Nom : .....

Prénom : .....

NOM Prénom : ...

Classe : CPGE 1ère année

Lycée Loges

Exercice I :	....	**/14**
Exercice II :	....	**/12**
Exercice III :	....	**/14**

Note finale : .... / 40

⚠ Dans l'ensemble du sujet, on prendra soin d'écrire les fonctions avec le type des paramètres (*e.g.* incorrect : `def somme(tab):` / correct : `def somme(tab: list) -> int:`).
Nota bene : Le barème n'est qu'indicatif. Veillez à proposer des explications claires et concises

Exercice I (/14)

Le but de ces deux parties est d'étudier un système d'identification par radio fréquence (RFID), similaire au système d'un pass Navigo. On étudiera ici une partie du système de correction des données transmises.

Le signal transmis par une liaison RFID 13,56MHz peut être perturbé par toutes sortes de facteurs pouvant provoquer des erreurs dans les données : entre autres signaux électromagnétiques, masses métalliques, imperfections du matériel électronique...

En pratique, il est donc indispensable de pouvoir détecter ces erreurs et, dans la mesure du possible, les corriger sans que cela ne nécessite une nouvelle transmission ; l'objet de cette partie est de mettre en place quelques algorithmes dans ce but.

Partie 1 : Bit de parité

Une technique simple et très répandue pour s'assurer qu'une donnée binaire sera lue correctement par son récepteur est de lui adjoindre un bit de parité, égal par définition à :

- ◇ 0 si la donnée contient un nombre pair de 1 (et, donc, si ses bits sont de somme paire),
- ◇ 1 si la donnée contient un nombre impair de 1 (et, donc, si ses bits sont de somme impaire).

Après réception de la donnée, le récepteur recalcule le bit de parité et le compare à celui que l'émetteur lui a addressé. Si la donnée n'a pas été altérée lors de la transmission, alors les deux bits de parité sont forcément identique.

☐**Q1 -** Donner les bits de parité associés aux représentations binaires des entiers 5, 16 et 37.

☐**Q2 -** Écrire une fonction `parite(bits: list) -> int`, prenant pour argument une liste `bits` constituée d'entiers valant 0 ou 1 et retournant l'entier 0 ou 1 correspondant à son bit de parité.

Les techniques de vérification les plus simples consistent à découper la donnée en blocs et à joindre un bit de parité à chaque bloc. Par exemple, certains protocoles transmettent sept bits de données pour un bit de parité.

☐**Q3 -** Donner un exemple d'erreur n'étant pas détectable par cette technique. Si une erreur a été détectée, est-il possible de la corriger sans retransmettre la donnée ?

Partie 2 : Code de Hamming

Le code de Hamming est un exemple d'utilisation des bits de parité pour détecter et corriger des erreurs. Nous nous intéressons ici au code $(7, 4)$, ainsi appelé car il consiste à joindre trois bits de parité à quatre bits de données, ce qui donne un message d'une longueur totale de sept bits.

Ces trois bits de parité sont définis ainsi : si la donnée s'écrit (d_1, d_2, d_3, d_4) avec $d_1 = 0$ ou 1, alors :

- ◇ p_1 est le bit de parité du triplet (d_1, d_2, d_4),
- ◇ p_2 est le bit de parité du triplet (d_1, d_3, d_4),
- ◇ p_3 est le bit de parité du triplet (d_2, d_3, d_4).

Le message encodé, que l'on transmet, s'écrit alors comme suit : $(p_1, p_2, d_1, p_3, d_2, d_3, d_4)$.

☐**Q4 -** Écrire une fonction `encode_hamming(donnee: list) -> list`, prenant pour argument une liste donnée de quatre bits (représentés par des entiers valant 0 ou 1) et retournant une liste de bits contenant le message encodé.

On pourra appeler la fonction `parite(bits)` précédemment définie.

Le contrôle après réception d'un message ainsi encodé est relativement simple. On pourrait naturellement recalculer les trois bits de parité de la donnée et les comparer aux valeurs transmises, mais la technique proposée par Hamming est de calculer les trois bits de contrôle suivants, notés (c_1, c_2, c_3), à partir du message complet (données et bits supplémentaires), noté $(m_1, \ldots, m_7)$:

- ◇ c_1 est le bit de parité de l'ensemble (m_4, m_5, m_6, m_7),
- ◇ c_2 est le bit de parité de l'ensemble (m_2, m_3, m_6, m_7),
- ◇ c_3 est le bit de parité de l'ensemble (m_1, m_3, m_5, m_7).

On montre que si le message a bien été encodé selon les règles précédentes et n'a pas été altéré, alors les trois bits de contrôle doivent être à 0. Si ce n'est pas le cas, alors il y a eu une erreur ; l'intérêt de la technique de Hamming est que dans le cas particulier où l'erreur est unique, le mot de contrôle donne la représentation binaire de la position de cette erreur en numérotant à partir de 1.

Par exemple, si $(c_1, c_2, c_3) = (0, 1, 1)$, alors l'erreur porte sur le troisième bit du message. Il suffit ainsi d'inverser ce bit (le mettre à 1 s'il est à 0, et inversement) pour corriger l'erreur.

