

Amérique du sud - septembre 2023 - sujet 1 (corrigé)

Exercice 1 (Réseaux, POO et bases de données)

Partie A.

1. On a la table de routage suivante :

Destination	Passerelle	Métrique
R1	R1	0
R2	R3	2
R3	R3	1
R4	R3	3
R5	R3	2
R6	R3	2
R7	R3	2
R8	R8	1

2. (a) Les instructions suivantes conviennent :

```
r4 = Routeur('R4')
r5 = Routeur('R5')
r1. ajout_destination (r4 , r3 , 3)
r1. ajout_destination (r5 , r3 , 2)
```

(b) La méthode suivante convient :

```
def ajout_destination (self, dest, pasr, m):
    self.table_routage[dest] = (pasr, m)
```

3. La méthode suivante convient :

```
def voisins(self):
    voisins = []
    for routeur in self.table_routage:
        if self.table_routage[routeur][1] == 1:
            voisins.append(routeur)
    return voisins
```

4. Le code suivant convient :

```
def calcul_route(self, dest):
    route = [self]
    routeur_courant = route[-1]
    while routeur_courant != dest:
        routeur_courant = routeur_courant.table_routage[dest][0]
        route.append(routeur_courant)
    return route
```

Partie B.

5. Dans la table `Routeur`, l'attribut `id` est une clé primaire, donc garantie l'unicité des enregistrements. Comme il existe déjà un `id` égal à 3, on obtient une erreur (contrainte d'unicité non vérifiée).

6. La requête suivante convient :

```
SELECT nom FROM Routeur WHERE prix >= 2500 AND prix <= 7000
```

7. On obtient la table suivante :

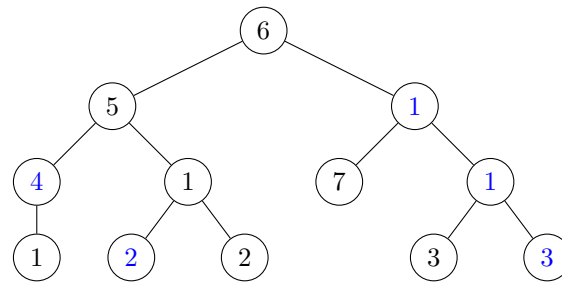
Knuth
Hooper
Pouzin

8. La requête suivante convient :

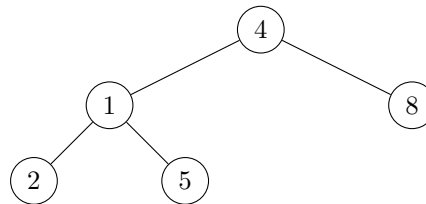
```
SELECT nom FROM Routeur  
JOIN Commande ON Commande.idRouteur = Routeur.id  
WHERE Commande.idClient = 2  
ORDER BY nom
```

Exercice 2 (Arbres binaires et files)

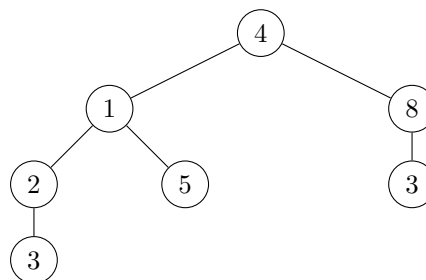
1. On a l'arbre mobile suivant :



2. (a) On a l'arbre suivant :



(b) On a l'arbre mobile suivant :



et les instructions suivantes :

```
n3 = [3, []]
n2 = [2, [n3, []]]
n5 = [5, []]
n8 = [8, [n3, []]]
n1 = [1, [n2, n5]]
a = [4, [n1, n8]]
```

3. Le code suivant convient :

```
1 def poids(arbre):
2     if arbre == []:
3         return 0
4     p = 0
5     file = creer_file()
6     enfiler(file, arbre)
7     while not est_vide(file):
8         arbre = defiler(file)
9         for sous_arbre in arbre[1]:
10             enfiler(file, sous_arbre)
11         p = p + arbre[0]
12     return p
```

4. La fonction suivante convient :

```
def est_mobile(arbre):  
    if arbre == []:  
        return True  
    elif not arbre[1][1] == []:  
        return True  
    else:  
        return est_mobile(arbre[1]) and poids(arbre[1][0]) == poids(arbre[1][1])
```

Exercice 3 (Programmation générale)

1. L'image contient $1\,000 \times 1\,500 = 1\,500\,000$ de pixels. Chaque pixel étant stocké sur trois octets (un pour chacune des composantes du pixel), on en déduit qu'il faut $1\,500\,000 \times 3 = 4\,500\,000$ octets, soit 4,5 Mo, pour stocker l'ensemble des pixels de l'image.
2. On a directement $0 \leq i < n$ et $0 \leq j < m$.
3. (a) La variable `tab_energie` est de type `list`.
(b) L'appel à la fonction `energie` se fait à l'intérieur de la seconde boucle. Il y a donc $n \times m$ appels à la fonction `energie`.
4. La fonction suivante convient :

```
def calcule_energie(couture, tab_energie):  
    energie = 0  
    for pixel in couture:  
        energie += tab_energie[pixel[0]][pixel[1]]  
    return energie
```

5. La fonction suivante convient :

```
def indices_proches(m, i, j):  
    indices = []  
    for y in range(j - 1, j + 2):  
        if y >= 0 and y <= m:  
            indices.append((i, y))  
    return indices
```

6. Le code suivant convient :

```
def calcule_couture(j, tab_energie):  
    n = len(tab_energie) # hauteur de l'image  
    m = len(tab_energie[0]) # largeur de l'image  
    couture = []  
    couture.append((0, j)) # premier pixel de la couture  
    for i in range(1, n):  
        j = couture[-1][1]  
        couture.append(min_energie(tab_energie, indices_proches(m, i, j)))  
    return couture
```

7. La fonction suivante convient :

```
def meilleure_couture(tab_energie):  
    n = len(tab_energie) # hauteur de l'image  
    m = len(tab_energie[0]) # largeur de l'image  
    j_min = 0  
    energie_min = calcule_energie(calcule_couture(0, tab_energie))  
    for j in range(1, m):  
        energie = calcule_energie(calcule_couture(j, tab_energie))  
        if energie < energie_min:  
            energie_min = energie  
            j_min = j  
    return j_min
```