En revenant au cas précédent

Jean-Jacques.Levy@inria.fr

http://para.inria.fr/~levy

tel: 01 39 63 56 89

Catherine Bensoussan

cb@lix.polytechnique.fr

Aile 00, LIX

tel: 34 67

http://w3.edu.polytechnique.fr/informatique/

Plan

1. **Fonctions récursives numériques**
2. **Procédures récursives**
3. **Récursivité et raisonnement inductif**
4. *QuickSort*
5. **Fractales**
6. *Splines*

Fonction numérique récursive

Une fonction peut s'appeler elle-même à l'intérieur de sa définition.

Identique aux définitions par récurrence en math ou encore au raisonnement par induction.

Exemple:

$$u_0 = u_1 = 1$$
$$u_n = u_{n-1} + u_{n-2}$$

pour la suite de Fibonnacci

```
static int fib (int n) {

  if (n <= 1)
      return 1;
  else
      return fib (n-1) + fib (n-2);
}
```

Calcul récursif de Fibonacci

```
fib(4) -> fib (3) + fib (2)
       -> (fib (2) + fib (1)) + fib (2)
       -> ((fib (1) + fib (1)) + fib (1)) + fib(2)
       -> ((1 + fib(1)) + fib (1)) + fib(2)
       -> ((1 + 1) + fib (1)) + fib(2)
       -> (2 + fib(1)) + fib(2)
       -> (2 + 1) + fib(2)
       -> 3 + fib(2)
       -> 3 + (fib (1) + fib (1))
       -> 3 + (1 + fib(1))
       -> 3 + (1 + 1)
       -> 3 + 2
       -> 5
```

Fonctions récursives: théorie inventée par Kleene [1909–1994]

Autres exemples

```
static int fact(int n)  {

  if (n <= 1)
      return 1;
  else
      return n * fact (n-1);
}

static int C(int n, int p)  {

  if ((n == 0) || (p == n))
      return 1;
  else
      return C(n-1, p-1) + C(n-1, p);
}
```

Gros calculs – Fonction d'Ackermann

```
static int ack(int m, int n) {

  if (m == 0)
      return n + 1;
  else
      if (n == 0)
          return ack (m - 1, 1);
      else
          return ack (m - 1, ack (m, n - 1));
}
```

Pourquoi cette fonction termine-t-elle (très lentement) pour tout m, n ?

Trouver les valeurs de `ack(0,n)`, `ack(1,n)`, `ack(2,n)`, ...

Récursivité imbriquée

```
static int f(int n)  {

  if (n > 100)
      return n - 10;
  else
      return f(f(n+11));
}

static int g(int m, int n) {

  if (m == 0)
      return 1;
  else
      return g(m - 1, g(m, n));
}
```

Quelle est la valeur de f? et de g?

Indication: se souvenir de l'appel par valeur.

Récursivité imbriquée

```
static int f(int n)  {

  if (n > 100)
      return n - 10;
  else
      return f(f(n+11));
}

static int g(int m, int n) {

  if (m == 0)
      return 1;
  else
      return g(m - 1, g(m, n));
}
```

$f(94) = f(f(105)) = f(95) = f(f(106)) = f(96) \ldots = f(101) = 91$
et donc $f(n) = 91$ **si** $n \leq 100$**, sinon** $f(n) = n - 10$

