23NSIJ2ME1

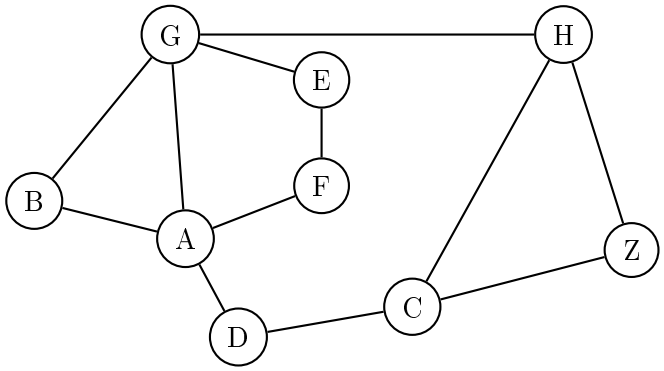
Exercice 1

1. Bien que le sujet demande d'étonnantes *tables de coût*, on donne les tables de **routage** des routeurs B et F, avec la colonne Lien.

Table de **routage** de B		
Destination	Lien	Coût
A	A	1
C	A	3
D	A	2
E	G	2
F	A	2
G	G	1
H	G	2

Table de **routage** de F		
Destination	Lien	Coût
A	A	1
B	A	2
C	A	3
D	A	2
E	E	1
G	E	2
H	E	3

   Rappelons que lorsque plusieurs routes existent, une seule figure dans les tables de routage ; celle qui est choisie dépend de l'historique des échanges de tables de routage entre les routeurs du réseau.
2. Les chemins possibles de F à H sont F-A-G-H et F-E-G-H.
3. La table fournie par l'énoncé montre que Z est relié directement à C ainsi qu'à H, ce qui permet de tracer le réseau après l'ajout de ce noeud.



4. Avec le protocole OSPF, le chemin de B à H est B-G-E-F-A-D-C-H, de coût total $1+1+1+1+10+10+10=34$.

   On peut le justifier en remarquant que pour arriver e H il faut passer par G ou bien par C, et passer par G impliquerait un coût total supérieur à 100 ; par conséquent le chemin optimal doit passer par C. En remontant vers le point de départ B, on rencontre D et A ; et pour arriver en A, le lien B-A de est de coût 10, passer par G, E et F est moins coûteux.

   On peut aussi utiliser l'algorithme de Dijkstra de recherche de plus court chemin dans un graphe pondéré.

Exercice 2

1. (a) Une clé primaire est un champ qui est unique à chaque ligne d'une table et qui permet dont d'identifier un enregistrement sans ambiguïté.

   (b) La requête `INSERT INTO Astronaute VALUES (3,'HAIGNERE','Claudie','français',3);` car la valeur 3 est déjà utilisée pour la clé primaire `id_astronaute` pou l'enregistrement de Mathias Maurer. L'erreur est une violation de contrainte d'intégrité.

   (c) Le schéma relationnel de la table `Fusee` est

   `Fusee(id_fusee : INT, modele : TEXT, constructeur : TEXT, nb_places : INT)`.
2. (a) La requête `SELECT COUNT(*) FROM Fusee WHERE constructeur = 'SpaceX' ;` renvoie 2, le nombre de fusées construites par SpaceX.

(b) La requête `SELECT modele, constructeur FROM Fusee WHERE nb_places >=4 ;` renvoie le modèle et le constructeur des fusées ayant au moins 4 places.

(c) La requête `SELECT nom, prenom FROM Astronaute ORDER BY nom ;` renvoie les noms et prénoms des astronautes dans l'ordre alphabétique du nom.

