Méthode de programmation

Partie II : diviser pour régner

Table des matières

I Principes généraux

Pour résoudre un problème de « taille » N, un algorithme récursif naïf fonctionne en général de la façon suivante :

- Extraire ou construire à partir de notre entrée un problème de taille $N-1$.
- Résoudre ce problème de taille $N-1$ (récursivité).
- Utiliser cette solution pour former la solution du problème de taille N.

La résolution du problème de taille N nécessite alors l'exécution de N étages de récursivité.

La méthode **diviser pour régner** consiste, pour résoudre un problème de taille N, à :

- Partager le problème en sous-problèmes de taille $\left\lfloor \frac{N}{2} \right\rfloor$ ou $\left\lceil \frac{N}{2} \right\rceil$.
- Résoudre ces différents sous-problèmes (généralement récursivement).
- Fusionner les solutions pour former la solution du problème de taille N.

On obtient beaucoup moins d'étages de récursivité : $\log_2 N = \dfrac{\ln N}{\ln 2}$, mais ces étages sont plus compliqués. Dans certains cas, la méthode **diviser pour régner** donne un algorithme de résolution plus rapide.

II Ordres de grandeurs

Proposition II - 1 : Relations de comparaisons sur les séries usuelles

Pour tous $q > 1$, $\alpha > 0$ et $\beta > 0$,

$$\sum_{k=0}^{n} q^k = \Theta\left(q^{n+1}\right), \quad \sum_{k=0}^{n} k^\alpha = \Theta\left(n^{\alpha+1}\right),$$

$$\sum_{k=1}^{n} k^\alpha \left(\ln(k)\right)^\beta = \Theta\left(n^{\alpha+1}\left(\ln(n)\right)^\beta\right)$$

Proposition II - 2 : Relations de comparaisons sur des suites récurrentes

Soient $a \in \mathbb{R}_+^*$, $(b_n)_n$ une suite réelle et $(u_n)_{n\in\mathbb{N}}$ une suite vérifiant

$$\forall n \in \mathbb{N} \qquad u_{n+1} = a u_n + b_n$$

On a alors

- Si $b_n = o\left(n^\nu\right)$, avec $\nu \in \mathbb{R}^+$, et $a > 1$, alors $u_n = \Theta\left(a^n\right)$.
- Si $b_n \sim \lambda b^n$ avec $\lambda > 0$ et $b < a$, alors $u_n = \Theta\left(a^n\right)$
- Si $b_n \sim \lambda a^n$ avec $\lambda > 0$, alors $u_n = \Theta\left(n a^n\right)$
- Si $b_n \sim \lambda b^n$ avec $\lambda > 0$ et $b > a$, alors $u_n = \Theta\left(b^n\right)$.

D'après le principe général de la méthode diviser pour régner on a la relation de récurrence suivante :

$$C(n) = aC\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + bC\left(\left\lceil \frac{n}{2} \right\rceil\right) + \beta_n$$

où β_n correspond à la complexité du partage et de la fusion.

Proposition II - 3 : Approximation du coût partage/fusion

Soit a et b deux entiers non simultanément nuls, $(\beta_n)_{n\in\mathbb{N}^*}$ et $(\beta'_n)_{n\in\mathbb{N}^*}$ deux suites de même ordre de grandeurs. Alors les suites $(u_n)_{n\in\mathbb{N}^*}$ et $(u'_n)_{n\in\mathbb{N}^*}$ telles que $u_1 = u'_1$, et

$$\forall n \in \mathbb{N}^* \qquad \begin{array}{l} u_n = au_{\lfloor \frac{n}{2} \rfloor} + bu_{\lceil \frac{n}{2} \rceil} + \beta_n \\ u'_n = au'_{\lfloor \frac{n}{2} \rfloor} + bu'_{\lceil \frac{n}{2} \rceil} + \beta'_n \end{array}$$

sont du même ordre de grandeur.

On voit donc pourquoi on peut se contenter d'un ordre de grandeur pour le calcul de la complexité du partage et de la fusion.

