Cours 3: Arbres. Parcours.

Olivier Bournez

bournez@lix.polytechnique.fr
LIX, Ecole Polytechnique

2011-12

Algorithmique

Aujourd'hui

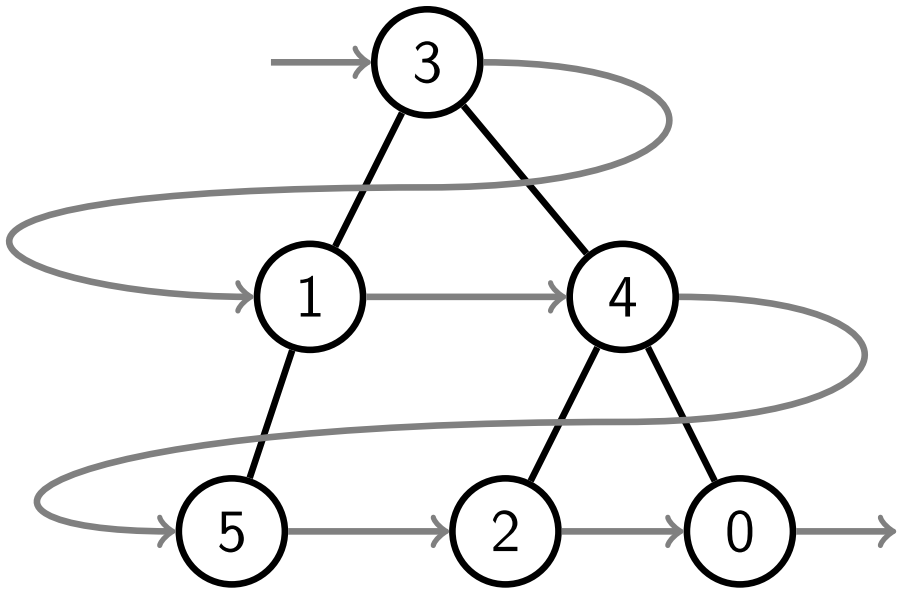
Parcours d'arbres

Parcourir un arbre

- Beaucoup d'algorithmes sur les arbres nécessitent de parcourir (traiter) tous les sommets
  - Exemple: Tester l'existence d'une valeur particulière dans un arbre.
  - Exemple: Afficher un arbre.
- Il existe une terminologie standard pour qualifier les parcours.
  - On peut parcourir de gauche à droite, ou de droite à gauche.
  - Une fois ce choix fait, on distingue les parcours
    - en largeur.
    - en profondeur:
      - préfixe,
      - infixe,
      - suffixe.

Parcourir en largeur

- Parcours en largeur d'abord.
  - on parcourt par distance croissante à la racine.



- Si le traitement d'un sommet consiste à l'afficher, on affichera dans l'ordre 3, 1, 4, 5, 2, 0.
- C'est peut-être le parcours le plus naturel, mais c'est le plus délicat à programmer avec les arbres binaires.

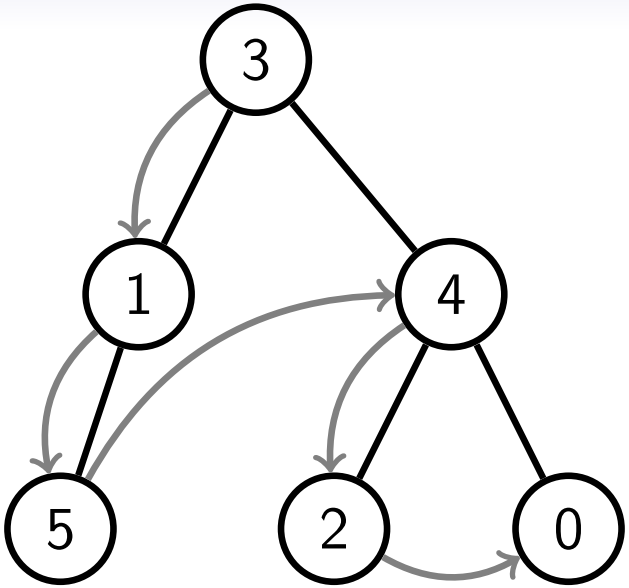
Parcourir en profondeur

- Parcours en profondeur d'abord:
  - on parcourt récursivement.

- Mais il reste trois possibilités

  - Préfixe: traiter la racine, parcourir le sous-arbre gauche, puis le sous-arbre droit.
  - Infixe: parcourir le sous-arbre gauche, traiter la racine, parcourir les sous-arbre droit.
  - Suffixe (appelé aussi postfixe): parcourir le sous-arbre gauche, le sous-arbre droit, puis traiter la racine.

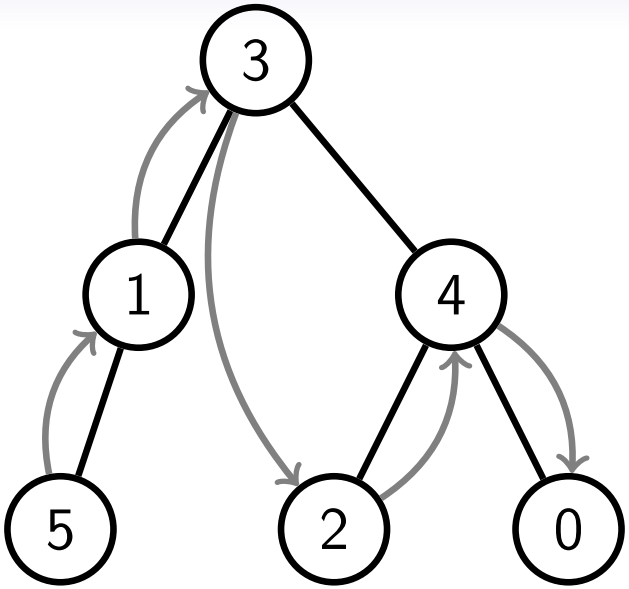
Parcours en profondeur: préfixe



- Si le traitement d'un sommet consiste à l'afficher, on affichera dans l'ordre 3, 1, 5, 4, 2, 0.

```
static void Affiche(Arbre a) {
   if (a==null) return;
   System.out.println(a.val+" ");
   Affiche(a.gauche);
   Affiche(a.droite);
 }
```

