

TP n° 5: Arbres binaires de recherche, arbres splay

BATOG Guillaume

01 février 2007

De nombreuses structures de données en informatique sont appelées *dictionnaires* : ce sont des structures de données supportant les opérations de requête, d'insertion et de suppression de clé. Le premier dictionnaire rencontré est implémenté par les listes triées : toutes les opérations de dictionnaire s'effectuent en temps $O(n)$ ce qui n'est pas acceptable. Les arbres binaires de recherche sont à peine plus satisfaisant car les opérations prennent un temps linéaire en la hauteur de l'arbre, dont les bornes sont données par le lemme suivant.

Lemme 1. *La hauteur h d'un arbre binaire de recherche à n nœuds (internes) vérifie les inégalités :*

$$\lceil \log n \rceil \leq h \leq n - 1 \quad \text{et} \quad \mathbb{E}[h] \sim C \log n$$

la moyenne étant considérée sur tous les arbres binaires de taille n .

L'objectif est de construire une structure d'arbres telle que la complexité logarithmique soit atteinte. On ne le réalisera pas dans ce TP mais on construira une structure *adaptive*, qui se transforme au cours d'une requête. Dans le pire des cas, la complexité d'une requête reste linéaire mais une suite de m opérations aura une complexité en $O(m \log n)$ d'où une complexité amortie en $O(\log n)$.

On implémentera les arbres binaires de recherche sous Caml de la façon suivante, où on introduit un champ `objet` qui permettra d'étendre la structure d'ABR à d'autres plus complexes.

```
type 'a node_type = { mutable gauche : 'a abr ; mutable droit : 'a abr ;  
                      mutable cle : int ; mutable objet : 'a }  
and 'a abr = Nil | Noeud of 'a node_type ;
```

Exercice 1 [Opérations élémentaires dans un ABR]

Écrire les fonctions usuelles `rechercher`, `inserer`, `predecesseur` et `supprimer` dans les arbres binaires de recherche.

Exercice 2 [Rotations]

Implémenter les opérations zig droite et zig gauche (opération inverse de zig droite, cf figure). Implémenter alors les opérations zig-zig et zig-zag gauches et droites.

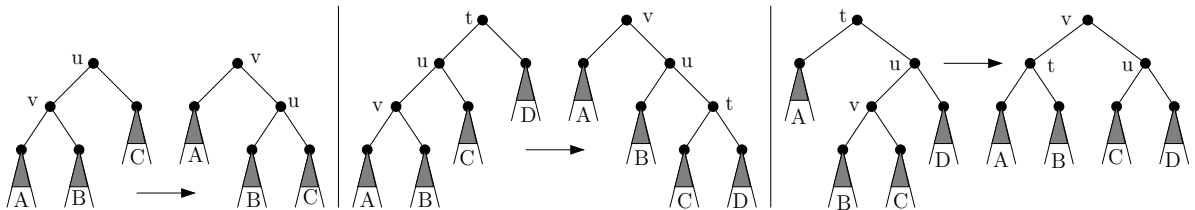


FIG. 1 – Opérations zig, zig-zig et zig-zag droites

Exercice 3 [Splay trees]

Une requête dans un splay tree s'effectue en deux étapes :

1. parcourir l'arbre en profondeur jusqu'à atteindre le noeud de clé recherchée, s'il n'existe pas, retourner immédiatement faux ;
2. remonter le noeud de clé recherchée jusqu'à la racine : si on note **courant**, le noeud courant de clé recherchée au cours de sa remontée, la transformation de l'arbre suit les règles suivantes :
 - si **courant** est la racine, retourner vrai ;
 - si **courant** est un fils de la racine, effectuer l'opération zig adéquate ;
 - sinon effectuer l'une des opérations zig-zig ou zig-zag adéquate.

Écrire une fonction de requête pour les splay trees.

Exercice 4 [Insertion dans un arbre rouge-noir] (5/2)

Un arbre rouge-noir est un ABR vérifiant les cinq propriétés suivantes :

- (A-1) Chaque nœud est soit rouge soit noir.
- (A-2) La racine est noire.
- (A-3) Chaque feuille est noire.
- (A-4) Si un nœud est rouge, ses deux fils sont noirs.
- (A-5) Pour un nœud x fixé, tous les chemins issus de x vers une feuille possèdent le même nombre de nœuds noirs.

