Arbres binaires #3 : arbres binaires de recherche

Pierre Karpman

Lycée Champollion MP2I
https://membres-ljk.imag.fr/Pierre.Karpman/CPGE/2025/

Table des matières

Arbre binaire de recherche

Un *arbre binaire de recherche* (en anglais : *binary search tree*) ou ABR est un arbre binaire tel que pour tout nœud, sa valeur (la valeur de son étiquette) est (strictement) supérieure (resp. inférieure) à celle de tous les nœuds de son sous-arbre gauche (resp. droit), pour une certaine relation d'ordre (totale)

Intérêt

Permet une recherche dichotomique sans structure de tableau explicite (par ex. dans un environnement fonctionnel) et sur des données dynamiques

Notation infixe (alphabétique)

Il est commode d'utiliser $\langle A, b, C \rangle$ pour dénoter l'arbre construit à partir d'un sous-arbre gauche (resp. droit) A (resp. C) dont la racine a l'étiquette b

- On utilise $\bullet$ pour l'arbre / un nœud vide

Exemples

ABR *bis*

Représentation

Comme n'importe quel arbre binaire, rien de particulier (sauf éventuellement stocker aussi une représentation de la relation d'ordre)

Propriété non locale

Pour tester si un arbre est un ABR, on **ne peut pas** se contenter de vérifier localement qu'un nœud est plus grand que ses enfants

- Légèrement non trivial de vérifier qu'un arbre est un ABR en temps linéaire en sa taille (*cf.* TP)

Parcours infixe

Un parcours en profondeur infixe d'ABR visite les nœuds par étiquettes croissantes

- Preuve ?

ABR *ter* : algorithmes

Recherche d'un élément

Évidente par dichotomie : comme un parcours en profondeur, mais on peut systématiquement éviter de visiter l'un des deux sous-arbres ; coût linéaire en la hauteur de l'arbre

Ajout d'un élément

Recherche par dichotomie du nœud qui doit être notre parent puis modification / construction d'un nouveau nœud ; coût linéaire en la hauteur de l'arbre

- version impérative : on modifie le nœud parent pour lui rajouter le nouvel élément (et l'on peut s'arrêter là)
- version fonctionnelle : on construit un nouveau nœud pour le parent qui contient son nouvel enfant **et son autre enfant inchangé**, et de même dans la remontée récursive jusqu'à la racine

ABR *quater* : algorithmes *bis*

Suppression d'un élément

Un peu moins immédiat ; on peut se doter d'une fonction `pop_min t` qui renvoie l'élément min de `t` et l'ABR obtenu en l'y retirant de `t` et ensuite :

$\texttt{del}\ \ b\ \langle \bullet, b, \bullet \rangle = \bullet$

$\texttt{del}\ \ b\ \langle A, b, \bullet \rangle = A$

$\texttt{del}\ \ b\ \langle \bullet, b, C \rangle = C$

$\texttt{del}\ \ b\ \langle A, b, C \rangle = \langle A, b', C' \rangle$ avec $b', C' = \texttt{pop\_min}\ \ C$

et recherche dichotomique quand l'étiquette est différente de celle à supprimer

ABR *quinquies*

Applications

Les ABRs offrent une solution (généralement) efficace et (possiblement) fonctionnelle pour représenter et manipuler :

- des ensembles (totalement ordonnables)
- des *tableaux associatifs* (« dictionnaires ») (d'un espace de clef totalement ordonnable)
- des files de priorité (même si l'on sait faire mieux)

et aussi par ex. trier (même si l'on sait faire (marginalement) mieux)

Coût

Les opérations de base sur les ABRs sont de coût linéaire en la hauteur de l'arbre

- $O(\log N)$ pour un arbre de N éléments **équilibré**

Comment s'assurer que l'on manipule des ABRs équilibrés

- dans le pire cas ?
- ou éventuellement « probabilistiquement » ?

