Proposition de correction

Exercice 1

Q1.a

prestations (id_prestation, #id_client, date, heure_debut, duree type, employe)

clients (id_client, nom, adresse, code_postal, ville, telephone)

souligné : identifie de façon unique les enregistrements prestations et clients

Q1.b

avec le # : clef primaire de la table clients

Q2.a

Nom	telephone
Ouellet	0475016031
Foucault	0475918885
Croteau	0475460794
Rivard	0475339127

Q2.b

```
SELECT date, heure_debut
FROM prestations
WHERE Employe = 'Didier' AND duree > 1
ORDER BY date, heure_debut
```

Q3

Nom

Rouze

Bonenfant

Foucault

Croteau

Rivard

Q4.a

En cas de changement de tarif

il faut mettre la relation en 3FN

Q4.b

- tarifs(id_tarif, type, tarif_horaire)
- prestations(id_prestation, #id_client, date, heure_debut, duree, #id_tarif, employe)
- clients(id_client, nom, adresse, code_postal ville, telephone)

clés primaires soulignées, clés étrangères précédées par #

Exercice 2

Q1

Routeur	destination	passerelle	interface	distance
A	G	C	eth0	3
B	G	C	eth2	3
C	G	F	eth0	2
D	G	E	eth1	3
E	G	F	eth1	2
F	G	F	eth1	1

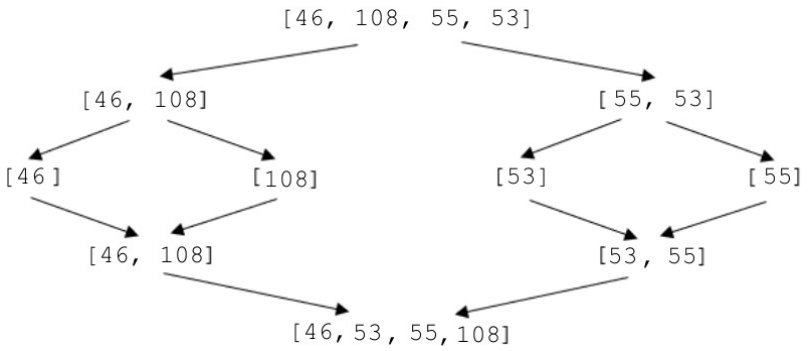
Q2

```
def calcul montant(prix_TF, quantite_TF, prix_TC, quantite_TC):
    """ Renvoie le montant de la commande
    Entrées :
            prx_TF : prix de ToutFruit
            quantite_TF : quantite de ToutFruit
            prix_TC : prix de ToutChoc
            quantite_TC : quantite de ToutChoc
    Sortie :
            montant de la commande apres reduction eventuelle
    """
    montant = quantite_TF * prix_IF + quantite_TC * prix_IC

    if montant >= 100 and montant < 200 :
            montant = montant - montant*0.10
    elif montant >= 200 :
            montant = montant – montant*0.20

    return montant
```

Q3.a



Q3.b

```
while i1 < len(liste1) or i2 < len(liste2):
```

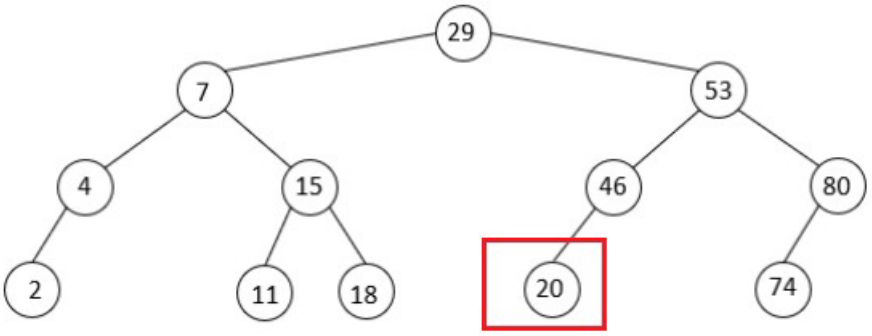
Q3.c

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

Exercice 3

Q1

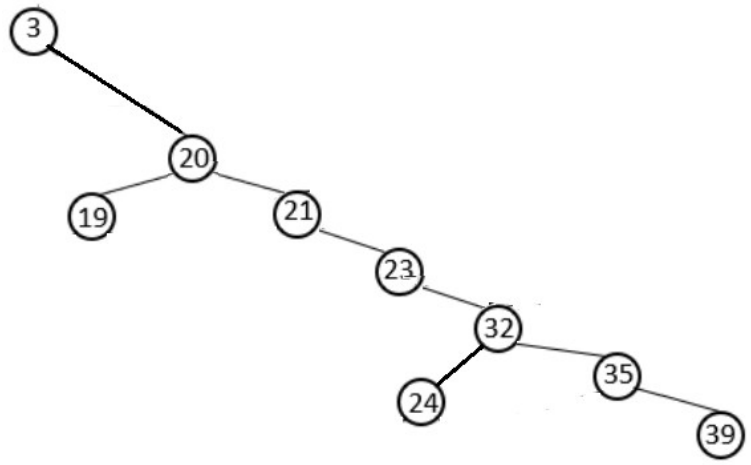
Arbre 3 : 20 < 29, doit être dans sous arbre gauche



Q2.a

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

Q2.b



Q2.c

- h(a1) = 4
- h(a2) = 7

Q2.d

```
def calculer hauteur(self):
    """ Renvoie la hauteur de l'arbre """
    if self.sa droit is None and self.sa_gauche is None:
            # l'arbre est réduit à une feuille
            return 1
    elif self.sa droit is None
            # arbre avec une racine et seulement un sous-arbre gauche
            return 1 + self.sa_gauche.calculer_hauteur()
    elif self.sa gauche is None:
            # arbre avec une racine et seulement un sous-arbre droit
            return 1 + self.sa droit.calculer hauteur()
    else:
            # arbre avec une racine et un sous-arbre droit et gauche
            return 1 + self.sa_gauche.calculer_hauteur() + self.sa droit.calculer hauteur()
```

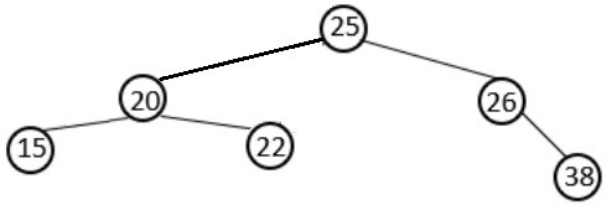
Q3.a

```
def rechercher_valeur(self, v):
    """ Renvoie True si la valeur v est trouvée dans l'ABR, False sinon """
    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 self.sa_gauche.rechercher_valeur(v) or self.sa_droit.rechercher_valeur(v)
```

Q3.b

4

Q4.a



Q4.b

```
def rotation_gauche(self):
    """ Renvoie une instance d'un ABR après une rotation gauche
    On suppose qu'il existe un sous-arbre droit"""
    pivot = self.sa_droit
    self.sa_droit = pivot.sa_gauche
    pivot.sa_gauche = self
    return ABR(pivot.valeur, pivot.sa_droit, pivot.sa_gauche)
```