Arbres rouge et noir [rn]
Algorithmique

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Version 21 mai 2018

Table des matières

Arbres rouge et noir

Mots-Clés Arbres rouge et noir ■
Requis Axiomatique impérative, Récursivité des actions, Complexité des algorithmes, Arbres binaires de recherche ■
Difficulté ● ● ○

Introduction
Les **arbres rouge et noir** sont **un** des schémas d'arbres binaires de recherche dits **équilibrés**. Ce module présente les définitions puis décrit les opérations de rotations, insertion et suppression, la recherche étant celles des arbres binaires de recherche.

1 Définitions, Représentation

Arbre rouge et noir

Un arbre binaire de recherche est un **arbre rouge et noir** s'il satisfait les propriétés suivantes :

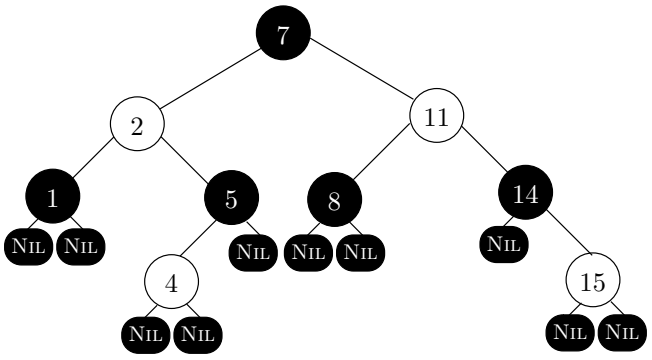
1. Chaque noeud est soit rouge, soit noir.
2. Chaque feuille (`Nil`) est noire.
3. Si un noeud est rouge alors ses deux fils sont noirs.
4. Tous les chemins descendants qui relie un noeud donné à une feuille (du sous-arbre dont il est la racine) contiennent le même nombre de noeuds noirs.

Hauteur noire

On appelle **hauteur noire** d'un noeud x, le nombre de noeuds noirs sur un chemin descendant de x à une feuille.

Exemple

La figure présente un exemple d'arbre rouge et noir. Dans toutes les figures, les noeuds « noir » sont à fond noir et les « rouges » sont à fond blanc.



Fonctionnalités

En plus de celles de l'arbre de recherche :

- Méthodes de la couleur
- Méthodes du grand-père

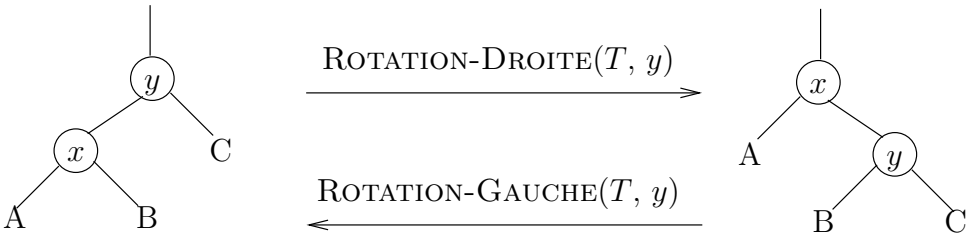
Représentation

Un noeud sera représenté par : `étiquette`, `filsgauche`, `filsdroit`, `père`, `couleur`.

2 Rotations

Les rotations

La figure présente les transformations d'arbres binaires appelées « rotations » pour d'évidentes raisons.



Attention

Les rotations préservent la propriété des arbres de recherche **mais pas** celle des arbres rouge et noir.

Algorithme RotationGauche

Réalise la rotation gauche pour un arbre binaire `T` en un noeud `x` (la rotation droite étant bien évidemment symétrique).

```
RotationGauche(T,x)
Début
  y <- droit(x)
  droit(x) <- gauche(y)
  Si gauche(y) <> Nil Alors
    père(gauche(y)) <- x
  FinSi
  père(y) <- père(x)
  Si père(y) = Nil Alors
    racine(T) <- y
  Sinon Si x = gauche(père(x)) Alors
    gauche(père(x)) <- y
  Sinon
    droit(père(x)) <- y
  FinSi
  gauche(y) <- x
  père(x) <- y
Fin
```

