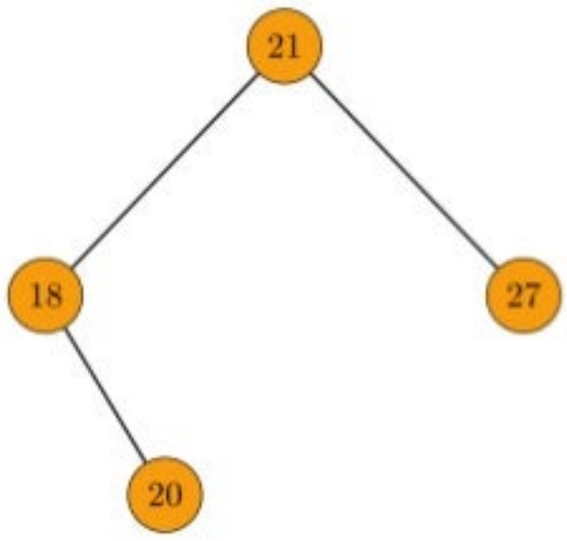
BAC GÉNÉRAL 2022
Épreuve de spécialité Numérique Sciences Informatiques (NSI)
Jeudi 12 mai 2022

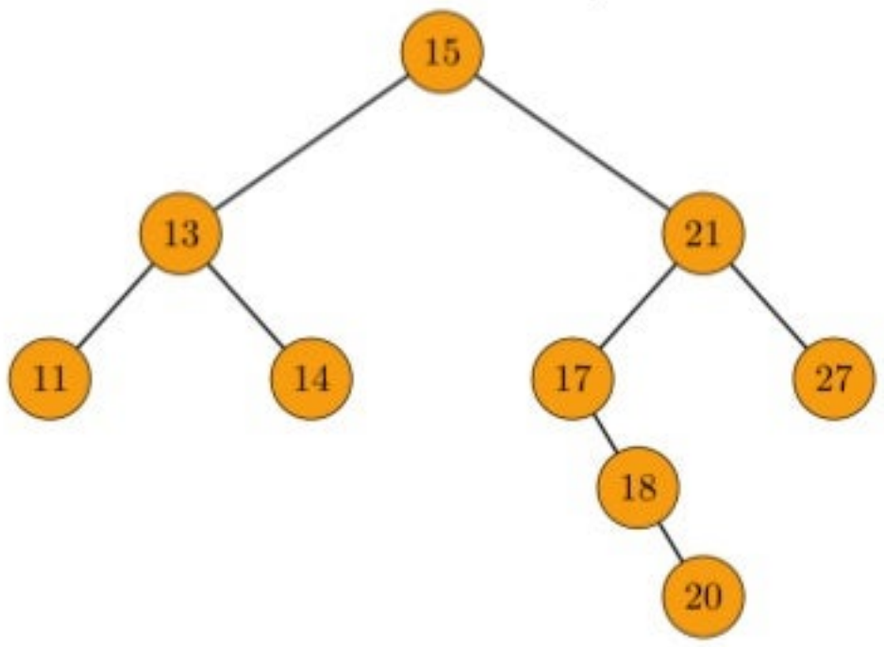
Exercice 1

1. **a.** Taille de l'arbre : 8 nœuds
   b. Hauteur de l'arbre : 4
   c. Sous-arbre droit du nœud de valeur 15 :



d. L'arbre est un ABR car, pour chaque nœud de l'arbre, chaque nœud de son SAG a une clé inférieure ou égale à la clé du nœud considéré, et chaque nœud de son SAD a une clé de valeur supérieure ou égale. Ainsi, pour le nœud de clé 15, toutes les clés de son SAG sont de valeur inférieure (11, 13, 14) et toutes les clés de son SAD sont de valeur supérieure (18, 20, 21, 27). Ce raisonnement peut être réitéré pour n'importe quel autre nœud.

e. Arbre de recherche binaire après inclusion de la clé 17 :



2. **a.** C'est la proposition C qui permet d'instancier l'ABR proposé.
   b. Code de la ligne 7 :

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

3. **a.** Il y aura au total 17 appels à la fonction `nb_sup`, soit 2 x taille + 1 car il y a un appel initial sur l'arbre et ensuite systématiquement 1 appel pour chacun des 2 sous-arbres (même vides) de chacun des nœuds de l'arbre.

