

Amérique du nord - mai 2022 - sujet 2 (corrigé)

Exercice 1 (Arbres binaires de recherche et POO)

1. (a) Le code suivant convient :

```
class Concurrent:  
    def __init__(self, pseudo, temps, penalite):  
        self.nom = pseudo  
        self.temps = temps  
        self.penalite = penalite  
        self.temps_tot = temps + penalite
```

(b) L'attribut `temps_tot` de `c1` est égal à 99.67 (= 87.67 + 12)

(c) Il faut saisir l'instruction `c1.temps_tot`

2. (a) Le code suivant convient :

```
L1 = resultats.queue()  
L2 = L1.queue()  
c1 = L2.tete()
```

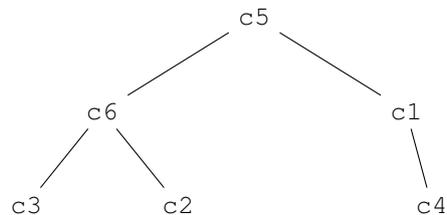
On peut aussi écrire directement : `c1 = resultats.queue().queue().tete()`

(b) Il faut saisir l'instruction `temps_total = resultats.tete().temps_tot`

3. Le code suivant convient :

```
def meilleur_conccurent(L):  
    conc_mini = L.tete()  
    mini = conc_mini.temps_tot  
    Q = L.queue()  
    while not(Q.est_vide()):  
        elt = Q.tete()  
        if elt.temps_tot < mini :  
            conc_mini = elt  
            mini = elt.temps_tot  
        Q = Q.queue()  
    return conc_mini
```

4. On a l'arbre suivant :



Exercice 2 (OS et processus)

1. (a) Il s'agit de la Proposition 2 car on affiche les éléments du dossier morgane
 (b) `cd lycee`
 (c) `mkdir algorithmique`
 (d) `rm image1.jpg`
2. (a) Le PID du parent du processus démarré par la commande `vi` est 927
 (b) Le PID d'un processus enfant du processus démarré par la commande `xfce4-terminal` est 1058
 (c) Les PID de deux processus qui ont le même parent sont 1153 et 1154 (le parent a pour PID 927)
 (d) Les PID des deux processus qui ont consommé le plus de temps processeur sont 923 et 1036
3. (a) On a le chronogramme suivant :

	P1	P2	P3	P1	P3	P1	P3	P3
0	1	2	3	4	5	6	7	8

- (b) On a le chronogramme suivant :

	P2	P1	P1	P1	P3	P3	P3	P3
0	1	2	3	4	5	6	7	8

4. (a) Un processus peut être dans un état : ELU, PRET ou BLOQUE
 Voici une situation qui peut provoquer un interblocage :
 - P1 est à l'état ELU, il demande R1, il l'obtient (car R1 est libre) puis passe à l'état PRET
 - P2 passe à l'état ELU, il demande R2, il l'obtient (car R2 est libre) puis passe à l'état PRET
 - P3 passe à l'état ELU, il demande R3, il l'obtient (car R3 est libre) puis passe à l'état PRET
 - P1 passe à l'état ELU, il demande R2, il ne l'obtient pas (car R2 est déjà utilisé par P2). P1 passe à l'état BLOQUE
 - P2 passe à l'état ELU, il demande R3, il ne l'obtient pas (car R3 est déjà utilisé par P3). P2 passe à l'état BLOQUE
 - P3 passe à l'état ELU, il demande R1, il ne l'obtient pas (car R1 est déjà utilisé par P1). P3 passe à l'état BLOQUE
 Les trois processus se retrouvent à l'état BLOQUE. Nous avons donc un phénomène d'interblocage.
- (b) Pour éviter le phénomène d'interblocage, il suffit d'inverser les deux lignes Demande R3 et Demande R1 pour le processus P3. On obtient alors :
 - P1 est à l'état ELU, il demande R1, il l'obtient (car R1 est libre) puis passe à l'état PRET
 - P2 passe à l'état ELU, il demande R2, il l'obtient (car R2 est libre) puis passe à l'état PRET
 - P3 passe à l'état ELU, il demande R1, il ne l'obtient pas (car R1 est déjà utilisé par P1). P3 passe à l'état BLOQUE
 - P1 passe à l'état ELU, il demande R2, il ne l'obtient pas (car R2 est déjà utilisé par P2). P1 passe à l'état BLOQUE
 - P2 passe à l'état ELU, il demande R3, il l'obtient (car R3 est libre) puis passe à l'état PRET
 - P2 libère R2
 - P2 libère R3
 - P1 passe à l'état ELU, il demande R2, il l'obtient (car R2 est libre) puis passe à l'état PRET
 - P1 libère R1
 - P1 libère R2
 - P3 passe à l'état ELU, il demande R1, il l'obtient (car R1 est libre) puis passe à l'état PRET
 - P3 passe à l'état ELU, il demande R3, il l'obtient (car R3 est libre) puis passe à l'état PRET
 - P3 libère R3
 - P3 libère R1

Exercice 3 (Bases de données et SQL)

1. (a) Une clé primaire d'une relation est un attribut (ou plusieurs attributs) dont la valeur permet d'identifier de manière unique un p -uplet de la relation.
- (b) Une clé étrangère est un attribut qui permet d'établir un lien entre deux relations.
- (c) Un abonné ne peut pas réserver plusieurs fois la même séance, car le couple `idAbonné` et `idSéance` est une clé primaire pour la relation `Réservation`. Il est donc impossible d'avoir deux fois la même séance pour le même abonné.
- (d) On a le tableau suivant :

<code>idAbonné</code>	<code>idSéance</code>	<code>nbPlaces_plein</code>	<code>nbPlaces_réduit</code>
13	737	3	2

2. (a) La requête suivante est la requête correcte :

```
SELECT titre, réalisateur FROM Film WHERE durée < 120
```

(b) Cette requête permet de déterminer le nombre de séances proposées les 22 et 23 octobre 2021.

