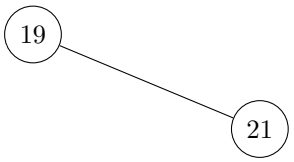
Métropole - mars 2021 - sujet 1 (corrigé)

Exercice 1 (Arbre binaire de recherche et POO)

1. (a) Il y a quatre feuilles. Leur valeurs sont `12`, `val`, `21` et `32`.

   (b) Le sous-arbre gauche du nœud `23` est le suivant :



   (c) ⋆ La hauteur est la longueur du plus long chemin de la racine à une feuille, donc la hauteur est 3.

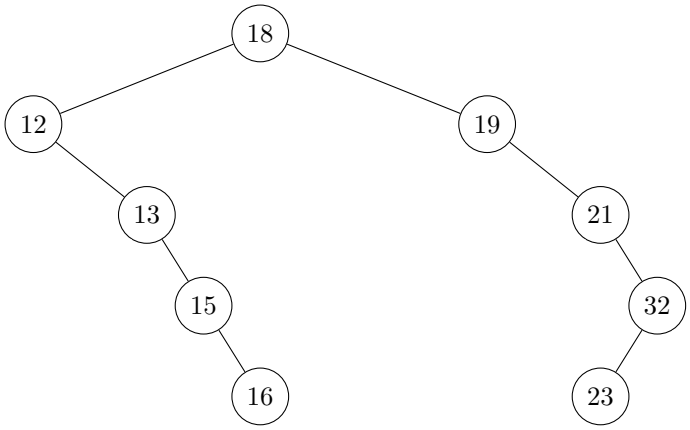
   ⋆ La taille est le nombre de nœuds de l'arbre, donc la taille est 9.

   (d) Comme on est dans le sous-arbre gauche de 18 et dans le sous-arbre droit de 15 et comme toutes les valeurs de l'arbre de distinctes, alors `val` peut prendre les valeurs 16 ou 17.

2. (a) On a l'affichage suivant : 12–13–15–16–18–19–21–23–32

   (b) On a l'affichage suivant : 12–13–16–15–21–19–32–23–18

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



   (b) Les instructions suivantes conviennent (mais il y a d'autres solutions possibles) :

```
racine = Noeud(18)
racine.insere_tout([15, 23, 13, 16, 12, 19, 21, 32])
```

   (c) Les blocs sont exécutés dans l'ordre 3, 2, 1.

4. La fonction suivante convient :

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

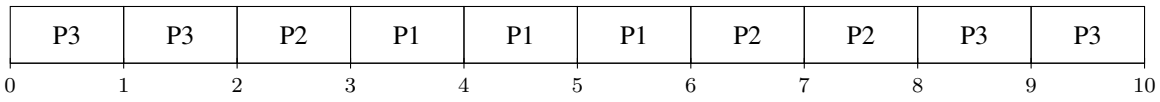
Exercice 2 (Processus et opérateurs booléens)

Partie A

1. b 2. c 3. b 4. d

Partie B

1. On a le schéma suivant :



2. Il s'agit du scénario 2 car on est dans la situation où P1 possède R1 et attend R2 avant de pouvoir continuer (il est donc bloqué) et P3 possède R2 et attend R1 avant de pouvoir continuer (il est donc bloqué lui aussi).

Partie C

1. (a)
   - Le nombre binaire `0110 0011` correspond à `63` en hexadécimal et donc à la lettre `c`.
   - Le nombre binaire `0100 0110` correspond à `46` en hexadécimal et donc à la lettre `F`.

   (b) On a le message chiffré suivant : `0b 1000 1101 1011 0110`

2. (a) On a la table suivante :

a	**b**	**a XOR b**	**(a XOR b) XOR b**
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

   (b) D'après la question précédente, on a : `(a XOR b) XOR b = a`.
   Ainsi, si `m' = m XOR k` est le message chiffré, alors `m' XOR k` redonne le message en clair `m`.
   Autrement dit, un système de chiffrement avec le `XOR` est un système de chiffrement symétrique.

Exercice 3 (SQL)

1. Le nom générique donné aux logiciels qui assurent, entre autres, la persistance des données, l'efficacité de traitement des requêtes et la sécurisation des accès pour les bases de données est SGBD pour Système de Gestion de Bases de Données.
2. (a) L'attribut `numT` est une clé primaire de la table de `Train` et une clé étrangère dans la table `Réservation`. La première requête supprime un enregistrement de la table `Train`. Quand on essaie d'exécuter la seconde requête cela provoque une erreur puisque la valeur `1241` de l'attribut `numT` n'est plus référencé dans la table `Train`.
   Pour que ces suppressions fonctionnent il faudrait échanger l'ordre de ces deux requêtes.

