POLYTECH SORBONNE

Printemps 2023

Arbres et algorithme récursifs : Les arbres binaires de recherches

TD/TP 4

Objectifs

- ✯ Algorithmes récursifs
- ✯ Représentation des arbres
- ✯ Algorithmes sur les arbres
- ✯ Les arbres binaires de recherches

Les arbres binaires de recherche (BST)

Un arbre binaire de recherche (abr) sert surtout lorsque l'on a souvent besoin de chercher un élément parmi un ensemble trié et cet ensemble de données est dynamique, c'est à dire qu'il peut souvent être modifié. Les opérations *d'insertion* et de *suppression* sont alors importantes.

Définition 1 (Un arbre binaire de recherche)**.** *est une structure de donnée ordonnée abstraite avec l'interface suivante*

-
-
-

Un abr est donc un arbre avec des propriétés supplémentaires:

- Chaque clé est unique (Il n'y a pas plusieurs fois la même valeur).
- Le sous-arbre gauche d'un noeud contient toutes les valeurs inférieures au noeud.
- Le sous-arbre droit d'un noeud contient toutes les valeurs supérieur au noeud.

La figure 1 illustre cette propriété.

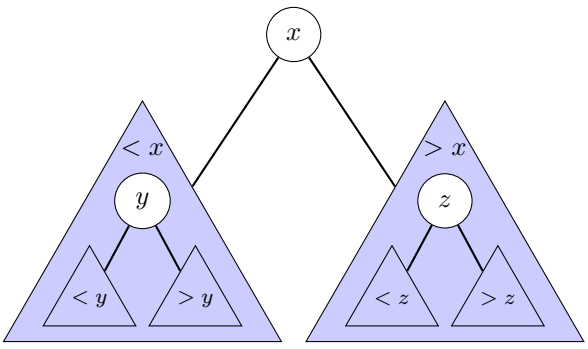

Figure 1: Illustration de la propriété des arbres binaires de recherche

La figure 2 est un exemple d'ABR. Les nœuds 5 et 16 peuvent être ajouté sans modifiés la propriété de l'arbre.

La hauteur d'un arbre peut être comprise entre n et $\log n$, dans la figure 2, l'arbre de gauche a une hauteur de et l'arbre de droite de hauteur .

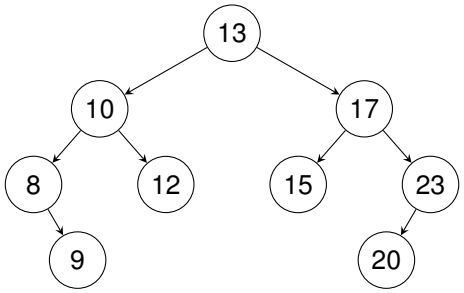

Figure 2: Exemple d'un arbre binaire de recherche

Implémentation d'un arbre binaire de recherche

Un arbre binaire est représenté par une structure qui contient la valeur du nœud et un pointeur sur son fils gauche, un pointeur sur son fils droit. Un arbre binaire en C peut être implémenté de la façon suivante (le type de contenu peut être définit ailleurs):

```
typedef struct s_btree {
  int key ;
  type_t content;
  struct s_btree * fd ;
  struct s_btree * fg ;
} btree ;
```

Dans la suite du TP on considérera que le `type_t` est un entier. Le code de la structure ainsi que la fonction de création de noeuds vous est déjà fournit. Un ensemble de fonctions de test vous est donnée, votre code devra donc passer au moins ces tests, vous pouvez bien évidemment en ajouter.

Exercice 1 Insérer des éléments

Les actions a effectué pour insérer un élément de l'arbre sont :

- chercher la place de l'élément : trouver le nœud parent :
  - Commence à la racine
  - traverse gauche ou droite
- Créer un nouveau nœud et affecter le fils du parent

Écrire une fonction récursive qui prend en paramètre un arbre et une valeur et insère le nœud correspondant à la nouvelle donnée.

