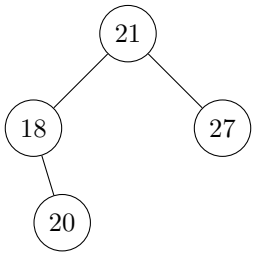
# Métropole - mai 2022 - Sujet 2 (corrigé)

**Exercice 1 (Arbres binaires de recherche, programmation orientée objet et récursivité)**

1. (a) La taille de l'arbre est 8 (il y a 8 nœuds).

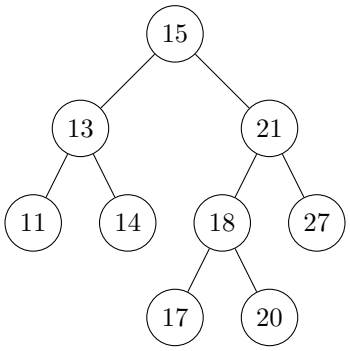
   (b) La hauteur de cet arbre est 4.

   (c) Le sous-arbre droit du nœud de valeur 15 est :




   (d) L'arbre de la figure 1 est bien un arbre binaire de recherche car les valeurs de tous les nœuds des sous-arbres gauches sont strictement inférieures à la valeur de la racine et les valeurs de tous les nœuds des sous-arbres droits sont strictement supérieures à la valeur de la racine.

   (e) Après l'insertion de la valeur 17 dans l'arbre de la figure 1 on obtient l'arbre suivant :




2. (a) L'instruction C permet d'instancier un objet représentant l'arbre demander.

   (b) On complète la ligne 7 de la façon suivante :

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

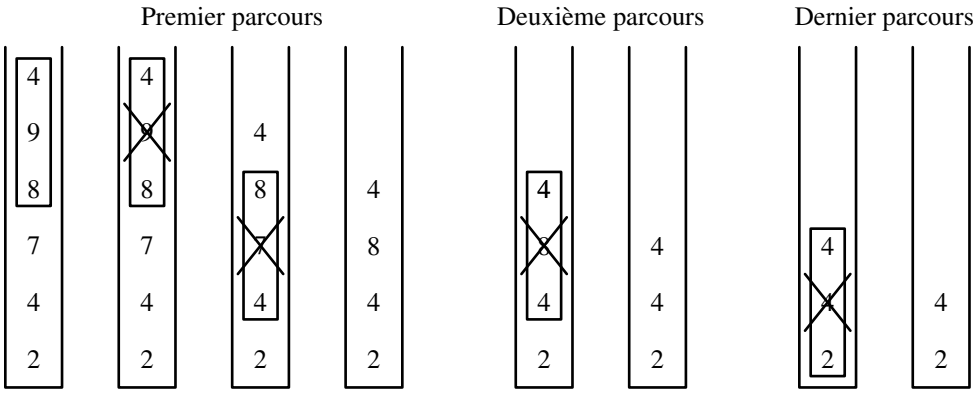
3. (a) A chaque appel sur un arbre non vide on effectue deux appels, comme l'arbre est de taille 8 il y a 16 sous appels. En comptant l'appel initial, il y a donc 17 appels à la fonction `nb_sup`.

   (b) Le code suivant convient :

```
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(abr.droit)
```

**Exercice 2 (Structure de données)**

1. (a) Voici les différentes étapes de réduction de la pile :




   (b) La pile B est la pile gagnante.

2. On complète les lignes 5 à 7 de la façons suivante :

```
if a % 2 != c % 2:
    empliler(p, b)
empiler(p, a)
```

3. (a) La taille minimale que doit avoir une pile pour être réductible est 3.

   (b) On complète la ligne 3 avec : `while taille(p) >= 3:`
   et les lignes 8 et 9 par :

```
e = depiler(q)
epiler(p, e)
```

