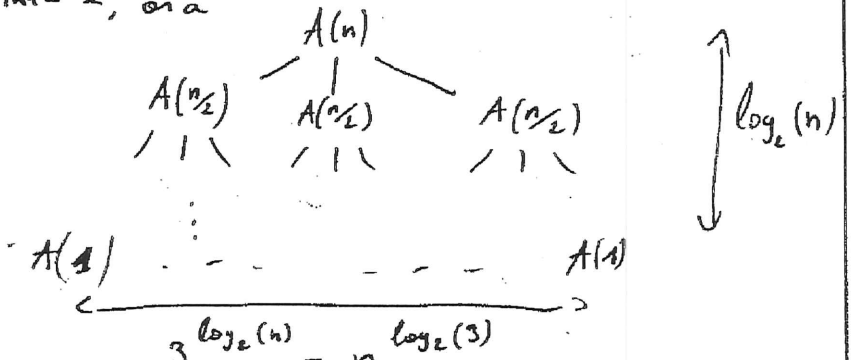
DIVISER POUR REGNER

I Le principe de la méthode

Il s'agit d'une méthode récursive pour donner une solution à un problème, en trois étapes

1. On divise le problème en sous problèmes de tailles inférieures
2. On traite récursivement les sous problème, et on "règne" directement sur les cas de base
3. On combine de manière appropriée les résultats

Remarque: La structure d'exécution de l'algorithme est un arbre, par exemple dans le cas d'une division en 3 sous problèmes de taille 2, on a



Exemple d'algorithme : recherche d'un element dans un arbre

```
appartient (x, T)
    Si T = feuille(y)
        renvoyer oui ssi y = x
    Sinon
        ∨ appartient(x, T')
          T' fils de T
```

II Correction d'algorithme

Toutes les preuves de correction se font par induction sur l'arbre de calcul.

Exemple d'algorithme: Recherche de distance minimale dans un ensemble de points

```
PPP ( A [(x_1,y_1), ... (x_n,y_n)] )
    Si n ≤ 3                                  (n ≥ 2)
        d = min ( |(x_1,y_1) - (x_2,y_2)|, |(x_1,y_1) - (x_3,y_3)|, |(x_2,y_2) - (x_3,y_3)| )
    Sinon
        Séparer A en A_G et A_D, de même taille et tels que les points de A_G soient à gauche des points de A_D
        Soit d = inf ( PPP(A_G), PPP(A_D) )
        Pour (x_i,y_i) ∈ A_D, (x_j,y_j) ∈ A_G
            d = min ( d, |(x_i,y_i) - (x_j,y_j)| )
        Retourner d
```

Preuve de correction

Cas $n \leq 3$: la distance est bien calculée.

Autres cas : • En divisant, on se retrouve avec au moins 2 points par ensembles
• La distance sur ces sous ensembles est calculée
• La distance minimale est atteinte, soit dans les parties droite et gauche, soit en chevauchement.

La correction des algorithmes de type diviser pour régner est assez simple généralement, l'étude de la complexité est des plus intéressants.

III Calcul de complexité

1) Un exemple : multiplication d'entiers

Soient x, y deux entiers codés sur $n = 2^u$ bits

$$\begin{cases} x = 2^{n/2} x_G + x_D \\ y = 2^{n/2} y_G + y_D \end{cases}$$ où x_G, y_G, x_D, y_D sont codés sur 2^{u-1} bits

Alors $x \times y = 2^n x_G y_G + 2^{n/2}(x_G y_D + x_D y_G) + x_G y_G$

Les additions sont de complexité linéaire

$$T(n) \geq 4\,T(n/2) + O(n)$$

$$\Rightarrow \frac{T(n)}{n^2} = \frac{T(n/2)}{(n/2)^2} + O\left(\frac{1}{n}\right) \Rightarrow \underline{T(n) = O(n^2)}$$

