Dirk Praetorius, Samuel Ferraz-Leite
Christoph Überhuber

Wintersemester 2010/11
03. November 2010

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 4

Aufgabe 4.1*. *Bubble-Sort* ist ein ineffizienter, aber kurzer Sortier-Algorithmus: Man vergleicht aufsteigend jedes Element eines Arrays x_j mit seinem Nachfolger x_{j+1} und - falls notwendig - vertauscht die beiden. Nach dem ersten Durchlauf muß zumindest das letzte Element bereits am richtigen Platz sein. Der nächste Durchlauf muß also nur noch bis zur vorletzten Stelle gehen, usw. Wie viele geschachtelte Schleifen braucht dieses Vorgehen? Schreiben Sie eine Funktion `bubblesort`, die ein gegebenes Array $x \in \mathbb{R}^n$ mittels Bubble-Sort aufsteigend sortiert, d.h. $x_1 \leq x_2 \leq \cdots \leq x_n$, und zurückgibt. Schreiben Sie ferner ein aufrufendes Hauptprogramm, das den dynamischen Vektor x und die Länge n einliest und x in sortierter Reihenfolge ausgibt. Speichern Sie den Source-Code unter `bubblesort.c` in das Verzeichnis `serie04`.

Aufgabe 4.2*. Schreiben Sie eine verbesserte Variante von Bubble-Sort, bei dem man sich die Stelle des letzten Tausches merkt. Ab dieser Stelle müssen die Daten ja bereits sortiert sein. Der nächste Durchlauf braucht also nur noch bis zu dieser Array-Position zu gehen. Zur Beschreibung von Bubble-Sort siehe Aufgabe 4.1. Speichern Sie den Source-Code unter `bubblesortV2.c` in das Verzeichnis `serie04`.

Aufgabe 4.3*. Schreiben Sie eine Funktion `unique`, die einen Vektor $x \in \mathbb{R}^n$ aufsteigend sortiert, doppelte Einträge streicht und den Vektor in gekürzter Form zurückgibt. Die Funktion soll also beispielsweise den Vektor $x = (4, 3, 5, 1, 4, 3, 4) \in \mathbb{R}^7$ durch den Vektor $x = (1, 3, 4, 5) \in \mathbb{R}^4$ überschreiben. Die Länge n der Vektoren ist dynamisch zu realisieren. Schreiben Sie ein aufrufendes Hauptprogramm, in dem n und $x \in \mathbb{R}^n$ eingelesen werden und das Ergebnis der Funktion `unique` ausgegeben wird. Speichern Sie den Source-Code unter `unique.c` in das Verzeichnis `serie04`.

Aufgabe 4.4*. Schreiben Sie eine Funktion `matrixvectorU`, die das Matrix-Vektor-Produkt $y = Ux$ mit einer oberen Dreiecksmatrix $U \in \mathbb{R}^{n \times n}$ mittels geeigneter Schleifen berechnet. Dabei bezeichnet man eine Matrix

$$U = \begin{pmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ & u_{22} & u_{23} & \dots & u_{2n} \\ & & \ddots & & \vdots \\ \mathbf{0} & & & & u_{nn} \end{pmatrix}$$

als obere Dreiecksmatrix. Mathematisch formuliert gilt also $U_{jk} = 0$ für $j > k$. Bei der Berechnung soll auf die offensichtlichen Nulleinträge von U nicht zugegriffen werden, um den Rechenaufwand gering zu halten. Schreiben Sie ferner ein aufrufendes Hauptprogramm in dem die Dimension $n \in \mathbb{N}$ sowie die Einträge der Matrix U und die Koeffizienten des Vektors x eingelesen und Ux ausgegeben werden. Speichern Sie den Source-Code unter `matrixvectorU.c` in das Verzeichnis `serie04`.

