CAPES NSI L3 – session 2026

Épreuve disciplinaire appliquée

Exercice 1

Dans le domaine de l'imagerie médicale, les images (IRM, scanners, radiographies) sont souvent acquises avec une profondeur de couleur élevée (par exemple, 12, 14 ou 16 bits par pixel) pour capturer un maximum de détails. Cependant, ces images volumineuses posent des problèmes de stockage et de transmission. Dans cet exercice, nous allons étudier une technique de compression simple, mais efficace, basée sur la réduction de la profondeur de couleur.

Partie A : Représentation des nombres

Question 1 Écrire la représentation binaire du nombre dont la représentation décimale est 43 en base 10.

Question 2 Écrire la représentation décimale du nombre dont la représentation binaire est 100010 en base 2.

Question 3 Écrire une fonction Python nommée `decimal_vers_binaire` qui prend un entier décimal `n` en entrée et renvoie sa représentation binaire sous forme de chaîne de caractères (l'utilisation de la fonction native `bin` de Python n'est pas autorisée).

Question 4 Écrire une fonction Python nommée `binaire_vers_decimal` qui prend une chaîne de caractères binaire `b` en entrée et renvoie sa valeur décimale.

Partie B : Manipulation de Bits

Vous trouverez en annexe (ANNEXE A.1) un rappel sur les opérateurs bit à bit.
Soit $a = 12_{10}$ et $b = 25_{10}$ (a et b sont des entiers non signés codés sur un octet).

Question 5 Évaluer les expressions suivantes en binaire et en décimal.
- $a \,\&\, b$
- $\sim a$
- $a \gg 2$

Question 6 Écrire une fonction Python `est_pair` qui prend en paramètre un entier `n` et qui renvoie `True` si `n` est pair et `False` si `n` est impair. Cette fonction devra obligatoirement utiliser un opérateur bit à bit (l'utilisation de l'opérateur `%` n'est pas autorisée).

Question 7 Écrire une fonction Python `extraire_bits` qui prend en paramètres trois entiers : `n`, `d` et `long`. Cette fonction extrait une séquence de longueur `long` de l'entier `n`, en commençant à la position `d` (le bit le plus à droite est en position 0). La fonction doit renvoyer la valeur décimale de la séquence de bits extraite. Cette fonction doit utiliser des opérateurs bit à bit.
Exemples :

```
>> extraire_bits(0b11010110, 2, 3)
5
>> extraire_bits(0b10101010, 0, 4)
10
```

Partie C : Imagerie médicale

On donne en annexe (ANNEXE A.2) la fonction `lire_image_png`. Pour vous aider à comprendre le code de cette fonction, un extrait de la documentation de la bibliothèque Python Pillow est fourni en annexe (ANNEXE A.3).

Question 8 Compléter la ligne `matrice = [...]` de la fonction `lire_image_png` (ligne 12)

On donne en annexe (ANNEXE A.4) la fonction `reduire_profondeur_couleur`.

Question 9 Expliquer le rôle de la variable `f` dans la fonction `reduire_profondeur_couleur` (ligne 16)

Question 10 Expliquer le rôle de la ligne : `nouveau_pixel = max(0, min(nouveau_pixel, 2**profondeur_cible - 1))` dans la fonction `reduire_profondeur_couleur` (ligne 22)

Question 11 Modifier la fonction `reduire_profondeur_couleur` afin d'utiliser des opérateurs bit à bit à la place de la division et de l'arrondi.

Question 12 Donner un avantage de l'utilisation des opérateurs bit à bit à la place de la division et de l'arrondi.

La réduction de la profondeur de couleur provoque souvent une perte de détails et l'apparition de bandes (effet de *posterization*). Une technique simple, mais efficace pour atténuer cet effet est le *dithering*. Le dithering consiste à répartir les erreurs d'arrondi sur les pixels voisins pour préserver la perception des dégradés. Il existe différentes méthodes pour faire du dithering, nous allons étudier une de ces méthodes : l'algorithme de Floyd-Steinberg. Le principe de cet algorithme est présenté en annexe (ANNEXE A.5). Dans la suite de l'exercice, nous considérerons que nous avons une image en niveau de gris, où chaque pixel est codé sur un octet. L'objectif sera de diminuer le nombre de bits utilisés pour coder chaque pixel de l'image tout en gardant une qualité acceptable (grâce à l'algorithme de Floyd-Steinberg). On désire passer d'une image codée sur 8 bits à une image codée sur 3 bits.

