Feuille 9 - Arbres Binaires de Recherche

1 Arbres de Recherche

Un arbre binaire de recherche peut être représenté en Java avec la définition suivante :

```
1 class BinSearchTree<T extends Comparable<? super T>> {
2         T data;
3         BinSearchTree<T> left, right, parent;
4         /*...*/
5 }
```

L'idée est que chaque noeud de cette structure contient :
— Un attribut de type `T` (`data`).
— Deux réferences (pointeurs) pour des sous-arbres, appellées `left, right`.
— Une réference pour son parent.

À l'instar d'une liste chaínée, cette organisation nous permet d'ajouter des éléments de manière dynamique, au fur et à mesure. La différence entre un arbre et une liste (doublement chaînée) est que l'arbre contient **deux** pointeurs "next". On utilise cette organisation pour trier les données contenues dans l'arbre, ce qui permet de localiser plus rapidement un élément cherché. Plus précisément, on adopt les conditions suivantes :
— Pour chaque noeud `v` de l'arbre, tous les éléments contenus dans l'arbre `v.left` sont inférieurs ou égaux à `v.data`.
— Pour chaque `v`, tous les éléments contenus dans `v.right` sont supérieurs à `v.data`.

Notez bien qu'on a exigé que les éléments de type `T` soient comparables, ce qui est indispensable pour une telle organisation.

1.1 Initialisation

Donner un constructeur pour cette classe. C'est une bonne idée de s'assurer que `data` n'est jamais `null` (alors, pas de constructeur par défaut). C'est aussi une bonne idée de définir comme parent de la racine de l'arbre la racine elle-même.

1.2 Membership Test

Programmer une méthode `boolean contains(T t)` qui décide si l'élément passé comme paramètre appartient à l'arbre.
— Donnez une version récursive.
— Donnez une version non-récursive qui utilise $O(1)$ de mémoire.

1.3 Max

Programmer une méthode non-récursive qui retourne la valeur maximum stockée dans l'arbre.

1.4 Sorting

Programmer une méthode `void printInOrder()` qui affiche tous les éléments de l'arbre en ordre croissant, c'est-à-dire, qui affiche une liste triée avec tous les éléments.

— Donnez une version récursive.
— Donnez une version non-récursive qui utlise une pile. Vous pouvez ici utiliser l'interface `DeQueue` de Java. Notamment, vous aurez besoin des méthodes `addLast, removeLast, isEmpty`.
— Donnez une version non-récursive qui utilise $O(1)$ de mémoire.

1.5 Insertion

Programmer une méthode `add` qui ajoute un élément dans l'arbre.

— Donnez une implémentation récursive.
— Donnez une implémentation qui utilise $O(1)$ de mémoire.