WebPédago
Partagez la connaissance !

BAC GÉNÉRAL
Correction épreuve de NSI

Exercice 1
(erreur ligne 3 de la représentation du graphe)

1. La ligne 9 associe aux predecesseurs du site 2 une liste vide, le nœud 2 n'ayant pas de predecesseurs.
2. s4.predecesseurs = [(s2,2) (s1, 1)]
   s5.predecesseurs = [(s3,3) (s1, 2), (s 4,6)]
3. >>> s2.successeurs[1][1]
   5

On va chercher le deuxième item du successeur d'indice N° 1 (le 2e) des sucesseurs de s2.

4. La popularité du site 1 est 6. Il faut ajouter les arrêtes des nœuds dont il est sucesseur.

5. def calculPopularite(self) :
   « »"on ajoute les valeurs des arrêtes dont le site est sucesseurs, c'est-à-dire de ces predecesseurs"" »

```
1. La ligne 9 associe aux predecesseurs du site 2 une liste vide, le nœud 2 n'ayant pas de predecesseurs.
2. → s4.predecesseurs = [(s2,2) (s1, 1)]
   → s5.predecesseurs = [(s3,3) (s1, 2), (s 4,6)]
3. → >>> s2.successeurs[1][1]
   → 5
popularite = 0
   for site, valeur in self.predecesseurs :
      popularite+=valeur
   return popularite

   #une autre manière de faire :
   for predecesseur in self.predecesseurs :
      valeur = predecesseur[1]
      popularite+=valeur
   return popularite
```

6. On ajoute à la fin (append) et on pop la première (pop[0]), ceci correspond donc à une **file**.

7. Pour un site donné, on va ajouter chaque successeur et parcourir les successeurs des successeurs (si existants). Donc, on aura dans la file d'abord les fils, puis les fils des fils.
Il s'agit donc d'un parcours en largeur.

8. >>> parcoursGraphe(s1)
[s1, s3, s4, s5]

9. if site.popularite > maxPopularite :
siteLePlusPopulaire = site
maxPopularite = site.popularite
10. >>> lePlusPopulaire(parcoursGraphe(s1)).nom
'site3'
Le site 3 a la plus grande popularité (12) et son nom est la chaîne de caractère 'site3'.

11. Il s'agit d'une recherche de maximum. Un parcours de graphe est appelé. Il est assez optimisé, vu qu'il ne repasse jamais deux fois par le même site (coloration de graphe). Python peut traiter sans soucis un graphe de cette taille, vu que la complexité n'est pas exponentielle.

Exercice 2
Partie A
1. Un SGBD permet de vérifier l'unicité d'un enregistrement (il ne peut exister deux clés primaires identiques) et il permet aussi de sécuriser les accès (via des utilisateurs différents par exemple).

2. Si on utilise le protocole RIP, on doit minimiser le nombre de routeurs traversés :
Bureau N° 1 → B → E → A → prestataire

3. Le protocole OSPF minime le « coût » de la transmission. Plus le chiffre est élevé, plus le coût est élevé.
Route possible : Bureau N° 2 → C → I → H → F → D → A → prestataire
Route possible : Bureau N° 2 → C → I → G → F → D → A → prestataire
Les coûts des routes sont identiques : 2,3

Partie B
4. Il peut exister plusieurs clients avec le même nom ou prénom ou plusieurs clients avec la même date de naissance ou encore plusieurs clients venant du même pays. Aucun ne peut donc être une clé primaire car elle doit n'apparaître qu'une fois en tout dans une table.
On crée donc un numéro à un client, unique.

5. Une clé étrangère est une entrée dans une table qui définit de manière unique un enregistrement dans une autre table. C'est souvent une clé primaire d'une autre table.

Ici, dans la table « reservations », il y a 2 clés étrangères, id_client qui référence le numéro du client et nom_croisière, qui référence le nom de la croisière.
Dans la table « croisières », il y a 4 clés étrangères, chacune faisant référence au nom d'une « ville ».

6. L'erreur provient du fait que « puerto sebo » n'existe pas. Il s'agit de « puerto Saibo ».

Il faut remplacer « Puerto sebo » par « puerto Saibo » dans la requête.

Partie C

7. Le gestionnaire a voulu récupérer l'ensemble des réservations faites par ce client.
La première requête permet de récupérer son id.
La deuxième permet de lister les réservations correspondant à ce numéro.

