Arbres binaires de recherche et arbres rouge noir

Rappels de cours et correction du TD

Arbres binaires de recherche : définitions

Un *arbre binaire de recherche* est un arbre binaire, dont les nœuds contiennent des éléments 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).

Un sous-arbre d'un arbre binaire de recherche est encore un arbre binaire de recherche.

Fonction RECHERCHER(n)

Entrées : Le nœud racine r d'un arbre binaire de recherche et la clé d'un élément c
Sorties : Le nœud de l'arbre contenant l'élément de clé c s'il en existe un, ou le nœud vide N sinon.

```
si EstVide(r) alors
  retourner N ;
sinon
  si c = Clé(r) alors
    retourner r ;
  sinon
    si c < Clé(r) alors
      retourner Rechercher(Gauche(r), c) ;
    sinon
      retourner Rechercher(Droite(r), c) ;
```

Les opérations élémentaires sont la recherche d'un élément à partir de sa clé (pseudo code ci-dessus), l'insertion et la suppression d'un élément, ainsi que la recherche de l'élément maximum et la recherche de l'élément minimum. Ces opérations prennent un temps en $O(h)$ où h est la hauteur de l'arbre. On compte également dans les opérations élémentaires le test à vide qui s'exécute en temps constant.

La hauteur h peut varier entre $\log n$ et n où n est le nombre d'éléments de l'arbre. La coloration rouge noir vue plus loin est une manière d'assurer le fait que h soit en $\Theta(\log n)$ et qu'ainsi toutes les opérations élémentaires soient en $O(\log n)$.

Le parcours infixe de l'arbre donne la liste des éléments qu'il contient, rangés par ordre croissant et ceci en un temps $\Theta(n)$. On considère également deux opération supplémentaires en $O(h)$, successeur et prédecesseur, qui, étant donné un nœud de l'arbre donnent le nœud contenant l'élément qui lui succède, respectivement qui le précède.

Le code C vue en cours d'un implantation des arbre binaires de recherche se trouve sur la page web : `http://www-lipn.univ-paris13.fr/~boudes/`. Dans cette implantation la clé d'un élément est l'élément lui-même (simplification).

On peut accepter ou non les répétitions de clés dans un arbre binaire de recherche. Ici on considère que les arbre sont sans répétitions : l'insertion d'un élément déjà présent dans l'arbre ne modifie pas l'arbre.

Convention : Pour la suite, 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 contiennent pas d'élément. On ne compte pas ces feuilles dans la hauteur de l'arbre.

Exercice 1 : insertion et suppression

Question A : insertion

Dans les deux exemples d'arbres binaires de recherche de la figure 1, où peut-on insérer un élément de clé 13 ?

