Amérique du nord - mars 2023 - sujet 2

Exercice 1 (Bases de données et protocoles de routage - 4 points)
Partie A
L'énoncé de cette partie utilise les mots du langage SQL suivant : `SELECT`, `FROM`, `WHERE`, `UPDATE`, `JOIN`, `ON`, `INSERT`, `INTO`, `VALUES`, `AND`, `COUNT`, `DISTINCT`.

Une compétition internationale de rugby féminin est organisé en France. Une base de données relationnelle est mise en place afin de gérer l'organisation.

Son schéma relationnel est donné ci-dessous :

`JOUEUSE(idjoueuse, #pays, nom, prenom, age, numero)`
`SELECTION(#idjoueuse, #idmatch, points)`
`MATCH(idmatch, stade, jour, horaire)`

Dans ce schéma, les clés primaires sont soulignées et les clés étrangères sont précédées du symbole `#`.

On donne ci-dessous un extrait de la relation `JOUEUSE` :

idjoueuse	pays	nom	prenom	age	numero
101	"France"	"Lefevre"	"Charline"	23	15
108	"Australie"	"Porteur"	"Cindy"	31	7
305	"Argentine"	"Gomez"	"Laëtitia"	35	8
318	"Tunisie"	"Char"	"Jo"	30	2

1. (a) Expliquer ce que renvoie la requête SQL suivante :

```
SELECT nom, prenom, numero FROM JOUEUSE WHERE pays = "France";
```

   (b) Ecrire une requête permettant d'obtenir le nom et le prénom de toutes les joueuses de l'équipe d'Argentine ayant au moins 30 ans.

2. (a) Une erreur de saisie a été commise : la joueuse Charline Lefevre n'a pas 23 ans, mais 33 ans. Ecrire une requête SQL permettant de mettre à jour son âge.

   (b) Ecrire une requête SQL permettant d'insérer la joueuse dont l'identifiant `idjoueuse` est 105, le nom Warm, le prénom Suzanna, l'âge 29 ans, le pays Angleterre et le numéro 6.

3. On suppose que la compétition est terminée et que les tables sont correctement remplies.

   (a) Recopier en intégralité sur la copie la requête SQL ci-dessous et la compléter de façon à ce qu'elle renvoie les noms des joueuses qui ont marqué au moins 10 points lors d'un match.

```
SELECT .....
FROM JOUEUSE
JOIN SELECTION ON .....
WHERE .....
```

   (b) On rappelle, qu'en langage SQL, la fonction d'agrégation `COUNT` permet de compter un nombre d'enregistrements. Par exemple, pour déterminer le nombre de joueuses dans la table `JOUEUSE`, on peut utiliser la requête suivante :

```
SELECT COUNT(idjoueuse) FROM JOUEUSE;
```

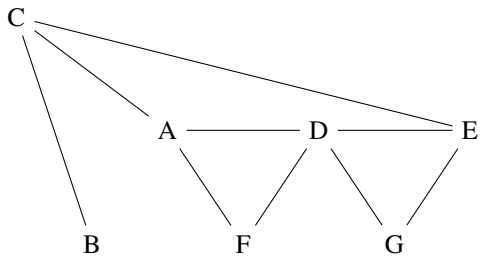
   Ecrire une requête permettant de connaître le nombre de sélections de la joueuse Laëtitia Gomez, dont l'identifiant `idjoueuse` est 305, lors de cette compétition.

   (c) Ecrire une requête permettant de connaître les noms des stades dans lesquels la joueuse Laëtitia Gomez, dont l'identifiant `idjoueuse` est 305, a joué lors de cette compétition.

Partie B

Lors de cette compétition, les échanges numériques mondiaux officiels (en particulier les différents flux vidéo) sont effectués de continent à continent en suivant les liaisons simplifiées ci-dessous.

Les routeurs principaux de chaque continent affectés à ces échanges sont identifiés par les lettres de A à G. Le réseau de ces routeurs est modélisé par le schéma suivant :



A : Amérique du Nord
B : Océanie
C : Asie
D : Europe
E : Moyen-Orient
F : Amérique du Sud
G : Afrique

Tous les routeurs utilisent le protocole RIP qui minimise la distance, exprimée en nombre de sauts, qui séparent deux routeurs.

