Algorithmische Mathematik I

Winter Semester 2015 / 2016
Prof. Dr. Sven Beuchler
Markus Siebenmorgen

universitätbonn

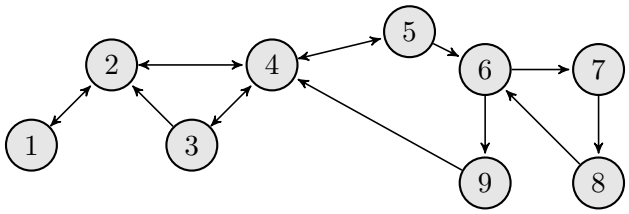
Aufgabenblatt 6.

Abgabedatum: **07.12.2015**.

Die Abgabe dieses Übungszettels findet aufgrund des Dies und des damit verbundenen Ausfalls der Vorlesung am 02.12. vor der darauf folgenden Vorlesung am Montag, den 07.12. um 10:00 Uhr statt.

Aufgabe 1. (Adjazenzmatrizen)

Wir betrachten den folgenden gerichteten Graphen, in dem $\leftrightarrow$ für eine Doppelkante steht (d.h. Kurzform für je eine Kante *vom* Knoten und eine Kante *zum* Knoten).



a) Geben Sie alle einfachen Pfade von Knoten 1 zu Knoten 8 sowie von Knoten 8 zu Knoten 1 an.

b) Eine Tabelle der nebenstehenden Form nennt man Adjazenzmatrix, wenn als Eintrag G_{ij} jeweils 1 für "es existiert eine Kante von Knoten i zu Knoten j" und entsprechend 0 für "es existiert *keine* Kante von i zu j" steht. Geben Sie die Adjazenzmatrix für obigen Graphen an. Geben Sie auch die Adjazenzmatrix an, wenn alle Kanten *ungerichtet* wären.

	1	2	3	...	10
1					
2					
3					
⋮					
10					

c) Gehen Sie nun davon aus, daß alle Kanten ungerichtet sind. Geben Sie die maximale Anzahl Kanten an, die entfernt werden können, so dass die Anzahl der Zusammenhangskomponenten immer noch gleich bleibt. Zeichnen Sie auch einen zugehörigen Graphen.

d) Zeichnen Sie einen Graph mit Knoten $1, 2, \ldots, 9$ und der Adjazenzmatrix:

	1	2	3	4	5	6	7	8	9
1		1			1				
2	1		1				1		
3		1							
4					1				
5	1			1					
6			1				1		
7		1				1		1	1
8							1		
9				1			1		

Muss der Graph gerichtet gezeichnet werden oder kann er auch ungerichtet sein?

(4 Punkte)

Aufgabe 2. (Wege)

Es sei $G = (V, E)$ mit ein einfacher ungerichteter Graph. Wir definieren die folgende Relation auf $V \times V$:

$$x \sim y \quad \text{es existiert ein } x\text{-}y \text{ Weg in } G.$$

a) Zeigen Sie, dass $\sim$ eine Äquivalenzrelation ist.

b) Bestimmen Sie alle Äquivalenzklassen von $\sim$.

(5 Punkte)

Aufgabe 3. (Kreise)

Sei $G = (V, E)$ ein ungerichteter Graph mit $|V| = n$ und $|E| = m$. Zeigen Sie, dass G einen Kreis besitzt, falls $m \geq n \geq 2$ ist.

(5 Punkte)

Aufgabe 4. (Eulersche Graphen)

Ein einfacher und ungerichteter Graph heißt *Eulerscher Graph*, falls es in ihm einen *Eulerzug* gibt, d.h. einen geschlossenen Kantenzug, der jede Kante des Graphen genau einmal enthält. Zeigen Sie, dass ein einfacher und ungerichteter Graph G ohne isolierten Ecken, d.h. es gibt keine Ecke, die in keiner Kante enthalten ist, genau dann ein Eulerscher Graph ist, wenn G zusammenhängend ist und alle Ecken geraden Grad haben.

Hinweis. Sie dürfen ohne Beweis verwenden, dass in einem einfachen, ungerichteten und zusammenhängenden Graphen jede Ecke in einem Kreis enthalten ist, falls jede Ecke geraden Grad hat.

(6 Punkte)

Aufgabe 5. (Präsenzübung: valgrind und gnuplot)

a) Machen Sie sich mit der Funktion `gettimeofday` vertraut. Diese erlaubt es durch wiederholtes Aufrufen und geeignete Differenzenbildung die Laufzeit einzelner Routinen eines Programms zu bestimmen.

b) Machen Sie sich mit dem Programm `gnuplot` vertraut. Dieses erlaubt es, Graphen aus Daten zu erzeugen. Laden Sie die Programme `bubble.cpp`, `merge.cpp` und `quick.cpp` und bestimmen Sie die Laufzeit der Programme für $n = 10^k$ für $k = 1, \ldots, 8$ Elemente (für bubblesort nur bis $k = 5$) und plotten Sie diese mit Hilfe von gnuplot in einen Graphen.

Die Präsenzübung wird in der Woche 30.11-04.12 in den Programmiertutorien besprochen.

