Arbres binaires de recherche [br]
Algorithmique

Karine Zampieri, Stéphane Rivière

Table des matières

1 Définition, Parcours, Représentation 2

2 Recherches 4
2.1 Recherche d'un élément 4
2.2 Minimum et maximum 5
2.3 Successeur et prédécesseur 5

3 Insertion d'un élément 8

4 Suppression d'un élément 9

5 Conclusion 12

Arbres binaires de recherche

Mots-Clés Arbres binaires de recherche ■
Requis Axiomatique impérative, Récursivité des actions, Complexité des algorithmes, Arbres enracinés ■
Difficulté • • ○

Introduction
Un **arbre binaire de recherche** est une structure de donnée qui permet de représenter un ensemble de valeurs si l'on dispose d'une relation d'ordre sur ces valeurs. Les opérations caractéristiques sont l'insertion, la suppression et la recherche d'une valeur. Ces opérations sont peu coûteuses si l'arbre n'est pas trop déséquilibré. En pratique, les valeurs sont des clés permettant d'accéder à des enregistrements.

Ce module les décrit puis définit les opérations caractéristiques (recherche, insertion, suppression).

1 Définition, Parcours, Représentation

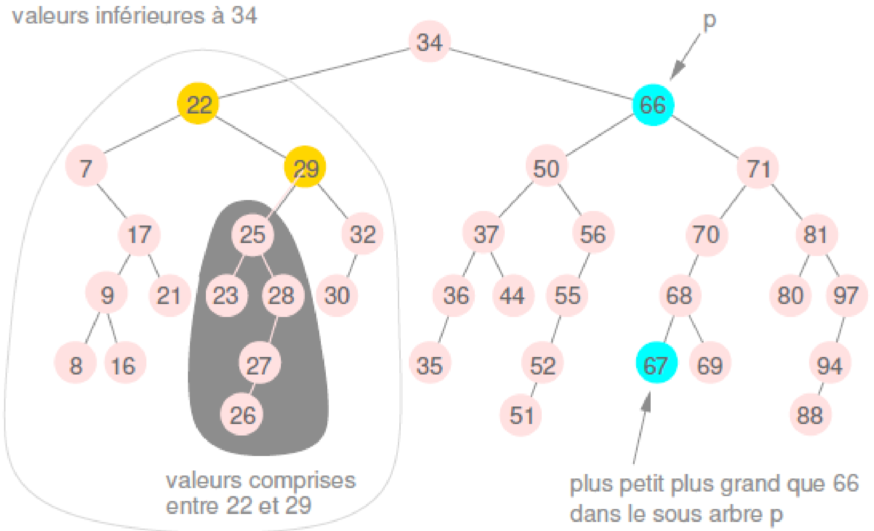
Définition d'un ABR

Un **arbre binaire de recherche** (abrégé ABR) est un arbre binaire vérifiant la propriété suivante : soient x et y deux noeuds de l'arbre :

- Si y est un noeud du sous-arbre gauche de x alors $cle(y) \leq cle(x)$
- Si y est un noeud du sous-arbre droit de x alors $cle(y) \geq cle(x)$

Exemple

L'arbre suivant est un ABR : pour tout noeud p de A, la valeur de p est strictement plus grande que les valeurs figurant dans son sous-arbre gauche et strictement plus petite que les valeurs figurant dans son sous-arbre droit.

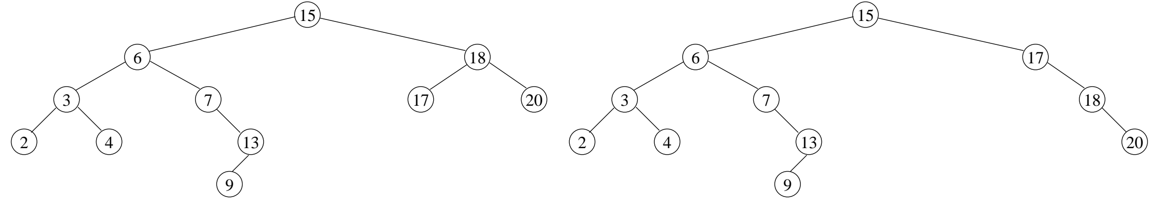


Remarque

La définition suppose donc qu'une valeur n'apparaît au plus qu'une seule fois dans un arbre de recherche.

