902 Diviser pour régner, Exemples et Applications.

Le paradigme "Diviser pour régner" est une stratégie très courante en algorithmique.

I) Présentation de la méthode

1) Principe général

Cette méthode aboutit naturellement à des algorithmes récursifs et s'articule autour de 3 étapes :

① Diviser : On sépare notre problème sur une entrée de taille n en a sous-problèmes sur des entrées de taille $\lfloor \frac{n}{b} \rfloor$.

② Régner : On arrive récursivement à un problème de taille suffisamment petite pour être traité facilement.

③ Combiner : On remet en commun les solutions des sous-problèmes résolus pour produire la solution totale.

exemple 1 : Recherche dichotomique dans un tableau trié T de taille n. $T = [\![e_1, \dots e_n]\!]$

cherche(T, x) : si T est vide $\to$ non
sinon si $e_{\lfloor \frac{n}{2} \rfloor} = x \to$ oui
sinon si $e_{\lfloor \frac{n}{2} \rfloor} > x \to$ cherche$([\![e_1, \dots e_{\lfloor \frac{n}{2} \rfloor}]\!], x)$
sinon cherche$([\![e_{\lfloor \frac{n}{2} \rfloor + 1}, \dots e_n]\!], x)$

2) Complexité de tels algorithmes [Pap]

On note $T(n)$ le temps d'exécution de l'algorithme sur une entrée de taille n.

On note $D(n)$ le temps nécessaire à l'étape ① et $C(n)$ le temps nécessaire pour l'étape ③.

On obtient donc une relation de récurrence de la forme : $T(n) = a\,T(\lfloor \frac{n}{b} \rfloor) + D(n) + C(n)$ $(*)$

Pour simplifier la résolution, on pose $f(n) = D(n) + C(n)$. Si $f(n) = O(n^d)$ pour un certain $d \geq 0$, on résoud la relation $(*)$ grâce au théorème suivant :

Théorème Général : Si $T(n) = a\,T(\lceil \frac{n}{b} \rceil) + O(n^d)$ avec $a > 0$, $b > 1$, $d \geq 0$ alors :

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

De plus, la complexité T est une fonction croissante donc si $T(n) = a\,T(\lfloor \frac{n}{b} \rfloor) + O(n^d)$ alors $T(n) \leq a\,T(\lceil \frac{n}{b} \rceil) + O(n^d)$ et on a bien les même résultats que dans le théorème. Ceci permet aussi d'ommettre les $\lceil\ \rceil$ ou $\lfloor\ \rfloor$.

Rq : Il existe une version plus précise de ce théorème mais l'on peut se contenter de celui-ci en première approche.

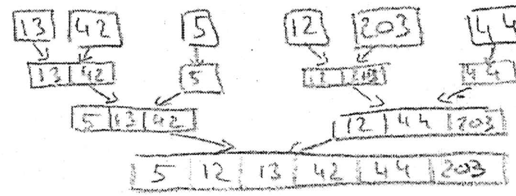
II - Quelques exemples classiques [Pap/COR]

1) Des tris

▶ le tri fusion : [Pap]

Il s'agit de séparer le tableau de taille n en 2 tableaux de taille $\frac{n}{2}$ et de les trier. La combinaison de deux tableaux triés de taille $\frac{n}{2}$ en un tableau trié de taille n se fait en $O(n)$.

ex : $T =$ | 13 | 42 | 5 | 12 | 203 | 44 |



$T(n) = 2\,T(\frac{n}{2}) + O(1) + O(n)$
$= 2\,T(\frac{n}{2}) + O(n)$
$\to O(n \ln n)$

▶ le tri rapide : [Pap]

On choisit arbitrairement un élément p du tableau, le pivot, puis on sépare les éléments du tableau en deux sous tableaux : les éléments $\leq p$ et les éléments $\geq p$. On trie ensuite ces deux sous tableaux :

tri $\leq p$	p	tri $\geq p$

La complexité au pire survient donc lorsque le tableau est tel que le pivot est ≥ ou < que tous les éléments du tableau (ex : tableau déjà trié quand le pivot est le premier élément du tableau).
On a alors $T(n) = \sum_{k=1}^{n} k-1 = \frac{n(n-1)}{2} = O(n^2)$. En moyenne on obtient deux sous tableaux de taille $(\frac{n}{2})$ et $T(n) = O(n \log n)$

