Exercices type bac

ABR, récursivité, POO, SGBD et algorithmique

Exercice 1 :

Cet exercice porte sur les arbres binaires de recherche, la programmation orientée objet et la récursivité.

Dans cet exercice, la taille d'un arbre est le nombre de nœuds qu'il contient. Sa hauteur est le nombre de nœuds du plus long chemin qui joint le nœud racine à l'une des feuilles (nœuds sans sous-arbres). On convient que la hauteur d'un arbre ne contenant qu'un nœud vaut 1 et la hauteur de l'arbre vide vaut 0.

1. On considère l'arbre binaire représenté ci-dessous:

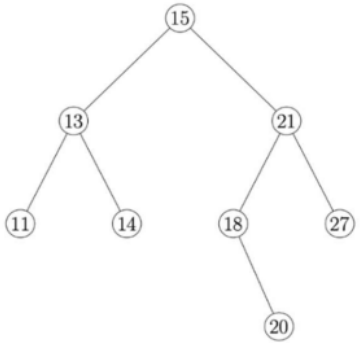


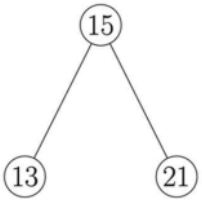
Figure 1

a. Donner la taille de cet arbre.
b. Donner la hauteur de cet arbre.
c. Représenter sur la copie le sous-arbre droit du nœud de valeur 15.
d. Justifier que l'arbre de la figure 1 est un arbre binaire de recherche.
e. On insère la valeur 17 dans l'arbre de la figure 1 de telle sorte que 17 soit une nouvelle feuille de l'arbre et que le nouvel arbre obtenu soit encore un arbre binaire de recherche. Représenter sur la copie ce nouvel arbre.

2. On considère la classe `Noeud` définie de la façon suivante en Python :

```
1  class Noeud:
2      def __init__(self, g, v, d):
3          self.gauche = g
4          self.valeur = v
5          self.droit = d
```

a. Parmi les trois instructions **(A)**, **(B)** et **(C)** suivantes, écrire sur la copie la lettre correspondant à celle qui construit et stocke dans la variable `abr` l'arbre représenté ci-contre.



(A) `abr=Noeud(Noeud(Noeud(None,13,None),15,None),21,None)`

(B) `abr=Noeud(None,13,Noeud(Noeud(None,15,None),21,None))`

(C) `abr=Noeud(Noeud(None,13,None),15,Noeud(None,21,None))`

b. Recopier et compléter la ligne 7 du code de la fonction `ins` ci-dessous qui prend en paramètres une valeur `v` et un arbre binaire de recherche `abr` et qui renvoie l'arbre obtenu suite à l'insertion de la valeur `v` dans l'arbre `abr`. Les lignes 8 et 9 permettent de ne pas insérer la valeur `v` si celle-ci est déjà présente dans `abr`.

```
1  def ins(v, abr):
2      if abr is None:
3          return Noeud(None, v, None)
4      if v > abr.valeur:
5          return Noeud(abr.gauche,abr.valeur,ins(v,abr.droit))
6      elif v < abr.valeur:
7          return ..........................................
8      else:
9          return abr
```

3. La fonction `nb_sup` prend en paramètres une valeur `v` et un arbre binaire de recherche `abr` et renvoie le nombre de valeurs supérieures ou égales à la valeur `v` dans l'arbre `abr`.
Le code de cette fonction `nb_sup` est donné ci-dessous :

```
1  def nb_sup(v, abr):
2      if abr is None:
3          return 0
4      else:
5          if abr.valeur >= v:
6              return 1+nb_sup(v, abr.gauche)+nb_sup(v, abr.droit)
7          else:
8              return nb_sup(v, abr.gauche)+nb_sup(v, abr.droit)
```

a. On exécute l'instruction `nb_sup(16, abr)` dans laquelle `abr` est l'arbre initial de la figure 1. Déterminer le nombre d'appels à la fonction `nb_sup`.

b. L'arbre passé en paramètre étant un arbre binaire de recherche, on peut améliorer la fonction `nb_sup` précédente afin de réduire ce nombre d'appels. Écrire sur la copie le code modifié de cette fonction.

Exercice 2 :

Cet exercice porte sur les bases de données relationnelles et le langage SQL.

