16 ERSTE ALGORITHMEN IN GRAPHEN

In dieser Einheit wollen wir beginnen, Algorithmen auch unter quantitativen Gesichtspunkten zu betrachten.

Als „Aufhänger“ werden wir eine vereinfachte Problemstellung betrachten, die mit einer der am Ende der Einheit 15 über Graphen aufgezählten verwandt ist: Man finde heraus, ob es in einem gegebenen gerichteten Graphen einen Pfad von einem gegebenen Knoten i zu einem gegebenen Knoten j gibt.

Wir beginnen in Abschnitt 16.1 mit der Frage, wie man denn Graphen im Rechner repräsentiert. In 16.2 nähern wir uns dann langsam dem Erreichbarkeitsproblem, indem wir uns erst einmal nur für Pfade der Länge 2 interessieren. Das führt auch zu den Konzepten Matrixaddition und Matrixmultiplikation. Auf der Matrizenrechnung aufbauend beginnen wir dann in 16.3 mit einem ganz naiven Algorithmus und verbessern ihn in zwei Schritten. Einen der klassischen Algorithmen, den von Warshall, für das Problem, werden wir in Abschnitt 16.4 kennenlernen.

Nachdem wir uns in dieser Einheit beispielhaft auch mit dem Thema beschäftigt haben werden, wie man — in einem gewissen Sinne — die Güte eines Algorithmus quantitativ erfassen kann, werden wir das in der nachfolgenden Einheit **??** über quantitative Aspekte von Algorithmen an weiteren Beispielen aber auch allgemein etwas genauer beleuchten.

16.1 REPRÄSENTATION VON GRAPHEN IM RECHNER

In der Vorlesung über Programmieren haben Sie schon von Klassen, Objekten und Attributen gehört und Arrays kennengelernt. Das kann man auf verschiedene Arten nutzen, um z. B. Graphen im Rechner zu repräsentieren. Ein erster Ansatz in Java könnte z. B. so aussehen:

```
class Vertex {
   String name;
}

class Graph {
   Vertex[] vertices;
   Edge[] edges;
}
```

```
class Edge {
   Vertex start;
   Vertex end;
}
```

Dabei hat man aber mehr hingeschrieben als man „eigentlich“ will, denn die Knoten (und auch die Kanten) sind durch die Nummerierung der Komponenten der Arrays total angeordnet worden. Das ist bei den Mengen der mathematischen Definition nicht der Fall.

Aber es schadet nicht. Da man die Nummern aber sowieso schon hat, macht man, zumindest wenn man besonders kurz und übersichtlich sein will, den Schritt und sagt, dass die Identitäten der Knoten einfach die Zahlen eines Anfangsstückes der natürlichen Zahlen sind. Solange man sich mit Problemen beschäftigt, die unter Isomorphie invariant sind, ergeben sich hierdurch keine Probleme. Deswegen ist für uns im folgenden bei allen Graphen $V = \mathbb{Z}_n$ für ein $n \geq 1$.

```
class Vertex {
   int id;
}

class Graph {
   Vertex[] vertices;
   Edge[] edges;
}
```

```
class Edge {
   Vertex start;
   Vertex end;
}
```

Gelegentlich verwendet man als Knotennummern auch Anfangsstücke der positiven ganzen Zahlen (also ohne Null). Lassen Sie sich von solchen kleinen technischen Details nicht verunsichern. Man macht, was einem gerade am besten erscheint.

Wenn man Graphen in Java wie oben skizziert implementieren würde, dann könnte man bei einer gegebenen Kante leicht auf deren Anfangs- und Endknoten zugreifen. Wie Sie bald sehen werden, will man aber mitunter umgekehrt zu einem gegebenen Knoten v z. B. auf die ihn verlassenden Kanten zugreifen. Das wäre aber nur umständlich möglich: Man müsste systematisch alle Kanten darauf hin überprüfen, ob sie bei v starten.

