TD/TP Structure de données - Arbres binaires de recherche

M. Tellene

1 Appréhender les arbres binaires de recherche

Exercice 1

On souhaite construire un arbre binaire de recherche T de nombres entiers. Dessiner le contenu de T obtenu après l'ajout, **dans l'ordre**, des entiers

1. 20, 10, 30, 5, 15, 25, 35
2. 20, 10, 30, 15, 5, 35, 25
3. 5, 10, 20, 30, 25, 35, 15

Donner la hauteur des arbres obtenues pour chacun des ordres d'insertion

Exercice 2

Donner tous les `ABR` formés de trois noeuds **et** contenant les entiers 1, 2 et 3

2 Manipuler un arbre binaire de recherche

Organisation : aller dans votre dossier personnel, puis créer un dossier « structure de données » (ou « SD »), dans ce dossier, créer un dossier « ABR », c'est ce dossier qui contiendra le travail fait lors de ce TP.

Pour ce TP, nous construirons un `ABR` sans classe noeud.

Vous créerez donc directement une classe `ABR`. Cette classe est composée de trois attributs : `valeur`, `fg` (pour fils gauche) et `fd` (pour fils droit).

Une fois fait, créer les méthodes associées à la structure de données d'`ABR`. Si vous ne vous en souvenez pas, elles sont données dans la suite.

- `est_vide()` : renvoie `True` si l'arbre est vide, `False` sinon
- `insertion(x)` : insère `x` dans l'arbre, l'insertion dans un arbre binaire se déroule de la manière suivante :
- `recherche(x)` : renvoie `True` si `x` est dans l'`ABR`, `False` sinon
- `parcours_infixe()` : affiche les valeurs contenues dans l'arbre dans l'ordre infixe
- `taille()` : renvoie la taille de l'arbre
- `hauteur()` : renvoie la hauteur de l'arbre
- `minimum()` : renvoie **la référence** (l'emplacement mémoire) du noeud de valeur minimum dans l'`ABR`
- `maximum()` : renvoie **la référence** (l'emplacement mémoire) du noeud de valeur maximum dans l'`ABR`

Exercice 3

Bien que nous avons créé une classe `ABR`, il est possible d'utiliser cette classe afin de créer des arbres binaires. Les deux fonctions, à trous, suivantes permettent de vérifier qu'un arbre binaire est un `ABR`. Complétez ces fonctions et intégrez les à votre classe `ABR`.

```
def est_abr(self):
    if self.valeur is None: return ..................................................
    (test, min, max) = self.est_abr_sous_arbre()
    return test

def est_abr_sous_arbre(self):
    minG = maxG = minD = maxD = self.valeur
    testG = testD = True
    if self.fg: (testG, minG, maxG) = ...............................................
    if self.fd: (testD, minD, maxD) = ...............................................
    if not testG or not testD : return (..........................., None, None)
    if ..... > self.valeur or ..... < self.valeur: return (False, None, None)
    return (........................, ........................, ........................)
```

Exercice 4

L'algorithme, à trous, suivant permet de tester l'égalité avec un arbre passé en argument. Complétez cet algorithme et intégrez le à votre classe `ABR`.

```
def est_egal(self, arbre_test):
    if self.valeur is None or arbre_test.valeur is None:
        return self.valeur is None and arbre_test.valeur is None

    if self.valeur != arbre_test: return ...........................................

    fgOk = True
    if self.fg:
        if arbre_test.fg:
            fgOk = ..................................................................
        else:
            return ..................................................................
    else:
        fgOk = arbre.fg is None

    fdOk = True
    if self.fd:
        if arbre_test.fd:
            fdOk = ..................................................................
        else:
            return ..................................................................
    else:
        fdOk = arbre.fd is None

    return ..........................................................................
```