L'énoncé de cet exercice utilise les mots du langage SQL suivants :

`SELECT, FROM, WHERE, JOIN, INSERT INTO, VALUES, UPDATE, SET, COUNT`

On rappelle qu'en SQL, la fonction d'agrégation `COUNT` permet de compter le nombre d'enregistrements dans une table.

L'entreprise "Vacances autrement" propose des séjours d'une durée d'une semaine dans différentes stations situées en France. Les séjours sont organisés autour d'une seule activité sportive. Par exemple, la station "La Tramontane Catalane" située à Leucate propose une formule autour de la planche à voile. Les clients pourront durant toute la durée du séjour pratiquer ce sport : le matériel est fourni et deux sessions encadrées par des moniteurs diplômés sont organisées chaque journée. Cette même station propose aussi des séjours pour les amateurs de kitesurf.
Les réservations se font par l'intermédiaire d'un site internet : chaque client complète un formulaire avec ses données personnelles (nom, prénom...) et choisit le séjour souhaité.
Ce site s'appuie sur une base de données relationnelle. Elle contient notamment les relations `Station`, `Sport`, `Client` et `Sejour`.

Voici le schéma relationnel où les clefs primaires sont soulignées et les clefs étrangères sont suivies du symbole # :

```
Station (nomStation, ville, region)
Sport (nomSport, nomStation#, prix)
Client (cID, nom, prenom, mail)
Sejour (cID, semaine, annee, nomStation#,nomSport#)
```

— Les différentes stations sont stockées dans la relation `Station`. Chaque station est identifiée par un nom caractéristique (`nomStation`). Les attributs `ville` et `region` permettent de décrire la localisation de chacune des stations.

— La relation `Sport` fait le lien entre les stations et les activités sportives qui y sont proposées. Les trois attributs sont `nomSport`, `nomStation` et `prix`. L'attribut `prix` est un nombre entier : il correspond au montant plein tarif (en euros) du séjour d'une semaine dans la station avec le sport associé.

— La relation `Client` caractérise les clients de l'entreprise : ils sont identifiés par un nombre entier (`cID`) et décrits par leur nom, prénom et adresse mail.

— La relation `Sejour` permet de lister les séjours auxquels les clients ont participé. Les deux attributs `semaine` et `annee` sont des nombres entiers : ils permettent d'identifier le moment où a été effectué chaque séjour (`semaine` est le numéro de la semaine pendant laquelle le séjour a été réalisé).

Exemple : un séjour ayant été effectué la semaine numéro 12 de l'année 2020 correspond à `semaine` = 12 et à `annee` = 2020.

On donne aussi un extrait des trois premières lignes de chacune de ces relations :

`Station`

nomStation	ville	region
La tramontane catalane	Leucate	Occitanie
La baie sauvage	La Torche	Bretagne
La pinède	Calvi	Corse

`Sport`

nomSport	nomStation	prix
planche à voile	La tramontane catalane	1200
kitesurf	La tramontane catalane	1100
plongée	La baie sauvage	950

`Client`

cID	nom	prenom	mail
1	GENEREUX	Eric	eric.genereux@mail.fr
2	PIERRE	Daniel	daniel.pierre@mail.fr
3	JOLY	Emilie	emilie.joly@mail.fr

`Sejour`

cID	semaine	annee	nomStation	nomSport
1	26	2020	La tramontane catalane	planche à voile
1	38	2020	La tramontane catalane	planche à voile
2	33	2020	La baie sauvage	plongée

1. (a) Donner la clé primaire et les éventuelles clés étrangères de la relation `Sport`.
   (b) Citer une contrainte d'intégrité de domaine puis une contrainte d'intégrité de relation et enfin une contrainte d'intégrité de référence que doivent respecter les données de la relation `Sport`.
2. (a) Le tarif du séjour à la station "La tramontane catalane" basé sur la planche à voile passe de 1 200 euros à 1 350 euros.
   La requête SQL suivante a été utilisée pour la mise à jour du tarif :

   ```
   INSERT INTO Sport VALUES ("planche à voile","La tramontane catalane",1350);
   ```

   Cette requête a été rejetée et la mise à jour n'a pas été effectuée. Après avoir expliqué pourquoi cette requête a été refusée, proposer une requête SQL qui permettra de modifier le tarif de ce séjour.
   (b) Une nouvelle station vient d'être référencée. Son nom est "Soleil Rouge". Elle est située à Bastia en Corse. Des séjours d'une semaine y seront organisés. Le sport pratiqué sera la plongée au tarif de 900 euros.
   Écrire les requêtes SQL permettant d'insérer ces nouvelles données dans la base.
3. (a) L'agence souhaite envoyer un mail d'information à ses clients pour leur présenter cette nouvelle possibilité de séjour mise en place à Bastia. Écrire une requête SQL permettant d'obtenir l'adresse mail de tous les clients.
   (b) Un client qui pratique la plongée souhaite réserver un séjour. Écrire une requête SQL permettant d'obtenir le nom de toutes les stations où l'on peut pratiquer la plongée.

