Numérique et Science Informatique

Bac Blanc - Corrigé

20 février 2023

Exercice 1 : Base de données 8 points

1. La table `Articles` utilise des clés étrangères des tables `Auteurs` et `Themes`. Si ces dernières sont vides, il n'est pas possible de lier les valeurs et donc on ne peut pas insérer de valeurs.

2. On saisit `INSERT INTO Traitements (article, theme) VALUES (2, 4)`

3. On saisit `UPDATE Auteurs SET nom = "Jèraus" WHERE idAuteur = 2`

4. **a.** Le titre des articles parus après le 1^{er} janvier 2022 inclus :

```sql
SELECT titre
FROM Articles
WHERE dateParution >= 20220101
```

b. Le titre des articles écrits par l'auteur Étienne Zola :

```sql
SELECT titre
FROM Articles
WHERE auteur = 3
```

c. Le nombre d'articles écrits par l'auteur Jacques Pulitzer (présent dans la table `Auteurs` mais on ne connaît pas son `idAuteur`) :

```sql
SELECT count(*)
FROM Articles
JOIN auteurs ON Articles.auteur = Auteurs.idAuteur
WHERE nom LIKE "Pulitzer" AND prenom LIKE "Jacques"
```

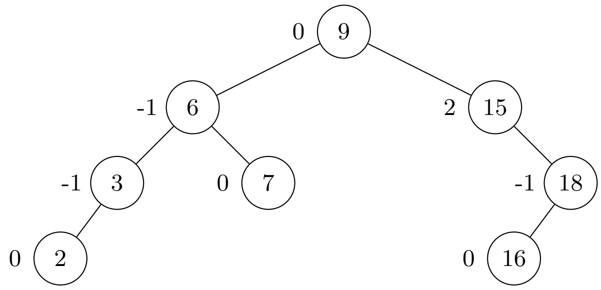
d. Les dates de parution des articles traitant du thème « Sport »

```sql
SELECT date
FROM Articles
JOIN Traitements ON Articles.idArticle = Traitements.article
JOIN Themes ON Traitements.theme = Themes.idTheme
WHERE Themes.themes LIKE "Sport"
```

Exercice 2 : Arbres binaires équilibrés 12 points

Partie A :

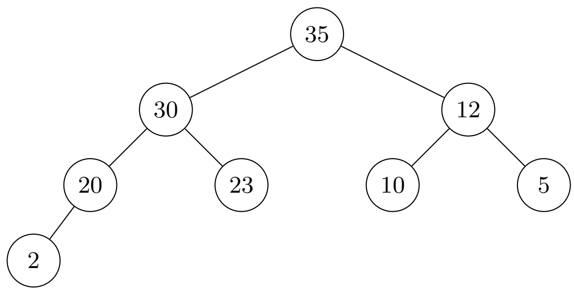
1. a. On obtient :



b. Cet arbre n'est pas équilibré car le nœud de valeur 15 a une balance de 2.

2. a. On obtient `[0, 45, 40, 48, 17, 43, 46, 49, 14, 19]`

b. On obtient :



3. a. `f(arbre, 1)` renvoie 3. En effet, si l'arbre est vide ou si la valeur de sa racine est `None`, on renvoie 0. Dans le cas contraire, on renvoie 1 plus de maximum des résultats des sous-arbres gauches et droits (indices `2*i` et `2*i+1`). On calcule ainsi la hauteur de l'arbre.

b. La fonction `f` permet de calculer la hauteur d'un arbre.

