Algorithmique des structures de données arborescentes

Feuille d'exercices 5

1 Rappel sur la hauteur d'un arbre binaire

Exercice 1 : hauteur

1. Rappeler la hauteur *minimale* d'un arbre binaire à n nœuds, et à quelle condition elle est atteinte.
2. Rappeler quelle est la hauteur *maximale* d'un arbre binaire à n nœuds, et à quelle condition cette valeur est atteinte.

2 Arbres AVL

Nous avons vu que pour un arbre binaire de recherche de hauteur h, les opérations de

- recherche d'un élément,
- insertion d'un élément,
- suppression d'un élément

peuvent se réaliser en temps $O(h)$ dans le cas le pire.
Afin de garantir une complexité logarithmique pour ces trois opérations, on s'intéresse aux familles d'arbres garantissant une hauteur au maximum $\alpha \log(n)$ pour des arbres à n nœuds (où la constante $\alpha > 0$ dépend de la famille d'arbres). De tels arbres sont dit "equilibrés". Intuitivement, un arbre d'une famille d'arbres équilibrés a beaucoup de nœuds, un nombre exponentiel par rapport à sa hauteur.

Une 1re famille de tels arbres est celle des arbres AVL. Le terme AVL provient des initiales des chercheurs ayant introduit ces arbres, Georgii Adelson-Velsky et Evguenii Landis.
Les arbres AVL sont des arbres binaires de recherche, mais ils imposent une contrainte supplémentaire d'équilibre au niveau de chaque nœud. Cette contrainte permet de borner la hauteur de l'arbre.

2.1 Définition de l'équilibre d'un nœud

On associe à chaque nœud d'un arbre binaire une notion d'*équilibre*. On rappelle que pour tout arbre binaire t, on note $h(t)$ sa *hauteur* et elle vaut -1 si l'arbre est vide.

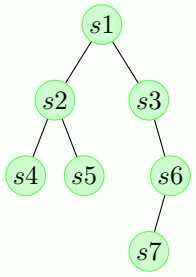
Définition : Équilibre d'un nœud

Soit t un arbre binaire. Soit x un nœud de t, ℓ le sous-arbre gauche de x et r le sous-arbre droit de x. L'*équilibre du nœud* x est la différence de hauteur entre ses deux sous-arbres : $h(\ell) - h(r)$.

Exemple

On donne les équilibres des nœuds de l'arbre ci-contre dans le tableau :

Nœuds	s1	s2	s3	s4	s5	s6	s7
Équilibres	-1	0	-2	0	0	1	0



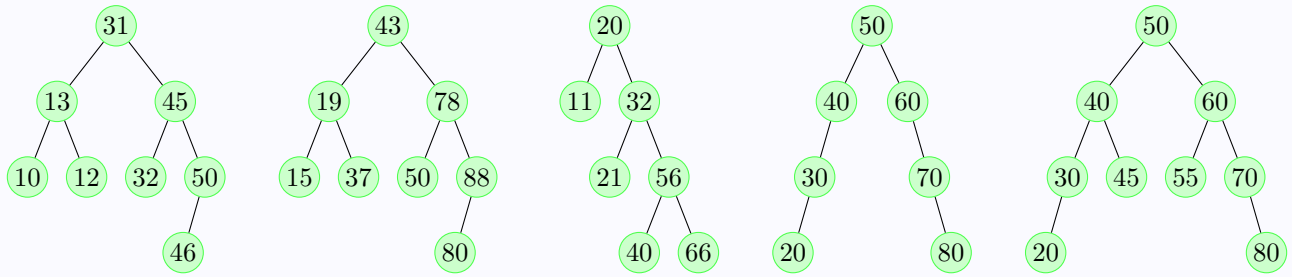
Dans les arbres de la famille AVL, la valeur de l'équilibre de chaque nœud reste bornée.

Définition : Arbres AVL

Un *AVL* est un arbre binaire de recherche dans lequel *tout* nœud a pour équilibre -1, 0 ou 1.

Exercice 2 : Exemples d'arbres AVL

Parmi les arbres binaires suivant, lesquels sont des AVLs ? Justifier.