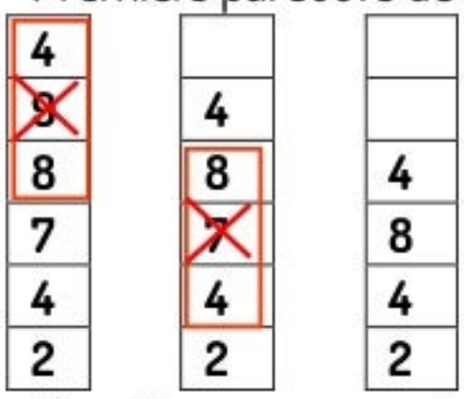
   b. L'arbre étant un ABR, il n'est pas nécessaire d'appeler la fonction sur un sous-arbre lorsque sa clé est inférieure car toutes les clés du sous-arbre seront nécessairement inférieures également. On modifie donc uniquement la ligne 8 en remplaçant la valeur retournée initialement dans le code par la valeur 0 (indentation inchangée) :

```
    return 0
```

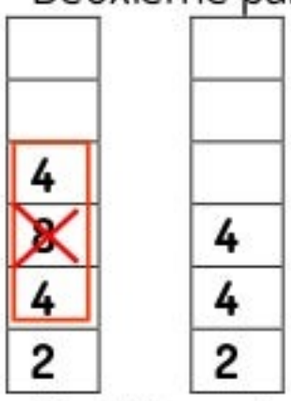
Exercice 2

1. Étapes de la réduction de la pile :

Premiers parcours de la pile :



Deuxième parcours de la pile :



Troisième et dernier parcours de la pile :

2. La pile B est gagnante, elle termine par l'empilement de 4 au-dessus de 0.

3. Code de la fonction `reduire_triplet_au_sommet` :

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

3. a. La pile doit avoir une taille minimale de 3 éléments pour être réductible.

b. Code de la fonction `parcourir_pile_en_reduisant` :

```
def parcourir_pile_en_reduisant(p):
q = créer_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. Code de la fonction `jouer` :

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

Exercice 3

1. **a.** Adresse du réseau : 192.168.1.0 car le masque couvre les 24 bits de gauche (les 3 premiers octets) et les 8 bits à droite (le dernier octet) sont mis à zéro.
   b. L'adresse de diffusion (*broadcast*) est : 192.168.1.255.
   c. Il y a au total 2^8 = 256 adresses possibles dans ce réseau. Lorsqu'on retire les deux adresses décrites ci-dessus, il reste 256 – 2 = 254 machines adressables.
   d. On peut ajouter une machine d'adresse 192.168.1.2 par exemple (le dernier octet prend une valeur entre 0 et 255, sauf .0, .1, .3 et .255).
2. **a.** Les routes possibles sont :
   - A – E – D
   - A – E – C – F – D
   - A – B – C – E – D
   - A – B – C – F - D

   b. Le fait d'avoir plusieurs routes possibles entre deux réseaux permet de maintenir l'existence d'au moins une route dans le cas d'une panne d'un routeur ou d'une coupure d'une liaison entre deux routeurs.
3. **a.** Table de routage du routeur A :

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

   b. Route de B à D en utilisant les tables : B – C – E – D.
   c. Nouvelles tables de routage pour A, B et C :

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

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

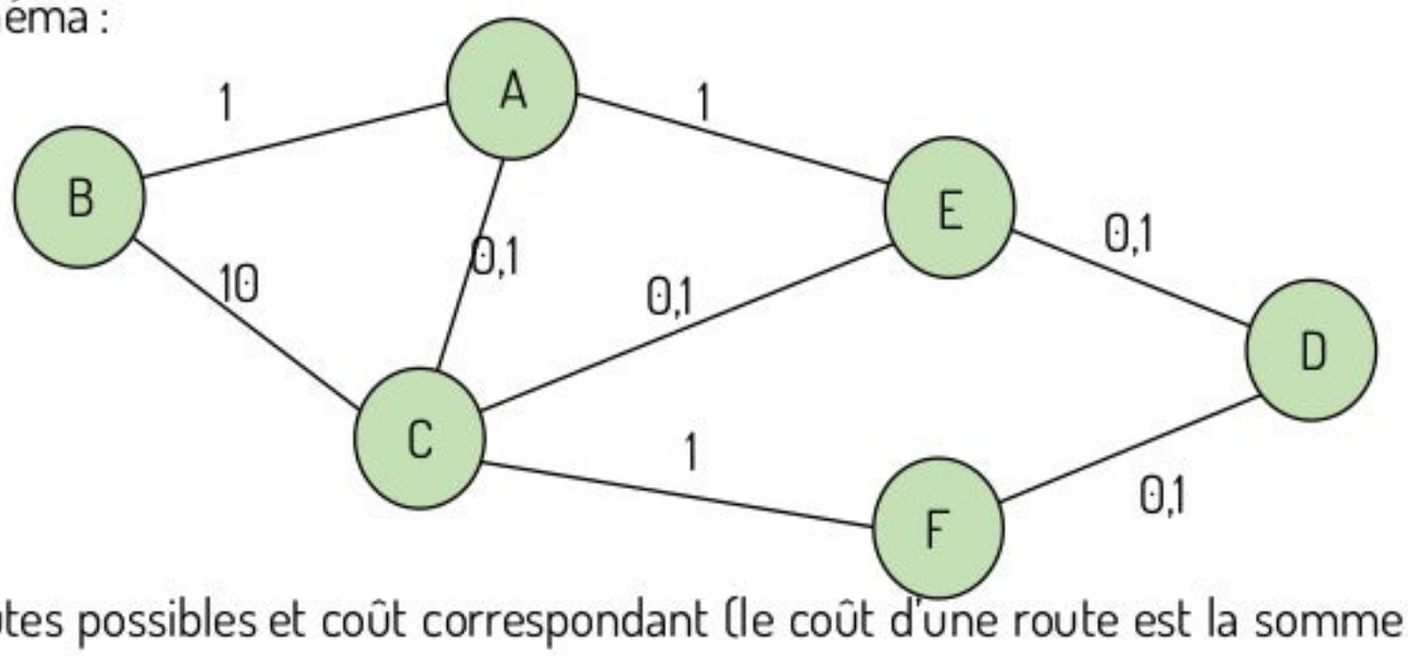
Routeur C	
Destination	passe par
A	B
B	A
D	E
E	E
F	C

   d. La nouvelle route sera : B – A – C – E – D.

4. **a.** Coût des liaisons :

