Algorithme 2

Arbres binaires de recherche

Louis-Claude Canon
louis-claude.canon@univ-fcomte.fr

Licence 2 Informatique – Semestre 4

Plan

Plan

Contexte d'utilisation des arbres binaires de recherche

- Structure de données qui permet les opérations de recherche, de minimum/maximum, de prédécesseur/successeur, d'insertion et de suppression.
- Peut servir de dictionnaire (comme une table de hachage) et de file de priorités.
- Temps des opérations basiques proportionnels à la hauteur de l'arbre.
  - Pour des arbres binaires complets avec n nœuds : $\Theta(\lg n)$.
  - Pour des chaînes linéaires de n nœuds : $\Theta(n)$.
- Date de la fin des années 1950.

Contexte d'utilisation des arbres binaires de recherche

- Structure de données qui permet les opérations de recherche, de minimum/maximum, de prédécesseur/successeur, d'insertion et de suppression.
- Peut servir de dictionnaire (comme une table de hachage) et de file de priorités.
- Temps des opérations basiques proportionnels à la hauteur de l'arbre.
  - Pour des arbres binaires complets avec n nœuds : $\Theta(\log n)$.
  - Pour des chaînes linéaires de n nœuds : $\Theta(n)$.
- Date de la fin des années 1950.

Contexte d'utilisation des arbres binaires de recherche

- Structure de données qui permet les opérations de recherche, de minimum/maximum, de prédécesseur/successeur, d'insertion et de suppression.
- Peut servir de dictionnaire (comme une table de hachage) et de file de priorités.
- Temps des opérations basiques proportionnels à la hauteur de l'arbre.
  - Pour des arbres binaires complets avec n nœuds : $\Theta(\log n)$.
  - Pour des chaînes linéaires de n nœuds : $\Theta(n)$.
- Date de la fin des années 1950.

Contexte d'utilisation des arbres binaires de recherche

- Structure de données qui permet les opérations de recherche, de minimum/maximum, de prédécesseur/successeur, d'insertion et de suppression.
- Peut servir de dictionnaire (comme une table de hachage) et de file de priorités.
- Temps des opérations basiques proportionnels à la hauteur de l'arbre.
  - Pour des arbres binaires complets avec n nœuds : $\Theta(\log n)$.
  - Pour des chaînes linéaires de n nœuds : $\Theta(n)$.
- Date de la fin des années 1950.

Contexte d'utilisation des arbres binaires de recherche

- Structure de données qui permet les opérations de recherche, de minimum/maximum, de prédécesseur/successeur, d'insertion et de suppression.
- Peut servir de dictionnaire (comme une table de hachage) et de file de priorités.
- Temps des opérations basiques proportionnels à la hauteur de l'arbre.
  - Pour des arbres binaires complets avec n nœuds : $\Theta(\log n)$.
  - Pour des chaînes linéaires de n nœuds : $\Theta(n)$.
- Date de la fin des années 1950.

Contexte d'utilisation des arbres binaires de recherche

- Structure de données qui permet les opérations de recherche, de minimum/maximum, de prédécesseur/successeur, d'insertion et de suppression.
- Peut servir de dictionnaire (comme une table de hachage) et de file de priorités.
- Temps des opérations basiques proportionnels à la hauteur de l'arbre.
  - Pour des arbres binaires complets avec n nœuds : $\Theta(\log n)$.
  - Pour des chaînes linéaires de n nœuds : $\Theta(n)$.
- Date de la fin des années 1950.

Structure d'arbre binaire

- Chaque nœud possède 4 champs :
  - *clé* valeur et données satellites ;
  - *gauche* enfant de gauche ;
  - *droite* enfant de droite ;
  - *parent* parent du nœud.
- Rappel :
- On représente souvent un arbre avec des pointeurs (ou équivalent).

Structure d'arbre binaire

- Chaque nœud possède 4 champs :
  - *clé* valeur et données satellites ;
  - *gauche* enfant de gauche ;
  - *droite* enfant de droite ;
  - *parent* parent du nœud.
- Rappel :
- On représente souvent un arbre avec des pointeurs (ou équivalent).

Structure d'arbre binaire

- Chaque nœud possède 4 champs :
  - *clé* valeur et données satellites ;
  - *gauche* enfant de gauche ;
  - *droite* enfant de droite ;
  - *parent* parent du nœud.
- Rappel :
- On représente souvent un arbre avec des pointeurs (ou équivalent).

