

Exercice 1

► Q1.

Table episode : id_epis est clé primaire (et id_sais est clé étrangère).
Table diffusion : une clé primaire est obtenue en considérant le quadruplet (id_sais, id_pf, pays_diff, annee_diff) (et id_sais et id_pf sont des clés étrangères).

► Q2.

```
SELECT * FROM serie
WHERE annee_crea_serie = 2022 AND pays_serie = "FR"
```

► Q3.

```
SELECT titre_serie FROM serie
WHERE pays_serie = "FR"
ORDER BY annee_crea_serie ASC
LIMIT 1
```

► Q4.

```
SELECT e.titre_epis
FROM saison s JOIN episode e ON s.id_sais = e.is_sais
WHERE s.id_serie = 123 AND s.num_sais = 1
ORDER BY e.num_epis ASC
```

► Q5.

```
SELECT s.num_sais, SUM(e.duree_epis)
FROM serie sr JOIN saison s ON sr.id_serie = s.id_serie
JOIN episode e ON s.id_sais = e.id_sais
WHERE sr.titre_serie = "GAME OF THRONES"
GROUP BY s.num_sais
ORDER BY s.num_sais ASC
```

► Q6.

```
SELECT sr.titre_serie, COUNT(s.id_sais) as nb
FROM serie sr JOIN saison s ON sr.id_serie = s.id_serie
JOIN diffusion d ON s.id_sais = d.id_sais
JOIN plateforme p ON d.id_pf = p.id_pf
WHERE d.annee_diff=2023 AND d.pays_diff="FR" AND p.nom_pf="NFX"
GROUP BY sr.id_serie
HAVING nb > 1
```

► Q7.

```
(SELECT DISTINCT sr.titre_serie
FROM serie sr JOIN saison s ON sr.id_serie = s.id_serie
JOIN diffusion d ON s.id_sais = d.id_sais
WHERE d.annee_diff = 2023 AND d.pays_diff = "USA")
```

EXCEPT

```
(SELECT DISTINCT sr.titre_serie
FROM serie sr JOIN saison s ON sr.id_serie = s.id_serie
JOIN diffusion d ON s.id_sais = d.id_sais
WHERE d.annee_diff = 2023 AND d.pays_diff = "FR")
```

Exercice 2

► Q1.

```
def dimensions(G):
    p = len(G)
    q = len(G[0])
    for i in range(1, p):
        if len(G[i]) != q:
            return (0, 0)
    return (p, q)
```

► Q2.

```
def EstRectangle(G, l1, l2, c1, c2):
    for i in range(l1, l2):
        for j in range(c1, c2):
            if G[i][j] != 0:
                return False
    return True
```

► Q3. Voici des exemples de tests (avec G la grille de l'énoncé) :

```
assert EstRectangle(G, 2, 5, 1, 4) # un rectangle de 0
assert not EstRectangle(G, 2, 6, 1, 4) # un rectangle avec un 1
assert EstRectangle(G, 0, 1, 2, 4) # une ligne de 0
assert not EstRectangle(G, 0, 1, 2, 6) # une ligne avec un 1
assert EstRectangle(G, 0, 1, 2, 3) # une case 0
assert not EstRectangle(G, 0, 1, 0, 1) # une case 1
```

► Q4. Il y a $(l2 - l1)(c2 - c1)$ passages dans les boucles et chaque passage a une complexité constante. La complexité est donc en $O((l2 - l1)(c2 - c1))$ et *a fortiori* en $O(pq)$.

► Q5.

```
def PlusGrandRectangle(G):
    p, q = dimensions(G)
    taille_max = 0
    for l1 in range(p):
        for l2 in range(l1+1, p+1):
            for c1 in range(q):
                for c2 in range(c1+1, q+1):
                    if EstRectangle(G, l1, l2, c1, c2) and (l2-l1)*(c2-c1) > taille_max:
                        taille_max = (l2-l1)*(c2-c1)
    return taille_max
```

► Q6. Le nombre d'itérations est :

$$\sum_{i_1=0}^{n-1} \sum_{i_2=i_1}^n \sum_{j_1=0}^{n-1} \sum_{j_2=i_1}^n 1$$

et, à chaque itération, il y a au plus 3 opérations, un test, éventuellement 3 opérations et une affectation mais il y a aussi l'appel à la fonction EstRectangle qui a une complexité en $O((l2 - l1) * (c2 - c1))$.

On a donc un nombre d'itérations en $O(n^4)$ avec, pour chaque itération, un nombre d'opérations en $O(n^2)$.

On a donc une complexité en $O(n^6)$.

► Q7. Tout d'abord pour i fixé : $hmax_{i,i}$ est la hauteur minimale entre les positions i et i donc c'est $H[i]$.

Par ailleurs, la définition de $hmax_{i,j}$ suppose que $i \leq j$ donc $hmax_{i,j} = 0$ lorsque $j < i$.

Enfin, si $i < j$ alors $hmax_{i,j}$ est la hauteur minimale entre les positions i et j : il s'agit de la même hauteur minimale qu'entre les positions i et $j - 1$ sauf si la colonne j est plus basse donc c'est la plus petite valeur entre $hmax_{i,j-1}$ et la hauteur «supplémentaire» $H[j]$.

On a donc :

$$hmax_{i,i} = H[i] \text{ et } hmax_{i,j} = \begin{cases} 0 & \text{si } j < i \\ \min(hmax_{i,j-1}, H[j]) & \text{si } j > i \end{cases}$$

► Q8.

```
def CalculHauteurs(H):
    n = len(H)
    hmax = [[0 for j in range(n)] for i in range(n)]
    for i in range(n):
        hmax[i][i] = H[i]
```

```
for j in range(i+1, n):
    if H[j] < hmax[i][j-1]:
        hmax[i][j] = H[j]
    else:
        hmax[i][j] = hmax[i][j-1]
return hmax
```

► Q9.

```
def ColonneZero(G):
    p, q = dimensions(G)
    tab = [[0 for j in range(q)] for i in range(p)]
    for j in range(q):
        tab[p-1][j] = 1 - G[p-1][j]
    i = p-2
    while i >= 0:
        for j in range(q):
            if G[i][j] == 1:
                tab[i][j] = 0
            else:
                tab[i][j] = 1 + tab[i+1][j]
        i = i - 1
    return tab
```

► Q10.

```
def PgrUptime(G):
    H = ColonneZero(G)
    n = len(H)
    maxi = 0
    for i in range(n):
        histo = H[i]
        t = PgrHistogramme(histo)
        if t > maxi:
            maxi = t
    return maxi
```

► Q11.

```
def convertir(image):
    p, q = dimensions(image)
    g = [[0 for j in range(q)] for i in range(p)]
    for i in range(p):
        for j in range(q):
            a, b, c = image[i][j]
            if (a + b + c)/3 > 120:
                g[i][j] = 1
    return g
```