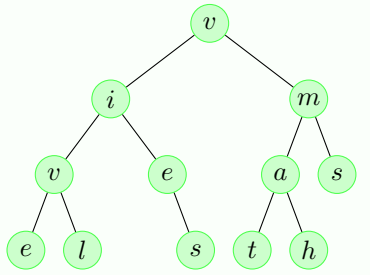


2.2 Notion d'arbres enrichis

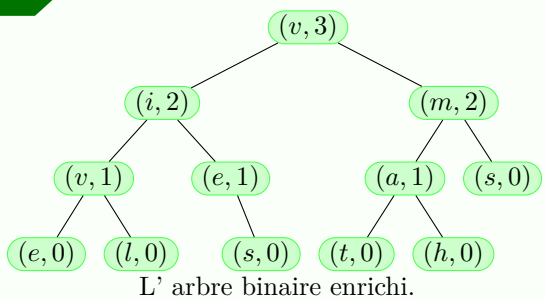
Calculer l'équilibre d'un nœud nécessite d'accéder à la hauteur des sous-arbres. Mais la fonction qui calcule la hauteur d'un arbre binaire a une complexité linéaire. Appeler cette fonction pour chaque nœud de l'arbre entraînerait donc une complexité quadratique (c'est-à-dire $\mathcal{O}(n^2)$ pour des arbres à n nœuds), incompatible avec la complexité logarithmique recherchée.

Pour résoudre ce problème nous allons utiliser des *arbres binaires de recherche "enrichis"*, c'est-à-dire des arbres de recherche où les nœuds stockent non seulement les clés, mais aussi des informations supplémentaires.
Dans le cas présent, l'information qui nous intéresse est la hauteur : si nous choisissons de stocker dans chaque nœud (en plus de la clé) la hauteur du sous-arbre dont il est racine, alors le calcul de l'équilibre des nœuds pourra se fera en temps constant.
Nous allons donc travailler avec des arbres binaires "enrichis dont les nœuds sont étiquetés par des paires : le premier élément sera la clé et le second sera la hauteur du sous-arbre. Ainsi, en OCaml, l'arbre t suivant :
`Node((5,-1), Empty, Node((10,0), Empty, Empty))` est un arbre enrichi de hauteur 1 avec l'élément 5 à la racine.

Exemple



Arbre binaire dont les clés sont des caractères



L' arbre binaire enrichi.

Exercice 3 : Arbres enrichis

Les fonctions suivantes vont nous permettre de manipuler les arbres binaires enrichis.

1. Écrire une fonction `get_height` de type `('a * int) btree -> int` qui prend en entrée un arbre binaire enrichi t et retourne sa hauteur (sa complexité doit être $\mathcal{O}(1)$).
2. Écrire une fonction `tag_node` de type `'a -> ('a * int) btree -> ('a * int) btree -> ('a * int) btree` qui prend en entrée une clé c et deux arbres binaires enrichis ℓ et r. Elle retourne l'arbre binaire enrichi dont la racine a pour clé c et les sous-arbres gauche et droit sont ℓ et r (sa complexité doit être $\mathcal{O}(1)$).
3. En utilisant les deux fonctions précédentes, écrire une fonction `tag_btree` de type `'a btree -> ('a * int) btree` qui prend en entrée un arbre binaire quelconque t et renvoie l'arbre enrichi avec les hauteurs (sa complexité doit être $\mathcal{O}(n)$).

Pour tester si un arbre enrichi est un AVL il faut tout d'abord tester si c'est un arbre binaire de recherche (on peut pour cela adapter la fonction `is_bst` vue dans la feuille précédente aux arbres binaires enrichis). Dans l'exercice suivant on suppose ce test déjà fait.

Exercice 4 : Appartenance à la famille AVL

Écrire une fonction `is_avl` de type `('a * int) btree -> bool` qui prend en entrée un arbre binaire de recherche enrichi et teste s'il s'agit d'un arbre AVL (sa complexité doit être $\mathcal{O}(n)$)

3 Hauteur d'un arbre AVL

On va maintenant montrer que la hauteur des arbres AVL est toujours *logarithmique* par rapport à leur taille. Plus précisément, on va montrer la propriété suivante :
Pour tout arbre AVL t de hauteur $h(t)$ et de taille $s(t)$, il existe une constant α strictement positive telle que :

$$h(t) \leqslant \alpha \times log_2(s(t)+1). \tag{5.1}$$

Ainsi l'algorithme de recherche dans un ABR appliqué à un arbre AVL t, aura pour complexité $\mathcal{O}(log_2(s(t)))$.

