Polynésie - juin 2024 - sujet 1 (corrigé)

Exercice 1 (Graphes, réseaux et protocoles de routage)
Partie A : réseau dans un lycée

1. Seul le dernier octet peut servir à définir les adresses IP des hôtes. Cet octet permet 256 possibilités d'adresses. Toutefois, comme deux adresses sont réservées (celle du réseau et celle de diffusion), on en déduit que l'on peut brancher au maximum 254 machines sur le réseau pédagogie.
2. Comme $217 = 2^7 + 2^6 + 2^4 + 2^3 + 2^0$, on en déduit que l'écriture binaire de 217 est 1101 1001
3. On a facilement $110010 = 2^5 + 2^4 + 2^1 = 32 + 16 + 2 = 50$
4. En appliquant le masque de sous-réseau `255.255.255.0`, on déduuit que cette machine appartient au réseau `110.217.53.0`, qui n'est pas celui du réseau pédagogie 2.
5. On a la table de routage suivante :

Destination	Passerelle	Interface
`110.217.50.0`	`on-link`	`110.217.50.254`
`110.217.52.0`	`110.217.54.253`	`110.217.54.254`
`110.217.54.0`	`on-link`	`110.217.54.254`
`110.217.56.0`	`110.217.54.253`	`110.217.54.254`

6. Seule la seconde ligne de la table de routage précédente est modifiée : pour accéder au réseau `110.217.52.0`, c'est-à-dire au réseau P2, il passe par R4 et non plus par R2. Ainsi la nouvelle passerelle devient `110.217.50.253` et la nouvelle interface devient `110.217.50.254`.
7. L'ajout de R4 ne modifie pas la table de routage de R2 car R2 n'est pas connecté à R4.

Partie B : réseaux et graphes

8. La fonction proposée effectue une recherche en profondeur mais ne garde pas des traces des sommets déjà visités. Cela peut donc entraîner des boucles infinies (plus particulièrement ici un dépassement de capacité de la pile d'appels puisque l'on a affaire à une fonction récursive) si le graphe contient des cycles.
9. La fonction suivante convient :

```
def recherche_v2(R1, R2, visités=[]):
    if R1 == R2:
        return True
    if R1 not in visités:
        visités.append(R1)
    for S in adjacents(R1, G):
        if S not in visités:
            if recherche_v2(S, R2, visités):
                return True
    return False
```

Exercice 2 (Récursivité)

1. Un score peut être marqué uniquement avec des pénalités si et seulement si c'est un multiple de 3. Ainsi, la fonction suivante convient :

```
def possible_avec_penalites_seules(score):
    if score%3 == 0:
        return True
    return False
```

2. On a le tableau suivant :

score	liste des solutions	nombre de solutions
0	`[0]`	1
1	`[]`	0
2	`[]`	0
3	`[0, 3]`	1
4	`[]`	0
5	`[0, 5]`	1
6	`[0, 3, 6]`	1
7	`[0, 7]`	1
8	`[0, 5, 8], [0, 3, 8]`	2
9	`[0, 3, 6, 9]`	1
10	`[0, 3, 10], [0, 7, 10], [0, 5, 10]`	3

3. Par calcul, on a facilement : $f(7)+f(5)+f(3)=1+1+1=3=f(10)$ et donc la formule est bien vraie pour $n=10$.
4. On a les différents cas de base suivants :
   - si $n=0$,3,5,6, on renvoie 1 ;
   - si $n=1$,2,4, on renvoie 0.
5. La fonction suivante convient (elle utilise la formule précédente) :

```
def nb_solutions(score):
    if score in [0, 3, 5, 6]:
        return 1
    if score in [1, 2, 4]:
        return 0
    return nb_solutions(score - 3) + nb_solutions(score - 5) + nb_solutions(score - 7)
```

