

[← Index des sujets 2022](#)

22-NSIJ1ME3 : Corrigé

Année : **2022**

Centre : **Métropole session de septembre**

Jour : **1**

Enoncé : [!\[\]\(e3f8612927870f2e0f9f5989e6dd3064_img.jpg\) PDF](#)

1. Exercice 1

algorithme, arbres binaires de recherche et leurs parcours

Partie A : Préambule

- Seul l'arbre 1 est un arbre binaire de recherche. En effet, dans l'arbre 2 la clé 4 apparaît dans le sous arbre droit de la racine de clé 4, or les clés du sous arbre droit doivent être *strictement supérieurs* à la racine. Pour la même raison, l'arbre 3 n'est pas non plus un arbre binaire de recherche, la valeur 1 figure dans le sous arbre droit du noeud d'étiquette 2.

Partie B : Analyse

- Le plus petit élément se trouve sur la feuille la plus à gauche de l'arbre. En effet en descendant récursivement à gauche, on trouve à chaque étape une valeur inférieure à toutes celles se trouvant à droite.



Attention

La question demande **le** plus petit élément, or les valeurs présentes dans un arbre binaire de recherche ne sont pas forcément uniques. A titre d'exemple, dans l'arbre binaire de recherche suivant, la plus petite valeur apparaît 2 fois :

```
graph TD
S5(5) --> S6(5)
S5 --> S8(8)
```

- Si l'arbre est vide alors il ne contient pas la clé cherchée. Sinon trois cas se présentent : la clé cherchée est la racine et on renvoie `True`, la clé cherchée est inférieure à la racine et on relance la recherche dans le sous arbre gauche (là où se trouvent les valeurs inférieures à la racine), la clé cherchée est supérieure à la racine et on relance la recherche dans le sous arbre droit.



Script Python

```
def RechercheValeur(cle, abr):
    if est_vide(abr):
        return False
    if racine(abr)==cle:
        return True
    elif racine(abr)>cle:
        return RechercheValeur(cle, sous_arbre_gauche(a))
    else:
        return RechercheValeur(cle, sous_arbre_droit(a))
```

2. a. C'est un parcours en profondeur infixe c'est à dire qu'on liste de façon récursive les clé du sous arbre gauche, puis la racine, puis les clé du sous arbre droit.
- b. En parcours préfixe, on liste récursivement la racine puis les noeuds du sous arbre gauche puis ceux du sous arbre droit. On obtient donc : 7 – 2 – 1 – 5 – 3 – 6 – 10 – 8 – 9.
- c. En parcours suffixe, on liste récursivement les noeuds du sous arbre gauche, ceux du sous arbre droit et enfin la racine. Ici, on obtient : 1 – 3 – 6 – 5 – 2 – 9 – 8 – 10 – 7.
- d. Le parcours en largeur correspond à l'ordre de la lecture (de gauche à droite et de haut en bas). Ici on obtient : 7 – 2 – 10 – 1 – 5 – 8 – 3 – 5 – 9.

2. Exercice 2

programmation orientée objet, itérations et récursivité



Bug

- Le code proposé définit pour un objet de la classe `Villa` un attribut `nom` ainsi qu'une méthode `nom`, ce qui n'est pas possible. La méthode est un *getter* qu'on pourrait renommer par exemple en `get_nom`.
- L'attribut `eqCuis` ne prend que deux valeurs `"eq"` ou `"noneq"`, il serait donc plus judicieux d'en faire un booléen.

1. a. La liste `v` contient 5 éléments (elle était initialement vide et on y ajouté 5 éléments à l'aide de 5 `append`)
- b. `v[1]` est le deuxième élément de la liste `v` c'est à dire un objet de la classe `Villa` sa méthode `self.nom` renvoie l'attribut `nom` c'est à dire `"Les goélands"`.
- c. Le calcul de la surface s'effectue naturellement en sommant la surface de chacune des pièces de la villa :



```
def surface(self):
    return self.sejour.sup() + self.ch1.sup() + self.ch2.sup()
```

2. On parcours la liste `v` des villas et on affiche les noms de celles ayant une cuisine équipée (la méthode `equip` renvoie `"eq"` lorsque la cuisine est équipée).

