Éléments de correction sujet 12 (2023)

Exercice 1

1.
   a. id_client pour la table client et id_prestation pour la table prestation. En effet, dans les 2 cas chaque ligne possède une valeur qui est unique
   b. La relation prestations possède une clé étrangère (id_client). Cette dernière permet de lier la table prestation à la table client.
   c. *id_client* est de type INT et *nom* est de type TEXT
2.
   a.

Ouellet	0475016031
Foucault	0475918885
Croteau	0475460794
Rivard	0475339127

   b.
```
SELECT date, heure_debut
FROM prestations
WHERE duree > 1 AND employe = “Didier”
```
3.

Foucault
Rouze
Bonenfant
Rivard
Croteau

4.
   a. Nous allons répéter plusieurs fois les mêmes données. Par exemple, pour chaque prestation de type tonte, on aura exactement la même valeur, ce qui est totalement inutile. En cas de changement de tarif de la prestation “tonte”, il faudra modifier le tarif pour toutes les entrées qui auront le type “tonte”
   b. clients (id_client, nom, adresse, code_postal, ville, telephone)
   prestations (id_prestation, #id_client, date, heure_debut, duree, #id_type, employe)
   type (id_type, nom, tarif_horaire)

Exercice 2

1.

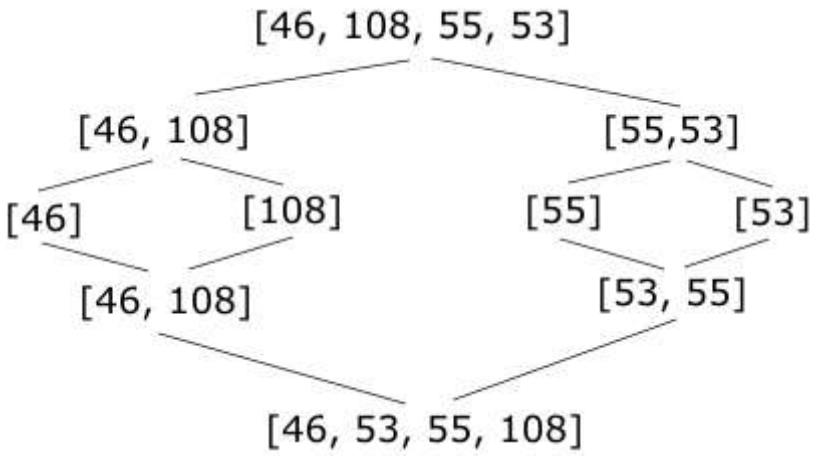
Routeur	destination	passerelle	interface	distance
A	G	C	eth0	2
B	G	C	eth2	2
C	G	F	eth0	1
D	G	E	eth1	2
E	G	F	eth1	1
F	G	-	eth1	0

2.

```
def calcul_montant(prix_TF, quantite_TF, prix_TC, quantite_TC):
    montant = quantite_TF * prix_TF + quantite_TC * prix_TC
    if montant >= 100 and montant < 200 :
        montant = montant - montant*0.10
    elif montant > 200:
        montant = montant - montant*0.20
    return montant
```

3.

a.



b.

```
def fusion(liste1, liste2):
    liste_finale = []
    i1, i2 = 0, 0
    while i1 < len(liste1) and i2 < len(liste2):
        if liste1[i1] <= liste2[i2]:
            liste_finale.append(liste1[i1])
            i1 = i1 + 1
        else :
            liste_finale.append(liste2[i2])
            i2 = i2 + 1
    while i1 < len(liste1):
        liste_finale.append(liste1[i1])
        i1 = i1 + 1
    while i2 < len(liste2):
        liste_finale.append(liste2[i2])
        i2 = i2 + 1
    return liste_finale
```

c.

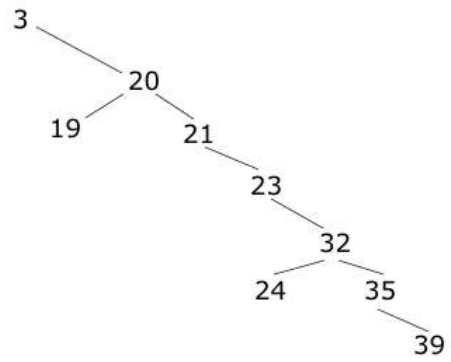
```
def tri_fusion(liste):
    if len(liste) <=1:
        return liste
    m = len(liste)//2
    l1 = tri_fusion(liste[0:m])
    l2 = tri_fusion(liste[m:len(liste)])
    return fusion(l1, l2)
```

Exercice 3

1. Il s'agit de l'arbre 3, car 20 est situé à droite de 29
2. a.

```
a0 = ABR(18, None, None)
a0.inserer_noeud(12)
a0.inserer_noeud(36)
```

b.



c.

hauteur a1 = 4
hauteur a2 = 7

d.

```
def calculer_hauteur(self):
        if self.sa_droit is None and self.sa_gauche is None:
            return 1
        elif self.sa_droit is None :
            return 1 + self.sa_gauche.calculer_hauteur()
        elif self.sa_gauche is None :
            #arbre avec une racine et seulement un sous-arbre
sa_droit
            return 1 + self.sa_droit.calculer_hauteur()
        else :
            return 1 + max(self.sa_gauche.calculer_hauteur(),
self.sa_droit.calculer_hauteur())
```

3.

a.

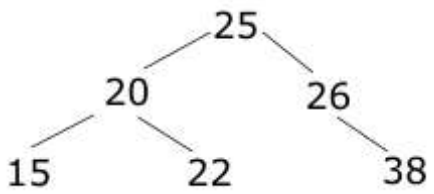
```
   def rechercher_valeur(self,v):
        if self.valeur == v :
            return True
        elif v < self.valeur and self.sa_gauche is not None:
            return self.sa_gauche.rechercher_valeur(v)
        elif v > self.valeur and self.sa_droit is not None:
            return self.sa_droit.rechercher_valeur(v)
        else :
            return False
```

b.

Il y a 4 appels à la méthode `rechercher_valeur` pour atteindre la valeur 20 dans l'arbre a1

4.

a.



b.

```
def rotation_gauche(self):
    pivot = self.sa_droit
    self.sa_droit = pivot.sa_gauche
    pivot.sa_gauche = self
    return ABR(pivot.valeur, pivot.sa_gauche, pivot.sa_droit)
```