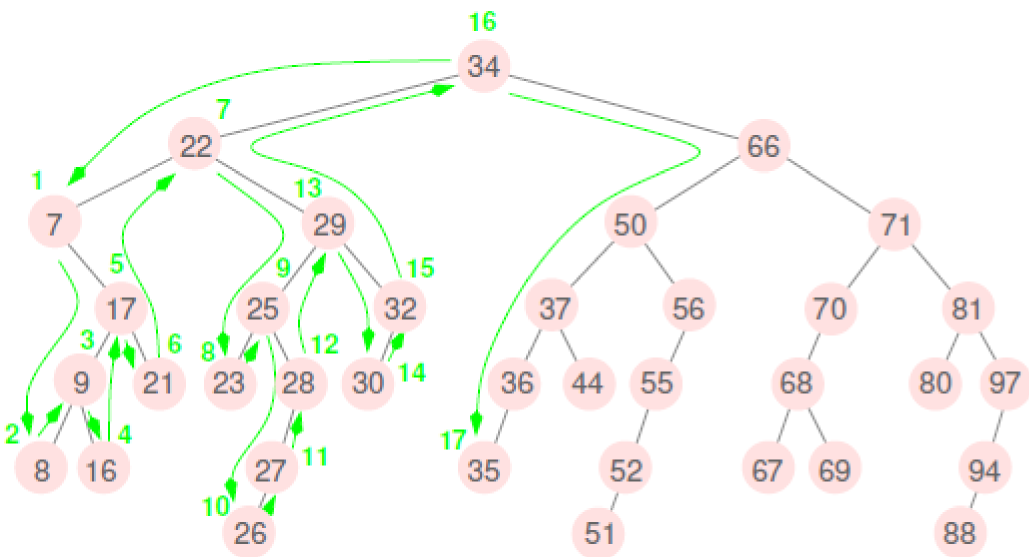
Attention

Bien que différents, ces deux arbres ABR **contiennent** exactement **les mêmes valeurs**.



Parcours d'un ABR

Le parcours infixé fournit la suite ordonnée des clés :



Représentation d'un ABR

Un noeud sera représentée par les champs : `étiquette`, `filsdroit`, `filsgauche`, `père`.

Champ père

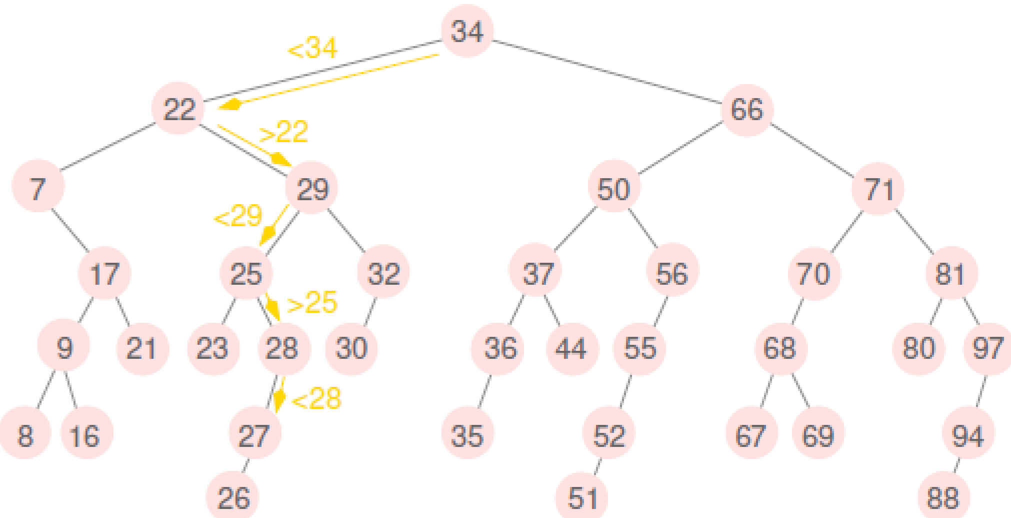
Il permet de traverser **itérativement** un ABR. Il permet également la recherche du **successeur** et **prédécesseur** d'un noeud.

