Licence informatique - L3 Année 2009/2010

Conception d'algorithmes et applications (LI325)

TD Algorithmes *diviser pour régner*

Séances 1,2,3

Rappel du théorème maître

Théorème 1. *(Résolution de récurrences « diviser pour régner »). Soient $a \geq 1$ et $b > 1$ deux constantes, soient f une fonction à valeurs dans $\mathbb{R}^+$ et T une fonction de $\mathbb{N}^*$ dans $\mathbb{R}^+$ vérifiant, pour tout n suffisamment grand, l'encadrement suivant : $aT(\lfloor n/b \rfloor) + f(n) \leq T(n) \leq aT(\lceil n/b \rceil) + f(n)$. Alors T peut être bornée asymptotiquement comme suit :*

1. *Si $f(n) = O(n^{(\log_b a)-\varepsilon})$ pour une certaine constante $\varepsilon > 0$, alors $T(n) = \Theta(n^{\log_b a})$.*
2. *Si $f(n) = \Theta(n^{\log_b a})$, alors $T(n) = \Theta(n^{\log_b a} \log n)$.*
3. *Si $f(n) = \Omega(n^{(\log_b a)+\varepsilon})$ pour une certaine constante $\varepsilon > 0$, et si on a asymptotiquement pour une constante $c < 1$, $af(n/b) < cf(n)$, alors $T(n) = \Theta(f(n))$.*

1 Notations Asymptotiques

Exercice 1.1. Rappeler les définitions des notations O, Ω, Θ pour les fonctions à valeur dans $\mathbb{R}_+^*$.

Exercice 1.2. Le but de cet exercice est de tester votre compréhension de la notion de complexité dans le pire des cas.

1. Si je prouve que la complexité dans le pire des cas d'un algorithme est en $O(n^2)$, est-il possible qu'il soit en $O(n)$ sur *certaines* données ?

2. Si je prouve que la complexité dans le pire des cas d'un algorithme est en $O(n^2)$, est-il possible qu'il soit en $O(n)$ sur *toutes* les données ?

3. Si je prouve que la complexité dans le pire des cas d'un algorithme est en $\Theta(n^2)$, est-il possible qu'il soit en $O(n)$ sur *certaines* données ?

4. Si je prouve que la complexité dans le pire des cas d'un algorithme est en $\Theta(n^2)$, est-il possible qu'il soit en $O(n)$ sur *toutes* les données ?

Exercice 1.3. Sur une échelle croissante, classer les fonctions suivantes selon leur comportement asymptotique : c'est-à-dire $g(n)$ suit $f(n)$ si $f(n) = O(g(n))$.

$f_1(n) := 2\,n$	$f_2(n) := 2^n$	$f_3(n) := \log(n)$	$f_4(n) := \frac{n^3}{3}$
$f_5(n) := n!$	$f_6(n) := \log(n)^2$	$f_7(n) := n^n$	$f_8(n) := n^2$
$f_9(n) := n + \log(n)$	$f_{10}(n) := \sqrt{n}$	$f_{11}(n) := \log(n^2)$	$f_{12}(n) := e^n$
$f_{13}(n) := n$	$f_{14}(n) := \sqrt{\log(n)}$	$f_{15}(n) := 2^{\log_2(n)}$	$f_{16}(n) := n\,\log(n)$

2 Fonctions définies par récurrence

Exercice 2.1. Soit un algorithme dont la complexité $T(n)$ est donnée par la relation de récurrence :

$$\begin{aligned} T(1) &= 1, \\ T(n) &= 3\,T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n^2. \end{aligned}$$

a. Calculer $T(n)$ en résolvant la récurrence.

b. Déterminer $T(n)$ à l'aide du théorème maître.

Remarque :
On a

$$a^{\log_x b} = b^{\log_x a} \quad \text{et} \quad \sum_{i=0}^{k} x^i = \frac{x^{k+1}-1}{x-1}.$$

Exercice 2.2. Le but de cet exercice est d'utiliser le théorème maître pour donner des ordres de grandeur asymptotique pour des fonctions définies par récurrence.

