**Sujet 1 (2023)**

**Exercice 1 :**

1. La table groupes a comme attributs : nom, style et nb\_pers.
2. L’attibut nom de la table musiciens ne peut pas être une clé primaire car on retrouve de fois la valeur « Parkeur » dans nom or une clé primaire est unique et ne peut pas être deux fois.
3. Cette requête affiche les noms des groupes qui ont un style « Latin Jazz ».
4. UPDATE concerts SET heure\_fin = ‘22h00’ WHERE idconc = 36;
5. Voici ma requête :

SELECT nom.groupes

FROM concerts

INNER JOIN groupes

ON groupes.idgrp = concerts.idgrp

WHERE scene = 1 ;

1. INSERT INTO groupes VALUES(15, 'Smooth Jazz Fourplay', ‘Free Jazz’, 4)
2. def recherche\_nom(table):

resultat = []

for element in table:

if element["nb\_concerts"] >= 4:

resultat.append(element["nom"])

return resultat

**Exercice 2 :**

1. Cette configuration est celle de Alice car le masque de son réseau est de 2 alors que celui de Bob est de 1 or on sait que celui de la configuration est de 2.
2. Coût = 10 000 Mbit/s / 1 000 Mbit/s = 10

Le coût du réseau WAN8 est de 10.

|  |  |  |
| --- | --- | --- |
| **Routeur R6** | | |
| Destination | Pass | Count |
| LAN1 | R5 | 21 |
| LAN2 | - | - |
| WAN1 | R5 | 11 |
| WAN2 | R5 | 20 |
| WAN3 | R5 | 11 |
| WAN4 |  |  |
| WAN5 |  |  |
| WAN6 | - | - |
| WAN7 | - | - |
| WAN8 |  |  |

**Exercice 3 :**

1. C’est une file car le robot fait la première tâche qu’il a mémorisé et ainsi de suite. Donc la première tache mémoriser et la première tache rentrer soit First in First out.
2. Le terme qui correspond au nombre total de nœuds est la taille.
3. Le terme qui correspond au nœud représentant la tâche restant à effectuer la plus ancienne la racine.
4. Le terme au correspond au nœud représentant la dernière tâche mémorisée est une feuille.
5. Les attributs de la classe nœud sont tache, indice, gauche et droite.
6. La méthode insere est dite récursive car elle s’appelle elle-même dans sa méthode à la ligne 27 et 29. Elle se termine bien car la fonction va continuer de s’exécuter jusqu’à trouver l’endroit pour placer le nouveau nœud donc si l’arbre n’est pas infini, la fonction se terminera bien un jour.

Justification du prof : Cette méthode se relance soit dans le sous arbre droit soit dans le sous arbre gauche. Elle tombera sur une feuille de l’arbre la fessant ajouter et la elle se terminera.

1. Le symbole de comparaison est > .
2. 

7

16

5

11

1. def est\_present(self, indice\_recherche) :

"""renvoie True si l’indice de priorité indice\_recherche

(int) passé en paramètre est déjà l’indice d’un noeud

de l’arbre, False sinon"""

if self.est\_vide():

return False

elif self.racine.indice == indice\_recherche:

return True

elif self.racine.indice > indice\_recherche.indice:

return self.racine.gauche.est\_present(indice\_recherche)

else:

return self.racine.droite.est\_present(indice\_recherche)

1. Voici le parcours infixe de la figure 1, si on n’utilise pas la figure 1 de l’exercice 3 D :

6, 8, 10, 12, 13, 14

Voici le parcours infixe de la figure 1, si on utilise la figure 1 de l’exercice 3 D :

5, 6, 10, 8, 7, 11, 12, 13, 16, 14

1. Il suffit de prendre la dernière valeur de la liste obtenue à l’aide d’un parcours infixe de l’arbre.
2. def tache\_prioritaire(self):

"""renvoie la tache du noeud situé le plus

à gauche de l'ABR supposé non vide"""

if self.racine.gauche.est\_vide(): #pas de noeud plus à gauche

return self.racine.tache

else:

return self.racine.gauche.tache\_prioritaire()

1. Etape 1 :

14

Etape 2 :

14

11

Etape 3 :

14