Lycée Les 3 Sources **Bourg-Lès-Valence**
NSI T^le Année **2025-26**

Devoir surveillé n°5 – Correction

Nom et prénom :

Exercice 1 : (12pt) *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.
Solution : `01100001`

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])`.
Solution : On va obtenir `[0,0,0, 0,0,0, 1,1,1, 0,0,0, 1,1,1]`.
3) Compléter les lignes 14 et 16 du code de la fonction `nb_occurrences`), donné sur la page suivante, 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.

```
if x in nb_occ:
    nb_occ[x] += 1
else:
    nb_occ[x] = 1
```

4) 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 `dico` et renvoie une clé du dictionnaire pour laquelle la valeur associée est la plus grande.

```
for cle in dico.keys():
    if dico[cle] > valeur_max:
        valeur_max = dico[cle]
        cle_max = cle
```

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

```python
def nb_occurrences(tab, i):
    '''
    Renvoie un dictionnaire qui associe, à chaque élément
    apparaissant dans tab entre la position 3i
    incluse et la position 3(i + 1) exclue,
    son nombre d'occurrences.
    >>> nb_occurrences([0, 0, 1, 1, 0, 1, 0, 1, 1], 1)
    {1: 2, 0: 1}
    '''
    nb_occ = {}
    for j in range(3 * i, 3 * (i + 1)):
        x = tab[j]
        if x in nb_occ:
            ...
        else:
            ...
    return nb_occ
```

```python
def majorite(dico):
    '''
    Renvoie une clé du dictionnaire dico pour laquelle la
    valeur associée est la plus grande.
    Précondition : dico est un dictionnaire dont toutes
    les valeurs sont positives.
    '''
    cle_max = None
    valeur_max = -1
    for cle in dico.keys():
        ...
            ...
            ...
    return cle_max
```

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

1	1	1
1	1	0
0	1	1

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.

```
def erreur_colonne(mat):
    for j in range(3):
        nb_1 = 0
        for i in range(3):
            if mat[i][j] == 1:
                nb_1 += 1
        if nb_1 % 2 == 1:
            return j
```

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.

Solution : Il y a une différence d'un bit avec `1110000`. Le mot de départ est donc `1000`.

On souhaite écrire une fonction de correction des codes reçus.

8) Compléter les lignes 18, 21 et 26 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],
```

```
    [1,0,0,1,1,0,0], [0,1,0,0,1,0,1],
    [1,1,0,0,1,1,0], [0,0,0,1,1,1,1],
    [1,1,1,0,0,0,0], [0,0,1,1,0,0,1],
    [1,0,1,1,0,1,0], [0,1,1,0,0,1,1],
    [0,1,1,1,1,0,0], [1,0,1,0,1,0,1],
    [0,0,1,0,1,1,0], [1,1,1,1,1,1,1]]

def corriger_erreur(code_recu):
    if code_recu in hamming_4_7:
        return code_recu
    else:
        # Copie du code reçu (par exemple, créée par compréhension)
        code = ...
        for indice in range(7):
            # Inversion du bit d'indice courant
            code[indice] = ...
            if code in hamming_4_7:
                return code
            else:
                # Réinit. du bit d'indice courant
                code[indice] = ...
```

```
code = list(code_recu)  # ou code = [c for c in code_recu]
code[indice] = 1 - code[indice]  # ou (code[indice]+1) % 2
code[indice] = 1 - code[indice]  # ou code_recu[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.