3. (a) La mise à jour demandée se traduit par les requêtes :

```
INSERT INTO Vol VALUES (5,3,'12/04/2023') ;
INSERT INTO Equipe VALUES (5,1) ;
INSERT INTO Equipe VALUES (5,4) ;
```

(b) La requête

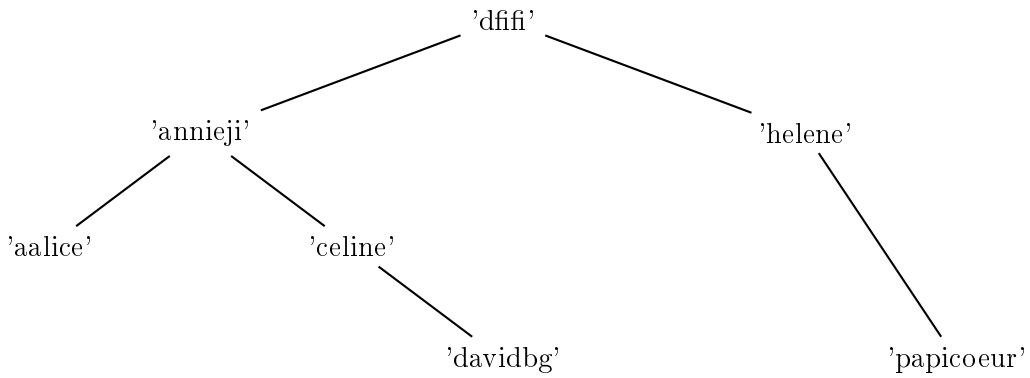
```
SELECT nom, prenom FROM Astronaute
JOIN Equipe ON Astronaute.id_astronaute = Equipe.id_astronaute
JOIN Vol ON Equipe.id_vol=Vol.id_vol
WHERE Vol.Date='25/10/2022' ;
```

permet d'obtenir le nom et le prénom des astronautes ayant décollé le 25 octobre 2022.

Exercice 3

Partie 1

1. La taille de l'arbre de la figure 1 est 5 nœuds, et sa hauteur est 3 (avec la définition de l'énoncé qui correspond au nombre de niveaux).
2. On a ajouté deux **feuilles** dans l'ABR de la figure 1.



3. Réponse **C**.

Pour parcourir l'arbre dans l'ordre lexicographique on utilise un parcours en profondeur dans l'ordre infixe.

4. On a complété la méthode `present` de la classe `ABR`, qui implémente la recherche dichotomique dans un ABR avec un paradigme orienté objet.

```
1    def present(self, identifiant):
2        if self.est_vide():
3            return False
4        elif self.racine() == identifiant:
5            return True
6        elif self.racine() < identifiant:
7            return self.sd().present(identifiant)
8        else:
9            return self.sg().present(identifiant)
```

Partie 2

5. (a) L'appel de fonction `est_vide(f1)` renvoie `False`.

   (b) Après l'exécution de `defiler(f1)`, la file `f1` est :

			⟶
	'bac'	'nsi'	'2023'

   (c) La file `f2` est :

			⟶
	'poule'	'python'	'castor'

6. On complète la fonction `longueur`.

```
def longueur(f):
    resultat=0
    g=creer_file()
    while not est_vide(f):
        elt=defiler(f)
        resultat=resultat+1
        enfiler(g,elt)
    while not est_vide(g):
        enfiler(f,defiler(g))
    return resultat
```

7. Le mot de passe accepté par la fonction `est_valide` est `'2!@59fgds'`.

8. On écrit une fonction `ajouter_mot(f,mdp)` qui enfile `mdp` en gardant une taille maximale de trois éléments dans cette file.

```
def ajouter_mot(f,mdp):
    enfiler(f,mdp)
    if longueur(f)>3:
        defiler(f)
```

   On pourrait envisager de remplacer le `if` par un `while`, mais ce n'est pas utile puisque le cahier des charges précise qu'on appelle la fonction avec une file `f` qui contient au plus trois éléments.

9. On complète les lignes 7 et 8 de la fonction `mot_file`.

```
def mot_file(f,mdp):
    g=creer_file()
    present=False
    while not est_vide(f):
        elt=defiler(f)
        enfiler(g,elt)
        if mdp==elt:
            present=True
    while not est_vide(g):
        enfiler(f,defiler(g))
    return present
```

10. La fonction `modification` utilise les fonctions `est_valide`, `mot_file` et `ajouter_mot` définies précédemment.

```
def modification(f,nv_mdp):
    if est_valide(mdp) and not mot_file(f,nv_mdp):
        ajouter_mot(f,nv_mdp)
        return True
    return False
```