Structure d'arbre binaire

- Chaque nœud possède 4 champs :
  - *clé* valeur et données satellites ;
  - *gauche* enfant de gauche ;
  - *droite* enfant de droite ;
  - *parent* parent du nœud.
- Rappel :
  - la racine d'un arbre est le nœud sans parent ;
  - une feuille est un nœud sans enfant (gauche = droite = NIL) ;
  - la *taille* d'un arbre est son nombre de nœuds ;
  - la *hauteur* d'un arbre est le nombre maximal d'arêtes séparant la racine d'une feuille.
- On représente souvent un arbre avec des pointeurs (ou équivalent).

Structure d'arbre binaire

- Chaque nœud possède 4 champs :
  - *clé* valeur et données satellites ;
  - *gauche* enfant de gauche ;
  - *droite* enfant de droite ;
  - *parent* parent du nœud.
- Rappel :
  - la *racine* d'un arbre est le nœud sans parent ;
  - une *feuille* est un nœud sans enfant ($gauche = droite = NIL$) ;
  - la *taille* d'un arbre est son nombre de nœuds ;
  - la *hauteur* d'un arbre est le nombre maximal d'arcs séparant la racine d'une feuille.
- On représente souvent un arbre avec des pointeurs (ou équivalent).

Structure d'arbre binaire

- Chaque nœud possède 4 champs :
  - *clé* valeur et données satellites ;
  - *gauche* enfant de gauche ;
  - *droite* enfant de droite ;
  - *parent* parent du nœud.
- Rappel :
  - la *racine* d'un arbre est le nœud sans parent ;
  - une *feuille* est un nœud sans enfant ($gauche = droite = NIL$) ;
  - la *taille* d'un arbre est son nombre de nœuds.
  - la *hauteur* d'un arbre est le nombre maximal d'arcs séparant la racine d'une feuille.
- On représente souvent un arbre avec des pointeurs (ou équivalent).

Structure d'arbre binaire

- Chaque nœud possède 4 champs :
  - *clé* valeur et données satellites ;
  - *gauche* enfant de gauche ;
  - *droite* enfant de droite ;
  - *parent* parent du nœud.
- Rappel :
  - la *racine* d'un arbre est le nœud sans parent ;
  - une *feuille* est un nœud sans enfant ($gauche = droite = NIL$) ;
  - la *taille* d'un arbre est son nombre de nœuds.
  - la *hauteur* d'un arbre est le nombre maximal d'arcs séparant la racine d'une feuille.
- On représente souvent un arbre avec des pointeurs (ou équivalent).

Structure d'arbre binaire

- Chaque nœud possède 4 champs :
  - *clé* valeur et données satellites ;
  - *gauche* enfant de gauche ;
  - *droite* enfant de droite ;
  - *parent* parent du nœud.
- Rappel :
  - la *racine* d'un arbre est le nœud sans parent ;
  - une *feuille* est un nœud sans enfant ($gauche = droite = NIL$) ;
  - la *taille* d'un arbre est son nombre de nœuds.
  - la *hauteur* d'un arbre est le nombre maximal d'arcs séparant la racine d'une feuille.
- On représente souvent un arbre avec des pointeurs (ou équivalent).

Structure d'arbre binaire

- Chaque nœud possède 4 champs :
  - *clé* valeur et données satellites ;
  - *gauche* enfant de gauche ;
  - *droite* enfant de droite ;
  - *parent* parent du nœud.
- Rappel :
  - la *racine* d'un arbre est le nœud sans parent ;
  - une *feuille* est un nœud sans enfant ($gauche = droite = NIL$) ;
  - la *taille* d'un arbre est son nombre de nœuds.
  - la *hauteur* d'un arbre est le nombre maximal d'arcs séparant la racine d'une feuille.
- On représente souvent un arbre avec des pointeurs (ou équivalent).

Structure d'arbre binaire

- Chaque nœud possède 4 champs :
  - *clé* valeur et données satellites ;
  - *gauche* enfant de gauche ;
  - *droite* enfant de droite ;
  - *parent* parent du nœud.
- Rappel :
  - la *racine* d'un arbre est le nœud sans parent ;
  - une *feuille* est un nœud sans enfant ($gauche = droite = NIL$) ;
  - la *taille* d'un arbre est son nombre de nœuds.
  - la *hauteur* d'un arbre est le nombre maximal d'arcs séparant la racine d'une feuille.
- On représente souvent un arbre avec des pointeurs (ou équivalent).

