Métropole - mars 2021 - sujet 2 (corrigé)

Exercice 1 (Arbre binaire de recherche et POO)

1. Le code suivant convient :

```
INSERT INTO Eleves VALUES (128, 'Dupont', 'Jean', 'T1')
INSERT INTO Eleves VALUES (200, 'Dupont', 'Jean', 'T1')
INSERT INTO Eleves VALUES (128, 'Dubois', 'Jean', 'T2')
```

2. L'instruction `b1.estim_prix()` renvoie `140000.0` (estimation du prix de `b1`). La valeur renvoyée est de type `float`, car `self.sf` et `self.pm` sont de type `float`.
3. La fonction suivante convient :

```
SELECT titre
FROM Livres
WHERE auteur = 'Molière'
```

4. La fonction suivante convient :

```
SELECT COUNT(*)
FROM Eleves
WHERE classe = 'T2'
```

5. (a) On doit effectuer un parcours infixe : `b2 - b4 - b1 - b5 - b3 - b6`

   (b) La fonction suivante convient :

```
INSERT INTO Emprunts
VALUES (640, 192, '9782070409228', '2020-09-15', NULL)
```

Exercice 2 (SQL)

1. Pour obtenir tous les attributs, il faut utiliser l'instruction `SELECT *`, donc c'est la requête R2 qui convient.
2. (a) La requête suivante convient :

```
SELECT nom, avis
FROM Client
INNER JOIN Reservation ON Client.idClient = Reservation.idClient
WHERE jour = '2021-06-05' AND heure = '19:30:00'
```

   (b) La requête suivante convient :

```
SELECT nom
FROM Plat
INNER JOIN Commande ON Plat.idPlat = Commande.idPlat
INNER JOIN Reservation ON Reservation.idReservation = Commande.idReservation
WHERE (Categorie = 'plat principal' OR Categorie = 'dessert') AND jour = '2021-04-12'
```

3. Cette requête permet d'ajouter un plat qui aura pour identifiant `58`, pour nom `'Pêche Melba'`, pour catégorie `'dessert'`, pour description `'Pêches et glace vanille'` et pour prix `6.5` euros.
4. (a) La requête suivante convient :

```
DELETE FROM Commande
WHERE idReservation = 2047
```

   (b) La requête suivante convient :

```
UPDATE Plat
SET prix = prix + 0.05*prix
WHERE prix < 20.0
```

Exercice 3 (Réseaux et protocoles de routage)

1. (a)
   - ⋆ R1 est dans le réseau L1 et la notation en `/24` indique que les trois premières valeurs de l'adresse IP de R1 correspondent à l'adresse réseau. Donc L1 a pour adresse `192.168.1.0/24`.
   - ⋆ S1 est dans le réseau L2 et la notation en `/16` indique que les deux premières valeurs de l'adresse IP de S1 correspondent à l'adresse réseau. Donc L2 a pour adresse `172.16.0.0/16`.

   (b)
   - ⋆ Pour L1, la plus petite adresse IP est `192.168.1.1` et la plus grande est `192.168.1.254`.
   - ⋆ Pour L2, la plus petite adresse IP est `172.16.0.1` et la plus grande est `172.16.255.254`.

   (c)
   - ⋆ On peut connecter $256 - 2 = 254$ machines au réseau L1 (le 2 correspond à l'adresse réseau et à l'adresse de diffusion qui ne peuvent pas être attribuées).
   - ⋆ On peut connecter $256^2 - 2 = 2^{16} - 2 = 65534$ machines au réseau L2 (le 2 correspond à l'adresse réseau et à l'adresse de diffusion qui ne peuvent pas être attribuées).

2. (a) Il est utile d'avoir plusieurs chemins possibles en cas de panne (routeur ou connexion entre routeurs) ou encore en cas de trafic réseau trop important au niveau d'un routeur.

   (b) Pour relier R1 à R6 il est possible d'effectuer seulement trois sauts : R1 - R2 - R5 - R6

   (c) On a les coûts suivants :

Liaison	R1-R2	R2-R5	R5-R6	R2-R3	R3-R4	R4-R5	R3-R5
Coût	10	10	10	1	1	1	10

   Le chemin reliant R1 à R6 ayant le plus petit coût est : R1 - R2 - R3 - R4 - R5 - R6.
   Son coût est de : 10 (R1-R2) + 1 (R2-R3) + 1 (R3-R4) + 1 (R4-R5) + 10 (R5-R6) = 23