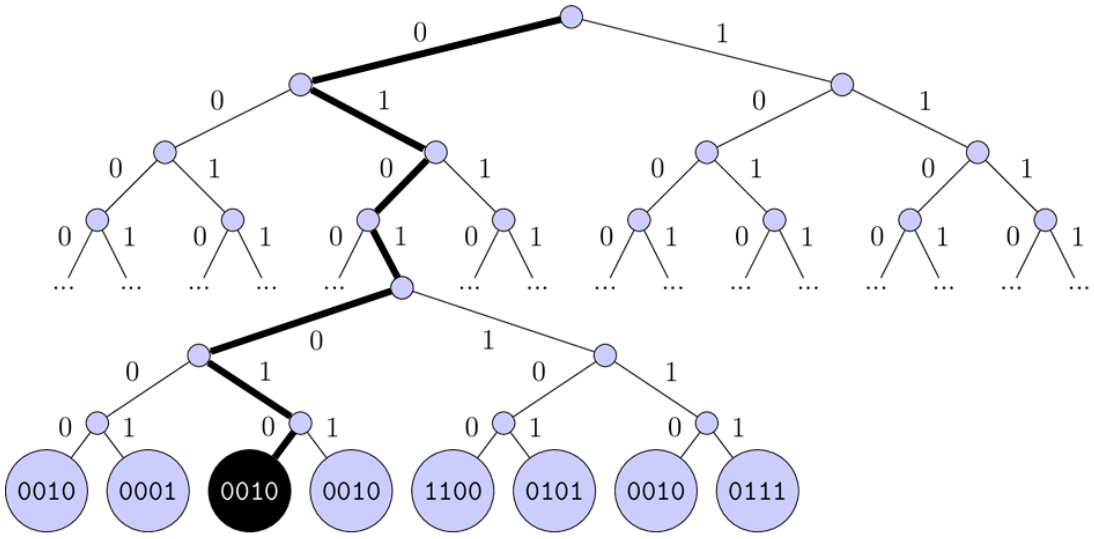

Figure 1. Représentation partielle de l'arbre décodeur du code de Hamming (4, 7)

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.

Solution : il y a $2^7 = 128$ 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) Compléter les lignes 11 à 13 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 tableau code
4      à partir de l'indice i et renvoie le mot étiquetant
5      la feuille atteinte.
6      Précondition : arbre est un arbre binaire de hauteur len(code) - i
7      '''
8      if i == len(code):
9          return arbre.etiquette
10     if code[i] == 0:
11         return decode(.........................)
12     if code[i] == 1:
13         return decode(.........................)
```

```
return decode(arbre.gauche, code, i+1)  # ligne 12
return decode(arbre.droit, code, i+1)  # ligne 14
```

Exercice 2 : (13pt) *Cet exercice porte sur les systèmes d'exploitation, les processus, les structures de données linéaires, la programmation en Python et en particulier la programmation orientée objet.*

Partie A

"Le système d'exploitation est chargé d'allouer les ressources (mémoires, temps processeur, entrées/sorties) nécessaires aux processus et d'assurer que le fonctionnement d'un processus n'interfère pas avec celui des autres."

Source : *Wikipédia, extrait de l'article consacré aux processus.*

1) Expliquer succinctement, dans ce contexte, ce qu'est un processus.
Solution : Un processus est un programme qui est en cours d'exécution.
2) Recopier et compléter le schéma ci-dessous avec les termes suivants :
élu, bloqué, prêt, élection, blocage, déblocage.

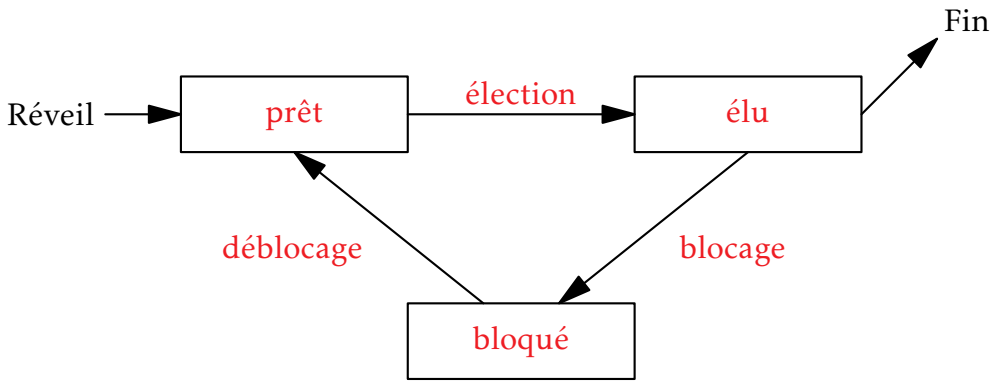

Figure 1. Schéma processus

On considère qu'un monoprocesseur est utilisé. Le système d'exploitation tel un chef d'orchestre, gère l'accès au processeur selon la règle du « premier arrivé, premier servi ». À chaque processus, on associe un instant d'arrivée (instant où le processus demande l'accès au processeur pour la première fois) et une durée d'exécution (durée d'accès au processeur nécessaire pour que le processus s'exécute entièrement).

3) Donner la structure de données la plus adaptée pour gérer l'accès des processus au processeur selon la règle du « premier arrivé, premier servi ».
Solution : Il s'agit de la structure de file.

Le tableau ci-contre présente les instants d'arrivées et les durées d'exécution de quatre processus.

4 processus		
Processus	instant d'arrivée	durée d'exécution
P1	0	4
P2	2	2
P3	3	4
P4	4	2

4) Compléter, à l'aide du tableau, le schéma ci-dessous avec les processus P1 à P4 en utilisant la règle du « premier arrivé premier servi ».

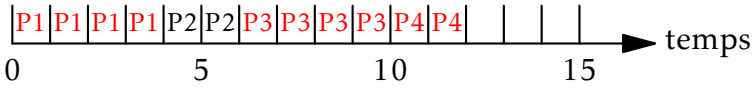

Figure 2. Utilisation du processeur

5) Déterminer le temps qu'a dû attendre le processus P4 avant de pouvoir accéder au processeur.
Solution : Il a attendu 6 cycles.

Partie B

6) Expliquer en quoi consiste la notion d'interblocage.
Solution : C'est lorsque plusieurs processus sont bloqués parce qu'ils attendent chacun que les autres libèrent une ressource dont ils ont besoin.

Afin d'éviter une situation d'interblocage, une solution consiste à attribuer à chaque processus un numéro de priorité.
On souhaite modéliser ce mode de fonctionnement mettant en jeu des numéros de priorité :

- en utilisant une liste de tuples, tuple constitué d'un entier représentant le numéro de priorité ainsi que d'une chaîne de caractères représentant le nom du processus ;
- le processus prioritaire est celui dont le numéro de priorité est le plus petit. Il est donc important que la liste soit et reste triée dans l'ordre décroissant des numéros de priorités.

Exemple :

