Option Informatique
Arbres Binaires de Recherche

Corrigé

23 octobre 2005

1 Retour sur les arbres binaires

1.1 Tri par ABR

1. `type 'a arbre = feuille | noeud of ('a arbre) * 'a * ('a arbre) ; ;`
2.

```
let rec Max x =
  match x with
    | feuille → failwith "arbre vide" ;
    | noeud (g,r,feuille) → r ;
    | noeud (g,r,d) → (max r (Max d)) ; ;

let rec Min x =
  match x with
    | feuille → failwith "arbre vide" ;
    | noeud (feuille, r, d) → r ;
    | noeud (g,r,d) → (min r (Min g)) ; ;
```

   Les fonctions `Max` et `Min` sont en $O(h)$.

3.

```
let rec insere e T =
  match T with
    | feuille → noeud (feuille, e, feuille) ;
    | noeud (g,r,d) as a →
        if e < r then
          noeud ((insere e g),r,d)
```

```
        else
          if e>r then
            noeud (g,r,(insere e d))
          else
            a ; ;
```

4.

```
let rec successeur x T =
  match T with feuille → feuille ;
    | noeud (g,r,d) when r < x → (successeur x d) ;
    | noeud (g,r,d) when x = r → T ;
    | noeud (feuille,r,d) when r > x → T ;
    | noeud (noeud a,r,d) when r> x →
        let (G,R,D) = a in
        if (R>x) then
          (successeur x (noeud a))
        else
          T ; ;
```

Ecrire `predecesseur` revient à intervertir les rôles :
- des symboles d'ordre `<` et `>`.
- de `g` et `d`.

5. Il suffit d'insérer un à un les éléments du tableau dans un arbre binaire initialement vide. On applique n fois la fonction `insere`, ce qui donne une complexité en $O(nh)$. Puis on vide les éléments de l'arbre dans un nouveau tableau de taille n, à l'aide de la fonction `successeur`. Ceci s'effectue à nouveau en temps $O(hn)$, d'où une complexité totale en $O(hn)$.

1.2 Dictionnaire

1.

Les éléments du tableau sont stockés dans un arbre binaire de la manière suivante :
- On marque les arcs "fils gauche" avec des 0, et les arcs "fils droit" avec des 1.
- Le chemin de la racine à un noeud arbitrairement choisi nous donne une suite de 0 et de 1 qui définit la clef de l'élément qui lui est associé de manière unique.

2.

```
let rec decode k T =
  match T with
    | feuille → failwith "arbre vide" ;
    | noeud (g,r,d) →
        let n = (string_length k) in
        if (n=0) then
          r
        else
          let queue = (sub_string k 1 (n-1)) in
          match (nth_char k 0) with
            | '0' → (decode queue g) ;
            | _ → (decode queue d) ; ;
```

3.

```
let rec encode B =
  let rec encode_aux B i =
    let n = (vect_length B) in
    if n = 0 then
```

```
    if i = 0 then
      feuille
    else
    failwith "Erreur : élément non présent"
  else
    if (2*i > n) then
      noeud (feuille, B.(i),feuille)
    else
      if (2*i+1 > n) then
        failwith "Erreur : tableau de taille incorrecte"
      else
        noeud  ((encode_aux B (2*i)),B.(i),(encode_aux B (2*i+1))) ;
in
(encode_aux B 1) ; ;
```

2 Centrale 2000, Algorithmique : Arbres rouges et noirs

2.1 Rappels et notations

2.2 Rotations

2.2.1 Rotation gauche

1. L'écriture de la fonction rotation gauche est naturelle :

```
let rotation_gauche a = match a with
ArbreVide → a
| Pointeur x →
  begin
    match x.droite with
        ArbreVide → a ;
      | Pointeur d →
          Pointeur

              etiquette = d.etiquette ;
              gauche = Pointeur etiquette = x.etiquette ;
                gauche = x.gauche ; droite = d.gauche ;
              droite = d.droite ;
           ;
  end ; ;
```

2. On suppose que A est un ABR. Notons B l'arbre obtenu) partir de A par rotation gauche et montrons que c'est un ABR.

   Si A est vide ou si son fils droit est vide alors $B = A$ et le résultat est acquis.

