

TC Informatique

PC N° 3
23 Novembre 2000

François Sillion

Récurtivité

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

Récurtivité

- La récurrence est classique en maths:

$$[P(0) \quad n, P(n) \quad P(n+1)] \quad n, P(n)$$

- En informatique, on utilise le même principe mais à l'envers.

```
public static int Factorielle (int n)
{
    if ( n == 0 ) { // test d'arrêt
        return 1; // cas de base ou terminal
    } else { // cas récursif
        return n * Factorielle(n-1);
    }
}
```

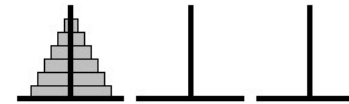
Fibonacci

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

```
public static int fibonacci (int n)
{
    if ( n < 2 ) {
        return 1;
    } else {
        return fibonacci(n-1) + fibonacci(n-2);
    }
}
```

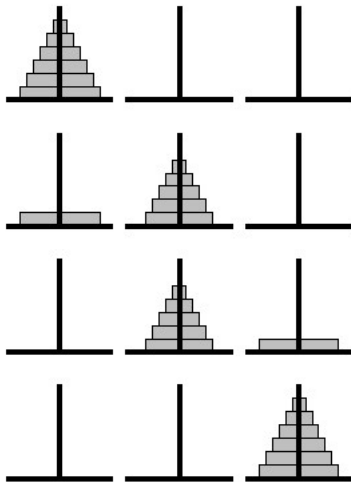
Nombre d'appels ?

Procédures récursives : Tours de hanoi



```
static void Hanoi(int n, int i, int j, int k)
{
    /*
     * Bouge n disques de la pile i ` a la pile k
     * en passant par la pile j
     */
    if (n > 0) {
        Hanoi(n - 1, i, k, j);
        System.out.println(i + " -> " + k);
        Hanoi(n - 1, j, i, k);
    }
}
```

Procédures récursives : Tours de hanoi



Propriétés de la récursivité

- On peut toujours "dérécursiver" un programme récursif (et certains langages n'autorisent pas la récursivité)
- Le coût de la récursivité est souvent caché (nombre d'appels)
- Cas particulier : récursivité terminale
- Indécidabilité du problème de l'arrêt

Indécidabilité

```
public static boolean Termine ( Fonction f )
{
    // retourne vrai ou faux en temps fini
    // selon que f termine ou non
}
public static boolean bizarre()
{
    while ( Termine( bizarre ) ) {
    }
}
```

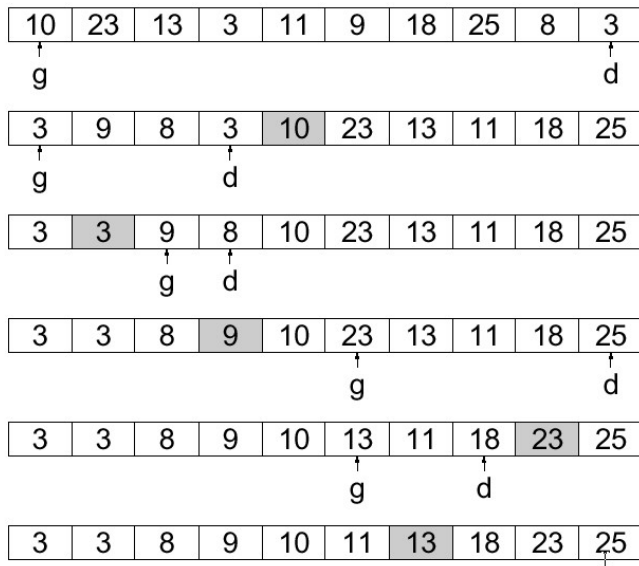
Que renvoie Termine(bizarre) ???
(Gödel, Turing)

Quicksort (Hoare, 1960)

- Paradigme de la méthode “Diviser pour régner” (Divide and Conquer)
- Principe de l’algorithme
 - Partitionne le tableau en fonction d'un de ses éléments
 - À gauche les éléments plus petits, à droite les plus grands
 - L’élément choisi est à sa place
 - On recommence sur les parties droite et gauche

```
static void QuickSort(int g, int d)
{
    if (g < d) {
        int i = Partition(g, d);
        QuickSort (g, i - 1);
        QuickSort (i + 1, d);
    }
}
```

