Sujet Zéro – 2023 – sujet1 - Correction

Exercice 3 (6 points)

---

1. Structure de données : file ou pile

Si les tâches sont extraites dans le même ordre qu'elles ont été mémorisées, il s'agit du comportement d'une **file (FIFO : First In, First Out)**.

👉 *Commentaire : Une file suit le principe « premier entré, premier sorti », contrairement à une pile qui fonctionne en LIFO.*

---

2. Vocabulaire des arbres binaires

a. Le nombre total de nœuds correspond à **la taille de l'arbre**.
b. La tâche la plus ancienne correspond à **la racine**.
c. La tâche la plus récente correspond à **une feuille**.

👉 *Commentaire : Dans un ABR construit par insertions successives, le premier élément inséré devient la racine et le dernier est nécessairement une feuille.*

---

3. ABR

a. Programmation objet : attributs d'une classe

Les attributs de la classe `Noeud` sont :

- `tache`
- `indice`
- `gauche`
- `droit`

👉 *Commentaire : Un nœud d'ABR contient sa valeur et deux références vers ses sous-arbres.*

---

b. Récursivité

La méthode `insere` est récursive car elle s'appelle elle-même sur un sous-arbre (`gauche` ou `droit`).

Elle se termine car à chaque appel on descend d'un niveau dans l'arbre jusqu'à atteindre un arbre vide.

👉 *Commentaire : La condition d'arrêt est atteinte lorsqu'on arrive sur un sous-arbre vide.*

---

c. Propriété d'un arbre binaire de recherche

Le symbole manquant est :

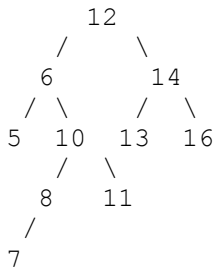
> (strictement supérieur)

👉 *Commentaire : Dans un ABR, les valeurs strictement inférieures vont dans le sous-arbre gauche.*

---

d. Insertion dans un arbre binaire de recherche

Après insertion de 11, 5, 16 et 7 :



👉 *Commentaire : À chaque insertion, on compare la valeur à la racine puis on descend récursivement à gauche ou à droite.*

---

4. Recherche dans un arbre binaire de recherche

```
def est_present(self, indice_recherche):
    """renvoie True si l'indice est présent, False sinon"""
    if self.est_vide():
        return False
    if indice_recherche == self.racine.indice:
        return True
    elif indice_recherche < self.racine.indice:
        return self.racine.gauche.est_present(indice_recherche)
    else:
        return self.racine.droit.est_present(indice_recherche)
```

👉 *Commentaire : Grâce à la propriété d'ordre de l'ABR, on ne parcourt qu'un seul sous-arbre à chaque étape.*

---

5. Parcours

a. Parcours infixe d'un arbre binaire de recherche

Ordre obtenu :

6, 8, 10, 12, 13, 14

👉 *Commentaire : Un parcours infixe d'un ABR renvoie les valeurs dans l'ordre croissant.*

---

b. Exploitation du parcours infixe

La première valeur obtenue correspond à la tâche prioritaire.

👉 *Commentaire : La plus petite valeur est située dans le nœud le plus à gauche.*

---

6. Recherche de la valeur minimale dans un arbre binaire de recherche

```
def tache_prioritaire(self):
    """renvoie la tache du noeud situé le plus à gauche"""
    if self.racine.gauche.est_vide():
        return self.racine.tache
    else:
        return self.racine.gauche.tache_prioritaire()
```

👉 *Commentaire : On descend récursivement vers la gauche jusqu'à atteindre un nœud sans fils gauche.*

---

7. Insertion et suppression dans un arbre binaire de recherche

Étape 1 : insertion 14

```
14
```

Étape 2 : insertion 11

```
   14
  /
11
```

Étape 3 : insertion 8

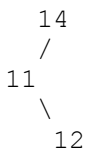
```
   14
   /
  11
 /
8
```

Étape 4 : suppression 8

```
   14
  /
11
```

👉 *Commentaire : 8 est une feuille, elle est supprimée directement.*

Étape 5 : insertion 12

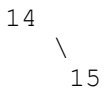


Étape 6 : suppression 11

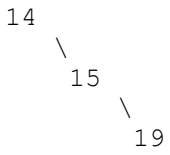
👉 *Commentaire : 11 possède un sous-arbre droit, il est remplacé par celui-ci.*

Étape 7 : suppression 12

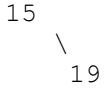
Étape 8 : insertion 15



Étape 9 : insertion 19



Étape 10 : suppression 14



👉 *Commentaire : La racine est remplacée par son sous-arbre droit non vide.*