Nouvelle-Calédonie - novembre 2025 - sujet 1 (corrigé)

Exercice 1 (Graphes, réseaux et POO)
Partie A

1. Il s'agit de ABCEDF car dans un parcours en largeur les voisins directs du sommet de départ doivent être visités avant les autres.
2. On a la table suivante :

Table de routage de F		
routeur	nombre de sauts	prochain routeur
D	1	–

3. On a les deux tables possibles suivantes :

Table de routage de A		
routeur	nombre de sauts	prochain routeur
B	1	–
C	1	–
E	1	–
D	2	C ou E

4. Les tables deviennent stables à partir de l'itération 3.
5. On a les deux tables possibles suivantes :

Table de routage de A		
routeur	nombre de sauts	prochain routeur
B	1	–
C	1	–
E	1	–
D	2	C ou E
F	2	E

Partie B

6. Le code suivant convient :

```
def relie(self, autre):
    if autre not in self.voisins:
        self.voisins.append(autre)
        self.nb_sauts[autre] = 1
        self.prochain[autre] = None
        if self not in autre.voisins:
            autre.relie(self)
```

7. La méthode suivante convient :

```
def relie_liste(self, lst):
    for routeur in lst:
        self.relie(routeur)
```

8. Les instructions suivantes conviennent :

```
C.relie_liste([E, D])
D.relie_liste([E, F])
```

9. Le code suivant convient :

```
def met_a_jour_table(self, autre):
    for r in autre.nb_sauts:
        if r != self:
            if (r not in self.nb_sauts or self.nb_sauts[r] > autre.nb_sauts[r] + 1):
                self.nb_sauts[r] = autre.nb_sauts[r] + 1
                self.prochain[r] = autre
```

10. La méthode suivante convient :

```
def itere_rip(self):
    for v in self.voisins:
        self.met_a_jour_table(v)
```

11. La fonction suivante convient :

```
def itere_rip_global(l_routeurs):
    for routeur in l_routeurs:
        routeur.itere_rip()
```

12. La méthode suivante convient :

```
def itere_rip(self):
    modif = False
    for v in self.voisins:
        b = self.met_a_jour_table(v)
        modif = b or modif
    return modif
```

13. Le code suivant convient :

```
mise_a_jourp = True
while mise_a_jourp:
    new_infop = False
    for routeur in liste_routeurs:
        b = routeur.itere_rip()
        new_infop = b or new_infop
    mise_a_jourp = new_infop
```

Exercice 2 (Bases de données, SQL et POO)

1. Cette requête renvoie `Lac Vert`
2. On peut atteindre le Lac Blanc et les Lacs Noirs
3. La requête suivante convient :

```
SELECT coord_GPS FROM parking
WHERE commune = 'Passy' AND altitude > 800 AND altitude < 1000
```

4. La requête suivante convient :

```
SELECT lac.nom FROM lac
JOIN rando ON rando.arrivee = lac.idL
JOIN parking ON parking.idP = rando.depart
WHERE parking.altitude = 1300 AND parking.commune = 'Cordon'
```

5. Les requêtes suivantes conviennent (il faut commencer par insérer le lac avant la randonnée) :

```
INSERT INTO lac VALUES (42, 'Lac d Anterne', 2059)
INSERT INTO rando VALUES (100, 3, 42)
```

6. La requête suivante convient :

```
UPDATE lac SET nom = 'Lac d Anterne' WHERE nom = 'Lc d Anterne'
```

7. Les requêtes suivantes conviennent (il faut commencer par supprimer les randonnées qui utilisent ce parking) :

```
DELETE FROM rando WHERE depart = 28
DELETE FROM parking WHERE idP = 28
```

8. Le code suivant convient :

```
def get_parking(randos):
    parkings = []
    for rando in randos:
        if rando.depart not in parkings:
            parkings.append(rando.depart)
    return parkings
```

9. Le code suivant convient :

```
def get_nb_rando(parking, randos):
    nb = 0
    for rando in randos:
        if rando.depart == parking:
            nb = nb + 1
    return nb
```

10. La fonction suivante convient :

```
def nb_rando_par_parking(randos):
    parkings = get_parking(randos)
    nb_rando = {}
    for parking in parkings:
        nb_rando[parking] = get_nb_rando(parking, randos)
    return nb_rando
```

Exercice 3 (Algorithmique, représentation binaire et programmation Python)

