# BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

**Décembre 2020 – Bac Blanc**

# N.S.I.
## Numérique et Sciences Informatiques

Durée de l'épreuve : **3h30**

*L'usage de la calculatrice n'est pas autorisé*

Le candidat doit traiter les **3 exercices**.
**Attention** : l'**Annexe 3** est à compléter et à rendre avec la copie.

Dès que ce sujet vous est remis, assurez-vous qu'il est complet.
Ce sujet comporte 9 pages numérotées de 1 à 9

**<u>Exercice 1 :</u> Hôpital de la Charité** *(7 points)*

La base de données de l'hôpital est détaillée en **Annexe 1**.

**1°)** Pour créer la structure des deux premiers tableaux, on a utilisé les commandes suivantes :

```
CREATE TABLE Patients (id INT, nom CHAR(30), prenom CHAR(15), sexe CHAR(1), naissance DATE, c_postal CHAR(5), PRIMARY KEY (id));
CREATE TABLE Médecins (matricule INT, nom_prenom CHAR(50), specialite CHAR(20), telephone CHAR(10), PRIMARY KEY (matricule));
```

Que signifient les nombres 30, 15, 1, 5, 50, 20 et 10 dans les parenthèses ?

**2°)** Pour créer la structure du troisième tableau, on a utilisé la commande suivante :

```
CREATE TABLE Ordonnances (code INT, id_patient INT, matricule_medecin INT, date_ord DATE, medicaments CHAR(50), PRIMARY KEY (code), FOREIGN KEY (id_patient) REFERENCES Patients(id), FOREIGN KEY (matricule_medecin) REFERENCES Medecins(matricule));
```

Que signifie l'expression FOREIGN KEY ? Expliquer son fonctionnement.

**3°)** Après avoir entré toutes les données figurant dans les tableaux de l'**Annexe 1**, on souhaite entrer un nouveau patient avec la commande suivante :

```
INSERT INTO Patients (id,nom,prenom,sexe,naissance) VALUES (10,'Didrit','Nicolas','M','1968-04-30')
```

Une erreur est alors affichée, pourquoi ?

**4°)** Quel résultat obtient-on en exécutant la commande suivante :

```
SELECT nom, prenom FROM Patients WHERE c_postal = '92500' ORDER BY nom
```

**5°)** Quelle commande faut-il saisir pour obtenir la liste des noms, prénoms et date de naissance des patients de sexe masculin nés avant l'an 2000, du plus âgé au plus jeune ?
Quel résultat obtient-on ?

**6°)** Quel résultat obtient-on en exécutant la commande suivante :

```
SELECT nom, prenom, O.date_ord FROM Patients JOIN Ordonnances as O ON O.id_patient = id
    WHERE O.medicaments LIKE '%Aspirine%'
```

**7°)** Quelle commande faut-il saisir pour obtenir la liste des dates des ordonnances et des noms des médecins qui ont prescrit des médicaments à Pierrette Martin ?
Quel résultat obtient-on ?

**<u>Exercice 2 :</u> Les cadeaux du Père Noël !** *(7 points)*

Les lutins doivent aider le Père Noël à ranger les cadeaux dans des malles, puis mettre les malles sur le traineau. Ils souhaitent optimiser le rangement pour avoir le moins de malles possible.

**1°)** Exemple :

```
cadeau_01 = {"nom":"BD", "taille":4}
cadeau_02 = {"nom":"Billes", "taille":2}
cadeau_03 = {"nom":"Cartes", "taille":2}
cadeau_04 = {"nom":"Console", "taille":20}
cadeau_05 = {"nom":"DVD", "taille":2}
cadeau_06 = {"nom":"Peluche", "taille":9}
cadeau_07 = {"nom":"Poupée", "taille":5}
cadeau_08 = {"nom":"Robot", "taille":8}
cadeau_09 = {"nom":"Skate", "taille":10}
cadeau_10 = {"nom":"Vélo", "taille":35}
malle_01 = [cadeau_01, cadeau_02, cadeau_03, cadeau_04]
malle_01 += [cadeau_05, cadeau_06, cadeau_07]
malle_02 = [cadeau_08, cadeau_09]
malle_02.append(cadeau_10)
traineau = [malle_01, malle_02]
```

a) Quel est le type des variables cadeau_01, …, cadeau_10 ?

b) Quel est l'affichage dans la console de l'instruction : print(malle_02) ?

