

Algorithmique et Approche Fonctionnelle

Cours 10 : Arbres équilibrés

24 Novembre 2008

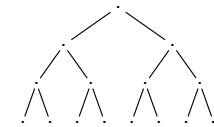
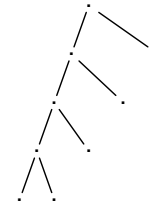
les arbres AVL

- premiers arbres binaires **équilibrés**
- inventés en 1962 par les russes Adelson-Velsky et Landis (d'où le nom AVL)
- ces arbres vérifient la propriété suivante :

la différence entre les hauteurs des fils gauche et des fils droit de tout nœud ne peut excéder **1**

rééquilibrage des arbres de recherche

- l'efficacité de la recherche dans un arbre binaire ordonné dépend fortement de la forme de l'arbre
- la forme **la pire** est celle du peigne : un arbre où chaque nœud n'a qu'un seul successeur gauche ou droit (l'arbre n'est alors ni plus ni moins qu'une liste); l'opération de recherche est alors en $O(n)$
- la forme **la meilleure** est celle de l'arbre "équilibré", i.e. où $taille \approx 2^{prof}$ soit $prof \approx \log_2(taille)$



il faut donc essayer de maintenir l'équilibre des arbres binaires de recherche dans les opérations d'ajout, de suppression etc.

définition des arbres AVL

- le type des AVL est similaire à celui des arbres binaires
- on ajoute un entier dans les nœuds afin de mémoriser la hauteur des arbres

```
type 'a arbre = Vide | Noeud of 'a arbre * 'a * 'a arbre * int
```

on manipule les hauteurs des arbres à l'aide des deux fonctions suivantes

```
let height = fonction
  Vide -> 0
  | Noeud(_, _, _, h) -> h

let creation l v r =
  Noeud(l, v, r, 1 + max (height l) (height r))
```

ajout d'un élément

```

let rec ajout x = function
  Vide ->
    creation Vide x Vide

  | Noeud(l, v, r, _) as t ->
    if x = v then t else
      if x < v then
        equilibrage (ajout x l) v r
      else
        equilibrage l v (ajout x r)
  
```

équilibrage

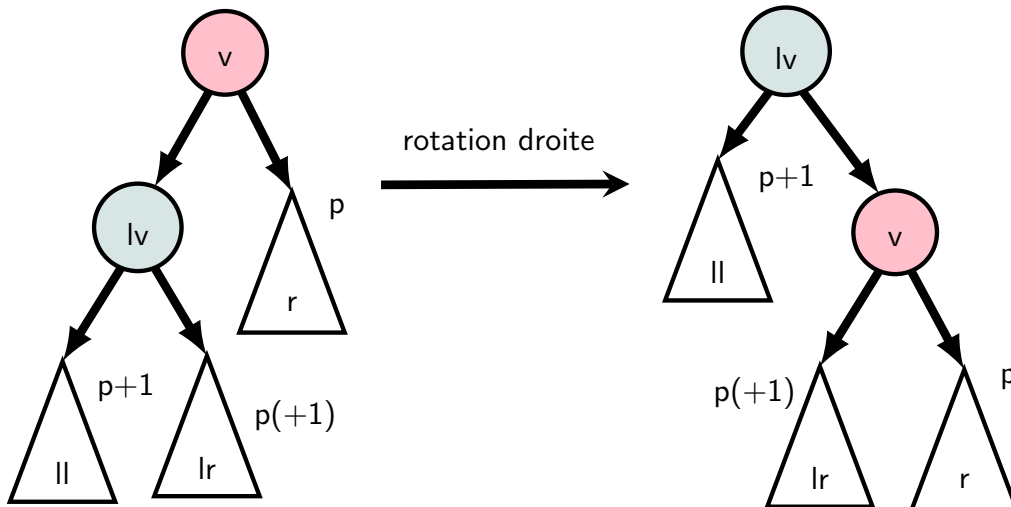
- soient deux arbres l et r équilibrés tels que $|\text{prof}(l) - \text{prof}(r)| \leq 2$
- la fonction `equilibrage` construit un arbre équilibré formé des mêmes éléments, et dans le même ordre, que `Noeud(l, v, r)`

```

let equilibrage l v r =
  let hl , hr = height l , height r in
  if hl > hr + 1 then begin
    match l with
    | Noeud(l1, lv, lr, _) when height l1 >= height lr ->
      creation l1 lv (creation lr v r)
    | Noeud(l1, lv, Noeud(lr1, lrv, lrr, _),_) ->
      creation (creation l1 lv lr1) lrv (creation lrr v r)
    | _ -> failwith "equilibrage"
  end else if hr > hl + 1 then begin
    (* cas symetrique *)
  end else creation l v r
  
