

## Algorithmique et structures de données

### CM 8 – Arbres binaires de recherche

Jean-Marie Le Bars  
jean-marie.lebars@unicaen.fr

## Plan du CM 8

Motivation

Définition d'un ABR

Recherche d'un élément dans un ABR

Insertion d'un élément dans un ABR

Suppression d'un élément dans un ABR

## Plan du CM 8

Motivation

Définition d'un ABR

Recherche d'un élément dans un ABR

Insertion d'un élément dans un ABR

Suppression d'un élément dans un ABR

# Manipulation de données

## Manipulation de données

On souhaite stocker des données et pouvoir les récupérer.  
Les fonctions principales sont

- la **recherche** : on souhaite récupérer une donnée stockée
- l'**insertion** : on souhaite ajouter une nouvelle donnée
- la **suppression** : on souhaite supprimer une donnée

## Une solution : les structures linéaires

Les structures linéaires telles que les **tableaux** et les **listes chaînées** permettent ces opérations, mais elles ne sont pas efficaces

voir le CM 9 et le TD 9 sur les complexités

## Recherche d'un élément dans un tableau – Rappels

### Tableau non trié

- on recherche un élément dans un tableau contenant  $n$  éléments.
- la recherche s'effectue **en moyenne en  $\frac{n+1}{2}$  instructions.**
- la recherche s'effectue **dans le pire des cas en  $n$  instructions.**

### Tableau trié

- On effectue une **recherche dichotomique.**
- Complexité en moyenne et dans le pire des cas en  **$\log_2 n$ .**

## Recherche d'un élément dans une liste chaînée – Rappels

### Liste chaînée

- on recherche un élément dans une liste contenant  $n$  nœuds.
- la recherche s'effectue **en moyenne en  $\frac{n+1}{2}$  instructions.**
- la recherche s'effectue **dans le pire des cas en  $n$  instructions.**

### Liste chaînée triée

- si la liste est triée, la complexité ne change pas,
- sauf dans le cas où l'élément n'appartient pas à la liste
  - on interrompt la recherche, lorsque l'on arrive sur un nœud de valeur supérieure à celle recherchée
- on peut également améliorer les complexités en mettant **plusieurs pointeurs** sur la liste.
- il existe d'autres structures de données intermédiaires entre listes chaînées et arbre binaire de recherche.

## Insertion et suppression dans un tableau – rappels

### Tableau non trié

- l'insertion et la suppression s'effectue **en moyenne en  $\frac{n}{2}$  instructions.**
- l'insertion et la suppression s'effectue **dans le pire des cas en  $n$  instructions.**

### Tableau trié

Pour le calcul de la complexité, nous devons considérer (au moins) deux types d'instructions

1. la consultation de la valeur d'une case
2. le décalage d'un élément du tableau

On obtient les résultats suivants :

1. on peut diminuer le nombre de consultations
2. on ne peut pas diminuer le nombre de décalages

## Insertion et suppression dans une liste chaînée – rappels

### Liste chaînée non trié

- l'insertion et la suppression s'effectue **en moyenne en  $\frac{n}{2}$  instructions.**
- l'insertion et la suppression s'effectue **dans le pire des cas en  $n$  instructions.**

### Liste chaînée triée

- mêmes complexités qu'avec une liste chaînée non trié.
- on peut améliorer les complexités en mettant plusieurs pointeurs sur la liste.
- amélioration pour les autres structures de données



## Notre objectif

### Effectuer les trois opérations suivantes

- recherche
- suppression
- insertion

avec une complexité en  $O(\log n)$  en moyenne et si possible aussi dans le pire des cas.

