

Slide 1

Le parcours d'un arbre

Les files de priorité

Slide 2

0. Résumé des épisodes précédents

Slide 3

Algorithmes sur les arbres :

arbres de recherche

parcours d'un arbre

files de priorité

Slide 4

I. Le parcours d'un arbre

Le parcours d'un arbre

Effectuer une opération p à chaque nœud d'un arbre

Typiquement : afficher le contenu

Abstraitement : visiter les nœuds

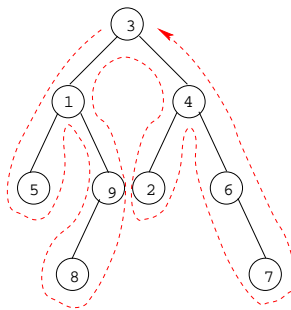
Même opération p

Différence entre deux parcours : l'ordre de visite des nœuds

Une méthode de parcours = un **ordre total** sur les nœuds

Slide 5

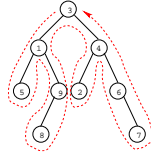
En profondeur d'abord



Slide 6

Slide 7

En profondeur d'abord



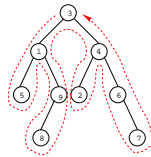
On visite un nœud **en descendant** : préfixe

Entre les deux sous-arbres : infixe

En remontant : postfixe

Slide 8

En profondeur d'abord



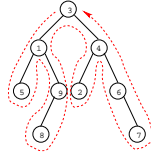
En descendant : préfixe : 3 1 5 9 8 4 2 6 7

Entre les deux sous-arbres : infixe

En remontant : postfixe

Slide 9

En profondeur d'abord



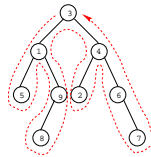
En descendant : préfixe : 3 1 5 9 8 4 2 6 7

Entre les deux sous-arbres : infix : 5 1 8 9 3 2 4 6 7

En remontant : postfixe

Slide 10

En profondeur d'abord

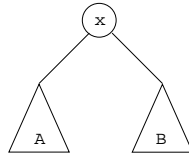


En descendant : préfixe : 3 1 5 9 8 4 2 6 7

Entre les deux sous-arbres : infix : 5 1 8 9 3 2 4 6 7

En remontant : postfixe : 5 8 9 1 2 7 6 4 3

Programmer un parcours infixé



Slide 11

Programmer un parcours infixé



Slide 12

Programmer un parcours infixé

Slide 13



Programmer un parcours infixé

Slide 14



Slide 15

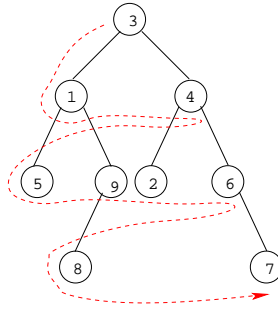
```
Pile p = Pile.empty ();
p.push(a);
while (!(p.testempty())) {
    Arbre b = p.top ();
    p.pop();
    if (b != null) {
        if (singleton(b))
            System.out.print(b.val + " ");
        else {
            p.push(b.droit);
            p.push(new Arbre (b.val,null,null));
            p.push(b.gauche);}}}
```

La récursivité permet d'éviter la pile

Slide 16

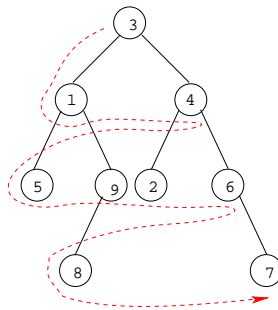
```
static void parcours (Arbre a) {
    if (a != null) {
        parcours(a.gauche);
        System.out.print(a.val + " ");
        parcours(a.droit);}}
```