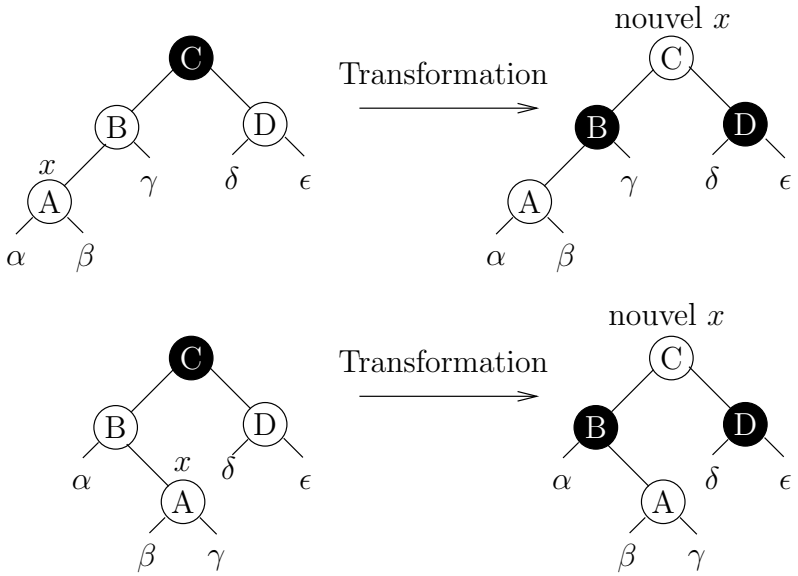
3 Insertion d'un élément

Principe de l'insertion dans un arbre rouge et noir

On peut essayer d'insérer le nouveau noeud comme si l'arbre n'était qu'un vulgaire arbre de recherche. Pour ce qui est du choix de la couleur du noeud inséré, le noir est *a priori* à proscrire puisqu'il provoquerait **systématiquement** une violation de la propriété 4. On choisit donc de colorier le nouveau noeud en rouge ce qui provoquera **parfois** des violations de la propriété 3 : un père et un fils étant tous les deux rouges. On étudie donc les différents cas de violations.

Cas pathologique 1

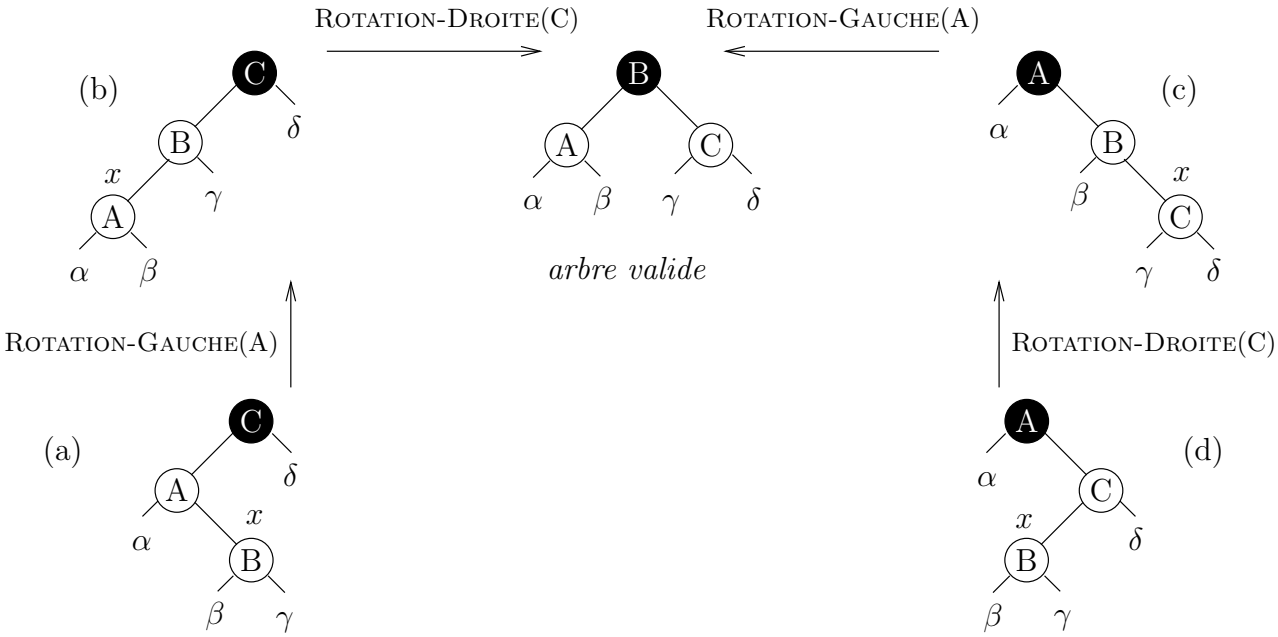
La figure ci-dessous présente une **première série** de configurations pathologiques pour l'insertion dans un arbre rouge et noir qui sont transformées en d'autres configurations. Les cas non traités sont symétriques de ceux présentés.



