IN 101 - TD 07
21 octobre 2011

Énoncé

ENSTA ParisTech

Le but du TD est la manipulation de matrices. Les matrices seront stockées sous la forme d'un tableau des lignes de type `int**` et chaque case contiendra une ligne de type `int*`, c'est-à-dire un pointeur sur la première case de la ligne. On ne considère ici que des matrices carrées, dont la taille sera passée en argument au programme (dans `argv[1]`) puis stockée dans une variable globale accessible à toutes les fonctions.

Exercice 1 : Gestion des matrices

1.a] Écrivez une fonction `matinit` qui réserve la mémoire et initialise les cases avec des valeurs aléatoires entre 0 et 9. Utilisez pour cela les fonctions `srand` et `rand` comme au TD 02. Cette fonction ne prend pas d'argument (la taille de la matrice est accessible via une variable globale) et retourne un `int**`. Pensez à bien tester que toutes les allocation mémoire ont marché !

1.b] Écrivez une fonction `matprint` qui affiche une matrice. Cette fonction prend un `int**` en argument et ne retourne rien.

1.c] Écrivez une fonction `matfree` qui libère la mémoire occupée par une matrice passée en argument. Dans le `main` il faudra utiliser cette fonction pour libérer toutes les matrices allouées en faisant bien attention de ne jamais perdre tous les pointeurs vers une matrice.

Exercice 2 : Quelques opérations

2.a] Écrivez une fonction `matadd` qui additionne deux matrices passées en argument et retourne un pointeur vers une nouvelle matrice (la somme des deux) et testez la. Cette fonction peut faire appel à `matinit` pour réserver la mémoire de la matrice du résultat.

2.b] Écrivez une fonction `matmul` qui multiplie deux matrices et testez la.

2.c] Écrivez une fonction `mattrans` qui retourne la transposée d'une matrice passée en argument. Pour vérifier vos fonctions, vérifiez à la main que le produit d'une matrice par sa transposée est bien symétrique.

Exercice 3 : Calcul du déterminant

3.a] Calculez le déterminant d'une matrice en utilisant un algorithme récursif. La fonction devra prendre en argument un `int**` correspondant à une matrice (ou sous-matrice) et un `int` correspondant à la taille de la sous-matrice. On rappelle la formule de Lagrange : $det(M) = \sum_i (-1)^{i+j} M_{i,j} det(M_j^i)$ où M_i^j est la sous-matrice obtenue en supprimant la i-ième ligne et la j-ième colonne. Il faudra penser à libérer toute la mémoire allouée pour les sous-matrices une fois qu'elle ne sert plus : les plus rapides pourront compiler leur programme en mode debug (`gcc -g`) et utiliser `valgrind` pour repérer d'éventuelles fuites mémoire.

Utilisez un développement suivant la dernière colonne. Sauvez la matrice avant de calculer la sous-matrice. Utilisez le fait que votre tableau de lignes permet de manipuler des lignes entières

directement. La comatrice pour `a[n − 1][n − 1]` *consiste simplement à enlever la dernière ligne et la dernière colonne. On n'a pas besoin d'enlever les valeurs sur la dernière colonne, il suffit de dire que la longueur de la ligne est* `n − 1`*. Ensuite, pour la comatrice du coefficient* `a[n − 2][n − 1]`*, il faut déplacer la dernière ligne à la place de l'avant-dernière ligne, ce qui se fait bien en manipulant simplement le pointeur vers la ligne. Il n'est pas nécessaire de copier toute la matrice, copier le tableau de lignes suffit.*

3.b] Quelle est la complexité de ce calcul de déterminant ? Peut-on faire mieux ?

3.c] (Pour ceux qui vont vite) Programmez un pivot de Gauss pour calculer le déterminant efficacement en mettant la matrice sous forme triangulaire supérieure.

Exercice 4 : Multiplication de Strassen (pour ceux qui vont très vite)

4.a] Quelle est la complexité de l'algorithme naïf de multiplication de deux matrices de taille $n \times n$ en nombre d'additions et de multiplications d'entiers ?

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & f \\ g & h \end{pmatrix}$$

$$\begin{aligned} p_1 &= a(g-h) \\ p_2 &= (a+b)h \\ p_3 &= (c+d)e \\ p_4 &= d(f-e) \\ p_5 &= (a+d)(e+h) \\ p_6 &= (b-d)(f+h) \\ p_7 &= (a-c)(e+g) \end{aligned}$$

On peut écrire :

$$\begin{aligned} r &= p_5 + p_4 - p_2 + p_6 \\ s &= p_1 + p_2 \\ t &= p_3 + p_4 \\ u &= p_5 + p_1 - p_3 - p_7 \end{aligned}$$

4.b] Calculez le nombre d'additions et de multiplications dans le cas de l'algorithme de multiplication naïf et avec celui de Strassen pour des matrices 2×2 ?

4.c] Supposons que $n = 2m$, décomposer la matrice initiale en 4 blocs de matrices $m \times m$ et appliquons une fois Strassen aux gros blocs. Calculez le nombre de multiplications et d'additions d'entiers en fonction de n ?

4.d] Supposons n une puissance de 2 (on peut toujours rajouter des zéros si ce n'est pas le cas). Soit $M(n)$ le nombre de multiplications réalisées par Strassen pour multiplier 2 matrices de taille n et $A(n)$ pour le nombre d'additions. Écrivez les formules de récurrences sur le nombre de multiplications et le nombre d'additions en fonction de n, et les résoudre.

4.e] Programmez la multiplication de Strassen et vérifiez que le produit de grandes matrices (de taille 1024 par exemple) est plus rapide qu'avec la multiplication normale.