TP 9

Juliusz Chroboczek

5 décembre 2023

On utilisera la structure de données suivante pour implémenter les arbres binaires d'entiers :

```
struct node {
   int v;
   struct node *left;
   struct node *right;
};
```

Exercice 1.

1. Écrivez une fonction `mknode` qui construit un nouvel arbre à partir d'une valeur et deux fils :

   ```
   struct node *mknode(int x,
                       struct node *left, struct node* right);
   ```

2. Écrivez une fonction `struct node* perfect_tree(int h)` qui construit un arbre complet de hauteur h dont les nœuds ont tous la valeur 1. (L'arbre de hauteur 0 est vide, l'arbre de hauteur 1 consiste d'un seul nœud.)
3. Ecrivez une fonction

   ```
   void print_infix(struct node *n);
   ```

   qui affiche la valeur des nœuds de l'arbre de racine n dans l'ordre infixe. Écrivez un programme qui affiche le contenu de l'arbre complet de hauteur 4, et assurez-vous qu'il contient $1 + 2 + 4 + 8 = 15$ nœuds. Que dit `valgrind`?
4. Écrivez une fonction

   ```
   void free_tree(struct node *n);
   ```

   qui libère l'espace mémoire occupé par l'arbre de racine n. Ajoutez un appel à cette fonction à votre programme, et vérifiez à l'aide de `valgrind` que tout l'espace est libéré

Exercice 2. Un arbre binaire de recherche (ABR) est un arbre binaire dont les nœuds respectent la contrainte suivante : chaque nœud porte une valeur qui est supérieure ou égale à toutes celles qui se trouvent dans fils gauche (pas seulement à la racine), et cette valeur est aussi strictement inférieure à toutes celles de son fils droit. Par exemple, l'arbre de la figure 1(a) est un ABR, celui de la figure 1(b) ne l'est pas.

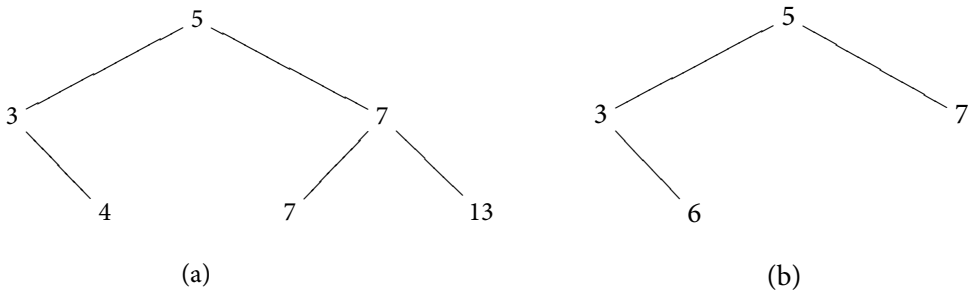

FIGURE 1 — Un ABR (a) et un arbre quelconque (b)

Pour trouver ou ajouter une valeur à un ABR, il suffit de parcourir une seule branche : la recherche et la mise à jour d'un ABR se font en moyenne en temps logarithmique. Par contre, il existe un cas pire (lequel ?) dans lequel l'ABR dégénère en une liste chaînée, et les accès se font alors en temps linéaire. (Il existe des raffinements des ABR qui n'ont pas ce problème, par exemple les arbres AVL et les *B-trees*.)

1. Écrivez une fonction

   ```
   int find(int i, struct node *abr);
   ```

   qui retourne vrai si et seulement si la valeur `i` est présente dans un ABR. Vous pouvez écrire votre fonction en style itératif ou en style récursif, mais dans ce dernier cas la récursion devra être linéaire. Vous n'avez pas besoin de vérifier que l'arbre passé en paramètre est un ABR.
2. Écrivez une fonction

   ```
   struct node *insert(int i, struct node *abr)
   ```

   qui insère la valeur `i` au bon endroit dans un ABR et retourne la racine du nouvel arbre.
3. Écrivez un programme qui lit un entier k, lit k entiers $a_1 \cdots a_k$, les stocke dans un ABR, puis lit un entier n et indique si n se trouve parmi $a_1 \cdots a_k$. Vérifiez à l'aide de `valgrind` que votre programme n'a pas de fuite de mémoire.

Exercice 3.

1. Écrivez une fonction qui prend en paramètre un arbre binaire et affiche la liste des valeurs qu'il contient en ordre infixe.
2. Écrivez un programme qui lit une suite d'entiers (supposés tous distincts) et qui, à chaque étape, affiche la liste d'entiers déjà rentrés en ordre croissant.