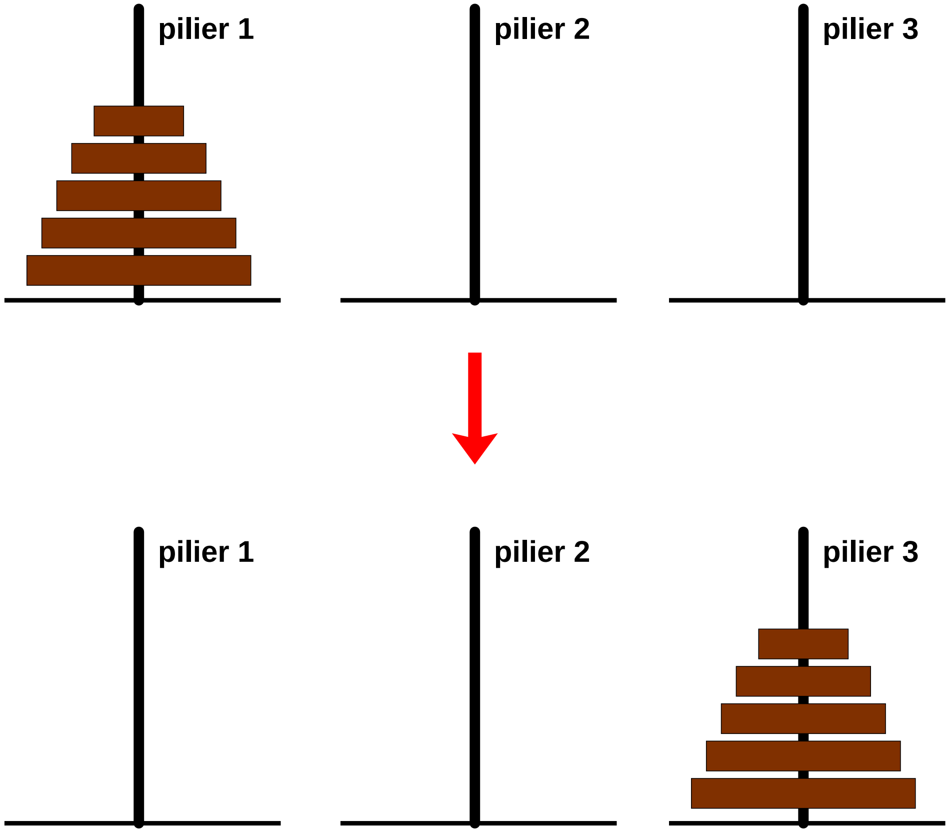
Récursivité imbriquée

```
static int f(int n)  {

  if (n > 100)
      return n - 10;
  else
      return f(f(n+11));
}

static int g(int m, int n) {

  if (m == 0)
      return 1;
  else
      return g(m - 1, g(m, n));
}
```

$f(94) = f(f(105)) = f(95) = f(f(106)) = f(96) \ldots = f(101) = 91$
et donc $f(n) = 91$ **si** $n \leq 100$**, sinon** $f(n) = n - 10$

$g(1,0) = g(0, g(1,0)) = g(0, g(0, g(1,0))) = \ldots$
$g(0,n) = 1$**,** $g(m,n)$ **indéfini quand** $m > 0$

Les tours de Hanoi

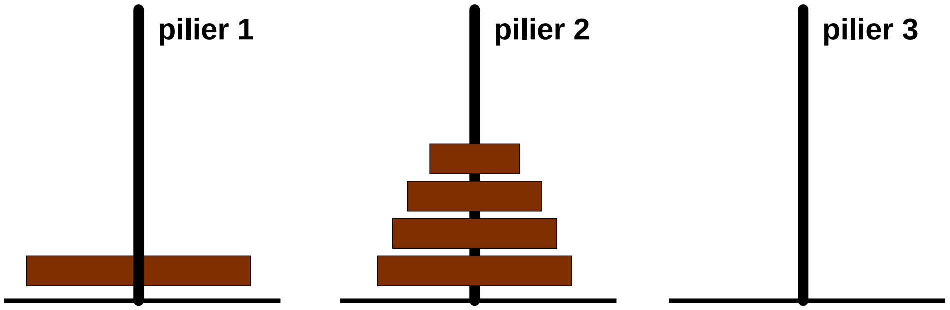


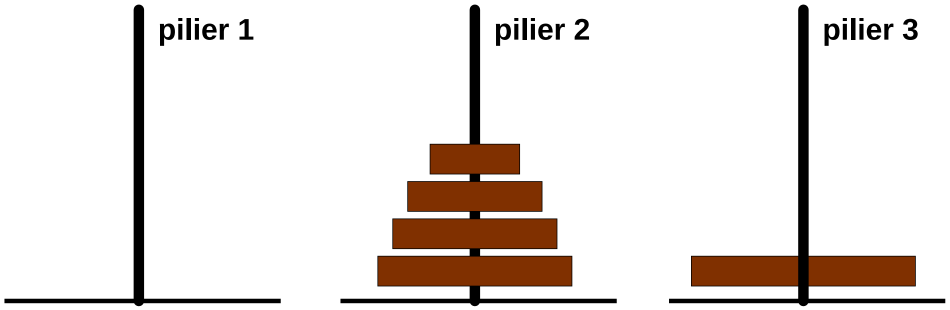
Règle du jeu

On déplace une rondelle à la fois.