Table des matières

Arbres binaires de recherche auto-équilibrés

Une implémentation de la structure de donnée d'ABR (ou plus généralement d'arbre de recherche) est dite *auto-équilibrée* (en anglais : *self-balancing*) si elle garantit (éventuellement de façon amortie ou probabilistiquement) ne manipuler (et donc construire) que des arbres binaires équilibrés

Beaucoup de littérature sur le sujet

- Arbres 2-3
- Arbres 2-3-4
- B-arbres
- Tarbres
- …

Pour nous

- Arbres rouge-noire ou bicolores (ceux au programme)
- Arbres AVL (les premiers historiques ; les plus simples ?)

Rotations

Un mécanisme commun à beaucoup de méthodes d'auto-équilibrage : les rotations

Si un ABR n'a pas la forme qu'on veut, on peut lui appliquer une *rotation*

- Doit préserver le caractère d'ABR
- Doit (pouvoir) changer la forme (pour être utile)

Exemple

$\langle\langle A, b, C\rangle, d, E\rangle \leftrightarrow \langle A, b, \langle C, d, E\rangle\rangle$

Informellement : maintient les lettres en place mais change les emplacements des $\langle$ et $\rangle$ en préservant leurs spécifications et le bon parenthésage

Table des matières

Arbres ~~AVL~~ АВЛ

АВЛ : **А**дельсон-**В**ельский — **Л**андіс

Critère d'équilibrage AVL

Un arbre binaire est (équilibré au sens) AVL si tout nœud $\langle A, b, C \rangle$ satisfait $|h_A - h_C| \leq 1$

Hauteur d'un arbre AVL

Le nombre de nœuds N_h d'un arbre AVL de hauteur h satisfait :

$$N_{h-1}^{\min} + N_{h-2}^{\min} + 1 \leq N_h \leq 2 \times N_{h-1}^{\max} + 1$$

- $N_h^{\max} = 2^{h+1} - 1$
- $N_h^{\min} = N_{h-1}^{\min} + N_{h-2}^{\min} + 1 = F_{h+2} - 1 \approx \varphi^h$

La hauteur d'un arbre AVL de N nœuds est un $\Theta(\log N)$

Les arbres AVL équilibrés sont équilibrés

Maintenir un équilibrage AVL

Constat

- `add` et `del` peuvent changer la hauteur d'un (sous-)arbre de 1
- Ceci peut rompre l'équilibrage AVL au niveau d'un parent
- Que l'on peut rétablir avec des rotations

L'équilibrage peut être maintenu aisément dans une implémentation fonctionnelle en utilisant des *smart constructors*

Approche par *smart constructors*

- Dans `add`, `del`, on remplace simplement les appels au constructeur de type par l'appel à une fonction qui en plus de construire un nouvel arbre maintient le critère AVL
- Approche *extrêmement modulaire* ; on découple totalement la logique de parcours de celle d'équilibrage
  - On peut « facilement » modifier l'un sans l'autre

Illustration sur add

```
let rec add x t
    = match t with
    | E -> N (E, x, E)
    | N (_A, b, _C)
        -> if b < x then N (_A, b, (add x _C)) else
           if b > x then N ((add x _A), b, _C) else
           t
(* devient *)
let rec add x t
    = match t with
    | E -> bn E x E
    | N (_, _A, b, _C)
        -> if b < x then bn _A b (add x _C) else
           if b > x then bn (add x _A) b _C else
           t
```

Quelles spécifications pour `bn` ?

Soit :

- A, C des ABRs équilibrés AVL, de hauteur h_A, h_C avec $|h_A - h_C| \leq 2$
- $t := \langle A, b, C \rangle$ un ABR de hauteur h_t

Alors :

- $t' :=$ `bn` A b C est un ABR équilibré AVL de mêmes éléments que t, et tel que $|h_{t'} - h_t| \leq 1$

Arguments

- Les conditions sur A, C sont satisfaites pour un appel à `bn` dans un cas de base de `add`, `del`…
- Elles le sont aussi lors des retours d'appels récursifs (en « remontant » vers la racine) grâce aux spécifications de `bn`, et notamment la condition sur $h_{t'}$ (et le fait qu'on appelle les fonctions sur des arbres initialement AVL)

