Baccalauréat Général

Session 2024

Épreuve : **Numérique et sciences de l'ingénieur**

Durée de l'épreuve : 3h30

Coefficient : 16

PROPOSITION DE CORRIGÉ

Exercice 1 :

Partie A :

1) On ne peut pas choisir l'attribut Nom_artiste comme clé primaire dans la relation CD car Nom_artiste peut apparaître plusieurs fois, or une clé primaire ne doit pas avoir de doublons.

2) La requête renvoie :

Nom_artiste
'Nightwish'
'The Rasmus'

3) La requête renvoie :

CD.Annee
1986
2001

4)
```
UPDATE CD
SET Annee = 2000
WHERE Titre_album = « Wishmaster » ;
```

5)
```
SELECT CD.Titre_album
FROM CD
JOIN Artiste ON CD.Nom_artiste = Artiste.Nom_artiste
JOIN Rangement ON CD.id_album = Rangement.id_album
WHERE Artiste.Style = 'Metal'
AND Rangement.Numero_etagere = 1;
```

6)

Supprimer les enregistrements dans la relation `Rangement`,
Supprimer les enregistrements dans la relation `CD`,
Supprimer les enregistrements dans la relation `Artiste` **(si nécessaire)** :

```
DELETE
FROM CD
WHERE Titre_album = 'Dead Letters';
```

Partie B – Sécurisation

7) Un algorithme de chiffrement symétrique est une méthode de cryptographie dans laquelle la même clé est utilisée pour chiffrer et déchiffrer les données.

8) Un algorithme de chiffrement asymétrique est une méthode de cryptographie qui utilise deux clés distinctes mais mathématiquement liées : une clé publique et une clé privée. Voici les principales caractéristiques de cet algorithme :

- **Clé publique** : Cette clé est accessible à tous et est utilisée pour chiffrer les données.
- **Clé privée** : Cette clé est gardée secrète par le propriétaire et est utilisée pour déchiffrer les données chiffrées avec la clé publique.

9) Pour envoyer la clé symétrique C de manière sécurisée à Bob, le serveur peut utiliser la cryptographie asymétrique.

Chiffrement de la clé symétrique C avec la clé publique de Bob : Le serveur utilise la clé publique de Bob pour chiffrer la clé symétrique C. Étant donné que seule la clé privée correspondante de Bob peut déchiffrer les données chiffrées avec sa clé publique, cela assure que seul Bob pourra accéder à C.
Envoi de la clé chiffrée à Bob : Le serveur envoie la clé symétrique chiffrée à Bob.
Déchiffrement de la clé symétrique C par Bob : Bob utilise sa clé privée pour déchiffrer la clé symétrique C. Cela garantit que seul Bob peut lire la clé symétrique, même si la transmission est interceptée.

EXERCICE 2 :

Partie A :
1)

```
class Marchandise:
   def __init__(self, p: int, v: int) -> 'Marchandise':
      assert v > 0,  # Le volume doit être strictement positif
      assert p >= 0, # Le prix doit être positif ou nul
      self.prix = p
      self.volume = v
```

2) `m1 = Marchandise(20, 7)`

3)

```
def ratio(self) -> float:
```

```
        return self.prix / self.volume
```

4)

```
def prixListe(tab: list) -> int:
        prix_total = 0
        for marchandise in tab:
                prix_total += marchandise.prix
        return prix_total
```

Partie B – Première approche de rangement

5)

Combinaison : [(40, 20)], Volume total : 20 litres, Prix total : 40 €
Combinaison : [(210, 70)], Volume total : 70 litres, Prix total : 210 €
Combinaison : [(160, 40)], Volume total : 40 litres, Prix total : 160 €
Combinaison : [(50, 50)], Volume total : 50 litres, Prix total : 50 €
Combinaison : [(40, 20), (210, 70)], Volume total : 90 litres, Prix total : 250 €
Combinaison : [(40, 20), (160, 40)], Volume total : 60 litres, Prix total : 200 €
Combinaison : [(40, 20), (50, 50)], Volume total : 70 litres, Prix total : 90 €
Combinaison : [(160, 40), (50, 50)], Volume total : 90 litres, Prix total : 210 €

la combinaison optimale qui maximise le prix est [(40, 20), (210, 70)], avec un volume total de 90 litres et un prix total de 250 €.

6) Le qualificatif qui s'applique le mieux à l'algorithme précédent est **glouton**.

7)

```
def tri(tab: list) -> None:
   n = len(tab)
   for i in range(1, n):
      marchandise = tab[i]
      j = i - 1
      while j >= 0 and tab[j].ratio() < marchandise.ratio():
         tab[j + 1] = tab[j]
         j  = j  - 1
      tab[j + 1] = marchandise
```

8)
Le nom de ce tri est **Tri par insertion**.
Son coût temporel dans le pire des cas est **quadratique** $O(n^2)$.