### Possible avec des arbres

- la profondeur de la plupart des feuilles doit être en  $O(\log n)$ 
  - ▶ pour la complexité en moyenne
- mieux :  $h$  –la hauteur de l'arbre– doit être en  $\Theta(\log n)$ 
  - ▶ pour la complexité dans le pire des cas

## Plan du CM 9

Motivation

Définition d'un ABR

Recherche d'un élément dans un ABR

Insertion d'un élément dans un ABR

Suppression d'un élément dans un ABR

## Définition d'un arbre binaire de recherche (ABR)

Les arbres binaires de recherche sont des arbres binaires

On utilise donc toujours la structure de nœud :

```
structure noeud
    valeur : entier
    gauche : pointeur sur noeud
    droit : pointeur sur noeud
```

type ABR = pointeur sur noeud

## Définition 1 d'un arbre binaire de recherche

On suppose ici que toutes les valeurs de l'arbre sont différents

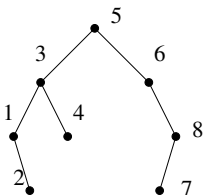
### Définition inductive d'un ABR

- **initialisation**  
l'arbre vide et l'arbre racine (pour n'importe quelle valeur) sont des ABR
- **hérédité ou induction**  
Soit  $A = (x, A_g, A_d)$  un arbre binaire,  $A$  est un ABR lorsque
  1.  $A_g$  et  $A_d$  sont des ABR
  2. tout nœud de  $A_g$  à une valeur inférieure à  $x$
  3. tout nœud de  $A_d$  à une valeur supérieure à  $x$

Attention, les conditions 2 et 3 ne suffisent pas

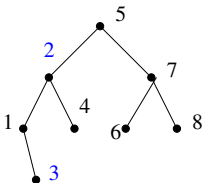
## Définition 1 d'un arbre binaire de recherche

### Exemple



### Contre-exemple

L'arbre binaire suivant satisfait les conditions 2 et 3, mais il n'est pas un ABR car la condition 1 n'est pas satisfaite.



## Définition 2 d'un arbre binaire de recherche

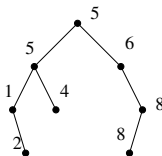
On suppose ici que certaines valeurs peuvent apparaître plusieurs fois

Nous sommes obligés de changer un peu la définition.

### Définition inductive d'un ABR

- **initialisation**  
l'arbre vide et l'arbre racine (pour n'importe quelle valeur) sont des ABR
- **hérédité ou induction**  
Soit  $A = (x, A_g, A_d)$  un arbre binaire,  $A$  est un ABR lorsque
  1.  $A_g$  et  $A_d$  sont des ABR
  2. tout nœud de  $A_g$  à une valeur inférieure ou égale à  $x$
  3. tout nœud de  $A_d$  à une valeur supérieure à  $x$

### Exemple



## Plan du CM 9

Motivation

Définition d'un ABR

**Recherche d'un élément dans un ABR**

Insertion d'un élément dans un ABR

Suppression d'un élément dans un ABR

## Recherche d'un élément $x$ dans un ABR

On retourne un pointeur sur un nœud de valeur  $x$ .

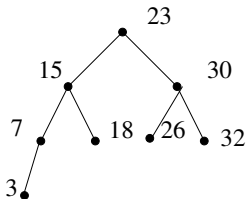
### Méthode

On parcourt les nœuds de la manière suivante

1. si le nœud vaut None, alors  $x$  n'apparaît pas dans l'ABR, on retourne None
2. si la valeur du nœud vaut  $x$ , on retourne un pointeur sur ce nœud
3. si la valeur du nœud est plus grande que  $x$ , on va vers le sous-arbre gauche
4. si la valeur du nœud est plus petite que  $x$ , on va vers le sous-arbre droit

### Exemple

Recherchez 7 et 28





## Recherche d'un élément $x$ dans un ABR

On retourne un pointeur sur un nœud de valeur  $x$ .