3. On a les deux tables suivantes :

IP réseau de destination	**Passerelle suivante**	**Interface**
`10.1.3.0/24`	`10.1.3.2/24`	Interface 1
`10.1.4.0/24`	`10.1.4.2/24`	Interface 2
`10.1.6.0/24`	`10.1.6.2/24`	Interface 3
`10.1.7.0/24`	`10.1.7.1/24`	Interface 4
`172.16.0.0/16`	`10.1.7.2/24`	Interface 4
`192.168.1.0/24`	`10.1.3.1/24`	Interface 1

Table de routage du routeur R5

IP réseau de destination	**Passerelle suivante**	**Interface**
`172.16.0.0/16`	`172.16.0.1/16`	Interface 1
`10.1.7.1/24`	`10.1.7.1/24`	Interface 2
`192.168.1.0/24`	`10.1.7.1/24`	Interface 2

Table de routage du routeur R6

Exercice 4 (Processus)
Partie A.

1. ⋆ Pour passer à l'exécution, le Programme 1 a besoin du modem, mais le modem est déjà utilisé par le Programme 2. Le processus p1 est donc bloqué.
   ⋆ Pour passer à l'exécution, le Programme 2 a besoin de l'imprimante, mais l'imprimante est déjà utilisée par le Programme 3. Le processus p2 est donc bloqué.
   ⋆ Pour passer à l'exécution, le Programme 3 a besoin de la table traçante, mais la table traçante est déjà utilisée par le Programme 1. Le processus p3 est donc bloqué.

   Les 3 processus sont bloqués et ne pourront pas libérer les ressources attendues par les autres processus (la libération des ressources se faisant après l'exécution). On est donc bien en situation d'interblocage.
2. La modification suivante convient :

```
Programme 3
demander(table traçante)
demander(imprimante)
exécution
libérer(table traçante)
libérer(imprimante)
```

3. Le processus p1 est en attente de ressource. Il est donc bloqué. La réponse est b.

Partie B.

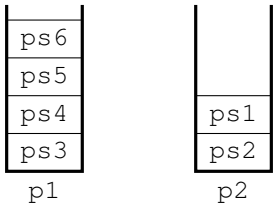
1. Il s'agit de l'instruction `ps -ef` (réponse b).
2. L'identifiant du processus parent à l'origine de tous les processus concernant le navigateur Web est 831 (on cherche le `PPID` du premier processus concernant le navigateur Web).
3. L'identifiant (`PID`) du processus dont le temps d'exécution est le plus long est 6211 (00 :01 :16)

Exercice 5 (Piles et files)

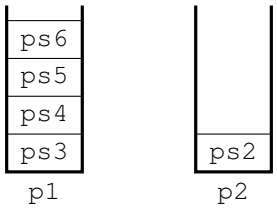
1. La structure de données la plus appropriée pour mettre en oeuvre le mode FIFO (First In First Out) est la file (réponse (d)).
2. Le code suivant convient :

```
def ajouter(lst, proc):
    lst.append(proc)
```

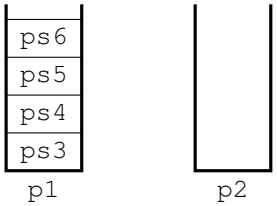
3. `enfiler(file,ps6)` produit :



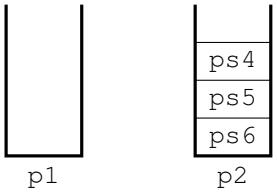
Le premier `defiler(file)` dépile `ps1` de `p2` :



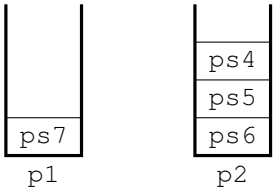
Le deuxième `defiler(file)` dépile `ps2` de `p2` :



Le dernier `defiler(file)` dépile tout `p1` dans `p2` avant de dépiler `ps3` de `p2` :



Finalement le contenu final après `enfiler(file,ps7)` des deux piles est :



4. (a) La file est vide si les deux piles représentant la file sont vides toutes les deux, ainsi :

```
def est_vide(f):
    return pile_vide(f[0]) and pile_vide(f[1])
```

(b) Quoiqu'il arrive, une nouvel élément est ajouté au sommet de la pile `p1`.

```
def enfiler(f, elt):
    empiler(f[0], elt)
```

(c) Deux cas sont à gérer selon si la pile `p2` est vide ou non.

```
def defiler(f):
    assert not est_vide(f), "la file est vide, aucun élément à défiler"
    if pile_vide(f[1]):
        while not pile_vide(f[0]):
            empiler(f[1],depiler(f[0]))
    return depiler(f[1])
```