Réalisation de bn

Il suffit de considérer tous les cas possibles en déconstruisant A ou C (il y en a seulement 2×2) et de les résoudre par rotation

Résoluble par une déconstruction et une rotation

- $\langle\langle A, b, C\rangle, d, E\rangle$ quand $h_A > h_E$
- $\langle A, b, \langle C, d, E\rangle\rangle$ quand $h_E > h_A$

Résoluble par deux déconstructions et deux rotations

- $\langle\langle A, b, \langle C, d, E\rangle\rangle, f, G\rangle$ quand $h_A = h_G \wedge \max h_C\ h_E = h_G$
- $\langle A, b, \langle\langle C, d, E\rangle, f, G\rangle\rangle$ quand $h_G = h_A \wedge \max h_C\ h_E = h_A$

Résultat

$\mathrm{bn}\ \langle A, b, C\rangle = \langle A, b, C\rangle$ quand $|h_A - h_C| \leq 1$
$\mathrm{bn}\ \langle\langle A, b, C\rangle, d, E\rangle = \langle A, b, \langle C, d, E\rangle\rangle$ quand $h_A > h_E$
$\mathrm{bn}\ \langle A, b, \langle C, d, E\rangle\rangle = \langle\langle A, b, C\rangle, d, E\rangle$ quand $h_E > h_A$
$\mathrm{bn}\ \langle\langle A, b, \langle C, d, E\rangle\rangle, f, G\rangle = \langle\langle A, b, C\rangle, d, \langle E, f, G\rangle\rangle$ quand $h_A = h_G \wedge \max h_C\ h_E = h_G$
$\mathrm{bn}\ \langle A, b, \langle\langle C, d, E\rangle, f, G\rangle\rangle = \langle\langle A, b, C\rangle, d, \langle E, f, G\rangle\rangle$ quand $h_A = h_G \wedge \max h_C\ h_E = h_G$

Implémentation possible en coût constant, si l'on maintient une information de hauteur pour tous les nœuds

Table des matières

Arbres ~~de Stendhal~~ rouge-noire

On enrichit le type d'arbre pour ajouter une couleur **rouge** ou **noire** à chaque nœud non vide ; *les nœuds vides sont noirs*

Critère d'équilibrage rouge-noire

Un arbre (binaire de recherche) rouge-noire est équilibré ss.'il satisfait les deux invariants que :

- Le parent (éventuel) d'un nœud rouge doit être noir
- Tous les nœuds vides ont la même *profondeur noire*, définie comme le nombre de nœuds noirs sur le chemin les reliant à la racine

Exemples

Les ARN équilibrés sont équilibrés

Lemme

Un nœud non vide noir de profondeur noire d d'un ARN équilibré a entre 2 et 4 descendants de profondeur noire $d+1$

Preuve. Sans distinguer les nœuds non-vides noirs des nœuds vides noirs, on a quatre cas possibles :

- $\langle \bullet, \bullet, \bullet \rangle$
- $\langle \langle \bullet, \bullet, \bullet \rangle, \bullet, \bullet \rangle$ et symétrique
- $\langle \langle \bullet, \bullet, \bullet \rangle, \bullet, \langle \bullet, \bullet, \bullet \rangle \rangle$

Corollaire

Par récurrence immédiate, le nombre de nœuds B de profondeur noire d d'un ARN équilibré de racine noire est encadré par :

$$2^d \leq B \leq 4^d$$

Les ARN équilibrés sont équilibrés *bis*

On considère un ARN équilibré de racine noire de N nœuds non vides, de hauteur noire h_n (définie comme la profondeur noire de ses nœuds vide) et hauteur $h_n \leq h \leq 2h_n$

- En considérant sa représentation comme arbre binaire strict, il a $N+1$ nœuds vides (noirs) [relation feuille-nœud]
- Ces nœuds ont tous la même profondeur noire
- Par le corollaire précédent $2^{h_n} \leq N+1 \leq 4^{h_n}$
- D'où $h_n \leq \log(N+1) \leq 2h_n \leq 2h$
- Et $h \leq 2\log(N+1)$

