -> str
```

Exemples :
```
>>> lettre('001')
'A'
>>> lettre('101')
'E'
```

Q13 Écrire la fonction `bpf` qui récupère les valeurs des bits de poids faible d'un pixel.

Cette fonction a 2 paramètres : l'image et la position du pixel. Elle retourne une chaîne de caractères composée de 3 bits.

Signature de la fonction `bpf` :
```
bpf(BmpImageFile, (int,int)) -> str
```

Exemple :
```
bpf(im, posi)
>>> '011'
```

Q14 Soit la fonction `valeur_delta()` dont le code est :

```
def valeur_delta(im, posi):
    px = im.getpixel(posi)
    inter = (px[0] + px[1] + px[2]) % (128 - 41) + 40
    if inter % 2 == 1:
        return inter + 1
    else:
        return inter
```

Cette fonction a 2 paramètres : l'image et les coordonnées du point de référence (le pixel du point de référence).

Signature de la fonction `valeur_delta` :
```
valeur_delta(BmpImageFile, (int,int)) -> int
```

Indiquer ce que fait cette fonction en précisant les valeurs retournées. Le résultat sera décrit sous la forme d'un intervalle ou d'une union d'intervalles de nombres entiers.

Q15 Écrire la fonction `position_bloc()` qui détermine les coordonnées d'un bloc de la trame se trouvant à un indice compris entre 0 et 7, l'indice 0 représentant le pixel de référence. Le premier bloc des caractères du code se trouve à l'indice 1 et le dernier à l'indice 7. Rappelons que l'indice 0 (le pixel de référence) est le bloc constitué du numéro du site où l'employé travaille.

Cette fonction a 4 paramètres : la position initiale du pixel de référence, la valeur de l'intervalle, la largeur de l'image et l'indice du bloc. Elle retourne les coordonnées du pixel du bloc recherché.

Signature de la fonction `position_bloc` :
```
position_bloc((int,int),int,int,int) -> (int,int)
```

Exemple :
```
>>> position_bloc(posi,70,imx,1)
(100, 20)
```

Q16 Écrire la fonction `num_site()` qui retourne comme résultat le numéro du site où la personne travaille.
Cette fonction a 2 paramètres : l'image et la position du pixel de référence.

Signature de la fonction `num_site` :
```
num_site(BmpImageFile,(int,int)) -> int
```

Exemple :
```
>>> num_site(im, posi)
3
```

Q17 Écrire la fonction `lecture_lettres()` qui retourne la séquence des lettres codées dans l'image, c'est-à-dire les 7 lettres constituant une partie du code de l'employé.

Cette fonction a 2 paramètres : l'image et la position du pixel de référence. Elle retourne une chaîne de caractères.

Signature de la fonction `lecture_lettres` :
```
lecture_lettres(BmpImageFile, (int,int)) -> str
```

Exemple :
```
>>> lecture_lettres(im, posi)
'DABGEFC'
```

Q18 Indiquer ce que fait la fonction suivante :

```
def lecture_code(im, posi):
    num = num_site(im, posi)
    msg = lecture_lettres(im, posi)
    return str(num) + msg
```