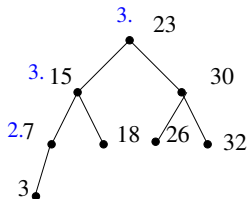
### Méthode

On parcourt les nœuds de la manière suivante

1. si le nœud vaut None, alors  $x$  n'apparaît pas dans l'ABR, on retourne None
2. si la valeur du nœud vaut  $x$ , on retourne un pointeur sur ce nœud
3. si la valeur du nœud est plus grande que  $x$ , on va vers le sous-arbre gauche
4. si la valeur du nœud est plus petite que  $x$ , on va vers le sous-arbre droit

### Exemple

Recherche de la valeur 7



## Recherche d'un élément $x$ dans un ABR

On retourne un pointeur sur un nœud de valeur  $x$ .

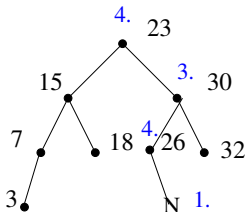
### Méthode

On parcourt les nœuds de la manière suivante

1. si le nœud vaut None, alors  $x$  n'apparaît pas dans l'ABR, on retourne None
2. si la valeur du nœud vaut  $x$ , on retourne un pointeur sur ce nœud
3. si la valeur du nœud est plus grande que  $x$ , on va vers le sous-arbre gauche
4. si la valeur du nœud est plus petite que  $x$ , on va vers le sous-arbre droit

### Exemple

Recherche de la valeur 28



## Recherche d'un élément $x$ dans un ABR

On retourne un pointeur sur un nœud de valeur  $x$ .

### Méthode

1. si le nœud vaut None, alors  $x$  n'apparaît pas dans l'ABR, on retourne None
2. si la valeur du nœud vaut  $x$ , on retourne un pointeur sur ce nœud
3. si la valeur du nœud est plus grande que  $x$ , on va vers le sous-arbre gauche
4. si la valeur du nœud est plus petite que  $x$ , on va vers le sous-arbre droit

### Algorithme récursif

```

rechercheABRrec(A : ABR, x : entier) : ABR
    si A = None alors                                1.
        retourner None
    si A->valeur = x alors                            2.
        retourner A
    si A->valeur > x alors                             3.
        retourner rechercheABRrec(A->gauche)
    retourner rechercheABRrec(A->droit)              4.
    
```

## Recherche d'un élément $x$ dans un ABR

On retourne un pointeur sur un nœud de valeur  $x$ .

### Méthode

1. si le nœud vaut None, alors  $x$  n'apparaît pas dans l'ABR, on retourne None
2. si la valeur du nœud vaut  $x$ , on retourne un pointeur sur ce nœud
3. si la valeur du nœud est plus grande que  $x$ , on va vers le sous-arbre gauche
4. si la valeur du nœud est plus petite que  $x$ , on va vers le sous-arbre droit

### Algorithme récursif

Nous pouvons regrouper 1. et 2.

```

rechercheABRrec(A : ABR, x : entier) : ABR
    si A = None ou A->valeur = x alors                1. et 2.
        retourner A
    si A->valeur > x alors                               3.
        retourner rechercheABRrec(A->gauche)
    retourner rechercheABRrec(A->droit)                 4.
    
```

## Recherche d'un élément $x$ dans un ABR

On retourne un pointeur sur un nœud de valeur  $x$ .

### Méthode

1. si le nœud vaut None, alors  $x$  n'apparaît pas dans l'ABR, on retourne None
2. si la valeur du nœud vaut  $x$ , on retourne un pointeur sur ce nœud
3. si la valeur du nœud est plus grande que  $x$ , on va vers le sous-arbre gauche
4. si la valeur du nœud est plus petite que  $x$ , on va vers le sous-arbre droit

### Algorithme itératif

La condition d'arrêt regroupe les conditions 1. et 2.

```

rechercheABRit(A : ABR, x : entier) : ABR
    tant que A <> None et A->valeur <> x faire
        si A->valeur > x alors
            A = A->gauche
        sinon A = A->droit
    retourner A
    
```

