Métropole – 2023 – sujet2 - Correction

Exercice 3 (6 points)

PARTIE 1 – Arbre binaire de recherche

1. Taille et hauteur de l'arbre

Arbre donné :

Racine : `'dfifi'`
Sous-arbre gauche : `'annieji'`
Sous-arbre droit : `'helene'`
Sous-arbre gauche de `'annieji'` : `'aalice'`
Sous-arbre droit de `'annieji'` : `'celine'`

Nombre total de nœuds :

- dfifi
- annieji
- helene
- aalice
- celine

Taille = 5

Hauteur :

- Chemin le plus long :
  dfifi → annieji → aalice
  (ou dfifi → annieji → celine)

Cela fait 3 nœuds.

Hauteur = 3

👉 *Commentaire : La hauteur correspond au nombre de nœuds du plus long chemin racine-feuille.*

---

2. Ajout de 'davidbg' puis 'papicoeur'

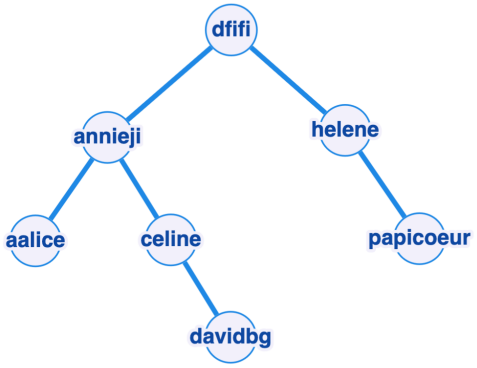
Ajout de `'davidbg'`

Comparaisons :

- davidbg < dfifi → on va à gauche
- davidbg > annieji → on va à droite
- davidbg > celine → insertion à droite de celine

Ajout de `'papicoeur'`

- papicoeur > dfifi → droite
- papicoeur > helene → insertion à droite de helene



👉 *Commentaire : Dans un ABR, on compare toujours lexicographiquement et on descend récursivement.*

---

3. Parcours donnant l'ordre lexicographique

Réponse :

👉 **C – Parcours en profondeur dans l'ordre infixe**

👉 *Commentaire : Dans un ABR, le parcours infixe (gauche → racine → droite) renvoie les valeurs triées.*

---

4. Méthode récursive present

Code complété :

```
def present(self, identifiant):
    if self.est_vide():
        return False
    elif self.racine() == identifiant:
        return True
    elif self.racine() < identifiant:
        return self.sd().present(identifiant)
    else:
        return self.sg().present(identifiant)
```

👉 *Commentaire : Si identifiant est plus grand que la racine, on cherche dans le sous-arbre droit, sinon dans le sous-arbre gauche.*

PARTIE 2 – Files

5.

a. Résultat de est_vide(f1)

La file f1 contient des éléments.

👉 Résultat : **False**

👉 *Commentaire : est_vide renvoie True uniquement si la file ne contient aucun élément.*

---

b. État de f1 après defiler(f1)

L'élément en tête (à droite) est retiré.

La file perd son élément le plus à droite: 'bac' → 'NSI' → '2023'

👉 *Commentaire : defiler retire toujours l'élément en tête de file (à droite ici).*

---

c. État de f2 après la boucle

Liste : `['castor', 'python', 'poule']`

On enfile dans cet ordre.

File obtenue : castor → python → poule
(poule est en tête)

👉 *Commentaire : enfiler ajoute toujours à la queue (à gauche ici).*

---

6. Fonction longueur

Code complété :

```
def longueur(f):
    resultat = 0
    g = creer_file()
    while not(est_vide(f)):
        elt = defiler(f)
        resultat = resultat + 1
        enfiler(g, elt)
    while not(est_vide(g)):
        enfiler(f, defiler(g))
    return resultat
```

👉 *Commentaire : On vide temporairement f dans g pour compter les éléments, puis on restaure f.*

---

7. Mot de passe validé

Fonction :

- Longueur ≥ 8
- Contient au moins un caractère spécial parmi ['!', '#', '@', ';', ':']

Analyse :

A - 'best@' → trop court ❌
B - 'paptap23' → pas de caractère spécial ❌
C - '2!@59fgds' → longueur ≥ 8 et contient ! et @ ✅

Réponse : **C**

👉 *Commentaire : Les deux conditions doivent être respectées.*

---

8. Fonction ajouter_mot

```
def ajouter_mot(f, mdp):
    enfiler(f, mdp)
    if longueur(f) > 3:
        defiler(f)
```

👉 *Commentaire : On ajoute d'abord le nouveau mot, puis on retire l'ancien si la taille dépasse 3.*

---

9. Fonction mot_file

```
def mot_file(f, mdp):

    g = creer_file()
    present = False
    while not(est_vide(f)):
        elt = defiler(f)
        enfiler(g, elt)
        if elt == mdp:
            present = True
    while not(est_vide(g)):
        enfiler(f, defiler(g))
    return present
```

👉 *Commentaire : On parcourt toute la file pour tester la présence, puis on la restaure.*

---

10. Fonction modification

```
def modification(f, nv_mdp):
    if est_valide(nv_mdp) and not mot_file(f, nv_mdp):
        ajouter_mot(f, nv_mdp)
        return True
    return False
```

👉 *Commentaire : Le nouveau mot de passe doit être valide ET ne pas être déjà présent dans la file.*
