Informatique – MP Faidherbe

TP6 : Skyscraper et arbres AVL

Corrigé

Alix Goguey - alix.goguey@gmail.com

année 2012-2013

1 Introduction

Les arbres AVL sont des arbres binaires de recherche équilibrés. Dans un arbre AVL, les hauteurs des deux sous-arbres d'un même noeud diffèrent au plus de un. Ce système permet d'améliorer la complexité des pires cas. Le nom des AVL provient de leurs inventeurs : Georgii Adelson-Velsky et Evguenii Landis.

Concrètement, la recherche, l'ajout et la suppression de noeud sont identiques à ceux d'un arbre binaire de recherche normale. La seule différence est un rééquilibrage de l'arbre après un ajout ou une suppression selon les règles suivantes : après un ajout (respectivement une suppression), on calcule l'équilibre $\delta = prof(fg) - prof(fd)$ du noeud.

Cinq cas s'offrent alors à nous (se référer ensuite à la Figure1) :

- **a** : $\delta \in [\![-1; 1]\!] \Rightarrow$ on ne fait rien...
- **b** : $\delta = -2 \Rightarrow$ on calcul le δ du fils droit (appelons-le δ_d)
  - **b1** : $\delta_d = -1 \Rightarrow$ *Right-Right case*
  - **b2** : $\delta_d = 1 \Rightarrow$ *Right-Left case*
- **c** : $\delta = 2 \Rightarrow$ on calcul le δ du fils gauche (appelons-le δ_g)
  - **c1** : $\delta_g = -1 \Rightarrow$ *Left-Right case*
  - **c2** : $\delta_g = 1 \Rightarrow$ *Left-Left case*

	En moyenne	Pire cas
Recherche	O(log n)	O(log n)
Insertion	O(log n)	O(log n)
Suppression	O(log n)	O(log n)

TABLE 1: Complexité d'un arbre AVL

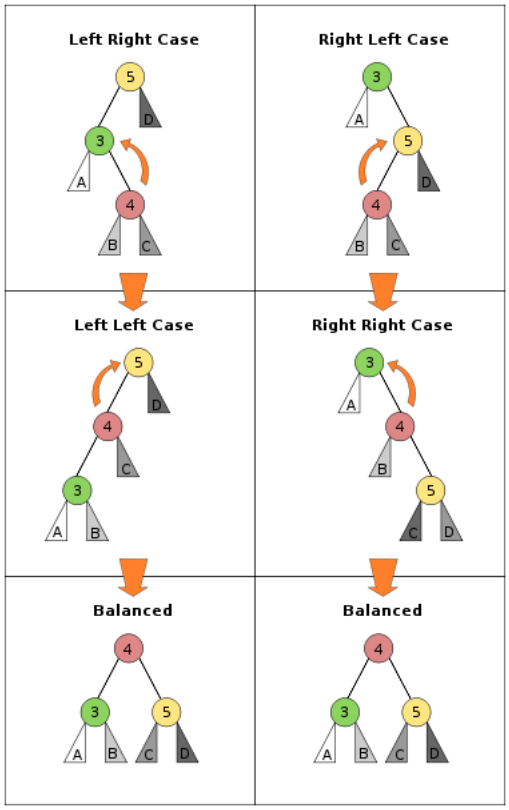

FIGURE 1: Cas rencontrés lors du rééquilibrage d'un arbre AVL

2 Un AVL un peu général

Voici le type AVL que nous manipulerons. Un noeud a une étiquette de type `info`. Pour l'instant, `info` est juste un entier mais en pensant la chose un peu plus générale, nous n'aurons pas grand chose à changer pour mettre n'importe quel type en guise d'étiquette.

```
type info = {id: int};;
type arbre_avl = Vide | N of info * arbre_avl * arbre_avl;;
```

Exercice 1 : Implémentez la fonction `prof` qui prend en argument un arbre et calcule la profondeur.

```
let rec prof noeud =
  match noeud with
  | Vide -> 0
  | N(n,fg,fd) -> 1+(max (prof fd) (prof fg))
;;
```

Exercice 2 : Implémentez la fonction `get_elem` qui prend en argument un arbre et renvoie l'étiquette de la racine.