Amélioration : $x_G y_D + x_D y_G = (x_D + x_G)(y_G + y_D) - x_G y_G - x_D y_D$

On peut économiser une multiplication en ajoutant des additions : $T'(n) = 3\,T'(n/2) + O(n)$

$$\Rightarrow \frac{T'(n)}{n^{\log_2(3)}} = \frac{T'(n/2)}{(n/2)^{\log_2(3)}} + O\left(n^{1-\log_2(3)}\right)$$

$$\Rightarrow T'(n) = O\left(n^{\log_2(3)}\right)$$

On retrouve cette idée par la multiplication de matrices

2) Théorème général

Soit $a \geq 1$, $b > 1$ et $T(n)$ définit par la relation de récurrence $\begin{cases} T(n) = a\,T(\lfloor n/b \rfloor) + f(n) \\ T(1) = 1 \end{cases}$

et si $f(n) = O(n^d)$ Alors

$$T(n) = \begin{cases} O(n^d) & \text{si } d > \log_b(a) \\ O(n^d \log_2(n)) & \text{si } d = \log_b(a) \\ O(n^{\log_b(a)}) & \text{si } d < \log_b(a) \end{cases}$$

remarque L'algorithme par la multiplication hon d'entiers est un exemple du troisième cas.

3) Application du deuxième cas : Algorithme du Tri fusion

Tri fusion $a[1, \ldots n]$

- Si $n \geq 1$
  - retourner fusion (tri fusion $a[1 \ldots \lfloor n/2 \rfloor]$, tri fusion $a[\lfloor n/2 \rfloor + 1, \ldots n]$)
- Sinon
  - retourner a

avec **fusion** $(a[1, \ldots u], b[1, \ldots l])$

- Si $u = 0$, retourner b
- si $l = 0$ retourner a
- Sinon
  - Si $a[1] \leq b[1]$
    - retourner $a[1] \circ$ fusion $(a[2, \ldots u], b[1 \ldots l])$
  - Sinon
    - retourner $b[1] \circ$ fusion $(a[1 \ldots u], b[2, \ldots l])$

- fusion s'exécute en $O(n)$

$$T(n) = 2\,T(n/2) + O(n) \quad \text{or} \quad 1 = \log_2(2)$$

$$\Rightarrow T(n) = O(n \log n)$$

On peut adapter le tri fusion à d'autres modèles de calcul : **TRI POLYPHASÉ** (Développement)

4) Division en sous problèmes de taille aléatoire

L'algorithme suivant calcule le k-ième élément par ordre croissant dans une liste non triée

Recherche (k, $S[1,\dots n]$)

pivot = $S[1]$

On sépare S en S_e : éléments égaux au pivot
S_i : éléments inférieurs au pivot
S_s : éléments supérieurs au pivot. } $O(n)$

Si $k \leq |S_i|$, Recherche (k, S_i)

Si $k > |S_i| + |S_e|$ Recherche $(k - |S_i| - |S_e|, S_s)$

Sinon retourne pivot

Complexité : Dans le pire des cas, on enlève un élément à chaque fois : $n + (n-1) + \dots + 1 = O(n^2)$

en moyenne, Si le pivot est dans le 2ème ou 3ème quart du tableau (50% des cas), on réduit le tableau d'au moins 3/4

L'espérance du nb d'itérations avant de trouver un élément dans le 2ème ou 3ème quart du tableau est 2.

Donc $E[T(n)] \leq E\left[T\left(\frac{3n}{4}\right)\right] + O(n) + O(n)$

$\log_{\frac{4}{3}}(1) < 1 \Rightarrow E[T(n)] = O(n)$

IV Applications en analyse numérique

1) Transformée de Fourier rapide

La transformée de Fourier rapide permet la multiplication de Polynômes de degré $\leq n$ en temps $O(n \log n)$ plutôt que $O(n^2)$ (Développement)

2) Multiplications de matrices

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$AB = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{21}B_{11} + A_{22}B_{21} \\ A_{11}B_{12} + A_{12}B_{22} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

On a $T(n) = 8T(n/2) + O(n^2) \Rightarrow T(n) = O(n^{\log_2(8)}) = O(n^3)$

L'algorithme de Strassen propose une méthode pour avoir à effectuer plus d'additions mais moins de multiplications de sous matrices (7) à la manière de la multiplication de grands entiers - On obtient une complexité

$$T(n) = 7T(n/2) + O(n^2) \Rightarrow T(n) = O(n^{\log_2(7)}).$$

De meilleurs algorithmes existent, avec des complexités jusqu'à $O(n^{2,376})$, en découpant la matrice en plus de sous problèmes plus petits.

Références :

- Algorithmique, Cormen, Leiserson, Rivest, Stein DUNOD
- Algorithmes, Dasgupta, Papadimitriou, Vazirani, McGraw-Hill Higher Ed.