4. Le code suivant convient :

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

**Exercice 3 (Réseaux et protocole de routage)**

1. (a) La machine se trouve dans le réseau $192.168.1.0/24$.
   (b) L'adresse de diffusion (ou de broadcast) de ce réseau est $192.168.1.255/24$.
   (c) Comme sur 8 bits ($32 - 24 = 8$) on code les 256 entiers de 0 à 255 et que les entiers 0 et 255 sont réservés à l'adresse de réseau et l'adresse de diffusion, le nombre maximal de machines que l'on peut connecter sur le réseau est 254.
   (d) On peut utiliser l'adresse $192.168.1.2$ par exemple qui n'est ni réservée, ni déjà utilisée par une machine du réseau de la figure 1.
2. (a) Il y a 6 routes possibles (on peut dessiner un arbre pour les construire et les dénombrer), il s'agit de : A-B-C-E-D, A-B-C-F-D, A-C-E-D, A-C-F-D, A-E-C-F-D et A-E-D
   (b) Il est utile d'avoir plusieurs routes en cas de panne d'un routeur ou d'une rupture d'une ligne.
3. (a) Voici la table de routage du routeur A complétée :

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

   (b) On consulte la table de routage de B pour aller vers D : celle-ci nous indique qu'il faut passer par le routeur C. Dans la table de routage de C, pour aller vers D il faut passer par E. Enfin, la table de routage de E nous indique que E est directement relié à D. Finalement la route sera : B-C-E-D.
   (c) On obtient les nouvelles tables de routage suivantes :

Routeur A

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

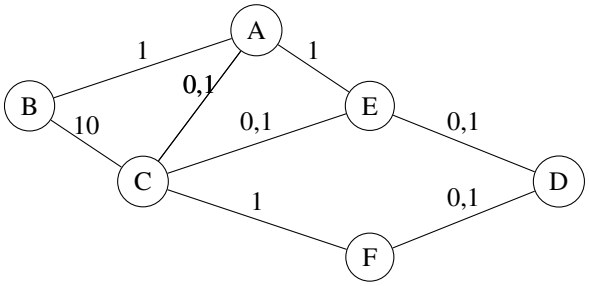
Routeur B

| 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) La nouvelle route empruntée sera : B-A-C-E-D
4. (a) Le coût d'une liaison Ethernet est 1°, celui d'une Fast-Ethernet est 1 et celui de la Fibre est 0,1.
   (b) On obtient le graphe suivant représentant le réseau de la figure 1 :




   (c) Il y 7 routes différentes : B-A-C-E-D (coût 1,3), B-A-C-F-D (coût 2,2), B-A-E-D (coût 2,1), B-A-E-C-F-D (coût 3,2), B-C-A-E-D (coût 11,2), B-C-E-D (coût 10,2), B-C-F-D (coût 11,1).
   (d) La route sélectionnée par le protocole OSPF est celle dont le coût est le plus faible. La route choisie sera donc : B-A-C-E-D.

**Exercice 4 (Base de données et langage SQL)**

**Partie A : Parcours d'un arbre**

1. (a) Le résultat de la requête est :

```
Hey Jude
I Want To hold Your Hand
```

   (b) La requête suivante convient :

```
SELECT nom FROM interpretes WHERE pays = "Angleterre";
```

   (c) Le résultat de la requête est :

```
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) La requête suivante convient :

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

   (e) La requête suivante convient :

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

2. (a) La clé étrangère de la relation `morceaux` est `id_interprete` car cet attribut fait référence à la clé primaire de la relation `interpretes`.

   (b) Le schéma relationnel des tables `morceaux` et `interpretes` est le suivant :

```
morceau(id_morceau, titre, annee, #id_interprete)
interpretes(id_interprete, nom, pays)
```

   Les clés primaires sont soulignées et les clés étrangères sont précédées d'un `#`.

   (c) La requête produit une erreur car la valeur `1` pour l'attribut `id_interprete` est déjà associée à l'interprète nommé Bob Dylan et les valeurs d'une clé primaire sont uniques.

3. (a) La requête suivante convient :

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

   (b) La requête suivante convient :

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

   (c) La requête suivante convient :

```
INSERT INTO morceaux VALUES (7, "My Generation", 1965, 6);
```

4. On utilise une jointure pour effectuer la requête demandée :

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

**Exercice 5 (Programmation orientée objet et méthode diviser pour régner)**

1. L'instruction suivante permet de créer l'instance demandée : `Cellule(True, False, True, True)`.
2. On complète les lignes 6 à 10 de la façon suivante :

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

3. L'instruction à ajouter à la ligne 19 est : `cellule2.murs['S'] = False`.
4. Les lignes 21 à 23 peuvent être complétées de la façons suivante :

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

5. Voici le code complété :

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

6. A la fin de l'exécution de l'algorithme, voici le labyrinthe obtenu :