   (b) Un cas pour lequel l'insertion d'un enregistrement dans la table `Reservation` ne serait pas possible serait par exemple de tenter d'ajouter une réservation pour un train qui n'existe pas.
3. Les requêtes SQL suivantes conviennent :

   (a) La requête suivante convient :

```
SELECT numT FROM Train WHERE destination = 'Lyon'
```

   (b) La requête suivante convient :

```
INSERT INTO Reservation(numR, nomClient, prenomClient, prix, numT)
VALUES (1307, 'Turing', 'Alan', 33, 654)
```

   (c) La requête suivante convient :

```
UPDATE Train SET horaireArrivee = 08:11 WHERE numT = 7869
```

4. La requête permet de compter le nombre d'enregistrements dans la table `Reservation` dont le client est Grace Hopper, autrement dit elle renvoie le nombre de réservations faites pour Grace Hopper.
5. La requête suivante convient :

```
def taille(self):
    if self.gauche == None and self.droit == None :
        return 1
    if self.gauche == None :
        return 1 + self.droit.taille()
    elif self.droit == None :
        return 1 + self.gauche.taille()
    else :
        return 1 + self.gauche.taille() + self.droit.taille()
```

Exercice 4 (Tri fusion et « diviser pour régner »)

1. (a) La complexité du tri fusion est en $O(n \log_2 n)$.

   (b) On peut citer les algorithmes du tri par insertion, du tri par sélection et du tri par permutations. Ces algorithmes sont moins performants que le tri fusion car leur complexités sont en $O(n^2)$ dans le pire des cas.

2. On a les affichages suivants :

```
# Renvoie un entier aléatoire N tel que a <= N <= b.
# Alias pour randrange(a,b+1).}
```

   La valeur renvoyée par la fonction est `[1, 2, 3, 4, 5, 6, 7, 8]`

3. La fonction suivante convient :

```
from random import randint
def melange(lst, ind):
    print(lst)
    if ind > 0:
        j = randint(0, ind)
        echange(lst, ind, j)
        melange(lst, ind-1)
```

4. La fonction suivante convient :

```
1  def fusion(L1, L2):
2      L = []
3      n1 = len(L1)
4      n2 = len(L2)
5      i1 = 0
6      i2 = 0
7      while i1 < n1 or i2 < n2:
8          if i1 >= n1:
9              L.append(L2[i2])
10             i2 = i2 + 1
11         elif i2 >= n2:
12             L.append(L1[i1])
13             i1 = i1 + 1
14         else:
15             e1 = L1[i1]
16             e2 = L2[i2]
17
18
19
20
21
22
23     return L
```

Exercice 5 (Routage)

1. Un paquet part du réseau local L1 à destination du réseau local L2.
   (a) D'après l'extrait de la table de routage de R1, pour se rendre dans le réseau de destination `54.37.122.0/24` qui est le réseau de `L2`, le paquet doit emprunter la passerelle `86.154.10.1` qui se situe dans le sous-réseau `86.154.10.0` (24 bits pour le masque) qui relie R1 et R2, le paquet est donc envoyé vers le routeur R2.
   (b) À partir du routeur R2, le paquet doit emprunter la passerelle `37.49.236.22` et donc être envoyé au routeur R6 qui est directement relié au réseau local L2. Finalement le chemin est : L1 - R1 - R2 - R6 - L2.
2. La liaison entre R1 et R2 est rompue.
   (a) Les plus courts chemins sont : R1 - R3 - R2 - R6 et R1 - R3 - R4 - R6.
   (b) La première ligne serait modifiée : la passerelle et l'interface seront alors des adresses du réseau `112.44.65.0/24` (R1 ver R3).
3. On a rétabli la liaison entre R1 et R2. Par ailleurs, pour tenir compte du débit des liaisons, on décide d'utiliser le protocole OSPF (distance liée au coût minimal des liaisons) pour effectuer le routage. Le coût des liaisons entre routeurs est donné ci-dessous :

Liaison	R1-R2	R1-R3	R2-R3	R2-R4	R2-R5	R2-R6	R3-R4	R4-R5	R4-R6	R5-R6
Coût	100	100	?	1	10	10	10	1	10	1

   (a) Le coût de la liaison R2-R3 est $\dfrac{10^8}{10 \times 10^6} = 10$.
   (b) Le chemin parcouru par un paquet partant du réseau L1 et arrivant au réseau L2, en utilisant le protocole OSPF est : L1 - R1 - R2 - R4 - R5 - R6 - L2 pour un coût de 103.
   (c) L'extrait de la table de routage sera modifié pour un paquet à destination de L2, avec le protocole OSPF, pour les routeurs R2 (vers R4 au lieu de R6) et R4 (vers R5 au lieu de R6).