Arbres binaires

Structures de données

Exercice 1 *QCM : vocabulaire et propriétés des arbres*

Pour chaque question, une seule réponse est correcte.

1. Quelle est la hauteur d'un arbre binaire réduit à sa racine (sans fils) ?

   A. -1
   B. 0
   C. 1
   D. Indéfinie

2. Un arbre binaire de hauteur 3 peut contenir au maximum :

   A. 3 nœuds
   B. 7 nœuds
   C. 8 nœuds
   D. 15 nœuds

3. Quel parcours d'un ABR produit les valeurs dans l'ordre croissant ?

   A. Préfixe (nœud, gauche, droit)
   B. Infixe (gauche, nœud, droit)
   C. Suffixe (gauche, droit, nœud)
   D. En largeur (niveau par niveau)

4. On cherche la valeur 42 dans un ABR équilibré contenant 1 000 nœuds. Le nombre maximal de comparaisons est environ :

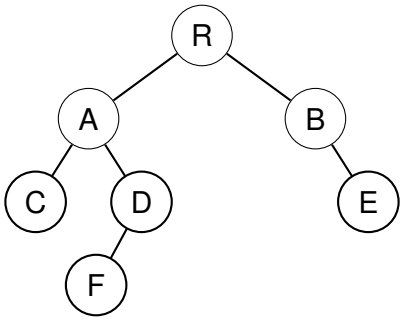
   A. 10
   B. 42
   C. 500
   D. 1 000

5. On insère les valeurs 1, 2, 3, 4, 5 dans cet ordre dans un ABR vide. Quelle forme a l'arbre obtenu ?

   A. Un arbre équilibré de hauteur 2
   B. Un arbre peigne (dégénéré) vers la droite, de hauteur 4
   C. Un arbre peigne vers la gauche, de hauteur 4
   D. Un arbre aléatoire dont la forme dépend de l'implémentation

Exercice 2 *Dérouler les parcours en profondeur*

On considère l'arbre binaire suivant :



1. Recopier et compléter le tableau en indiquant l'ordre de visite pour chaque parcours (préfixe, infixe, suffixe).
2. Dérouler le parcours en largeur en complétant l'état de la file à chaque étape.
3. Quelle est la taille de cet arbre ? Sa hauteur ?

Exercice 3 *Compléter des fonctions récursives à trous*

On utilise la classe `Arbre` définie en cours. Un arbre vide est représenté par `None`.

a) Compléter la fonction qui renvoie `True` si une valeur `x` est présente dans un arbre binaire quelconque :

```
def contient(arbre, x):
    if arbre is None:
        return ...
    if arbre.valeur == x:
        return ...
    return contient(..., x) or contient(..., x)
```

b) Compléter la fonction qui renvoie la liste des feuilles :

```
def liste_feuilles(arbre):
    if arbre is None:
        return ...
    if arbre.gauche is None and arbre.droit is None:
        return ...
    return liste_feuilles(...) + liste_feuilles(...)
```

c) Compléter la fonction qui renvoie le parcours préfixe sous forme de liste :

```
def prefixe_liste(arbre):
    if arbre is None:
        return []
    return [...] + prefixe_liste(...) + prefixe_liste(...)
```

d) Pour la fonction `contient` complétée en a), identifier, sans exécuter le code, les cas de base (quelles conditions arrêtent la récursion ?) et les appels récursifs (comment le problème se réduit-il ?).

Exercice 4 *Construire un ABR par insertions successives*

On insère les valeurs suivantes dans un ABR initialement vide : 8, 3, 10, 1, 6, 14, 4, 7, 13.

1. Dessiner l'ABR obtenu après chaque insertion.
2. Donner la taille et la hauteur de l'ABR final.
3. Effectuer le parcours infixe et vérifier l'ordre croissant.
4. Recherche de la valeur 7 : lister les comparaisons.

Exercice 5 *Programmer des fonctions sur les arbres*

Écrire les fonctions suivantes :

1. `somme(arbre)` : renvoie la somme de toutes les valeurs.
2. `miroir(arbre)` : renvoie un nouvel arbre miroir.
3. `meme_structure(a1, a2)` : renvoie `True` si les deux arbres ont la même structure.

Exercice 6 *Arbre d'expression arithmétique (synthèse)*

Un arbre d'expression est un arbre binaire où les feuilles contiennent des nombres et les nœuds internes contiennent des opérateurs $(+, -, \times, /)$.

1. Dessiner l'arbre d'expression correspondant à $(5 + 1) \times 2 - (9/3)$.
2. Écrire une fonction récursive `evaluer(arbre)` qui évalue un arbre d'expression.
3. Quel parcours correspond à la notation polonaise inversée (NPI) ? Donner la NPI de l'arbre de la question 1.
4. **Bonus.** Écrire `infixe_parenthesee(arbre)`.