TP 6 - Arbre équilibré: les AVL

Amaury Pouly (amaury.pouly@ens-lyon.fr)

4 et 11 janvier 2011

Le but du jour :

Implémenter des arbres équilibré particuliers : les AVL

Vous pourrez trouver les énoncés ainsi que les corrections des TP sur ma page personnelle :
`http://perso.ens-lyon.fr/amaury.pouly/`

1 Préliminaires

Un AVL (du nom de ses inventeurs, G.M. Adelson-Velskii et E.M. Landis) est un arbre binaire de recherche **équilibré**. Cette structure de données ajoute un champ à chaque nœud qui permet lors des insertions et des suppressions de rééquilibrer l'arbre si nécessaire. Cette information supplémentaire est la hauteur de l'arbre : chaque nœud sait quelle est la hauteur du sous-arbre dont il est la racine. On impose de plus que pour tout nœud, la différence de hauteur entre le fils gauche et le fils droit soit au plus 1 en valeur absolue.

Dans ce sujet, les AVL seront représentés en Caml par le type suivant :

```
type 'a avl =
| Node of int * 'a * ('a avl) * ('a avl)
| Nil;;
```

Exercice 1
Complétez l'information manquante sur la figure 1 (les hauteurs) et vérifier que c'est bien un AVL. Écrivez cet arbre en Caml.

Exercice 2
Implémentez les fonctions `height`, `left`, `right` et `data` retournant respectivement la hauteur, le fils gauche, le fils droit et la donnée d'un nœud. Toutes ces fonctions doivent avoir une complexité en $O(1)$. Implémentez une fonction `mk_node` telle que `mk_node v l r = Node(h, v, l, r)` où `h` est la hauteur de l'arbre obtenu.

On représente un ordre $\preceq$ par une fonction de comparaison qui étant donné v et v' retourne -1 si $v \prec v'$, 0 si $v \approx v'$ et 1 sinon. Pour les entiers, on pourra utiliser la fonction `compare` de la bibliothèque standard.

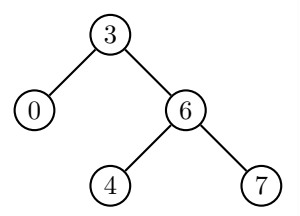

Figure 1 – Exemple d'AVL

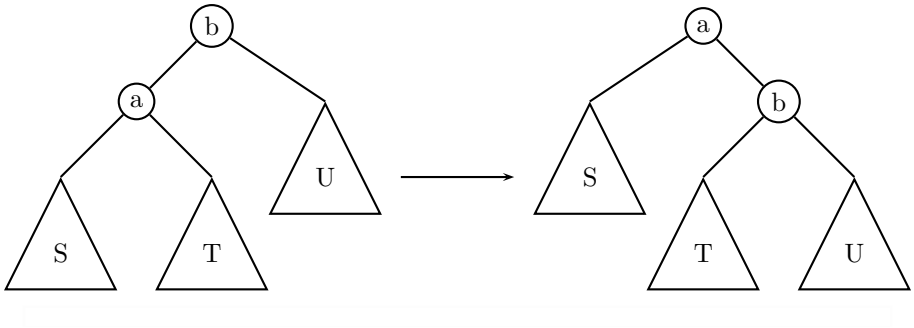

FIGURE 2 – Rotation droite dans un arbre binaire

```
type 'a comp_fn == 'a -> 'a -> int;;
```

Exercice 3
Implémentez une fonction `is_bst : 'a comp_fn -> 'a avl -> bool` qui décide si un arbre est un arbre de recherche valide pour un certain ordre. On ignorera le champ concernant la hauteur dans cette fonction. Testez sur plusieurs exemple pour bien vérifier qu'elle fonctionne.

Exercice 4
Implémentez une fonction `is_avl : 'a avl -> bool` qui décide si un arbre est AVL valide. On ignorera les données contenues aux nœuds pour cette question. Testez sur plusieurs exemple pour bien vérifier qu'elle fonctionne.

2 Rotations

Afin de rééquilibrer un arbre binaire de recherche, il faut procéder à des modifications de sa structure mais tout en conservant l'ordre sur les nœuds. À cette fin, deux opérations servent de base à pratiquement tous les arbres binaires équilibrés : les **rotations**. On distingue deux rotations : les rotations gauche et les rotations droites. La figure 2 illustre la rotation droite, la rotation gauche étant son symétrique.