Ici, seule la propriété 3 est violée, x étant le noeud source du problème. Les sous-arbres α, β, γ, δ et ϵ sont tous de racine noire et ont tous la même hauteur noire. L'algorithme fait « descendre » la couleur noire du grand-père de x sur le père et l'oncle de x, ce qui revient à faire remonter le problème dans l'arbre, seule la propriété 3 pouvant être violée par cette transformation.

Cas pathologique 2

La figure suivante présente une **deuxième série** de configurations pathologiques, ces configurations là pouvant être résolues par l'application d'une ou de deux rotations selon les cas.



Ici aussi, seule la propriété 3 est violée : x est le noeud source du problème et nous ne sommes pas dans un des cas traités par la figure @[Cas pathologique 1]. Les sous-arbres α, β, γ et δ sont tous de racine noire et ont tous la même hauteur noire (et les transformations d'un cas à l'autre préservent cette propriété). Ici on conjugue des rotations et des changements de couleur des noeuds.

Algorithme RNInsertion

Insertion d'un élément en `x` dans un arbre rouge et noir `T` (suite à cette étude) :

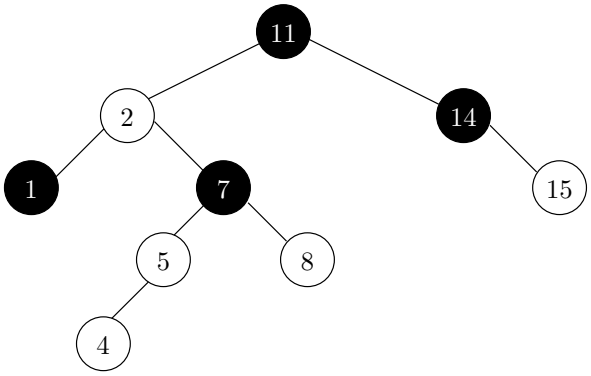
```
RNInsertion(T,x)
Début
 ABRInsertion(T,x)
 couleur(x) <- Rouge
 TantQue x <> racine(T) et couleur(père(x)) = Rouge Faire
   Si père(x) = gauche(grand-père(x)) Alors
     y <- droit(grand-père(x))
     Si couleur(y) = Rouge Alors
       // Pathos 1
       couleur(père(x)) <- Noir
       couleur(y) <- Noir
       couleur(grand-père(x)) <- Rouge
       x  <- grand-père(x)
     Sinon
       Si x = droit(père(x)) Alors
         // Pathos 2(a)
         x <- père(x)
         RotationGauche(T,x)
       FinSi
       // Pathos 2(b)
       couleur(père(x)) <- Noir
       couleur(grand-père(x)) <- Rouge
       RotationDroite(T, grand-père(x))
     FinSi
   Sinon
     (même chose que précédemment en échangeant droit et gauche)
```

```
   FinSi
 FinTantQue
 couleur(racine(T)) <- Noir
Fin
```

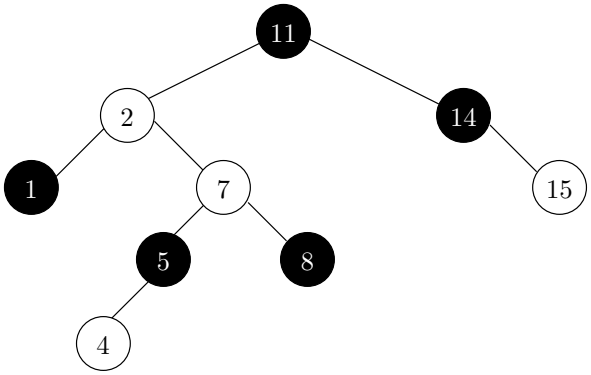
Exemple

La figure présente les différents cas pathologiques pouvant apparaître après l'insertion. À chaque fois le noeud x et son père sont tous les deux rouges.