Une rondelle ne doit jamais se trouver sous une rondelle plus grosse qu'elle-même.

Les tours de Hanoi – positions intermédiaires

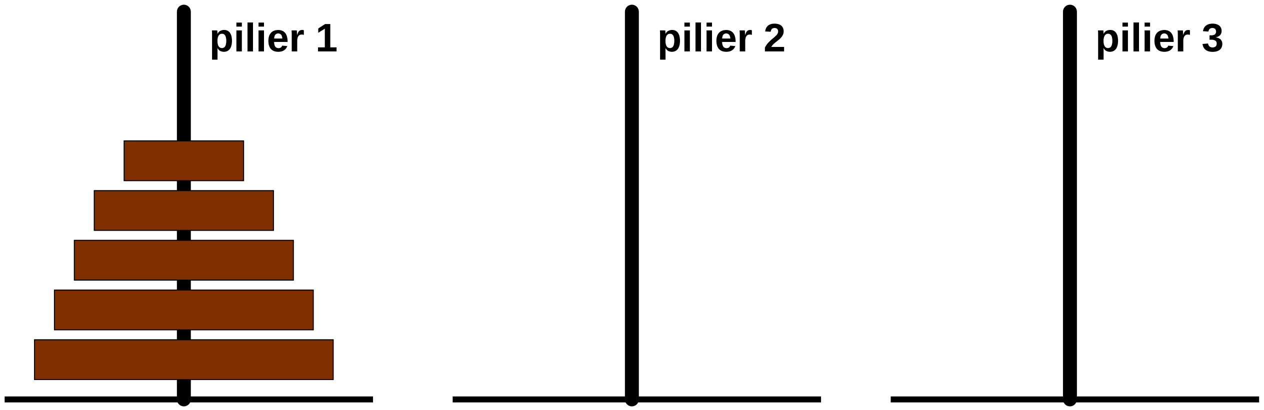




Les tours de Hanoi

C'est l'exemple du raisonnement inductif dépassant le cas des récurrences numériques.

```
static void hanoi(int n, int i, int j)  {

  if (n > 0) {
      hanoi (n-1, i, 6-(i+j));
      System.out.println (i + " -> " + j);
      hanoi (n-1, 6-(i+j), j);
  }
}
```

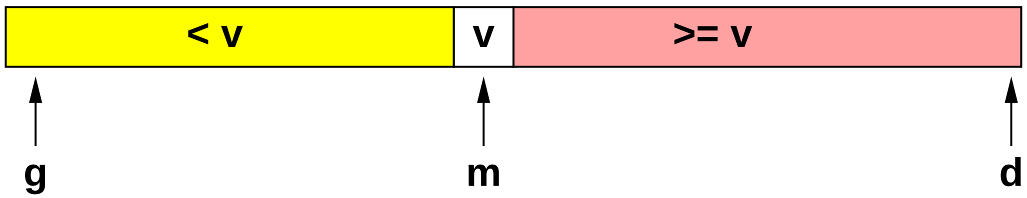


hanoi (5, 1, 3)

Exercice: faire les tours de Hanoi avec un programme itératif!

QuickSort [Hoare 1960]

```
static void qSort(int g, int d)  {

  if (g < d) {
      v = a[g];
      Partitionner le tableau autour de la valeur v
      et mettre v à sa bonne position m
      qSort (g, m-1);
      qSort (m+1, d);
  }
}
```