2 Recherches

2.1 Recherche d'un élément

Principe

La recherche d'une valeur dans un ABR consiste, en partant de la racine, à parcourir une branche en descendant chaque fois sur le fils gauche ou sur le fils droit selon que la clé portée par le noeud est plus grande ou plus petite que la valeur cherchée. La recherche s'arrête dès que la valeur est rencontrée ou que l'on a atteint l'extrémité d'une branche (le fils sur lequel il aurait fallu descendre n'existe pas).



Algorithme de recherche

Recherche d'une valeur `k` à partir d'un noeud `x` d'un ABR :

```
ABRRecherche(x,k)
Début
  Si x = Nil Alors
    Retourner Faux
  Sinon Si k = cle(x) Alors
    Retourner Vrai
  Sinon Si k < cle(x) Alors
    Retourner ABRRecherche(gauche(x),k)
  Sinon
    Retourner ABRRecherche(droit(x),k)
  FinSi
Fin
```

2.2 Minimum et maximum

Principe
Pour accéder à la clé la plus petite (resp. la plus grande) dans un ABR, il suffit de descendre sur le fils gauche (resp. le fils droit) autant que possible. Le dernier noeud visité qui n'a pas de fils gauche (resp. fils droit), porte la valeur la plus petite (resp. la plus grande) de l'arbre.

Algorithme Minimum
Recherche du minimum à partir d'un noeud `x` d'un ABR :

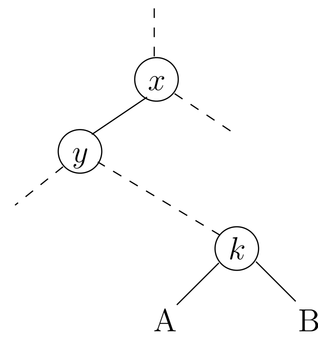
```
ABRMinimum(x)
Début
  TantQue gauche(x) <> Nil Faire
    x <- gauche(x)
  FinTantQue
  Retourner (x)
Fin
```

Algorithme Maximum
Par symétrie :

```
ABRMaximum(x)
Début
  TantQue droit(x) <> Nil Faire
    x <- droit(x)
  FinTantQue
  Retourner (x)
Fin
```

2.3 Successeur et prédécesseur

Principe
Si toutes les clés dans l'arbre ABR sont distinctes, le successeur d'un noeud k est le noeud contenant la plus petite clé supérieure à k. Considérons le fragment d'arbre présenté sur la figure :

