T.D. 9

Arbres binaires de recherche et arbres rouge noir

Arbres binaires de recherche : définitions

Un *arbre binaire de recherche* est un arbre binaire, dont les nœuds sont munis d'une clé dans un ensemble totalement ordonné, et qui vérifie la propriété suivante :
Soit x un nœud d'un arbre binaire de recherche. Si y est un nœud du sous-arbre gauche de x, alors clé$(y) \leq$ clé(x). Si y est un nœud du sous-arbre droit de x, alors clé$(y) \geq$ clé(x).

Convention : dans l'ensemble de ce TD, on considère que les nœuds d'un arbre binaire ont soit deux fils, soit aucun fils. Les nœuds sans fils sont les feuilles de l'arbre, notées `N` ou `NULL`, et ne possèdent pas de clé.

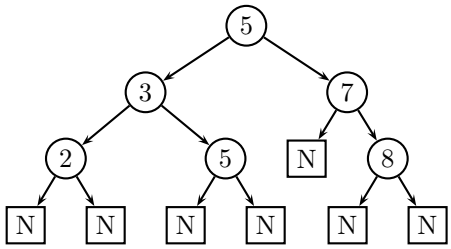

FIG. 1 – Un exemple d'arbre binaire de recherche

Exercice 1 : implémentation d'un arbre binaire de recherche

Question A : type

Proposer une définition de type en language `C` permettant de représenter des arbres binaires de recherche. On supposera que chaque nœud possède un champ `cle`, de type `unsigned int`, sur lequel se fait l'indexation de l'arbre, et un champ `element` contenant les données indexées, de type `element_t`.

Question B : parcours infixe

Proposer un algorithme permettant d'afficher toutes les clés d'un arbre binaire de recherche donné dans l'ordre croissant.

Question C : requêtes

Proposer des algorithmes permettant :
- de rechercher un élément repéré par sa clé dans un arbre binaire de recherche. L'algorithme retournera un pointeur vers le nœud contenant cet élément, s'il existe, ou le pointeur `NULL` sinon.
- de rechercher un élément de clé minimale dans un arbre binaire de recherche. L'algorithme retournera un pointeur vers le nœud contenant cet élément, s'il existe, ou le pointeur `NULL` si l'arbre est vide.
- de rechercher un élément de clé maximale dans un arbre binaire de recherche. L'algorithme retournera un pointeur vers le nœud contenant cet élément, s'il existe, ou le pointeur `NULL` si l'arbre est vide.

– de rechercher le successeur d'un élément `e`, c'est-à-dire un élément dont la clé est la plus petite supérieure à la clé de `e`. L'algorithme retournera un pointeur vers le nœud contenant cet élément, s'il existe, ou le pointeur `NULL` sinon.
– de rechercher le prédécesseur d'un élément. L'algorithme retournera un pointeur vers le nœud contenant cet élément, s'il existe, ou le pointeur `NULL` sinon.

et évaluer leur complexité.

Exercice 2 : insertion et suppression

Question A : insertion

Dans les deux exemples d'arbres binaires de recherche de la figure 2, où peut-on insérer un élément de clé 13 ? En déduire un algorithme d'insertion d'un nœud dans un arbre binaire de recherche, et évaluer sa complexité.

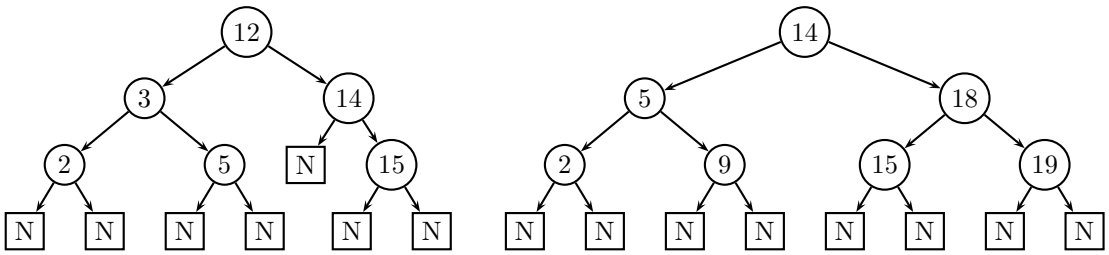

FIG. 2 – Deux arbres binaires de recherche

Question B : suppression

Dans les deux exemples d'arbres binaires de recherche de la figure 2, comment peut-on supprimer l'élément de clé 14 ? En déduire un algorithme de suppression d'un nœud dans un arbre binaire de recherche, et évaluer sa complexité.