NB : Si le noeud est vide renvoyez une erreur.

```
let get_elem noeud =
  match noeud with
  | Vide -> failwith "Erreur dans get_elem (Vide)"
  | N(n,fg,fd) -> n
;;
```

Exercice 3 : Implémentez la fonction `get_fg` qui prend en argument un arbre et renvoie le fils gauche de la racine.

NB : Si le noeud est vide renvoyez une erreur.

```
let get_fg noeud =
  match noeud with
  | Vide -> failwith "Erreur dans get_fg (Vide)"
  | N(n,fg,fd) -> fg
;;
```

Exercice 4 : Implémentez la fonction `get_fd` qui prend en argument un arbre et renvoie le fils droit de la racine.

NB : Si le noeud est vide renvoyez une erreur.

```
let get_fd noeud =
  match noeud with
  | Vide -> failwith "Erreur dans get_fd (Vide)"
  | N(n,fg,fd) -> fd
;;
```

Exercice 5 : Implémentez la fonction `balance` qui prend en argument un arbre et renvoie le δ de la racine.

```
let balance noeud =
  match noeud with
  | Vide -> 0
  | N(n,fg,fd) -> (prof fg)-(prof fd)
;;
```

Exercice 6 : Implémentez la fonction `right_case` qui prend en argument un arbre et effectue les *Right-Right case* et *Right-Left case*.

NB : Si le noeud est vide renvoyez une erreur.

```
let right_case noeud =
  match noeud with
  | Vide -> failwith "Erreur dans la rotation droite (Vide)"
  | N(n,fg,fd) -> match (balance fd) with
          | -1 -> N(get_elem fd,
                    N(n,fg,get_fg fd),
                    get_fd fd)
          |  1 -> N(get_elem (get_fg fd),
                     N(n,fg,get_fg (get_fg fd)),
                    N(get_elem fd,get_fd (get_fg fd),get_fd fd))
          |  _ -> failwith "Erreur dans la rotation droite :
                  les fils ne sont pas des AVL"
;;
```

Exercice 7 : Implémentez la fonction `left_case` qui prend en argument un arbre et effectue les *Left-Right case* et *Left-Left case*.

NB : Si le noeud est vide renvoyez une erreur.

```
let left_case noeud =
  match noeud with
  | Vide -> failwith "Erreur dans la rotation gauche (Vide)"
  | N(n,fg,fd) -> match (balance fd) with
          | -1 -> N(get_elem (get_fd fg),
                    N(get_elem fg,get_fg fg,get_fg (get_fd fg)),
                    N(n,get_fd (get_fd fg),fd))
          |  1 -> N(get_elem fg,
                     get_fg fg,
                    N(n,get_fd fg,fd))
          |  _ -> failwith "Erreur dans la rotation gauche :
                  les fils ne sont pas des AVL"
;;
```

Exercice 8 : Implémentez la fonction `get_ind` qui prend en argument un objet de type `info` et renvoie la valeur selon laquelle les noeuds de l'arbre seront triés.

NB : Pour le moment cette valeur est l'entier `id` de l'objet info.

```
let get_ind inf =
  inf.id
;;
```

Exercice 9 : Implémentez la fonction `ajout` qui prend en argument un arbre et un objet de type `info` et qui ajoute cet objet à l'arbre en respectant les règles des arbres AVL.

```
let rec ajout noeud elem =
  match noeud with
  | Vide -> N(elem,Vide,Vide)
  | N(n,fg,fd) -> if (get_ind elem) < (get_ind n) then
          begin
            let aux = ajout fg elem in
            match ((prof aux)-(prof fd)) with
            | -1 | 0 | 1 -> N(n,aux,fd)
            | -2 -> right_case (N(n,aux,fd))
            |  2 -> left_case (N(n,aux,fd))
            |  _ -> failwith "Erreur dans l'insertion (<)"
          end
          else if (get_ind elem) > (get_ind n) then
          begin
            let aux = ajout fd elem in
            match ((prof fg)-(prof aux)) with
            | -1 | 0 | 1 -> N(n,fg,aux)
            | -2 -> right_case (N(n,fg,aux))
            |  2 -> left_case (N(n,fg,aux))
            |  _ -> failwith "Erreur dans l'insertion (>)"
          end
          else
            N(n,fg,fd)
;;
```