4. Le routeur G de l'Afrique doit transmettre un message au routeur B de l'Océanie, en effectuant un nombre minimal de sauts. Déterminer le trajet parcouru.
5. On s'intéresse à la table de routage du routeur F de l'Amérique du Sud une fois les tables stabilisées. Recopier et compléter sur la copie la table suivante :

Table du routeur F de l'Amérique du Sud		
Destination	Routeur suivant	Distance
A		
B		
C		
D		
E		
G		

6. Déterminer une panne qui obligerait toutes les données échangées entre l'Afrique et l'Amérique du Sud à transiter par le routeur d'Asie. Justifier la réponse.

Exercice 2 (Arbres binaires et ABR - 5 points)
Partie A
Dans cet exercice, on considère que la hauteur de l'arbre vide est 0.

On considère l'arbre binaire suivant, noté `A` dans la suite de l'exercice.

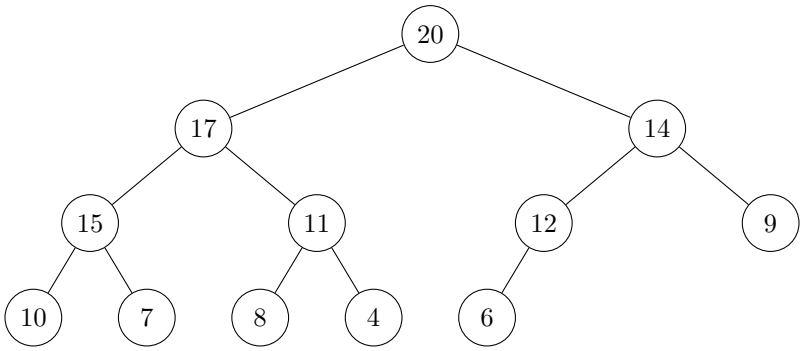


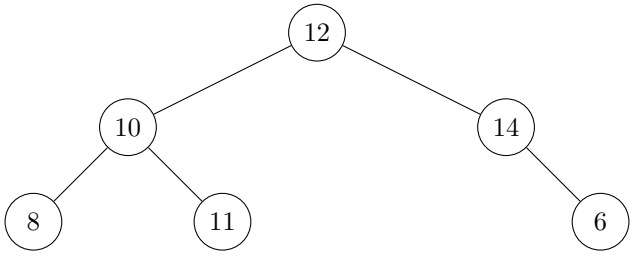
FIGURE 1 – L'arbre binaire `A`

1. Déterminer, sans justifier, la hauteur et la taille de l'arbre binaire `A`.
2. On utilise l'interface suivante pour un arbre binaire :
   - ⋆ `arbre_vide()` : renvoie un arbre vide ;
   - ⋆ `est_vide(A)` : renvoie `True` si l'arbre binaire `A` est vide, `False` sinon ;
   - ⋆ `cons(v, SAG, SAD)` : renvoie un arbre binaire dont la valeur de la racine est `v`, le sous-arbre gauche est `SAG` et le sous-arbre droit est `SAD` ;
   - ⋆ `gauche(A)` : renvoie le sous-arbre gauche de l'arbre binaire `A` ;
   - ⋆ `droit(A)` : renvoie le sous-arbre droit de l'arbre binaire `A` ;
   - ⋆ `racine(A)` : renvoie la valeur du nœud racine de l'arbre binaire `A`.

   Cette interface a été implémentée en langage Python.
   (a) Dessiner l'arbre binaire obtenu avec l'instruction suivante :

```
cons(7,
     cons(4,
          cons(2, arbre_vide(), arbre_vide()),
          cons(6,
               cons(5, arbre_vide(), arbre_vide()),
               arbre_vide())),
     cons(9,
          arbre_vide(),
          cons(8, arbre_vide(), arbre_vide())))
```

   (b) Déterminer la ou les instructions en langage Python permettant d'obtenir l'arbre binaire suivant :



3. (a) Déterminer l'ordre des valeurs obtenu lorsqu'on parcourt l'arbre `A` à l'aide d'un parcours infixe.
   (b) La fonction `parcours` incomplète ci-après prend en paramètre un arbre binaire et renvoie un tableau contenant les valeurs de l'arbre, dans l'ordre obtenu lors d'un parcours infixe.

```
1  def parcours(arbre):
2      if est_vide(arbre):
3          return []
4      else:
5          .....
```

Déterminer, parmi les trois propositions suivantes, quelle instruction doit-être écrite à la ligne 5.