1. En utilisant le théorème maître, donner un ordre de grandeur asymptotique pour $T(n)$:

a.
$$T(1) = 1, \quad T(n) = 2\,T\left(\frac{n}{2}\right) + n^2;$$

b.
$$T(1) = 0, \quad T(n) = 2\,T\left(\frac{n}{2}\right) + n;$$

c.
$$T(1) = 1, \quad T(n) = 2\,T\left(\frac{n}{2}\right) + \sqrt{n};$$

d.
$$T(1) = 0, \quad T(n) = T\left(\frac{n}{2}\right) + \Theta(1).$$

2. Essayer de donner des exemples d'algorithmes dont le coût est calculé par l'une de ces récurrences.

Exercice 2.3. Le but de cet exercice est de choisir l'algorithme de type « diviser pour régner » le plus rapide pour un même problème.

1. Pour résoudre un problème manipulant un nombre important de données, on propose deux algorithmes :

a. un algorithme qui résout un problème de taille n en le divisant en 2 sous-problèmes de taille $n/2$ et qui combine les solutions en temps quadratique,

b. un algorithme qui résout un problème de taille n en le divisant en 4 sous-problèmes de taille $n/2$ et qui combine les solutions en temps $O(\sqrt{n})$.

Lequel faut-il choisir ?

2. Même question avec les algorithmes suivants :

a. un algorithme qui résout un problème de taille n en le réduisant à un sous-problème de taille $n/2$ et qui combine les solutions en temps $\Omega(\sqrt{n})$,

b. un algorithme qui résout un problème de taille n en le divisant en 2 sous-problèmes de taille $n/2$ et qui combine les solutions en temps constant.

Exercice 2.4. Considérons maintenant un algorithme dont le nombre d'opérations sur une entrée de taille n est donné par la relation de récurrence :

$$T(1) = 0 \quad \text{et} \quad T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2\,n}{3}\right) + n.$$

1. Construire un arbre représentant les appels récursifs de l'algorithme. En déduire la complexité de $T(n)$.

2. Vérifier le résultat par récurrence (méthode par « substitution »).

3. Cette récurrence peut-elle être résolue en utilisant le théorème maître ?

3 Recherche d'éléments, de suite d'éléments

Exercice 3.1. Donnez un algorithme de recherche d'un élément dans un tableau **trié** (dans l'ordre croissant) :

a. en utilisant une recherche séquentielle,

b. en utilisant le principe « diviser pour régner ».

Donnez la complexité (en nombre de comparaisons) dans le pire cas de chacun de ces deux algorithmes en fonction de la taille n du tableau.

Exercice 3.2. On cherche le k-ième plus petit élément dans un tableau **non trié**.

a. Proposer un algorithme utilisant le principe « diviser pour régner » en s'inspirant de celui du tri rapide.

b. Faire la trace de l'algorithme sur le tableau | 2 | 4 | 1 | 6 | 3 | 5 | pour $k = 4$.

c. Donnez la complexité (en nombre de comparaisons) dans le pire cas de cet algorithme en fonction de la taille n du tableau.

d. Que se passe-t-il si, à chaque étape, le tableau considéré est coupé en 2 sous-tableaux de tailles équivalentes ?

Exercice 3.3. On dispose d'un tableau d'entiers *relatifs* de taille n. On cherche à déterminer la suite d'entrées consécutives du tableau dont la somme est maximale. Par exemple, pour le tableau $T = [-1, 9, -3, 12, -5, 4]$, la solution est 18 (somme des éléments de $T[2..4] = [9, -3, 12]$).

a. Proposer un algorithme élémentaire pour résoudre ce problème. Donner sa complexité.

b. Donner un (meilleur) algorithme de type « diviser pour régner ». Quelle est sa complexité ?

Exercice 3.4. Un élément x est majoritaire dans un ensemble E de n éléments si E contient strictement plus de $n/2$ occurrences de x. La seule opération dont nous disposons est la comparaison (égalité ou non) de deux éléments.