   Si A est non vide ainsi que son fils droit, avec les ntoations de l'énoncé :
   - Tout noeud n du fils droit de B vérifie $e(n) \leq e(i2)$ car le fils droit de A est un ABR.
   - Le fils gauche de B est un ABR car :
     - Tout noeud n de A_{g_1} vérifie $e(n) \leq e(i_1)$ car A est un ABR.
     - Tout noeud n de A_{g_2} vérifie $e(n) \geq e(i_2)$ car A est un ABR.
   - Tout noeud n du fils gauche de B vérifie $e(n) \leq e(i_1)$ car on a les trois cas :
     - $n \in A_{g_1}$ et A est un arbre binaire.
     - $n \in A_{g_2}$ et A est un arbre binaire.
     - $n = i_1$ et A est un arbre binaire.

2.2.2 Rotation droite

Pour que la rotation droite soit l'inverse de la rotation gauche, il suffit de reprendre le dessin de la rotation gauche et de changer le sens de la flèche ... Cela donne une écriture naturelle de la fonction :

```
let rotation_droite a = match a with
    ArbreVide → a
  | Pointeur x →
      begin
        match x.gauche with
            ArbreVide → a ;
          | Pointeur g →
              Pointeur

                  etiquette = g.etiquette ;
                  gauche = g.gauche ;
                  droite = Pointeur etiquette = x.etiquette ;
                    gauche = g.droite ; droite = x.droite ;
                ;
      end ; ;
```

2.2.3 Double rotations

L'écriture des deux opérations de double rotation ne pose aucun problème, ainsi que leur effet sur l'arbre exemple.

```
let rotation_gauche_droite a =
  match a with
      ArbreVide → a ;
    | Pointeur x → rotation_droite (Pointeur etiquette = x.etiquette ;
      gauche = (rotation_gauche x.gauche) ; droite = x.droite) ; ;

let rotation_droite_gauche a =
  match a with
      ArbreVide → a ;
    | Pointeur x → rotation_gauche (Pointeur etiquette = x.etiquette ;
      gauche = x.gauche ; droite = (rotation_droite x.droite)) ; ;
```

2.3 Arbres bicolores

2.3.1 Nouvelles représentations

2.3.2 Hauteur d'un arbre bicolore

1.

On désignera, dans toute la suite, par $t(n)$ le nombre de noeuds internes de l'arbre de racine n. Soit p un entier et H_p la propriété : pour tout noeud n avec $hn(n) = p$, le sous-arbre de racine n vérifie $t(n) \geq 2^p - 1$.

Montrons cette propriété par récurrence forte sur p.

- Si $p = 0$, le résultat est immédiat, car $2^{p-1} = 0$.
- Soit $p \geq 0$ fixé. Supposons la propriété vraie jusqu'au rang p, et montrons-la au rang $p + 1$.
  Soit n un noeud tel que $hn(n) = p + 1$. Montrons que $t(n) \geq Z^{p+1} - 1$.
  - Puisque $hn(n) > 0$, n n'est pas une feuille et a deux fils que l'on notera g et d.
  - On peut remarquer que $hn(d) = p$ si d est noir et $hn(d) = p + 1$ sinon. De même pour g. On va traiter deux cas.
  - *Premier cas :* g et d sont noirs.
    On a donc $hn(g) = hn(d) = p$ et par HR : $t(d) \geq 2^p - 1$ et $t(g) \geq 2^p - 1$. Mais $t(n) = 1 + t(d) + t(g) \leq 2^{p+1} - 1$.

- *Deuxième cas :* g ou d est blanc. Traitons le cas où g est blanc (l'autre cas étant symétrique).
  Puisque g est blanc, g n'est pas une feuille et a deux fils qui sont nécessairement noirs (règle A-4).
  Notons gg et dd ces deux fils : $hn(gg) = p$ et $hn(gd) = p$
  Par HR, on en déduit que $t(gg) \geq 2^p - 1$ et $t(gd) \geq 2^p - 1$ puis que $t(g) = 1 + t(gg) + t(gd) + \geq 2^{p+1} - 1$
  Or $t(n) > t(g)$. Donc $t(n) \geq 2^{p+1} - 1$.

