Karlsruher Institut für Technologie

Tutorium 42, #10

Max Göckel– *uzkns@student.kit.edu*

Institut für Theoretische Informatik - Grundbegriffe der Informatik

Adjazenzmatrizen

Man kann eine Graph nicht nur als Zeichnung darstellen, sondern auch als Matrix.
Diese Matrizen heißen Adjazenzmatirzen, sie sind immer quadratisch und gehören eindeutig zu einem Graphen.

An einer Adjazenzmatrix kann man die Kanten eines Graphen ablesen.

Adjazenzmatrizen: Aufbau

- Jede Zeile i steht für einen Knoten $k_i \in V$
- Jede Spalte j steht für die von k_i direkt erreichbaren Knoten
  - $1 \Leftrightarrow \exists\{k_i, k_k\} \in E$, es existiert eine Kante zwischen den beiden Knoten
  - $0 \Leftrightarrow \neg\exists\{k_i, k_k\} \in E$, es existiert keine Kante

Adjazenzmatrizen: Beispiel

Die folgende Beispielmatrix beschreibt einen Graphen mit 4 Knoten:

$$A = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Adjazenzmatrizen: Ungerichtete Graphen

Bei ungerichteten Graphen ändert sich die Adjazenzmatrix nicht, die einzige Besonderheit ist dass sie symmetrisch ist.

Die folgende Beispielmatrix beschreibt den selben Graphen wie eben als ungerichteten Graph:

$$B = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

Adjazenzmatrizen: Aufgaben

Bestimme den Graph anhand der Adjazenzmatrix:

$$X = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Adjazenzmatrizen: Aufgabe

Bestimme zu den Graphen C, D die Adjazenzmatrizen.

Adjazenzmatrizen: Lösung

$$A_C = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

$$A_D = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Adjazenzmatrizen: Pfade von Länge 2

Eine Adjazenzmatrix A zeigt an, ob es einen direkten Pfad von k_i zu k_j gibt.
Um nun zu gucken ob es einen Pfad der Länge 2 gibt (von k_i über k_k zu k_j) müssen wir A^2 errechnen.

Adjazenzmatrizen: Pfade von Länge 2

Eine Adjazenzmatrix A zeigt an, ob es einen direkten Pfad von k_i zu k_j gibt.
Um nun zu gucken ob es einen Pfad der Länge 2 gibt (von k_i über k_k zu k_j) müssen wir A^2 errechnen.

Matrixmultiplikation:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \\ 2 & 1 & 4 \end{pmatrix} \text{ wird dann zu } A^2 = \begin{pmatrix} 13 & 7 & 19 \\ 10 & 9 & 19 \\ 13 & 9 & 24 \end{pmatrix}$$

Element $a^2_{33} = a_{31} \cdot a_{13} + a_{32} \cdot a_{23} + a_{33} \cdot a_{33} = 2 \cdot 3 + 1 \cdot 2 + 4 \cdot 4 = 6 + 2 + 16 = 24$

Für Pfade Länge x berechnet man dann einfach A^x.

Adjazenzmatrizen: Beispiel

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 \end{pmatrix}$$

Die 2 bedeutet es gibt 2 Wege von Knoten 4 zu 2.

Wegematrizen

Eine x-mal mit sich selbst multiplizierte Adjazenzmatrix zeigt uns einen Pfad der Länge x an.
Was ist nun wenn wir wissen wollen ob *überhaupt* ein Pfad von A nach B existiert (in egal wie vielen Schritten).

Dazu brauchen wir die Wegematrix, eine Adjazenzmatrix, die für einen Pfad bel. Länge von A zu B eine 1 hat und sonst eine 0.

Wegematrizen: Aufgabe

Bestimme zu Graph E eine 4x4-Wegematrix.

Wegematrizen: Aufgabe

Bestimme zu Graph E eine 4x4-Wegematrix.

$$A_E = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

Warshallalgorithmus

Um eine Wegematrix zu errechnen hilft der Warshall-Algorithmus.

Dieser errechnet in n Durchgängen (n ist dabei die Anzahl an Knoten in einem Graph) aus einer Adjazenzmatrix eine Wegematrix.