Les ARN équilibrés sont équilibrés

Maintenir un équilibrage ARN

- Il est nettement plus compliqué de maintenir un ARN équilibré qu'un AVL, surtout dans le cas de la suppression de nœuds
- Le cas de l'insertion nécessite de résoudre 4 cas (très symétriques), avec des rotations & recoloriages

Stratégie

- Quand on insert un nœud, on le colorie en rouge
  - Ceci n'augmente pas la profondeur noire des nœuds vides
- Si le parent est noir, l'arbre est équilibré
- Sinon on demande au grand-parent (forcément noir) de résoudre le problème
- Ce qu'on fait *localement* au niveau du sous-arbre dont il est racine
- Ce qui « change » la couleur de la racine en rouge
- On répète le même processus en remontant jusqu'à la racine
- Que l'on colorie en noir

Équilibrage local pour l'insertion : contraintes & libertés

Soit t un ABR à équilibrer, le résultat :

- doit être un ABR de mêmes éléments que t
- doit être équilibré
- doit avoir la même hauteur noire que t [nécessaire pour pouvoir être appelé récursivement]
- peut recolorier un nœud rouge en noir, mais pas l'inverse [on ne connaît pas la couleur des enfants du nœud recolorié], à part la racine

Quatre cas symétriques à considérer

A, C, E, G, sont des ABRs équilibrés ARN de même hauteur noire h ; t de racine noire et hauteur noire $h+1$ peut être l'un de :

$\langle\langle\langle A, \mathbf{b}, C\rangle, \mathbf{d}, E\rangle, \mathbf{f}, G\rangle$

$\langle\langle A, \mathbf{b}, \langle C, \mathbf{d}, E\rangle\rangle, \mathbf{f}, G\rangle$

$\langle A, \mathbf{b}, \langle\langle C, \mathbf{d}, E\rangle, \mathbf{f}, G\rangle\rangle$

$\langle A, \mathbf{b}, \langle C, \mathbf{d}, \langle E, \mathbf{f}, G\rangle\rangle\rangle$

Dans tous les cas le résultat est :

$\langle\langle A, \mathbf{b}, C\rangle, \mathbf{d}, \langle E, \mathbf{f}, G\rangle\rangle$

qui est bien un ABR de mêmes éléments équilibré ARN et de hauteur noire $h+1$

Implémentation possible en coût constant

Équilibrage en cas de suppression

- La suppression est plus délicate car elle peut avoir lieu « n'importe où » (pas seulement sur une feuille)
- En plus de violer l'invariant de parentalité, cela peut maintenant aussi violer celui sur la hauteur noire
- De même qu'une suppression de $\langle \bullet, \mathbf{s}, \bullet \rangle$
- Une approche : introduire des nœuds « double noirs » qui comptent pour deux dans la hauteur noire
- Les analyses de cas doivent maintenant essayer de supprimer les nœuds double noirs, en plus de réparer les éventuels problèmes de parentalité et de hauteur noire

Exemple

$\langle\langle A, \mathbf{b}, C\rangle, \mathbf{d}, \langle E, \mathbf{f}, G\rangle\rangle$

se transforme en :

$\langle\langle\langle A, \mathbf{b}, C\rangle, \mathbf{d}, E\rangle, \mathbf{f}, G\rangle$

Table des matières

Bilan

- Les arbres binaires de recherche offrent une solution pour implémenter des structures de données abstraites (plus) avancées
- C'est efficace quand les arbres sont équilibrés, ce que l'on peut garantir (plus ou moins facilement)
- Solution privilégiée dans un contexte fonctionnel
  - Par exemple quand « imposé » par le langage, ou utile pour la persistance
- Ou s'il est intéressant de pouvoir parcourir les données « dans l'ordre »

Alternative impérative classique : les tables de hachage (*cf.* un peu plus tard)