Métropole - juin 2026 - sujet 1 (corrigé)

Exercice 1 (Réseaux, routage et sécurisation des communications)
Partie A : configuration IP du site3

1. On a les écritures binaires suivantes :
   - `Adresse IP : 10101100.00010000.00100000.00001111`
   - `Masque : 11111111.11111111.11110000.00000000`
2. - D'après le masque donné, il y a 12 bits réservées à la partie machine dans les adresses IP de ce réseau. On peut donc représenter $2^{12} = 4096$ adresses différentes. Comme l'adresse réseau et l'adresse de diffusion ne sont pas attribuables, il ne reste que $4096 - 2 = 4094$ adresses disponibles attribuables.
   - La première adresse attribuable est `10101100.00010000.00100000.00000001`, soit `172.16.32.1`.
   - La dernière adresse attribuable est `10101100.00010000.00101111.11111110`, soit `172.16.47.254`.
3. Le paramètre mal configuré est la passerelle. Comme vu à la question précédente, elle devrait être à `172.16.47.254`.
4. Vu le masque CIDR, il n'y a que $2^{32-30} - 2 = 2$ adresses disponibles pour les hôtes. Le dernier octet s'écrit 0000 1000 en binaire, et les adresses attribuables auront pour dernier octet, en binaire, les valeurs `00001001` et `00001010`, soit les adresses : `192.168.0.9` et `192.168.0.10`.

Partie B

5. Le chemin pris est R2 - R3 - R4 - R5 avec un coût de 3.
6. Comme le réseau utilise le protocole RIP et minimise le nombre de sauts, le chemin pris devrait être R2 - R4 - R5 : celui proposé est trop long. Le dysfonctionnement possible doit se trouver au niveau de la liaison entre les routeurs R2 et R4, qui ne doit pas être correctement configurée.
7. Dans un réseau informatique, on cherche à minimiser le *coût* et maximiser le *débit*.
8. Le chemin suivit est R2 - R1 - R3 - R4 - R5, pour un coût total de

$$\frac{10^8}{100 \times 10^6} + \frac{10^8}{4 \times 10^9} + \frac{10^8}{4 \times 10^9} + \frac{10^8}{4 \times 10^9} = 1{,}075$$

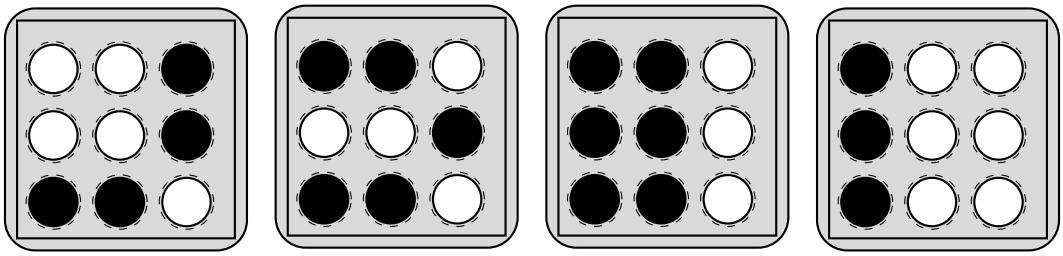
Partie C

9. Si Alice et Bob ne se sont pas mis d'accord en dehors du réseau sur une clé secrète, alors il faut utiliser un chiffrement asymétrique. Une autre possibilité est d'utiliser un protocole ssymétrique pour échanger une clé secrète, puis utiliser un protocole symétrique pour chiffrer les messages (en général plus efficace).
10. Le protocole HTTPS a une couche de sécurité supplémentaire par rapport au protocole HTTP, qui est le chiffrement des données échangées entre le client et le serveur.

Exercice 2 ()

Partie A : représentation binaire des configurations

1. Chaque jeton possède 2 faces, et il y a 9 jetons, donc le nombre de configurations possibles est 2^9.
2. On a les configurations successives suivantes :



état initial ligne 1 ligne 2 colonne 2

3. La liste associée à la configuration de la figure 1 est `[0, 0, 1, 0, 0, 1, 1, 1, 0]`, soit, en décimal : $0+0+64+0+0+8+4+2+0=78$
4. La fonction suivante convient :

```
def representant(lst):
    n = len(lst)
    decimal = 0
    for i in range(n):
        decimal += lst[i]*(2**(n-i-1))
    return decimal
```

5. La fonction suivante convient :

```
def xor_etendu(l_x, l_y):
    res = []
    for i in range(len(l_x)):
        res.append(xor(l_x[i], l_y[i]))
    return res
```

6. Le code suivant convient :

```
def configurations_possibles(config):
    config_bin = binaire(config)
    voisins = []
    for coup in coups_possibles.values():
        voisin = xor_etendu(config_bin, coup)
        voisins.append(representant(voisin))
    return voisins
```