## Complexité de la recherche

Branche = chemin de la racine jusqu'à une feuille

### Parcours d'une branche

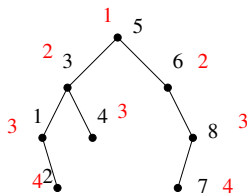
L'algorithme de recherche (récursif ou itératif) effectue le parcours d'une branche

- complètement dans le cas où  $x$  apparaît dans une feuille ou qu'il n'est pas une valeur de l'ABR
- partiellement dans les autres cas

### Complexité

On suppose que le coût est le nombre de comparaisons.  
Le coût est donc au plus de  $h$ , où  $h$  est la hauteur de l'arbre.

### Exemple



## Plan du CM 9

Motivation

Définition d'un ABR

Recherche d'un élément dans un ABR

**Insertion d'un élément dans un ABR**

Suppression d'un élément dans un ABR

## Insertion sur les feuilles

### Insertion

On souhaite ajouter un nœud contenant la valeur  $x$ .

Si  $x$  est déjà présent, on ne fait rien.

### Méthode

On se déplace sur une branche (comme pour la recherche) et on effectue l'insertion en fin de branche.

### Algorithme récursif

```
insererABRrec(A : ABR, x : entier) : ABR
  si A = None alors
    A=Nouveau(noeud), A->valeur=x, A->gauche=None, A->droit=None
  sinon
    si A->valeur > x alors
      A->gauche = insererABRrec(A->gauche, x)
    sinon
      si A->valeur < x alors
        A->droit = insererABRrec(A->droit, x)
      #si A->valeur = x on ne fait rien
  retourner A
```



## Insertion sur les feuilles

### Algorithme itératif (voir TD)

On peut effectuer le déplacement sur une branche de manière itérative.  
Pour cela, on utilise l'instruction

```
tmp = tmp->gauche
```

ou

```
tmp = tmp->droit
```

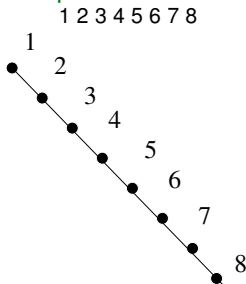
où tmp est un pointeur qui parcourt les nœuds d'une branche

## Ordre de l'insertion

### Forme de l'arbre

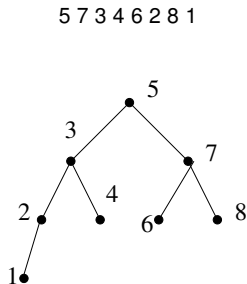
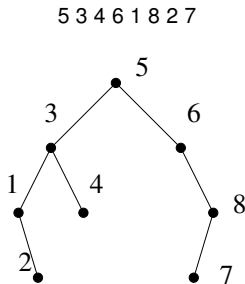
- on souhaite se rapprocher d'un arbre complet
- l'ordre d'insertion des nœuds est très important

### Exemples



arbre dégénéré

Mauvais



arbre parfait

Bon

## Ordre de l'insertion

### Forme de l'arbre en moyenne

- on tire aléatoirement des permutations sur  $\{1, \dots, n\}$  afin d'obtenir un ABR à  $n$  nœuds
- on regarde la hauteur moyenne des ABR que l'on obtient

## Complexité de la recherche d'un élément

### Dépend de la profondeur

- la recherche d'un élément correspond à parcourir une branche jusqu'à trouver l'élément ou arriver à None
- le coût de la recherche dépend donc de la profondeur du nœud contenant la valeur (lorsqu'elle apparaît dans l'ABR)
- les nœuds proches de la racine nécessitent un coût faible
- les nœuds proches des feuilles nécessitent un coût plus fort

## Complexité de l'insertion d'un élément

### Insertion sur les feuilles

- avec l'insertion sur les feuilles, les dernières valeurs insérées sont de grande profondeur
- le coût de la recherche est plus élevé pour les dernières valeurs insérées

### Insertion à la racine (voir TD 8)

- avec l'insertion à la racine, les dernières valeurs insérées sont de petite profondeur
- le coût de la recherche est faible pour les dernières valeurs insérées

### Obsolescence – Effet de mode

Souvent nous avons

- les dernières valeurs insérées sont les valeurs les plus recherchées
- les premières valeurs insérées sont les valeurs les moins recherchées

Dans ce cas, l'insertion à la racine est plus pertinente.