En Caml, on implémentera ces arbres de la façon suivante :

```
type couleur = Rouge | Noir
and 'a partielRN = { mutable couleur : couleur ; mutable objet : 'a }
and 'a arbreRN == ('a partielRN) abr ;;
```

Écrire une fonction d'insertion pour les arbres rouge-noir en utilisant les rotations et recoloiages adéquats. (*partie I, concours Centrale-Supélec 2000*)

Complexité amortie

Une opération sur une structure de données a une *complexité amortie* en $O(f(n))$ s'il existe une fonction $g(n)$ telle que tout suite de m opérations au cours desquelles l'objet abstrait sous-jacent reste de taille inférieure à n , s'effectue en temps au plus $g(n) + O(mf(n))$, c'est-à-dire que cette opération s'effectue en temps $O(f(n))$ en moyenne sur une longue suite de telles opérations. (À ne pas confondre avec la complexité moyenne d'un algorithme où la moyenne est prise sur l'ensemble des entrées de l'algorithme). En analyse amortie, on introduit souvent une fonction potentielle V : une différence de potentiel résultant d'une opération traduit le déséquilibre apporté par celle-ci suivant une mesure adéquate. Ici, ce sont les hauteurs des noeuds qui nous intéressent d'où les définitions suivantes.

Définition Pour tout arbre de recherche et toute affectation de poids (strictement positifs) aux noeuds de l'arbre, on définit :

- la somme des poids $s(u)$ d'un nœud u comme étant la somme des poids des nœuds du sous-arbre de racine u ,
- le rang $r(u)$ du nœud u définit par $r(u) = \log(s(u))$,
- le potentiel V de l'arbre comme étant la somme des rangs de ses nœuds.

Exercice 5 On note V_{avant} , r_{avant} , s_{avant} et $V_{\text{après}}$, $r_{\text{après}}$, $s_{\text{après}}$ les fonctions correspondantes avant et après une opération de rebalancement. On utilisera les notations de la figure.

- a) Montrer l'inégalité $V_{\text{après}} - V_{\text{avant}} \leq 3(r_{\text{après}}(v) - r_{\text{avant}}(v))$ au cours d'un rebalancement de type zig.
- b) On se place dans le cas d'un rebalancement de type zig-zig.
 - . Montrer que $(r_{\text{avant}}(v) - r_{\text{après}}(v)) + (r_{\text{après}}(t) - r_{\text{après}}(v)) \leq -2$.

- . En déduire que $V_{\text{après}} - V_{\text{avant}} \leq 3(r_{\text{après}}(v) - r_{\text{avant}}(v)) - 2$.
- c) On se place dans le cas d'un rebalancement de type zig-zag.
 - . Montrer que $(r_{\text{après}}(u) - r_{\text{après}}(v)) + (r_{\text{après}}(t) - r_{\text{après}}(v)) \leq -2$.
 - . En déduire que $V_{\text{après}} - V_{\text{avant}} \leq 3(r_{\text{après}}(v) - r_{\text{avant}}(v)) - 2$.
- d) En déduire que si l'opération de requête d'un nœud v dans un splay tree requiert k opérations zig, alors on a l'inégalité $k + (V_{\text{après}} - V_{\text{avant}}) \leq 1 + 3(r_{\text{après}}(v) - r_{\text{avant}}(v))$.

Exercice 6 Majorer le nombre total d'opérations zig effectuées au cours d'une suite de m requêtes sur un splay tree de taille n . En déduire que toute suite de m requêtes sur un splay tree de taille n s'effectue en temps $O(m \log(n) + n \log(n))$. Qu'en est-il si on considère une suite de m opérations de type requête, insertion et suppression ?

Exercice 7 Quel est la complexité amortie des requêtes d'un splay tree de taille n lorsque ces requêtes sont choisies indépendamment suivant une distribution de probabilités $(p_i)_{i=1\dots n}$?

Références

- [1] B. Allen et J.I. Munro. Self-organizing binary search trees. *Journal ACM*, 25 :526-535, 1978. Essentiellement les techniques développées dans ce TP.
- [2] P. Brass. *Advanced Data Structures*, Version provisoire 07/2006, chapitre 3.8. <http://www-cs.cuny.cuny.edu/~peter/dsa4.ps>
- [3] D.D. Sleator et R.E. Tarjan. Self-adjusting binary search trees. *Journal ACM*, 32 :652-686, 1985. Les véritables splay-trees avec d'autres propriétés adaptatives.