Tronc Commun Informatique

Dominique Perrin

X & Université de Marne la Vallée
perrin@univ-mlv.fr

Menu

1. Amphi 0
2. 10 petites classes
3. 10 t.p.
4. une composition HC
5. une composition
6. un projet

Bibliographie

Danièle Beauquier, Jean Berstel, Philippe Chretienne, *Elements d'Algorithmique*, Masson, 1992.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, *Algorithms*, MIT Press, 1990.

Donald E. Knuth, *The Art of Computer programming, vol. 1: Fundamental Algorithms, vol. 2: Seminumerical Algorithms, vol. 3: Sorting and Searching*, Addison Wesley, 1973.

Jan van Leeuwen,*Handbook of Theoretical Computer Science*, vols. A et B, Elsevier, 1990.

Robert Sedgewick, *Algorithms in C*, Addison Wesley, 1991.

P.C. 1
Introduction

Objectifs généraux et orientations de l'algorithmique:

- Conception de stuctures de données
- Construction d'algorithmes simples et efficaces
- Evaluation de la complexité des problèmes

Ecriture des algorithmes:

- en pseudocode
- dans un langage de programmation (ici: langage C)

Un exemple : le calcul du pgcd

Le calcul du pgcd de deux entiers m, n par l'algorithme d'Euclide repose sur la division entière de n par m comme $n = mq + r$ et la règle facile à vérifier

$$\mathrm{pgcd}(m, n) = \mathrm{pgcd}(r, m)$$

On en déduit directement le programme

```
int pgcd(int m, int n) {
  if (m == 0) return n;
  else return pgcd(n%m, m);
}
```

ou encore, sans appel récursif

```
int pgcd(int m, int n) {
  int r;
  while (m != 0) {
    r= n%m; m=n; n=r;
  }
  return n;
}
```

Un autre exemple : le développement en fraction continue

Un développement en *fraction continue* d'un nombre réel positif α est une suite $(a_0, a_1, \ldots)$ d'entiers naturels telle que:

$$\alpha = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots}}}$$

Quand α est un nombre rationnel m/n, la suite des quotients obtenus en appliquant l'algorithme d'Euclide à (m, n) est un développement en fraction continue de α.

En effet si $m = qn + r$, on a

$$\frac{m}{n} = q + 1/(\frac{n}{r})$$

On en déduit la procédure suivante, en C, qui affiche les termes successifs du développement :

```
void cont(int m, int n)
{
  if (n != 0)
  {
    printf("%d\n", m / n);
    cont( n, m % n);
   }
 }
```

Pour le développement d'un nombre réel α, on aura :

```
void cont(float alpha, int n)
{
  int a,i;

  for (i = 1; i<= n; i++)
  {
   a =  alpha;
   printf("%d\n",a);
   alpha = 1/(alpha-a);
  }
}
```

Le résultat de l'appel de `cont(3.14159,4)` est

```
3
7
15
1
```

Complexité des algorithmes

Donnée de taille n

- Temps de calcul $T(n)$
- Espace mémoire utilisé $E(n)$

Evaluation:

- En moyenne
- Au pire des cas

Calcul à une constante multiplicative près: pas d'unité de temps ou d'espace.

Notation O

Pour deux fonctions réelles positives f, g, on écrit

$$f = O(g)$$

si f ne croit pas plus vite que g; formellement:

$$\begin{array}{rcl} \exists m \geq 0, \exists k & \geq & 0, \forall n \geq m \\ 0 \leq f(n) & \leq & kg(n) \end{array}$$

Exemples de classes de complexité:

$T = O(1)$	: temps constant
$T = O(\log n)$	: temps logarithmique
$T = O(n)$	: temps linéaire
$T = O(n \log n)$	
$T = O(n^2)$	: temps quadratique

Autres notations utilisées:
$f = \Omega(g)$ pour $g = O(f)$
$f = \Theta(g)$ pour $f = O(g)$ et $g = O(f)$.
La notation $f = O(g)$ est un abus désigant $f \in O(g)$ mais commode pour écrire