⋆ Proposition 1 :

```
return [racine(arbre)] + parcours(gauche(arbre)) + parcours(droit(arbre))
```

⋆ Proposition 2 :

```
return parcours(gauche(arbre)) + [racine(arbre)] + parcours(droit(arbre))
```

⋆ Proposition 3 :

```
return parcours(gauche(arbre)) + parcours(droit(arbre)) + [racine(arbre)]
```

Partie B

Un arbre binaire de recherche est un arbre binaire dans lequel chaque nœud possède une valeur, telle que chaque nœud du sous-arbre gauche ait une valeur inférieure ou égale à celle du nœud considéré, et que chaque nœud du sous-arbre droit ait une valeur supérieure à celle-ci.

Par exemple, l'arbre suivant, noté `B` dans la suite de l'exercice, est un arbre binaire de recherche.

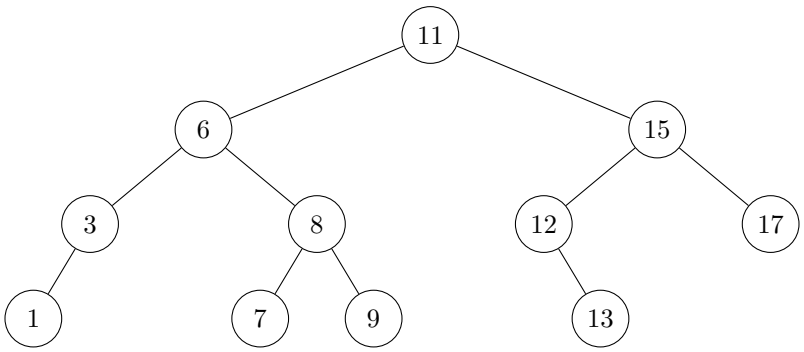
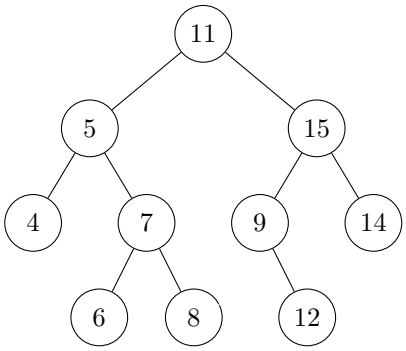
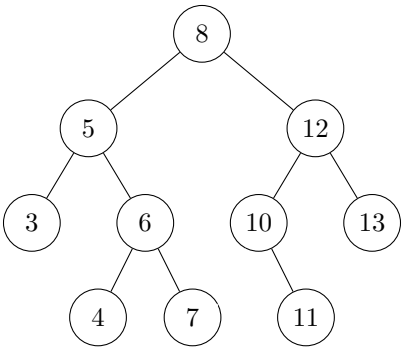


FIGURE 2 – L'arbre binaire de recherche `B`

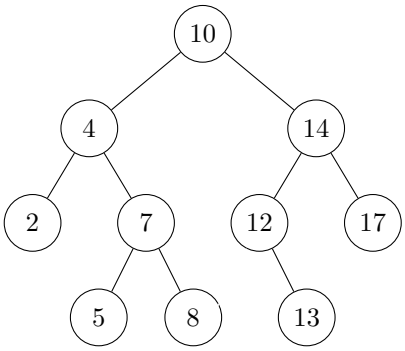
4. Déterminer en justifiant, parmi les arbres suivants, ceux qui ne sont pas des arbres binaires de recherche.



Arbre 1



Arbre 2



Arbre 3

5. Déterminer, en justifiant, quel parcours d'un arbre binaire de recherche permet d'obtenir les valeurs de l'arbre classées par ordre croissant.

6. On ajoute à l'interface des arbres binaires une fonction `inserer_dans_ABR` qui prend en paramètres un arbre binaire de recherche et une valeur, et qui ajoute une feuille contenant la valeur, à sa place dans l'arbre binaire de façon à ce que la structure d'arbre binaire de recherche soit toujours respectée.
   (a) Dessiner l'arbre `B` obtenu après l'exécution de l'instruction `inserer_dans_ABR(B, 16)`.
   (b) Dessiner l'arbre binaire de recherche que l'on obtient, en partant de l'arbre vide, et en insérant les valeurs suivantes dans cet ordre : 5, 3, 7, 8, 4, 1, 9, 2 et 6.
7. On veut écrire une fonction `tri` qui prend en paramètre un tableau de nombres `T` et qui renvoie un tableau trié en utilisant l'algorithme suivant :
   - on construit un arbre binaire de recherche en insérant, à partir d'un arbre vide, les éléments du tableau `T` ;
   - on construit le tableau que l'on obtient en parcourant cet arbre binaire de recherche grâce à la fonction `parcours` de la partie A.