```

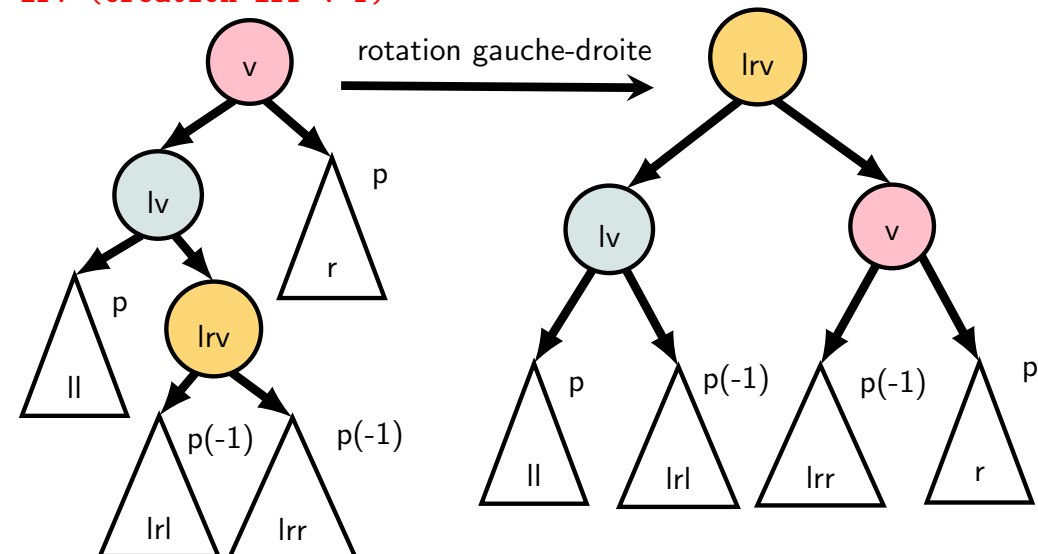
équilibrage à droite par simples rotations

si $h_l = h_r + 2$ et $\text{height}(l_1) = p + 1$ alors on réalise une rotation simple à droite `creation l1 lv (creation lr v r)`



équilibrage à droite par doubles rotations

si $h_l = h_r + 2$ et $\text{height}(l_1) = p$ et $\text{height}(l_r) = p + 1$ alors on réalise une rotation double gauche-droite `creation (creation l1 lv lr1) lrv (creation lrr v r)`



la suppression d'un élément x dans un AVL $\text{Noeud}(l, v, r, h)$ consiste à :

- fusionner l et r si $x=v$
- supprimer x dans l ou r selon la valeur de $x < v$
- re-équilibrer l'arbre obtenu

```
let rec suppression x = fonction
  Vide -> Vide
  | Noeud(l, v, r, _) ->
    if x = v then fusion l r else
    if x < v then equilibrage (suppression x l) v r else
    equilibrage l v (suppression x r)
```

- soient deux AVLs $t1$ et $t2$ tels que $|\text{height}(t1) - \text{height}(t2)| \leq 1$
- tous les éléments de $t1$ sont plus petits que les éléments de $t2$
- la fonction `fusion` construit un AVL formé des mêmes éléments que $t1$ et $t2$

```
let fusion t1 t2 =
  match (t1, t2) with
  (Vide, t) | (t, Vide) -> t
  | (_, _) -> equilibrage t1 (min_elt t2) (suppr_min_elt t2)
```

la fonction `min_elt` retourne le plus petit élément d'un arbre binaire de recherche

```
let rec min_elt = fonction
  Vide -> failwith "min_elt"
  | Noeud(Vide, v, r, _) -> v
  | Noeud(l, v, r, _) -> min_elt l
```

la fonction `suppr_min_elt` supprime le plus petit élément d'un AVL

```
let rec suppr_min_elt = fonction
  Vide -> failwith "suppr_min_elt"
  | Noeud(Vide, v, r, _) -> r
  | Noeud(l, v, r, _) -> equilibrage (suppr_min_elt l) v r
```

LGL : Langues et Génie Logiciel

- Programme :
 - compléments de Caml : traits impératifs, compilation, modules
 - principes de développement de logiciels
 - graphisme, réseau
- cours/TD (pas de cours d'amphi)
- projet de programmation en plusieurs phases sur tout le semestre

cette année : le jeu **worms**