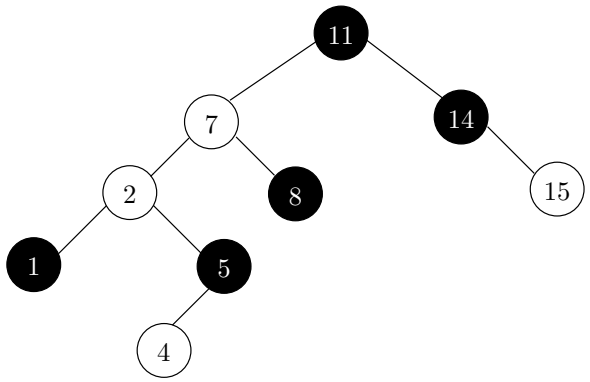
- (a) L'oncle de x (qui vaut 4) est également rouge : on se trouve donc dans le @[Cas pathologique 1].



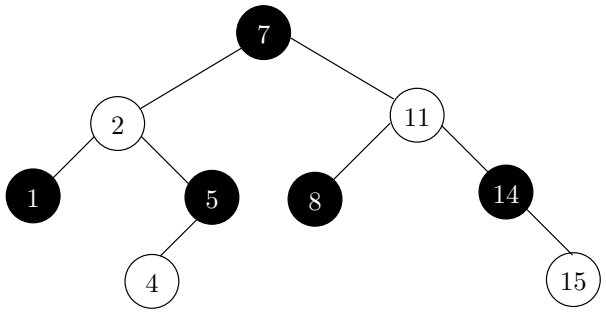
La couleur noire du grand-père descend sur le père et l'oncle de x, lequel est repeint en rouge, puis le problème est remonté de deux crans : ici x vaut 7.



- (b) L'oncle de x est noir et x est le fils droit de son père : on se trouve donc dans le @[Cas pathologique 2(a)]. On applique alors une rotation gauche sur le père de x et on aboutit à la situation du cas suivant : ici x vaut 2.



- (c) L'oncle de x est noir, x est le fils gauche de son père et son père est le fils gauche de son grand-père : on se trouve donc dans le @[Cas pathologique 2(b)]. On applique alors une rotation droite sur le grand-père de x. Le père de x est alors repeint en noir et l'ancien grand-père de x en rouge, ce qui nous donne un arbre rouge et noir valide.



4 Suppression d'un élément

Principe de la suppression dans un arbre rouge et noir

On commence d'abord par appliquer l'algorithme de suppression pour les arbres de recherche. Si l'élément supprimé était de couleur rouge, aucune des propriétés des arbres rouge et noir n'est violée. Par contre, si le noeud supprimé était noir la propriété 4 (tous les chemins descendants d'un noeud à une feuille contiennent le même nombre de noeuds noirs) peut être violée. Il nous faut donc **rajouter** un noeud noir sur tous les chemins perturbés.

Pour ce, on rajoute un noeud noir à l'unique fils du noeud supprimé (pour l'unicité du fils, voir l'algorithme de suppression @[ABRSuppression]). Si ce fils était rouge, l'arbre obtenu est un arbre rouge et noir. Si ce fils était déjà noir, on a deux « noirs » empilés sur un même noeud et il nous faut les répartir.

Cas pathologiques

La figure suivante présente les configurations pathologiques pour la suppression dans un arbre rouge et noir et les méthodes de résolutions associées.

Si le noeud supprimé n'avait pas de fils, on rajoute un « noir » à la feuille `Nil` correspondante de son père. Pour pouvoir réaliser cette manipulation, on utilise une sentinelle : un noeud spécial valant `Nil` et qui permet de ne pas traiter à part les feuilles `Nil`.

