Métropole – 2024 – sujet1 - Correction

Exercice 3 (8 points)

Partie A – Classe `Chien`

1. Instanciation de `chien40`

```
chien40 = Chien(40, "Duke", "wheel dog", 10)
```

👉 *Commentaire : on respecte l'ordre des paramètres du constructeur : id_chien, nom, role, id_prop.*

---

2. Méthode `changer_role`

```
def changer_role(self, nouveau_role):
    self.role = nouveau_role
```

👉 *Commentaire : il suffit de modifier l'attribut role de l'objet courant (self).*

---

3. Modifier le rôle de Duke en "leader"

```
chien40.changer_role("leader")
```

👉 *Commentaire : on appelle la méthode sur l'objet concerné.*

Partie B – Classe `Equipe`

4. Méthode `retirer_chien`

```
def retirer_chien(self, numero):
    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
```

👉 *Commentaire : Cela peut-être fait de plusieurs façons. Soit on parcourt la liste et on supprime le chien correspondant à son identifiant, soito n crée une nouvelle liste. On retient ici la méthode avec le plus classique* `append`.

---

5. Retirer Helka (chien 46)

```
eq11.retirer_chien(46)
```

👉 *Commentaire : on passe le numéro d'inscription en paramètre.*

---

6. Résultat de `convert('4h36')`

Calcul effectué :

- heure = 4
- minutes = 36
- 36 / 60 = 0,6

Résultat :

```
4.6
```

👉 *Commentaire : la fonction renvoie un nombre décimal représentant la durée en heures.*

---

7. Fonction `temps_course`

```python
def temps_course(equipe):
    total = 0
    for temps in equipe.liste_temps:
        total += convert(temps)
    return total
```

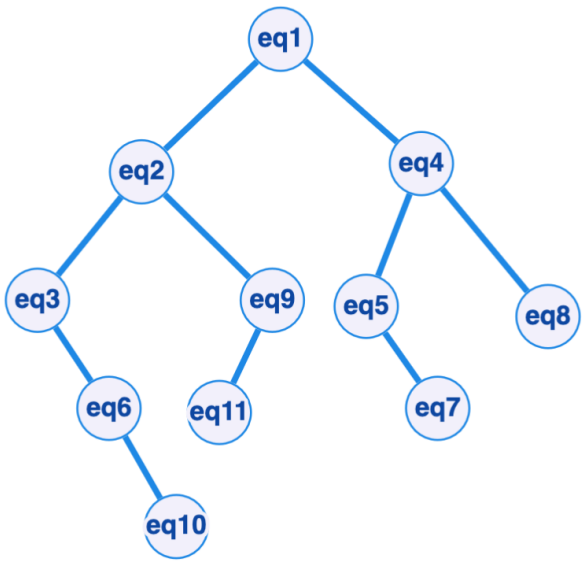
👉 *Commentaire : on additionne les temps convertis en heures décimales.*

Partie C – Arbre Binaire de Recherche

8. Construction de l'ABR

Rappel :

- Sous-arbre gauche → équipes plus rapides
- Sous-arbre droit → équipes plus lentes ou égales



👉 *Commentaire : on compare les temps à chaque insertion.*

---

9. Parcours donnant le classement

Il faut effectuer un **parcours infixe** :

- gauche
- racine
- droite

👉 *Commentaire : dans un ABR, le parcours infixe donne les valeurs triées.*

---

10. Pourquoi `inserer` est récursive ?

Parce qu'elle **s'appelle elle-même** jusqu'à atteindre un sous-arbre vide (cas de base).

👉 *Commentaire : chaque appel traite un sous-arbre plus petit.*

---

11. Fonction `inserer` (complétée)

Version classique :

```
def inserer(arb, eq):
    if eq.temps_etape < arb.racine.temps_etape:
        if arb.gauche == None:
            arb.gauche = Noeud(eq)
        else:
            inserer(arb.gauche, eq)
    else:
        if arb.droit == None:
            arb.droit = Noeud(eq)
        else:
            inserer(arb.droit, eq)
```

👉 *Commentaire : on descend récursivement jusqu'à trouver une position libre.*

---

12. Fonction `est_gagnante`

Le gagnant est l'équipe la plus rapide → minimum → tout à gauche.

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

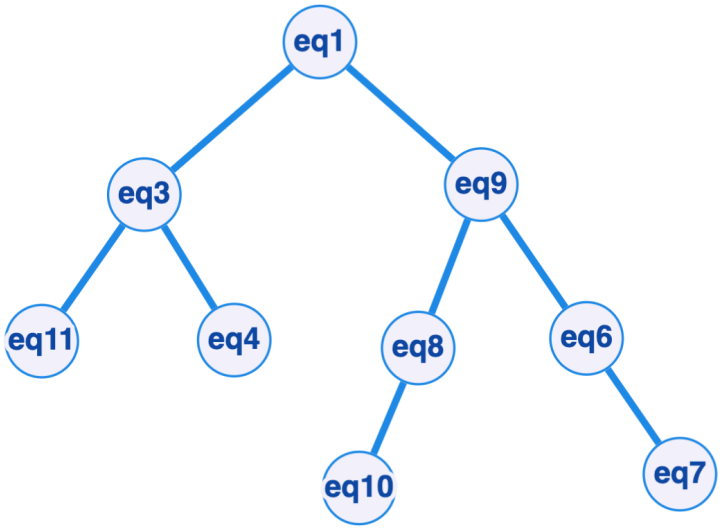
👉 *Commentaire : on descend toujours à gauche jusqu'à la feuille la plus à gauche.*

Partie D – Suppressions

13. Suppression de eq2 et eq5

- eq2 possède deux fils → on le remplace par son successeur
- eq5 possède un seul fils (eq7)

Nouvelle structure:



👉 *Commentaire : on applique les règles standards de suppression dans un ABR.*

---

14. Fonction `rechercher`

```
def rechercher(arbre, equipe):
    if arbre == None:
        return False
    if equipe.num_dossard == arbre.racine.num_dossard:
        return True
    elif equipe.temps_etape < arbre.racine.temps_etape:
        return rechercher(arbre.gauche, equipe)
    else:
        return rechercher(arbre.droit, equipe)
```

👉 *Commentaire : on applique le principe de recherche dichotomique propre aux ABR.*
