IN 101 - TD 11
3 décembre 2010

Énoncé

ENSTA ParisTech

Exercice 1 : Hauteur d'un arbre binaire

On appelle hauteur d'un arbre la longueur du plus long chemin de la racine à l'une de ses feuilles.

1.a] Étant donné un arbre de hauteur h, quel est le nombre minimal d'éléments qu'il peut contenir ?

1.b] Étant donné un arbre de hauteur h, quel est le nombre maximal d'éléments qu'il peut contenir ?

1.c] Inversement, quelle est la hauteur minimale d'un arbre contenant n éléments ?

1.d] On considère maintenant uniquement des arbres binaires dont les nœuds ont soit deux fils (on parle alors de *nœud interne*), soit aucun fils (et on parle alors de *feuille*). Démontrer que le nombre de nœuds internes est égal au nombre de feuilles moins un.

Exercice 2 : Arbres binaires de recherche

On définit un *arbre binaire de recherche* comme un arbre binaire dont les clefs appartiennent à un ensemble totalement ordonné et qui satisfait la propriété suivante :

> *Toute clef d'un nœud de l'arbre est supérieure ou égale à celle des descendants de son fils gauche, et inférieure ou égale à celle des descendants de son fils droit.*

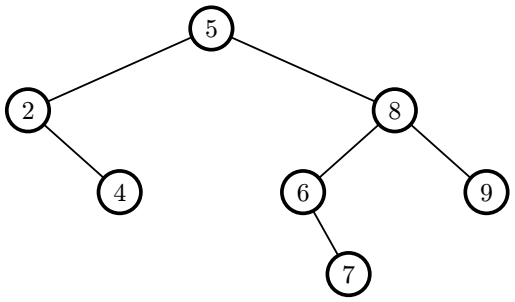

FIGURE 1 – Exemple d'arbre binaire de recherche obtenu en insérant les nœuds 5, 8, 2, 6, 9, 4 et 7 dans un arbre initialement vide.

2.a] Montrer que tout sous-arbre d'un arbre binaire de recherche est lui-même un arbre binaire de recherche.

2.b] Montrer qu'un parcours infixe d'un arbre binaire de recherche imprime les clefs dans l'ordre croissant. En déduire un nouvel algorithme de tri.

2.c] Écrire le pseudo-code d'une fonction recherchant un élément dans un arbre binaire de recherche. Quelle est la complexité dans le cas le pire de la recherche d'une clef ?

2.d] Écrire le pseudo-code d'une fonction insérant un élément dans un arbre binaire de recherche. Quelle est la complexité de l'opération d'insertion ?

2.e] En partant d'un arbre binaire de recherche initialement vide, on ajoute successivement les entiers suivants :

6, 8, 13, 7, 4, 1, 5, 11, 3, 14, 10, 2, 9, 12

Quel arbre obtient-on finalement ? Quelle est sa profondeur ?

2.f] Dans quel ordre s'affichent les éléments de cet arbre pour chacun des trois parcours *préfixe*, *infixe* et *postfixe* ?

2.g] On veut faire un parcours *en largeur* (par niveaux) de cet arbre binaire. Dans quel ordre les éléments doivent-ils s'afficher ? Comment peut-on implémenter un tel parcours ?

L'opération de *suppression* d'un élément dans un arbre binaire de recherche est plus subtile. Supposons que l'on souhaite supprimer d'un arbre le nœud contenant une clef donnée, on commencera par la rechercher en renvoyant le sous-arbre dont la racine contient la clef cherchée. Si la recherche est fructueuse, on sera alors amené à distinguer trois cas selon que le nœud à supprimer possède 2, 1 ou aucun fils.

2.h] Comment supprimer :
- un nœud sans fils ?
- un nœud ayant un seul fils ?
- un nœud ayant 2 fils ? (c'est plus difficile)

Quelle est la complexité de l'opération de suppression ?

2.i] Comment retirer la racine de l'arbre construit à la question 2.e ? Quel arbre binaire de recherche obtient-on après sa suppression ?

Exercice 3 : Dénombrement d'arbres binaires de recherche

On considère un arbre binaire de recherche construit en insérant successivement n nœuds comportant tous des valeurs distinctes. On suppose que les valeurs de ces nœuds sont dans un ordre aléatoire et on veut calculer la probabilité d'avoir un arbre bien équilibré (en fonction des $n!$ ordres possibles sur les nœuds d'entrée). On ne fait ici que des insertions, aucune suppression qui pourrait modifier l'ordre des nœuds.

