Math93.com

BAC NSI - Correction Amérique Nord - 2025 - Sujet 2A - NSI

Thèmes des exercices (sur 20 points)

- Exercice 1 (6 points) : les tableaux, les dictionnaires, les arbres binaires, la programmation en Python et la récursivité.
- Exercice 2 (6 points) : la gestion des bugs, l'algorithmique, les structures de données et la programmation orientée objet .
- Exercice 3 (8 points) : les bases de données, la programmation en Python, la récursivité et les algorithmes de parcours de graphes..
- Lien vers le sujet

Exercice 1. les tableaux, les dictionnaires, les arbres binaires, la programmation en Python et la récursivité. 6 points

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

Partie A

1. Écriture binaire du caractère 'a'

Corrigé

Le code ASCII de la lettre 'a' est 97. On le convertit en binaire sur 8 bits :
$97 = 01100001$.

2. Appel de la fonction `replique`

Corrigé

`replique([0,0,1,0,1])` renvoie :
$[0,0,0,0,0,0,1,1,1,0,0,0,1,1,1]$

3. Complétion de la fonction `nb_occurrences`

```
# Dans l'éditeur PYTHON
if x in nb_occ:
    nb_occ[x] += 1
else:
    nb_occ[x] = 1
```

4. Complétion de la fonction `majorite`

```
# Dans l'éditeur PYTHON
for cle in dict.keys():
    if dict[cle] > valeur_max:
        valeur_max = dict[cle]
        cle_max = cle
```

Partie B

5. Détection d'une erreur dans la matrice reçue

Corrigé

On suppose qu'une erreur a affecté un bit de donnée dans la matrice suivante :

1	1	1
1	1	0
0	1	1

On calcule la parité (somme des bits) pour chaque ligne :

- Ligne 0 : $1+1+1=3$ (impair) $\Rightarrow$ erreur
- Ligne 1 : $1+1+0=2$ (pair)
- Ligne 2 : $0+1+1=2$ (pair)

Et pour chaque colonne :

- Colonne 0 : $1+1+0=2$ (pair)
- Colonne 1 : $1+1+1=3$ (impair) $\Rightarrow$ erreur
- Colonne 2 : $1+0+1=2$ (pair)

L'erreur se situe donc à l'intersection de la ligne 0 et de la colonne 1, soit à la position (0,1). C'est ce bit de donnée qui doit être inversé.

6. Fonction `erreur_colonne`

```
# Dans l'éditeur PYTHON
def erreur_colonne(mat):
    for j in range(len(mat[0])):
        s = 0
        for i in range(len(mat)):
            s += mat[i][j]
        if s % 2 != 0:
            return j
```

Partie C

7. Correction du mot reçu 1010000

Corrigé

Le code reçu est : 1010000. On le compare à chaque code du tableau de Hamming. Il ne correspond exactement à aucun d'eux. On cherche donc celui qui diffère par un seul bit. En comparant au tableau, on trouve :

Code reçu : 1010000
Code Hamming : 1110000 $\Rightarrow$ Une seule différence à la position 1

Ce code correspond au mot 1000.

Le mot de 4 bits initial est donc 1000

8. Complétion de la fonction `corriger_erreur`

```
def corriger_erreur(code_recu):
    if code_recu in hamming_4_7:
        return code_recu
    else:
        code = code_recu[:]
        for indice in range(7):
            code[indice] = (code[indice] + 1) % 2
            if code in hamming_4_7:
                return code
            else:
                code[indice] = (code[indice] + 1) % 2
```

9. Nombre de feuilles de l'arbre décodeur

Corrigé

Un arbre binaire de hauteur 7 peut avoir au maximum $2^7 = 128$ feuilles. Mais ici, seules les 16 combinaisons de 4 bits sont utilisées pour encoder les messages. Donc, le nombre de feuilles correspond aux 16 mots de 4 bits.

L'arbre décodeur comporte 16 feuilles

10. Complétion de la fonction `decode`

```
if code[i] == 0:
    return decode(arbre.gauche, code, i + 1)
if code[i] == 1:
    return decode(arbre.droit, code, i + 1)
```

Exercice 2. la gestion des bugs, l'algorithmique, les structures de données et la programmation orientée objet . 6 points

Cet exercice porte sur : les bases de données, la programmation en Python, la récursivité et les algorithmes de parcours de graphes..