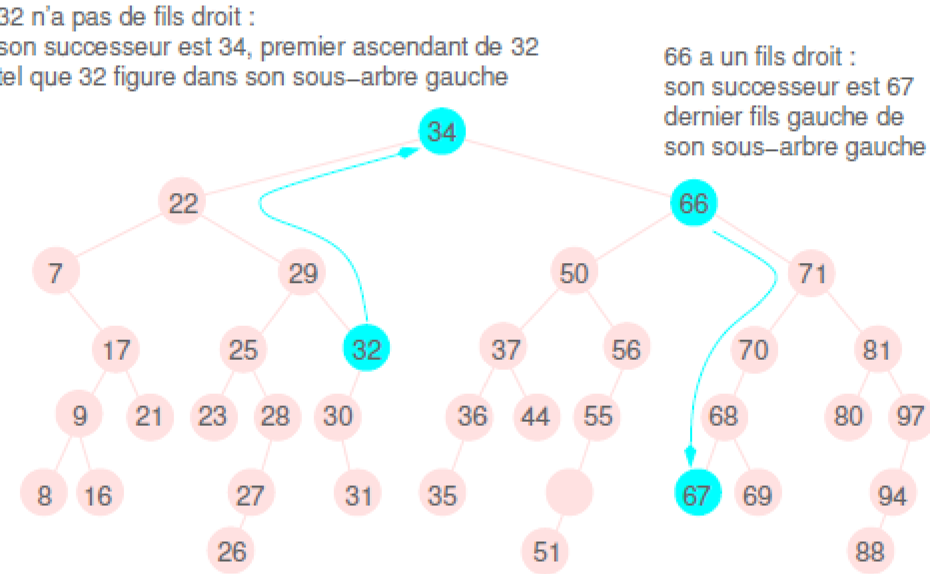


De par la propriété des ABR :

1. Le sous-arbre A ne contient que des clés inférieures à k : il ne peut pas contenir le successeur de k.
2. Le sous-arbre B ne contient que des clés supérieures à k : il peut contenir le successeur de k que s'il n'est pas vide.
3. y désigne le plus proche ancêtre de k qui soit le fils gauche de son père ($y = k$ si k est fils gauche de son père). Tous les ancêtres de k jusqu'à y sont inférieurs ou égaux à k et leurs sous-arbres gauches ne contiennent que des valeurs inférieures à k.
4. x est le père de y. Sa valeur est supérieure à toutes celles contenues dans son sous-arbre gauche (de racine y) et donc à k et à celles de B. Toutes les valeurs de son sous-arbre droit sont supérieures à x.

En résumé : si B est non vide, son minimum est le successeur de k, sinon le successeur de k est le premier ancêtre (ascendant) de k dont le fils gauche est aussi ancêtre de k. Si cet ascendant n'existe pas c'est que k portait la valeur la plus grande dans l'arbre.

Exemple



Remarquez qu'il est nécessaire d'avoir pour chaque noeud un lien vers son père pour mener à bien cette opération.

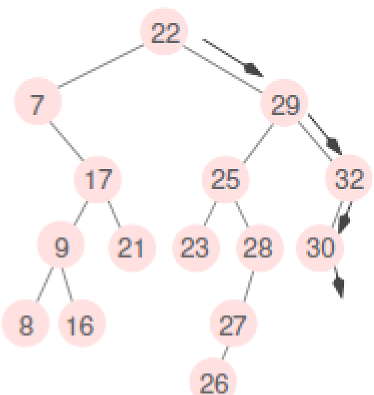
Algorithme Successeur

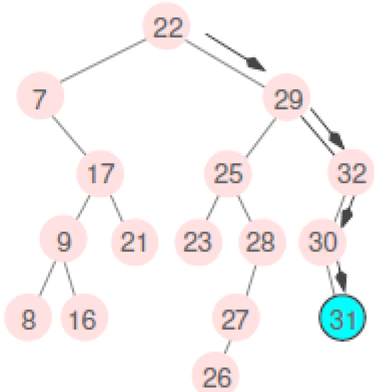
```
ABRSuccesseur(x)
Début
  Si droit(x) <> Nil Alors
    Retourner ABRMinimum(droit(x))
  Sinon
    y <- père(x)
    TantQue y <> Nil et x = droit(y) Faire
      x <- y
      y <- père(x)
    FinTantQue
    Retourner (y)
  FinSi
Fin
```

3 Insertion d'un élément

Principe

L'élément à ajouter est inséré là où on l'aurait trouvé s'il avait été présent dans l'arbre. L'algorithme d'insertion recherche donc l'élément dans l'arbre et, quand il aboutit à la conclusion que l'élément n'appartient pas à l'arbre (l'algorithme aboutit sur `Nil`), il insère l'élément comme fils du dernier noeud visité.