4. (a) Pour faire son choix, le client souhaiterait connaître les villes où sont situées les stations dans lesquelles il pourra pratiquer la plongée.
   Écrire une requête SQL permettant d'obtenir le nom des villes ainsi que le nom des stations où l'on peut pratiquer la plongée.

   (b) Écrire une requête SQL permettant de déterminer le nombre total de séjours effectués en Corse durant l'année 2020.

Exercice 3 : **Structures de données (matrices), programmation**

Le « jeu de la vie » se déroule sur une grille à deux dimensions dont les cases, qu'on appelle des « cellules », par analogie avec les cellules vivantes, peuvent prendre deux états distincts : « vivante » (= 1) ou « morte » (= 0).

Une cellule possède au plus huit voisins, qui sont les cellules adjacentes horizontalement, verticalement et diagonalement.

À chaque étape, l'évolution d'une cellule est entièrement déterminée par l'état de ses huit voisines de la façon suivante :

- Règle 1 : une cellule morte possédant exactement trois voisines vivantes devient vivante (elle naît) ; sinon, elle reste à l'état « morte »
- Règle 2 : une cellule vivante possédant deux ou trois voisines vivantes reste vivante, sinon elle meurt.

Voici un exemple d'évolution du jeu de la vie appliquée à la cellule centrale :

Voici un exemple d'évolution du jeu de la vie appliquée à la cellule centrale :

1	1	0
0	0	0
0	0	1

devient par la règle 1

	1	

1	0	0
0	1	1
1	0	0

reste par la règle 2

	1	

0	0	0
1	1	0
0	0	0

devient par la règle 2

	0	

1	1	0
0	1	1
1	1	0

devient par la règle 2

	0	

Pour initialiser le jeu, on crée en langage Python une grille de dimension **8x8**, modélisée par une liste de listes.

1. Initialisation du tableau :

a. Parmi les deux scripts proposés, indiquer celui qui vous semble le plus adapté pour initialiser un tableau de 0. Justifier votre choix

Choix 1	Choix 2

`1 | ligne = [0,0,0,0,0,0,0,0]`
`2 | jeu = []`
`3 | for i in range(8) :`
`4 |  jeu.append(ligne)`
	`1 | jeu = []`
	`2 | for i in range(8) :`
	`3 |  ligne = [0,0,0,0,0,0,0,0]`
	`4 |  jeu.append(ligne)`

b. Donner l'instruction permettant de modifier la grille `jeu` afin d'obtenir

```
>>> jeu
[[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0]]
```

2.

a. Ecrire en langage Python une fonction `remplissage(n, jeu)` qui prend en paramètres un entier `n` et une grille `jeu`, et qui ajoute aléatoirement exactement `n` cellules vivantes dans le tableau `jeu`.

b. Quelles sont les préconditions de cette fonction pour la variable `n` ?

On propose la fonction en langage Python `nombre_de_vivants(i, j, jeu)` qui prend en paramètres deux entiers `i` et `j` ainsi qu'une grille `jeu` et qui renvoie le nombre de voisins **vivants** de la cellule `tab[i][j]` :

```
1 | def nombre_de_vivants(i, j, jeu) :
2 |     nb = 0
3 |     voisins = [(i-1,j-1), (i-1,j), (i-1,j+1), (i,j+1),
4 |                (i+1,j+1), (i+1,j), (i+1,j-1), (i,j-1)]
5 |     for e in voisins :
6 |         if 0 <= ... < 8 and 0 <= ... < 8 :
7 |             nb = nb + jeu[...][...]
8 |     return nb
```

3. Recopier et compléter les pointillés pour que la fonction réponde à la demande.

4. En utilisant la fonction `nombre_de_vivants(i, j, jeu)` précédente, écrire en langage Python une fonction `transfo_cellule(i, j, jeu)` qui prend en paramètres deux entiers `i` et `j` ainsi qu'une grille `jeu` et renvoie le nouvel état de la cellule `jeu[i][j]` (0 ou 1)

