Cours Programmation
MIM 1, 2000–2001, Devoir 2
Arbres binaires de recherche

Luc Bougé, Alice Bonhomme
Conception initiale : Frédéric Vivien, octobre 1995

20 octobre 2000

Présentation du devoir

Ce devoir est à faire par groupes de 2. Il est à rendre pour le

vendredi 10 novembre

Vous devez rendre pour chaque groupe :

- une description de la méthode de programmation utilisée ;
- les listings des programmes ;
- une copie d'écran des sorties (utiliser le programme snapshot du menu Workspace/ Programs du fond d'écran).

Au cours du TP suivant, chaque groupe devra faire la démonstration de son programme.

L'ensemble est évalué en tout à 6 heures de travail et 2 heures de rédaction pour chacun. Pour toute difficulté, contactez votre "TDman" par courrier électronique :

Luc.Bouge@ens-lyon.fr, Alice.Bonhomme@ens-lyon.fr

Une documentation complète sur le graphisme en *Objective Caml* est disponible à l'URL suivant :

http://caml.inria.fr/ocaml/htmlman/

Attention! *La qualité de la rédaction est très importante. La notation prend en compte le soin, la structure et l'orthographe. En particulier, l'aspect "esthétique" de vos programmes est déterminant : indentation, aération, lisibilité, choix des identificateurs, etc. Vos programmes doivent être aussi élégants qu'une démonstration de maths !*

1 Affichage graphique d'un arbre

Écrire une procédure d'affichage graphique d'un arbre binaire. En montrer les qualités et les limites sur divers exemples : arbres très petits, très grands, très profonds, très larges, etc.

Attention! *À vous de proposer un jeu de tests* argumentés *et de bien en tirer les conclusions !*

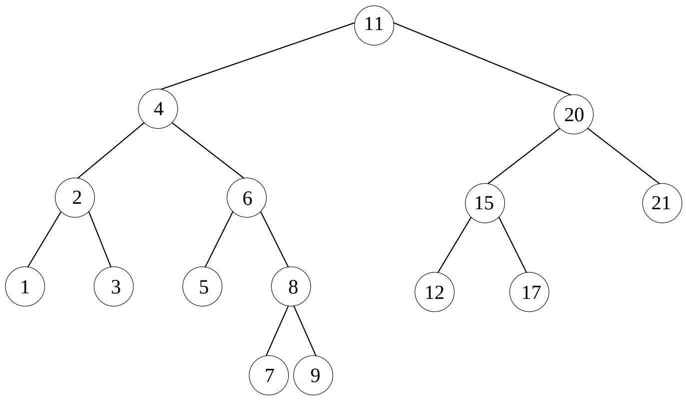

FIG. 1: Exemple d'arbre AVL

2 Parcours d'arbres

Écrire deux fonctions de parcours d'arbres binaires : l'une en profondeur préfixe (parcours du sous-arbre gauche, puis du du sous-arbre droit) ; l'autre en largeur préfixe (le fils gauche, puis le fils droit, puis les petit-fils, etc.). Faire varier la couleur du nœud visité pour rendre lisible le parcours sur l'affichage graphique. Faire une démonstration sur des arbres de grande taille. À vous de bien choisir votre échantillon !

3 Recherche d'une clé

Écrire une fonction de recherche d'une clé dans un arbre binaire de recherche. Afficher en rouge du nœud contenant la clé, et en vert les nœuds visités. Compter le nombre de nœuds visités. À vous de voir comment visualiser cette information de manière pertinente !

4 Définition des AVL

Les AVL sont des arbres binaires de recherches particuliers. Soit $t = (\text{clé}, t_1, t_2)$ un arbre binaire. Définissons δ la fonction qui à tout arbre associe sa fonction de déséquilibre à la racine :

$$\delta(\text{clé}, t_1, t_2) = \text{hauteur}(t_1) - \text{hauteur}(t_2)$$

Dans notre cas, chaque nœud de l'arbre contiendra les champs suivants : key, color, delta, left_son, right_son.