**2°)** Les malles ont une contrainte, la somme des tailles des cadeaux qui s'y trouvent ne peux pas dépasser 50.

a) Les deux malles proposées dans l'exemple précédents respectent-elles cette contrainte ? Si non, proposer un autre rangement respectant cette contrainte.

b) Ecrire, en Python, une fonction « disponible(malle) », qui prend en paramètre une malle donnée, et qui renvoie la taille encore disponible en comptant tous les cadeaux présents dans la malle et en respectant la contrainte de tailles imposée par l'énoncé.

**3°)** Tous les cadeaux sont contenus dans une liste « liste_cadeaux » :

```
liste_cadeaux = [cadeau_01, cadeau_02, cadeau_03, cadeau_04, cadeau_05,  cadeau_06, cadeau_07, cadeau_08, cadeau_09, cadeau_10]
```

Les lutins proposent l'algorithme suivant :

On initialise la liste des malles comme étant une liste vide
On prend chaque cadeau de la liste des cadeaux un par un
On parcourt les malles de la liste des malles une par une jusqu'à trouver une malle ayant assez de place pour y mettre le cadeau.
Si une telle malle est trouvée, on place le cadeau dedans
Si aucune malle n'est trouvée, on crée une nouvelle malle, on y place le cadeau, on ajoute cette malle à la liste des malles
On renvoie la liste des malles

a) Ecrire, en Python, une fonction « ranger(liste_cadeaux) » utilisant cet algorithme qui prend en paramètre la liste des cadeaux et qui renvoie la liste des malles à mettre dans le traineau ?

b) Quelle instruction permet de connaitre le nombre de malles présentent dans le traineau après avoir utilisée l'instruction : traineau = ranger(liste_cadeaux) ?

c) Combien de malles sont utilisées en utilisant la fonction précédente sur l'exemple du **1°)** ? Est-ce un rangement optimal ?

**4°)** On propose une amélioration de l'algorithme précédent en effectuant un tri des cadeaux avant d'appliquer l'algorithme.

a) Quelle instruction serait-il judicieux d'ajouter au début de la fonction « ranger(liste_cadeaux) » ?

b) Quel serait alors le rangement de l'exemple du **1°)** obtenu avec une telle amélioration ?

c) Trouver un exemple de liste de cadeaux pour laquelle l'algorithme des lutins ne trouve pas un rangement optimal, même avec l'amélioration proposée. (Une liste de 6 cadeaux peut suffire)

**<u>Exercice 3 :</u> Une méthode de compression : le codage de Huffman** *(7 points)*

Considérons la phrase suivante, contenant 62 caractères (en comptant les espaces) :

UN CHASSEUR SACHANT CHASSER DOIT SAVOIR CHASSER SANS SON CHIEN

**1°)** Codage standard :

a) Si chaque caractère (lettre ou espace) est codé en ASCII étendu sur un octet, combien de bits seront utilisés pour stocker cette phrase en ASCII étendu ?

b) Comme il n'y a ni accents, ni caractères spéciaux, l'ASCII standard suffirait (7 bits par caractère). On peut même utiliser une table de codage spécifique pour coder uniquement les caractères (lettre ou espace) présents dans la phrase…
Pourquoi un codage sur 4 bits pourrait suffire ici ?

c) On choisit la table de codage suivante :

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0000 | A | 0100 | H | 1000 | R | 1100 | V |
| 0001 | C | 0101 | I | 1001 | S | 1101 | espace |
| 0010 | D | 0110 | N | 1010 | T | | |
| 0011 | E | 0111 | O | 1011 | U | | |

Décoder le code : 000101000010100110110
Combien de bits seront nécessaire pour coder la phrase donnée au début en utilisant cette table de codage ?