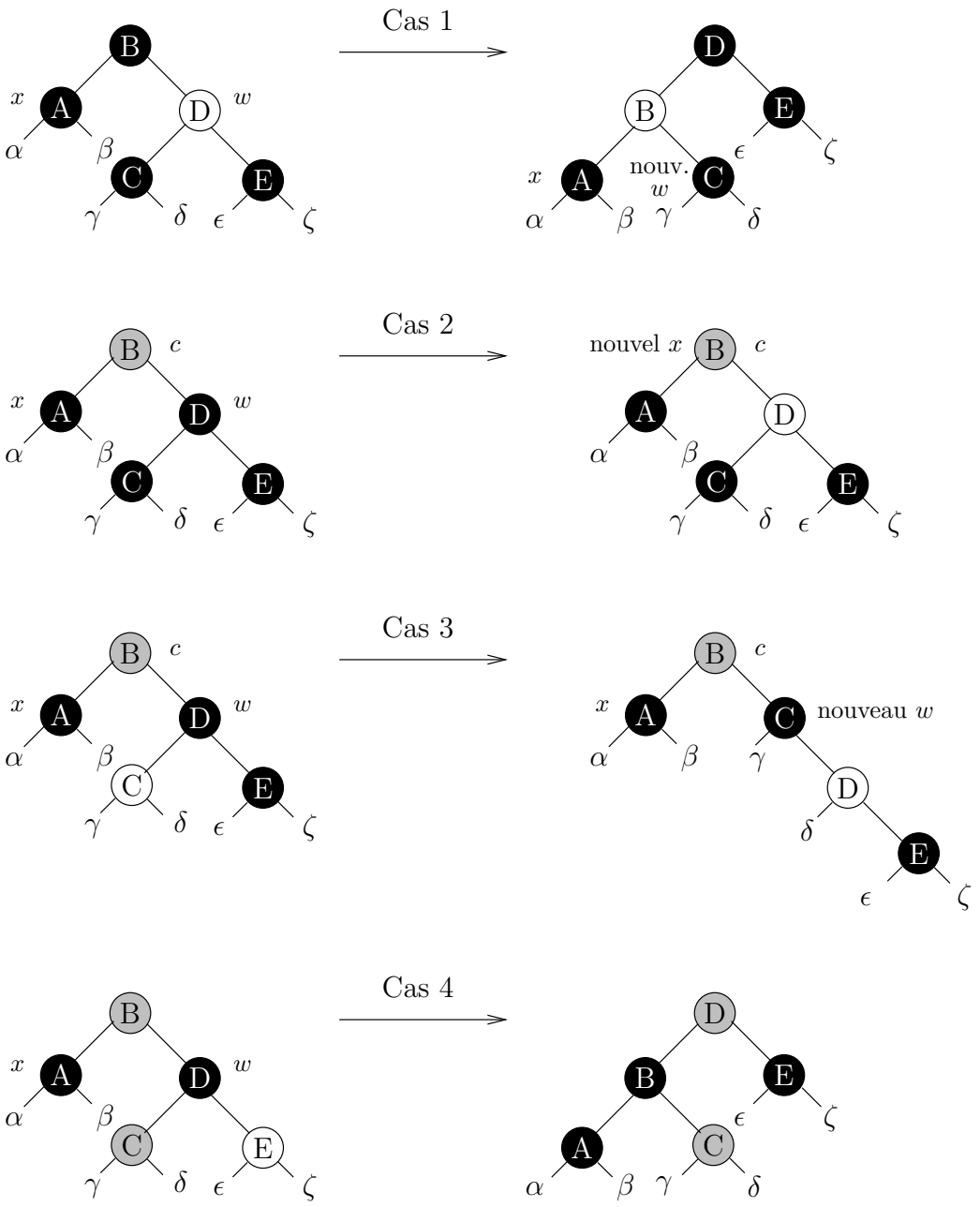
Les noeuds à fond noir sont des noeuds « noirs », ceux à fond blanc sont « rouges » et ceux à fond grisé sont soit « noirs » soit « rouges ». Les arbres α, β, γ, δ, ϵ et ζ sont quelconques. Le noeud x comporte un noir supplémentaire.

Cas 1 Ce cas est transformé en cas 2, 3 ou 4. (Rotation gauche en D avec recoloration D et B.)