   Ecrire la fonction `tri` en Python en utilisant cet algorithme.

```
1 def tri(T):
2     ''' Trie le tableau de nombres T '''
```

Exercice 3 (Système d'exploitation, programmation générale et dictionnaires - 3 points)
Dans un système d'exploitation de type UNIX, on considère l'arborescence suivante dans laquelle les noms des répertoires sont en gras et ceux des fichiers en italique :

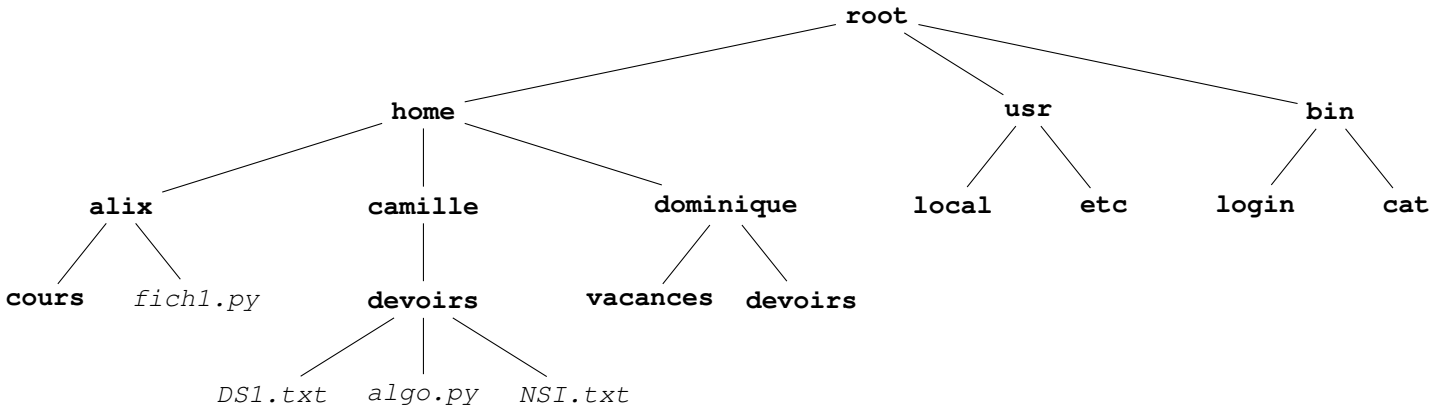


FIGURE 1

On donne ci-après un extrait de quelques commandes de base à utiliser dans un terminal sous linux :

- `ls` : affiche le contenu d'un répertoire
  `ls repertoire1` affiche le contenu de `repertoire1`
- `cd` : permet de changer de répertoire courant
  Après la commande `cd repertoire1`, le répertoire courant devient `repertoire1`
- `cp` : crée une copie d'un fichier
  `cp fichier1.py fichier2.py` copie `fichier1.py` sous le nom `fichier2.py`
- `mv` : déplace ou renomme un fichier ou un répertoire
  `mv fichier.txt dossier1` déplace `fichier.txt` dans `dossier1`
- `rm` : supprime un fichier ou un répertoire
  `rm mon_fichier.mp3` supprime `mon_fichier.mp3`
- `mkdir` : crée un répertoire
  `mkdir dossier2` crée le répertoire `dossier2`
- `cat` : affiche le contenu d'un fichier
  `cat fichier1.txt` affiche le contenu de `fichier1.txt`
- `touch` : crée un fichier qui n'existait pas jusqu'alors
  `touch fichier2.txt` crée `fichier2.txt`

Dans l'arborescence décrite à la figure 1 ci-dessus, `alix`, `camille` et `dominique` sont respectivement les répertoires personnels des utilisateurs Alix, Camille et Dominique.

On souhaite explorer et modifier les répertoires et fichiers grâce à l'utilisation d'un terminal.