La donnée décodée est alors constituée des quatre bits (d_1, d_2, d_3, d_4) qui se trouvent respectivement en position 3, 5, 6 et 7 (toujours en numérotant à partir de 1) conformément à la description de l'encodage donnée ci-dessus.

☐**Q5 -** Écrire une fonction `decode_hamming(message: list) -> list`, prenant pour argument une liste de sept bits et retournant une liste de quatre bits contenant la donnée décodée. En cas d'erreur, on affichera à l'écran un avertissement indiquant la position du bit affecté et on effectuera la correction. On supposera dans cette question que s'il y a une erreur, alors elle est unique.

☐**Q6 -** Déterminer le codage de Hamming de la donnée `1011`, puis la donnée décodée par l'algorithme dans l'hypothèse où les deux premiers bits du message codé ont été incorrectement transmis. Quel a été l'effet de la « correction » sur la donnée dans ce cas ?

☐**Q7 -** Sans coder, proposer un moyen simple de différencier une double erreur d'une erreur unique au moyen d'un bit de parité supplémentaire et expliquer comment cela permet d'éviter le problème mis en évidence à la question précédente. On s'appuiera sur les techniques introduites dans cette partie. On ne demande pas d'essayer de corriger la double erreur.

Exercice II (/12)

☐**Q1 -** Proposer en `python` une fonction `maxi` prenant en argument une liste d'entiers naturels `L` et renvoyant le maximum des entiers de cette liste. *On n'utilisera pas de fonction spécifique de python déterminant ce maximum.*

☐**Q2 -** Écrire une fonction `ind` prenant en argument une liste d'entiers naturels `L` et renvoyant la liste des indices $[i_1, i_2, \dots i_r]$ avec $i_1 < i_2 < \dots i_r$ telle que pour tout $k \in [\![1, r]\!]$, $L[i_k]$ soit non nul.

Par exemple, si `L=[0,1,3,0,7]`, alors `ind(L)` renvoie `[1,2,4]`.

☐**Q3 -** Écrire une fonction `nb_oc` prenant en argument une liste d'entiers naturels `L` et renvoyant la liste `T` de longueur `M = maxi(L)+1` où, pour tout $i \in [\![0; M]\!]$, `T[i]` est le nombre d'occurences dans la liste `L` de l'entier i.

Par exemple, si `L=[3,1,4,1,5]`, alors `T=[0,2,0,1,1,1]` (rq : on pourra utiliser la fonction `maxi`).

☐**Q4 -** **a)** Soit `L` une liste d'entiers naturels. Déterminer le nombre de fois, noté n, où la liste `L` est parcourue lors de l'exécution de `nb_oc(L)`.

b) On veut que n soit indépendant de `M`. Si c'est le cas, expliquer en quoi la condition est respectée. Si ce n'est pas le cas, modifier la fonction `nb_oc` afin de respecter cette condition.

☐**Q5 -** Soit A une suite d'entiers. On définit alors la suite de Robinson $(L_n)_{n\in\mathbb{N}}$ associée à la liste A par récurrence comme suit :

- ◇ $L_0 = A$
- ◇ si L_n est construite, alors
  - on détermine $T_n = nb\_oc(L_n)$
  - on détermine $I_n = ind(T_n)$
  - si $I_n = [i_1, \dots i_r]$, alors $L_{n+1} = [T[i_r], i_r, \dots T[i_1], i_1]$

Par exemple, si A = [4,4,1,2] on a :
- L_0 = [4,4,1,2]
- L_1 = [2,4,1,2,1,1] (il y a deux '4', un '2' et un '1' dans la liste L_0)
- L_2 = [1,4,2,2,3,1] (il y a un '4', deux '2' et trois '1' dans la liste L_1)

a) On donne A = [2,0,4,1,3,3,2,3,1,1]. Déterminer L_3 et L_{2018}.

b) On donne B = [2,4,1,1,1,2]. Si l'on suppose que $L_1 = B$, donner toutes les solutions possibles pour L_0

c) On donne C = [2,4,1,0]. Si l'on suppose que $L_1 = C$, donner toutes les solutions possibles pour L_0

d) Proposer alors une fonction `rob(A: list, n: int) -> list`, qui prend en argument une liste `A` et un entier naturel `n` et qui renvoie l'élément L_n de la suite de Robinson associée à `A`.

Exercice III (/14)

Si rien n'est précisé, la variable `mat` sera la matrice représentant l'image du fichier `'./photo.bmp'`.

☐**Q1 -** Proposer une fonction `sum(tab: list) -> int`, qui, pour un tableau `tab` ne contenant que des entiers, renvoie la somme des entiers de `tab`.

☐**Q2 -** Proposer une fonction `operationF(mat: list, F: 'function' = lambda x: x) -> list`, qui renvoie une nouvelle matrice pour laquelle la fonction `F` a été appliquée à chaque coefficient `mat[i][j]` de la matrice initiale.