**2°)** On peut faire mieux !

a) En utilisant un codage de longueur variable, on peut utiliser un code plus court pour les caractères les plus présents et un code plus long pour les caractères les moins présents…

C'est le cas du code Morse par exemple :

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| • − | A | • • • • | H | • − • | R | • • • − | V |
| − • − • | C | • • | I | • • • | S | | espace |
| − • • | D | − • | N | − | T | | |
| • | E | − − − | O | • • − | U | | |

Mais malheureusement, on doit séparer les caractères pour éviter les confusions !
En effet, sans séparateurs, on peut avoir plusieurs interprétations différentes pour un même code …

Donner deux décodages différents du code morse suivant : • • – • • • • –

b) Si l'on veut utiliser un système de codage de longueur variable, il faut s'assurer qu'aucun code n'est préfixe d'un autre code !

Exemple :

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | A | 11001 | H | 1110001 | R | 1110101 | V |
| 100 | C | 11010 | I | 1110010 | S | 1110110 | espace |
| 101 | D | 11011 | N | 1110011 | T | | |
| 11000 | E | 1110000 | O | 1110100 | U | | |

Décoder le code : 111000111010100110011100 0

**3°)** Le codage de Huffman :

Le codage précédent n'est pas vraiment adapté à la phase du début car on utilise 7 bits pour coder le S alors qu'il y a 11 fois la lettre S, mais on n'utilise que 3 bits pour coder le D alors qu'il n'y en a qu'un seul dans la phrase !
Le codage de Huffman permet d'utiliser un code plus court pour les caractères les plus représentés et un code plus long pour les caractères les moins représentés, tout en s'assurant que le décodage puisse se faire sans ambiguïté…
L'ensemble des fonctions utilisées est donné en **Annexe 2**.

a) On entre les instructions suivantes :

```
phrase = "UN CHASSEUR SACHANT CHASSER DOIT SAVOIR CHASSER SANS SON CHIEN"
frequences = compter(phrase)
print("frequences =", frequences)
```

On obtient le résultat suivant :

```
frequences = {'U': 2, 'N': 5, ' ': 9, 'C': 5, 'H': 5, 'A': 7, 'S': 11, 'E': 4, 'R': 4, 'T': 2, 'D': 1, 'O': 3, 'I': 3, 'V': 1}
```

Expliquer le rôle et le fonctionnement de la fonction « compter(texte) ».

b) On entre les instructions suivantes :

```
liste_arbres = [Noeud(None, [frequences[c], c], None) for c in frequences]
arbre_fin = huffman(liste_arbres)
table_code = table_codage(arbre_fin)
```

Compléter l'arbre situé en **Annexe 3** correspondant à la structure créée :

c) On entre l'instruction suivante :

```
print("table_code =", table_code)
```

On obtient le résultat suivant :

```
table_code = {'S': '00', 'A': '010', 'I': '0110', 'E': '0111', 'R': '1000', 'U': '10010', 'T': '10011', ' ': '101', 'N': '1100', 'C': '1101', 'H': '1110', 'D': '111100', 'V': '111101', 'O': '11111'}
```

Combien de bits seront nécessaire pour coder la phrase donnée au début en utilisant cette méthode de codage ?

d) Si on entre les instructions suivantes :

```
table_decode = {table_code[clef]: clef for clef in table_code}
print("table_decode =", table_decode)
```

Quel sera le résultat affiché ?
Décoder le message suivant : 1101111011111011000011010001011100000110