Partie B : graphe des configurations

7. Soit x et y deux configurations telles que $x \to y$. Alors, par définition, il existe un coup c qui permet de passer de x à y. On remarque alors que ce même coup c permet de repasser de y à x. Ainsi, $y \to x$.
8. D'après la question 1, le graphe possède 2^9 noeuds. Vu la figure 3, chaque noeud possède huit arrêtes.
   - Avec une représentation par dictionnaire, il y a donc besoin de stoker 2^9 clés et 8 valeurs dans chaque liste associée à chaque clé, soit $2^9 \times 8 = 2^{12}$ nombres.
   - Avec une représentation par matrice d'adjacence, il y a besoin de stoker $2^9 \times 2^9 = 2^{18}$ nombres.

   Il semble donc bien plus efficace de représenter le graphe à l'aide d'un dictionnaire.
9. Le code suivant convient :

```
def parcours_profondeur(graphe, configuraiton, vus):
    vus.append(configuraiton)
    for s in graphe[configuraiton]:
        if s not in vus:
            parcours_profondeur(graphe, s, vus)
```

10. La configuration gagnante correspondant à l'entier $511 = 256 + 255$, les instructions suivantes conviennent :

```
vus = []
parcours_profondeur(graphe, 4, vus)
assert 511 not in vus
```

11. Pour chercher à minimiser le nombre de coups pour passer d'une configuration à la configuration gagnante, il vaut mieux utiliser une parcours en largeur. En effet, un parcours en profondeur peut explorer des chemins très longs avant de trouver la solution, alors qu'un parcours en largeur explore toutes les configurations à une distance donnée avant de passer à la distance suivante, garantissant ainsi de trouver le chemin le plus court vers la configuration gagnante.

Exercice 3 (Arbres et bases de données)

Partie A : structure arborescente des arguments

1. Il s'agit de la racine
2. Non, ce n'est pas un arbre binaire, car un nœud peut avoir plusieurs enfants. Par exemple, la racine dans la figure 1 possède quatre enfants.
3. Les différents attributs sont :
   - `contenu` de type `str`
   - `sorte` de type `str`
   - `arguments` de type `list`
   - `nb_likes` de type `int`
   - `nb_dislikes` de type `int`
4. Le code suivant convient :

```
def soutenir(self, argument):
    assert argument.sorte == "", "L'argument a déjà été utilisé dans l'arbre."
    self.arguments.append(argument)
    argument.sorte = "pour"
```

5. La méthode suivante convient :

```
def nb_contre(self):
    n = 0
    for argument in self.arguments:
        if argument.sorte == "contre":
            n += 1
    return n
```

6. La méthode `mystère` renvoie le nombre d'arguments **contre** maximal pour une même Affirmation
7. La méthode suivante convient :

```
def evaluation(self):
    total = self.nb_likes - self.nb_dislikes
    for arg in self.arguments:
        if arg.sorte == "pour":
            total = total + arg.evaluation()
        else:
            total = total - arg.evaluation()
    return total
```

Partie B : base de données des utilisateurs et de leurs contributions

8. Les propositions (a), (c) et (d) font parties des rôles d'un SGBD
9. L'attribut `email` aurait également pu servir de clé primaire, en supposant qu'un compte mail n'est utilisé que par une personne
10. La requête suivante convient :

```
SELECT COUNT(*) FROM affirmation WHERE nb_likes >= 50
```

11. La requête suivante convient :

```
SELECT contenu, nb_likes FROM affirmation
JOIN utilisateur ON auteur = pseudo
WHERE prenom = 'Pierre' AND nom = 'Durand' AND sorte = 'sujet'
```

12. Cette requête renvoie les affirmations et leurs contrarguments dont le nombre de *likes* est deux fois plus élevé que le nombre de *likes* de l'affirmation.

13. Les requêtes suivantent conviennent :

```
INSERT INTO reaction VALUES (108, 'i<3descartes', 'like')
UPDATE affirmation SET nb_likes = nb_likes + 1 WHERE id_aff = 108
```

14. Si on supprime l'utilisateur avant ses réactions, s'il en a fait, le SGBD va renvoyé une erreur, car une valeur de la clé étrangère `utilisateur` de la table `reaction` ne correspondra plus à une valeur de la clé primaire `pseudo` de la table `utilisateur` : c'est la contrainte d'intégrité référentielle. Il faut donc supprimer les réactions de l'utilisateur avant de supprimer l'utilisateur lui-même.

15. Les requêtes suivantent conviennent :

```
UPDATE affirmation SET auteur = NULL WHERE auteur = 'i<3rgpd'
DELETE FROM reaction WHERE utilisateur = 'i<3rgpd'
DELETE FROM utilisateur WHERE pseudo = 'i<3rgpd'
```