3. (a) La requête suivante convient :

```
SELECT nom, prénom FROM Abonné
```

(b) La requête suivante convient :

```
SELECT titre, durée
FROM Film
JOIN Séance ON Film.idFilm = Séance.idFilm
WHERE date = 2021-10-12 AND heure = 21:00
```

4. (a) La requête suivante convient :

```
UPDATE Film
SET durée = 127
WHERE titre = "Jungle Cruise"
```

(b) `idSéance` est une clé étrangère pour la relation `Réservation`. La suppression d'une séance risque donc de provoquer des problèmes dans la relation `Réservation` (avec un `Réservation.idSéance` ne correspondant à aucun `Séance.idRéservation`).

- (c) La requête suivante convient :

```
DELETE FROM Séance WHERE idSéance = 135
```

Exercice 4 (Arbres binaires)

1. (a) La racine de l'arbre B est Milka.
- (b) Les feuilles de l'arbre B sont Nemo, Moka, Maya, Museau et Noisette.
- (c) Nuage est une femelle puisque c'est la mère de Nougat.
- (d) Etoile a pour père Ulk et pour mère Maya.
2. (a) Le code suivant convient :

```
def present(arb, nom):
    if est_vide(arb):
        return False
    elif racine(arb) == nom:
        return True
    else :
        return present(droit(arb), nom) or present(gauche(arb), nom)
```

- (b) La fonction suivante convient :

```
def parents(arb):
    if est_vide(gauche(arb)):
        pere=""
    else :
        pere = racine(gauche(arb))
    if est_vide(droit(arb)):
        mere=""
    else :
        mere = racine(droit(arb))
    return (pere, mere)
```

3. (a) Mango et Cacao ont le même père (Domino) et Milka et Cacao ont la même mère (Nougat).
- (b) La fonction suivante convient :

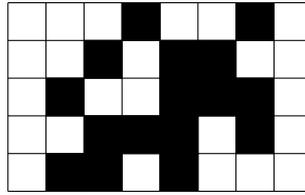
```
def frere_soeur(arbre1, arbre2):
    parents1 = parents(arbre1)
    parents2 = parents(arbre2)
    return parents1[0]==parents2[0] or parents1[1]==parents2[1]
```

4. La fonction suivante convient :

```
def nombre_chiens(arb, n):
    if est_vide(arb):
        return 0
    elif n == 0:
        return 1
    else:
        return nombre_chiens(gauche(arb), n-1) + nombre_chiens(droit(arb), n-1)
```

Exercice 5 (Tableaux et programmation)**Partie A.**

1. (a) On a le dessin suivant :



- (b) Il s'agit du pixel situé à la ligne 3 et à la colonne 0.
 (c) * Le pixel situé en haut à gauche est à la ligne $li - 1$ et à la colonne $co - 1$.
 * Le pixel situé en haut à droite est à la ligne $li - 1$ et à la colonne $co + 1$.
2. (a) Si $image[li - 1][co - 1]$ est égal à $image[li - 1][co + 1]$ alors $image[li][co]$ est égal à 1.
 (b) Le code suivant convient :

```
def remplir_ligne(image, li):
    image[li][0] = 0
    image[li][7] = 0
    for co in range(1, 7):
        if image[li-1][co-1] != image[li-1][co+1]:
            image[li][co] = 1
```

- (c) La fonction suivante convient :

```
def remplir(image):
    for li in range(1, 5):
        remplir_ligne(image, li)
```

Partie B.

1. (a) Le tableau représente le nombre binaire 00101100 et donc sa représentation en base 10 est $2^2 + 2^3 + 2^5 = 44$.
 (b) La fonction suivante convient :

```
def conversion2_10(tab):
    nb_bit = len(tab)
    s = 0
    for i in range(nb_bit):
        s = s + tab[i]*2**(nb_bit-1-i)
    return s
```

- (c) Puisque $78 = 2^6 + 2^3 + 2^2 + 2$, le tableau associé à 78 est $[0, 1, 0, 0, 1, 1, 1, 0]$.
2. (a) Il faut que le bit de poids fort soit à zéro (l'entier ne doit pas dépasser 127) et que le bit de poids faible soit aussi à zéro (l'entier doit être pair). On peut donc utiliser tous les nombres pairs entre 0 et 126.
 (b) La fonction suivante convient :

```
def generer(n, k):
    tab = [None for i in range(k)]
    image = [[0 for j in range(8)] for i in range(k+1)]
    t = conversion10_2(n)
    for i in range(8):
        image[0][i] = t[i]
    tab[0]=n
    for li in range(1, k):
        remplir_ligne(image, li)
        tab[li] = conversion2_10(image[li])
    return tab
```