Théorème II - 1 : maitre

Soient a et b deux entiers non simultanément nuls, et $(C(n))_{n\in\mathbb{N}^*}$ une suite telle que

$$\forall n \in \mathbb{N}^* \qquad C(n) = aC\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + bC\left(\left\lceil \frac{n}{2} \right\rceil\right) + \Theta\left(N^p\right)$$

avec $k = a + b$, en posant $\omega = \log_2 k$:

- si $2^p < k$ (*i.e.* $p < \omega$), alors $C(N) = \Theta\left(N^\omega\right)$;
- si $2^p = k$ (*i.e.* $p = \omega$), alors $C(N) = \Theta\left(N^\omega \log_2 N\right)$;
- si $2^p > k$ (*i.e.* $p > \omega$), alors $C(N) = \Theta\left(N^p\right)$.

Remarque :

- On peut remplacer Θ par O.
- Lorsque partage et fusion ont un coût nul, on convient que $p = -\infty$ et $2^p = 0$.
- On peut généraliser ces résultats à des partages en sous-problèmes de taille N/b ($b > 2$) en remplaçant 2 par b (et $\log_2$ par $\log_b$!).

III Exemples

1) Exponentiation rapide

Énoncé du problème : À partir d'un élément x d'un groupe multiplicatif et d'un entier naturel n, calculer x^n en effectuant uniquement des multiplications.

Algorithme naïf : On calcule les puissances successivement, on effectue $\Theta(n)$ multiplications.

Algorithme diviser pour régner : On remarque que :
Si N est pair, $x^N = (x*x)^{\lfloor N/2 \rfloor}$, ou encore $x^N = x^{\lfloor N/2 \rfloor} * x^{\lfloor N/2 \rfloor}$.
Si N est impair, $x^N = x*(x*x)^{\lfloor N/2 \rfloor}$, ou encore $x^N = x * x^{\lfloor N/2 \rfloor} * x^{\lfloor N/2 \rfloor}$.

Complexité de diviser pour régner : On estime $C(N)$, le nombre de multiplications dans le groupe nécessaires au calcul de x^N. Partage et fusion nécessitent 1 ou 2 multiplications, et on effectue un appel récursif, donc :

$$C(N) = C(\lfloor N/2 \rfloor) + \Theta(1)$$

Donc, $a+b=1$, $\omega = 0 = p$, on en déduit :

$$C(N) = \Theta(\log_2 N)$$

Programmation récursive : On se contente de transcrire l'algorithme, en n'oubliant pas de traiter le ou les cas terminaux (initialisation).

Programmation itérative : La récursivité se traduit par une boucle `while`.
Pour comprendre l'algorithme, il est utile de faire le lien avec l'écriture en base 2 : si $n = \sum_{k=0}^{d} a_k 2^k$, alors

$$x^n = x^{a_0}(x^2)^{a_1}(x^4)^{a_2}\cdots(x^{2^d})^{a_d}$$

Les a_k sont les restes des divisions successives de n par 2.

```
let puissance x n =
  let res = ref 1. and n2 = ref n and puiss = ref x in
  while !n2 > 0 do
    if (!n2 mod 2) = 1 then res := !puiss *. !res;
    puiss := !puiss *. !puiss;
    n2 := !n2 / 2;
  done;
  !res;;
```

2) Multiplications rapides

a) Polynômes

Énoncé du problème : Étant donné deux polynômes $P(X), Q(X)$ de degré N décrits par des tableaux ou listes de coefficients, calculer les coefficients du polynôme $(P \cdot Q)(X)$.

Algorithme naïf : On utilise la formule du produit de polynômes. Pour cela, on doit calculer $\Theta(N^2)$ multiplications de scalaires (tous les produits possibles entre un coefficient de P et un coefficient de Q.

Produit de polynômes par la méthode de Karatsuba

Etant donnés deux polynômes $P\,(X)$ et $Q\,(X)$ de degré $N = 2^n$, on écrit :

$$P = A + X^{N/2} \cdot B \quad \text{et} \quad Q = C + X^{N/2} \cdot D \quad \text{où} \quad A, B, C, D \text{ sont de degré } N/2$$

et l'on développe :

$$P \cdot Q = A \cdot C + X^{N/2} \cdot (A \cdot D + B \cdot C) + X^N \cdot (B \cdot D) \ .$$