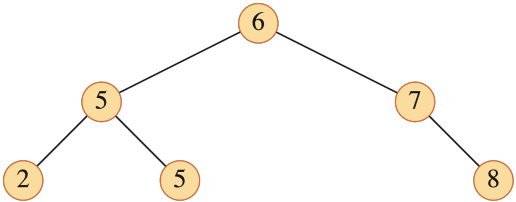
Structure d'arbre binaire

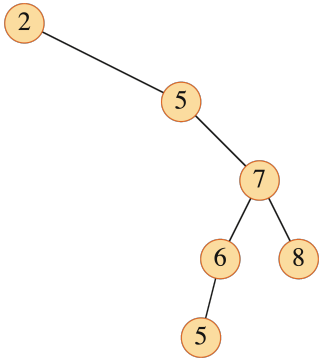
- Chaque nœud possède 4 champs :

clé	valeur et données satellites ;
gauche	enfant de gauche ;
droite	enfant de droite ;
parent	parent du nœud.

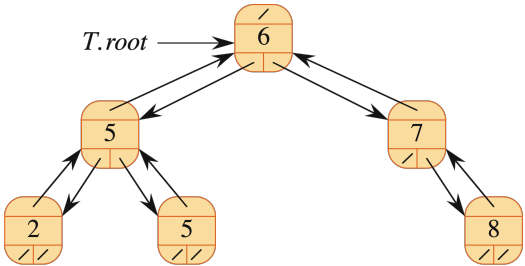
- Rappel :
  - la *racine* d'un arbre est le nœud sans parent ;
  - une *feuille* est un nœud sans enfant ($gauche = droite = NIL$) ;
  - la *taille* d'un arbre est son nombre de nœuds.
  - la *hauteur* d'un arbre est le nombre maximal d'arcs séparant la racine d'une feuille.
- On représente souvent un arbre avec des pointeurs (ou équivalent).

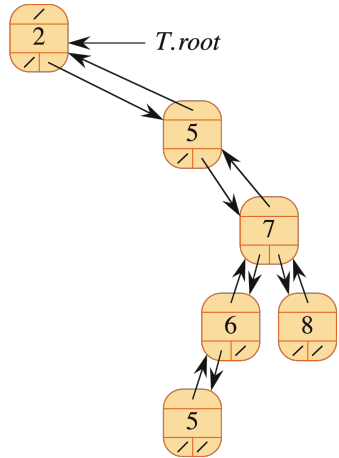
Exemples d'arbres binaires





Exemples d'arbres binaires

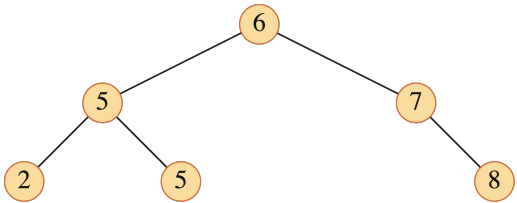


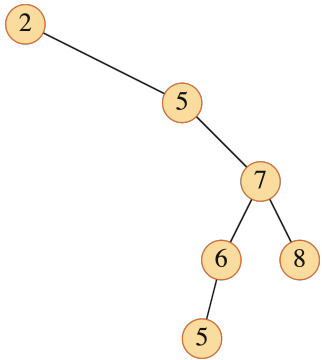


Propriété des arbres de recherche

- Soit x un nœud d'un arbre binaire de recherche.
- Si y est un nœud du sous-arbre de gauche de x, alors $y.clé \leq x.clé$.
- Si y est un nœud du sous-arbre de droite de x, alors $x.clé \leq y.clé$.

Exemples d'arbres binaires





Parcours infixe

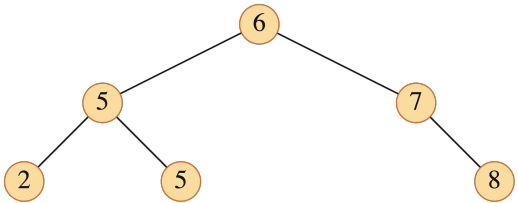
Algorithme qui parcourt l'ensemble des nœuds d'un arbre binaire :

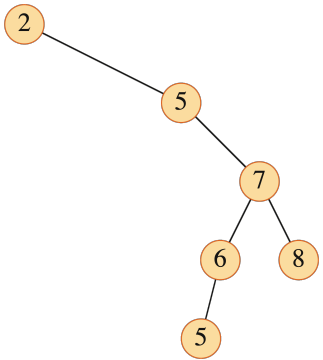
PARCOURS-INFIXE(x)

si $x \neq NIL$ **alors**
  PARCOURS-INFIXE($x.gauche$)
  afficher $x.clé$
  PARCOURS-INFIXE($x.droite$)

Le parcours infixe affiche les clés d'un arbre binaire de recherche par ordre croissant de valeur.

Exemples d'arbres binaires





Analyse de complexité

- Soit un nœud x dont le sous-arbre de gauche a k nœuds (celui de droite en a $n-k-1$).
- Soit $T(n)$ le temps nécessaire pour traiter un arbre de taille n.
- $T(0) = c$: temps du test $x \neq NIL$.
- $T(n) = T(k) + T(n-k-1) + c + d$ où d est le temps de l'affichage de $x.clé$.

