Proposition de correction

Exercice 1

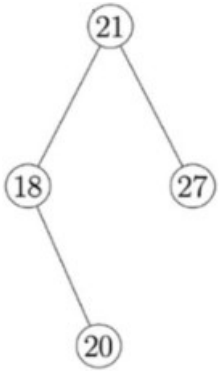
Q1.a

8

Q1.b

3

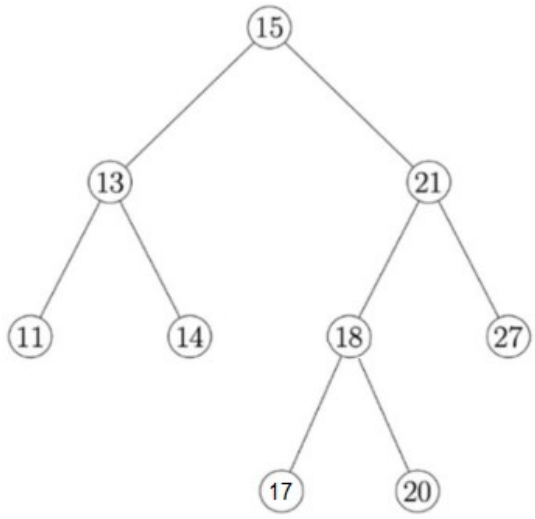
Q1.c



Q1.d

chaque nœud du sous-arbre *gauche* possède une clé ≤ à celle de sa racine, et chaque nœud du sous-arbre *droit* possède une clé ≥ à celle-ci.

Q1.e



Q2.a

C

Q2.b

```
def ins(v, abr):
    if abr is None:
            return Noeud(None, v, None)
    if v > abr.valeur:
            return Noeud(abr.gauche, abr.valeur, ins(v, abr.droit))
    elif v < abr.valeur:
            return Noeud(ins(v, abr.gauche), abr.valeur, abr.droit)
    else:
            return abr
```

Q3.a

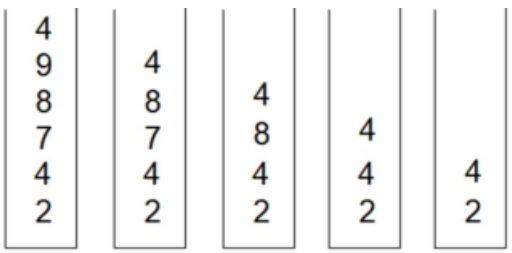
18

Q3.b

```
def nb_sup_optimise(v, abr):
    if abr is None:
        return 0
    else:
        if abr.valeur >= v:
            return 1 + nb_sup_optimise(v, abr.gauche) + nb_sup_optimise(v, abr.droit)
        else:
            return nb_sup_optimise(v, abr.droit)
```

Exercice 2

Q1.a



Q1.b

Pile B

Q2

```
def reduire_triplet_au_sommet(p):
    a = depiler(p)
    b = depiler(p)
    c = sommet(p)
    if a % 2 != c % 2 :
            empiler(p, b)
    empiler(p, a)
```

Q3.a

3

Q3.b

```
def parcourir_pile_en_reduisant(p):
    q = creer_pile_vide()
    while taille(p) >= 3:
            reduire_triplet_au_sommet(p)
            e = depiler(p)
            empiler(q, e)
    while not est_vide(q):
            e = depiler(q)
            empiler(p, e)
    return p
```

Q4

```
def jouer(p):
    q = parcourir_pile_en_reduisant(p)
    if q == p :
            return p
    else:
            return jouer(q)
```