Es gibt (neben anderen) zwei gängige Methoden, dieses Problem zu beseitigen. Die eine besteht darin, zu jedem Knoten eine Liste der ihn verlassenden Kanten oder der über solche Kanten erreichbaren Nachbarknoten mitzuführen. Wenn man diese Liste als Array implementiert, dann wäre

```
class Vertex {
   int id;
   Vertex[] neighbors;
}
```

```
class Graph {
    Vertex[] vertices;
}
```


Man spricht dann davon, dass für jeden Knoten die *Adjazenzliste* vorhanden ist. *Adjazenzliste*

Wenn man mit kantenmarkierten Graphen arbeiten muss, benutzt man statt dessen lieber die *Inzidenzlisten*. Das ist für einen Knoten die Liste der Kanten, die ihn als einen Endpunkt haben. *Inzidenzliste*

Wir wollen im folgenden aber eine andere Methode benutzen, um die Beziehungen zwischen Knoten zu speichern. Wenn man zu einem Knoten u womöglich oft und schnell herausfinden möchte, ob ein Knoten v Nachbar von u ist, dann ist es bequem, wenn man das immer leicht herausfinden kann. Man müsste dann (unter Umständen) nur noch die Klassen für einzelne Knoten und einen Graphen implementieren, z. B. so:

```
class Vertex {
   int id;
   boolean[] is_connected_to;
}
```

```
class Graph {
    Vertex[] vertices;
}
```

Für einen Knoten, also ein Objekt u der Klasse `Vertex`, wäre `is_connected_to` also ein Feld mit genau so vielen Komponenten wie es Knoten im Graphen gibt. Und u`.is_connected_to[`v`.id]` sei genau dann `true`, wenn eine Kante von u nach v existiert, und ansonsten `false` .

Betrachten wir als Beispiel den Graphen aus Abbildung 16.1:

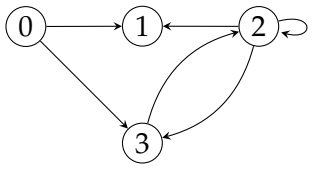


Abbildung 16.1: Ein kleiner Beispielgraph

Für das Objekt u der Klasse `Vertex`, das den Knoten 0 repräsentiert, würde z. B. gelten:

u.id	u.is_connected_to			
0	false	true	false	true
	0	1	2	3

Schreibt man das für alle vier Knoten untereinander, erhält man:

u.id	u.is_connected_to			
0	false	true	false	true
1	false	false	false	false
2	false	true	true	true
3	false	false	true	false
	0	1	2	3

Wenn man in dieser zweidimensionalen Tabelle nun noch `false` durch 0 und `true` durch 1 ersetzt, erhält man die sogenannte *Adjazenzmatrix* des Graphen.

Adjazenzmatrix

Manche haben Matrizen inzwischen in der Vorlesung „Lineare Algebra“ kennengelernt, andere haben zumindest schon ein Beispiel gesehen, in dem ein Graph als zweidimensionale Tabelle repräsentiert wurde. Im allgemeinen können Zeilenzahl m und Spaltenzahl n einer Matrix A verschieden sein. Man spricht dann von einer $m \times n$-Matrix. Die einzelnen Einträge in Matrizen werden in dieser Vorlesung immer Zahlen sein. Für den Eintrag in Zeile i und Spalte j von A schreiben wir auch A_{ij} (oder manchmal $(A)_{ij}$ o. ä.).

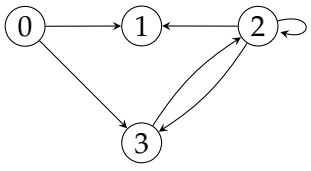
Für die Menge aller $m \times n$-Matrizen, deren Einträge alle aus einer Menge M stammen, schreiben wir gelegentlich $M^{m \times n}$.

Typischerweise notiert man eine Matrix ohne die ganzen senkrechten und waagerechten Striche, aber mit großen runden (oder manchmal auch eckigen) Klammern außen herum. Wenn es hilfreich ist, notiert man außerhalb der eigentlichen Matrix auch die Nummerierung der Zeilen bzw Spalten, wie es z. B. in Abbildung 16.2 gemacht ist.

Die Adjazenzmatrix eines gerichteten Graphen $G = (V, E)$ mit n Knoten ist eine $n \times n$-Matrix A mit der Eigenschaft:

$$A_{ij} = \begin{cases} 1 & \text{falls } (i,j) \in E \\ 0 & \text{falls } (i,j) \notin E \end{cases}$$

Als Beispiel ist in Abbildung 16.2 noch einmal der Graph mit vier Knoten und nun auch die zugehörige Adjazenzmatrix angegeben.