2) Multiplications en tout genre

▶ multiplication d'entiers (Karatsuba)

On considère deux entiers x et y et leur écriture binaire :

$$\left.\begin{array}{l} x = \boxed{x_G}\boxed{x_D} = 2^{n/2} x_G + x_D \\ y = \boxed{y_G}\boxed{y_D} = 2^{n/2} y_G + y_D \end{array}\right\} xy = 2^n x_G y_G + 2^{n/2}(x_G y_D + y_G x_D) + x_D y_D$$

On remarque que $x_G y_D + y_G x_D = (x_G + x_D)(y_G + y_D) - x_G y_G - x_D y_D$
Pour multiplier 2 entiers de taille n, il suffit de faire 3 multiplications sur des entiers de taille $\frac{n}{2}$.

$$T(n) = 3T(\lceil \tfrac{n}{2} \rceil) + O(n) \xrightarrow[\text{théorème}]{} O(n^{1.59})$$

▶ multiplication de matrices (Strassen)

On considère deux matrices carrées A et B que l'on considère de taille $n = 2^n$, quitte à rajouter un bloc identité.

$$A = \left[\begin{array}{c|c} A_1 & A_2 \\ \hline A_3 & A_4 \end{array}\right] \qquad B = \left[\begin{array}{c|c} B_1 & B_2 \\ \hline B_3 & B_4 \end{array}\right]$$

on a alors $AB = \left[\begin{array}{c|c} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ \hline P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{array}\right]$

avec $\cdot P_1 = A_1(B_2 - B_4)$; $P_2 = (A_1 + A_2)B_4$; $P_3 = (A_3 + A_4)B_4$
$P_4 = A_4(B_3 - B_1)$; $P_5 = (A_1 + A_4)(B_1 + B_4)$; $P_6 = (A_2 - A_4)(B_3 + B_4)$;
$P_7 = (A_1 - A_3)(B_1 - B_3)$

Il suffit donc de calculer les 7 matrices P_i qui demandent chacune 1 multiplication de matrices de taille $\frac{n}{2}$.

$$T(n) = 7T(\tfrac{n}{2}) + O(n^2) \xrightarrow[\text{théorème}]{} T(n) = O(n^{\log_2 7})$$

▶ multiplication de polynôme (FFT)

On considère deux polynômes P et Q de degré $n = 2^k$. On peut alors les multipliers grâce à un procédé de transformée de Fourier rapide (FFT) :

$$\begin{array}{ccc} P = (p_m)_{m \in \mathbb{N}} ; Q = (q_m)_{m \in \mathbb{N}} & \xrightarrow{\text{évaluation}} & (x_j, P(x_j)), (x_j, Q(x_j)) \\ \Big\downarrow {\scriptstyle \text{multiplication directe en } O(n^2)} & & \Big\downarrow {\scriptstyle \text{multiplication en } O(n)} \\ PQ = \left(\sum_{i+j=m} p_i q_j\right)_m & \xleftarrow{\text{interpolation}} & (x_j, PQ(x_j)) \end{array}$$

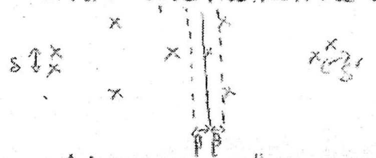
→ complexité totale en $O(n \log n)$

IDEV

III - Des exemples un peu moins classique : les algorithmes géométriques.

1) Recherche des points les plus proches

Une stratégie naïve consiste à tester toutes les paires de points, cet algorithme donne une complexité en $O(n^2)$. Pour améliorer ça, on utilise une méthode diviser-pour-régner.
On sépare l'ensemble des points en deux demi plans contenant chacun $(\#P)/2$ éléments. Il y a alors 3 possibilités
- Soit les points les plus proches sont dans le demi-plan de droite, soit dans le demi plan de gauche soit un de chaque côté. On teste les deux premiers cas dans la récursion et le dernier au moment de la combinaison.



où $\beta = \min(\delta, \delta')$

→ on obtient de cette manière une complexité en $O(n \log n)$.