```
>>> exemple = [(10, 'Edge'), (8, 'Firefox'), (5, 'Chrome'), (1, 'Vivaldi')]
>>> # La liste est triée, le processus le plus prioritaire est 'Vivaldi'
```

On considère la classe `Priority_Queue` dont l'attribut `liste_priorite` est une liste de tuples, constitués d'un numéro de priorité et d'un nom de processus comme dans l'exemple ci-avant.

```
1  class Priority_Queue:
2      def __init__(self):
3          self.liste_priorite = []
4
5      def est_vide(self):
6          """Renvoie Vrai si la liste_priorite
7             est vide, Faux sinon"""
8          return self.liste_priorite == []
9
10     def sortir(self):
11         """Retire et renvoie le dernier élément de
12            liste_priorite"""
13         assert ...
14         ...
15
16     def index_insertion_element(self, element):
17         """Renvoie la position/index d'insertion
18            d'element dans liste_priorite triée
19            par ordre décroissant de numéro priorité"""
20         if self.est_vide():
21             ...
22         else:
23             debut = 0
24             fin = len(self.liste_priorite) - 1
25             milieu = (debut + fin) // 2
26             while ............... <= fin:
27                 if self.liste_priorite[milieu][0] > ...............:
28                     debut = ...
29                 elif self.liste_priorite[milieu][0] < ................:
30                     fin = ...
31                 else:
32                     # cas d'égalité de priorité
33                     ............... milieu
34                 milieu = ...
35             return milieu + 1
```

```
37     def inserer(self, element):
38         """Modifie liste_priorite en insérant
39            element à la position adéquate
40            dans l'ordre décroissant de
41            numéro de priorité"""
```

7) Écrire l'instruction permettant d'instancier `navigateurs`, un objet de la classe `Priority_Queue`.

Solution : `navigateurs = Priority_Queue()`

On rappelle que la méthode `pop()`, appelée sans argument, supprime et renvoie le dernier élément d'une liste.

```
>>> fruits = ['pomme', 'pomme', 'raisin', 'orange', 'poire']
>>> fruits.pop()
'poire'
>>> fruits
['pomme', 'pomme', 'raisin', 'orange']
```

8) Compléter les lignes 13 et 14 du code de la méthode `sortir` qui après avoir vérifié, sous la forme d'une précondition, que l'objet n'est pas vide, retire et renvoie le dernier élément de `liste_priorite`.