$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

Abbildung 16.2: Ein Graph und seine Adjazenzmatrix

Im Falle eines ungerichteten Graphen $U = (V, E)$ versteht man unter seiner Adjazenzmatrix die des zugehörigen gerichteten Graphen $G = (V, E_g)$.

Allgemein kann man jede binäre Relation $R \subseteq M \times M$ auf einer endlichen Menge M mit n Elementen durch eine $n \times n$-Matrix $A(R)$ repräsentieren, indem man definiert:

$$(A(R))_{ij} = \begin{cases} 1 & \text{falls } (i,j) \in R \quad \text{d. h. also } iRj \\ 0 & \text{falls } (i,j) \notin R \quad \text{d. h. also } \neg(iRj) \end{cases}$$

Dabei gehören zu verschiedenen Relationen auf der gleichen Menge M verschiedene Matrizen und umgekehrt.

So, wie man die Kantenrelation E eines gerichteten Graphen als Adjazenzmatrix darstellen kann, kann man natürlich auch jede andere Relation durch eine entsprechende Matrix darstellen, z. B. die „Erreichbarkeitsrelation“ E^*. Die zugehörige Matrix W eines Graphen wird üblicherweise *Wegematrix* genannt. Sie hat also die Eigenschaft:

Wegematrix

$$\begin{aligned} W_{ij} &= \begin{cases} 1 & \text{falls } (i,j) \in E^* \\ 0 & \text{falls } (i,j) \notin E^* \end{cases} \\ &= \begin{cases} 1 & \text{falls es in } G \text{ einen Pfad von } i \text{ nach } j \text{ gibt} \\ 0 & \text{falls es in } G \text{ keinen Pfad von } i \text{ nach } j \text{ gibt} \end{cases} \end{aligned}$$

Im folgenden wollen wir uns mit dem algorithmischen Problem beschäftigen, zu gegebener Adjazenzmatrix A die zugehörige Wegematrix W zu berechnen.

16.2 Berechnung der 2-Erreichbarkeitsrelation und Rechnen mit Matrizen

Es sei A die Adjazenzmatrix eines Graphen $G = (V, E)$; Abbildung 16.3 zeigt ein Beispiel. Wir interessieren uns nun zum Beispiel für die Pfade der Länge 2 von Knoten 2 zu Knoten 4. Wie man in dem Graphen sieht, gibt es genau zwei solche Pfade: den über Knoten 1 und den über Knoten 6.

Wie findet man „systematisch“ alle solchen Pfade? Man überprüft *alle* Knoten $k \in V$ daraufhin, ob sie als „Zwischenknoten“ für einen Pfad $(2, k, 4)$ in Frage kommen. Und $(2, k, 4)$ ist genau dann ein Pfad, wenn sowohl $(2, k) \in E$, also eine Kante, ist, als auch $(k, 4) \in E$, also eine Kante, ist. Und das ist genau dann der Fall, wenn in der Adjazenzmatrix A von G sowohl $A_{2k} = 1$ als auch $A_{k4} = 1$ ist. Das kann man auch so schreiben, dass $A_{2k} \cdot A_{k4} = 1$ ist.

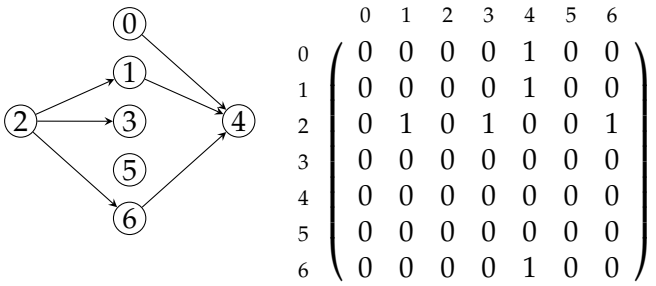


Abbildung 16.3: Beispielgraph für die Berechung von E^2

Wenn man nacheinander alle Elemente A_{2k} inspiziert, dann durchläuft man in A nacheinander die Elemente in der *Zeile* für Knoten 2. Und wenn man nacheinander alle Elemente A_{k4} inspiziert, dann durchläuft man in A nacheinander die Elemente in der *Spalte* für Knoten 4.