Le but de l'exercice est d'étudier le problème suivant : étant donné un ensemble E, représenté par un tableau, existe-t-il un élément majoritaire, et si oui quel est-il ?

1. Algorithme naïf

a. Écrivez un algorithme NombreOccurrences qui, étant donné un élément x et deux indices i et j, calcule le nombre d'occurrences de x entre $E[i]$ et $E[j]$. Quelle est sa complexité (en nombre de comparaisons) ?

b. Écrivez un algorithme Majoritaire qui vérifie si un tableau E contient un élément majoritaire, et si oui le retourne. Quelle est sa complexité ?

2. Algorithme de type « diviser pour régner »
Pour calculer l'élément majoritaire de l'ensemble E (s'il existe), on découpe l'ensemble E en deux sous-ensembles E_1 et E_2 de même taille, et on calcule récursivement dans chaque sous-ensemble l'élément majoritaire. Pour qu'un élément x soit majoritaire dans E, il suffit que l'une des conditions suivantes soit vérifiée :
– x est majoritaire dans E_1 et dans E_2,
– x est majoritaire dans E_1 et non dans E_2, mais suffisamment présent dans E_2,
– x est majoritaire dans E_2 et non dans E_1, mais suffisamment présent dans E_1.
Écrivez un algorithme utilisant cette approche « diviser pour régner ». Analysez sa complexité.

3. Pour améliorer l'algorithme précédent, on propose de construire un algorithme PseudoMajoritaire tel que :
– soit l'algorithme garantit que E ne possède pas d'élément majoritaire,
– soit il rend un couple (x, p) tel que $p > n/2$, x est un élément apparaissant au plus p fois dans E, et tout élément $y \neq x$ de E apparaît au plus $n - p$ fois dans E.
Donnez un algorithme récursif vérifiant ces conditions. Donnez sa complexité. Déduisez-en un algorithme de recherche d'un élément majoritaire, et donnez sa complexité.

4 Tris

Exercice 4.1 (Tri fusion)**.** Le but de l'exercice est de trier un tableau en utilisant l'approche « diviser pour régner » : on commence par couper le tableau en deux, on trie chaque moitié avec notre algorithme, puis on fusionne les deux moitiés.

1. Écrire un algorithme Fusion(A, B) permettant de fusionner deux tableaux A et B déjà triés de tailles n_A et n_B en un seul tableau trié T. Quelle est sa complexité ?

2. À l'aide de l'algorithme décrit ci-dessus, écrire l'algorithme du tri fusion.

3. Quelle est la complexité du tri fusion ?

5 Algèbre et arithmétique

Exercice 5.1 (Multiplication de deux polynômes)**.** On représente un polynôme $p(x) = \sum_{i=0}^{m-1} a_i x^i$ par la liste de ses coefficients $[a_0, a_1, \ldots, a_{m-1}]$. Nous nous intéressons au problème de la multiplication de deux polynômes : étant donnés deux polynômes $p(x)$ et $q(x)$ de degré au plus $n-1$, calculer $pq(x) = p(x)q(x) = \sum_{i=0}^{2n-2} c_i x^i$.

1. Préliminaires : énumérez les opérations basiques (de coût 1) qui serviront à calculer la complexité de vos algorithmes. Vérifiez que l'addition de deux polynômes de degré $n-1$ se fait en $\Theta(n)$.

2. Méthode directe.
Donnez un algorithme calculant directement chaque coefficient c_i. Quelle est sa complexité ?

3. On découpe chaque polynôme en deux parties de tailles égales :

$$p(x) = p_1(x) + x^{n/2} p_2(x),$$
$$q(x) = q_1(x) + x^{n/2} q_2(x),$$

et on utilise l'identité

$$pq = p_1 q_1 + x^{n/2}(p_1 q_2 + p_2 q_1) + x^n p_2 q_2.$$

Donnez un algorithme récursif, utilisant le principe « diviser pour régner », qui calcule le produit de deux polynômes à l'aide de l'identité précédente. Si $T(n)$ est le coût de la multiplication de deux polynômes de taille n, quelle est la relation de récurrence vérifiée par $T(n)$? Quelle est la complexité de votre algorithme ? Comparez-la à la complexité de la méthode directe.