## Plan du CM 9

Motivation

Définition d'un ABR

Recherche d'un élément dans un ABR

Insertion d'un élément dans un ABR

Suppression d'un élément dans un ABR

## Suppression du maximum

Le maximum (la valeur maximum) correspond au nœud le plus à droite.

### Suppression du maximum – algorithme récursif

On retourne l'ABR sans le nœud contenant le maximum et la valeur du maximum aussi. On définit pour cela la structure suivante

```
structure arbreMaxi
```

```
  A : ABR
```

```
  m : entier
```

```
supprimerMaxrec(A : ABR) : arbreMaxi
```

```
  AM : arbreMaxi ; maxi : entier
```

```
  si A->droit = None alors
```

```
    maxi = A->valeur
```

```
    tmp : ABR ; tmp = A ; A = A->gauche
```

```
    désallouer(tmp)
```

```
    AM.A = A ; AM.m = maxi
```

```
  sinon
```

```
    AM = supprimerMaxrec(A->droit)
```

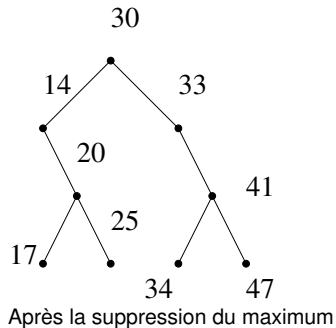
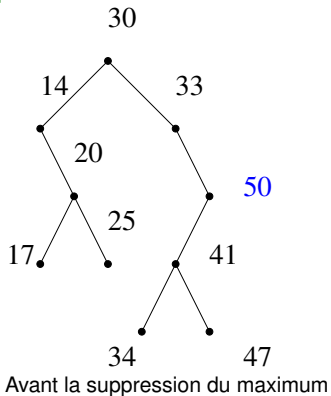
```
    A->droit = AM.A
```

```
  retourner AM
```

### Exercice

Écrivez une procédure itérative pour supprimer le maximum.

## Exemple





# Suppression d'un nœud d'une valeur quelconque

## Objectif

On souhaite supprimer le nœud N de valeur  $x$ .  
On suppose ici que l'arbre contient un tel nœud.

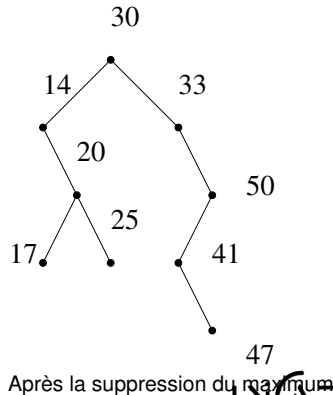
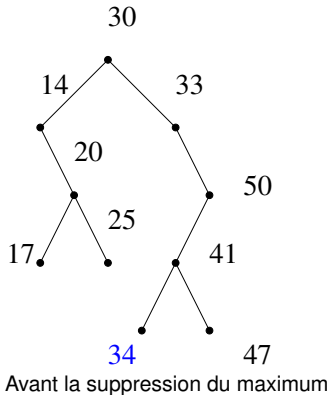
## Cas possibles

1. si N est une feuille  
→ sa suppression ne nécessite pas de réorganisation.
2. si N n'a qu'un seul enfant  
→ on peut le remplacer par cet enfant.
3. si N a deux enfants  
→ on peut le remplacer par le nœud maximum de son enfant gauche.  
On utilise pour cela la procédure *supprimerMaxrec*.