Script Python

```
for villa in v:
    if villa.equip() == "eq":
        print(villa.nom())
```

3. Un appel récursif est caractérisé par *appel d'une fonction par elle-même*

4. Script Python

```
def max_surface(v):
    if len(v) == 1:
        return v[0].nom()
    if v[0].surface() > v[1].surface():
        v.pop(1)
    else:
        v.pop(0)
    return max_surface(v)
```

Remarque

Le type `list` étant mutable, la fonction ci-dessous vide en même temps la liste `v` qui contiendra à la fin un unique élément (celui ayant la surface maximale)

3. Exercice 3

bases de données relationnelles et langage SQL

1. L'attribut `Num_objet` peut être choisi comme clé primaire car il est unique pour chacun des enregistrements présent dans la table.
2. Le schéma relationnel de la table `Type` s'écrit :
Type(Type_objet : TEXT, Libelle_Objet : TEXT)
3. Seule l'instruction **b)** ne provoque pas d'erreur dans les autres cas une erreur se produit sur le type des données :
 - "8" au lieu de 8 pour l'instruction a)
 - WISEA J085510 au lieu de "WISEA J085510" pour l'instruction c)
 - "133.781" au lieu de 133.781 pour l'instruction d)
4. Cette instruction ne fonctionne pas car `Type_Objet` est une clé primaire or un enregistrement ayant la valeur "BD" existe déjà dans la table.
5. Cette requête renvoie les champs "Nom_Objet" et `Parallaxe` pour les objets de type "Planet" ce qui donne sur l'extrait présenté :

Nom_objet	Parallaxe
WISEA J085510	133.781

Nom_objet	Parallaxe
Proxima Cen b	768,067
HF 95735	392,753

6. Requête SQL

```
SELECT Gaia.Nom_systeme, Gaia.Nom_objet, Type.Libelle_Objet FROM Gaia
JOIN Type ON Gaia.Type_Objet = Type.Type_Objet
WHERE Gaia.Parallaxe > 400 and Gaia.Type_Objet = "***";
```

7. a)

Requête SQL

```
INSERT INTO Type VALUES ('ST', 'Etoile')
```

b)

Requête SQL

```
UPDATE Gaia SET Type_Objet = "ST" WHERE Type_Objet = "***";
```

Une fois la requête précédente effectuée, on peut supprimer le type objet *** de la table type :

Requête SQL

```
DELETE FROM Type WHERE Type_Objet = "***";
```

4. Exercice 4

architecture matérielle, gestion de processus et réseaux

1. C'est le schéma a) dans lequel aussi bien l'unité de contrôle (uc) que l'unité arithmétique et logique (UAL) peuvent échanger des informations avec la mémoire. Et aussi dans lequel c'est l'UAL qui reçoit les entrées et renvoie les sorties.
2. On peut proposer n'importe quelle adresse du type 192.168.10.x avec x entre 1 et 254. En effet la partie réseau contient 24 bits et 192.168.10.0 est réservé au réseau et 192.168.10.255 au broadcast
3. On peut connecter 254 machines au réseau LAN01 (voir remarque précédente)
4. Le rôle d'un switch est de connecter plusieurs ordinateurs d'un même sous réseau. Dans l'exemple du réseau ci-dessus, le switch permet de connecter entre eux les ordinateurs du sous réseau LAN01.
5. Le rôle d'un routeur est de connecter plusieurs sous réseaux entre eux. Dans l'exemple du réseau ci-dessus, le routeur R1 permet de connecter les ordinateurs du réseau LAN01 à Internet

6. Le tableau complété :

Destination	Passerelle	Métrique
192.168.10.0/24	0.0.0.0	0
2.100.40.0/24	2.100.40.1	1
3.100.30.0/24	3.100.30.2	1
4.10.10.0/24	4.10.10.2	1
4.20.10.0/24	3.100.30.2	2
7.30.40.0/24	3.100.30.2	3
6.10.30.0/24	2.100.40.1	2
90.10.20.0/24	2.100.40.1	2

7. La dernière ligne du tableau précédent serait modifiée en

Destination	Passerelle	Métrique
90.10.20.0/24	4.10.10.2	4

5. Exercice 5

notion de file et programmation en Python

1. C'est **la situation 2** qui est associé à une structure de file puisque le premier travail envoyé sera aussi le premier à être imprimé.
2. a. Val contiendra Prioritaire , V contiendra Client3, Client2, Client1 et F contiendra Client4 .
- b.

Script Python

```
def longueur_file(F):
    V = creer_file()
    n = 0
    while not est_vide(F):
        n = n + 1
        val = defiler(F)
        enfiler(V, val)
    while not est_vide(F):
        val = defiler(V)
```

```
    enfiler(F, val)
    return n
```

- c. On reprend la fonction précédente en ajoutant une condition de façon à ne compter que les personnes prioritaires

Script Python

```
def longueur_file(F):
    V = creer_file()
    prio = 0
    while not est_vide(F):
        val = defiler(F)
        if val == "Prioritaire":
            prio = prio + 1
        enfiler(V, val)
    while not est_vide(F):
        val = defiler(V)
        enfiler(F, val)
    return prio
```