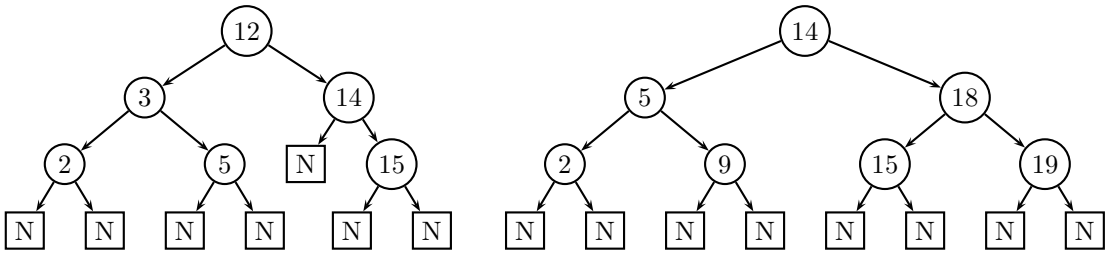

FIG. 1 – Deux arbres binaires de recherche

Correction

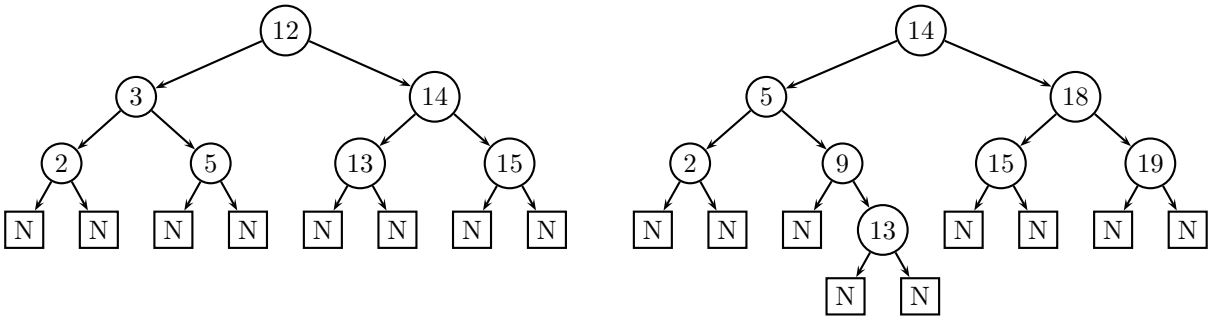

FIG. 2 – Insertion - `corrigé`

Question B : suppression

Dans les deux exemples d'arbres binaires de recherche de la figure 1, comment peut-on supprimer l'élément de clé 14 ?

Correction

L'idée de l'algorithme est la suivante : si le nœud à supprimer n'a pas de fils, on l'ôte. S'il n'a qu'un fils, on l'ôte et on rebranche. S'il a deux fils, on l'échange avec son successeur (ou son prédecesseur, ça marche aussi), qui n'a pas de fils, et on l'ôte.

FIG. 3 – Suppression - `corrigé`

Pour la fonction SUCCESSEUR y a deux cas de figure :
- soit le sous-arbre droit du nœud n'est pas vide, et dans ce cas il suffit de rechercher l'élément de clé minimale (le plus à gauche) dans ce sous-arbre droit.
- soit le sous-arbre droit du nœud est vide, et dans ce cas il faut rechercher le premier ancêtre dont le fils gauche est un ancêtre du nœud.

Mais lors de la suppression c'est toujours le premier cas qui s'applique.

Exercice 2 : rotations

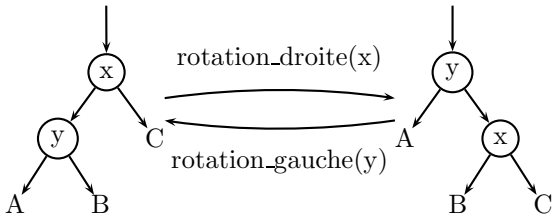

FIG. 4 – Rotations gauche et droite

1. Sur le premier exemple de la figure 1, 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 1, 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.

Correction

1. Figure 5.
2. rotation à gauche de centre 5, puis rotation à droite de centre 14.
3. Il faut montrer la préservation de la propriété d'arbre de recherche par passage d'un sous-arbre à l'autre. Pour le reste, l'arbre qui les contient ne fait aucune différence : les deux sous-arbres ont mêmes éléments. Pour chacun des deux sous-arbres on montre que le fait d'être un arbre de recherche est équivalent à la propriété : $a \in A$, $b \in B$, $c \in C$, $\text{cle}(a) \leq \text{cle}(x) \leq \text{cle}(b) \leq \text{cle}(y)$.

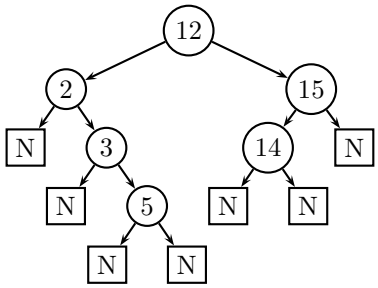

FIG. 5 – Question 3.1 - corrigé

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.