Algorithme d'insertion

```
ABRInsertion(T,z)
Début
  x <- racine(T)
  père_de_x <- Nil
  TantQue x <> Nil Faire
    père_de_x <- x
    Si cle(z) < cle(x) Alors
      x <- gauche(x)
    Sinon
      x <- droit(x)
    FinSi
  FinTantQue
  père(z) <- père_de_x
  Si père_de_x = Nil Alors
    racine(T) <- z
  Sinon si cle(z) < cle(x) Alors
    gauche(père_de_x) <- z
  Sinon
    droit(père_de_x) <- z
  FinSi
fin
```

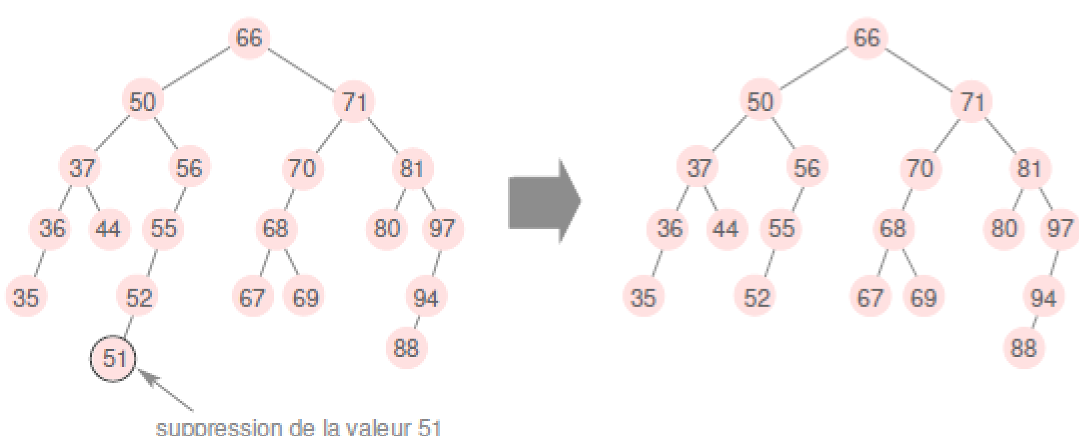
Écrivez une version récursive de l'opération.

4 Suppression d'un élément

L'opération de suppression dépend du nombre de fils du noeud à supprimer dans l'ABR. Les différents cas de figure possibles sont les suivants (les noeuds à supprimer sont en gris foncé) :

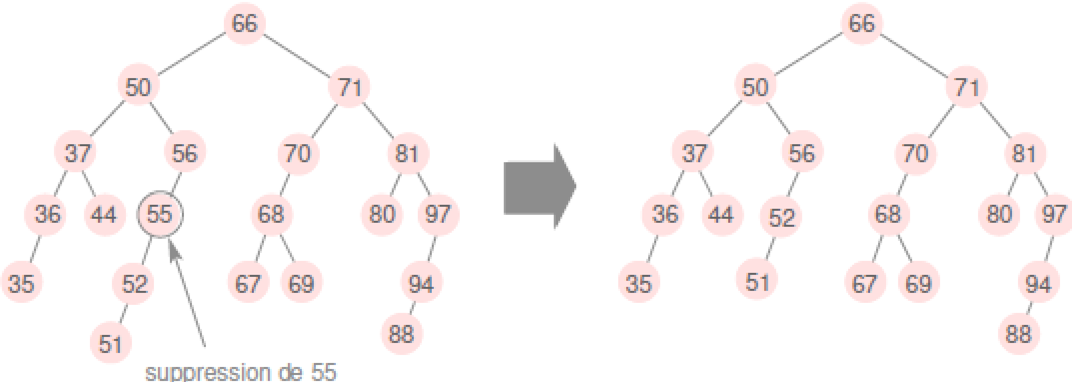
Cas 1 : Cas d'une feuille
Le noeud à supprimer n'a pas de fils : on l'élimine simplement de l'arbre en modifiant le lien de son père. Si le père n'existe pas, l'arbre devient l'arbre vide.