2. Soit a un arbre bicolore comportant un noeud interne.
   Notons $h(n)$ la hauteur d'un noeud n, c'est à dire la hauteur d'un arbre associé.
   Notons r la racine de a : $t(r) = 1$ et $hn(r) \leq h(r)$ par définition de h et hn.
   D'après la question précédente : $t \geq 2^{hn(r)-1}$.
   Pour poursuivre, il nous faut une minoration de $hn(r)$ en fonction de $h(r)$.
   Soit e un chemin de r à une feuille de longueur $h(r)$: un tel chemin existe par définition de la hauteur d'un rabre.
   Notons $hb(r)$ le nombre de noeuds blancs situés sur ce chemin. On a $h(r) = hn(r) + hb(r)$.
   Mais à tout noeud blanc du chemin correspond au moins un noeud noir du chemin (son père) avec les règles (A-4) et (A-3).
   Donc $hb(r) \leq hn(r)$ puis $h(r) \leq 2hn(r)$ ou encore $hn(r) \geq \frac{h(r)}{2}$.
   On en déduit $h(r) \leq 2\log_2(l+1)$.
3. Un arbre bicolore est un cas particulier des arbres binaires de recherche. La complexité de la recherche d'un éléméent dans un tel arbre est donc proportionnelle à la hauteur de l'arbre. On en déduit que cette complexité est un $O(log_2 l)$, où t est le nombre de noeuds internes de l'arbre.

2.4 Insertion dans un arbre bicolore

1. si l'arbre a est vide, après insertion, on a $a = n$, et onaura un arbre bicolore en donnant la couleur noire à n.
2. Si a est non vide et si, après l'insertion initiale, le père de n est noir :
   L'insertion dans un arbre de recherche se fait aux feuilles de l'arbre. On remplace donc une feuille (noire) par un noeud blanc et deux feuilles (noires). La règle (A-5) n'est donc pas modifiée pour les chemins menant à ces deux nouvelles feuilles, ni bien entendu pour les autres chemins. L'arbre est donc bicolore puisque les autres règles ne sont pas affectées par l'insertion.
3. On suppose que le père de n est blan après l'insertion initiale.
   Le père de f étant blanc, avec la règle (A-4), f est noir. De plus c'est une feuille, car sinon le nombre de noeuds noirs sur un chemin issu de p serait égal à 2 au moins pour le chemin droit et 1 pour le chemin gauche.
4. On suppose que l'oncle o de n est blanc. La transformation peut affecter la règle (A-3) et/ou la règle (A-4). On aura donc un arbre bicolore si et seulement si q n'est pas la racine de l'arbre et si son père est noir.
5. Dans le cas contraire :

   - Si q est la racine : q devient noir.
   - Si $p(q)$ est blanc : on recommence comme en 3. avec comme noeud n le noeud q.

6. On suppose toujours que o est blanc.

   - Cas 2 : On fait une rotation sur le fils gauche et on se retrouve dans le cas 1.
   - Cas 3 : Changement des couleurs de o et p.
   - Cas 4 : Changement des couleurs de o et p.

7. Il s'agit de la rotation droite sur le noeud q.
8. L'arbre global est bicolore car :

   - C'est un arbre binaire de recherche comme image d'un tel arbre dans une opération de rotation.

- Les règles (A-1) à (A-4) sont clairement vérifiées.
- Pour la règle (A-5) : soit x un noeud du nouvel arbre et c un chemin de x à une feuille.
  - Le chemin est inchangé si cette feuille est différente de fg, fd, f, o. Donc le nombre de noeuds noirs est inchangé.
  - Sinon, le chemin est modifie, dans sa partie de p à une feuille qui est issue de la partie q à la même feuille dans l'arbre initial.

  Mais dans les qautre cas, La partie modifiée du chemin comporte toujours deux noeuds noirs. Le nombre total de noeuds noirs est donc toujours inchangé.

Comme l'arbre initia vérifiait la règle (A-5), l'arbre final la vérifie aussi.

9. Dans le cas 2, une rotation droite invalide la règle (A-5), car il y aura un noeud noir d'un côté et deux de l'autre. Toute modification de la couleur en respectant (A-4) ne résoud pas le problème (la vérification des cas est laissée au soin du lecteur).
10. Dans le cas 2, il suffit d'appliquer une rotation gauche sur l'arbre gauche pour se retrouver dans le cas 1. En tout, on fait une double rotation gauche-droite.
11. On le traite comme dans le cas 1 où o est blanc : double rotation droite-gauche.
12. Rotation gauche sur le fils droit, et on se retrouve dans le cas 3.