Notion mathématique

Méthode de substitution pour borner une récurrence :

- on conjecture une borne ;
- on la vérifie pour $n = 0$ (ou autre condition initiale) ;
- en la supposant vrai pour tout $0 \leq n' < n$, on montre qu'elle tient aussi pour n.

Méthode de substitution

- On conjecture que $T(n) = (2c + d)n + c$.
- C'est valide pour $n = 0$: $T(0) = c$.
- On substitue dans $T(n) = T(k) + T(n - k - 1) + d$:

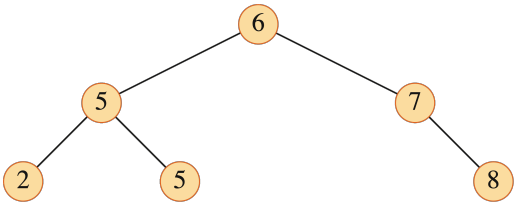
$$\begin{aligned} T(n) &= ((2c + d)k + c) + ((2c + d)(n - k - 1) + c) + c + d && (1)\\ &= c + (2c + d)(n - 1) + c + c + d && (2)\\ &= (2c + d)n + c && (3) \end{aligned}$$

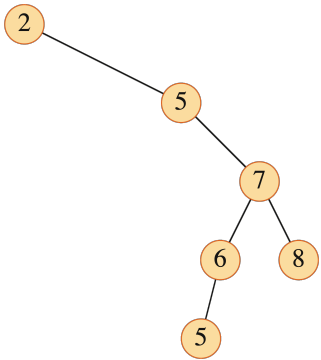
- Si x est la racine d'un sous-arbre à n nœuds, alors l'appel PARCOURS-INFIXE(x) prend un temps $T(n) = \Theta(n)$.

Autres parcours

- Parcours *préfixe* : l'affichage de la clé s'effectue avant chaque sous-arbre.
- Parcours *postfixe* : l'affichage de la clé s'effectue après chaque sous-arbre. Ces trois premiers parcours sont des parcours en profondeur.
- Parcours *en largeur* (BFS) : on considère les nœuds niveau par niveau. Le tableau représentant un tas correspond à son parcours en largeur.

Exemples d'arbres binaires





Question

Lesquels de ces ordres ne sont ni infixe, ni préfixe, ni postfixe pour le tas $(16, 14, 10, 8, 7, 9, 3, 2, 4, 1)$?

1. 2, 8, 4, 14, 1, 7, 16, 9, 10, 3
2. 16, 14, 10, 8, 7, 9, 3, 2, 4, 1
3. 16, 14, 8, 2, 4, 7, 1, 10, 9, 3
4. 2, 4, 8, 1, 7, 14, 9, 3, 10, 16
5. 1, 2, 3, 4, 7, 8, 9, 10, 14, 16

Question

Lesquels de ces ordres ne sont ni infixe, ni préfixe, ni postfixe pour le tas $(16, 14, 10, 8, 7, 9, 3, 2, 4, 1)$?

1. 2, 8, 4, 14, 1, 7, 16, 9, 10, 3 infixe
2. 16, 14, 10, 8, 7, 9, 3, 2, 4, 1 ✓(BFS)
3. 16, 14, 8, 2, 4, 7, 1, 10, 9, 3 préfixe
4. 2, 4, 8, 1, 7, 14, 9, 3, 10, 16 postfixe
5. 1, 2, 3, 4, 7, 8, 9, 10, 14, 16 ✓(tri)

Plan

Arbre-Rechercher

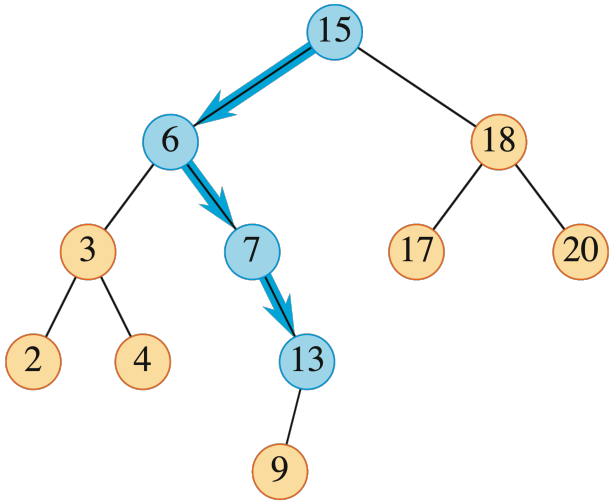
Algorithme qui recherche le nœud contenant une valeur donnée :