Aufgabe 4.5. Das Produkt $L = AB$ zweier unterer Dreiecksmatrizen $A, B \in \mathbb{R}^{n \times n}$ ist wieder eine untere Dreiecksmatrix. Man beweise diese Aussage zunächst mathematisch, indem man sich die Formel

für das Matrix-Matrix-Produkt hinschreibe und mittels der Voraussetzung an A und B die Indizes vereinfache. Danach schreibe man eine Funktion `matrixmatrixL`, die die Produktmatrix berechnet und zurückgibt. Dabei sollen natürlich nur die nicht-trivialen Einträge von L, d.h. L_{jk} für $j \geq k$, berechnet werden. Ferner soll auf die trivialen Einträge von A und B nicht zugegriffen werden, d.h. man verwende die anfangs hergeleitete Formel.

Aufgabe 4.6. Schreiben Sie eine Funktion `spaltensummennorm`, die die Spaltensummennorm

$$\|A\|_S := \max_{j=1,\ldots,n} \sum_{i=1}^{n} |A_{ij}|$$

einer Matrix $A \in \mathbb{R}^{n\times n}$ zurückgibt.

Aufgabe 4.7. Die Frobeniusnorm einer Matrix $A \in \mathbb{R}^{m\times n}$ ist durch

$$\|A\|_F := \Big(\sum_{j=1}^{m} \sum_{k=1}^{n} A_{jk}^2 \Big)^{1/2}$$

definiert. Schreiben Sie eine Funktion `frobeniusnorm`, die für gegebene Matrix A und gegebene Dimensionen $m, n \in \mathbb{N}$ die Frobeniusnorm berechnet. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem die Zeilen- und Spaltendimensionen $m, n \in \mathbb{N}$ und A eingelesen werden und $\|A\|_F$ ausgegeben wird.

Aufgabe 4.8. Schreiben Sie eine Funktion `transpose`, die zu einer dynamisch gespeicherten Matrix $A \in \mathbb{R}^{m\times n}$ (vom Typ `double**`) die transponierte Matrix $A^T \in \mathbb{R}^{n\times m}$ berechnet. Dabei sind die Einträge von A^T gerade durch $(A^T)_{jk} = A_{kj}$ definiert. Im Hauptprogramm sollen die Dimensionen $m, n \in \mathbb{N}$ sowie die Einträge der Matrix A eingelesen und A^T ausgegeben werden.

Aufgabe 4.9. Eine Matrix $A \in \mathbb{R}^{n\times n}$ ist symmetrisch, falls $A_{jk} = A_{kj}$ für alle $j, k = 1, \ldots, n$ gilt. Schreiben Sie eine Funktion `issymmetric`, die eine Matrix A auf Symmetrie überprüft (Rückgabewert `1` bei Symmetrie und `0` bei Nicht-Symmetrie). Schreiben Sie ein aufrufendes Hauptprogramm, in dem A eingelesen wird und ausgegeben wird, ob A symmetrisch ist oder nicht. Speichern Sie A als C-Matrix, also vom Typ `double**`.

Aufgabe 4.10. Implementieren Sie den Quicksort-Algorithmus, um einen Vektor $x \in \mathbb{R}^n$ zu sortieren: Quicksort wählt willkürlich ein Pivotelement aus der zu sortierenden Liste x, z.B. x_1. Dann zerlegt man die Liste in zwei Teillisten: $x^{(<)}$ enthält alle Elemente $< x_1$, $x^{(\geq)}$ enthält alle Elemente $\geq x_1$. Diese Teillisten werden rekursiv sortiert. Anschließend wird das Ergebnis zusammengesetzt. Es besteht (in der Reihenfolge) aus der unteren Liste $x^{(<)}$, dem Pivotelement und der oberen Liste $x^{(\geq)}$. — Eine direkte Implementation dieses Algorithmus hat allerdings den Nachteil, dass zusätzlicher Speicher benötigt wird. Um dies zu vermeiden, geht man wie folgt vor: Beginnend mit $j = 2$ sucht man ein Element $x_j \geq x_1$, d.h. x_j gehört zu $x^{(\geq)}$. Ferner sucht man beginnend bei $k = n$ ein Element $x_k < x_1$, d.h. x_k gehört zu $x^{(<)}$. In diesem Fall vertauscht man x_j und x_k. Wenn sich die Zähler j und k treffen, liegt die Liste x bereits in der Form $(x^{(<)}, x_1, x^{(\geq)})$ vor. Es müssen nun nur noch die Teillisten sortiert werden.