3.a] Quelle est la probabilité que l'un des deux sous-arbres de la racine soit vide une fois tous les éléments insérés (et donc que la racine n'ai qu'un fils à la fin) ?

3.b] Si la racine n'a qu'un fils, quelle est la probabilité que ce fils n'ai lui aussi qu'un seul fils ? Déduisez-en la probabilité que l'arbre final soit une seul branche et hauteur $n - 1$.

3.c] Supposons que $n = 2p + 1$, quelle est la probabilité que les deux sous-arbres de la racine contiennent exactement p nœuds chacun ?

3.d] Si $n = 2^{h+1} - 1$, déduisez de la réponse précédente la probabilité que l'arbre ait une hauteur de h exactement (et donc que tous ses niveaux soient entièrement remplis).

Exercice 4 : Arbres pour la représentation d'expressions arithmétiques

On s'intéresse dans cet exercice à des arbres représentants des expressions arithmétiques.

- les nœuds internes de l'arbre peuvent contenir les symboles `+`, `-`, `*` ou `/` et avoir 2 fils, ou les symboles `exp` ou `log` et avoir un seul fils,
- les feuilles contiennent soit un nombre, soit le symbole `x`,
- un nœud contenant un `+` représente l'expression de la somme entre le fils gauche et le fils droit de ce nœud. Il en est de même pour les autre symboles.

Nous allons étudier comment implémenter de tels arbres pour évaluer une expression en une valeur de `x` donnée, ou pour dériver une expression par rapport à `x`.

4.a] Quels arbres représentent les expressions suivantes : $\frac{x+4}{3-x}$, $1 + e^{1-\log(x)}$ et $(x+1)^3$ (pensez à convertir les puissances en `exp` et `log`) ?

4.b] On veut évaluer une expression en une certaine valeur de x. Expliquez comment faire cela efficacement et écrivez un algorithme (en pseudo-code) qui prend en argument un arbre d'expression et une valeur a et évalue l'expression correspondant à l'arbre en $x = a$.

4.c] En suivant le même principe que pour l'évaluation, on peut facilement dériver une expression arithmétique par rapport à x à l'aide de son arbre. Écrivez une fonction qui prend un arbre (un pointeur sur la racine) en argument et construit l'arbre correspondant à la dérivé de l'expression puis le retourne. Vous pouvez supposer que vous avez accès à une fonction permettant de cloner un arbre et votre fonction devra regarder le contenu du nœud (si c'est un `+`, un `*`, un `exp`...) qu'on lui passe et construire (récursivement) l'arbre adéquate en fonction. Construisez les arbres correspondants aux dérivées des expressions de la question 4.a.

Exercice 5 : Pour aller plus loin : mise en œuvre des ABR

Compléter le squelette de programme suivant, implémentant divers algorithmes de gestion d'arbres binaires de recherche :

```
1  #include <stdio.h>
2
3  typedef struct st_node {
4    int key;
5    struct st_node *left;
6    struct st_node *right;
7  } node;
8
9  /* Construit un arbre a partir de la valeur v de la clef de la racine
10    et des sous-arbres droit (rs) et gauche (ls) */
11 node* build (int v, node* L, node* R) {
12   node* nw = (node*) malloc(sizeof(node));
13   nw->key = v;
```

```
14   nw->left = L;
15   nw->right = R;
16   return nw;
17 }
18 /* Impression infixe d'un arbre binaire */
19 void print_inorder (node* A) { ... }
20
21 /* Recherche de v dans l'arbre binaire de recherche A */
22 int search (int v, node* A) { ... }
23
24 /* Recherche du minimum dans un arbre binaire de recherche */
25 int minimum (node* A) { ... }
26
27 /* Recherche du maximum */
28 int maximum (node* A) { ... }
29
30 /* Comptage du nombre d'elements contenus dans l'arbre A */
31 int number_of_nodes (node* A) { ... }
32
33 /* Mesure de la hauteur de l'arbre A */
34 int height (node* A) { ... }
35
36 /* Insertion de v dans l'arbre A */
37 void insert (int v, node** A) { ... }
38
39 /* Recherche puis suppression de v dans l'arbre A */
40 int delete (int v, node** A) { ... }
41
42 /* Liberation complete de la memoire occupee par un arbre */
43 void free_tree (node* A) { ... }
```