Arbre-Rechercher(x, k)

---

si $x = NIL$ **ou** $k = x.clé$ **alors**
  retourner x
si $k < x.clé$ **alors**
  retourner Arbre-Rechercher($x.gauche, k$)
sinon
  retourner Arbre-Rechercher($x.droite, k$)

Recherche d'un élément



Arbre-Minimum et Arbre-Maximum

Algorithmes (itératifs) qui recherchent les nœuds avec les plus petites et plus grande valeurs :

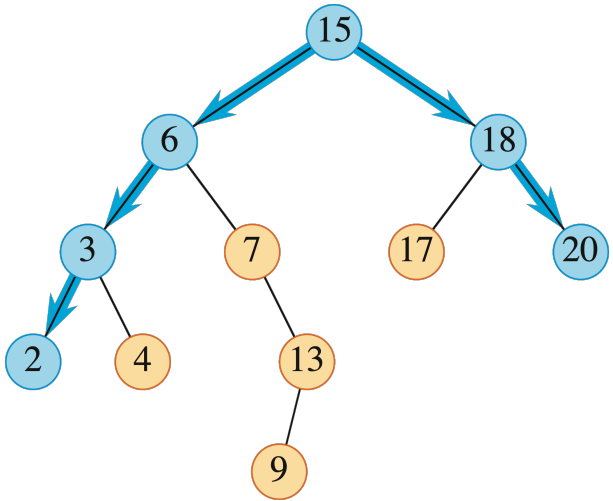
Arbre-Minimum(x)

tant que $x.gauche \neq NIL$ **faire**
  $x \leftarrow x.gauche$
retourner x

Arbre-Maximum(x)

tant que $x.droite \neq NIL$ **faire**
  $x \leftarrow x.droite$
retourner x

Recherche du minimum et du maximum



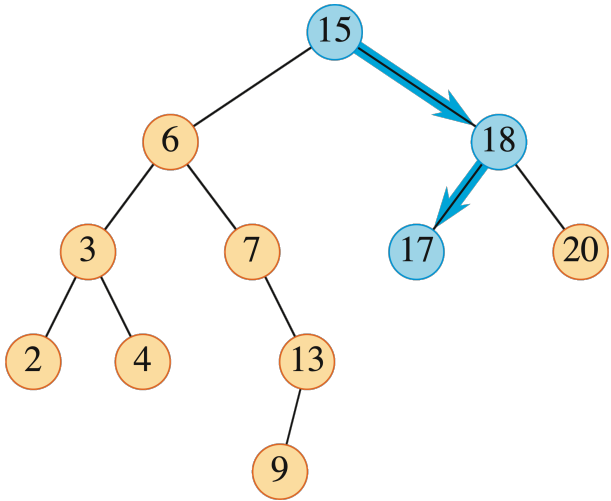
Arbre-Successeur

Arbre-Successeur(x)

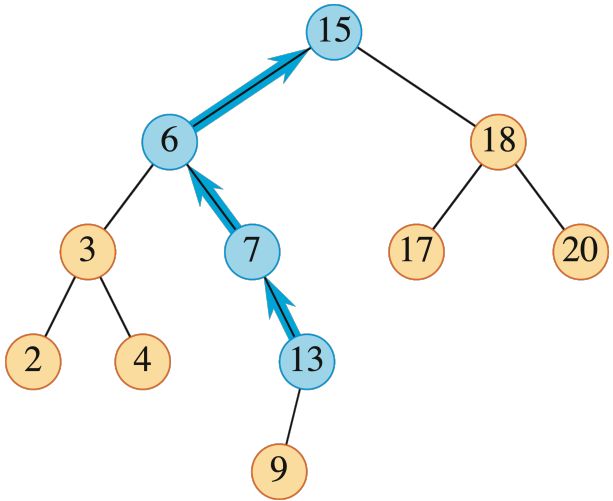
si $x.droite \neq NIL$ **alors**
  retourner Arbre-Minimum($x.droite$)
$y \leftarrow x.parent$
tant que $y \neq NIL$ **et** $x = y.droite$ **faire**
  $x \leftarrow y$
  $y \leftarrow y.parent$
retourner y

- Recherche du *successeur*, le nœud ayant la valeur la plus petite qui est supérieure à x.
- Si le sous-arbre de droite n'est pas vide, alors c'est son nœud minimum.
- Sinon, il faut remonter dans l'arbre jusqu'au premier nœud par lequel on arrive par la gauche.