En largeur d'abord



Slide 17

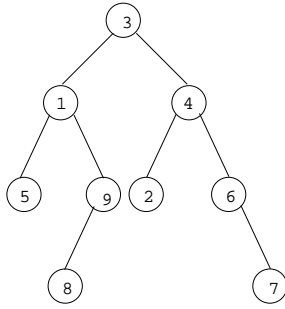
En largeur d'abord



Slide 18

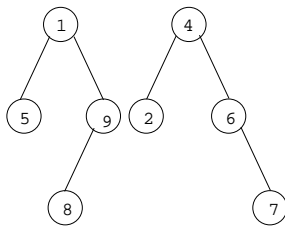
3 1 4 5 9 2 6 8 7

En largeur d'abord



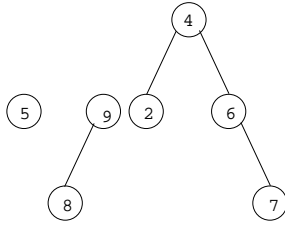
Slide 19

En largeur d'abord



Slide 20

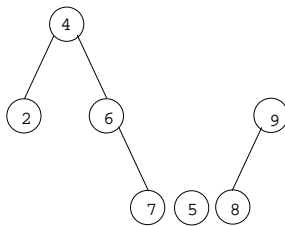
En largeur d'abord



Slide 21

3 1

En largeur d'abord



Slide 22

3 1

En largeur d'abord

Slide 23

La pile est remplacée par une **file**

Peut aussi se programmer récursivement

Mais la récursivité n'économise pas la file

Slide 24

II. Les files de priorité

Urgences

Slide 25

Chaque patient muni d'une priorité

Opérations : recherche du max, insertion d'un élément, suppression du max

Une liste : recherche linéaire

Une liste ordonnée : insertion linéaire

Remplacer la liste par un arbre

Arbre de recherche ordonné par priorité

Slide 26

Recherche, insertion et suppression logarithmique

Mais il faut maintenir l'arbre équilibré

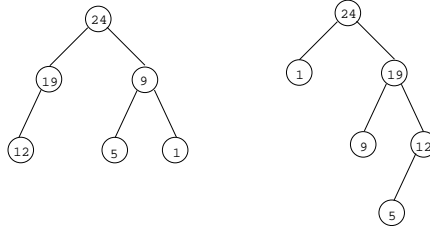
Beaucoup plus simple : arbre partiellement ordonné

Les arbres partiellement ordonnés

Arbre de recherche à gauche : plus petit, à droite : plus grand

Arbre partiellement ordonné tous les enfants : plus petit

Slide 27



Les arbres partiellement ordonnés

Recherche du max : la racine

Suppression du max : le meilleur sous-chef devient chef (et récursivement)

Insertion : une feuille puis on permute avec son chef tant qu'on est meilleur

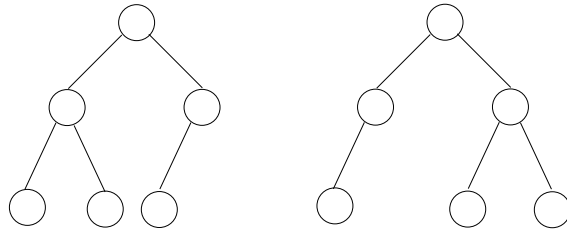
Slide 28

Équilibrer un arbre partiellement ordonné

Arbre toujours de hauteur minimale, et même **tassé**

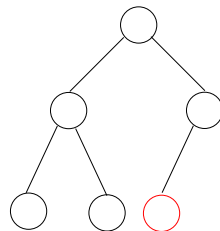
Hauteur minimale et feuille le plus à gauche possible