9)
```
def charge(tab: list, volume: int) -> list:
    tri(tab)
    chargement = []
    n = len(tab)
    v_restant = volume
    for i in range(n):
        if tab[i].volume <= v_restant:
            chargement.append(tab[i])
            v_restant -= tab[i].volume
    return chargement
```

Partie C – Rangement optimisé par récursivité

10)
```
def chargeOptimale(tab: list, v_restant: int, i: int) -> list:
    n = len(tab)
    if i >= n:
        return []  # Cas de base : toutes les marchandises ont été essayées, renvoie une liste vide
    else:
        if tab[i].volume > v_restant:
            return chargeOptimale(tab, v_restant, i+1)
        # La marchandise i est trop volumineuse, passer à la suivante
        else:
            # Option 1 : Utiliser la marchandise i
            option1 = chargeOptimale(tab, v_restant - tab[i].volume, i + 1)
            option1 = [tab[i]] + option1  # Ajouter la marchandise i au chargement

            # Option 2 : Ne pas utiliser la marchandise i
            option2 = chargeOptimale(tab, v_restant, i + 1)

            # Comparer les options en fonction du prix total transporté
            if prixListe(option1) > prixListe(option2):
                return option1
            else:
                return option2
```

EXERCICE 3 :

1)

Les attributs de la classe `Piste` et leur type sont les suivants :

1. `nom` (type : `str`) : Le nom de la piste.
2. `denivele` (type : `int`) : Le dénivelé de la piste en mètres.
3. `longueur` (type : `float`) : La longueur de la piste en kilomètres.
4. `couleur` (type : `str`) : La couleur de la piste (par exemple : verte, bleue, rouge, noire). Cet attribut est initialisé avec une chaîne vide.
5. `ouverte` (type : `bool`) : Un indicateur qui précise si la piste est ouverte ou fermée. Cet attribut est initialisé avec la valeur `True`.

2)

```
def set_couleur(self):
        if self.denivele >= 100:
                self.couleur = 'noire'
        elif self.denivele >= 70:
                self.couleur = 'rouge'
        elif self.denivele >= 40:
                self.couleur = 'bleue'
        else:
                 self.couleur = 'verte'
```

3) Proposition D : une liste d'objets de type Piste.

4)

```
for piste in lievre_blanc.get_pistes():
        if piste.get_couleur() == 'verte':
                piste.ouverte = False
```

5)

```
def pistes_de_couleur(lst, couleur):
    return [piste.get_nom() for piste in lst if piste.get_couleur() == couleur]
```

6)

```
def semi_marathon(L):
    distance = 0
    liste_pistes = lievre_blanc.get_pistes()
    for nom in L:
        for piste in liste_pistes:
            if piste.get_nom() == nom:
                distance += piste.get_longueur()
    return distance > 21.1
```

Partie B – Recherche par force brute

7) print(domaine['E']['F'])

8)

```
def voisins(G, s):
    if s in G:
        return list(G[s].keys())
    else:
        return []
```

9)

```
def longueur_chemin(G, chemin):
    precedent = chemin[0]
    longueur = 0
    for i in range(1, len(chemin)):
        longueur = longueur + G[precedent][chemin[i]]
        precedent =  chemin[i]
    return longueur
```

10) La fonction `parcours` est une fonction récursive car elle s'appelle elle-même à partir de la ligne 7, c'est-à-dire qu'elle fait référence à elle-même dans son propre corps,

11)

```
def parcours_dep_arr(G, depart, arrivee):
    liste_chemins = parcours(G, depart)
    chemins_dep_arr = []
    for chemin in liste_chemins:
        if chemin[-1] == arrivee:
            if chemin not in chemins_dep_arr:
                chemins_dep_arr.append(chemin)
    return chemins_dep_arr
```

12)

```
def plus_court(G, depart, arrivee):
    liste_chemins = parcours_dep_arr(G, depart, arrivee)
    if not liste_chemins:
        return None
    chemin_plus_court = liste_chemins[0]
    minimum = longueur_chemin(G, chemin_plus_court)
    for chemin in liste_chemins[1:]:
        longueur = longueur_chemin(G, chemin)
```

```
        if longueur < minimum:
            minimum = longueur
            chemin_plus_court = chemin
    return chemin_plus_court
```

13) La distance minimale ne prend pas en compte d'autres facteurs importants tels que la difficulté du terrain, les conditions météorologiques, la disponibilité des équipements nécessaires, etc. Une approche plus intégrée, prenant en considération plusieurs critères comme le temps de parcours estimé et la difficulté des pistes, permettrait au secouriste d'arriver efficacement et en toute sécurité sur le lieu de l'incident.