Arbres bien équilibrés

Pour rendre les ABR efficaces, il faut minimiser la hauteur par rapport au nombre de nœuds

Définition

Un arbre est bien équilibré si :

- *la différence de hauteur entre ses deux sous-arbres est d'au plus 1*
- *chacun de ses sous-arbres est bien équilibré*

Hauteur d'un arbre bien équilibré

Théorème

Pour un arbre bien équilibré à n nœuds et de hauteur h

$$\log_2(1+n) \leq h \leq \alpha \log_2(2+n)$$

avec $\alpha = \frac{1}{\log_2\left(\frac{1+\sqrt{5}}{2}\right)} < 1,44$

Par conséquent, rechercher, insérer et supprimer sont en $O(\log n)$ dans les arbres bien équilibrés

Arbres auto-équilibrants

Problème : l'insertion et la suppression peuvent déséquilibrer un arbre bien équilibré

- Modifier les fonctions d'insertion et de suppression pour garantir un invariant de bon équilibre

Arbres AVL (Adelson-Velskii et Landis, 1962) : rééquilibrage par rotations

Encapsulation

On veut ne manipuler que des arbres bien équilibrés

- Il faut interdire la création d'arbres non équilibrés à l'extérieur du module

On utilise l'encapsulation liée à la modularité : la seule façon de construire des AVL sera via le constructeur intelligent `bal`

Intérêt : dans `bal` on peut supposer que les arguments sont bien équilibrés, puisqu'ils proviennent des appels aux constructeurs intelligents

Types privés en OCaml

```
type t = private A of u | B of v
```

On ne peut utiliser les constructeurs `A` et `B` qu'à l'intérieur du module
Toujours possible de filtrer `A` et `B` en dehors du module

Intérêt : on définit des constructeurs intelligents

```
let a (x : u) = let x' = ... in A x'

let b (y : v) = let y' = ... in B y'
```

En dehors du module, obligé de passer par les constructeurs intelligents pour faire un `t`
Toutefois, on peut toujours discriminer les objets de type `t`

Complexité

Après une insertion, une seule rotation (éventuellement double) est nécessaire

Après une suppression, la rotation à effectuer peut entraîner un déséquilibre au-dessus, il y a donc au plus h rotations à effectuer

Comme les rotations sont effectuées en temps constant, les opérations pour rééquilibrer l'arbre sont en $O(\log n)$

Par conséquent, dans un arbre AVL, la recherche, l'insertion et la suppression sont en $O(\log n)$, y compris dans le pire des cas