$$f = g + O(h)$$

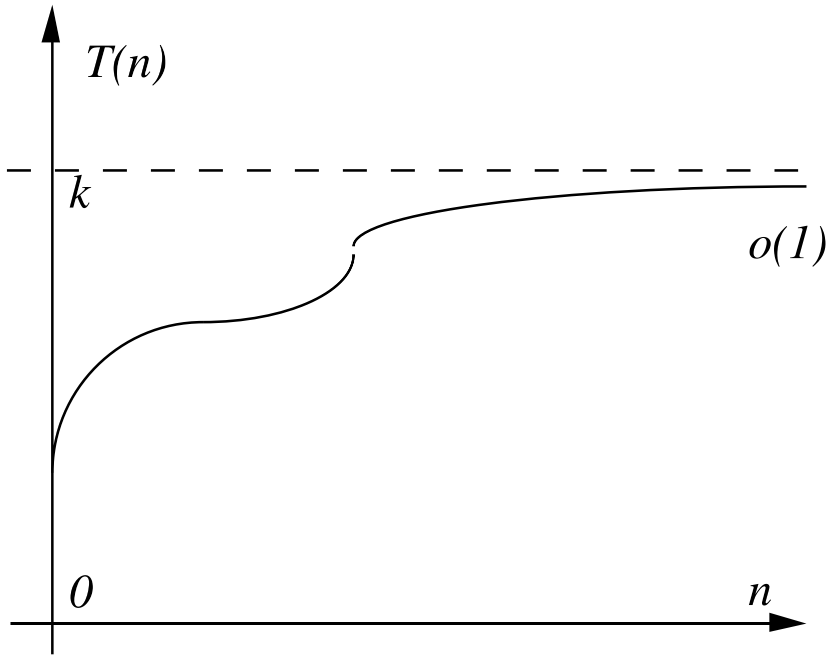

Figure 1:

Calcul de complexité

- Concaténation d'instructions:

$$T(P;Q) = T(P) + T(Q)$$

- Itération:

$$T(\text{pour } i = 1 \text{ à } n \text{ faire } P(i)) = \sum_{i=1}^{n} T(P(i))$$

- Règles de calcul:

  Si $f = O(g)$ et $f' = O(g')$ alors

$$f + f' = O(g + g'), ff' = O(gg')$$

  Si $f = O(n^k)$ et si $h = \sum_{i=0}^{n} f(i)$ alors

$$h = O(n^{k+1})$$

Dichotomies

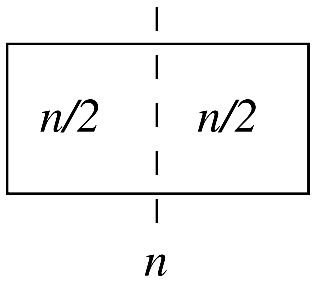

Figure 2:

$$T(n) \leq T(n/2) + b$$

Solution:

$$\boxed{T(n) = O(\log n)}$$

Démonstration : il suffit de vérifier par récurrence sur n que

$$T(n) \leq a \log n + b$$

avec $a = T(0)$.

Un exemple : le calcul du pgcd

Le calcul du pgcd de m et n par l'algorithme d'Euclide obéit, en posant $m = nq + r$, à la règle :

$$\text{pgcd}(m, n) = \text{pgcd}(n, r)$$

La complexité est logarithmique. En effet, on a, en supposant $0 \leq n \leq m$ les inégalités

$$2r \leq r + n \leq m$$

ou encore

$$2r \leq m$$

d'où la conclusion.

Complexité en $n \log n$

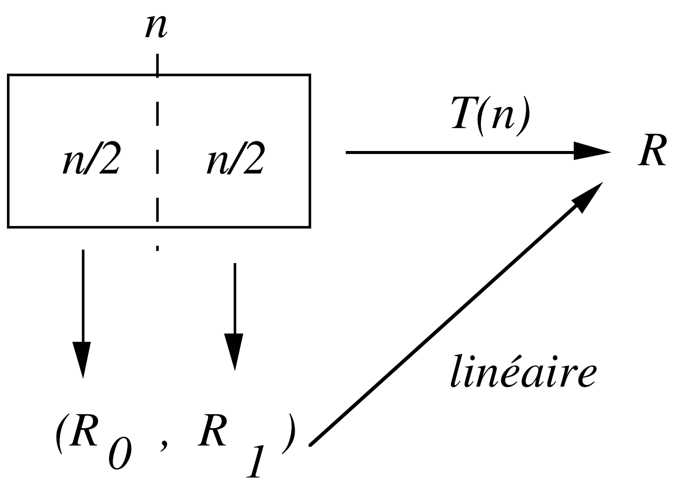

Figure 3:

$$T(n) \leq 2T(n/2) + bn$$

Solution: $\boxed{T(n) = O(n \log n)}$

Démonstration : il suffit de diviser les deux membres par n pour conclure que $T(n)/n = O(\log n)$.

Exemple : tri par fusion

On considère le problème du tri d'un tableau $a[0], \ldots, a[n-1]$. Le principe utilisé est de trier séparément les deux moitiés puis de les fusionner.

```
void TriFusion(int i, int j)
{
 int k;

  if (i<j)
  {
    k =(i+j) / 2;
    TriFusion(i,k);
    TriFusion(k+1,j);
    Fusion(i,j);
  }
}
```

La fusion des deux moitiés triées utilise un tableau auxiliaire b.

```
void Fusion(int g,int d)
{
 int b[N];
 int i,j,m;

 m=(g+d)/2;
 for(i=m; i>=g; --i)
   b[i]=a[i];
 for (j=m+1;j<=d;  ++j)
   b[d+m+1-j]=a[j];
 i=g;j=d;
 for (k=g; k<=d;++k)
   if (b[i]< b[j]){
     a[k]=b[i]; ++i;
   } else {
     a[k]=b[j];--j;
   }
}
```