Recherche du successeur



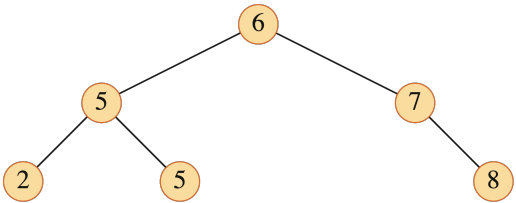
Recherche du successeur

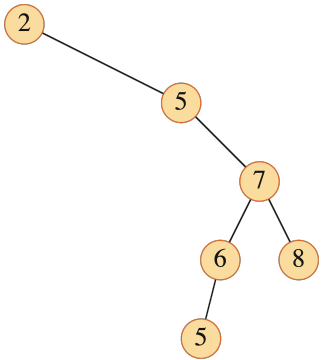


Analyse de complexité

Les opérations Rechercher, Minimum/Maximum et Successeur/Prédécesseur peuvent se faire en temps $O(h)$ sur un arbre binaire de recherche de hauteur h.

Exemples d'arbres binaires





Question

On suppose que des entiers compris entre 1 et 1000 sont disposés dans un arbre binaire de recherche et que l'on souhaite trouver le nœud dont la valeur est 363. Parmi les séquences suivantes, lesquelles ne pourraient pas être la suite des nœuds parcourus ?

1. 2, 252, 401, 398, 330, 344, 397, 363
2. 924, 220, 911, 244, 898, 258, 362, 363
3. 925, 202, 911, 240, 912, 245, 363
4. 2, 399, 387, 219, 266, 382, 381, 278, 363

Question

On suppose que des entiers compris entre 1 et 1000 sont disposés dans un arbre binaire de recherche et que l'on souhaite trouver le nœud dont la valeur est 363. Parmi les séquences suivantes, lesquelles ne pourraient pas être la suite des nœuds parcourus ?

1. 2, 252, 401, 398, 330, 344, 397, 363
2. 924, 220, 911, 244, 898, 258, 362, 363
3. 925, 202, 911, 240, 912, 245, 363 ✓$(912 > 911)$
4. 2, 399, 387, 219, 266, 382, 381, 278, 363

Plan

Arbre-Insérer

```
Arbre-Insérer(T, z)
y ← NIL
x ← T.racine
tant que x ≠ NIL alors
    y ← x
    si z.clé < x.clé alors
        x ← x.gauche
    sinon
        x ← x.droite
z.parent ← y
si y = NIL alors
    T.racine ← z
sinon si z.clé < y.clé alors
    y.gauche ← z
sinon
    y.droite ← z
```

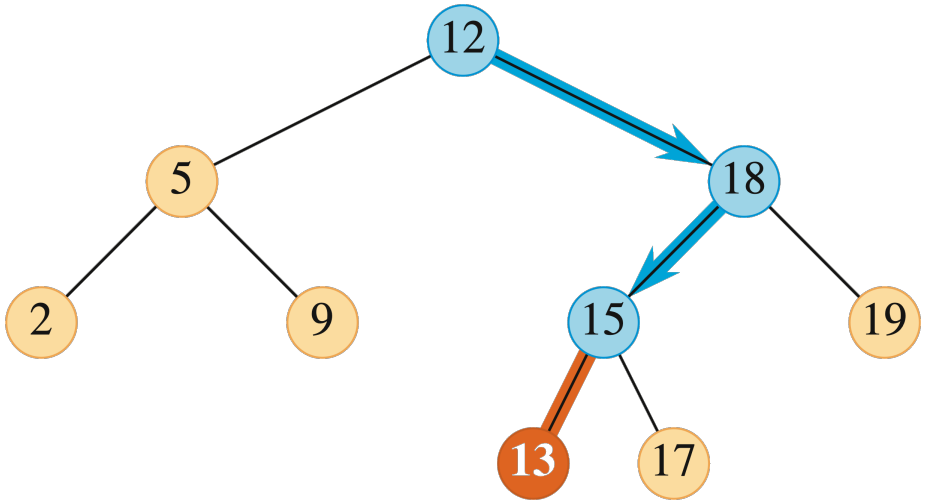
- Descente de l'arbre pour trouver où insérer le nœud (similaire à une recherche).
- On conserve dans y le parent de x (lorsque x atteint *NIL*, on insère dans y).
- Insertion en tant que nouvelle feuille soit à gauche, soit à droite.

Arbre-Insérer

```
Arbre-Insérer(T, z)
y ← NIL
x ← T.racine
tant que x ≠ NIL alors
    y ← x
    si z.clé < x.clé alors
        x ← x.gauche
    sinon
        x ← x.droite
z.parent ← y
si y = NIL alors
    T.racine ← z
sinon si z.clé < y.clé alors
    y.gauche ← z
sinon
    y.droite ← z
```

- Descente de l'arbre pour trouver où insérer le nœud (similaire à une recherche).
- On conserve dans y le parent de x (lorsque x atteint *NIL*, on insère dans y).
- Insertion en tant que nouvelle feuille soit à gauche, soit à droite.