Programmieraufgabe 1. (Speicherung dünnbesetzter Matrizen)

Für das Abspeichern von Graphen werden häufig sogenannte Adjazenzmatrizen eingesetzt. Diese haben $|V|$ viele Zeilen und Spalten. Für jede Kante existiert dann ein Eintrag in der Matrix. Das Fehlen einer Kante zwischen zwei Knoten wird durch den Wert Null in der Matrix dargestellt. Dies führt dazu, dass Adjazenzmatrizen fast immer viele Null-Einträge aufweisen.

Ein Abspeichern dieser Null-Einträge im Rechner wäre sehr ineffizient. Daher wurden in der Vergangenheit verschiedene Datenformate entwickelt, mit denen man diese sogenannten *dünnbesetzten* Matrizen speichereffizienter ablegen kann. Eines hiervon ist das CSR-Format (compressed sparse row).

Bei diesem Format wird die Matrix mit Hilfe dreier Vektoren `row_offsets`, `column_indices` und `entries` gespeichert. Der letztgenannte Vektor enthält alle nicht-Null-Einträge der Matrix zeilenweise sortiert. Für $i = 0...N-1$ (N Anzahl der Zeilen) gibt `row_offsets[i]` die erste Indexposition in `entries` an, die nicht-Null-Elemente aus der i-ten Zeile enthält. Schließlich enthält `column_indices[j]` zum Eintrag `entries[j]` die entsprechende Spalten-Position. Um eine konsistente Abfrage der Anzahl von nicht-Null-Einträgen in der letzten Zeile zu erhalten, wird an `row_offsets[i]` ein weiterer Eintrag `row_offsets[N]` angehängt, welcher die Länge von `entries` beinhaltet.

Für die Matrix

$$\begin{pmatrix} 1 & 0 & -2 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

ergibt sich dann die nachfolgende Darstellung:

	0	1	2	3
entries	1	-2	1	1
column_indices	0	2	1	0
row_offsets	0	2	3	4

a) Schreiben Sie die Matrix

$$\begin{pmatrix} 1 & 2 & 0 & 0 & 1 \\ 0 & 1 & 5 & 0 & 0 \\ 0 & 1 & 0 & 2 & 2 \\ 0 & 2 & 3 & 0 & 0 \\ 0 & 0 & 1 & 0 & 4 \end{pmatrix}$$

im CSR-Matrix-Format auf.

b) Wir erinnern uns, dass der i-te Eintrag des Ergebnis-Vektors aus der Multiplikation einer Matrix $A \in \mathbb{R}^{n \times m}$ mit einem Vektor $x \in \mathbb{R}^m$ durch $(Ax)_i = \sum_{j=1}^{m} A_{ij} x_j$ gegeben ist. Ueberlegen sie sich auf dem Papier einen Algorithmus, mit dem Sie die Matrix-Vektor-Multiplikation einer Matrix im CSR-Format durchführen können.

c) Implementieren Sie nun diese Matrix-Vektor-Multiplikation in einer Methode `void MatVecMultCSR(int* row_offsets, int* column_indices, double* entries, double* x, double* y, int n, int m)`, wobei der Ausgabewert in das Feld `y` gespeichert wird. Sichern Sie diese Methode so, dass Sie diese später wieder verwenden können.

d) Vervollständigen Sie Ihr Testprogramm derart, dass sie damit die Matrix aus der ersten Teilaufgabe mit dem Vektor $(11111)^T$ multiplizieren.

e) Implementieren sie nun ebenfalls eine Matrix-Vektor-Multiplikation für eine vollständig abgespeicherte Matrix: `void MatVecMultDense(double* A, double* x, double* y, int n, int m)`. Die Matrix A ist hierbei ein eindimensionales Array, das die Zeilen der Matrix (auch mit Null-Einträgen) hintereinander aufgereiht enthält.

f) Schreiben Sie nun eine Routine, die zu einem gegebenen n die Tridiagonalmatrix $A \in \mathbb{R}^{n \times n}$ folgender Gestalt

$$A = \begin{bmatrix} -2 & 1 & & 0 \\ 1 & -2 & \ddots & \\ & \ddots & \ddots & 1 \\ 0 & & 1 & -2 \end{bmatrix},$$

erzeugt und diese mit allen Einträgen abspeichert und ebenfalls im CSR-Format speichert.

g) Führen Sie nun eine Laufzeitanalyse (analog zur Präsenzübung) mit Plot-Ausgabe über gnuplot durch, bei der Sie die Laufzeit der beiden implementierten Matrix-Vektor-Multiplikationen für mehrere Matrizengrößen $n = 2^k$ für $k = 6, 8, 10, 12, \ldots, 20$ testen (für die vollbesetzten Matrizen so weit wie möglich (nicht weiter als $k = 14$)). Welches ist die offensichtliche asymptotische Komplexität der beiden Verfahren?

(15 Punkte)

Die Programmieraufgabe wird in der Woche vom 07.12-11.12 im Cip-Pool Endenicher Allee oder im Cip-Pool Wegelerstraße abgegeben/vorgestellt. In der Woche vom 30.11-04.12 werden in den Cip-Pools Listen für die Abgabe aushängen.