Dans le fichier `main.c`, vous trouverez les tests d'insertion, pour ne pas avoir de fuit mémoire il faut aussi implémenter la fonction `deleteTree(btree*)`.

Exercice 2 Affichage

Question 1
Écrire une fonction qui affiche l'arbre en commençant par la racine puis les nœuds à gauche, puis ceux à droite. Pour l'arbre figure 2 : `13 10 8 9 12 17 15 23 20`

Question 2
Écrire une fonction qui affiche les valeurs dans l'ordre croissant.

Vous testerez vos fonctions dans le `main`.

Exercice 3 Chercher des éléments

Les actions à effectuer pour chercher un élément de l'arbre sont :

- Commence à la racine
- traverse gauche ou droite
- retourne True ou False

Écrire une fonction récursive `btree* search(btree*, int key)` qui prend en paramètre un arbre et une clé et retourne le noeud trouvé ou NULL.

Exercice 4 Min et Max

Écrire deux fonctions `btree* minBST(btree *)` et `btree* maxBST(btree*)` qui retournent le noeud avec la valeur minimale et maximale de l'arbre.

Exercice 5 Suppression

Écrire une fonction `btree* delete(btree*t, int key)` qui supprime le noeud correspondant à la clé.

Exercice 6 Arbre binaire de recherche équilibré

Un arbre binaire de recherche est dit parfaitement équilibré si pour tout noeud de l'arbre, le nombre de noeuds dans le sous-arbre gauche et le nombre de noeuds dans le sous-arbre droit diffèrent au plus de 1. Un arbre binaire de recherche est dit équilibré si pour tout noeud de l'arbre, la hauteur du sous-arbre gauche et la hauteur du sous-arbre droit diffèrent au plus de 1. On considère, dans la suite de l'énoncé, des arbres binaires de recherche équilibrés contenant des entiers.

Question 1
Décrire l'arbre binaire de recherche équilibré obtenu si on ajoute les nombres entiers de 1 à 10 dans un arbre binaire équilibré vide.

Question 2
Écrire une fonction qui retourne la hauteur d'un arbre binaire donné.

Question 3
Écrire une fonction qui permet de réaliser une rotation droite d'un arbre binaire. La figure 3 présente une telle rotation.

Question 4
Ecrire une fonction qui permet de réaliser une rotation gauche d'un arbre binaire. La figure 3 présente une telle rotation.

Question 5
Ecrire une fonction qui insère un élément dans un ABR équilibré et retourne un arbre équilibré. Pour ce faire, on insère tout d'abord le nouvel élément en utilisant la fonction d'insertion dans un ABR. Soit A, l'arborescence ainsi obtenue. Soit G le sous-arbre gauche de A et D le sous-arbre droit de A. On réorganise alors A de la manière suivante :

- Si $|h(G) - h(D)| < 2$, aucune réorganisation est nécessaire.
- Si $h(G) - h(D) = 2$, alors soient g et d les sous-arborescences gauche et droite de G. Si $h(g) < h(d)$ alors on effectue une rotation gauche de G. Dans tous les cas, on effectue une rotation droite de A.
- Si $h(G) - h(D) = -2$ alors soient g et d les sous-arborescences gauche et droite de D. Si $h(d) < h(g)$ alors on effectue une rotation droite de D. Dans tous les cas, on effectue une rotation gauche de A.

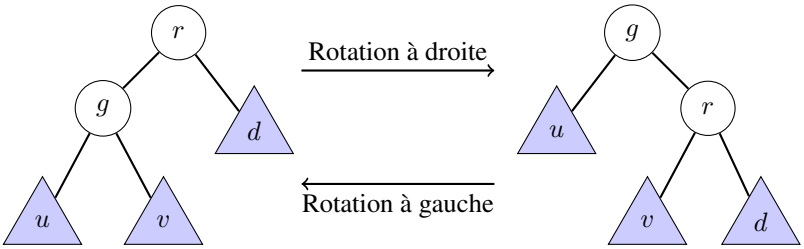

Figure 3: Rotation droite et gauche d'un arbre binaire