Amérique du Nord – 2025 – sujet2

Exercice 1 (6 points)

Cet exercice porte sur les tableaux, les dictionnaires, les arbres binaires, la programmation en Python et la récursivité.

Lors de la transmission de données, des erreurs peuvent se glisser.

On se propose d'étudier des techniques permettant de minimiser les conséquences de telles erreurs.

PARTIE A

Pour encoder un texte en binaire, on traduit chaque caractère en un octet, par exemple en utilisant le code ASCII. La table ASCII permet de traduire les caractères classiques en entiers compris entre 0 et 127, qui peuvent ensuite être écrits en binaire sur un octet, c'est-à-dire une suite de 8 bits valant chacun `0` ou `1`.

Dans la table ASCII, le code associé au caractère `a` est 97.

1. Donner l'écriture binaire de `a` sur 8 bits.

Pour pouvoir corriger les erreurs durant les transmissions, on peut ajouter de la redondance dans les informations transmises, c'est-à-dire ajouter des moyens permettant de retrouver le message initial même si un ou plusieurs bits ont été modifiés. Une manière de le faire consiste à envoyer trois fois le même message.

Ainsi, même si l'une des copies se retrouve modifiée, on peut retrouver le message tant que la majorité des copies a été transmise sans erreur.

La fonction `replique` suivante implémente cette stratégie. Cette fonction prend en paramètre une liste `tab` composée de `0` et de `1`.

```
1   def replique(tab):
2       n = len(tab)
3       return [tab[i // 3] for i in range(3 * n)]
```

2. Donner le résultat de l'appel `replique([0,0,1,0,1])`.

3. Recopier et compléter les lignes 14 et 16 du code de la fonction `nb_occurrences`, donné ci-après, qui prend en paramètres une liste `tab` et un entier `i` et qui renvoie un dictionnaire qui associe, à chaque élément son nombre d'occurrences.

```
 1 def nb_occurrences(tab, i):
 2   '''
 3   Renvoie un dictionnaire qui associe, à chaque élément
 4   apparaissant dans tab entre la position 3i
 5   incluse et la position 3(i + 1) exclue,
 6   son nombre d'occurrences.
 7   >>> nb_occurrences([0, 0, 1, 1, 0, 1, 0, 1, 1], 1)
 8   {1: 2, 0: 1}
 9   '''
10   nb_occ = {}
11   for j in range(3 * i, 3 * (i + 1)):
12     x = tab[j]
13     if x in nb_occ:
14       ...
15     else:
16       ...
17   return nb_occ
```

Recopier et compléter à partir de la ligne 10 (le nombre de lignes et l'indentation sont suggérés mais ne sont pas obligatoires) du code de la fonction `majorite`, donné ci-après. Cette fonction prend en paramètre un dictionnaire `dict` et renvoie une clé du dictionnaire pour laquelle la valeur associée est la plus grande.

```
 1 def majorite(dict):
 2   '''
 3   Renvoie une clé du dictionnaire dict pour laquelle la
 4   valeur associée est la plus grande.
 5   Précondition : dict est un dictionnaire dont toutes
 6   les valeurs sont positives.
 7   '''
 8   cle_max = None
 9   valeur_max = -1
10   for cle in dict.keys():
11     ...
12       ...
13       ...
14   return cle_max
```

Pour transmettre 4 bits d'information, il faut envoyer 12 bits par cette méthode.

PARTIE B

On s'intéresse à présent à une autre solution, reposant sur l'ajout de bits de parité.

Pour transmettre 4 bits $b_3b_2b_1b_0$, on les place dans une matrice comme suit, et on complète les lignes et les colonnes par un bit de parité (`0` ou `1`) pour que chaque ligne et chaque colonne possède un nombre pair de bits valant `1`.

b_3	b_2	bit de parité ligne 1
b_1	b_0	bit de parité ligne 2
bit de parité colonne 1	bit de parité colonne 2	bit de parité total

Lorsqu'il n'y a qu'une seule erreur, elle se situe à l'intersection de la ligne et de la colonne qui possèdent un nombre impair de bits valant `1`.

5. On reçoit le tableau suivant.

1	1	1
1	1	0
0	1	1

Sachant qu'une unique erreur de transmission s'est produite, recopier le tableau et entourer le bit qui a subi cette erreur (transformation d'un `0` en `1` ou d'un `1` en `0`).

On représente une telle matrice en Python par la liste de ses lignes où chaque ligne est elle-même représentée par la liste de ses bits (`0` ou `1`).

6. Écrire le code d'une fonction `erreur_colonne` qui prend en paramètre une matrice `mat` dans laquelle exactement une erreur a eu lieu lors de la transmission et qui renvoie l'indice de la colonne ayant une parité erronée.

Grâce à cette méthode, on peut transmettre 4 bits d'information en utilisant 9 bits, tout en détectant et corrigeant une unique erreur.

PARTIE C

Richard Hamming a mis au point une méthode qui permet d'arriver au même résultat avec seulement 7 bits transmis.

Le tableau suivant établit une correspondance entre chaque mot de 4 bits et une unique suite de 7 bits.

Code de Hamming (4, 7)				
Mot	Code associé		Mot	Code associé
0000	0000000		1000	1110000
0001	1101001		1001	0011001
0010	0101010		1010	1011010
0011	1000011		1011	0110011
0100	1001100		1100	0111100
0101	0100101		1101	1010101
0110	1100110		1110	0010110
0111	0001111		1111	1111111

On admet que ce tableau est construit de sorte que, étant donné une suite de 7 bits,

- soit elle est présente dans le tableau ;
- soit, dans le cas contraire, il existe un unique code du tableau qui ne diffère avec elle que d'un bit.