Exemple



Cas 2 : Cas d'un unique fils
Le noeud à supprimer a un unique fils : on détache le noeud et on relie directement son père et son fils. Si ce père n'existe pas, l'arbre est réduit au fils unique du noeud supprimé.

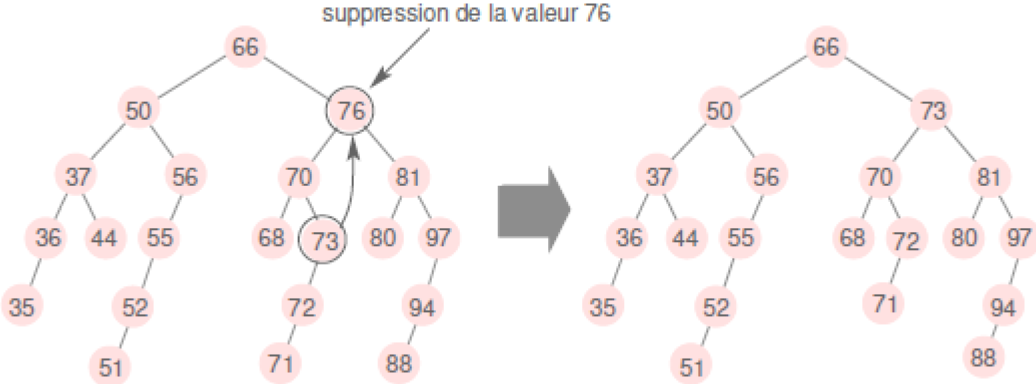
Exemple



Cas 3 : Cas de deux fils

Le noeud à supprimer a deux fils : on le remplace par son prédécesseur q qui, dans ce cas, est toujours le maximum de son sous-arbre gauche (on peut prendre indifféremment son successeur qui est alors le minimum de son sous-arbre droit). Puisque le noeud q a la valeur la plus grande dans le fils gauche, il n'a donc pas de fils droit, et peut être décroché comme on l'a fait dans les cas 1 et 2.

Exemple



Algorithme de suppression

Il tient compte des conditions aux limites (changement de racine).

```
ABRSuppression(T,x)
Début
 Si gauche(x) = Nil et droit(x) = Nil Alors
   Si père(x) = Nil Alors
     racine(T) <- Nil
   Sinon
     Si x = gauche(père(x)) Alors
       gauche(père(x)) <- Nil
```

```
    Sinon
      droit(père(x))  <- Nil
    FinSi
  FinSi
 Sinon Si gauche(x) = Nil Ou droit(x) = Nil Alors
  Si gauche(x)  <> Nil Alors
    filsde_x <- gauche(x)
  Sinon
    filsde_x <- droit(x)
  FinSi
  père(filsde_x) <- père(x)
  Si père(x) = Nil Alors
    racine(T) <- filsde_x
  Sinon
    Si gauche(père(x)) = x Alors
      gauche(père(x)) <- filsde_x
    Sinon
      droit(père(x)) <- filsde_x
    FinSi
  FinSi
 Sinon
  xmin <- ABRMinimum(droit(x))
  cle(y) <- cle(xmin)
  ABRSuppression(T,xmin)
 FinSi
 Retourner T
Fin
```

5 Conclusion

Si h est la hauteur de l'arbre, on peut aisément montrer que tous les algorithmes précédents ont une complexité en $O(h)$. Malheureusement, un arbre binaire quelconque de n noeuds a une hauteur comprise, en ordre de grandeur, entre $\lg n$ et n. Pour éviter les cas les plus pathologiques, on s'intéresse à des arbres de recherches **équilibrés**.