1. Donner les indices dans l'ordre dans lequel les bonbons sont mangés dans le cas où le collier possède initialement 8 bonbons et l'indice du bonbon restant.

Corrigé

On suppose que les bonbons sont disposés en cercle et numérotés de 0 à 7.
Au départ, tous les bonbons sont présents. Le premier bonbon mangé est celui d'indice 0.
Ensuite, à chaque étape, on saute deux bonbons encore présents (non mangés) et l'on mange le troisième. Comme les bonbons sont disposés circulairement, on tourne autour du collier en tenant compte uniquement des bonbons encore disponibles.
Voici l'ordre exact :

- Étape 1 : on mange le bonbon d'indice **0**
- Étape 2 : on saute 1 (vivant), 2 (vivant) mange **3**
- Étape 3 : on saute 4 (vivant), 5 (vivant) mange **6**
- Étape 4 : on saute 7 (vivant), 1 (vivant) mange **2**
- Étape 5 : on saute 4 (vivant), 5 (vivant) mange **7**
- Étape 6 : on saute 1 (vivant), 4 (vivant) mange **5**
- Étape 7 : on saute 1 (vivant), 4 (vivant) mange **1**

Les indices des bonbons mangés sont donc :

$$\boxed{0, 3, 6, 2, 7, 5, 1}$$

Le bonbon restant est celui d'indice :

$$\boxed{4}$$

2. Expliquer ce qu'est une erreur de type NameError et comment la corriger dans l'instruction proposée.

Corrigé

L'erreur `NameError` est levée lorsqu'un nom de variable ou de constante est utilisé sans avoir été défini. Ici, `true` est mal orthographié.
En Python, le mot-clé correct est `True` (avec une majuscule).
Correction : `collier = [True for i in range(8)]`

3. Compléter la fonction `dernier`.

```
def dernier(n):
    collier = [True for _ in range(n)]
    indice = 0
    collier[indice] = False
    for etape in range(n - 1):
        nb_bonbons_vus = 0
        while nb_bonbons_vus < 3:
            indice += 1
            if indice == n:
                indice = 0
            if collier[indice]:
                nb_bonbons_vus += 1
        collier[indice] = False
    return indice
```

4. Donner l'acronyme le plus adapté à la structure de donnée File.

Corrigé

Il s'agit d'une structure de type $\boxed{FIFO}$ (First In, First Out), c'est-à-dire que les éléments sont retirés dans l'ordre où ils ont été ajoutés.

5. Déterminer l'affichage réalisé lors de l'exécution des instructions.

Corrigé

Initialement, la file contient les éléments dans cet ordre : 0, 1, 2, 3, 4. Après `defile()`, le 0 est retiré.
Puis `enfile(f.defile())` ajoute 1 à la fin, puis 2 à la fin.
La file contient donc : 3, 4, 1, 2.
(Tête) 3 4 1 2 (Queue)

6. Écrire le code de la fonction `dernier_file`.

```
def dernier_file(n):
    f = File()
    for i in range(n):
        f.enfile(i)
    while len(f.file) > 1:
        f.defile()
        f.enfile(f.defile())
        f.enfile(f.defile())
    return f.defile()
```

7. Donner le terme correspondant aux variables `pred`, `valeur` et `succ` en programmation orientée objet.

Corrigé

Ce sont des **attributs d'instance** de la classe `Bonbon`. Chaque objet `Bonbon` contient une valeur et des liens vers ses voisins.

8. Déterminer les valeurs des variables `a` et `b`.

Corrigé

`a = un.valeur = 1`, et `b = un.succ.succ.pred.valeur = deux.pred.valeur = 2`.

Donc : $\boxed{a = 1}$, $\boxed{b = 2}$

9. Compléter la fonction `creer_collier`.

```
def creer_collier(n):
    premier = Bonbon(0)
    actuel = premier
    for i in range(1, n):
        nouveau = Bonbon(i)
        actuel.succ = nouveau
        nouveau.pred = actuel
        actuel = nouveau
    actuel.succ = premier
    premier.pred = actuel
    return premier
```

10. Représenter le collier après suppression du bonbon d'indice 0.

Corrigé

Après suppression de l'élément 0, ses voisins 1 et 7 deviennent adjacents : `1.pred = 7` et `7.succ = 1`. Le collier reste circulaire avec les indices 1 à 7.