4. On utilise maintenant la relation

$$p_1 q_2 + p_2 q_1 = (p_1 + p_2)(q_1 + q_2) - p_1 q_1 - p_2 q_2.$$

Donnez un algorithme calculant le produit pq à l'aide de cette relation, et explicitez le nombre de multiplications et d'additions de polynômes de degré $n/2$ utilisées. Quelle est la complexité de votre algorithme ?

5. Pensez-vous que cet algorithme a une complexité optimale ?

Exercice 5.2 (Algorithme de Strassen pour la multiplication de deux matrices)**.** La multiplication de matrices est très fréquente dans les codes numériques. Le but de cet exercice est de proposer un algorithme rapide pour cette multiplication.

1. Donnez un algorithme direct qui multiplie deux matrices A et B de taille $n \times n$. Donnez sa complexité en nombre de multiplications élémentaires.

2. Notons $C = A\,B$. Dans le but d'utiliser le principe « diviser pour régner », on divise les matrices A, B et C en quatre sous-matrices $n/2 \times n/2$

$$C = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = A\,B.$$

Les sous-matrices de C peuvent être obtenues en effectuant les produits et additions de matrices $n/2 \times n/2$ suivants :

$$c_{11} = a_{11}b_{11} + a_{12}b_{21}, \qquad c_{12} = a_{11}b_{12} + a_{12}b_{22},$$
$$c_{21} = a_{21}b_{11} + a_{22}b_{21}, \qquad c_{22} = a_{21}b_{12} + a_{22}b_{22}.$$

Notons $T(n)$ le coût de la multiplication de deux matrices de taille n, quelle est la relation de récurrence vérifiée par $T(n)$ dans ce cas ? Donnez la solution de cette relation de récurrence, comparez-la à la complexité de la méthode directe.

3. Strassen (en 1969) a suggéré de calculer les produits intermédiaires suivants :

$$m_1 = (a_{12} - a_{22})\,(b_{21} + b_{22}), \qquad m_4 = (a_{11} + a_{12})\,b_{22},$$
$$m_2 = (a_{11} + a_{22})\,(b_{11} + b_{22}), \qquad m_5 = a_{11}\,(b_{12} - b_{22}),$$
$$m_3 = (a_{11} - a_{21})\,(b_{11} + b_{12}), \qquad m_6 = a_{22}\,(b_{21} - b_{11}),$$
$$m_7 = (a_{21} + a_{22})\,b_{11},$$

puis de calculer C comme suit :

$$c_{11} = m_1 + m_2 - m_4 + m_6, \qquad c_{12} = m_4 + m_5,$$
$$c_{21} = m_6 + m_7, \qquad c_{22} = m_2 - m_3 + m_5 - m_7.$$

a. Montrez que C est bien le produit de A par B (attention, la multiplication de deux matrices n'est pas commutative !).

b. Donnez la relation de récurrence vérifiée par $T(n)$. Déduisez-en la complexité de l'algorithme de Strassen.

c. Pensez-vous que cet algorithme a une complexité optimale ?

Exercice 5.3 (Matrices Toeplitz)**.** Une matrice Toeplitz est une matrice de taille $n \times n$ (a_{ij}) telle que $a_{i,j} = a_{i-1,j-1}$ pour $2 \leq i, j \leq n$.

1. La somme de deux matrices Toeplitz est-elle une matrice Toeplitz ? Et le produit ?

2. Trouver un moyen d'additionner deux matrices Toeplitz en $O(n)$.

3. Comment calculer le produit d'une matrice Toeplitz $n \times n$ par une vecteur de longueur n ? Quelle est la complexité de l'algorithme ?