6. Lors de l'appel `nb_solutions(score)`, le nombre d'appels récursif augmente très rapidement car on ne garde en mémoire aucun des calculs déjà réalisés et on refait plusieurs fois les mêmes calculs. Une façon de diminuer le nombre d'appels récursif est d'utiliser une approche par programmation dynamique.
7. Comme $11=8+3=6+5=5+6=3+8$, il faut utiliser les lignes correspondantes à $n\in\{3,5,6,8\}$. On remarquera qu'il y a d'autres décompositions possibles de 11 comme $11=7+4$ mais elles utilisent toutes des lignes pour lesquelles il n'y a pas de solution.
8. Le code suivant convient :

```
def solutions_possibles(score):
    if score < 0:
        resultat = []
    elif score == 0:
        resultat = [[0]]
    else:
        resultat = []
        for coup in [3, 5, 7]:
            liste = solutions_possibles(score - coup)
            for solution in liste:
                solution.append(coup + solution[-1])
            resultat.append(liste)
    return resultat
```

Exercice 3 (Dictionnaires, bases de données et SQL)
Partie A

1. Il faut saisir successivement les instructions suivantes :
   - `participants['PHILIPSEN Jasper']`
   - `classement_general[participants['PHILIPSEN Jasper']]`
   - `temps_etapes[participants['PINOT Thibaut']][3]`
2. La fonction suivante convient :

```
def calcul_temps_total(d):
    liste_temps = temps_etapes[d]
    temps_total = 0
    for temps in liste_temps:
        temps_total += temps
    return temps_total
```

3. Le code suivant convient :

```
classement = []

for numero_dossard in temps_etapes:
    element = (numero_dossard, calcul_temps_total(numero_dossard))
    classement.append(element)
    pos = len(classement) - 2

    while pos >= 0 and element[1] < classement[pos][1]:
        classement[pos + 1] = classement[pos]
        pos = pos - 1
        classement[pos + 1] = element

for i in range(len(classement)):
    classement_general[classement[i][0]] = i + 1
```

4. Le code suivant convient :

```
tableau_final = []
difference_temps = 0
premier = True
for ligne in tableau_temps:
    coureur = [ligne[0]]
    coureur.append(ligne[1])
    if premier:
        temps_premier = ligne[2]
        coureur.append(temps_premier)
        premier = False
    else:
        difference_temps = ligne[2] - temps_premier
        coureur.append(difference_temps)
    tableau_final.append(coureur)
```

Partie B

5. Une clé primaire garantie l'unicité d'un enregistrement. En particulier, elle n'apparaît qu'une seule fois dans une relation. Comme la relation `Temps` contient l'intégralité des temps de tous les coureurs à toutes les étapes et comme il y a plusieurs coureurs et plusieurs étapes, un numéro de dossard ou un numéro d'étape apparaît nécessairement plusieurs fois. Par conséquent, prendre uniquement l'un des deux comme clé primaire n'est pas possible. En revanche, le couple `(numDossard, numEtape)` n'apparaît, lui, qu'une seule fois dans la table car il n'y a qu'un dossard donné à une étape donnée. On a donc bien une clé primaire en choisissant ce couple.
6. Cette requête renvoie le nom de tous les coureurs de l'équipe Cofidis.
7. La requête suivante convient :

```
SELECT Date FROM Etapes WHERE Type = 'contre-la-montre'
```

8. La requête suivante convient :

```
SELECT directeurSportif FROM Equipes
JOIN Coureurs ON Equipes.nomEquipe = Coureurs.Equipe
WHERE nomCoureur = 'BARDET Romain'
```

9. L'attribut `numEtape` de la relation `Temps` est une clé étrangère faisant référence à l'attribut `numEtape` de la relation `Etapes`. L'étape 5 n'existant pas dans la relation `Etapes`, on ne peut pas l'utiliser dans la relation `Temps` (c'est la contrainte d'inclusion).
10. Il suffit d'intervertir les deux instructions.
11. La requête suivante convient :

```
SELECT SUM(tempsRealise) FROM Temps
JOIN Coureurs ON Coureurs.numDossard = Temps.numDossard
WHERE nomCoureur = 'BARDET Romain'
```