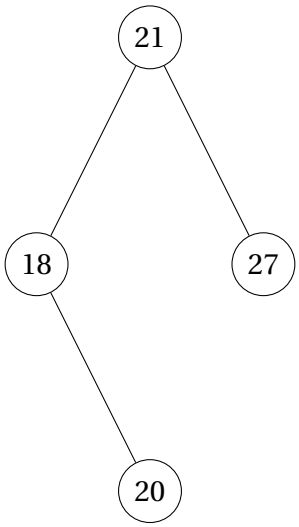
2022 Métropole sujet 2

Exercice 1

1.a. [0.25 point] La taille est 8.

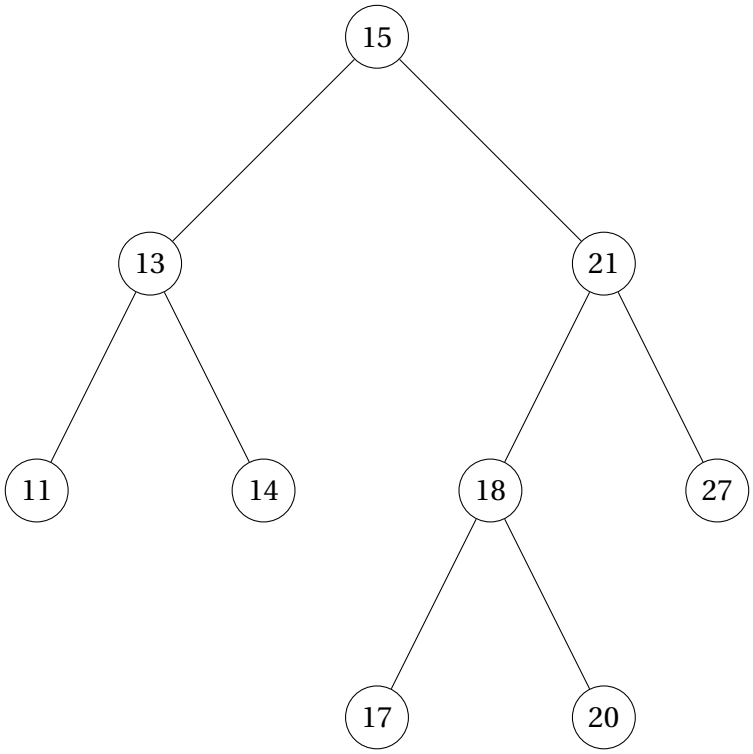
1.b. [0.25 point] La hauteur est 4.

1.c. [0.25 point]



1.d. [0.5 point] C'est un arbre binaire car chaque nœud possède au maximum deux fils. C'est un arbre binaire de recherche car la valeur d'un nœud est plus grande que les valeurs de son sous-arbre gauche et plus petite que les valeurs de son sous-arbre droit.

1.e. [0.5 point]



2.a. [0.5 point] Réponse (C).

2.b. [0.5 point] Noeud(ins(v,abr.gauche), abr.valeur, abr.droit).

3.a. [0.5 point] `nb_sup` est appelé 17 fois si on compte l'appel initial.

3.b. [0.75 point]

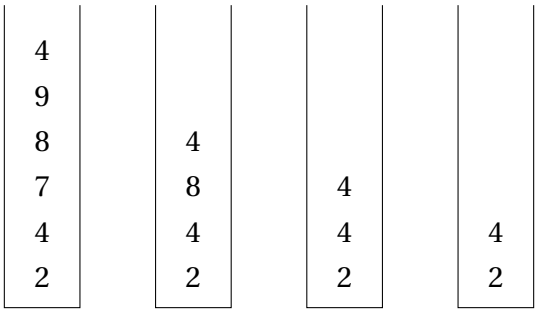
```
def nb_sup (v, abr):
    if abr is None:
        return 0
    else:
        if abr.valeur >= v:
            return 1 + nb_sup(v,abr.gauche) + nb_sup(v,abr.droit)
        else:
            return nb_sup(v,abr.droit)
```

En étant plus fin, on peut également proposer :

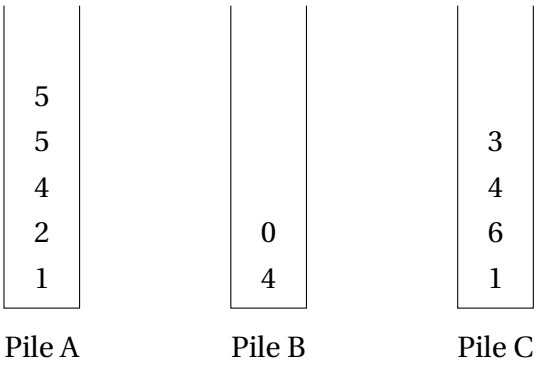
```
def nb_sup (v, abr):
    if abr is None:
        return 0
    else:
        if abr.valeur >= v:
            return 1 + nb_sup(v,abr.gauche) + nb_sup(v,abr.droit)
        elif abr.valeur == v:
            return 1 + nb_sup(v,abr.droit)
        else:
            return nb_sup(v,abr.droit)
```

Exercice 2

1.a. [0.75 point] Voici les étapes après chaque parcourt :



1.b. [0.5 point] Voici les états finaux de chacune des piles :



La pile gagnante est la B.