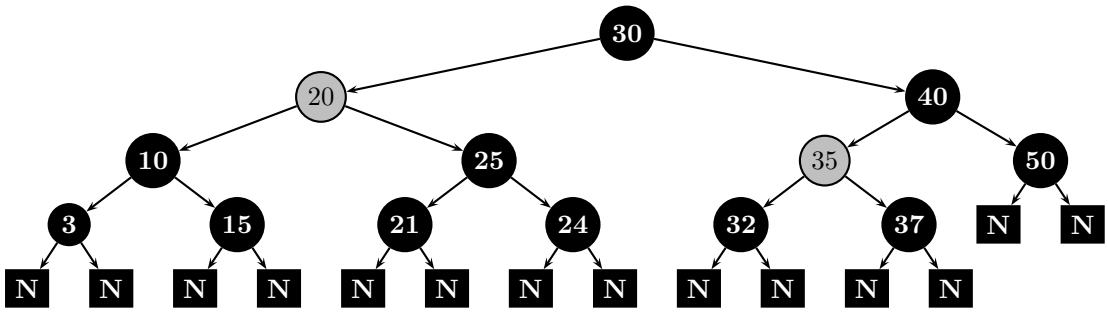

FIG. 6 – Un exemple d'arbre rouge noir

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 3 : 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 6, que valent $H_n(30), H_n(20), H_n(35), H_n(50)$?

Correction

$H_n(30) = 3$, $H_n(20) = 3$, $H_n(35) = 2$, $H_n(50) = 1$.

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.

Correction

On peut prouver cette proposition par induction sur la hauteur de x.
Si la hauteur de x est 0, x est une feuille et le sous-arbre enraciné à x contient 0 nœud interne, c'est-à-dire $2^{H_n(x)} - 1 = 2^0 - 1$.
Soit x un nœud de hauteur $h(x) > 0$. Ce nœud a deux fils, g et d. Dans ce cas, $H_n(g) \geq H_n(x) - 1$ et $H_n(d) \geq H_n(x) - 1$. De plus, la hauteur de g et de d est inférieure à la hauteur de x, donc, en appliquant l'hypothèse d'induction à g et d, les sous-arbres enracinés à g et d possèdent chacun au moins $2^{H_n(g)} - 1 \geq 2^{H_n(x)-1} - 1$ et $2^{H_n(d)} - 1 \geq 2^{H_n(x)-1} - 1$ nœuds internes. Il s'ensuit que le sous-arbre enraciné à x possède au moins $2(2^{H_n(x)-1} - 1) + 1 = 2^{H_n(x)} - 1$ nœuds internes.

Question C

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

Correction

Soit h la hauteur de l'arbre D'après la propriété 3, au moins la moitié des nœuds d'un chemin simple reliant la racine de l'arbre à une feuille doivent être noirs (preuve facile par l'absurde). En conséquence, la hauteur noire de la racine vaut au moins $h/2$, donc $n \geq 2^{h/2} - 1$, d'où $h \leq 2\log(n+1)$.

Exercice 4 : 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 ?

Correction

1. Il faut recolorier le nœud que le vient d'insérer en noir.
2. Deux cas sont possibles :
   - Soit l'autre fils, F, du parent P de N est une feuille (noire), et dans ce cas il ne faut rien faire.
   - Soit F est rouge, parent de deux feuilles (noires). Dans ce cas, il ne faut rien faire.
   - Tous les autres cas de figure (F noir non-feuille, ou F rouge parent de nœuds non-feuilles) entraînent une violation de la propriété 4 et sont impossibles.
3. par la propriété 5, P ne peut pas être la racine. N a donc un grand parent G. Puisque P est rouge, G ne peut pas être rouge aussi : cela violerait la propriété 3. G est donc noir. O est simplement l'autre fils de G. O peut être une feuille.
4. L'autre fils de P, F, est nécessairement une feuille (noire). En effet, si F est rouge, P viole la propriété 3. Si F est noir et n'est pas une feuille, P viole la propriété 4. De même, O est nécéssairement une feuille : sinon, G viole la propriété 4. Il suffit donc d'effectuer une rotation de centre P, de recolorier P en noir et G en rouge pour rétablir les propriétés souhaitées, suivant la figure 7. Le sens de la rotation dépend bien évidemment de positions de P parmis les fils de G.