2) Enveloppe convexe

Pour faire le calcul de l'enveloppe convexe d'un ensemble de point P, on sépare comme dans la recherche de paire de points les plus proches notre ensemble P en deux sous ensembles séparés par une droite verticale. On calcule les enveloppes convexes de ces deux sous-ensembles, puis on les réunit intelligemment ~~(fusion)~~

Cet algorithme donne le résultat en $O(n\log n)$.

3) Diagramme de Voronoi

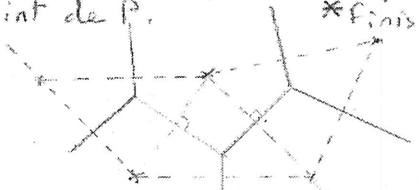
Un troisième exemple d'algorithme géométrique est l'algorithme permettant le calcul du diagramme de Voronoi d'un ensemble de points du plan.

Cet algorithme peut notamment servir pour la modélisation de croissance de structures cristallines.

[Sha]

Le diagramme de Voronoï d'un ensemble P de points du plan est un pavage du plan par des polygones* tel que chacun de ces polygones contient un et un seul élément x de P et pour tout point à l'intérieur du polygone, ce point est plus proche de x que de n'importe quel autre point de P.

*finis ou non



Pour calculer le diagramme on sépare les points de P comme pour le calcul de l'enveloppe convexe, on calcule le diagramme de Voronoi de chacun des 2 sous-ensembles et on fusionne en $O(n)$.

$\rightarrow T(n) = 2T(\frac{n}{2}) + O(n) \rightarrow O(n\ln n)$

IV - Application à la théorie

1) Complexité en espace

[Car]

On peut montrer par une méthode diviser pour régner le théorème de Savitch :

[DEV]

Théorème : Soit s une fonction $s : \mathbb{N} \to \mathbb{R}^+$, $s = \Omega(n)$. Alors toute machine de Turing non déterministe de complexité spatiale en $O(s(n))$ est équivalente à une machine de Turing déterministe de complexité spatiale en $O(s^2(n))$.

Pour démontrer ce théorème, on utilisera le lemme technique (admis) suivant.

Lemme Soit M une machine de Turing de complexité temporelle $t(n)$ et de complexité spatiale $s(n)$ alors il existe $k>0$ telle que $t(n) \leq 2^{ks(n)}$

Le théorème de Savitch permet alors de démontrer que PSpace = NPSpace.

2) Approximation du problème de voyageur de commerce euclidien

Le problème du voyageur de commerce consiste à trouver, étant donné un ensemble P de points du plan, un chemin de longueur minimal passant exactement une fois par chaque point de P.

Pour cela, il existe une approche gloutonne mais cette approche tourne en temps quadratique ce qui n'est pas assez rapide lorsque l'on a beaucoup de points, c'est à dire très fréquemment en pratique.

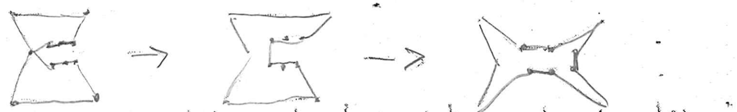
Par une méthode diviser-pour-régner on cherche une solution acceptable, bonne sans forcément être optimale en réduisant à une complexité $O(n\ln n)$.

Pour cela on considère le plus petit rectangle contenant tous les points de P.

À l'étape de partition, on sépare le rectangle en deux rectangle horizontalement ou verticalement tel que la séparation est parallèle au plus petit côté, passe par un point de P et sépare les points en deux sous-ensembles de même cardinal et ce jusqu'à avoir suffisamment peu de point pour utiliser une méthode de recherche exhaustive.

L'étape de combinaison des sous problèmes est alors soit optimale mais très coûteuse, soit approchée mais plus rapide.

ex :



Le point important est de garder les points proches dans les même sous-problèmes c'est pourquoi on peut aussi penser à une approche diviser pour régner basée sur les diagramme de Voronoi.

Référence :

[Pap] : Papadimitriou, Algorithms (KL)
(DASGUPTA)

[Cor] : Cormen, Introduction à l'algorithmique (partout)

[Boi] : Boissonnat - Yvinec, géométrie algorithmique (KL à Irmar)

[Car] : Carton, langage formel (KL à [illegible])

[Sha] : Preparata - Shamos, Computational geometry (IRMAR)