1. Dans cette question, la problématique des droits d'accès n'est pas prise en compte.
   (a) Dans l'arborescence précédente, on suppose que le répertoire courant est `camille`. Parmi les quatre propositions suivantes, déterminer la commande qui permet d'afficher son contenu :

   Proposition 1 : `mv` **Proposition 3 :** `ls`
   Proposition 2 : `cp` **Proposition 4 :** `rm`

   (b) Dans l'arborescence précédente, on suppose que le répertoire courant est `camille`. Parmi le quatre propositions suivantes, déterminer toutes celles qui permettent que le répertoire `cours` d'Alix devienne le répertoire courant :

   Proposition 1 : `cd ../alix/cours` **Proposition 3 :** `cd alix/cours`
   Proposition 2 : `cd cours` **Proposition 4 :** `cd /home/alix/cours`

   (c) Dans l'arborescence précédente, on suppose que le répertoire courant est `alix`. Déterminer l'instruction permettant de copier le fichier `fich2.txt` qui se trouve dans le répertoire d'Alix dans le répertoire `devoirs` de Camille.

2. Dans cette question, la problématique des droits d'accès est prise en compte.
Camille, depuis le répertoire `camille/devoirs` utilise la commande `ls -l` et obtient le résultat suivant :

```
-rw-r--r--  1  root     root   1096  juil. 22 17:10  DS1.txt
-rw-r--r--  1  camille  eleve   128  juil. 22 16:03  algo.py
-r--r--r--  1  camille  eleve   128  juil. 22 18:21  NSI.txt
```

(a) Expliquer pourquoi l'instruction `rm DS1.txt` est refusée si l'utilisateur est Camille.

On donne ci-après un extrait du mode d'emploi de la commande `chmod` à utiliser dans un terminal sous linux :

`chmod` : modifie les permissions d'un fichier ou d'un dossier
`chmod` peut prendre en paramètres les caractères suivants et un nom de fichier

- `u` pour l'utilisateur (user)
- `g` pour le groupe (group)
- `o` pour les autres utilisateurs (other)
- `+` pour ajouter des droits
- `-` pour supprimer des droits
- `r` pour modifier les droits de lecture (read)
- `w` pour modifier les droits d'écriture (write)
- `x` pour modifier les droits d'exécution

Exemple :
L'instruction `chmod u+rwx o-x exo.py` ajoute sur `exo.py` des droits en lecture, écriture et exécution à l'utilisateur et supprime les droits d'exécution aux autres utilisateurs.

(b) Camille souhaite interdire la lecture du contenu du fichier `algo.py` aux autres utilisateurs (même ceux de son groupe). Elle souhaite également se donner la possibilité de l'exécuter. Ecrire l'instruction permettant de modifier les droits sur `algo.py`. On suppose que Camille bénéficie déjà des droits de super utilisateur.

3. Dans son répertoire `vacances`, Dominique a conservé tous les fichiers durant une année. Il décide d'indexer ces fichiers selon leur type : `"photo"`, `"vidéo"` ou `"son"`, le mois de leur création (un nombre entre 1 et 12), et leur taille en kilo-octets. Puis il les stocke dans une liste, implémentée par le type `list` de Python. Un extrait de cette liste est donné ci-desous :

```
fichiers_vacances = [
{"nom_fichier": "plage.jpg", "type": "photo", "mois": 7, "taille": 152},
{"nom_fichier": "mouette.mp3", "type": "son", "mois": 7, "taille": 54},
{"nom_fichier": "famille1.png", "type": "photo", "mois": 12, "taille": 257},
...]
```

(a) Recopier et compléter sur la copie le code de la fonction `classement_type` ci-dessous qui prend en paramètre une liste `L` au format précédent et qui renvoie trois listes `photos`, `sons` et `videos` contenant les éléments du type correspondant.

```
def classement_type(L):
    photos = []
    sons = []
    videos = []
    for fichier in ..... :
        if ..... == 'photo':
            photos.append(fichier)
        elif ..... == 'son':
            .....
        else:
            .....
    return photos, sons, videos
```

(b) Dominique souhaite maintenant compter le nombre de photos datant du mois de juillet. Ecrire une fonction `photo_juillet` qui prend en paramètre une liste de fichiers et qui renvoie le nombre de fichiers de type `"photo"` du mois de juillet.

(c) A partir des fichiers de la liste `photos_vacances`, Dominique souhaite stocker le plus de fichiers possible sur une clé USB d'une capacité de 64 Go. Proposer une démarche algorithmique pour y parvenir. On ne demande pas de la programmer en Python.