Proposition de correction

Exercice 1

Q1.a

Clef primaire : nomSport

Clef étrangère : nomStation

Q1.b

- contrainte domaine : prix > 0, réel, non nulle
- contrainte relation : nomSport clef unique, non nulle
- contrainte référence : clef nomStation doit exister dans table Station

Q2.a

La clef « planche à voile » existe déjà

```
UPDATE Sport
SET prix = 1350
WHERE nomSport = 'planche à voile'
```

Q2.b

```
INSERT INTO Station VALUES('Soleil Rouge', 'Bastia', 'Corse') ;
INSERT INTO Sport VALUES('plongée', 'Soleil Rouge', 900) ;
```

Q3.a

```
SELECT mail
FROM Client
```

Q3.b

```
SELECT nomStation
FROM Sport
WHERE nomSport = 'plongée'
ORDER BY nomStation
```

Q4.a

```
SELECT Station.ville, Station.nomStation
```

```
FROM Station, Sport
WHERE Sport.nomSport = 'plongée'
AND Sport.nomStation = Station.nomStation
ORDER BY Station.ville, Station.nomStation
```

Q4.b

```
SELECT COUNT(*)
FROM Sejour, Station
WHERE Sejour.annee = 2020
AND Sejour.nomStation = Station.nomStation
AND Station.region = 'Corse'
```

Exercice 2

Q1

R2 → R1 → R4 → R7

R7 → R4 → R3 → R2

Q2.a

Isolation des sous réseaux R1, R2, R3 et R5, R6, R7

Il faut mailler les sous réseaux entre eux (créer de nouvelles routes)

Q3.a

Table de routage de R8		
Destination	Lien	Distance
R1	R2	2
R2	R2	1
R3	R2	2
R4	R6	2
R5	R6	2
R6	R6	1
R7	R6	2

Q3.b

Table de routage de R2		
Destination	Lien	Distance
R1	R2	1
R3	R2	1
R4	R1	2
R5	R3	3
R6	R8	2
R7	R3	3
R8	R8	1

Q4.a

$$C_{FE}=\frac{10^8}{BP_{FE}} \Leftrightarrow BP_{FE}=\frac{10^8}{1}=100\,Mb/s$$

$$C_E=\frac{10^8}{BP_E}=\frac{10^8}{10.10^6}=10\,Mb/s$$

Q4.b

R2 → R1 → R4 : C = 49 + 65 = 114

R2 → R3 → R4 : C = 65 + 10 = 75

R4 → R5 : C = 49

R4 → R6 → R5 : C = 10 + 10 = 20

R4 → R7 → R6 → R5 : C = 1 + 1 + 10 = 12

la route est donc R2 → R3 → R4 → R7 → R6 → R5 : C = 87

Exercice 3

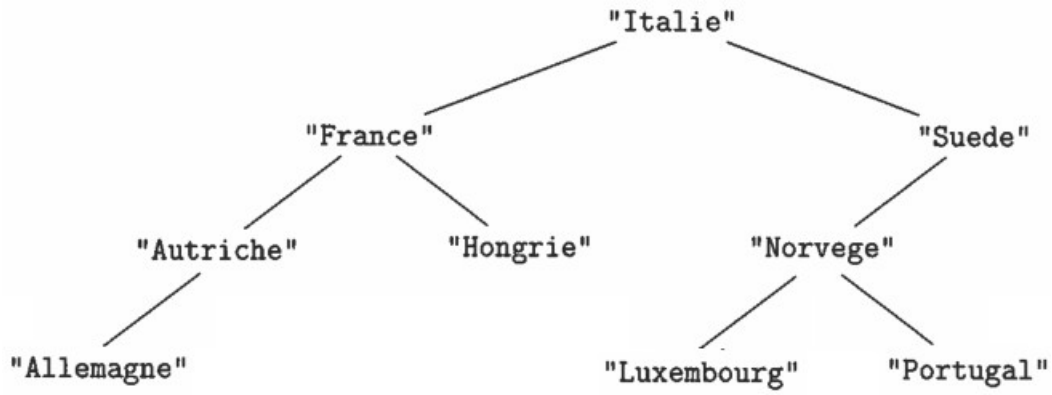
Q1.a

3

Q1.b

Vrai

Q1.c



Q2

Italie, France, Suede, Autriche, Hongrie, Norvege

Q3

```
def recherche(arb, val)
    """ recherche une valeur val dans un arbre arb.
    Renvoie  True si val est présente dans arb, et False sinon """
    if est_vide(arb) :
            return False
    if val == racine(abr) :
            return True
    if val < racine(arb) :
            recherche(gauche(abr), val)
    else :
            recherche(droite(abr), val)
```

Q4

```
def taille(abr : object) -> int :
    """ calcul la taille d'un arbre arb.
    NB : taille(racine) = 1 """
    if est_vide(abr) :
            return 0
    return 1 + taille(gauche(abr)) + taille(droite(abr))
```

Exercice 4

Q1.a

Proposition 3

Q1.b

- txt[0] : b
- txt[taille – 1] : r
- interieur : o

Q2

```
assert palindrome("BOB")                    # est un palindrome
assert palindrome("BOA") == False# n'est pas un palindrome
```

Q3

```
def palindrome(txt : str) -> bool :
    milieu = len(str) // 2
    for i in range(milieu) :
            if txt[i] != txt[len(txt) – 1 – i] :
                    return False
    return True
```

Q4.a

```
def complementaire(txt : str) -> str :
    lettre = "ATCG"
    complement = "TACG"
    chaine = ""
    for c in txt :
            if c in lettre :
                    chaine += complement[lettre.index(c)]
    return chaine
```

Q4.b

GATCGT → GTACGA

n'est pas palindromique

Q4.c

```
def est_palindromique(txt : str) -> bool :
    return palindrome(txt + complementaire(txt))
```

Exercice 5

Q1.a

Proposition 2

Q1.b

```
F = creer_file_vide()
enfiler(F, 15)
enfiler(F, 17)
enfiler(F, 14)
```

Q2

```
def longueur_file(F)
    """ File -> int """
    G = cree_file_vide()      # file temporaire
    n = 0                     #initialisation du nbr d'éléments
    while not est_vide(F) :
            G.enfiler(F.defiler())
            n += 1
    while not est_vide(G) :
            F.enfiler(G.defiler())
    return n
```

Q3

```
def variations(F)
    """ File -> tableau """
    taille = longueur_file(F)
    if taille == 1 :
            return []
    else :
            tab = [0 for k in range(taille – 1)]
            element1 = defiler(F)
            for i in range(taille – 1)
                    element2 = defiler(F)
                    tab[i] = element2 – element1
                    element1 = element2
            return tab
```

Q4

```
def nombre_baisses(tab : list) -> tuple :
    if len(tab) == 0 :
            return None
```

```
negative, minimum = 0, 0
for i in tab :
        if i < 0 :
                negative += 1
                if minimum > i :
                        minimum = i
return negative, minimum
```