2024 Métropole septembre jour 1

Exercice 1

Partie A

1. Ce champ pourrait être de type entier (INT).

2. 5 (c'est le nombre d'agences avec un numéro de téléphone distinct, mais elles ont toutes un téléphone différent...)

3. Il faut qu'aucune agence n'ai le même numéro car une clé primaire doit être unique.

4. `couple_voitures_agences : (#id_agence : INT, #id_voiture : INT)`
La clé primaire est le couple `(id_agence, id_voiture)` et les deux champs sont des clés étrangères.
[On peut aussi créer un champ supplémentaire pour une clé primaire mais c'est inutile]

5.

```
INSERT INTO couple_voitures_agences
VALUES (5, 2);
```

6.

```
UPDATE couple_voitures_agences
SET id_agence = 2
WHERE id_agence = 5 AND id_voiture = 2;
```

7.

```
SELECT type, marque, Agence
FROM Agences
JOIN couple_voitures_agences
ON Agences.id_agence = couple_voitures_agences.id_agence
JOIN  Voitures
ON Voitures.id_voiture = couple_voitures_agences.id_voiture;
```

Partie B

8.
[On ne peut pas créer une telle fonction car il faut l'id_voiture de la voiture et rien ne nous permet de le récupérer.]

```
def insert_voiture(lv, id_agence):
    req = "INSERT INTO Voitures
    (marque, modele, kilometrage, nombre_places, type, carburant)
    VALUES ("+lv[0]+","+lv[1]+","+lv[2]+","+lv[3]+","+lv[4]+","+lv[5]+");"
    # On suppose que l'id_voiture est renvoyé sinon c'est impossible
    id_voiture = execute_requete_insert(req)

    req = "INSERT INTO couple_voitures_agences
    VALUES ("+id_agence+","+id_voiture+");"
    rep = execute_requete_insert(req)
    return rep
```

9. `lv` doit bien posséder les 6 paramètres de la voiture et id_agence doit exister dans la table `Agences`.

Exercice 2

Partie A

1. Il est sur le même réseau que P1 donc `192.168.1.11`.

2. L'adresse du réseau est `192.168.18.0/24`. On enlève généralement les deux adresses (en 0 et 255), il reste donc $256 - 2 = 254$ adresses possibles.

Partie B

3.

```
G  = {  "R1" : ["R2", "R3", "R4", "R6"],
        "R2" : ["R1", "R3", "R5"],
        "R3" : ["R1", "R2", "R5", "R6"],
        "R4" : ["R1", "R6"],
        "R5" : ["R2", "R3", "R6", "R7"],
        "R6" : ["R1", "R3", "R4", "R5", "R7", "R8"],
        "R7" : ["R5", "R6", "R8", "R9"],
        "R8" : ["R6", "R7", "R9"],
        "R9" : ["R7", "R8"]
    }
```

4.

Destination	Suivant	Nombre de sauts
R2	R2	1
R3	R3	1
R4	R4	1
R5	R3	2
R6	R6	1
R7	R6	2
R8	R6	2
R9	R6	3

5.
R1 - R6 - R8 - R9 avec 3 sauts.

6.

```
M  =   [[0, 1, 1, 1, 0, 1, 0, 0, 0],
        [1, 0, 1, 0, 1, 0, 0, 0, 0],
        [1, 1, 0, 0, 1, 1, 0, 0, 0],
        [1, 0, 0, 0, 0, 1, 0, 0, 0],
        [0, 1, 1, 0, 0, 1, 1, 0, 0],
        [1, 0, 1, 1, 1, 0, 1, 1, 0],
        [0, 0, 0, 0, 1, 1, 0, 1, 1],
        [0, 0, 0, 0, 0, 1, 1, 0, 1],
        [0, 0, 0, 0, 0, 0, 1, 1, 0]]
```

7.

```
def degre(MATRICE):
    d = []
    for ligne in MATRICE:
        cpt = 0
        for e in ligne:
            cpt = cpt + e
        d.append(cpt)
    return d
```

8. `[4, 3, 4, 2, 4, 6, 4, 3, 2]`

9. Ce graphe possède deux sommets de degré impair et il est connexe. Il admet donc une chaîne eulérienne. Donc le robot pourra parcourir l'ensemble du réseau en empruntant chaque fibre une et une seule fois.

Partie C

10.
On calcule les coût pour chaque type de liaison :

— $100 : c = \frac{10^8}{100 \times 10^6} = 1$
— $50 : c = \frac{10^8}{50 \times 10^6} = 2$
— $10 : c = \frac{10^8}{10 \times 10^6} = 10$

Le chemin sera : R1 - R2 - R5 - R6 - R7 - R9 avec un coût de 6.

Exercice 3

1. Les quatre cases manquantes :

$2 \times 5 + 1 \times 1 = 11$	$11 \times 20^1 = 220$
$3 \times 5 + 0 \times 1 = 15$	$15 \times 20^0 = 15$

2. On doit ajouter les résultats de la question précédente : $3200 + 220 + 15 = 3435$.

3.

```
M = Maya()
M.ajouter([0, 0, 3])
M.ajouter([0, 1, 2])
M.ajouter([0, 3, 1])
```

4.

```
def nbEtages(self) :
    return len(self.nombre)
```

5.

```
def valeurChiffre(L):
    return L[1] + 5 * L[2]
```

6.

```
def MayaToDec(self):
    coeff = 20**(self.nbEtages() - 1)
    ch_Dec = 0
    while not self.estVide() :
        ch_Maya = self.retirer()
        ch_Dec = ch_Dec + (valeurChiffre(ch_Maya)) * coeff
        coeff = coeff // 20
    return ch_Dec
```

7.

```
def decompChiffre(n):
    if n == 0:
        c = 1
    else:
        c = 0
    p = n % 5
    t = n // 5
    return [c, p, t]
```

8.

```
def DecToMaya(n):
    b20 = DecToVige(n)
    M = Maya()
    for e in b20:
        M.ajouter(decompChiffre(e))
    return M
```

9.

```
def multiplie(self):
    self.nombre = [[1, 0, 0]] + self.nombre
```

10.
[Il y a un problème d'indentation dans la fonction, un = devrait être un == et (m1[0] == 1 and m2[0] == 1).]

```
>>> mystere([0, 1, 1], [0, 3, 1], 0)
([0, 4, 2], 0)
>>> mystere([0, 1, 1], [0, 4, 2], 0)
([1, 0, 0], 1)
```

11.

```
def somme(self, maya2):
    r = 0
    res = Maya()
    for i in range(self.nbEtages()):
        s, r = mystere(self.nombre[i], maya2.nombre[i], r)
        res.ajouter(s)
    if r == 1:
        res.ajouter([0, 1, 0])
    return res
```