TD d'algorithmique numéro 5 : Arbres bien balancés
Daniel Hirschkoff
22 octobre 2000

Question 1

⋆ Arbres binaires de recherche ⋆

N.B.: pour cette question, on a utilisé l'exercice de Jean-Christophe Filliâtre sur les arbres binaires de recherche, que l'on peut consulter à l'adresse
`http://caml.inria.fr/polycopies/filliatre/index-fra.html`.

Nous allons travailler sur des *arbres binaires*, servant à stocker des éléments auxquels on accède par l'intermédiaire de *clefs*, et que l'on représentera en Caml par le type suivant:

```
type 'a arbre = Noeud of 'a arbre * 'a * 'a arbre | Feuille
```

On suppose disposer d'une relation d'ordre total sur les clefs. Une telle structure de données peut alors être utilisée pour faciliter le stockage et la recherche des éléments que les clefs représentent, par exemple dans un dictionnaire, ou dans la table de symboles d'un compilateur (la librairie `Set` d'O'Caml utilise aussi une forme d'arbre binaire balancé pour la représentation des ensembles).
Définition: Un arbre binaire de recherche est un arbre binaire tel que pour tout nœud de fils A_1 et A_2 et de clef k, on ait

$$\forall k_1 \in A_1.\, k_1 \leq k \quad \text{et} \quad \forall k_2 \in A_2.\, k \leq k_2\,.$$

Dans ce qui suit, on va faire l'hypothèse supplémentaire que l'on manipule des clefs qui sont toutes distinctes entre elles, ce qui nous permet de remplacer les inégalités larges par des strictes dans la propriété ci-dessus.

Diverses techniques existent pour traiter le cas général où des doublons sont admis. On peut enrichir le type des clefs pour assurer qu'elles sont toutes distinctes, ou bien stocker des listes chaînées à un nœud de l'arbre, ou alors bien entendu définir la structure de données et les algorithmes y afférants de manière à prendre en compte la possibilité de doublons.

Question 1.1

Écrire la fonction d'insertion d'une nouvelle clef dans un arbre binaire de recherche. Quelle en est la complexité?

L'insertion est-elle commutative, i.e. obtient-on le même arbre en insérant successivement les éléments de deux permutations d'une même liste de clefs? Quelle est la profondeur minimale d'un arbre binaire de recherche ayant k clefs? La profondeur maximale?

Question 1.2

Définir une fonction permettant de supprimer une clef d'un arbre binaire de recherche (la fonction renvoie l'arbre inchangé si la clef n'y figure pas).

Question 1.3

On suppose que la recherche d'une clef dans un arbre binaire de recherche se termine sur une feuille. On définit trois ensembles:

- A: ensemble des clefs situées à gauche du chemin de recherche;
- B: ensemble des clefs situées sur le chemin de recherche;
- C: ensemble des clefs situées à droite du chemin de recherche.

Est-il vrai que $\forall(a, b, c) \in A \times B \times C.\, a \leq b \leq c$? Si oui, démontrer la proposition, si non, donner un contre-exemple.

Question 1.4

Si on voulait écrire une version impérative des algorithmes ci-dessus, on pourrait écrire des choses de la forme suivante:

`recherche_itérative`$(x, k) :=$
  tant que $x \neq$ NIL et $k \neq clef[x]$
    faire si $k < clef[x]$
      alors $x \leftarrow f\_gauche[x]$
      sinon $x \leftarrow f\_droit[x]$
  renvoyer x

Quelles sont à votre avis les différences importantes entre approche impérative et approche fonctionnelle?

Question 2 — ⋆ Arbres Rouge et Noir ⋆

Définition: un arbre rouge et noir (*arbre RN*) est un arbre binaire de recherche vérifiant les propriétés suivantes:

1. chaque nœud est rouge ou noir;
2. chaque feuille est noire;
3. si un nœud est rouge, ses fils sont noirs;
4. tous les chemins issus d'un nœud quelconque vers l'une de ses feuilles comportent le même nombre de nœuds noirs (ce nombre est appelé *hauteur noire* du nœud en question).