Question 13 Déterminer l'erreur de quantification pour un pixel qui a une intensité de 125 dans l'image d'origine.

Question 14 Déterminer la complexité en fonction du nombre de pixels de l'image de l'algorithme de Floyd-Steinberg.

Question 15 Écrire une fonction Python `floyd_steinberg_dither` qui implémente l'algorithme de Floyd-Steinberg. Cette fonction prend en paramètre une matrice d'entiers, `matrice_pixels` qui représente l'image source et `profondeur_cible` le nombre de bits par pixel de l'image cible. La fonction renvoie une matrice d'entiers qui représente l'image cible.

Question 16 Citer un autre domaine où ce principe de quantification (réduction du nombre de bits) est régulièrement utilisé.

Exercice 2

Une entreprise de logistique souhaite optimiser la livraison de colis en utilisant des drones. La zone de livraison est modélisée par un graphe où les nœuds représentent des points de livraison potentiels et les arêtes représentent les routes aériennes possibles entre ces points. Chaque arête a un poids associé, représentant la distance en Km entre les deux points. On donne en annexe la classe Graphe (ANNEXE B.1) et un exemple de représentation graphique de la zone de livraison (ANNEXE B.2). La méthode `ajouter_arete` permet d'ajouter une arête entre les sommets `sommet1` et `sommet2`.

Question 17 Écrire le code permettant d'implémenter le graphe donné dans l'annexe B.2.

Question 18 Compléter la méthode `ajouter_arete`.

Vous trouverez en annexe (ANNEXE B.3) le code de la fonction `bfs` (parcours en largeur).

Question 19 Donner votre avis sur la ligne `sommet_courant = file.pop(0)` de la fonction `bfs` (ligne 11)

Une erreur s'est glissée dans la fonction `bfs`. Par exemple, actuellement, `bfs(g, 0)` renvoie `{0: 0, 1: inf, 2: inf, 3: inf, 4: inf}` alors que l'on attend `{0: 0, 1: 9, 2: 5, 3: 17, 4: 6}` (avec g le graphe donné dans l'annexe B.2)

Question 20 Corriger l'erreur présente dans la fonction `bfs`.

Question 21 Donner la complexité temporelle dans le pire des cas de l'algorithme bfs.

L'algorithme bfs n'est pas approprié pour calculer le temps de vol minimal entre 2 points de livraison. Il est préférable d'utiliser l'algorithme de Dijkstra.

Question 22 Expliquer le principe de l'algorithme de Dijkstra. Votre explication devra s'appuyer sur un exemple concret : quel est le chemin le plus court pour aller du point de livraison 0 au point de livraison 3 ?

On donne en annexe (ANNEXE B.4) une implémentation incomplète de l'algorithme de Dijkstra en Python. La première ligne du code proposé en ANNEXE B.4 est : `import heapq`. On donne en annexe (ANNEXE B.5) un extrait de la documentation du module `heapq`.

Question 23 Expliquer la notion de tas évoqué dans la documentation du module `heapq`.

Question 24 Compléter la fonction `dijkstra` donnée dans l'annexe (ANNEXE B.4) (recopier sur votre copie uniquement les lignes avec des `...`).

L'entreprise aimerait savoir s'il existe un moyen de partir d'un point, de passer par tous les points de livraison exactement une fois et de revenir au point de départ en une seule fois (sans avoir besoin de recharger le drone). Autrement dit, il faut que la distance totale à parcourir soit inférieure à l'autonomie du drone.

Question 25 Existe-t-il un algorithme exact permettant de résoudre le problème posé ci-dessus dans un temps raisonnable (moins d'une heure de calculs sur une machine standard) pour une zone de livraison comportant une vingtaine de points de livraison ? Si oui, donnez cet algorithme (l'algorithme devra renvoyer VRAI s'il est possible d'effectuer cette tournée pour une zone de livraison donnée et renvoyer FAUX dans le cas contraire).

