

---

# Numérique et Sciences Informatiques

*Épreuve type BAC*

---

20 février 2023

Durée de l'épreuve : 2 heures 00

L'usage de la calculatrice et du dictionnaire n'est pas autorisé.

Dès que ce sujet vous est remis, assurez-vous qu'il est complet. Ce sujet comporte 6 pages numérotées de 1/6 à 6/6.

Nom : \_\_\_\_\_

Classe : \_\_\_\_\_

Exercice 1 : Base de données

8 points

Dans cet exercice, on pourra utiliser les mots clés suivants du langage SQL : **SELECT**, **FROM**, **WHERE**, **JOIN**, **ON**, **INSERT INTO**, **UPDATE**, **SET**, **VALUES**, **COUNT**, **OR**, **AND**, **LIKE**.

Les articles d'un journal sont recensés dans une base de données **Journal**. Cette base contient des informations sur chaque article (titre, auteur et date de parution). Chaque auteur est caractérisé par son nom et son prénom. On associe de plus à certains articles des thèmes.

Le schéma relationnel de la base de données est donné ci-dessous :

- **Articles** (idArticle, titre, #auteur, dateParution)
- **Auteurs** (idAuteur , nom, prenom)
- **Themes** (idTheme , theme)
- **Traitements** (idTraitement , #article, #theme)

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

L'attribut **auteur** de la relation **Articles** est une clé étrangère faisant référence à l'attribut **idAuteur** de la relation **Auteurs**.

L'attribut **article** de la relation **Traitements** est une clé étrangère faisant référence à l'attribut **idArticle** de la relation **Articles**.

L'attribut **theme** de la relation **Traitements** est une clé étrangère faisant référence à l'attribut **idTheme** de la relation **Themes**.

Tous les attributs dont le nom est préfixé par **id** sont des nombres entiers ainsi que l'attribut **dateParution** (les dates sont notées au format **aaaaMMjj**, par exemple 20220103 pour le 3 janvier 2022). Les autres attributs sont des chaînes de caractères.

Les entrées de la relation **Traitements** correspondent aux thèmes abordés par des articles. Un article peut aborder 0, 1 ou plusieurs thèmes.

1. Expliquer pourquoi il n'est pas possible d'insérer une entrée dans la relation **Articles** si les relations **Auteurs** et **Themes** sont vides.

On fournit les extraits suivants de chacune des relations.

Articles			
idArticle	titre	auteur	dateParution
1	Présidence française de l'UE	2	20220112
2	Les bleues en demies !	3	20220110

Auteurs			Themes	
idAuteur	nom	prenom	idTheme	themes
1	Londres	Alfred	1	Politique
2	Jeraus	Jean	2	Sport
3	Zola	Étienne	3	Europe
			4	Handball

Traitements		
idTraitement	article	theme
1	1	1
2	1	3
3	2	2
4	2	3

2. L'article nommé « *Les bleues en demies!* » portait sur la coupe d'Europe de handball. Écrire une requête **SQL** permettant d'ajouter le thème **handball** à cet article dans la table (ou relation) **Traitements**.
3. Le journaliste « Jean Jèraus » tient à ce que son nom soit écrit avec l'accent grave. Ce n'est pas le cas pour l'instant dans la base de données. Écrire la requête **SQL** permettant de mettre à jour les informations afin de satisfaire sa demande.
4. Pour chacun des items suivants, écrire une requête **SQL** permettant d'extraire les informations demandées.
  - a. Le titre des articles parus après le 1<sup>er</sup> janvier 2022 inclus.
  - b. Le titre des articles écrits par l'auteur Étienne Zola (on sait que son **idAuteur** est le 3).
  - c. Le nombre d'articles écrits par l'auteur Jacques Pulitzer (présent dans la table **Auteurs** mais on ne connaît pas son **idAuteur**).
  - d. Les dates de parution des articles traitant du thème « Sport ».

### Exercice 2 : Arbres binaires équilibrés

12 points

On s'intéresse dans cet exercice aux arbres binaires **équilibrés**.

On dit qu'un arbre binaire est équilibré si, quel que soit le nœud considéré, les hauteurs des sous-arbres issus de ses fils gauche et droit ne diffèrent au maximum que de 1. Dans la figure 1, l'arbre de gauche est équilibré alors que celui de droite ne l'est pas.

On considère que la hauteur d'un arbre réduit à sa racine vaut 1. La hauteur d'un arbre vide vaut donc 0.