Exercice 5
Illustrer la rotation gauche. Vérifiez que ces deux types de rotations conservent l'ordre dans un arbre binaire de recherche.

Exercice 6
Implémentez les fonctions `rotate_left : 'a avl -> 'a avl` et `rotate_right : 'a avl -> 'a avl` qui réalisent une rotations gauche et droite dans un AVL. On prendra garde à conserver un champ de hauteur valide. L'arbre obtenu est-il forcément un AVL ?

Dans le cas des AVL, on utilise deux rotations supplémentaires : les doubles rotations gauche et droite. La figure 2 illustre la double rotation droite, la double rotation gauche étant son symétrique.

Exercice 7
Montrez comment on peut réaliser une double rotation droite à partir d'une rotation gauche et d'une rotation droite bien choisies. Faites de même pour une double rotation gauche. Implémentez les fonctions `double_rotate_left : 'a avl -> 'a avl` et `double_rotate_right : 'a avl -> 'a avl` qui réalisent ces opérations. L'arbre obtenu est-il forcément un AVL ?

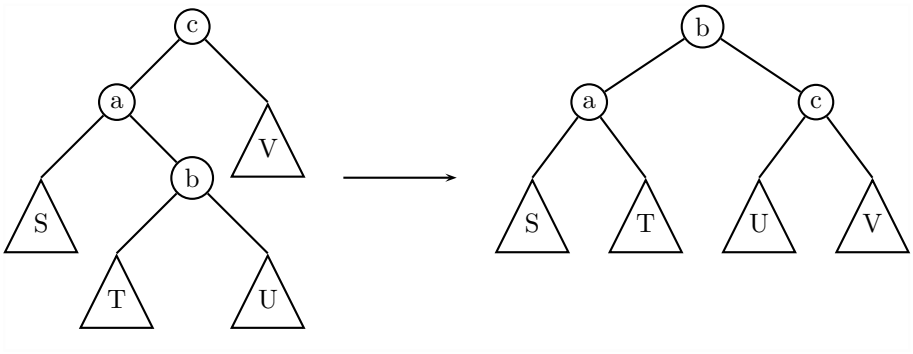

FIGURE 3 – Double rotation droite dans un arbre binaire

3 Rééquilibrage

Après une insertion ou une suppression, il se peut que l'AVL deviennent invalide, c'est à dire que la différence de hauteur entre le fils gauche et le droit d'un nœud vaille 2 en valeur absolue. On cherche alors à restaurer la condition de l'AVL à l'aide des rotations définies précédemment.

Exercice 8
Montrez qu'il y a quatre cas distincts et que dans chaque cas, on peut obtenir un AVL valide à l'aide d'une **seule** rotation (simple ou double).

Exercice 9
Implémentez une fonction `rebalance : 'a avl -> 'a avl` qui étant donné un AVL, retourne cet AVL si la racine est correctement équilibré, ou bien rééquilibre la racine si elle est déséquilibrée.

Exercice 10
À l'aide de la fonction `rebalance`, implémentez une fonction `insert : 'a -> 'a comp_fn -> 'a avl -> 'a avl` qui insère une valeur dans un AVL et rééquilibre l'arbre lorsque c'est nécessaire.
Note : l'insertion se fait comme dans un arbre binaire de recherche, à ceci près qu'on rééquilibre chaque nœud après avoir insérer une valeur dans l'un de ses fils.

Exercice 11
Testez votre fonction d'insertion en insérant des valeur aléatoire dans l'arbre puis en vérifiant que l'arbre obtenu est un arbre binaire de recherche et un AVL valide.

Exercice 12
Montrez qu'un AVL contenant n nœuds est de hauteur au plus $\log_\phi(n+2) - 1$ où ϕ désigne le nombre d'or.

Exercice 13
Implémentez la suppression dans un AVL.

Exercice 14
Quelle est la complexité des différentes opérations dans un AVL ?

Exercice 15(Facultatif)
On désigne par xAVL(δ) un AVL où la différence de hauteur entre le fils gauche et le fils vaut au plus δ en valeur absolue. Un AVL est un xAVL(1). Vérifiez que tout ce précède est encore valide. Quels sont les avantages et inconvénient par rapport à un AVL classique ?