Exercice 3

Q1.a

192.168.1.0

Q1.b

192.168.1.255

Q1.c

256 – 2 = 254

Q1.d

192.168.1.2

Q2.a

A → B → C → E → D

A → B → C → F → D

A → C → E → D

A → C → F → D

A → E → D

Q2.b

alternative si un routeur est HS ou saturé

Q3.a

Routeur A	
Destination	passe par
B	B
C	C
D	E
E	E
F	C

Q3.b

B → C → E → D

Q3.c

Routeur A	
Destination	passe par
B	B
C	C
D	C
E	C
F	C

Routeur B	
Destination	passe par
A	A
C	A
D	A
E	A
F	A

Routeur C	
Destination	passe par
A	A
B	A
D	E
E	E
F	F

Q3.d

B → A → C → E → D

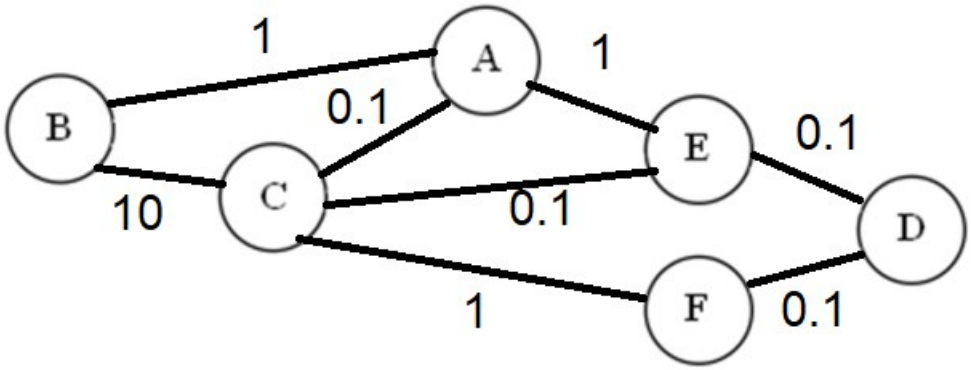
Q4.a

Ethernet (d = 10^7 bit/s) : 10

Fast-Ethernet (d = 10^8 bit/s) : 1

Fibre (d = 10^9 bit/s) : 0,1

Q4.b



Q4.c

B → A → E → D, coût = 2,1

B → A → E → C → F → D, coût = 3,2

B → A → C → F → D, coût = 2,2

B → C → A → E → D, coût = 11,2

B → C → E → D, coût =10,2

B → C → F → D, coût = 11,1

Q4.d

Protocole OSPF : coût min

B → A → E → D

Exercice 4

Q1.a

Hey Jude

I Want To hold Your Hand

Q1.b

SELECT nom

FROM interpretes

WHERE pays = 'Angleterre'

ORDER BY nom

Q1.c

I Want To hold Your Hand, 1963

Like a Rolling Stone, 1965

Respect, 1967

Hey Jude, 1968

Imagine, 1970

Smells Like Teen Spirit, 1991

Q1.d

SELECT COUNT(*)

FROM morceaux

Q1.e

SELECT titre

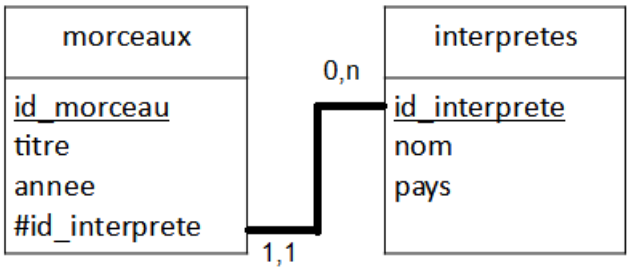
FROM moceaux

ORDER BY titre

Q2.a

id_interprete : identifie l'interprète de la chanson dans la table interpretes

Q2.b



Q2.c

La clef primaire 1 existe déjà dans la table

Q3.a

UPDATE morceaux

SET annee = 1971

WHERE titre = 'Imagine'

Q3.b

INSERT INTO interpretes

VALUES(6, 'The Who', 'Angleterre')

Q3.c

INSERT INTO morceaux

VALUES(7, 'My Generation', 1965, 6)

Q4

SELECT morceaux.titre

FROM morceaux, interpretes

WHERE interpretes.id_interprete = morceaux.id_interprete

AND morceaux.pays = 'États-Unis'

ORDER BY morceaux.titre

Exercice 5

Q1

```
cellule = Cellule(True, False, True, True)
```

Q2

```
class Labyrinthe:
    def __init__(self, hauteur, longueur):
            self.grille=self.construire_grille(hauteur, longueur)
    def construire_grille(self, hauteur, longueur):
            grille = []
            for i in range(hauteur):
                    ligne = []
                    for j in range(longueur):
                            cellule = Cellule(True, True, True, True)
                            ligne.append(cellule)
            grille.append(ligne)
    return grille
```

Q3

```
cellule2.murs['S'] = False
```

Q4

```
if c1_lig == c2_lig and c1_col - c2_col == 1:
    cellule1.murs['O'] = False
    cellule2.murs['E']  = False
```

Q5

```
def creer_labyrinthe(self, ligne, colonne, haut, long):
    if haut == 1 :      # Cas de base
            for k in range(long):
                    self.creer_passage(ligne, k, ligne, k+1)
    elif long == 1:    # Cas de base
            for k in range(haut):
                    self.creer_passage(k, colonne, k+1, colonne)
    else:     # Appels récursifs
    # Code non étudié (Ne pas compléter)
```

Q6