11. Quelle expression permet de savoir s'il ne reste qu'un seul bonbon ?

Corrigé

L'expression correcte est : Proposition C : `bonbon.valeur == bonbon.succ.valeur`

12. Compléter la fonction `dernier_chaine`.

```
def dernier_chaine(n):
    bonbon = creer_collier(n)
    while bonbon.pred != bonbon.succ:
        bonbon.pred.succ = bonbon.succ
        bonbon.succ.pred = bonbon.pred
        bonbon = bonbon.succ.succ.succ
    return bonbon.valeur
```

Exercice 3. les bases de données, la programmation en Python, la récursivité et les algorithmes de parcours de graphes.. 8 points

Cet exercice porte sur : les bases de données, la programmation en Python, la récursivité et les algorithmes de parcours de graphes..

1. Pourquoi `id_vol` ne peut pas être une clé primaire de `reservation`

Corrigé

`id_vol` n'est pas unique dans la table `reservation` (plusieurs passagers peuvent réserver le même vol), donc il ne peut pas être une clé primaire.

2. Clé primaire appropriée pour `reservation`

Corrigé

Le couple `(id_vol, id_passager)` constitue une clé primaire, car chaque réservation est unique pour un vol et un passager.

3. Rôle d'une clé étrangère

Corrigé

Une clé étrangère établit un lien entre deux tables en imposant qu'une valeur d'un attribut correspond à une clé primaire dans une autre table, assurant ainsi l'intégrité référentielle.

4. Résultat de la requête SQL

Corrigé

```
SELECT id_vol FROM vol WHERE aeroport_arr = 'CDG';
```

renvoie :

- AI0015
- AI0258
- AI0292

5. Villes destinations au départ de CDG

Corrigé

```
SELECT DISTINCT aeroport_arr FROM vol WHERE aeroport_dep = 'CDG';
```

Résultats : IAD, SYD, NRT

6. Mise à jour de la table `passager`

Corrigé

```
UPDATE passager SET d_totale = 16 WHERE id_passager = 5;
```

7. Correction de l'insertion incorrecte

Corrigé

L'erreur vient du fait que `id_vol = 'AI0256'` existe déjà. Il faut une nouvelle clé primaire :
`INSERT INTO vol VALUES ('AI9999', 'CDG', 'YUL', 6);`

8. Valeur de `graphe_airinfo['T']['P']`

Corrigé

`graphe_airinfo['T']['P']`
renvoie la distance entre Tokyo et Paris : 10

9. Fonction `vol_direct`

```
def vol_direct(graphe, ville1, ville2):
    return ville2 in graphe[ville1]
```

10. Fonction `liste_villes_proches`

```
def liste_villes_proches(graphe, ville, d_max):
    return [v for v, d in graphe[ville].items() if d <= d_max]
```

11. Graphe Droidevant (description)

Corrigé

Graphe composé de deux composantes connexes :

- W —6— P —1— B (et retour)
- T —8— S (et retour)

12. Graphe connexe ?

Corrigé

Proposition A : le graphe de la compagnie AirInfo est connexe.

13. Pourquoi `parcours` est récursive ?

Corrigé

Car la fonction s'appelle elle-même à la ligne 12 pour explorer les voisins non visités.

14. Valeurs des variables `visitees1` et `visitees2`

Corrigé

- `visitees1` = ['W', 'P', 'T', 'B', 'S']
- `visitees2` = ['W', 'P', 'B']

15. Type de parcours

Corrigé

Proposition C : parcours en profondeur

16. Complétion de `est_connexe`

```
def est_connexe(graphe):
    depart = ville_arbitraire(graphe)
    visitees = []
    parcours(graphe, visitees, depart)
    return len(visitees) == len(graphe)
```

17. Résultat de l'appel `mystere(graphe_airinfo, 'W', [], 0, 'B')`

Corrigé

Affiche deux chemins :

- $['W','P','T','B']$ avec coût 25

18. Rôle général de la fonction `mystere`

Corrigé

Elle affiche tous les chemins possibles entre deux villes dans un graphe, avec leur coût total (somme des distances), en évitant les cycles.