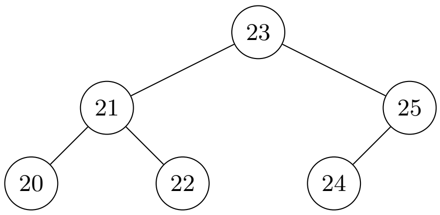
4.

```python
def estEquilibre(arbre: list, i : int) -> bool:
    if i >= len(arbre) or arbre[i] is None:
        return True
    else:
        balance = f(arbre, 2*i+1) - f(arbre, 2*i)
        reponse = balance in [-1, 0, 1]
        return reponse and estEquilibre(arbre,2*i) and estEquilibre(arbre, 2*i+1)
```

Partie B :

1.
- Parcours *préfixe* : 45, 40, 17, 14, 19, 43, 48, 46, 49
- Parcours *infixe* : 14, 17, 19, 40, 43, 45, 46, 48, 49
- Parcours *suffixe* : 14, 19, 17, 43, 40, 46, 49, 48, 45

2. On obtient :



3. On propose :

```python
def infixe(arbre: list) -> list:
    pile = []
    visites = []
    n = 1
    repetition = True
    while repetition :
        while n < len(arbre) and arbre[n] is not None :
            pile.append(n)
            n = 2*n
        if len(pile) == 0 :
            repetition = False
        else :
            n = pile.pop()
            visites.append(arbre[n])
            n = 2*n+1
    return visites
```

4. On propose :

```python
def construitABR(i, ordre):
    while len(nouveau) <= i:
        nouveau.append(None)

    i_milieu = len(ordre)//2
    nouveau[i] = ordre[i_milieu]

    gauche = ordre[:i_milieu]
    if len(gauche) > 0:
        construitABR(2*i, gauche)

    droite = ordre[(i_milieu+1):]
    if len(droite) > 0:
        construitABR(2*i+1, droite)
```

<u>Exercice 3 :</u> Bin-Packing

1. On propose la répartition suivante : [26, 4], [17, 13], [15, 11] et [5]. Il faut 4 boîtes.

2. On fait simplement `len(repartition)`.

3. On propose :

```python
def poidsBoite(boite: list) -> int:
poids = 0
for objet in boite:
poids += objet

return poids
```

4. **a.** On obtient `[8, 2], [3, 1], [9], [7]`.

b. On pourrait faire `[8, 2], [3, 7], [9, 1]`. On utiliserait alors 3 boîtes au lieu de 4. La méthode de la première position n'est pas optimale.

5.

```python
def premierePosition(objets : list, pMaxi : int) -> list:
    # La répartiton
    repartition = []
    # On ajoute une boîte vide
    repartition.append([])

    for objet in objets : # parcours des objets
        ajout = False # permet de savoir si l'objet a été ajouté
        for boite in repartition :
            if poidsBoite(boite) + objet <= pMaxi :
                # l'objet tient dans cette boite
                boite.append(objet) # on l'ajoute
                ajout = True
                break
        if not ajout : # l'objet ne tient dans aucune des premières boîtes...
            repartition.append([objet]) # on l'ajoute dans une nouvelle boîte

    return repartition
```

6. On considère des objets de poids [8, 1, 9, 2] et un poids maximal de 10. En appliquant la méthode de la meilleure position, on obtient la répartition [8, 1], [9], [2] alors qu'une répartition optimale ne fait intervenir que 2 boîtes : [8, 2], [9, 1].

7.

```python
# On "remonte" cette boîte à sa position triée
while i > 0 and poidsBoite(repartition[i]) > poidsBoite(repartition[i-1]) :
    repartition[i], repartition[i-1] = repartition[i-1], repartition[i]
    i = i-1
```

8. Le « *tri par sélection* » est de complexité quadratique $\mathcal{O}(n^2)$ alors que le « *tri fusion* » est de complexité semi-logarithmique $\mathcal{O}(n\log(n))$. Le second est donc plus efficace.

9. On obtient :

Méthode	Répartition (avec les **poids**)
Première position **sans** tri	[1, 2, 1, 3, 3, 1, 1], [10], [10], [6]
Meilleure position **sans** tri	[1, 2, 1, 3, 3, 1, 1], [10], [10], [6]
Première position **avec** tri	[10, 3, 2], [10, 3, 1, 1], [6, 1, 1]
Meilleure position **avec** tri	[10, 3, 2], [10, 3, 1, 1], [6, 1, 1]

10. Les méthodes avec tris nécessitent moins de boîtes : elles semblent plus efficaces.