Éléments de correction sujet 04 (2024)

Exercice 1

1.
```
def echange(tab, i, j):
    tmp = tab[i]
    tab[i] = tab[j]
    tab[j] = tmp
```
2.
```
def triStooge(tab, i, j):
    if tab[i] > tab[j]:
        echange(tab, i, j)
    if (j - i) > 1:
        k = (j - i + 1)//3
        triStooge(tab, i, j-k)
        triStooge(tab, i+k, j)
        triStooge(tab, i, j-k)
```
3. l'algorithme est récursif, car la fonction triStooge s'appelle elle-même.
4. Nous avons, lors du premier appel, i = 0 et j = 5 donc k = (5 - 0 + 1) // 3 = 2
5. On compte les cases de la figure 1, on en trouve 39
6. case 3 : `triStooge(A, 0, 3)`
   case 1 : `triStooge(A, 1, 3)`
   case 2 : `triStooge(A, 2, 3)`
7. Pour moi, il y a des erreurs dans le tableau donné, la question n'est pas faisable.

8. L'algorithme de tri-fusion a un coût strictement meilleur avec $n.\log_2(n)$. Même chose pour le tri par sélection ou par insertion avec n^2 car 2 < 8/3

Exercice 2

1.

Dufour	Marc
Martin	Sophie

2.
```
SELECT nom_medic
FROM medicament
WHERE prix < 3
```
3.
```
INSERT INTO client
VALUES (3, 'Durand', 'Nathalie', '269054958815780')
```
4. `id_client` permet de faire la relation avec la table `client`
   `id_medic` permet de faire la relation avec la table `medicament`
5. ligne 7 : paracétamol 6 comprimés donc 1 boite
   ligne 8 : acide ascorbique 28 comprimés donc 3 boites

6.
```
UPDATE medicament
SET quantite = 447
WHERE id_medic = 4

ou

UPDATE medicament
SET quantite = quantite - 3
WHERE id_medic = 4
```
7.
1 x 3,5 + 3 x 5,5 = 20 euros
8.
```
SELECT nom_medic
FROM medicament
JOIN ordonnance ON ordonnance.id_medic = meidcament.id_medic
WHERE id_ordo = 6
```

Exercice 3

Partie A

1. adresse possible : 192.168.1.3 ; Vu le masque de sous-réseau (255.255.255.0), les 3 premiers octets constituent la partie réseau de l'adresse et le dernier octet constitue la partie machine de l'adresse. La machine de Charlie peut donc avoir une adresse IP de type 192.168.1.X avec X compris entre 3 et 254 (pas 0 car adresse réseau, pas 1 car adresse IP de la machine d'Alice, pas 2 car adresse IP de la machine de Bob et pas 255 car adresse de broadcast (diffusion)).
2.
```
[Transaction('Alice', 'Charlie', 10), Transaction('Bob', 'Alice', 5)]
```
3.
Le paramètre `bloc_precedent` du `bloc0` est à `None` car il n'y a pas de bloc précédent puisque le `bloc0` est le premier bloc créé.
4.
l'attribut `bloc_precedent` du `bloc1` doit être égal au `bloc0`
5.
```
ma_blockchain = Blockchain()
bloc1 = Bloc([Transaction('Alice', 'Charlie', 50), Transaction('Charlie',
'Bob', 30)], ma_blockchain.tete)
ma_blockchain.tete = bloc1
bloc2 = Bloc([Transaction('Bob', 'Charlie', 20), Transaction('Bob',
'Charlie', 20), Transaction('Charlie', 'Alice', 30)], ma_blockchain.tete)
ma_blockchain.tete = bloc2
```
6.
solde Bob = 100 + 30 - 20 - 20 = 90
7.
```
def ajouter_bloc(self, liste_transactions):
    bloc = Bloc(liste_transactions, self.tete)
    self.tete = bloc
```
8.
Il faut utiliser l'adresse de broadcast (adresse de diffusion) : 192.168.1.255

9. Il y a des erreurs dans l'énoncé :
   - à la ligne 6, ce n'est pas `bloc` mais `self`.
   - dans la classe Bloc, l'attribut est `bloc_precedent` et pas `precedent`

```
def calculer_solde(self, utilisateur):
    if self.bloc_precedent is None:
        solde = 0
    else:
        solde = self.bloc_precedent.calculer_solde(utilisateur)
        for transaction in self.liste_transactions:
            if transaction.expediteur == utilisateur:
                solde = solde - transaction.montant
            elif transaction.destinataire == utilisateur :
                solde = solde + transaction.montant
    return solde
```

10. Si on a toujours une instance de la classe `Blockchain ma_blockchain` :
`ma_blockchain.tete.calculer_solde('Alice')`

Partie B

11. On fait une recherche exhaustive quand on teste toutes les possibilités.
12. Le `bloc0` est le premier bloc, l'attribut `bloc_precedent` du `bloc0` est `None`.
L'analyse des lignes 11 à 15 de la classe Bloc nous permettent d'affirmer que l'attribut `hash_bloc_precedent` est `'0'`
13. hash codé sur 256 bits donc 2^{256} possibilités
14. Selon moi, encore un problème dans cette question. On doit partir du principe que la méthode `calculer_hash` utilise `self.nonce` pour effectuer le calcul du hash, mais ce n'est indiqué nulle part !

```
def minage_bloc(self):
    self.nonce = 0
    self.hash = self.calculer_hash()
    while self.hash[0] != '0' or self.hash[1] != '0'  :
        self.nonce = self.nonce + 1
        self.hash = self.calculer_hash()
```