Exemple d'insertion

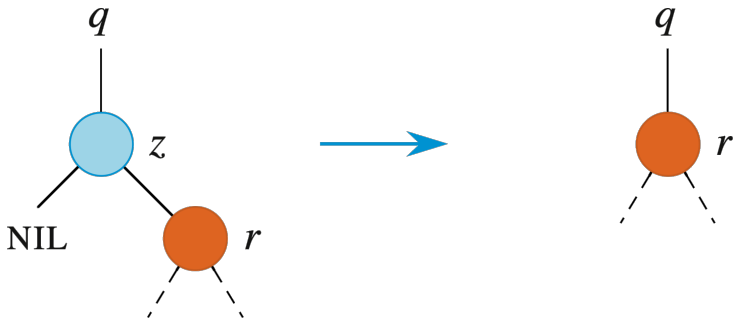


Description de ARBRE-SUPPRIMER

Plusieurs cas sur le nœud à supprimer :

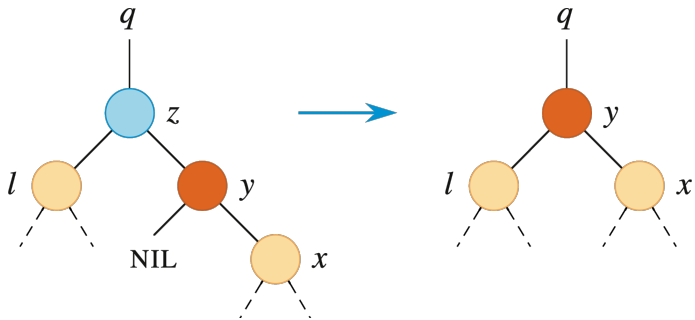
- Il n'a pas d'enfant (feuille) : il suffit de l'enlever et de mettre *NIL* dans son parent.
- Il a exactement un enfant : on élève ce dernier pour qu'il remplace le nœud à supprimer.
- Il a deux enfants : on élève le successeur (qui n'a au plus qu'un seul enfant) pour qu'il remplace le nœud à supprimer.

Cas pour la suppression

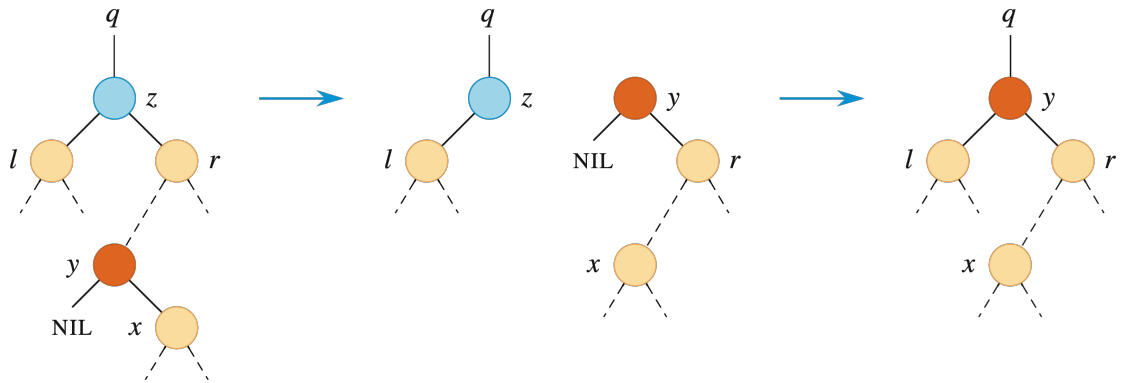


Cas pour la suppression

Cas pour la suppression



Cas pour la suppression



Transplanter

Transplanter(T, u, v)

si $u.parent = NIL$ **alors**
  $T.racine \leftarrow v$
sinon si $u = u.parent.gauche$ **alors**
  $u.parent.gauche \leftarrow v$
sinon
  $u.parent.droite \leftarrow v$
si $v \neq NIL$ **alors**
  $v.parent \leftarrow u.parent$

- Sert à déplacer des sous-arbres dans l'arbre binaire de recherche.
- Remplace le sous-arbre enraciné en u par le sous-arbre enraciné v.
- Le nœud u peut être vu comme le nœud à supprimer.