Un arbre binaire de recherche est un arbre *AVL* si pour chacun de ses sous-arbres la fonction δ vaut au plus un en valeur absolue. La figure 1 montre un arbre AVL. Vous trouverez une introduction détaillée à cette catégorie d'arbres de recherche dans tous les livres d'algorithmique sur les arbres. À vous de chercher, ô futurs chercheurs !

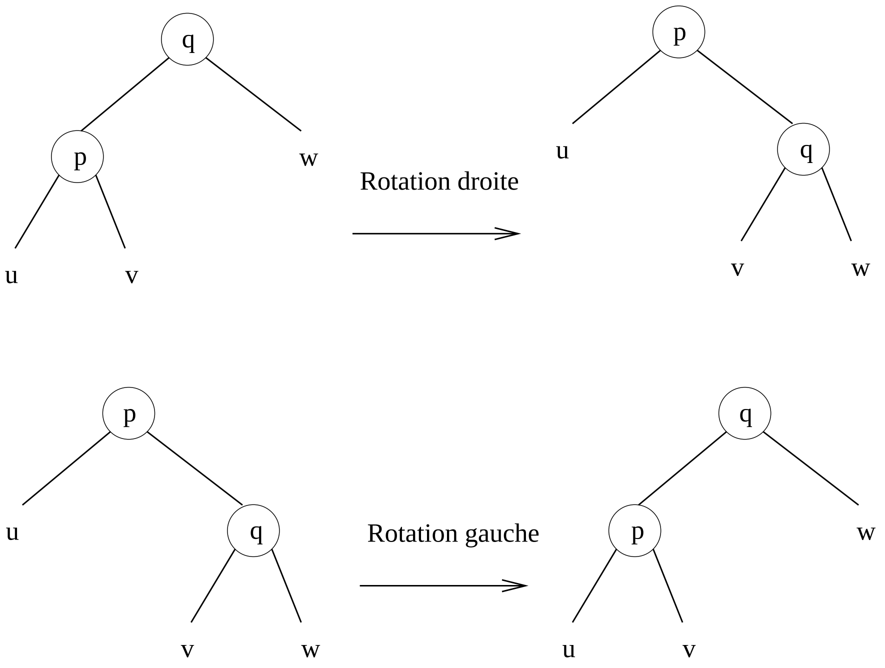

FIG. 2: Rotations sur les AVL

5 Insertion d'une clé dans un AVL

Algorithme. Soit $t = (clé, t_1, t_2)$ un arbre AVL. Soit *nouvelle*, la clé à insérer. On suppose que *nouvelle* n'est pas déjà présente dans t. Si $nouvelle < clé$, l'ajout s'effectuera dans le sous-arbre t_1 ce qui fournira un arbre t'_1. Plusieurs cas de figures se présentent :

- Si hauteur(t'_1) = hauteur(t_1), le résultat de l'ajout sera $(clé, t'_1, t_2)$
- Si hauteur(t'_1) = hauteur$(t_1) + 1$, et si $\delta(t) = 0$ ou $\delta(t) = -1$, le résultat de l'ajout sera encore $(clé, t'_1, t_2)$.Il faudra toutefois positionner correctement l'information d'équilibre à la racine de l'arbre résultat.
- Si hauteur(t'_1) = hauteur$(t_1) + 1$, et si $\delta(t) = 1$, l'arbre $t' = (clé, t'_1, t_2)$ n'est pas un AVL car $\delta(t') = 2$. Il convient de le rééquilibrer en utilisant les rotations. Ici, il faut considérer l'équilibre de l'arbre t'_1. Cet arbre ne peut pas être équilibré. S'il l'était, il aurait la même hauteur que t_1. Il n'y a donc que deux cas possibles :
  - $\delta(t'_1) = 1$. t' peut être rééquilibré par une rotation droite. Le résultat de l'ajout sera donc $rd((clé, t'_1, t_2))$.
  - $\delta(t'_1) = -1$. t' peut être rééquilibré par une rotation gauche sur t'_1 suivie d'une rotation droite sur l'arbre tout entier . Le résultat de l'ajout sera donc $rd((clé, rg(t'_1), t_2))$.
- Les autres cas sont symétriques des précédents.

