Polynésie – 2024 – sujet2 - Correction

Exercice 2 (6 points)

Partie A

1. Texte à compléter

Le nœud initial est appelé **racine**.

Un nœud qui n'a pas de fils est appelé **feuille**.

Un arbre binaire est un arbre dans lequel chaque nœud a **au maximum** deux fils.

Un arbre binaire de recherche est un arbre binaire dans lequel tout nœud est associé à une clé qui est:

- supérieure à chaque clé de son **sous-arbre gauche** ;
- inférieure à chaque clé de son **sous-arbre droit**.

👉 *Commentaire : dans un ABR, toutes les valeurs du sous-arbre gauche sont inférieures à la racine, celles du sous-arbre droit sont supérieures.*

---

2. Parcours préfixe de l'arbre n°1

Rappel : préfixe = racine → gauche → droite.

On liste les clés en suivant cet ordre: 1 - 0 - 2 - 3 - 4 - 5 -6

👉 *Commentaire : toujours commencer par la racine.*

---

3. Parcours suffixe (postfixe) de l'arbre n°2

Suffixe = gauche → droite → racine.

On liste les clés en suivant cet ordre: 0 - 1 - 2 - 6 - 5 - 4 - 3

👉 *Commentaire : la racine est visitée en dernier.*

---

4. Parcours infixe de l'arbre n°3

Infixe = gauche → racine → droite.

Dans un ABR, le parcours infixe donne les clés **dans l'ordre croissant**: 0 - 1 - 2 - 3 - 4 -5 - 6

👉 *Commentaire : propriété fondamentale des ABR.*

---

5. Construction des trois arbres

Exemple possible (plusieurs réponses possibles) :

```
arbre_no1 = ABR()
arbre_no2 = ABR()
arbre_no3 = ABR()

for cle_a_inserer in [1, 0, 2, 3, 4, 5, 6]
    arbre_no1.inserer(cle_a_inserer)

for cle_a_inserer in [3, 2, 4, 1, 5, 0, 6]:
    arbre_no2.inserer(cle_a_inserer)

for cle_a_inserer in [3, 1, 5, 0, 2, 4, 6]:
    arbre_no3.inserer(cle_a_inserer)
```

👉 *Commentaire : l'ordre d'insertion détermine la structure de l'ABR.*

---

6. Hauteur des trois arbres

Rappel :

```
hauteur = 1 + max(hauteur_gauche, hauteur_droite)
```

La hauteur correspond au nombre d'arêtes du plus long chemin.

👉 arbre_no1 : hauteur = 5
👉 arbre_no2 : hauteur = 3
👉 arbre_no3 : hauteur = 3

---

7. Méthode `est_present`

```
def est_present(self, cle_a_rechercher):
    if self.est_vide():
        return False
    elif cle_a_rechercher == self.cle():
        return True
    elif cle_a_rechercher < self.cle():
        return self.sag().est_present(cle_a_rechercher)
    else:
        return self.sad().est_present(cle_a_rechercher)
```

👉 *Commentaire : on exploite la propriété d'ordre de l'ABR.*

---

8. Moins d'appels récursifs

`arbre_no3.est_present(7)` car la recherche sera la plus rapide dans l'arbre le plus équilibré. L'arbre le plus équilibré nécessite le moins d'appels récursifs.

👉 *Commentaire : complexité moyenne O(log n) pour un arbre équilibré, O(n) pour un arbre dégénéré. Petite erreur d'énoncé: il faut lire* `est_present` et non `est_presente`.

Partie B

9. Arbre partiellement équilibré

Un arbre est partiellement équilibré si :

`| hauteur(sag) - hauteur(sad) | ≤ 1`

👉 *Commentaire : seule la racine est testée.*

---

10. Deux arbres partiellement équilibrés

On vérifie uniquement la différence de hauteur à la racine.

👉 Deux arbres respectent la condition. Arbre_1 n'est pas partiellement équilibré, car la hauteur du sous-arbre gauche de 1 est égale à 0 alors que la hauteur du sous-arbre droit de 1 est égale à 4.

Arbre_2 et Arbre_3 sont partiellement équilibrés.

👉 *Commentaire : même si les sous-arbres ne sont pas équilibrés.*

---

11. Un seul arbre équilibré

Un arbre équilibré doit :

- être partiellement équilibré
- ET avoir ses sous-arbres eux-mêmes équilibrés

Seul Arbre_3 est équilibré

👉 *Commentaire : c'est une définition récursive.*

---

12. Méthode `est_equilibre`

```
def est_equilibre(self):
    if self.est_vide() :
        return True
    else :
        return self.est_partiellement_equilibre() and
self.sag().est_equilibre() and self.sad().est_equilibre()
```