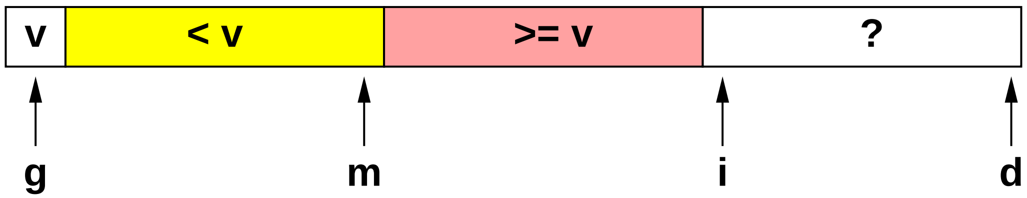


Complexité: en moyenne, $C_n \simeq 1,38\ n \log n$**, très bon.**

$O(n^2)$ **dans le pire cas.**

QuickSort – suite

```
static void qSort(int g, int d)  {

  int i, m, x, v;

  if (g < d) {
      v = a[g];
      m = g;
      for (i = g+1; i <= d; ++i)
          if (a[i] < v) {
              ++m;
              x = a[m]; a[m] = a[i]; a[i] = x;
          }
      x = a[m]; a[m] = a[g]; a[g] = x;
      qSort (g, m-1);
      qSort (m+1, d);
  }
}
```



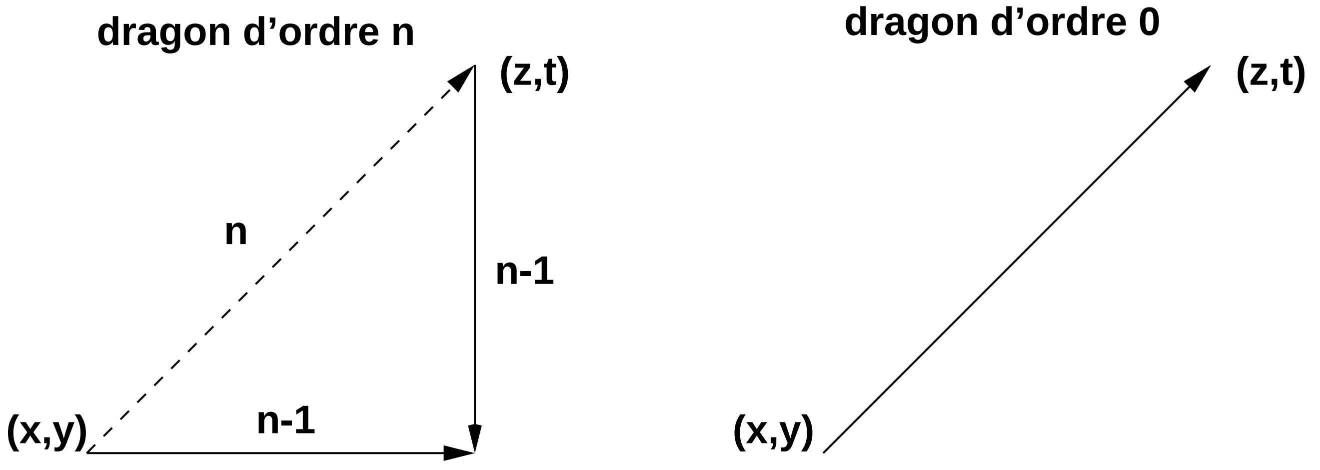
Récursivité graphique, courbes fractales

Gaston Julia et Benoît Mandelbrot en sont les pères fondateurs.

Le flocon de von Koch [1]
La courbe du dragon [1, 2]
La courbe de Hilbert [1]
La courbe de Sierpinsky [1 2 3 4]
Les fougères de Barnsley [1]
Les systèmes de Lindenmayer (arbres) [1]

La courbe du dragon

```
static void dragon(int n, int x, int y, int z, int t) {
  if (n == 1) {
      moveTo (x, y);
      lineTo (z, t);
  } else {
      int u = (x + z + t - y) / 2;
      int v = (y + t - z + x) / 2;
      dragon (n-1, x, y, u, v);
      dragon (n-1, z, t, u, v);
  }
}
```



La courbe du dragon – sans lever le crayonn

```
static void dragon (int n, int x, int y, int z, int t)  {
  if (n == 1) {
      moveTo (x, y);
      lineTo (z, t);
  } else {
      int u = (x + z + t - y) / 2;
      int v = (y + t - z + x) / 2;
      dragon (n-1, x, y, u, v);
      dragonBis (n-1, u, v, z, t);
  }
}
```

La courbe du dragon – sans lever le crayonn

```
static void dragonBis(int n, int x, int y, int z, int t)  {
  if (n == 1) {
      moveTo (x, y);
      lineTo (z, t);
  } else {
      int u = (x + z - t + y) / 2;
      int v = (y + t + z - x) / 2;
      dragon (n-1, x, y, u, v);
      dragonBis (n-1, u, v, z, t);
  }
}
```

Autre exemple de tracé récursif: *splines*

Interpolation entre plusieurs points (coniques, cubiques, etc). Les cubiques sont les plus utilisées en CAO, en graphique interactif, pour générer des polices de caractères (PostScript, Metafont). Les plus simples à décrire sont les courbes paramétriques de Bézier (ingénieur à Renault) et de de Casteljau.