Slide 29



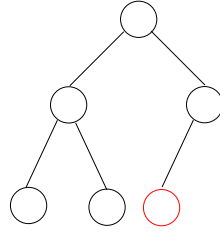
Numéroter les nœuds d'un arbre binaire

Slide 30



a.droite.gauche

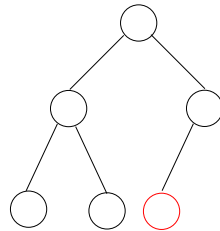
Numéroter les nœuds d'un arbre binaire



Slide 31

droite gauche

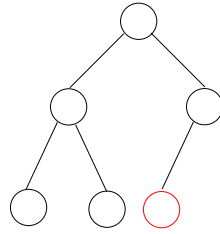
Numéroter les nœuds d'un arbre binaire



Slide 32

1 0

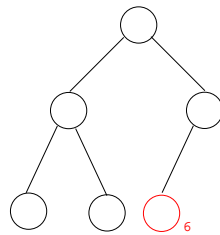
Numéroter les nœuds d'un arbre binaire



Slide 33

1 1 0

Numéroter les nœuds d'un arbre binaire

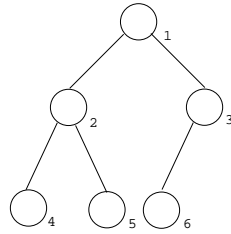


Slide 34

6

Slide 35

Numéroter les nœuds d'un arbre binaire



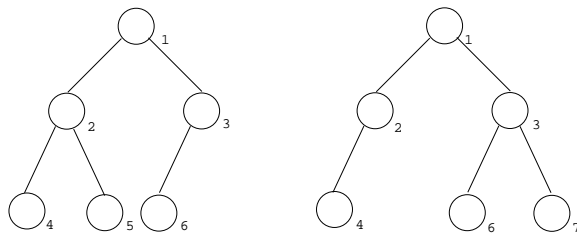
La racine : 1

L'enfant gauche : $2n$

L'enfant droit : $2n + 1$

Slide 36

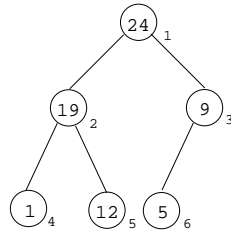
Numéroter les nœuds d'un arbre binaire



Arbre tassé : les numéros des nœuds forment un [intervalle](#) $1 \dots n$

Les tas

Un tas : un arbre tassé partiellement ordonné

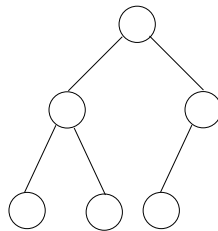


Slide 37

Insertion dans un tas

Recherche : trivial

Insertion



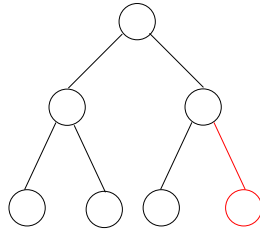
Slide 38

Insertion dans un tas

Recherche : trivial

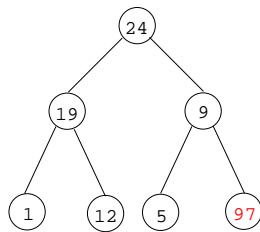
Insertion

Slide 39



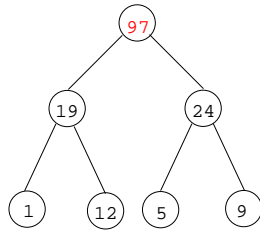
Insertion dans un tas

Slide 40



Permutation avec le parent tant qu'il est plus prioritaire

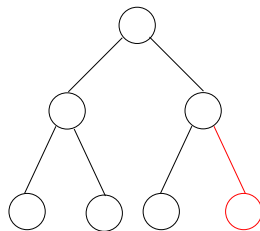
Insertion dans un tas



Slide 41

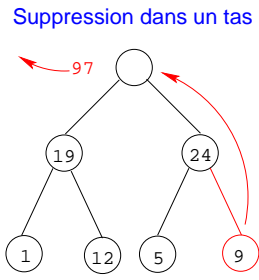
Permutation avec le parent tant qu'il est plus prioritaire

Suppression dans un tas

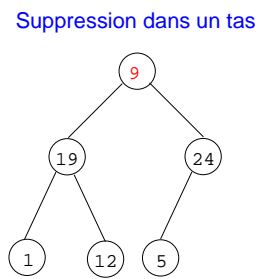


Slide 42

Slide 43

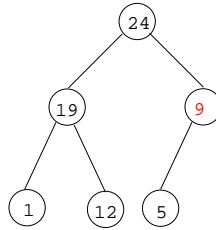


Slide 44



Tant que 9 inférieur à un enfant permutation avec le plus grand

Suppression dans un tas



Slide 45

Tant que 9 inférieur à un enfant permutation avec le plus grand

Les tas

Slide 46

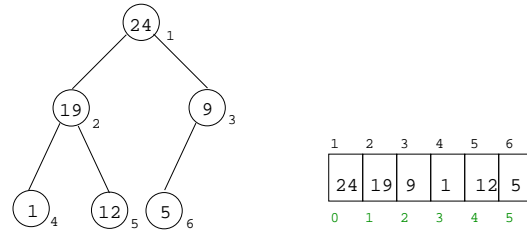
Recherche du max : constant

Insertion et suppression : logarithmique

Les tas dans les tableaux

Les numéros des nœuds sont contigus

Slide 47



Slide 48

```
static void insert (int [] t, int i) {  
    int j = i;  
    while (j >= 1 && t[(j - 1) / 2] <= t[j]) {  
        int z = t[j];  
        t[j] = t[(j - 1) / 2];  
        t[(j - 1) / 2] = z;  
        j = (j - 1) / 2;}}}
```

Slide 49

Exercices : 9.3 et 9.10

Slide 50

La prochaine fois : le tri