Exercice 5 : Hauteur d'un arbre AVL

Pour tout entier $h \geqslant 0$, on note $S_{min}(h) \in \mathbb{N}$ la taille minimale d'un arbre AVL de hauteur h.

1. Quelle est la valeur de $S_{min}(0)$, $S_{min}(1)$?
   Quelle est la relation de récurrence vérifiée par $S_{min}(h)$ pour $h \geqslant 2$?
2. On pose $u(h) = S_{min}(h) + 1$ pour tout $h \geqslant 0$. Quelle est la relation de récurrence vérifiée par la suite u ?
3. On note φ le nombre d'or, c'est-à-dire l'unique solution positive de l'équation $x^2 = x + 1$ (on a $\varphi = \frac{1+\sqrt{5}}{2}$).
   Montrer que pour tout $h \geqslant 0$, on a,
   $$\varphi^h \leqslant u(h).$$
4. On admettra que $\frac{1}{log_2(\varphi)} \leqslant 1.45$. Utiliser la question précédente pour montrer (5.1).

4 Insertion dans un arbre AVL

Insérer une nouvelle clé dans un AVL risque de détruire l'équilibre de l'arbre. Cependant, nous allons voir que l'on peut modifier l'arbre après l'insertion pour en faire un AVL, et ceci sans coût additionnel en complexité. On commencera par transformer l'arbre AVL en un AVL enrichis avec les hauteurs.
L'algorithme usuel d'insertion dans un AVL se réalise en 2 phases :

- on insère d'abord la nouvelle clé comme dans un ABR classique, en remplaçant un sous-arbre vide par une nouvelle feuille contenant cette clé,

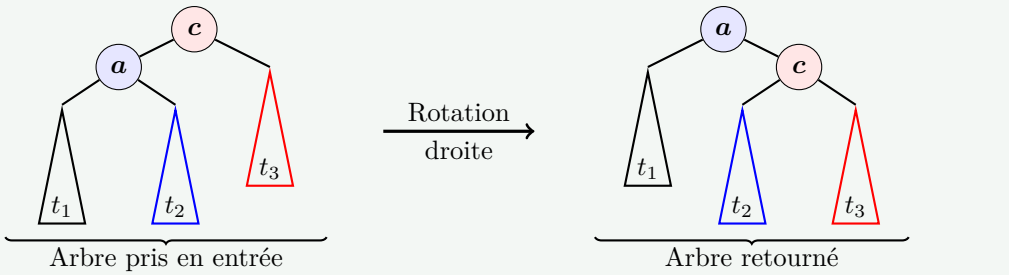
- si la propriété d'être un AVL a été détruite par l'insertion, on rééquilibre l'arbre en le réorganisant.

La phase de rééquilibrage de l'arbre est basée sur la notion de *rotation* que nous allons introduire.

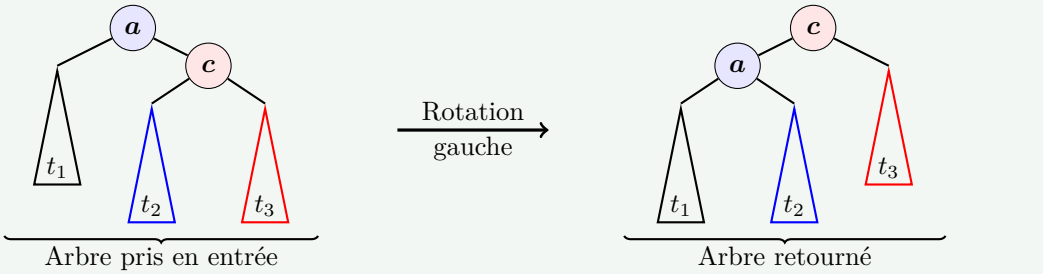
4.1 Rotations

On va définir quatre transformations qui peuvent s'appliquer à un arbre binaire et s'implémentent chacune en temps constant. Elle produisent un nouvel arbre obtenu en réorganisant les nœuds proches de la racine. On les représente graphiquement pour la définition.