Partie A : modélisation du problème

1. La valeur minimale d'une cible est 1 ce qui correspond à une seule case égale à 1. La valeur maximale d'une cible est 45 ce qui correspond à toutes les valeurs égales à 9.
2. La cible minimale est 2 (le minimum de la ligne) et la cible maximale est $6 + 4 + 5 + 8 + 2 = 25$.
3. La fonction suivante convient :

```
def extraire_ligne(plateau, i):
    return plateau[i]
```

4. La fonction suivante convient :

```
def extraire_colonne(plateau, i):
    return [plateau[j][i] for j in range(5)]

# Autre possibilité :
def extraire_colonne(plateau, i):
    colonne = []
    for j in range(5):
        colonne.append(plateau[j][i]
    return colonne
```

Partie B : simplification du problème

5. On a le plateau suivant :

```
solution = [[0, 9, 2, 0, 2],
            [8, 0, 0, 0, 1],
            [7, 0, 3, 2, 0],
            [0, 4, 0, 0, 2],
            [0, 0, 0, 0, 4]]
```

6. On a le plateau suivant :

```
[[7, 9, 2, 0, 2],
 [8, 6, 3, 0, 1],
 [7, 7, 3, 2, 7],
 [6, 4, 5, 0, 2],
 [0, 0, 0, 0, 4]]
```

7. Le code suivant convient :

```
def regle1(plateau, cibles_lignes, cibles_colonnes):
    for i in range(5):
        tab = extraire_ligne(plateau, i)
        cible = cibles_lignes[i]
        for j in range(5):
            if tab[j] > cible:
                plateau[i][j] = 0
    for j in range(5):
        tab = extraire_colonne(plateau, j)
        cible = cibles_colonnes[j]
        for i in range(5):
            if tab[i] > cible:
                plateau[i][j] = 0
```

8. La fonction suivante convient :

```
def un_impair(tab):
    n = 0
    for el in tab:
        if el % 2 == 1:
            n = n + 1
    return n == 1
```

9. Le code suivant convient :

```
def regle2(plateau, cibles_lignes, cibles_colonnes):
    for i in range(5):
        ligne = extraire_ligne(plateau, i)
        cible = cibles_lignes[i]
        if cible % 2 == 0 and un_impair(ligne):
            for j in range(5):
                if plateau[i][j] % 2 == 1:
                    plateau[i][j] = 0
    for j in range(5):
        colonne = extraire_colonne(plateau, j)
        cible = cibles_colonnes[j]
        if cible % 2 == 0 and un_impair(colonne):
            for i in range(5):
                if plateau[i][j] % 2 == 1:
                    plateau[i][j] = 0
```

Partie C : problème sur une ligne et représentation binaire

10. La liste `[1, 1, 0, 0, 1]` est un masque solution du problème proposé car $1 + 2 + 2 = 5$.
Les autres masques solutions sont `[0, 1, 1, 0, 0]`, `[0, 0, 1, 0, 1]` et `[0, 0, 0, 1, 0]`.

11. La fonction suivante convient :

```
def somme(tab, masque):
    s = 0
    for i in range(5):
        if masque[i] == 1:
            s = s + tab[i]
    return s
```

12. $26 = 2^4 + 2^3 + 2$, donc 26 est représenté par la liste `[1, 1, 0, 1, 0]`

13. Sur 5 bits, le minimum est `[0, 0, 0, 0, 0]`, soit 0 en décimal, et le maximum est `[1, 1, 1, 1, 1]`, soit 31 en décimal.

14. La fonction suivante convient :

```
def dec2bin(n):
    sortie = [0 for i in range(5)]
    i = 4
    while n > 0:
        sortie[i] = n%2
        n = n//2
        i = i-1
    return sortie

# Autre possibilité :
def dec2bin(entier):
    representation = [0 for i in range(5)]
    for i in range(4, -1, -1):
        representation[i] = entier%2
        entier = entier//2
    return representation
```

15. La fonction suivante convient :

```
def masques_solutions(tab, cible):
    solutions = []
    for i in range(32):
        tab_bin = dec2bin(i)
        if somme(tab, tab_bin) == cible:
            solutions.append(tab_bin)
    return solutions
```

Partie D : retour au jeu Objectif Somme

16. La fonction suivante convient :

```
def teste_solution(tab, ciblesL, ciblesC):
    for i in range(5):
        somme = 0
        ligne = extraire_ligne(tab,i)
        for case in ligne:
            somme = somme+case
        if ciblesL[i] != somme:
            return False
    for j in range(5):
        somme = 0
        colonne = extraire_colonne(tab,j)
        for case in colonne:
            somme = somme+case
        if ciblesC[j] != somme:
            return False
    return True
```