Un mot de 4 bits ayant été encodé selon cette correspondance est transmis.

7. Une unique erreur se glisse dans cette transmission, de sorte que le code reçu est `1010000`. Déterminer, en justifiant, le mot de 4 bits initial.

On souhaite écrire une fonction de correction des codes reçus.

8. Recopier et compléter les lignes 17, 20 et 25 du code de la fonction `corriger_erreur` ci-après, qui prend en paramètre la liste des entiers 0 ou 1 correspondant au code reçu et qui renvoie cette liste si c'est un code associé, ou le code associé qui ne diffère que d'un bit de celle-ci sinon.

   Exemples :

```
>>> corriger_erreur([1,1,0,1,0,0,1])
[1, 1, 0, 1, 0, 0, 1]
>>> corriger_erreur([1,0,1,0,0,0,0])
[1, 1, 1, 0, 0, 0, 0]
```

```
 1 # liste composée de tous les codes
 2 # associés de Hamming(4, 7).
 3 hamming_4_7 = [
 4     [0,0,0,0,0,0,0], [1,1,0,1,0,0,1],
 5     [0,1,0,1,0,1,0], [1,0,0,0,0,1,1],
 6     [1,0,0,1,1,0,0], [0,1,0,0,1,0,1],
 7     [1,1,0,0,1,1,0], [0,0,0,1,1,1,1],
 8     [1,1,1,0,0,0,0], [0,0,1,1,0,0,1],
 9     [1,0,1,1,0,1,0], [0,1,1,0,0,1,1],
10     [0,1,1,1,1,0,0], [1,0,1,0,1,0,1],
11     [0,0,1,0,1,1,0], [1,1,1,1,1,1,1]]
12 def corriger_erreur(code_recu):
13   if code_recu in hamming_4_7:
14     return code_recu
15   else:
16     # Copie du code reçu créée par compréhension
17     code = ...
18     for indice in range(7):
19       # Inversion du bit d'indice courant
20       code[indice] = (code[indice] + 1) ...
21       if code in hamming_4_7:
22         return code
23       else:
24         # Réinit. du bit d'indice courant
25         code[indice] = ...
```

On se propose de construire un décodeur pour le code de Hamming (4, 7) à l'aide d'un arbre binaire. Il s'agit d'un arbre binaire de hauteur 7 dont chaque feuille est étiquetée par le mot de 4 bits susceptible d'avoir donné le code correspondant au chemin menant à cette feuille.

Pour décoder un code reçu, on descend dans l'arbre en lisant ce code de la gauche vers la droite. Si on rencontre un bit valant `0`, on continue dans le sous-arbre gauche. Si on rencontre un bit valant `1`, on continue dans le sous-arbre droit.

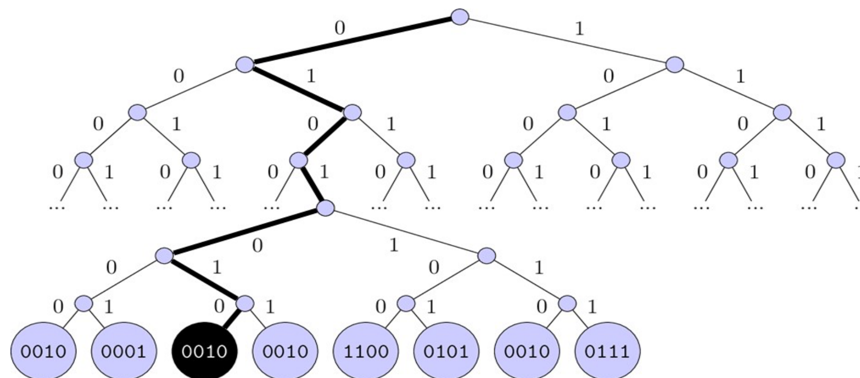

Figure 1. Représentation partielle de l'arbre décodeur du code de Hamming (4, 7).

Par exemple, le chemin indiqué en gras sur la Figure 1 indique comment on peut retrouver le mot `0010` après avoir reçu le code `0101010`.

9. Indiquer combien l'arbre décodeur complet du code de Hamming (4,7) comporte de feuilles.

Un arbre binaire non vide est représenté en Python par une classe `Noeud` qui possède 3 attributs :

- `gauche` correspond à son sous-arbre gauche s'il s'agit d'un nœud interne, et vaut `None` s'il s'agit d'une feuille ;
- `droit` correspond à son sous-arbre droit s'il s'agit d'un nœud interne, et vaut `None` s'il s'agit d'une feuille ;
- `etiquette` correspond à la chaîne de caractères désignant le mot décodé s'il s'agit d'une feuille, et vaut `None` s'il s'agit d'un nœud interne.

10. Recopier et compléter les lignes 12 à 14 du code de la fonction récursive `decode`, donné ci-après. Cette fonction prend en paramètres un arbre décodeur `arbre`, une liste `code` et un indice `i` et renvoie le mot étiquetant la feuille atteinte.

```
 1 def decode(arbre, code, i):
 2   '''
 3   Descend dans l'arbre binaire arbre en lisant le
 4   tableau code à partir de l'indice i et renvoie
 5   le mot étiquetant la feuille atteinte.
 6   Précondition : arbre est un arbre binaire
 7   de hauteur len(code) - i
 8   '''
 9   if i == len(code):
10     return arbre.etiquette
11   if code[i] == 0:
12     return decode(...)
13   if code[i] == 1:
14     return decode(...)
```