## Cas 1

- si N est une feuille  
→ sa suppression ne nécessite pas de réorganisation.

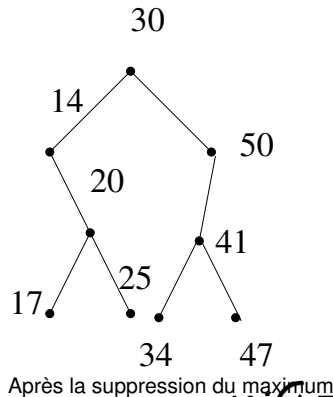
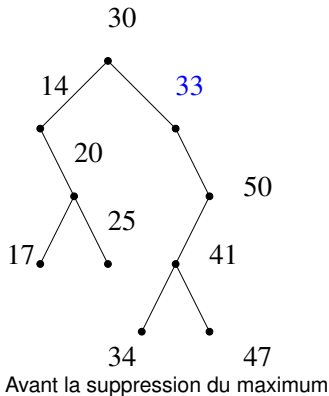
## Suppression de 34



## Cas 2

- si N n'a qu'un seul enfant  
→ on peut le remplacer par cet enfant.

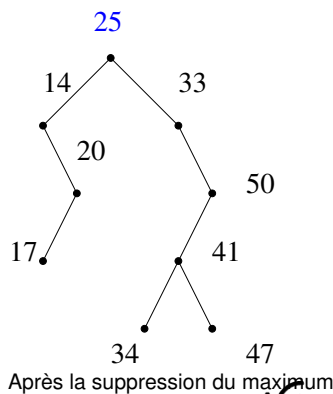
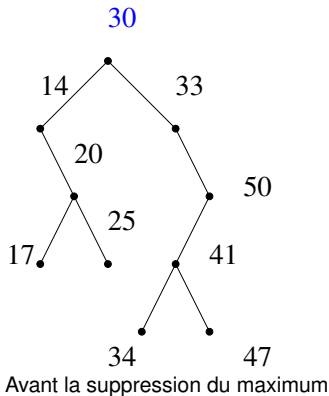
### Suppression de 33



## Cas 3

- si N a deux enfants  
→ on peut le remplacer par le nœud maximum de son enfant gauche.

## Suppression de 30



## Suppression d'un nœud d'une valeur quelconque

### Procédure récursive

```

supprimerABR(A : ABR, x : entier) : ABR
si A = None alors retourner None
si A->valeur > x alors
    A->gauche = supprimerABR(A->gauche, x)
sinon si A->valeur < x alors
    A->droit = supprimerABR(A->droit, x)
    sinon #A->valeur = x
        tmp : ABR
        si A->gauche = None alors
            tmp = A ; A = A->droit
            désallouer(tmp)
        sinon
            si A->droit = None alors
                tmp = A ; A = A->gauche
                désallouer(tmp)
            sinon
                AM : arbreMaxi ; AM = supprimerMaxrec(A->gauche)
                A->gauche = AM.A ; A->valeur = AM.m
retourner A
    
```

## Valeurs d'un ABR présents dans un intervalle

### Problème

On veut afficher tous les valeurs  $x$  telles que  $a \leq x \leq b$ .

### Méthode naïve

On parcourt tous les nœuds et on affiche la valeur si elle est dans l'intervalle.

Méthode valable pour tous les arbres binaires.

```
afficheIntervalle(A:ABR, a:entier, b:entier)
  si A <> None alors
    si A->valeur >= a et A->valeur <= b:
      afficher A->valeur
    afficheIntervalle(A->gauche, a, b)
    afficheIntervalle(A->droit, a, b)
```

### Meilleure solution (voir TP 4)

- nous devons tenir compte de l'ordre des valeurs d'un ABR
- il faut effectuer les deux appels récursifs que lorsque c'est nécessaire