

TD Algo2 – session 3 – Arbres binaires de recherche

11 avril 2025

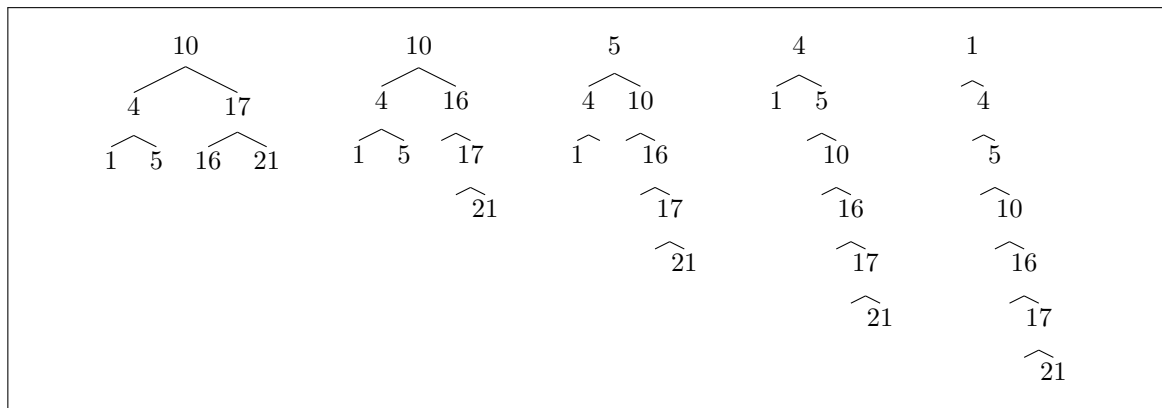
Objectifs d'apprentissage :

- manipuler la structure d'arbre binaire de recherche et étudier les implications de la propriété d'arbre de recherche ;
- concevoir des algorithmes relatifs aux arbres binaires de recherche ;
- mettre en pratique des techniques de preuve pour la validité ou la complexité d'algorithmes.

Les 6 premiers exercices sont essentiels. Plus d'exercices sur <https://algs4.cs.princeton.edu/32bst/>.

Exercice 1 : Hauteurs différentes

Pour les valeurs (1, 4, 5, 10, 16, 17, 21), construire des arbres binaires de recherche de hauteur 2, 3, 4, 5 et 6.



Exercice 2 : Énumération

Combien y a-t-il d'arbres binaires de recherche dont les valeurs sont (3, 5, 8, 12) ?

Il y a 14 possibilités :

- 5 avec 3 en tant que racine.
- 2 avec 5 en tant que racine.
- 2 avec 8 en tant que racine.
- 5 avec 12 en tant que racine.

On peut aussi utiliser le nombre de Catalan :

$$\frac{(2n)!}{(n+1)!n!}$$

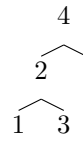
où n est le nombre de nœud.

Exercice 3 : Contre-exemple

Le professeur Szmoldu pense avoir découvert une remarquable propriété des arbres binaires de recherche. Supposer que la recherche d'une clé k dans un arbre binaire de recherche se termine sur

une feuille. On considère trois ensembles : A , les clés situées à gauche du chemin de recherche ; B , celles situées sur le chemin de recherche ; et C , les clés situées à droite du chemin de recherche. Le professeur Szmoldu affirme que, étant donnés trois $a \in A, b \in B, c \in C$ quelconques, ils doivent satisfaire à $a \leq b \leq c$. Donner un contre-exemple qui est le plus petit possible.

Pour $k = 1$:



Exercice 4 : Récursivité

Donner une version récursive de la procédure ARBRE-INSÉRER.

```

ARBRE-INSÉRER( $T, z$ )
  si  $T.racine = NIL$  alors
     $T.racine \leftarrow z$ 
  sinon
    ARBRE-INSÉRER-RÉCURSIF( $z, T.racine$ )
  
```

```

ARBRE-INSÉRER-RÉCURSIF( $z, x$ )
  si  $z.clé < x.clé$  alors
    si  $x.gauche = NIL$  alors
       $x.gauche \leftarrow z$ 
       $z.parent \leftarrow x$ 
    sinon
      ARBRE-INSÉRER( $z, x.gauche$ )
  sinon
    si  $x.droite = NIL$  alors
       $x.droite \leftarrow z$ 
       $z.parent \leftarrow x$ 
    sinon
      ARBRE-INSÉRER( $z, x.droite$ )
  
```

Exercice 5 : Valeur inférieure

Proposer le pseudo-code d'un algorithme qui trouve le nœud qui possède la valeur la plus grande mais qui soit inférieure ou égale à celle qui est donnée à l'algorithme.

```

VALEUR-INFÉRIEURE( $T, v$ )
   $y \leftarrow NIL$ 
   $x \leftarrow T.racine$ 
  tant que  $x \neq NIL$  faire
    si  $x.clé \leq v$  alors
       $y \leftarrow x$ 
       $x \leftarrow x.droite$ 
    sinon
       $x \leftarrow x.gauche$ 
  retourner  $y$ 
  
```

Exercice 6 : Rang

On souhaite un algorithme qui calcule le rang d'une valeur, c'est-à-dire, le nombre de nœuds dont les

valeurs sont inférieures ou égales à celle qui est donnée à l'algorithme. Comment faudrait-il procéder pour avoir un algorithme en $O(h)$? Proposer en pseudo-code cet algorithme.