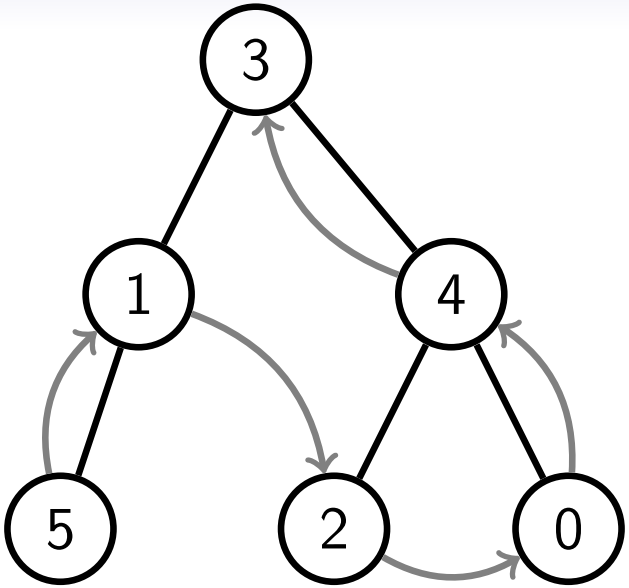
Parcours en profondeur: infixe



- Si le traitement d'un sommet consiste à l'afficher, on affichera dans l'ordre 5, 1, 3, 2, 4, 0.

```
static void Affiche(Arbre a) {
   if (a==null) return;
   Affiche(a.gauche);
   System.out.println(a.val+" ");
   Affiche(a.droite);
 }
```

Parcours en profondeur: postfixe



- Si le traitement d'un sommet consiste à l'afficher, on affichera dans l'ordre 5, 1, 2, 0, 4, 3.

```
static void Affiche(Arbre a) {
   if (a==null) return;
   Affiche(a.gauche);
   Affiche(a.droite);
   System.out.println(a.val+" ");
 }
```

Définir un type abstrait proprement

- On veut définir le type abstrait “sac”

```
class Sac  {
  ...
  Sac() { ... } //Construire un sac vide
  boolean EstVide() { ... } //Tester si un sac est vide
  void Ajouter(int e) { ... } //Ajouter un entier à un sac
  int Enlever() { ... } //Enlever un entier d'un sac
}
```

- Pour le faire proprement en JAVA, on peut définir

```
interface Sac  {
  boolean EstVide(); //Tester si un sac est vide
  void Ajouter(int e); //Ajouter un entier à un sac
  int Enlever(); //Enlever un entier d'un sac
}
```

Ce qui permet de l'utiliser

- A ce moment là, `sac` n'est pas une classe, mais on peut l'utiliser comme un type...

```
static void count(Sac s) {
  for (int i=0; !s.EstVide(); i++) {
    s.Enlever();
}}
```

- ... sans se préoccuper de l'implémentation.

Créer une implémentation

- On utilise le mot clé **implements**.

Implémentation 1:

```
class Pile implements Sac {

 private int[] t;
 private int sp;
 Pile() { t=new int[100]; sp=0; }

 public boolean EstVide() {
   return (sp ≤0); }
 int
Tete() {return t[sp-1];}

 public void Ajouter(int e) {
   t[sp++] = e;}

 public int Enlever() {
   return t[--sp];} }}
```

Implémentation 2:

```
class File implements Sac {

 private int[] t;
 private int in,out;
 File() { t=new int[100];
   in=out=0;}

 public boolean EstVide() {
   return (in == out); }

 public void Ajouter(int e) {
   t[out]=e; out=(out +1)% 100;}

 public int Enlever() {
   int c=t[in]; in=(in+1) % 100;
   return c;} }
```

Intérêts

- On peut travailler sans se préoccuper de l'implémentation.

  1. ce qui permet le travail en groupe, ou modulaire.
  2. et de remplacer une implémentation par une autre sans modifier le reste du programme.

- Le compilateur refuse de compiler si une des méthodes de l'**interface** est manquante,
  - et donc aide à vérifier que l'implémentation de chacune des fonctionnalités est bien présente.

- Note: Les méthodes déclarées dans l'interface doivent être qualifiées de **public**.

De l'itératif? Parcourir en largeur?

- Voici un algorithme générique itératif de parcours d'arbre.

```
static void parcours(Arbre a, Sac s) {
   s.Ajouter(a);       //  encore à traiter
   while (!s.EstVide()) {
     Arbre t = s.Enlever(); // On traite un sommet
     if (t ≠ null) {
       System.out.println(t.val);
       s.Ajouter(t.gauche); // Et on stocke le travail futur
       s.Ajouter(t.droite);
     }}}
```

- Si le sac s est implémenté
  - par une pile, on parcourt en profondeur (préfixe).
  - par une file, on parcourt en largeur.