On peut se contenter de trois produits de polynômes de degré $N/2$, à savoir

$$A \cdot C,\ B \cdot D,\ (A + B) \cdot (C + D)$$

en remarquant que

$$A \cdot D + B \cdot C = (A + B) \cdot (C + D) - A \cdot C - B \cdot D \ .$$

Ainsi, le coût $C\,(N)$ en nombre de multiplications vérifie la relation de récurrence :

$$C\,(N) = 3 \cdot C\,(N/2)$$

et donc, avec $k = 3,\ \omega = \log_2 3 \approx 1,585,\ p = -\infty,\ k > 2^p$:

$$C\,(N) = \Theta\left(N^{\log_2 3}\right).$$

Remarque :

- Si on s'intéresse à la complexité en nombre d'additions de scalaires, le partage et la fusion s'effectuent en $\Theta(N^1)$. Comme $1 < \log_2(3)$, on garde la même complexité asymptotique.
- Il existe une méthode asymptotiquement plus rapide (mais beaucoup plus compliquée) : la *transformée de Fourier rapide*. Cette méthode repose sur le théorème d'interpolation de Lagrange, et permet d'effectuer le produit $O(N \log_2 N)$ multiplications de scalaires.
- L'algorithme de Karatsuba peut être adapté pour effectuer le produit de grands entiers. Si on travaille en base b, on écrit $p = \sum_{k=0}^{N} p_k b^k$, et $q = \sum_{k=0}^{N} q_k b^k$. Le calcul des chiffres de $p \times q$ en base b ressemble au calcul des coefficients d'un produit de polynôme.

b) Matrices

Produit de matrices carrées par la méthode de Strassen

Pour effectuer le produit de deux matrices carrées d'ordre $N = 2^n$, en les découpant en quatre blocs d'ordre $N/2$, on peut se contenter de sept produits de matrices d'ordre $N/2$. Ainsi, le coût $C(N)$ en nombre de multiplications vérifie la relation de récurrence :

$$C(N) = 7 \cdot C(N/2)$$

et donc, avec $k = 7,\ \omega = \log_2 7 \approx 2,81,\ p = -\infty,\ k > 2^p$:

$$C(N) = \Theta(N^\omega).$$

Ici, le coût de l'application naïve de la définition du produit matriciel est un $\Theta(N^3)$.

Pour les détails, afin de calculer :

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \times \begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{pmatrix},$$

on calcule les sept produits :

$$\begin{array}{rcl} M_1 & = & (B-D)\times(G+H)\ ;\ M_2 = (A+D)\times(E+H)\ ;\ M_3 = (A-C)\times(E+F)\ ; \\ M_4 & = & (A+B)\times H\ ;\ M_5 = A\times(F-H)\ ;\ M_6 = D\times(G-E)\ ;\ M_7 = (C+D)\times E \end{array}$$

et l'on remarque (habilement !) que :

$$\begin{array}{rcl} AE+BG & = & M_1+M_2-M_4+M_6 \\ AF+BH & = & M_4+M_5 \\ CE+DG & = & M_6+M_7 \\ CF+DH & = & M_2-M_3+M_5-M_7 \end{array}$$

3) Quelques algorithmes de tri

Pour trier un tableau de N valeurs, nous avons vu des algorithmes naïfs (tri par sélection, par insertion, ...) dont la complexité (en nombre de comparaisons entre éléments de tableau) est un $\Theta(N^2)$; et maintenant, divisons pour régner...

a) Tri fusion

L'idée récursive est naturelle : couper le tableau en deux, trier (récursivement) les deux sous-tableaux obtenus, puis fusionner les résultats.

Pour fusionner on compare les premiers éléments des deux sous tableaux adjacents et on prend le plus petit. On compare celui qui reste avec le suivant du tableau où on a pris le plus petit. Ceci jusqu'à

Ici, la partition ne nécessite aucune comparaison. On peut implémenter la fusion en $\Theta(\texttt{d}-\texttt{g}+1)$, où d, g sont les indices minimum et maximum des tableaux adjacents à fusionner. Ainsi, le coût $C(N)$ (en nombre de comparaison de tableau) du tri d'un tableau de taille N vérifie la relation de récurrence :

$$C(N) = C(\lfloor N/2 \rfloor) + C(\lceil N/2 \rceil) + \Theta(N)$$

donc, avec ici $k=2,\ \omega=1,\ p=1$:

$$C(N) = \Theta(N \log_2 N) \ .$$

Remarque : Plus précisément, $C(N) = C(\lfloor N/2 \rfloor) + C(\lceil N/2 \rceil) + \lambda.N$ donne $C(N) \sim \lambda N \log_2 N$.

b)Tri rapide (ou "quicksort", par C.A.R. Hoare – 1960)

L'idée consiste à partitionner d'abord le tableau à trier. Pour cela on choisit un pivot et, en parcourant le tableau, on met avant tous les éléments plus petits que le pivot et après les éléments les plus grands.