Exercice 3 : rotations

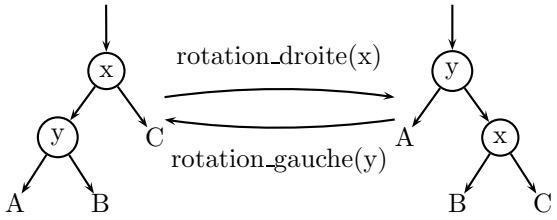

FIG. 3 – Rotations gauche et droite

1. Sur le premier exemple de la figure 2, faire : une rotation à droite de centre le nœud de clé 3 ; puis une rotation à gauche de centre le nœud de clé 14.
2. Sur le second exemple de la figure 2, à l'aide de rotations dont vous préciserez le sens et le centre, amener le nœud de clé 9 à la racine.
3. Démontrer que toute rotation préserve la propriété d'être un arbre de recherche.
4. Écrire l'algorithme de rotation à droite en `C`.

Arbres rouge noir : définitions

Un *arbre rouge noir* est un arbre binaire de recherche comportant un champ suppplémentaire par nœud : sa *couleur*, qui peut valoir soit `ROUGE`, soir `NOIR`.

En outre, un arbre rouge noir satisfait les propriétés suivantes :

1. Chaque nœud est soit rouge, soit noir.
2. Chaque feuille est noire.
3. Si un nœud est rouge, alors ses deux fils sont noirs.
4. Pour chaque nœud de l'arbre, tous les chemins descendants vers des feuilles contiennent le même nombre de nœuds noirs.
5. La racine est noire.

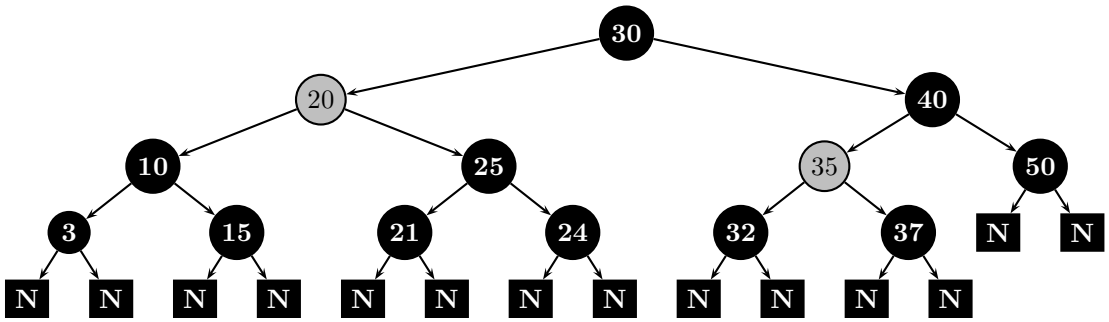

Fig. 4 – Un exemple d'arbre rouge noir

Soit x un nœud d'un arbre rouge noir. On appelle *hauteur noire* de x, notée $H_n(x)$, le nombre de nœuds noirs présents dans un chemin descendant de x (sans l'inclure) vers une feuille de l'arbre. D'après la propriété 4, cette notion est bien définie.

Exercice 1 : profondeur d'un arbre rouge noir

Le but de cet exercice est de montrer que la hauteur d'un arbre rouge noire est logarithmique en son nombre de nœuds.

Question A

Dans l'arbre rouge noir donné en figure 4, que valent $H_n(30), H_n(20), H_n(35), H_n(50)$?

Question B

Montrer que, pour un nœud x quelconque dans un arbre rouge noir, le sous-arbre enraciné à x contient au moins $2^{H_n(x)} - 1$ nœuds internes.

Question C

En déduire qu'un arbre rouge noir comportant n nœuds internes a une hauteur au plus égale à $2\lg(n+1)$.

Exercice 2 : insertion

Méthode générale. Pour réaliser l'insertion dans un arbre rouge noir : on applique l'insertion des arbres binaires de recherche ; puis on colorie en rouge le nouveau nœud ; et enfin on rétablit les propriétés d'arbre rouge noir.

Dans cet exercice on se propose d'établir l'algorithme qui permet de rétablir les propriétés d'arbre rouge noir.

1. Que faut-il faire lorsque l'arbre avant insertion ne contenait aucun élément ?
2. Que faut-il faire lorsque le parent du nœud que l'on vient d'insérer, N, est noir ?
3. On suppose désormais que le parent de N, P est rouge. Montrer que N a un grand parent G qui est noir et un oncle O.
4. Dans le cas où O est noir, montrer que vous pouvez rétablir les propriétés d'arbre rouge noir, par au plus deux rotations et deux recoloriages de noeuds.
5. Dernier cas : O est rouge. On remarque que P viole la condition 3 et que c'est la seule raison pour laquelle l'arbre n'est pas rouge noir. Que peut-on faire si G est la racine de l'arbre (indication : utiliser deux recoloriages) ? Autrement, comment peut-on repousser le problème que pose P, à un problème similaire sur un noeud plus haut dans l'arbre ?
6. Dans quels cas, la hauteur noire de l'arbre augmente ?
7. Écrire le code `C` de l'insertion dans un arbre rouge noir.