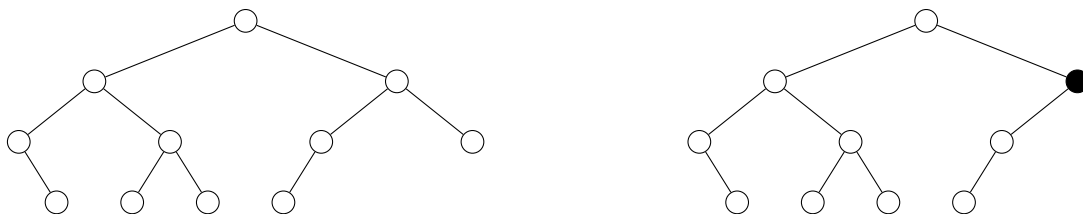


FIGURE 1 – Un arbre binaire équilibré (à gauche) et un non-équilibré (à droite)

### Partie A

Afin de déterminer si un arbre est équilibré ou non, on calcule la **balance** de chaque nœud. Cette grandeur est égale à la différence des hauteurs de son fils droit et de son fils gauche. Dans la figure 1, le nœud colorié en noir a une balance de  $-2$ . En effet, il n'a pas de fils droit et son fils gauche a une hauteur de 2 : sa balance vaut donc  $0 - 2 = -2$ .

Un arbre binaire est équilibré si la balance de chacun de ses nœuds vaut  $-1$ ,  $0$  ou  $1$ .

1. a. On considère l'arbre de la figure 2. Compléter la figure sur cet énoncé en indiquant à côté de chaque nœud sa balance.

b. Cet arbre est-il équilibré ?

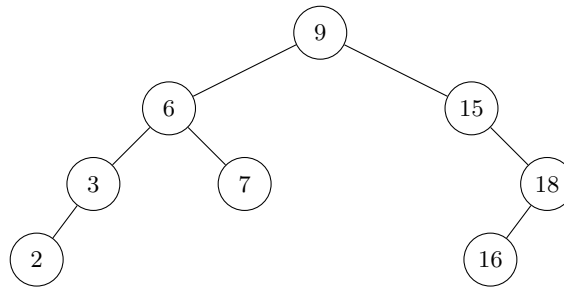
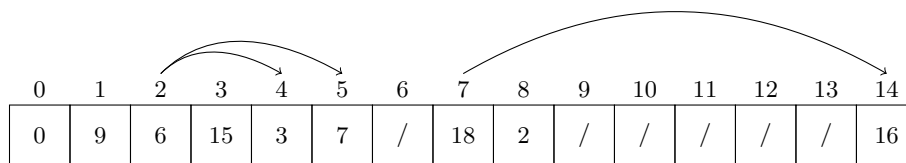


FIGURE 2

On implémente les arbres binaires à l'aide de listes `python`. Dans ce cadre :

- On place la valeur 0 dans la cellule d'indice 0. **Cette valeur sera ignorée dans les différents traitements**
- la valeur de la racine de l'arbre est dans la cellule d'indice 1
- si l'on considère un nœud dont la valeur est stockée dans la cellule  $n$  :
  - la valeur de son fils gauche est stockée dans la cellule  $2n$
  - la valeur de son fils droit est stockée dans la cellule  $2n + 1$
- si un nœud est vide, on stocke une valeur absurde dans le tableau (**None** en `python`)
- le tableau se termine à la dernière valeur non-vide



[0, 9, 6, 15, 3, 7, **None**, 18, 2, **None**, **None**, **None**, **None**, **None**, 16]

FIGURE 3 – Représentation de l'arbre de la figure 2 dans un tableau et une liste `python`

2. a. Donner la liste `python` représentant l'arbre de la figure 4.

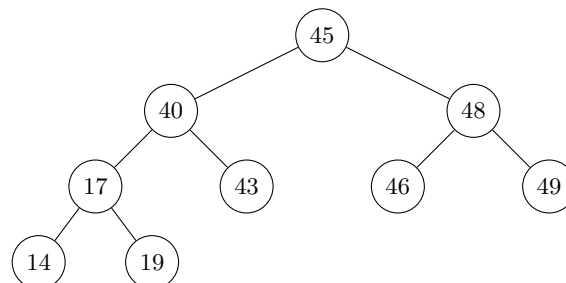


FIGURE 4

b. Dessiner l'arbre représenté par la liste suivante [0, 35, 30, 12, 20, 23, 10, 5, 2].

On fournit la fonction `f` prenant en argument un arbre donné sous la forme d'une liste `python` ainsi que l'indice du nœud racine d'un de ses sous-arbres et renvoyant un nombre entier.

```

1 def f(arbre: list, i: int) -> int:
2     if i >= len(arbre) or arbre[i] is None:
3         return 0
4     else:
5         g = f(arbre, 2*i)
6         d = f(arbre, 2*i + 1)
7         return 1 + max(g, d)

```

3. a. On considère la variable `arbre` égale à `[0, 30, 20, 40, 18, 25, None, 47]`. Que renvoie l'appel `f(arbre, 1)`? Justifier.

b. Que permet de calculer la fonction `f`?

4. Compléter sur cet énoncé le code de la fonction suivante indiquant si un arbre est équilibré ou non. On rappelle que par définition un arbre vide est équilibré. `arbre` est la liste python contenant un arbre et `i` l'indice d'un nœud, racine du sous-arbre étudié.

```

1 def estEquilibre(arbre: list, i : int) -> bool:
2     if i >= len(arbre) or arbre[i] is None:
3         return .....
4     else:
5         balance = f(arbre, ..... ) - f(arbre, ..... )
6         reponse = balance in [..., ..., ...]
7         return reponse and estEquilibre(arbre,.....) and .....(.....,.....)