Notez qu'un ajout donne lieu à au plus une opération de rotation (simple ou double). Les rotations gauches et droites sont définies sur la figure 2. Écrire une fonction d'insertion d'une clé dans un AVL. Afficher en couleur la partie de l'arbre modifiée. Faire une démonstration sur un échantillon représentatif d'arbres mettant en action les différents cas.

Si vous avez le temps... Suppression d'une clé d'un AVL

L'opération de retrait dans un arbre AVL repose, comme le retrait dans les arbres binaires de recherche, sur le remplacement de l'élément à éliminer par le plus grand élément du fils gauche du nœud où se trouve cet élément.

L'élimination du plus grand élément d'un arbre est susceptible de réduire la hauteur du sous-arbre droit de cet arbre et de modifier l'équilibre de l'arbre tout entier.

La fonction d'élimination du plus grand élément devra donc retourner également une information supplémentaire indiquant si la taille de l'arbre a décru ou non.

La fonction principale d'élimination recherche l'élément *usagé* à éliminer dans l'arbre, et, lorsqu'il est trouvé, appelle la fonction d'élimination du plus grand élément sur le sous-arbre $(\textit{usagé}, t_1, t_2)$. Cet appel renvoie un arbre $(\textit{plus\_grand\_élément\_ de\_t\_1}, t'_1, t_2)$ et une information sur la variation de hauteur entre t_1 et t'_1. Cette dernière information est utilisée ensuite pour équilibrer le cas échéant l'arbre $(\textit{plus\_grand\_ élément\_de\_t\_1}, t'_1, t_2)$.

De plus le rééquilibrage de l'arbre $(\textit{plus\_grand\_élément\_de\_t\_1}, t'_1, t_2)$ peut produire un arbre de taille inférieure à $(\textit{usagé}, t_1, t_2)$ qu'il remplace et on doit donc répercuter cette information de façon à procéder à d'autres rééquilibrages en remontant vers la racine.

Et si vous avez encore du temps... Test de performance

Utiliser une fonction CAML pour produire des clés aléatoires en très grand nombre (à vous de choisir le nombre !) et les insérer successivement dans un arbre initialement vide. Mesurez pour chaque série le temps *utilisateur*. Pour un nombre donné de clés, faites de nombreuses expériences (à vous de choisir combien !) et tracer sur un graphique les différents temps observés. Calculer le temps moyen, et le visualiser sur le graphique. (Utiliser le logiciel gnuplot et son aide en ligne par commande help.) Interprétez les résultats !

Répétez cette série d'expériences pour des nombres de clés différents, et tracer la courbe qui donne le temps de construction moyen de l'arbre en fonction du nombre de clés.

Bien sûr, tout ce travail doit être automatisé. La production du fichier de données gnuplot sa visualisation doivent être lancées *automatiquement* depuis CAML, sans intervention de votre part.

Vous aurez à utiliser les fonctions de mesure du temps (fonction Unix.times) et d'interaction avec Unix (fonction Unix.system) de la bibliothèque Unix en CAML (chapitre 20). Ceci exige de fabriquer un *toplevel* CAML spécifique, comme pour l'utilisation de la libraire graphique. Le temps qui vous intéresse est le temps utilisateur Unix.tms_utime, en secondes.

```
shadoks% ocamlmktop -o mytop unix.cma graphics.cma
shadoks% ./mytop
        Objective Caml version 3.00
# open Unix;;
# let rec f n = if (n > 0) then (f (n-1); f (n-1));;
val f : int -> unit = <fun>
# let t1 = (times()).tms_utime
  in let result = (f 25)
  in let t2 = (times()).tms_utime
  in begin
      print_float (t2 -. t1);
      print_newline()
  end;;
```

```
0.59
- : unit = ()
# Unix.system "latex devoir2";;
[...]
- : Unix.process_status = Unix.WEXITED 0
```