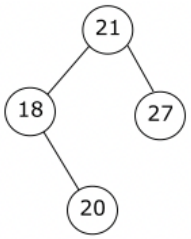
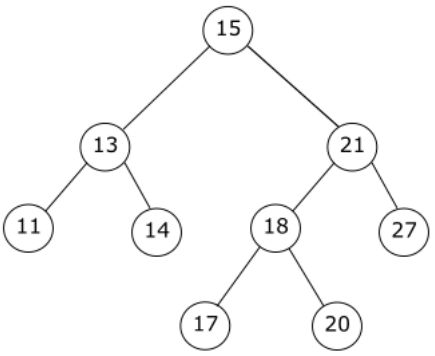
Éléments de correction sujet 03 (2022)

Exercice 1

1.
   a. la taille de l'arbre est 8
   b. la hauteur de l'arbre est 4
   c.



   d. si on effectue un parcours infixe de l'arbre (11-13-14-15-18-20-21-27), on obtient les valeurs dans l'ordre croissantes, nous avons donc bien un arbre binaire de recherche.
   e.



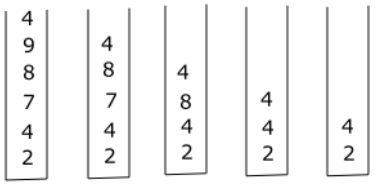
2.
   a. (C) `abr=Noeud(Noeud(None,13,None),15,Noeud(None,21,None))`
   b.

```
Noeud(ins(v,abr.gauche),abr.valeur,abr.droit)
```

3.
   a. En comptant l'appel initial, nous avons 17 appels à la fonction nb_sup
   b.

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

Exercice 2

1.
   a.
   
   b.
   La pile gagnante est la pile B
2.

```
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)
```

3.
   a. La taille minimum d'une pile pour être réductible est 3.
   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
```

4.

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

Exercice 3

1.
   a. 192.168.1.0
   b. 192.168.1.255
   c. il est possible de connecter 254 machines (256-2)
   d. 192.168.1.2
2.
   a.
      SW1 → Routeur A → Routeur E → Routeur D → SW4
      SW1 → Routeur A → Routeur E → Routeur C → Routeur F → Routeur D → SW4
      SW1 → Routeur A → Routeur C → Routeur F → Routeur D → SW4
      SW1 → Routeur A → Routeur C → Routeur E → Routeur D → SW4
      SW1 → Routeur A → Routeur B → Routeur C → Routeur F → Routeur D → SW4
      SW1 → Routeur A → Routeur B → Routeur C → Routeur E → Routeur D → SW4
   b.
      Il est utile d'avoir plusieurs routes possibles reliant 2 réseaux, car en cas de panne d'un routeur, le paquet de données pourra emprunter un autre chemin qui évitera le routeur en panne.
3.
   a.

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

   b.
      Routeur B → Routeur C → Routeur E → Routeur D
   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

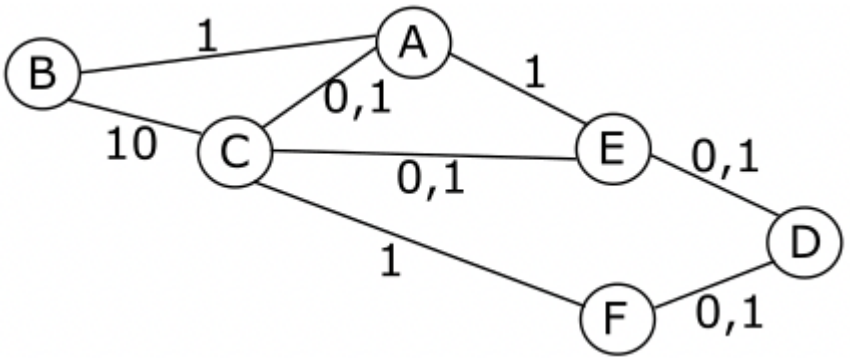
d.

Routeur B → Routeur A → Routeur C → Routeur E → Routeur D

4.

a. Ethernet coût = 10 ; Fast-Ethernet coût = 1 ; Fibre coût = 0,1

b.



c.

SW2 → Routeur B → Routeur A → Routeur E → Routeur D → SW4 ; coût = 2,1
SW2 → Routeur B → Routeur A → Routeur E → Routeur C → Routeur F → Routeur D → SW4 ; coût = 3,2
SW2 → Routeur B → Routeur A → Routeur C → Routeur E → Routeur D → SW4 ; coût = 1,3
SW2 → Routeur B → Routeur A → Routeur C → Routeur F → Routeur D → SW4 ; coût = 2,2
SW2 → Routeur B → Routeur C → Routeur A → Routeur E → Routeur D → SW4 ; coût = 11,2
SW2 → Routeur B → Routeur C → Routeur E → Routeur D → SW4 ; coût = 10,2
SW2 → Routeur B → Routeur C → Routeur F → Routeur D → SW4 ; coût = 11,1

d.

Le chemin choisit sera : SW2 → Routeur B → Routeur A → Routeur C → Routeur E → Routeur D → SW4 car cette route a le plus faible coût (1,3)

Exercice 4

1.
    a.
    Résultats obtenus : Hey Jude et I Want To hold Your Hand
    b.

```
SELECT nom
FROM interpretes
WHERE pays = 'Angleterre'
```

    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

    d.

```
SELECT COUNT(*)
FROM morceaux
```

    e.

```
SELECT titre
FROM morceaux
ORDER BY titre
```

2.
    a.
    La clé étrangère de la table morceaux est *id_interprete*, car cet attribut permet d'établir un lien avec la table *interpretes* (il correspond à l'attribut *id_interprete* de la table *interpretes*.
    b.
    interpretes(id_interprete : INT, nom : TEXT, pays : TEXT)
    morceaux(id_morceau : INT, titre : TEXT, annee : INT, #id_interprete : INT)
    c.
    Cette requête provoque une erreur, car elle essaye d'ajouter à la table *interpretes* une entrée ayant pour *id_interprete* 1. Or, l'attribut *id_interprete* (qui est une clé primaire) a déjà une entrée avec la valeur 1 (la clé primaire doit être unique).
3.
    a.

```
UPDATE morceaux
SET annee = 1971
WHERE titre = 'Imagine'
```

b.

```
INSERT INTO interpretes
(id_interprete, nom, pays)
VALUES
(6, 'The Who', 'Angleterre')
```

c.

```
INSERT INTO morceaux
(id_morceau, titre, annee, id_interprete)
VALUES
(7, 'My Generation', 1965, 6)
```

4.

```
SELECT titre
FROM morceaux
JOIN interpretes ON interpretes.id_interprete = morceaux.id_interprete
WHERE pays = 'États-Unis'
```

Exercice 5

1.

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

2.

```
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
```

3.

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

4.

```
def creer_passage(self, c1_lig, c1_col, c2_lig, c2_col):
    cellule1 = self.grille[c1_lig][c1_col]
    cellule2 = self.grille[c2_lig][c2_col]
    if c1_lig - c2_lig == 1 and c1_col == c2_col:
        cellule1.murs['N'] = False
        cellule2.murs['S'] = False
    # cellule2 à l'Ouest de cellule1
    elif c1_col - c2_col == 1 and c1_lig == c2_lig:
        cellule1.murs['O'] = False
        cellule2.murs['E'] = False
```

5.

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

6.