```

## Partie B :

On s'intéresse dans cette sous-partie aux arbres binaires de recherche. On rappelle que dans un tel arbre, quel que soit le nœud considéré, son fils gauche, s'il existe, a une valeur qui lui est inférieure ou égale et son fils droit, s'il existe, une valeur qui lui est strictement supérieure. Les différents arbres dessinés plus haut sont tous des arbres binaires de recherche.

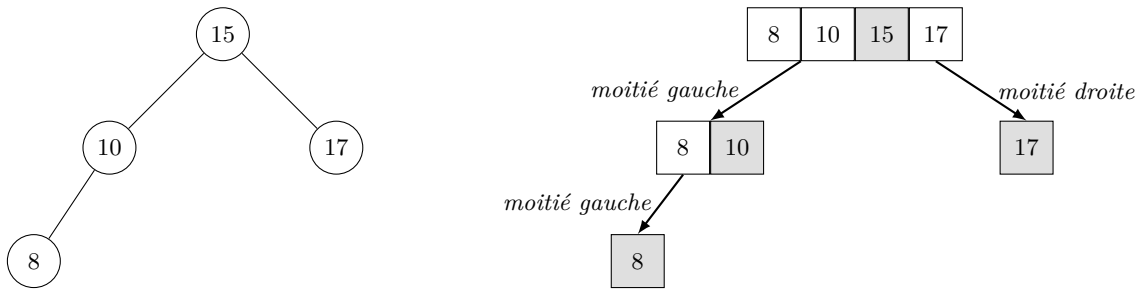
1. Donner les parcours *préfixe*, *infixe* et *suffixe* de l'arbre de la figure 4.

Dans le cas où un arbre binaire de recherche n'est pas équilibré, la méthode ci-dessous permet de construire un arbre binaire de recherche équilibré contenant les mêmes valeurs :

- (1) Déterminer l'ordre de parcours *infixe* de l'arbre. Le stocker dans une liste `ordre`
- (2) Créer un arbre vide nommé `nouveau` (une liste contenant la valeur 0 selon l'implémentation choisie ici). La liste `nouveau` sera réutilisée dans la suite de l'algorithme sans avoir besoin d'être passée en argument de fonctions
- (3) La variable `i` prend la valeur 1 (indice de la racine de `nouveau`)
- (4) Appeler la fonction récursive `construireABR(i, ordre)` qui :
  - (a) Ajoute la valeur `None` à la liste `nouveau` jusqu'à ce qu'elle soit de longueur `i+1`
  - (b) Donne la valeur du « milieu » de `ordre` à `nouveau[i]`
  - (c) Détermine la moitié gauche de `ordre`
  - (d) Si celle-ci est non-vide, appelle `construireABR(2*i, moitié gauche de ordre)`
  - (e) Détermine la moitié droite de `ordre`
  - (f) Si celle-ci est non-vide, appelle `construireABR(2*i+1, moitié droite de ordre)`

La moitié gauche de `ordre` (resp. moitié droite de `ordre`) est la liste des valeurs de `ordre` d'indice strictement inférieur (resp. supérieur) à celui du milieu. On la crée facilement en faisant `ordre[:len(ordre)//2]` (resp. `ordre[(len(ordre)//2+1):]`)

Cette méthode, appliquée à l'ordre de parcours [8, 10, 15, 17] permet de construire l'arbre stocké dans la liste [0, 15, 10, 17, 8].



2. Dessiner l'arbre construit par cette méthode lorsque l'ordre proposé est [20, 21, 22, 23, 24, 25].

On donne ci-dessous le pseudo-code d'une fonction renvoyant l'ordre de parcours infixe d'un arbre fourni sous forme d'une liste python. Cette fonction utilise une pile.

```

1 Fonction infixe(arbre) :
2     p est une Pile vide
3     visites est une liste vide
4     n est égal à 1 # l'indice du nœud étudié dans arbre
5     repetition est égal à Vrai
6     Tant que repetition est Vrai :
7         Tant que n < longueur(arbre) et arbre[n] n'est pas None :
8             Ajouter n à la pile
9             n est égal à l'indice du fils gauche du nœud d'indice n
10        Si la longueur de la pile est nulle :
11            repetition est égal à Faux
12        Sinon :
13            Dépiler une valeur de la pile dans n
14            Ajouter arbre[n] à la fin de visites
15            n est égal à l'indice du fils droit du nœud d'indice n
16    Renvoyer visites
    
```

3. Transcrire cette fonction en python. On pourra utiliser une liste python afin d'implémenter la structure de pile.

4. Écrire en python le code de la fonction construireABR décrite plus haut (point (4) de l'encadré).