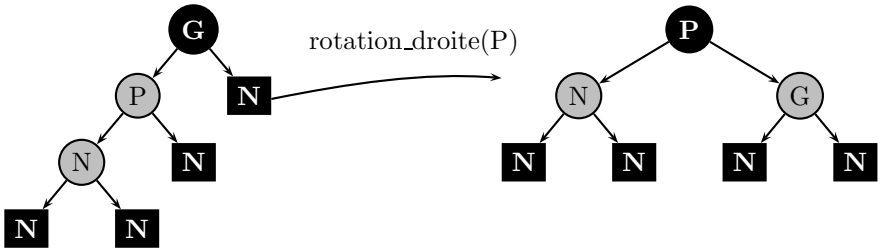

FIG. 7 – Cas où P est rouge et O noir.

5. Si G est la racine de l'arbre, il suffit de recolorier P et O en noir : cela ne change en rien la propriété 4, et rétablit la propriété 3 pour P.

   Si G n'est pas la racine de l'arbre, il faut recolorier P et O en noir, et G en rouge. Ensuite, il faut réevaluer le coloriage de l'arbre au-dessus de G, c'est-à-dire reconsidérer tous les cas ci-dessus en considérant G comme le nouveau nœud.

Exercice 5 : suppression

Même chose que pour l'insertion : on cherche un algorithme pour rétablir les propriétés d'arbre rouge noir après une suppression (on suppose donné l'algorithme de suppression des arbres de recherche). Quels sont les cas faciles ? Les cas difficiles ? Donner un algorithme.

Correction

La suppression des arbre de recherche ôte toujours un seul nœud X qui est :

– soit un nœud ayant un seul fils F autre qu'un feuille N et dans ce cas F vient remplacer X ;

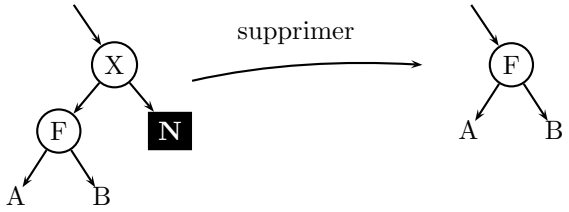

FIG. 8 – Cas où X a un fils F.

– soit un nœud dont les deux fils sont des feuilles N et dans ce cas la suppression remplace ce nœud X par une feuille N.

Premier cas facile : Si le nœud ôté X était rouge, alors il n'y a rien à faire, aucune des propriétés des arbres rouge noir n'est violée par le nouvel arbre.

Si le nœud ôté X était noir alors la hauteur noire de ses descendants change.

Deuxième cas facile : X avait un fils rouge, qui est nécessairement F,il suffit alors de colorier F en noir pour rétablir la hauteur.

Troisième cas facile : si X était la racine et n'avait pas de fils rouge alors la hauteur noir a diminué partout de 1, ainsi toutes les propriétés des arbres rouge noir sont vérifiées par le nouvel arbre.

Reste comme cas difficile : le nœud ôté X était noir sans être la racine et il n'avait pas de fils rouge. La situation est que la hauteur noire des feuilles en dessous de F est maintenant en déficit de 1 par rapport aux autre nœuds.

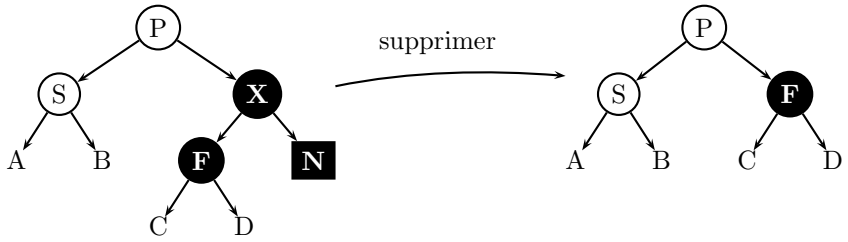

FIG. 9 – Cas où X noir a un fils F noir.

Il faut alors faire plusieurs cas en fonction de la couleur de P, de S et des fils de S. Certains de ces cas s'appellent récursivement les un les autres. C'est assez compliqué, on ne détaille pas.

Il faut juste retenir que le nombre d'appels récursif est borné asymptotiquement par la hauteur de X, et que le traitement de chacun de ces appels se fait en temps constant. Donc la procédure de maintien des propriétés des arbre rouge noir après une suppression est en $O(h)$, comme celle après une insertion. Ceci assure que l'insertion et la suppression bien que plus compliquée restent en $O(\log n)$.