Voici une fonction d'affichage qui peut éventuellement être utile pour vérifier votre arbre.

```
let rec aux noeud s =
  match noeud with
  |Vide -> ()
  |N(n,fg,fd) -> aux fd (s^"    ");
    print_endline (s^("-> n = "^(string_of_int (get_ind n))));
    aux fg (s^"    ");
;;
let print noeud =
 aux noeud ">  "
;;
```

3 Des skyscrapers...

Nous avons maintenant une structure d'arbre. Quoi de plus normal que de l'utiliser pour fabriquer notre Skyline... Changeons notre type `info` pour qu'il accueille les carac-

téristiques d'un immeuble (`prof` pour la profondeur, `dec` pour le décalage en x, `lar` pour la largeur, `haut` pour la hauteur et `ant` pour savoir si oui ou non celui-ci possède une antenne).

```
type info = {prof: int; dec: int; lar: int;
             haut: int; ant: bool};;
```

Si tout est bien fait, seule la fonction `get_ind` est a changer.

Exercice 10 : Changer la fonction `get_ind` pour que l'arbre AVL de gratte-ciel soit trié selon la profondeur.

```
let get_ind inf =
  inf.prof
;;
```

Maintenant que nous avons tout à disposition, créons nos grattes-ciel... Le mieux c'est d'introduire un peu d'aléatoire.

Exercice 11 : Implémentez la fonction `rand_sky` qui prend en argument un entier `n` et crée un arbre AVL de `n` objets `info` alétoires et renvoie l'arbre créé.

```
let rand_sky n tree =
  let avl = ref tree in
  for k = 1 to n do
    avl := ajout (!avl) {prof = (rand 0 40);
                         dec = (rand 10 500);
                         lar = (rand 30 100);
                         haut = (rand 100 500);
                         ant = Random.bool()}
  done;
  (!avl)
;;
```

```
(* quelques fonctions utiles *)
Random.bool();;
let rand a b =
  a+(Random.int (b-a))
;;
```

Voila, il ne reste plus qu'à afficher la skyline.

Exercice 12 : Implémentez la fonction `drawSky` qui prend en argument un objet de type `info` et dessine le gratte-ciel correspondant.

```
let drawSky s =
  let col = (255-(200*s.prof)/40) in
  set_color (rgb col col col);
  fill_rect s.dec 0 s.lar s.haut;
  set_color black;
  draw_rect s.dec 0 s.lar s.haut;

  let top = [|((s.dec+(2*s.lar/5)),s.haut);
              ((s.dec+(s.lar/2)),s.haut+100);
              ((s.dec+(3*s.lar/5)),s.haut)|] in
  set_color red;
  if (s.ant = true)
  then
    fill_poly top
;;
```

Exercice 13 : Implémentez la fonction `drawSkyLine` qui prend en argument un arbre de gratte-ciel et les dessine.

```
let rec drawSkyLine noeud =
  match noeud with
  | Vide -> moveto 0 0
  | N(n,fg,fd) -> drawSkyLine fd; drawSky n; drawSkyLine fg
;;
```

NB : parcourez judicieusement votre arbre. Les grattes-ciel les plus profonds doivent être plus sombres et... au fond !

```
#load "graphics.cma";;
open Graphics;;
open_graph " 640x600";;
```

```
(* quelques fonctions utiles *)
clear_graph();;
fill_rect x y w h;;
set_color (rgb r g b);;
(* allez voir la fonction fill_poly *)
```

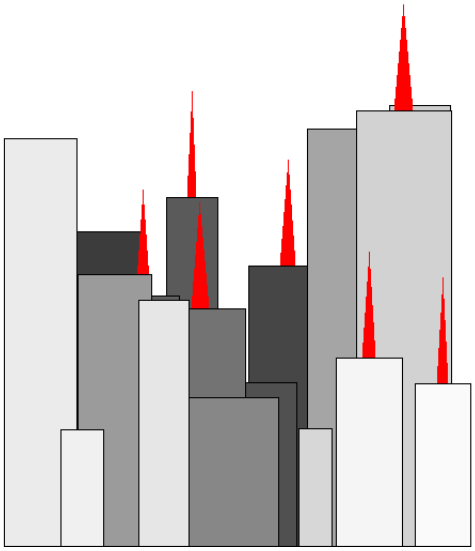

FIGURE 2: Exemple de skyline