☐**Q3 -** Proposer une fonction `histogramme(mat: list) -> list`, qui prend en argument la matrice `mat` d'une image et renvoie un tableau de taille 256 contenant à l'indice `i` le nombre de pixels ayant le niveau de gris `i`.
e.g. `histogramme([[0,254],[0,255]])` renverra le tableau `[2,0,... 0,1,1]`

☐**Q4 -** Écrire une fonction `indiceMinMax(hist: list) -> tuple`, qui prend en argument un tableau de 256 cases et renvoie le tuple avec le plus petit indice de valeur non nulle ainsi que le plus grand.
e.g. `indiceMinMax([0,0,1,5,0,0,4,0]` renverra `(2,6)`.

☐**Q5 -** Afin d'augmenter le contraste, on souhaite redéfinir la valeur des gris de sorte que la valeur minimale de la matrice corresponde au noir, tandis que la valeur maximale au blanc.
Préciser l'expression de la fonction linéaire f définie par $\begin{cases} f(vmin) = 0 \\ f(vmax) = 255 \end{cases}$ puis en déduire une fonction `extension(x: int) -> int`, qui renvoie l'image entière de x par f, sachant que `mat` possède au moins deux valeurs non-nulles (*i.e.* `vmin < vmax`), que x est l'une des valeurs non-nulle de `mat` (*i.e.* `vmin <= x <= vmax`).
En outre les variables `vmin` et `vmax` sont déjà définies et utilisables telles quelles dans la fonction `extension`. *i.e.* `def extension(x:int) -> int: return vmax-vmin` ne lèvera pas d'erreur.

☐**Q6 -** Choisir et recopier l'appel de la fonction `operationF` qui permet de renvoyer la matrice codant pour l'image contrastée selon la méthode vue.

◇ `operationF("./photo.bmp", extension)` ◇ `operationF(mat, extension(x))`
◇ `operationF("./photo.bmp", extension(x))` ◇ `operationF(mat, extension)`

☐**Q7 -** Afin d'obtenir un meilleur rendu, on souhaite coder la méthode d'égalisation d'histogramme. L'idée consiste à appliquer à chacun des pixels une transformation T définie par $T(x_k) = \frac{L-1}{N}\sum_{j=0}^{k} n_j$, où

- ◇ L est le nombre de niveaux de gris que l'on souhaite utiliser (ici 256).
- ◇ x_k est un pixel de la classe k (*i.e.* si $mat[i][j] = k$ alors $mat[i][j]$ est un pixel de la classe k)
- ◇ n_j est le nombre de pixels dans la classe j, N leur somme.

Définir la fonction `T(x: int, hist: list) -> int`, définie ci-dessus.

☐**Q8 -** Sans justifier, préciser la complexité temporelle asymptotiques (en grand $\mathcal{O}$) de la fonction `T`. Expliquer la ou les améliorations qui auraient amélioré cette complexité temporelle et de combien.

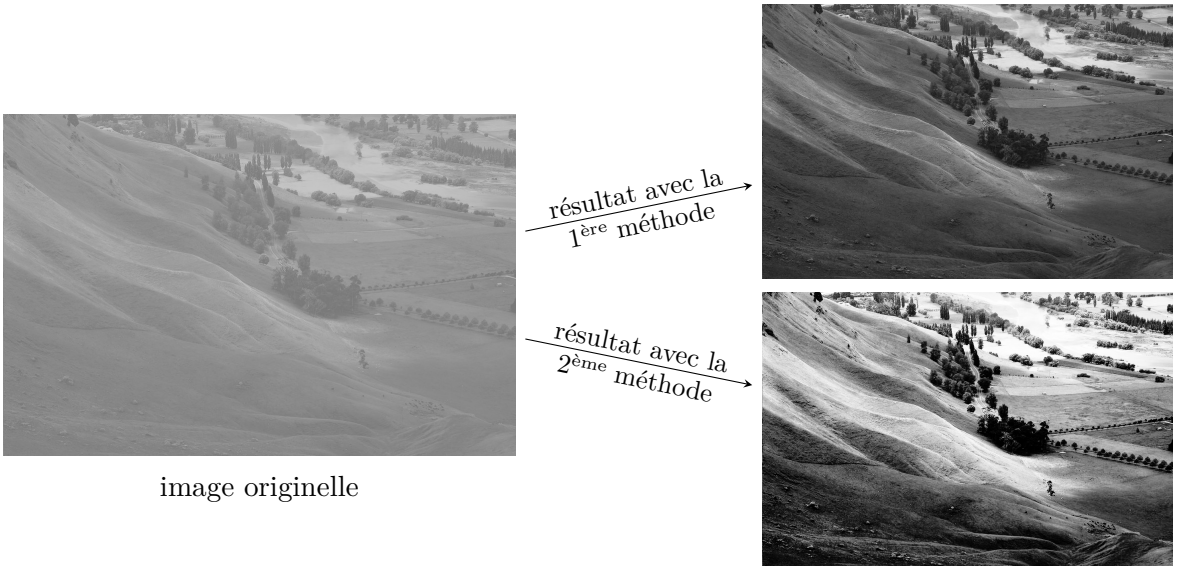

image originelle

image originelle tirée de [https://fr.wikipedia.org/wiki/Égalisation_d'histogramme]