# Annexe 1

## Patients

| id | nom | prenom | sexe | naissance | c_postal |
|---|---|---|---|---|---|
| 1 | Dupond | Pierre | M | 1965-06-05 | 92200 |
| 2 | Dupond | Marie | F | 1968-12-31 | 92200 |
| 3 | Durand | Jacques | M | 1985-10-11 | 92500 |
| 4 | Martin | Jacques | M | 1977-11-22 | 92000 |
| 5 | Martin | Chantal | F | 1980-05-05 | 92000 |
| 6 | Martin | Pierrette | F | 2008-06-20 | 92000 |
| 7 | Renard | Jean | M | 1992-12-21 | 92500 |
| 8 | Lapin | Eric | M | 1970-04-10 | 92500 |
| 9 | Tortue | Marie | F | 2000-10-01 | 92500 |
| 10 | Poulet | Christophe | M | 2001-10-02 | 92380 |

## Medecins

| matricule | nom_prenom | specialite | telephone |
|---|---|---|---|
| 100230 | Bonneau Jean | Chirurgie | 0607080910 |
| 100231 | Terrieur Alain | Chirurgie | 0666077700 |
| 200560 | Dinateur Laure | Radiologie | 0609696906 |
| 300120 | Terrieur Alex | Cardiologie | 0612345678 |
| 400410 | Neymar Jean | Neurologie | 0654321098 |
| 500210 | Versaire Annie | Pneumologie | 0611223344 |

## Ordonnances

| code | id_patient | matricule_medecin | date_ord | medicaments |
|---|---|---|---|---|
| 2020090001 | 2 | 100230 | 2020-09-01 | Trucalex, Pastille rouge |
| 2020090002 | 1 | 300120 | 2020-09-01 | Aspirine |
| 2020090003 | 6 | 200560 | 2020-09-03 | Bidulax, Placebo, Bonbon |
| 2020090004 | 6 | 100230 | 2020-09-04 | Trucalex, Aspirine |
| 2020090005 | 9 | 100231 | 2020-09-06 | Aspirine, Machintruc |
| 2020090006 | 8 | 500210 | 2020-09-10 | Vaziafon, Pilule bleue |
| 2020090007 | 5 | 200560 | 2020-09-11 | Camoulox, Pilule rouge |

# Annexe 2

```
class Noeud :
    def __init__(self, g, v, d):
        self.gauche = g
        self.valeur = v
        self.droite  = d

def compter(texte) :
    frequences = {}
    for c in texte :
        frequences[c] = texte.count(c)
    return frequences

def extraire_mini(liste_arbres) :
    mini = 10**10
    for i in range(len(liste_arbres)) :
        arbre = liste_arbres[i]
        poids = arbre.valeur[0]
        if poids < mini :
            mini = poids
            i_mini = i
    return liste_arbres.pop(i_mini)

def huffman(liste_arbres) :
    while len(liste_arbres) > 1 :
        arbre_1 = extraire_mini(liste_arbres)
        arbre_2 = extraire_mini(liste_arbres)
        nv_poids = arbre_1.valeur[0] + arbre_2.valeur[0]
        nv_arbre = Noeud(arbre_1, [nv_poids], arbre_2)
        liste_arbres.append(nv_arbre)
    return liste_arbres[0]

def table_codage(arbre, table={}, code="") :
    if arbre.gauche == arbre.droite == None :
        poids, caractere = arbre.valeur
        table[caractere] = code
    else :
        table_codage(arbre.gauche, table, code+"0")
        table_codage(arbre.droite, table, code+"1")
    return table

def coder(texte, table) :
    texte_code = ""
    for c in texte :
        texte_code += table[c]
    return texte_code

def decoder(texte, table) :
    texte_decode = ""
    i_debut = 0
    while i_debut < len(texte) :
        i_fin = i_debut + 1
        trouve = False
        while not(trouve) :
            lettre_codee = texte[i_debut:i_fin]
            if lettre_codee in table :
                trouve = True
            else :
                i_fin += 1
        lettre_decodee = table[lettre_codee]
        texte_decode += lettre_decodee
        i_debut = i_fin
    return texte_decode
```

Nom & Prénom : Classe :

# Annexe 3

*(à rendre avec la copie)*