Quicksort en action



```
static void QuickSort(int g, int d)
{
    if (g < d) {
        int v = a[d];
        int i = g - 1;
        int j = d;
        int t;
        do {
            do i = i + 1; while (a[i] < v);
            do j = j - 1; while (a[j] > v);
            t = a[i]; a[i] = a[j]; a[j] = t;
        } while (i < j);
        a[j] = a[i]; a[i] = a[d]; a[d] = t;
        QuickSort (g, i - 1);
        QuickSort (i + 1, d);
    }
}
```

Nécessite une sentinelle à gauche.

Complexité de Quicksort

- Exprimée en nombre de comparaisons
- Le meilleur cas :

$$C_N = 2C_{N/2} + N = O(N \log_2 N)$$

Profondeur de récursion : $\log_2 N$

- Le cas le pire (tableau déjà trié) :

$$C_N = C_{N-1} + N = O(N^2)$$

Profondeur de récursion : N

Quicksort : Complexité en moyenne

$$C_N = N + 1 + \frac{1}{N} \sum_{1 \leq k \leq N} (C_{k-1} + C_{N-k})$$

$$C_N = N + 1 + \frac{2}{N} \sum_{1 \leq k \leq N} C_{k-1}$$

$$NC_N - (N-1)C_{N-1} = N(N+1) - (N-1)N + 2C_{N-1}$$

$$NC_N = (N+1)C_{N-1} + 2N$$

$$\frac{C_N}{N+1} = \frac{C_{N-1}}{N} + \frac{2}{N+1} = \frac{C_2}{3} + \sum_{3 \leq k \leq N} \frac{2}{k+1}$$

$$\frac{C_N}{N+1} \simeq 2 \sum_{1 \leq k \leq N} \frac{1}{k} \simeq 2 \int_1^N \frac{1}{x} dx = 2 \ln N$$

$$C_N = 1.38N \log_2 N$$

Quicksort : améliorations

- “Randomisation”.
- Prendre comme pivot le médian des éléments $a[g]$, $a[(g+d)/2]$ et $a[d]$, préalablement triés (plus besoin de sentinelles).
- Pour de petits tableaux, faire un tri par insertion
(arrêter QuickSort dès que $d-g < M$).

Tri fusion

```
static void TriFusion(int[] a, int[] b,
                    int g, int d)
{
    if (g >= d)
        return;
    // Trie séparément les deux moitiés du tableau
    int m = (g + d) / 2;
    TriFusion(a, b, g, m);
    TriFusion(a, b, m + 1, d);

    // Combine les deux moitiés du tableau...
    ...
}

static void Tri(int[] a, int n) {
    int[] b = new int[n];
    TriFusion(a, b, 0, n - 1);
}
```

Interclassement de deux tableaux

```
public static int[] interclasser(int a[], int b[])
{
    int[] result = new int[a.length+b.length];
    int i = 0, j = 0, k = 0;
    while ( k < a.length+b.length ) {
        if ( i < a.length
            && ( ( j < b.length && a[i] <= b[j] )
                || ( j >= b.length) ) ) {
            result[k] = a[i]; i++;
        } else {
            result[k] = b[j]; j++;
        }
        k++;
    }
    return result;
}
```

Tri fusion avec retournement

```
// Recopie les moitiés triées dans b
// en renversant la moitié droite

for (int i = g; i <= m; i++)
    b[i] = a[i];
for (int i = m + 1; i <= d; i++)
    b[d + m + 1 - i] = a[i];

// Fusion des deux moitiés triées dans a
int gg = g;
int dd = d;
for (int i = g; i <= d; i++)
    if (b[gg] < b[dd]) {
        a[i] = b[gg];
        gg++;
    } else {
        a[i] = b[dd];
        dd--;
    }
}
```

Complexité du tri fusion

- Même dans le cas le pire !

$$C_N = 2C_{N/2} + N = O(N \log_2 N)$$