8. On peut utiliser la syntaxe avec JOIN :

SELECT id_reservation FROM reservations
JOIN clients ON reservations.id_client = clients.id_client
WHERE clients.nom = 'Barc' AND clients.prenom = 'Jean' and client.date_naissance ='1972/06/29' and clients.pays = 'Allemagne' ;

9.
UPDATE reservations
```
SET nom_croisiere =  « Croisière Puerto»
WHERE id_reservation = 20456;
```

10. SELECT nom,prenom,date_naissance
FROM clients
JOIN reservations ON reservations.id_client = clients.id_client
JOIN croisieres ON croisieres.nom = reservations.nom_croisiere
WHERE reservations.nom_croisiere = 'Croisiere Piano' OR reservations.nom_croisiere = 'Croisiere Puerto'

Exercice 3

Partie A

1. >>> chien 40 = Chien (40, « Duke », « wheel dog », 10)

2. def changer_role(self, nouveau_role) :
        self.role = nouveau_role
3. >>> chien40.changer_role(« leader »)

Partie B

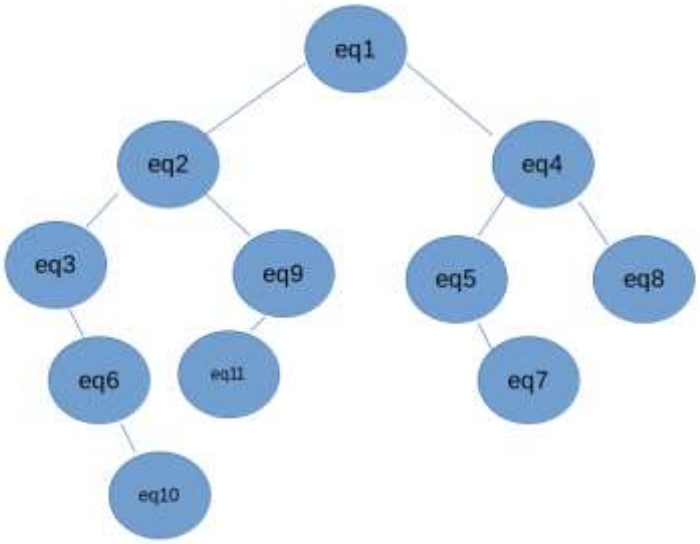
4. def retirer_chien(self, numero) :
   #on parcourt l'ensemble des chiens et on retire celui qui a pour id numero.
   #on peut le faire de plusieurs façons, soit on cherche la position du chien dans la liste et on utilise la méthode ».remove(id) » soit on crée une nouvelle liste. C'est ce qui est proposé ici :

```
   nouvelle_liste_chien = []
   for chien in self.liste_chiens :
      if chien.id_chien!=numero :
         nouvelle_liste_chien.append(chien)
   self.liste_chiens = nouvelle_liste_chien
```

5. >>> eq11.retirer_chien(46)

6. >>> convert('4h36')
4,6
7. def temps_course(equipe) :
    temps_total = 0
    for temps in equipe.liste_temps :
        temps_total+=convert(temps)
    return temps_total

Partie C
8.



9. Dans un arbre binaire de recherche, le parcours infixe permet de classer du plus petit au plus grand.

10. La fonction inserer() est une fonction récursive car elle s'appelle elle-même (ligne 8).

11.
if convert(eq.temps_etape)< convert(arb.racine.temps_etape) :
    if arb.gauche is None :
        arb.gauche = **Noeud(eq)**
    else :
        inserer(**arb.gauche**, eq) #appel recursif sur le sous arbre de gauche, vu que le temps est plus petit.
else :
    if arb.droit is None :
        arb.droit = Noeud(eq)

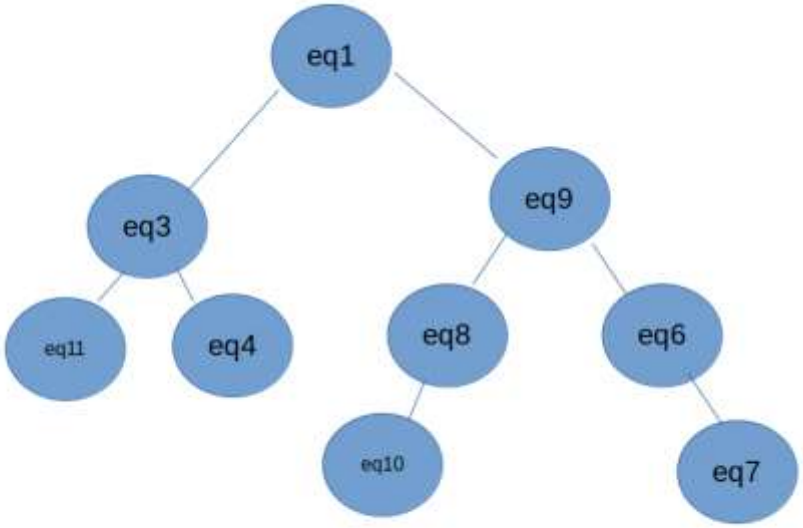
```
    else :
        inserer(arb.droit, eq)
```

12.

```
def est_gagnante(arbre) :
    if arbre.gauche == None :
        return arbre.racine
    else :
        return est_gagnante(arbre.gauche)
```

Partie D

13.



14.

```
def recherche(arbre, equipe) :
    if arbre is None :
        return False
    if arbre.racine==equipe :
        return True

    if recherche(arbre.gauche, equipe) or recherche(arbre.droit, equipe) :
        return True
    else :
        return False
```