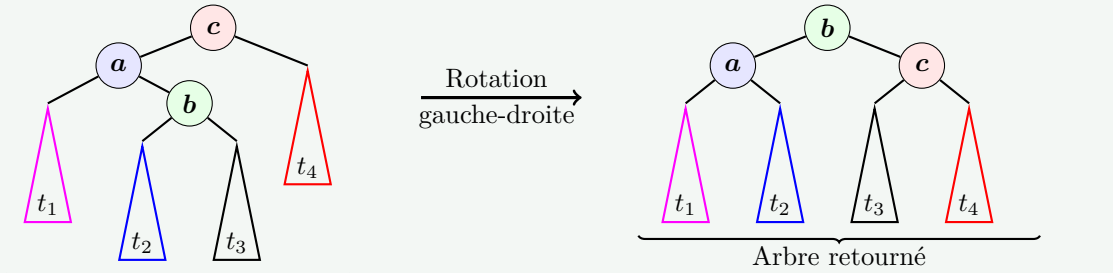
Définition : Rotation droite



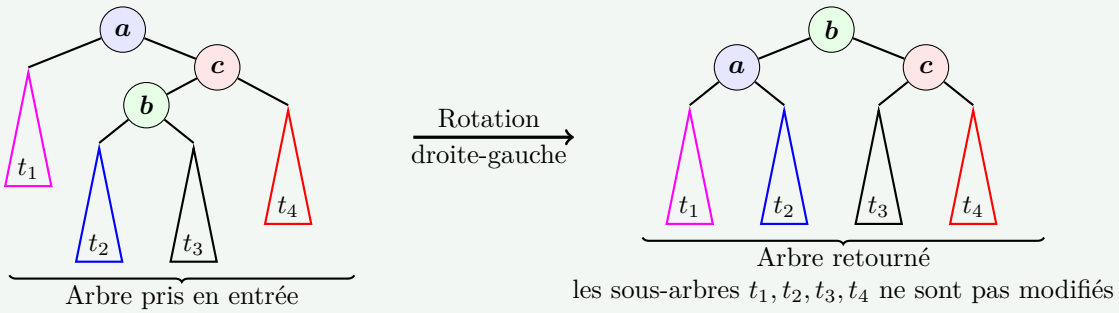
Définition : Rotation gauche



Définition : Rotation gauche-droite



Définition : Rotation droite-gauche



Chaque rotation appliquée à un arbre binaire de recherche produit un nouvel arbre qui reste un binaire de recherche.

Exercice 6 : Propriété de l'arbre après rotation

Soit t un ABR et t' un second arbre construit à partir de t en appliquant l'une des quatre rotations. Montrer que t' est toujours un ABR.

Exercice 7 : Rotations

Écrire quatre fonctions `l_rotate`, `r_rotate`, `lr_rotate` et `rl_rotate`, chacune de type `('a * int) btree -> ('a * int) btree`, qui implémentent les quatre rotations pour les arbres binaires *enrichis* (leur complexité en temps doit être $\mathcal{O}(1)$).

Attention : Puisque nous travaillons avec des arbres binaires *enrichis*, il ne faut pas oublier de corriger les hauteurs mémorisées dans les nœuds déplacés par la rotation.

4.2 Réparation de l'équilibre

L'algorithme de rééquilibrage est un algorithme "bottom-up" remontant depuis la feuille contenant la nouvelle clé jusqu'à la racine.

Exercice 8 : Exemple

Soit la suite de clés $10, 20, 30, 40, 50, 60, 70, 45, 47$. Dessiner les étapes d'insertion de ces clés dans un arbre AVL initialement vide. On prendra soin d'indiquer les coefficients d'équilibrage des noeuds pour chaque arbre intermédiaire.

Nous pouvons maintenant récapituler toutes les étapes de l'insertion dans un arbre AVL.

Exercice 9 : Insertion

1. Proposer un algorithme d'insertion qui prend en entrée un arbre AVL enrichi t et une clé c. Il doit construire un nouvel arbre AVL enrichi t' dont l'ensemble de clé est celui de t auquel on a ajouté c et qui satisfait $h(t) \leqslant h(t') \leqslant h(t)+1$. La complexité de votre algorithme doit être $\mathcal{O}(h(t))$. On justifiera que cet algorithme est correct.

2. Écrire une fonction `avl_insert` de type `('a * int) btree -> ('a * int) btree` qui implémente l'algorithme de la question précédente.