Les arbres RN représentent un moyen de garantir qu'un arbre binaire de recherche ne soit pas trop déséquilibré. Il existe d'autres arbres dans cette famille, comme par exemple les arbres AVL (de Adel'son, Vel'skii et Landis, leurs inventeurs en 1960), définis de la manière suivante:

> *Un arbre AVL de hauteur 0 ou -1 est un arbre comportant une clef. Un arbre AVL de hauteur $h > 0$ est un nœud comportant une clef et ayant deux fils ayant chacun une hauteur égale à $h-1$ ou $h-2$.*

Question 2.1

Peut-on changer la couleur de la racine d'un arbre rouge et noir? Dessiner l'arbre binaire de recherche complet à 7 éléments. En le coloriant en arbre RN, combien de valeurs différentes peut-on obtenir pour la hauteur noire de l'arbre?

Question 2.2

Quelle est la valeur maximale du rapport entre la longueur de deux chemins issus d'un même nœud d'un arbre RN?

Minorer le nombre de nœuds internes d'un sous-arbre de racine x d'un arbre BN en fonction de la hauteur noire $hn(x)$ de x.

Question 2.3

Définir (en style fonctionnel) l'algorithme d'insertion d'une clef dans un arbre RN (en préservant la propriété RN, bien entendu).

Question 2.4

Quelle subtilité supplémentaire peut-on prendre en compte pour l'insertion d'une clef dans une version impérative de l'algorithme?

Question 3 $\star$ B-arbres $\star$

On abandonne maintenant les arbres binaires, et on chercher à équilibrer un arbre en agissant sur le degré de ses nœuds. Les B-arbres sont utilisés pour stocker des données sur des supportts tels que les disques magnétiques, dont les temps d'accès sont considérablement plus lents que pour la mémoire à accès aléatoire. Dans ce contexte, un facteur déterminant pour la performance est le nombre d'accès disque effectués, qui augmente avec la hauteur de l'arbre (on essaye donc de limiter cette dernière).

Un *B-arbre* T est caractérisé par les données suivantes:

- sa racine $racine[T]$;
- pour chaque nœud x:
  - $n[x]$: nombre de clés du nœud (les B-arbres ne sont pas binaires)

- $n[x]$ clés $k_1[x] \leq k_2[x] \leq \cdots \leq k_{n[x]}[x]$
- un booléen $feuille[x]$ qui indique si x est une feuille
- si x n'est pas une feuille, $n[x]+1$ pointeurs $c_1[x] \ldots c_{n[x]+1}[x]$ vers des sous-arbres, construits selon la définition des arbres binaires de recherche (les $n[x]$ clés définissent $n[x]+1$ intervalles, et quand on descend dans un fils on n'y trouve que des clés qui sont dans l'intervalle correspondant).

- toutes les feuilles ont la même profondeur $h[T]$
- a chaque B-arbre est associé un entier t appelé son *degré minimum* (souvent fixé en fonction des caractéristiques du support de stockage utilisé). À t sont associées les deux contraintes suivantes:
  - Tout nœud autre que la racine contient au moins $t-1$ clés (et donc s'il n'est pas feuille, il a au moins t fils). Si l'arbre n'est pas vide, la racine doit posséder au moins une clé.
  - Tout nœud peut contenir au plus $2t-1$ clés.

Remarque: un B-arbre avec $t = 2$ est appelé arbre 2-3-4, et est apparu avant les B-arbres dans la littérature. Les arbres 2-3-4 descendent eux-mêmes des arbres 2-3, inventés en 1970 par Hopcroft.

Question 3.1 Calculer la hauteur au pire des cas d'un B-arbre en fonction du nombre de clés n qu'il contient.

Question 3.2 Écrire l'algorithme de recherche et en calculer la complexité.

Question 3.3 Pour définir l'algorithme d'insertion, on commence par définir le découpage, qui, à partir d'un nœud complet ($2t-1$ clés), fait remonter la clef médiane et produit deux nœuds à $t-1$ clés (on remarque donc que le père, qui "reçoit" la clé qui remonte, ne doit pas être complet: c'est une condition que l'on va assurer lors des appels à cette fonction de découpage).

Écrire la fonction `découper_fils`(x, i, y), où x est un nœud incomplet, et $y = c_i[x]$ est un nœud complet (fils de x).

Écrire l'algorithme d'insertion, qui, en progressant, commence par couper en deux un nœud complet avant de descendre dans l'arbre.