Pseudo-code d'Arbre-Supprimer

```
Arbre-Supprimer(T, z)
si z.gauche = NIL alors
    Transplanter(T, z, z.droite)
sinon si z.droite = NIL alors
    Transplanter(T, z, z.gauche)
sinon
    y ← Arbre-Minimum(z.droite)
    si y ≠ z.droite alors
        Transplanter(T, y, y.droite)
        y.droite ← z.droite
        y.droite.parent ← y
    Transplanter(T, z, y)
    y.gauche ← z.gauche
    y.gauche.parent ← y
```

- Les deux premiers appels à Transplanter correspondent aux cas les plus directs.
- Le troisième sert à élever le successeur dans le cas où celui-ci n'est pas un enfant direct du nœud à supprimer.
- Le dernier appel supprime finalement le nœud.

Pseudo-code d'ARBRE-SUPPRIMER

ARBRE-SUPPRIMER(T, z)

si $z.gauche = NIL$ **alors**
  TRANSPLANTER$(T, z, z.droite)$
sinon si $z.droite = NIL$ **alors**
  TRANSPLANTER$(T, z, z.gauche)$
sinon
  $y \leftarrow$ ARBRE-MINIMUM$(z.droite)$
  si $y \neq z.droite$ **alors**
    TRANSPLANTER$(T, y, y.droite)$
    $y.droite \leftarrow z.droite$
    $y.droite.parent \leftarrow y$
  TRANSPLANTER(T, z, y)
  $y.gauche \leftarrow z.gauche$
  $y.gauche.parent \leftarrow y$

- Les deux premiers appels à TRANSPLANTER correspondent aux cas les plus directs.
- Le troisième sert à élever le successeur dans le cas où celui-ci n'est pas un enfant direct du nœud à supprimer.
- Le dernier appel supprime finalement le nœud.

Analyse de complexité

Les opérations Insérer et Supprimer peuvent se faire en temps $O(h)$ sur un arbre binaire de recherche de hauteur h.

Complexité en moyenne

Curiosité

- Hauteur moyenne d'un arbre résultant de n insertions aléatoires : $O(\log n)$ (même preuve que pour la complexité moyenne du tri rapide).
- Hauteur moyenne d'un arbre de taille n après des insertions et suppressions aléatoires : $O(\sqrt{n})$.
- Problème ouvert : quand l'algorithme de suppression alterne successeur/minimum et prédécesseur/maximum, on conjecture que la hauteur moyenne est en $O(\log n)$.

Question

Quelle succession d'opérations dans un arbre vide aboutit à un arbre le mieux équilibré ?

1. Insérer 1, 7, 2, 6, 3, 4, 5, puis supprimer 1
2. Insérer 4, 2, 6, 1, 3, 5, 7, puis supprimer 4
3. Insérer 4, 3, 5, 2, 6, 1, 7, puis supprimer 4
4. Insérer 1, 2, 3, 4, 5, 6, 7, puis supprimer 1

Question

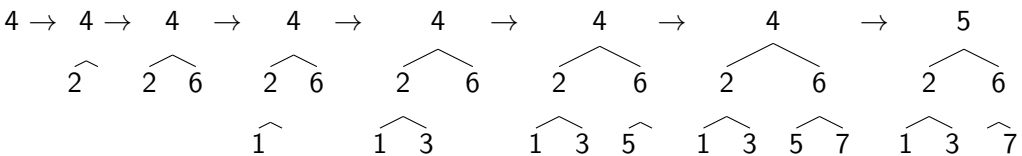
Quelle succession d'opérations dans un arbre vide aboutit à un arbre le mieux équilibré ?

1. Insérer 1, 7, 2, 6, 3, 4, 5, puis supprimer 1
2. Insérer 4, 2, 6, 1, 3, 5, 7, puis supprimer 4 ✓(hauteur de 2)
3. Insérer 4, 3, 5, 2, 6, 1, 7, puis supprimer 4
4. Insérer 1, 2, 3, 4, 5, 6, 7, puis supprimer 1

Question

Quelle succession d'opérations dans un arbre vide aboutit à un arbre le mieux équilibré ?

1. Insérer 1, 7, 2, 6, 3, 4, 5, puis supprimer 1
2. Insérer 4, 2, 6, 1, 3, 5, 7, puis supprimer 4 ✓(hauteur de 2)
3. Insérer 4, 3, 5, 2, 6, 1, 7, puis supprimer 4
4. Insérer 1, 2, 3, 4, 5, 6, 7, puis supprimer 1



Plan

Résumé

Contenu

- Arbre binaire de recherche : arbre binaire + propriété clés inférieures à gauche, supérieures à droite.
- Parcours infixe, préfixe, postfixe.
- Recherche, minimum/maximum, prédécesseur/successeur, insertion et suppression en $O(h)$ (h la hauteur de l'arbre).