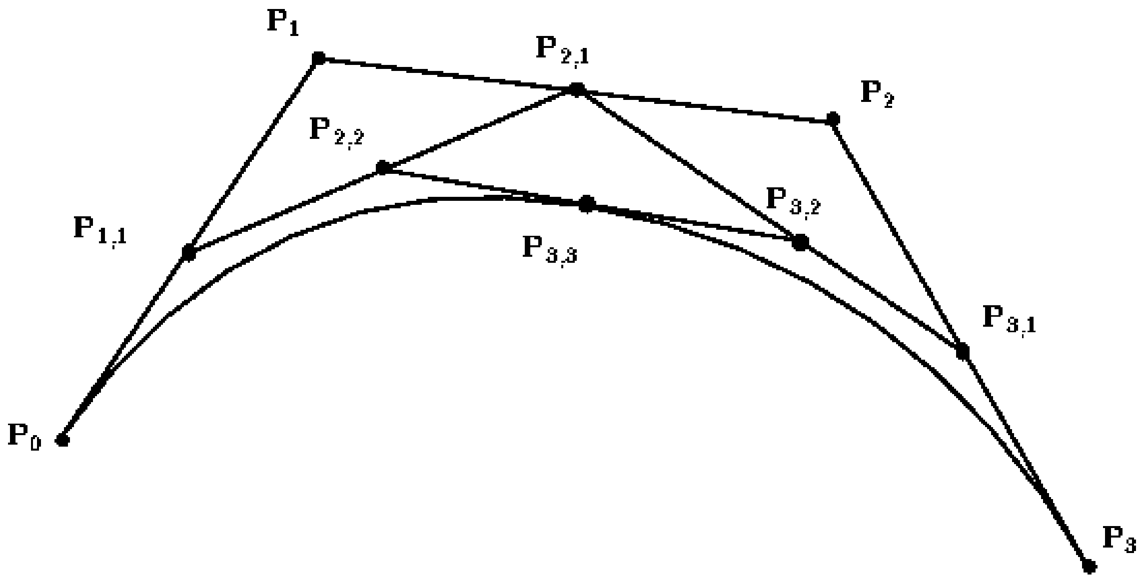
Les cubiques de Bézier (splines) sont données par 4 points, P_0, P_1, P_2, P_3. La cubique est tangente en P_0 et P_3, les vecteurs P_0P_1 et P_2P_3 représentent les valeurs des dérivées en P_0 et P_3. La cubique est toujours inscrite dans le quadrilatère $P_0P_1P_2P_3$.

On les dessine facilement avec une méthode Diviser pour Régner, car on trouve une définition récursive, en prenant les milieux:

$P_{1,1}$ de [P_0P_1], $P_{2,1}$ de [P_1P_2], $P_{3,1}$ de [P_2P_3], $P_{2,2}$ de [$P_{1,1}P_{2,1}$], $P_{3,2}$ de [$P_{2,1}P_{3,1}$], $P_{3,3}$ de [$P_{2,2}P_{3,2}$]

et en considérant les courbes de Bézier pour $P_0P_{1,1}P_{2,2}P_{3,3}$ et $P_{3,3}P_{3,2}P_{3,1}P_3$. (Il suffit de tracer le segment [P_0P_3] pour un quadrilatère de petite surface).

Splines – bis



Un expert est Lyle Ramshaw (DEC/SRC Tech Report 19). On en parle au cours Graphique de Sillion dans la majeure Math et Info, M1, en 2ème année.

Ce n'est qu'un au revoir.

A l'année prochaine!

Continuez à apprendre l'informatique. Travaillez plus tard dans l'informatique. Bonnes vacances à tous!