Warshallalgorithmus: Aufbau

- In der Adjazenzmatrix alle Diagonalwerte auf 1 setzen
- n Durchgänge: Im *i*-ten Durchgang alle Nullen in der Matrix angucken
  - 0 durch 1 ersetzten, wenn in der Zeile und Spalte an der i-ten Stelle je eine 1 steht

Warshallalgorithmus: Beispiel

Ein kurzes Beispie aus Durchgang 2 ($i = 2$):
Wir betrachten $a_{33} = 0$

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

Da a_{33} in der 3. Spalte und Zeile steht und $i = 2$ nehmen wir von den Zeilen/Spalten das zweite Element (a_{32} und a_{23}).
Diese sind beide 1, so wird a_{33} im nächsten Durchgang auch zur 1.

Warshallalgorithmus: Aufgabe

Bestimme zu den Graphen F, G die Wegematrix.

Warshallalgorithmus: Lösung

Bestimme zu den Graphen F, G die Wegematrix.

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

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

Algorithmen: Zeitkomplexität

Wir kennen (aus Programmieren) schon mehrere Sortieralgorithmen:

- BubbleSort
- InsertionSort
- BogoSort / RandomSort
- MergeSort

Wieso gibt es mehrere Sortieralgorithmen wenn doch einer ausreichen würde um Arrays zu sortieren?

Algorithmen: Vergleiche

Verschiedene Algorithmen haben unterschiedliche Laufzeiten *("Zeitkomplexität")*.
So ist MergeSort im worst-case immer noch schneller als InsertionSort im worst-case.

Um die Zeitkomplexität von Algorithmen anzugeben interessiert uns immer der worst-case. Wir probieren eine obere Schranke zu finden und zwar nicht in Sekunden/Millisekunden, sondern in Abhängigkeit von n.
Konstante Faktoren und Konstanten werden dabei irgnoriert, d.h. $1000n + 10$ und $0,5n$ haben die gleiche Komplexität, nämlich n.

Algorithmen: Laufzeiten

Die häufigsten unterschiedlichen Größenordnungen sind (aufsteigend sortiert):

- 1 (atomare Ausführung)
- $log(n)$
- n
- $n \cdot log(n)$
- n^2
- n^3
- 2^n
- $n!$
- n^n

Algorithmen: O-Kalkül

Um Algorithmen zu vergleichen und einzuordnen gibt es verschiedene Schranken:

- Obere Schranke O (Groß-O): In $O(f(n))$ sind alle Funktionen die langsamer oder gleich schnell wie f wachsen.
- Obere Schranke Ω (Omega): In $\Omega(f(n))$ sind alle Funktionen die schneller oder gleich schnell wie f wachsen.
- Obere Schranke Θ (Theta): In $\Theta(f(n))$ sind alle Funktionen die genau gleich schnell wie f wachsen.

Bei den Vergleichen schaut man sich immer ein möglichst großes n an (also $\lim_{n\to\infty} f(n)$), so gilt für einen Algorithmus g der anfangs schneller wächst als f, dann aber ab $n = 10000$ von f überholt wird immer noch $g \in O(f(n))$.

Groß-O: Definition

Das Groß-O ist mathematisch so definiert:

- $O(f(n)) = \{g(n) | \exists c \in \mathbb{R}_+ : \exists n_0 \in \mathbb{N}_0 : \forall n \geq n_0 : g(n) \leq c \cdot f(n)\}$

$O(f(n))$ sind also alle g die ab irgendeinem n_0 von f überholt werden.

Ω: Definition

Das Ω ist mathematisch so definiert:

- $\Omega(f(n)) = \{g(n) | \exists c \in \mathbb{R}_+ : \exists n_0 \in \mathbb{N}_0 : \forall n \geq n_0 : g(n) \geq c \cdot f(n)\}$

$\Omega(f(n))$ sind also alle g die ab einem n_0 immer schneller oder gleich schnell wachsen wie f.

Θ: Definition

Das Θ ist mathematisch so definiert:

- $\Theta(f(n)) = \{g(n) | \exists c_1, c_2 \in \mathbb{R}_+ : \exists n_0 \in \mathbb{N}_0 : \forall n \geq n_0 : c_1 f(n) \leq g(n) \leq c_2 f(n)\}$
- Alternativ: $\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$

$\Theta(f(n))$ legt ab n_0 einen Bereich um f ab in dem g liegt.