6 Géométrie algorithmique

Exercice 6.1 (Recherche des deux points les plus rapprochés)**.** Le problème consiste à trouver les deux points les plus rapprochés (au sens de la distance euclidienne classique) dans un ensemble P de n points. Deux points peuvent coïncider, auquel cas leur distance vaut 0. Ce problème a notamment des applications dans les systèmes de contrôle de trafic : un contrôleur du trafic aérien ou maritime peut avoir besoin de savoir quels sont les appareils les plus rapprochés pour détecter des collisions potentielles.

1. Donnez un algorithme naïf qui calcule directement les deux points les plus rapprochés, et analysez sa complexité (on représentera P sous forme de tableau et on analysera la complexité de l'algorithme en nombre de comparaisons).

Nous allons construire un algorithme plus efficace basé sur le principe « diviser pour régner ». Le principe est le suivant :

a. Si $|P| \leq 3$, on détermine les deux points les plus rapprochés par l'algorithme naïf.

b. Si $|P| > 3$, on utilise une droite verticale Δ, d'abscisse l, séparant l'ensemble P en deux sous ensembles (Pgauche et Pdroit) tels que l'ensemble Pgauche contienne la moitié des éléments de P et l'ensemble Pdroit l'autre moitié, tous les points de Pgauche étant situés à gauche de ou sur Δ, et tous les points de Pdroit étant situés à droite de ou sur Δ.

c. On résout le problème sur chacun des deux sous-ensembles Pgauche et Pdroit, les deux points les plus rapprochés sont alors :
- soit les deux points les plus rapprochés de Pgauche,
- soit les deux points les plus rapprochés de Pdroit,
- soit un point de Pgauche et un point de Pdroit.

On supposera qu'initialement l'ensemble P est trié selon des abscisses et ordonnées croissantes ; en d'autres termes l'algorithme prend la forme : PlusProche(PX, PY), où PX est un tableau correspondant à l'ensemble P trié selon les abscisses (et selon les ordonnées pour les points d'abscisses égales) et PY à l'ensemble P trié selon les ordonnées. Les ensembles P, PX et PY contiennent les mêmes points $P[i]$ (seul l'ordre change).

2. Écrivez un algorithme DiviserPoints qui calcule à partir de P les tableaux PgaucheX (resp. PgaucheY) correspondant aux points de l'ensemble Pgauche triés selon les abscisses (resp. les ordonnées), de même pour les tableaux PdroitX et PdroitY. Montrez que cette division peut être effectuée en $O(n)$ comparaisons.

3. Notons δ_g (resp. δ_d) la plus petite distance entre deux points de Pgauche (resp. Pdroit), et $\delta = \min(\delta_g, \delta_d)$. Il faut déterminer s'il existe une paire de points dont l'un est dans Pgauche et l'autre dans Pdroit, et dont la distance est strictement inférieure à δ.

a. Si une telle paire existe, alors les deux points se trouvent dans le tableau PY', trié selon les ordonnées, et obtenu à partir de PY en ne gardant que les points d'abscisse x vérifiant $l - \delta \leq x \leq l + \delta$. Donnez un algorithme qui calcule PY' en $O(n)$ comparaisons.

b. Montrez que si 5 points sont situés à l'intérieur ou sur le bord d'un carré de coté $a > 0$, alors la distance minimale entre 2 quelconques d'entre eux est strictement inférieure à a.

c. Montrez que s'il existe un point $p_g = (x_g, y_g)$ de Pgauche et un point $p_d = (x_d, y_d)$ de Pdroit tels que $\text{dist}(p_g, p_d) < \delta$ et $y_g < y_d$, alors p_d se trouve parmi les 7 points qui suivent p_g dans le tableau PY'.

d. Donnez un algorithme, en $O(n)$ comparaisons, qui vérifie s'il existe une paire de points dans Pgauche et Pdroit dont la distance est strictement inférieurs à δ, et si oui la retourne.

4. En déduire un algorithme recherchant 2 points à distance minimale dans un ensemble de n points.

5. Donnez la relation de récurrence satisfaite par le nombre de comparaisons effectuées par cet algorithme. En déduire sa complexité.