Cas 2 Le noir supplémentaire est remonté sur le père de x, l'oncle de x étant repeint en rouge ; si le père de x était rouge l'arbre est de nouveau valide, sinon on rapplique l'algorithme de correction cette fois-ci sur le père de x.

Cas 3 Ce cas est transformé en cas 4.

Cas 4 Le noir supplémentaire est éliminé par rotation gauche sur le père de x et recoloriage du père et de l'oncle de x.



Algorithme RNSuppression

Suppression d'un élément en `x` dans un arbre rouge et noir `T` avec utilisation de sentinelles et appel de l'algorithme de correction :

```
RNSuppression(T,x)
Début
  Si gauche(x) = Nil(T) et droit(x) = Nil(T) Alors
    Si père(x) = Nil(T) Alors
      racine(T) <- Nil(T)
    Sinon
      Si x = gauche(père(x)) Alors
        gauche(père(x)) <- Nil(T)
      Sinon
        droit(père(x)) <- Nil(T)
      FinSi
      Si couleur(x) = Noir Alors
```

```
        père(Nil(T)) <- père(x)
        RNCorrection(T,x)
      FinSi
    FinSi
  Sinon Si gauche(x) = Nil(T) ou droit(x) = Nil(T) Alors
    Si gauche(x) <> Nil(T) Alors
      filsde_x  <- gauche(x)
    Sinon
      filsde_x  <- droit(x)
    FinSi
    père(filsde_x) <- père(x)
    Si père(x) = Nil(T) Alors
      racine(T) <- filsde_x
    Sinon si gauche(père(x)) = x Alors
      gauche(père(x)) <- filsde_x
    Sinon
      droit(père(x)) <- filsde_x
    FinSi
    Si couleur(x) = Noir Alors
      RNCorrection(T,filsde_x)
    Sinon
      min <- ABRMinimum(droit(x))
      clé(y)  <- clé(min)
      RNSuppression(T,min)
    FinSi
  FinSi
  Retourner racine(T)
Fin
```

Algorithme RNCorrection

Celui qui répartit les « noirs » surnuméraires, d'un arbre rouge et noir `T` après suppression d'un élément en `x` :

```
RNCorrectiontion(T,x)
Début
  TantQue x <> racine(T) et couleur(x) = Noir Faire
    Si x = gauche(père(x)) Alors
      w <- droit(père(x))
      Si couleur(w) = Rouge Alors
        // cas 1
        couleur(w) <- Noir
        couleur(père(w)) <- Rouge
        RotationGauche(T,père(x))
        w <- droit(père(x))
      FinSi
      Si couleur(gauche(w)) =  Noir et couleur(droit(w)) = Noir Alors
        // Cas 2
        couleur(w) <- Rouge
        x <- père(x)
      Sinon Si couleur(droit(w)) = Noir Alors
        // Cas 3
        couleur(gauche(w)) <- Noir
        couleur(w) <- Rouge
        RotationDroite(T,w)
        w <- droit(père(x))
      FinSi
      // Cas 4
```

```
      couleur(w) <- couleur(père(x))
      couleur(père(x)) <- Noir
      couleur(droit(w)) <- Noir
      RotationGauche(T,père(x))
      x <- racine(T)
    Sinon
      (même chose que précédemment en échangeant droit et gauche)
    FinSi
  FinTantQue
  couleur(x) <- Noir
Fin
```

5 Complexité

Théorème
Un arbre rouge et noir contenant n noeuds internes a une hauteur au plus égale à $2\lg(n+1)$.

Démonstration
Par induction, on peut montrer que le sous-arbre (d'un arbre rouge et noir) enraciné en un noeud x quelconque contient au moins $2^{hn(x)}$ noeuds internes, où $hn(x)$ est la hauteur noire de x. Sachant que la hauteur est toujours inférieure au double de la hauteur noire on en déduit la borne du théorème. ■

Conséquence
Ce théorème montre que les arbres rouge et noir sont relativement **équilibrés** : la hauteur d'un arbre rouge et noir est au pire du double de celle d'un arbre binaire parfaitement équilibré.

Conclusion
Toutes les opérations sur les arbres rouge et noir sont de coût $O(h)$, c.-à-d. $O(\lg n)$. Ceci justifie leur utilisation par rapport aux arbres binaires de recherche classiques.