1 Problématique

Les arbres binaires, les tas imposent des contraintes aux structures arborescentes. Il en résulte des objets avec des propriétés très utiles. Par exemple, la complexité du tri par tas est $O(n) = n.log(n)$.

Comment obtenir une méthode de recherche efficace avec les arbres ?

2 Arbre binaire de recherche

2.1 Définition

Imposons une contrainte à chaque nœud d'un arbre binaire :

— les valeurs du sous-arbre gauche sont plus petites que celle du nœud,

— les valeurs du sous-arbre droit sont plus grandes que celle du nœud.

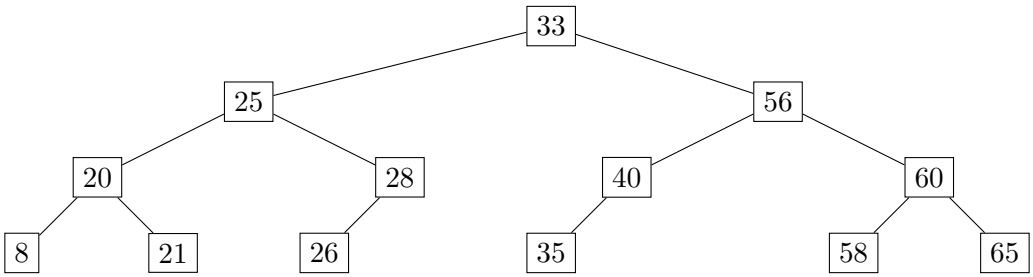

FIGURE 1 – Un Arbre Binaire de Recherche (ABR)

Remarque

On suppose que chaque valeur n'apparaît qu'une seule fois dans l'arbre.

Activité 1 :

1. Placer les valeurs 23, 27, 55, 59 dans l'ABR.
2. Où se trouve la plus grande valeur ? La plus petite ?
3. Effectuer un parcours infixe de l'arbre. Que remarque-t-on ?

2.2 Propriété

Soit n la taille de l'arbre et h sa hauteur. Ces grandeurs sont liées par la relation :

$$h + 1 \leq n \leq 2^{h+1} - 1$$

Cette propriété peut s'écrire :

$$\lfloor \log_2 n \rfloor \leq h$$

Considérons le tableau 1 et la construction de l'arbre binaire de recherche qui en découle (figure 1).

Christophe Viroulaud

```
1 tab = [33, 25, 56, 20, 28, 40, 60, 8, 21, 26, 35, 58, 65]
```

Code 1 – Tableau de données

Activité 2 :

1. Quelle la complexité temporelle dans le pire des cas de la recherche d'un élément dans le tableau ?
2. Que devient cette complexité pour l'ABR ?

Remarque

Pour que la recherche dans l'arbre soit efficace il faut qu'il soit *équilibré*.

3 Implémentation

La programmation objet peut nous permettre d'implémenter un ABR facilement.

Activité 3 :

1. Créer la classe *Noeud* qui possède trois attributs :
   - *valeur* un entier,
   - *gauche* un Noeud,
   - *droit* un Noeud.

   Le constructeur acceptera trois paramètres *(v : int, g = None, d = None)*.
2. Écrire la méthode **inserer(self, v : int) → None** qui crée récursivement le nœud contenant la valeur v dans le sous-arbre gauche ou droit du nœud.
3. Écrire la méthode **rechercher(self, v : int) → bool** qui renvoie *True* si la valeur v est dans le nœud ou dans un de ses sous-arbres.
4. Créer la classe ABR et son constructeur qui possédera un attribut *racine* initialisé à *None*.
5. Écrire la méthode **est_vide(self) → bool** qui renvoie *True* si l'arbre est vide.
6. Écrire la méthode **inserer(self, v : int) → None** qui :
   - crée un nœud contenant v si l'arbre est vide,
   - appelle la méthode *inserer* du nœud racine sinon.
7. Écrire la méthode **rechercher(self, v : int) → None** qui renvoie *True* si v est présent dans l'arbre.

Christophe Viroulaud