Exercice 3

Une petite entreprise souhaite évaluer et améliorer le système de sécurité de ses bureaux. Le système actuel repose sur un nombre limité de capteurs et de règles logiques. Le système doit être conçu de telle sorte que les règles de sécurité sont toujours respectées. On fournit la documentation de ce système simplifié en annexe (ANNEXE C.1). On trouvera aussi en annexe (ANNEXE C.2) un rappel des notations utilisées en logique propositionnelle.

Question 26 Traduire les trois règles de sécurité données dans l'annexe (ANNEXE C.1) en formule propositionnelle.

On donne en annexe (ANNEXE C.3) une table de vérité partielle du système (certaines données sont manquantes).

Question 27 Compléter la table de vérité donnée en annexe (ANNEXE C.3).

Question 28 Énumérer le(s) problème(s) mis en lumière par la table de vérité.

Question 29 Proposer une modification des règles afin de supprimer le problème évoqué à la Question 28. Exprimez cette modification en logique propositionnelle et en langage naturel.

L'entreprise envisage d'ajouter un système de badge d'identification. La variable propositionnelle B est VRAI si un badge valide est détecté. L'alarme ne doit pas se déclencher si quelqu'un a badgé.

Question 30 Modifier le système de règles pour tenir compte de la mise en place du système de badge.

Exercice 4

Une base de données CINEPHILES est destinée à gérer les films vus par un groupe de spectateurs, ainsi que les réalisateurs, genres, avis et distributeurs. Les relations sont données en annexe (ANNEXE D.1).

Question 31 Représenter les liens entre les relations de la base de données CINEPHILES. On indiquera les cardinalités (1..1, 1..n, etc.).

Question 32 Exprimez la requête "Titres des films réalisés par des français et distribués par Gaumont" en algèbre relationnelle (en utilisant les opérateurs fournis en annexe (ANNEXE D.2)).

Question 33 Écrire une requête SQL permettant d'obtenir le nom des réalisateurs ayant réalisé des films de genre "science-fiction".

Question 34 Écrire une requête SQL permettant d'obtenir le nom des réalisateurs n'ayant jamais réalisé de film de genre "science-fiction".

Question 35 Écrire une requête SQL permettant d'obtenir le nombre d'avis par genre et par spectateur.

ANNEXES

ANNEXE A : documents de l'exercice 1

ANNEXE A.1 : Opérateurs bit à bit

Opérateur	Symbole	Description
ET	&	1 si les 2 bits sont 1
OU	|	1 si au moins un bit est 1
OU exclusif (XOR)	^	1 si un seul des 2 bits est 1
NON	~	inverse chaque bit (avec complément à 2)
Décalage à gauche	<<	décale les bits vers la gauche
Décalage à droite	>>	décale les bits vers la droite

ANNEXE A.2 : fonction `lire_image_png`

```
from PIL import Image

def lire_image_png(chemin_image):
    """
    Lit une image PNG et retourne une matrice de pixels en niveaux de gris
    ainsi que la largeur et la hauteur de cette matrice.
    """
    try:
        image = Image.open(chemin_image).convert('L') # Convertit en niveaux de gris
        largeur, hauteur = image.size
        pixels = list(image.getdata())
        matrice = [...]
        return matrice, largeur, hauteur
    except FileNotFoundError:
        print(f"Erreur fichier")
        return None, None, None
    except Exception as e:
        print(f"Erreur lors de la lecture de l'image: {e}")
        return None, None, None
```

ANNEXE A.3 : Extrait de la documentation de la bibliothèque Python Pillow

`Image.getdata(band: int | None = None) → core.ImagingCore [source]` Returns the contents of this image as a sequence object containing pixel values. The sequence object is flattened, so that values for line one follow directly after the values of line zero, and so on. Note that the sequence object returned by this method is an internal PIL data type, which only supports certain sequence operations. To convert it to an ordinary sequence (e.g. for printing), use `list(im.getdata())`.