Liaison	Débit (bit/s)	Coût
Ethernet	10^7	10
Fast-Ethernet	10^8	1
Fibre	10^9	0,1

b. Schéma :



c. Routes possibles et coût correspondant (le coût d'une route est la somme des coûts des liaisons entre routeurs traversés) :

- B – C – A – E – D : 11,2
- B – C – E – D : 10,2
- B – C – F – D : 11,1
- B – A – C – E – D : 1,3
- B – A – C – F – D : 2,2
- B – A – E – C – F – D : 3,2
- B – A – E – D : 2,1

d. La route qui sera empruntée (entre les routeurs B et D) sera celle qui minimise le coût estimé, soit le chemin le plus court B – A – C – E – D, de coût 1,3.

Exercice 4

1. **a.** Résultat :

Hey Jude
I Want To Hold Your Hand

b. Requête :

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

c. Résultat :

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. Requête :

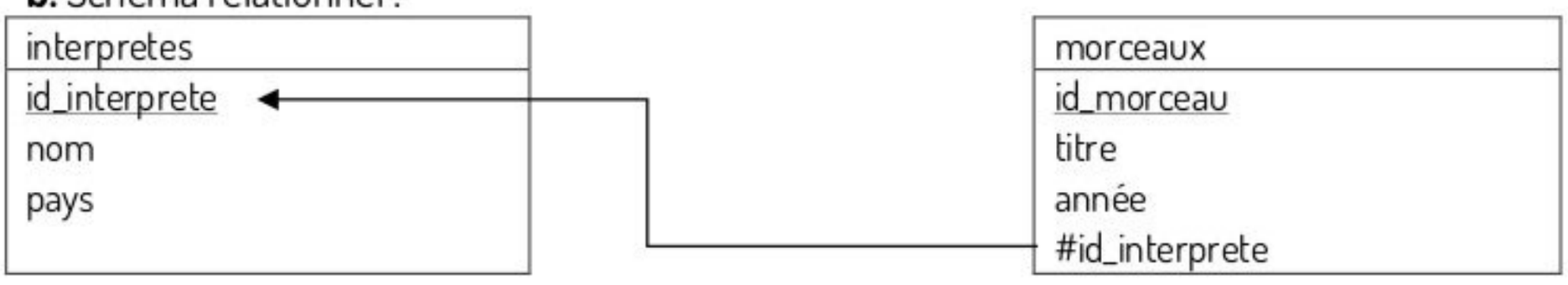
```
SELECT COUNT(*)
FROM morceaux;
```

e. Requête :

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

2. **a.** La clé étrangère de la table morceaux est `id_interprete` car elle fait référence à la clé primaire d'une autre table (`interpretes`).

b. Schéma relationnel :



c. La requête d'insertion produit une erreur car elle conduirait à créer un nouvel enregistrement dont l'attribut `id_interprete` aurait la valeur `1`, ce qui serait un doublon (ce qui impossible car cet attribut est une clé primaire de la table et doit donc être unique).

3. **a.** Requête de mise à jour :

```
UPDATE morceaux
SET annee = 1971
WHERE id_morceau = 3;
```

b. Requête d'insertion :

```
INSERT INTO interpretes
VALUES (6, 'The Who', 'Angleterre');
```

c. Requête d'insertion :

```
INSERT INTO interpretes
VALUES (7, 'My Generation', 1965, 6);
```

4. Requête listant les interprètes venant des États-Unis :

```sql
SELECT titre
FROM morceaux
JOIN interpretes
     ON morceaux.id_interprete = interpretes.id_interprete
WHERE interpretes.pays = 'Angleterre';
```

Exercice 5

1. Création d'une instance de cellule possédant tous ses murs sauf le mur Est :

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

2. Lignes 6 à 10 de la classe `Labyrinthe` :

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

3. Instruction à ajouter ligne 19 (même indentation que la ligne 18) :

```
cellule2.murs('S') = False
```

4. Lignes 21 à 23 :

```
elif c1_col - c2.col == 1 and c1_lig == c2_lig:
    cellule1.murs('O') = False
    cellule2.murs('E') = False
```

5. Lignes 25 à 30 de la méthode `creer_labyrinthe` :

```
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)
```

6. Dessin du labyrinthe obtenu :