Eléments de correction

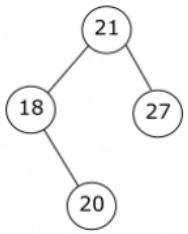
Exercice 1

1.

a. la taille de l'arbre est 8

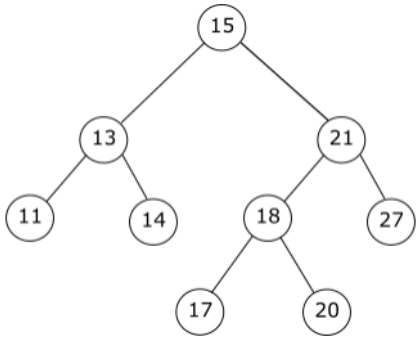
b. la hauteur de l'arbre est 4

c.



d. si on effectue un parcours infixe de l'arbre (11-13-14-15-18-20-21-27), on obtient les valeurs dans l'ordre croissantes, nous avons donc bien un arbre binaire de recherche.

e.



2.

a. (C) `abr=Noeud(Noeud(None,13,None),15,Noeud(None,21,None))`

b.

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

3.

a. En comptant l'appel initial, nous avons 17 appels à la fonction nb_sup

b.

```
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. La relation Sport a pour clé primaire le couple d'attribut (*nomSport*, *nomSation*) et pour clé étrangère l'attribut *nomStation*.

b.

- contrainte d'intégrité de domaine : l'attribut *prix* est de type nombre entier
- contrainte d'intégrité de relation : chaque couple d'attributs (*nomSport*, *nomSation*) doit être unique.
- contrainte d'intégrité de référence : chaque valeur de l'attribut *nomStation* doit correspondre aux valeurs de l'attribut *nomStation* de la relation *Station*

2.

a. Une requête d'insertion a été utilisée à la place d'une requête de mise à jour. L'entrée avec le couple ("planche à voile", "La tramontane catalane") existe déjà dans la relation Sport, d'où l'erreur (rappel : chaque couple d'attributs (*nomSport*, *nomSation*) doit être unique).
Requête correcte :

```
UPDATE Sport
SET prix = 1350
WHERE nomSport = "planche à voile" AND nomSation = "La tramontane
catalane"
```

b.

```
INSERT INTO Station
VALUES
("Soleil Rouge", "Bastia", "Corse")
```

et

```
INSERT INTO Sport
VALUES
("plongée", "Soleil Rouge", 900)
```

3.

a.

```
SELECT mail
FROM Client
```

b.

```
SELECT nomStation
FROM Sport
WHERE nomSport = "plongée"
```

4.

a.

```
SELECT Station.ville, Station.nomStation
FROM Station
JOIN Sport ON Station.nomStation = Sport.nomStation
WHERE Sport.nomSport = "plongée"
```

b.

```
SELECT COUNT(*)
FROM Sejour
JOIN Station ON Sejour.nomStation = Station.nomStation
WHERE Station.region = "Corse" AND Sejour.annee = 2020
```

Exercice 3 :

1.

a.

Le choix 2 est le plus adapté. En effet, dans les choix 1, toutes les lignes du tableau seront liées les unes aux autres (toutes les lignes seront identiques). La modification d'une ligne entrainera la modification de toutes les autres lignes.

b. L'instruction permettant de modifier le tableau : `jeu[5][2]=1`.

2.

a.

```
def remplissage(n, jeu):
    for i in range(n):
        x = random.randint(0,7)
        y = random.randint(0,7)
        while jeu[y][x] != 0:
            x = random.randint(0,7)
            y = random.randint(0,7)
        jeu[y][x] = 1
```

b.

La variable n doit être un entier compris entre 0 et 64 (il y a 64 cases dans le tableau)

3.

```
def nombre_de_vivants(i, j, jeu) :
    nb=0
    voisins = [(i-1,j-1), (i-1,j), (i-1,j+1), (i,j+1),(i+1,j+1),
(i+1,j), (i+1,j-1), (i,j-1)]
    for e in voisins :
        if 0 <= e[0] < 8 and 0 <= e[1] < 8 :
            nb = nb + jeu[e[0]][e[1]]
    return nb
```

4.

```
def transfo_cellule(i, j, jeu):
    nb_v = nombre_de_vivants(i, j, jeu)
    if jeu[i][j] == 0 and nb_v == 3 :
        return 1
    if jeu[i][j] == 1 and (nb_v == 3 or nb_v == 2) :
        return 1
    return 0
```