ANNEXE A.4 : fonction `reduire_profondeur_couleur`

```
def reduire_profondeur_couleur(matrice_pixels, p_source, p_cible):
    """
    Reduit la profondeur de couleur d'une matrice de pixels.

    Args:
        matrice_pixels: La matrice de pixels en niveaux de gris.
        p_source: Le nombre de bits par pixel de l'image source.
        p_cible: Le nombre de bits par pixel de l'image cible.

    Returns:
        Une nouvelle matrice de pixels avec la profondeur de couleur reduite.
    """
    if not matrice_pixels:
        return None

    f = 2**p_source / 2**p_cible
    nouvelle_matrice = []
    for ligne in matrice_pixels:
        nouvelle_ligne = []
        for pixel in ligne:
            nouveau_pixel = round(pixel / f)
            nouveau_pixel = max(0, min(nouveau_pixel, 2**p_cible - 1))
            nouvelle_ligne.append(nouveau_pixel)
        nouvelle_matrice.append(nouvelle_ligne)
    return nouvelle_matrice
```

ANNEXE A.5 : principe de l'algorithme de Floyd-Steinberg

L'algorithme de dithering de Floyd-Steinberg est une technique de diffusion d'erreur utilisée pour réduire le nombre de couleurs dans une image (réduction de la profondeur de couleur) tout en minimisant les artefacts visuels tels que les bandes de couleur (banding). Il permet de simuler une plus grande variété de couleurs que celles disponibles dans la palette réduite.

Principe de l'algorithme :

- **Parcours de l'image** : L'algorithme parcourt l'image pixel par pixel, généralement de gauche à droite et de haut en bas.
- **Quantification** : Pour chaque pixel, on détermine la couleur la plus proche dans la palette de couleurs réduite. La différence entre la couleur originale du pixel et sa couleur quantifiée est appelée "erreur de quantification".
- **Diffusion de l'erreur** : Au lieu de simplement ignorer l'erreur de quantification, l'algorithme la distribue aux pixels voisins non encore traités. L'erreur est répartie selon des poids spécifiques, définis par les coefficients de Floyd-Steinberg.

Coefficients de Floyd-Steinberg : L'erreur de quantification est distribuée aux pixels voisins de la manière suivante :

- Le pixel en haut à gauche reste inchangé
- Le pixel au-dessus reste inchangé
- Le pixel en haut à droite reste inchangé
- Le pixel à gauche reste inchangé
- Le pixel à droite reçoit 7/16 de l'erreur.
- Le pixel en bas à gauche reçoit 3/16 de l'erreur.
- Le pixel en dessous reçoit 5/16 de l'erreur.
- Le pixel en bas à droite reçoit 1/16 de l'erreur.

Prenons un exemple pour le calcul de l'erreur de quantification : Nous désirons passer d'une image dans laquelle chaque pixel est codé sur 8 bits à une image dans laquelle chaque pixel est codé sur 4 bits. Pour l'image d'origine, nous avons 256 niveaux possibles, pour l'image finale, nous avons uniquement 16 niveaux possibles.

- 256 niveaux possibles : tous les entiers entre 0 et 255

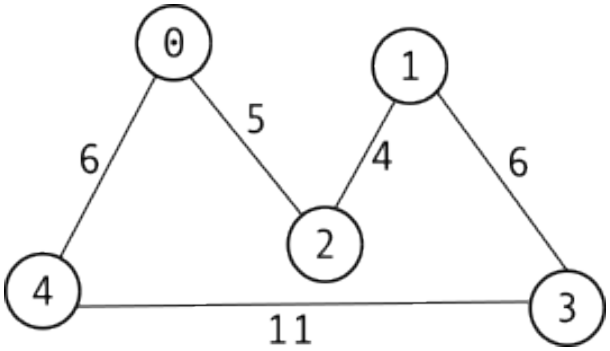
— 16 niveaux possibles : [0, 17, 34, 51, 68, 85, 102, 119, 136, 153, 170, 187, 204, 221, 238, 255]
Pour l'exemple, prenons un pixel qui a pour intensité 210 (dans l'image sur 8 bits), la valeur la plus proche parmi les 16 niveaux est 204, l'erreur de quantification est donc de +6.