Le pivot se trouve donc à sa place définitive après ce partage.

Cependant les deux sous-tableaux n'ont pas un nombre équilibré d'éléments.

Ici, c'est la fusion' qui ne coûte rien, la partition se fait au prix de $\texttt{d}-\texttt{g}$ comparaisons. Mais nous n'avons pas de relation de récurrence du type précédent, car le partage ne se fait pas toujours en deux sous-tableaux de tailles égales.

Ainsi, dans le pire des cas (par exemple lorsque le tableau est initialement trié !), le coût $C(N)$ vérifie

$$C(1)=0 \quad \text{et} \quad C(N) = N-1+C(N-1) \quad \text{d'où} \quad C(N) = \frac{N(N-1)}{2} \sim \frac{N^2}{2} \ .$$

Pour ce qui est de la complexité en moyenne, encore notée $C(N)$, en considérant — à chaque étape — les différentes valeurs de `p` comme équiprobables, j'obtiens successivement, en posant $N = d-g+1$, $n = p-g+1$ et $C(0)=0$:

$$C(1)=0 \quad \text{et} \quad C(N) = N-1+\frac{1}{N}\sum_{n=1}^{N}\big(C(n-1)+C(N-n)\big) = N-1+\frac{2}{N}\sum_{k=1}^{N-1}C(k) \ .$$

$$NC(N) = N(N-1)+2\sum_{k=1}^{N-1}C(k) \ ; \quad (N+1)C(N+1) = N(N+1)+2\sum_{k=1}^{N}C(k) \ ;$$

$$(N+1)\,C\,(N+1) = (N+2)\,C\,(N) + 2N\;; \quad NC\,(N) = (N+1)\,C\,(N-1) + 2\,(N-1)\;;$$

$$\frac{C\,(N)}{N+1} = \frac{C\,(N-1)}{N} + 2\frac{N-1}{N\,(N+1)}\;; \quad \frac{C\,(N)}{N+1} = 2\sum_{n=2}^{N}\frac{n-1}{n\,(n+1)}\;.$$

On peut en déduire que $C\,(N) \sim 2N\ln N = (2\ln 2)\,N\log_2 N$.

4) Plus petite distance dans un nuage de points

On considère un ensemble de n points du plan et on désire rechercher les deux points dont la distance est la plus petite parmi le nuage de points.

Un algorithme naïf aura une complexité en $\Theta(n^2)$, avec la méthode diviser pour régner on obtient une complexité en $\Theta(n\log(n))$.

Le principe : On considère un nuage de N points.

- On commence par créer deux tableaux triés $\mathscr{P}$ et $\mathscr{P}'$. Le premier contenant les coordonnées des points dans l'ordre des abscisses croissantes (ordre lexicographique) et le second contenant les coordonnées des points dans l'ordre des ordonnées croissantes(ordre lexicographique inversé).
- On sépare alors le premier tableau en deux tableaux $\mathscr{P}_1$, $\mathscr{P}_2$, de tailles $\lfloor\frac{N}{2}\rfloor$ et $\lceil\frac{N}{2}\rceil$. On crée alors deux tableaux issus de $\mathscr{P}'$ correspondants aux points des tableaux précédents.
- On résout le problème pour les tableaux $\mathscr{P}_1$, $\mathscr{P}_2$, donnant des distances minimales d_1, d_2 et des points correspondants (M_1, M_2') et (M_2, M_2').
- On va fusionner : on définit $d = \min(d_1, d_2)$, $m = \dfrac{x_{\lfloor\frac{N}{2}\rfloor} + x_{\lfloor\frac{N}{2}\rfloor+1}}{2}$ et la bande T correspondant aux points d'abscisses appartenant à $[m-d, m+d]$. A l'aide de $\mathscr{P}'$ on crée la liste des points contenus dans cette bande.
  Pour un point d'ordonnée y_0 dans cette bande il y a au maximum 7 points dans le rectangle $[m-d, m+d]\times[y_0, y_0+d]$. On trouve alors de façon linéaire le couple (M_T, M_T') de distance d_T.
- On conclut en prenant la plus petite distance des 3.