Pour que la méthode soit en $O(h)$, il faut rajouter un champ à chaque nœud et le mettre à jour à chaque insertion ou suppression : la taille du sous-arbre.

```

RANG( $T, v$ )


---


rang  $\leftarrow 0$ 
 $x \leftarrow T.racine$ 
tant que  $x \neq NIL$  faire
    si  $x.clé \leq v$  alors
        rang  $\leftarrow$  rang + 1
        si  $x.gauche \neq NIL$  alors
            rang  $\leftarrow$  rang +  $x.gauche.taille$ 
         $x \leftarrow x.droite$ 
    sinon
         $x \leftarrow x.gauche$ 
retourner rang


---



```

Exercice 7 : Arbre-Supprimer

Comment pourrait-on modifier l'algorithme de suppression pour alterner alternant la recherche/élévation du successeur avec la recherche/élévation d'un prédécesseur. Fournir le pseudo-code.

On pourrait soit faire un choix aléatoire, soit tester la parité de la clé insérée pour plus de rapidité.

```

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 si  $z.clé \bmod 2 = 0$  alors
     $y \leftarrow$  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( $T, z, y$ )
     $y.gauche \leftarrow z.gauche$ 
     $y.gauche.parent \leftarrow y$ 
sinon
     $y \leftarrow$  ARBRE-MAXIMUM( $z.gauche$ )
    si  $y \neq z.gauche$  alors
        TRANSPLANTER( $T, y, y.gauche$ )
         $y.gauche \leftarrow z.gauche$ 
         $y.gauche.parent \leftarrow y$ 
    TRANSPLANTER( $T, z, y$ )
     $y.droite \leftarrow z.droite$ 
     $y.droite.parent \leftarrow y$ 


---



```

Exercice 8 : Prédécesseur

Fournir le pseudo-code permettant de rechercher le prédécesseur du nœud x .

ARBRE-PRÉDÉCESSEUR(x)

si $x.gauche \neq NIL$ **alors**
 retourner ARBRE-MAXIMUM($x.gauche$)
 $y \leftarrow x.parent$
tant que $y \neq NIL$ et $x = y.gauche$ **faire**
 $x \leftarrow y$
 $y \leftarrow y.parent$
retourner y

Exercice 9 : Équilibre

On considère tous les nombres compris entre 1 et 1000. Donner deux ordres d'insertion de ces nombres dans un arbre binaire de recherche :

- l'un qui va donner un arbre complètement déséquilibré, c'est-à-dire de hauteur maximale ;
- l'autre qui va donner un arbre équilibré, c'est-à-dire le moins haut possible.

Écrire un algorithme qui les génère.

- Dans le premier cas, il suffit d'insérer dans un ordre croissant ou décroissant ;
- Dans le second cas, il faut réaliser une "dichotomie" sur les valeurs : commencer par le milieu puis le milieu de chaque segment ainsi obtenu, etc. Voici la méthode avec $a = 1$ et $b = 1001$.

GÉNÉRER-SÉQUENCE(a, b)

$i \leftarrow (a + b)/2$
AFFICHER(i)
si $a < b - 1$ **alors**
 GÉNÉRER-SÉQUENCE(a, i)
 GÉNÉRER-SÉQUENCE($i + 1, b$)

Exercice 10 : Ancêtre

Étant donné un arbre, comment vérifier si v est un ancêtre de w ?

v est un ancêtre de w si et seulement si v apparaît avant w dans l'ordre préfixe et après dans l'ordre postfixe.

Exercice 11 : Tri arborescent

On peut trier un ensemble donné de n nombres en commençant par construire un arbre binaire de recherche contenant ces nombres (en répétant ARBRE-INSÉRER pour insérer les nombres un à un), puis en imprimant les nombres via un parcours infixe de l'arbre. Analyser les temps d'exécution de cet algorithme de tri, dans le pire et dans le meilleur des cas ?

- Pire cas : n insertions de valeurs triées, donc de coût 1, 2, ..., n . $O(n^2)$ en tout.
- Meilleur cas : arbre équilibré de hauteur $\log(n)$. Donc 1 insertion de coût 1, 2 insertions de coût 2, 4 insertions de coût 3, ... : $\sum_{i=0}^{\log n} 2^i \times O(i) = O(n \log n)$.