2. [1 point]

```
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. [0.5 point] La taille minimale est 3.

3.b. [0.75 point]

```
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. [0.5 point] Voici la réponse attendue :

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

Mais cela ne fonctionne pas à cause des effets de bord de la fonction `reduire_triplet_au_sommet` car `p` y est modifiée. Pour qu'elle fonctionne il faut mémoriser la taille de `p` :

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

Exercice 3

1.a. [0.25 point] 192.168.1.0

1.b. [0.25 point] 192.168.1.255

1.c. [0.25 point] On peut connecter $256 - 2 = 254$ machines sur ce réseaux.

1.d. [0.25 point] 192.168.1.100. (Les adresses finissant par 0, 1, 3 ou 255 étant déjà prises ou réservées)

2.a. [0.25 point]
- A-E-D
- A-E-C-F-D
- A-B-C-E-D
- A-B-C-F-D
- A-C-E-D
- A-C-F-D

2.b. [0.25 point] Cela entraîne une redondance et donc une meilleure résilience face aux pannes.

3.a. [0.25 point]

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

3.b. [0.25 point] B-C-E-D

3.c. [0.5 point]

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

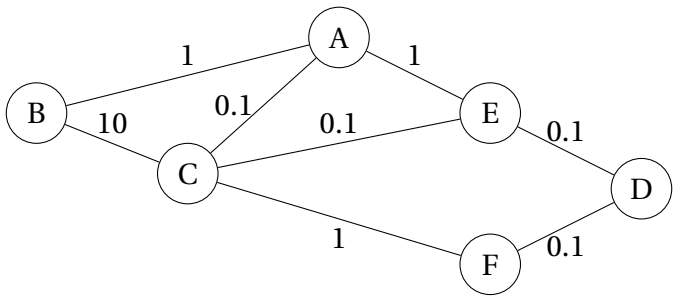
Routeur C	
Destination	passe par
A	A
B	A
D	E ou F
E	E
F	F

3.d. [0.25 point] B-A-C-E-D ou B-A-C-F-D

4.a. [0.25 point]

— Ethernet : $\frac{10^8}{10^7} = 10$;
— Fast-Ethernet : $\frac{10^8}{10^8} = 1$;
— Fibre : $\frac{10^8}{10^9} = 0.1$;

4.b. [0.25 point]



4.b. [0.5 point] On calcule le coût de toutes les routes entre B et D :

— B-A-E-D : 2.1
— B-A-E-C-F-D : 3.2
— B-A-C-E-D : 1.3
— B-A-C-F-D : 2.2
— B-C-A-E-D : 11.2
— B-C-E-D : 10.2
— B-C-F-D : 11.1

4.c. [0.25 point] OSPF minimise la somme des coûts. Cela correspond à la route B-A-C-E-D avec un coût de 1.3.

Exercice 4

1.a. [0.25 point]

Titre
Hey Jude
I Want To Hold Your Hand

1.b. [0.25 point] `SELECT nom FROM interpretes WHERE pays = "Angleterre";`

1.c. [0.25 point]

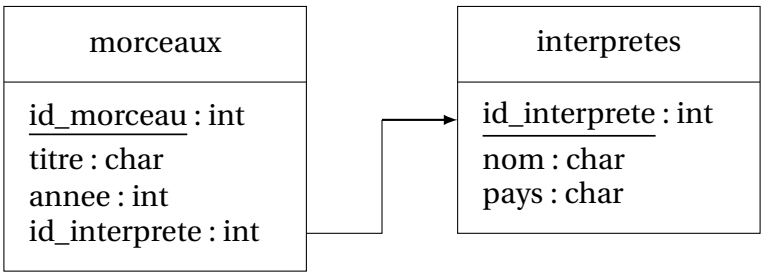
Titre	annee
I Want To Hold Your Hand	1963
Like A Rolling Stone	1965
Respect	1967
Hey Jude	1968
Imagine	1971
Smell Like Teen Spirit	1991

1.d. [0.25 point] `SELECT COUNT(*) FROM morceaux;`

1.e. [0.25 point] `SELECT titre FROM morceaux ORDER BY titre;`

2.a. [0.5 point] La clé étrangère de la table morceaux est `id_interprete` car elle fait référence à la clé primaire de la table `interpretes`.

2.b. [0.5 point] Les clés primaires sont soulignées, les flèches indiquent le lien d'une clé étrangère vers une clé primaire.



2.c. [0.5 point] On essaie d'insérer une nouvelle ligne avec la valeur 1 comme clé primaire. Or cette valeur est déjà utilisée et une clé primaire ne peut pas avoir deux fois la même valeur. Il va donc y avoir une erreur de clés dupliquées.

3.a. [0.25 point] `UPDATE morceaux SET annee = 1971 WHERE id_morceau = 3;`

3.b. [0.25 point] `INSERT INTO interpretes VALUES (6, 'The Who', 'Angleterre');`

3.c. [0.25 point] `INSERT INTO morceaux VALUES (7, 'My Generation', 1965, 6);`

4. [0.5 point]

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

Exercice 5

1. [0.25 point] `cellule = Cellule(True, False, True, True)`

2. [1 point]

```python
for i in range(hauteur):
    ligne = []
    for j in range(longueur):
        cellule = Cellule(True, True, True, True)
        ligne.append(cellule)
```

3. [0.25 point] `cellule2.murs['S'] = False`

4. [1 point]

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

5. [0.75 point]

```python
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)
```

6. [0.75 point]