ANNEXE B : documents de l'exercice 2

ANNEXE B.1 : classe Graphe (graphe non orienté)

```
class Graphe:
    def __init__(self, nb_sommets):
        self.nb_sommets = nb_sommets
        self.adjacence = [[] for _ in range(nb_sommets)]
    def ajouter_arete(self, sommet1, sommet2, poids):
        ...
```

ANNEXE B.2 : zone de livraison



ANNEXE B.3 : parcours en largeur (BFS)

```
def bfs(graphe, sommet_depart):
    """
    Parcours en largeur du graphe en partant du sommet de depart.
    Renvoie un dictionnaire des distances depuis le sommet de depart.
    """
    #le float('inf') ci-dessous signifie "la valeur infinie positive"
    distances = {sommet: float('inf') for sommet in range(graphe.nb_sommets)}
    distances[sommet_depart] = 0
    file = [sommet_depart]
    while file:
        sommet_courant = file.pop(0)
        for voisin, poids in graphe.adjacence[sommet_courant]:
            if distances[voisin] == 0:
                distances[voisin] = distances[sommet_courant] + poids
                file.append(voisin)
    return distances
```

ANNEXE B.4 : fonction dijkstra

```
import heapq
def dijkstra(graphe, depart, cible):
    distances = [float('inf')] * graphe.nb_sommets
    predecesseurs = [None] * graphe.nb_sommets
    distances[depart] = ...
    tas = [(0, depart)]
    while tas:
        dist_courante, sommet = heapq.heappop(tas)
        if sommet == ... :
            break
        if dist_courante > distances[sommet]:
            continue

        for voisin, poids in ... :
            nouvelle_dist = dist_courante + poids
            if nouvelle_dist < ...
                distances[voisin] = nouvelle_dist
                predecesseurs[voisin] = ...
                heapq.heappush(tas, (nouvelle_dist, voisin))
    chemin = []
    s = cible
    if distances[s] < ...
        while s is not None:
            chemin.append(...)
            s = predecesseurs[...]
        chemin.reverse()
    return distances[cible], chemin
```

ANNEXE B.5 : extrait de la documentation du module Python `heapq`

Ce module expose une implémentation de l'algorithme de file de priorité, basée sur un tas. Les fonctions suivantes sont fournies : `heapq.heappush(heap, item)` Introduit la valeur `item` dans le tas `heap`, en conservant l'invariance du tas. `heapq.heappop(heap)` Extraie le plus petit élément de `heap` en préservant l'invariant du tas. Si le tas est vide, une exception `IndexError` est levée. Pour accéder au plus petit élément sans le retirer, utilisez `heap[0]`.

ANNEXE C : documents de l'exercice 3

ANNEXE C.1 : Description du système de sécurité.

Le système de sécurité fonctionne sur la base d'un ensemble de règles propositionnelles. Les variables propositionnelles sont les suivantes :

— P : La porte porte est ouverte
— M : Un détecteur de mouvement est activé
— N : C'est la nuit
— A : L'alarme est déclenchée

Les règles de sécurité sont les suivantes :

— Règle 1 : L'alarme est déclenchée si la porte est ouverte pendant la nuit,
— Règle 2 : l'alarme se déclenche quand un détecteur de mouvement est activé
— Règle 3 : si la porte est fermée, l'alarme ne peut pas se déclencher durant la journée

ANNEXE C.2 : Rappel des notations utilisées en logique propositionnelle

Symbole	Nom
$\neg$	Négation
$\wedge$	Conjonction
$\vee$	Disjonction
$\rightarrow$	Implication conditionnelle
$\leftrightarrow$	Équivalence logique

ANNEXE C.3 Table de vérité du système

n° ligne	P	M	N	A	règle 1	règle 2	règle 3
1	V	V	V	V	V	V	V
2	V	V	F	V	?	?	?
3	V	F	F	F	?	?	?
4	F	V	F	?	?	?	?
5	F	V	V	V	?	?	?
6	F	F	V	F	?	?	?
7	F	F	F	?	V	V	V
8	V	F	V	?	?	?	?

ANNEXE D : documents de l'exercice 4

ANNEXE D.1 : relations de la base de données CINEPHILES

- `REALISATEUR(*num, nom, prenom, paysOrigine)`
- `FILM(*num, titre, anneeSortie, mumRealisateur, codeGenre, numDistributeur, resume)`
- `GENRE(*code, genre)`
- `DISTRIBUTEUR(*num, nom)`
- `SPECTATEUR(*num, nom, prenom)`
- `AVIS(*numFilm, *numSpectateur, commentaire, note, dateVisionnage)`

ANNEXE D.2 : les opérateurs relationnels

Symbole	Nom
σ	Sélection
π	Projection
$\times$	Produit cartésien
$\bowtie$	Jointure
$-$	Différence
$\cup$	Union
$\cap$	Intersection
$\div$	Division