```
def sortir(self):
    assert not self.est_vide()
    return self.liste_priorite.pop()
```

Pour maintenir la liste de priorités triée dans l'ordre décroissant des numéros de priorités, il est indispensable de savoir à quelle position on doit insérer un nouvel élément en fonction de sa priorité.
Cette question ne porte que sur la détermination de la position à laquelle devrait être inséré un élément et cela sans effectuer d'insertion.
On considère, par exemple, que `navigateurs.liste_priorite` contient ;
`[(10, 'Edge'), (8, 'Firefox'), (5, 'Chrome'), (1, 'Vivaldi')]`
Si on souhaite insérer :

- l'élément `(12, 'Opera')` on devrait l'insérer au tout début, à la position 0 de `navigateurs.liste_priorite` ;
- l'élément `(6, 'Brave')` on devrait l'insérer à la position 2, juste avant `(5, 'Chrome')` ;
- l'élément `(0, 'Safari')` on devrait l'insérer à la position 4, c'est-à-dire l'ajouter à la fin de la liste.

Pour déterminer la position d'insertion d'un nouvel élément on adapte la méthode dite de recherche dichotomique dans une liste triée dans l'ordre décroissant des numéros de priorités (voir la méthode `index_insertion_element`).
On compare la priorité du tuple `element` à la priorité du tuple se situant au milieu de la `liste_priorite`.

- si elle est strictement supérieure on recommence dans la moitié gauche de `liste_priorite` ;
- si elle est strictement inférieure on recommence dans la moitié droite de `liste_priorite` ;
- si elle est égale la position devra être le milieu.

9) Donner le coût en temps (complexité) de la recherche dichotomique.
Solution : C'est une recherche par dichotomie. Le coût est donc logarithmique puisqu'on divise par deux l'espace de recherche à chaque étape.

10) Compléter les huit lignes 21, 26, 27, 28, 29, 30, 33, et 34 du code de la méthode `index_insertion_element` qui prend en paramètre un élément `element` et qui renvoie la position d'insertion de l'élément `element` en utilisant une méthode dichotomique.

```python
def index_insertion_element(self, element):
    if self.est_vide():
        self.liste_priorite.append(element)
    else:
        debut = 0
        fin = len(self.liste_priorite) - 1
        milieu = (debut + fin) // 2
        while debut <= fin:
            if self.liste_priorite[milieu][0] > element[0]:
                debut = milieu + 1
            elif self.liste_priorite[milieu][0] < element[0]:
                fin = milieu - 1
            else:
                # cas d'égalité de priorité
                return milieu
            milieu = (debut + fin) // 2
        return milieu + 1
```

11) Écrire, sans utiliser la méthode `insert` des listes Python, une méthode inserer qui prend en paramètre un élément `element`, et modifie `liste_priorite` en insérant l'élément `element` à la position adéquate de la liste triée par ordre décroissant des numéros de priorités.
On pourra par exemple ajouter l'élément à la fin de la liste et l'échanger avec les éléments précédents jusqu'à ce qu'il ait pris sa place.
Exemples :

```
>>> navigateurs.liste_priorite
[(10, 'Edge'), (8, 'Firefox'), (5, 'Chrome'), (1, 'Vivaldi')]
>>> navigateurs.inserer((16, 'Brave'))
>>> navigateurs.liste_priorite
[(16, 'Brave'), (10, 'Edge'), (8, 'Firefox'), (5, 'Chrome'), (1, 'Vivaldi')]
>>> navigateurs.inserer((6, 'Safari'))
>>> navigateurs.liste_priorite
[(16, 'Brave'), (10, 'Edge'), (8, 'Firefox'),(6, 'Safari'), (5, 'Chrome'),
 (1, 'Vivaldi')]
>>> navigateurs.inserer((0, 'Lynx'))
>>> navigateurs.liste_priorite
[(16, 'Brave'), (10, 'Edge'), (8, 'Firefox'),(6, 'Safari'), (5, 'Chrome'),
 (1, 'Vivaldi'), (0, 'Lynx')]
```

```python
    def inserer(self, element):
        i = self.index_insertion_element(element)
        j = len(element)
        self.liste_priorite.append(element)
        while i < j:
            self.liste.priorite[i] = self.liste.priorite[i-1]
            self.liste.priorite[i-1] = element
```