In Abbildung 16.4 haben wir schräg übereinander zweimal die Matrix A hingeschrieben, wobei wir der Deutlichkeit halber einmal nur die Zeile für Knoten 2 explizit notiert haben und das andere Mal nur die Spalte für Knoten 4. Außerdem haben wir im oberen linken Viertel alle Produkte $A_{2k} \cdot A_{k4}$ angegeben. Die Frage, ob es einen Pfad der Länge zwei $(2, k, 4)$ gibt, ist gleichbedeutend mit der Frage, ob eines dieser Produkte gleich 1 ist.

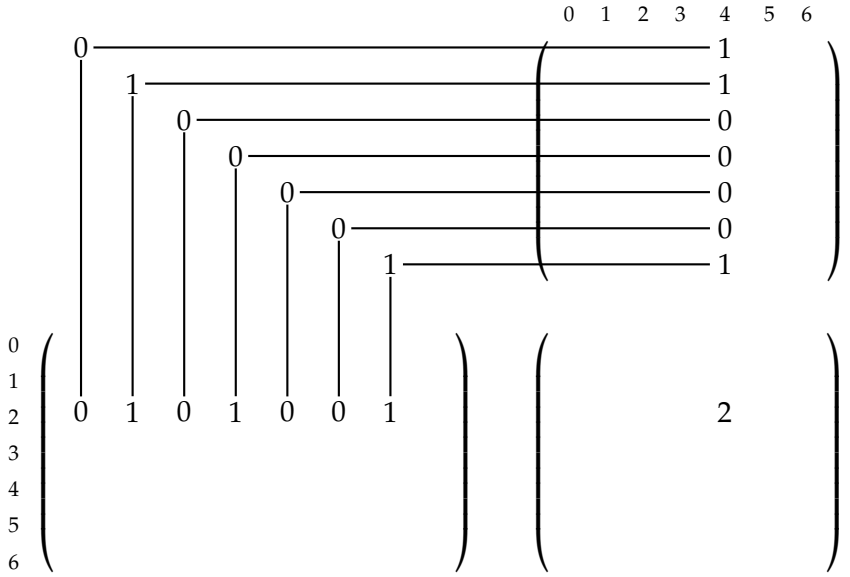


Abbildung 16.4: Der erste Schritt in Richtung Matrixmultiplikation

Wir tun jetzt aber noch etwas anderes. Wir addieren die Werte alle zu einer Zahl

$$P_{24} = \sum_{k=0}^{6} A_{2k} \cdot A_{k4}$$

und notieren Sie in einer dritten Matrix im unteren rechten Viertel der Darstellung. Dabei ist jetzt wichtig:

- Der Wert P_{24} gibt an, wieviele Pfade der Länge zwei es von Knoten 2 nach Knoten 4 gibt.
- Analog kann man für jedes andere Knotenpaar (i, j) die gleiche Berechnung durchführen:
$$P_{ij} = \sum_{k=0}^{n-1} A_{ik} \cdot A_{kj}$$
Dann ist P_{ij} die Anzahl der Pfade der Länge zwei von Knoten i zu Knoten j.

16.2.1 Matrixmultiplikation

Was wir eben aufgeschrieben haben ist nichts anderes als ein Spezialfall von *Matrixmultiplikation*. Ist A eine $\ell \times n$-Matrix und B eine $n \times m$-Matrix, so heißt die $\ell \times m$-Matrix C, bei der für alle i und alle j *Matrixmultiplikation*

$$C_{ij} = \sum_{k=0}^{n-1} A_{ik} \cdot B_{kj}$$

gilt, das Produkt der Matrizen A und B und man schreibt auch $C = A \cdot B$. (Wichtig: Selbst wenn einmal $\ell = n = m$ ist, ist trotzdem im Allgemeinen $A \cdot B \neq B \cdot A$!)

Algorithmisch notiert sähe die naheliegende Berechung des Produktes zweier Matrizen so aus, wie in Algorithmus 16.1 dargestellt. Wir werden im nächsten Kapitel aber sehen, dass es durchaus andere Möglichkeiten gibt!

Von besonderer Wichtigkeit sind die sogenannten *Einheitsmatrizen*. Das sind diejenigen $n \times n$-Matrizen I, bei denen für alle i und j gilt: *Einheitsmatrix*

$$\mathrm{I}_{ij} = \begin{cases} 1 & \text{falls } i = j \\ 0 & \text{falls } i \neq j \end{cases}$$

Zu beliebiger $m \times n$-Matrix A gilt für die jeweils passende Einheitsmatrizen:

$$\mathrm{I} \cdot A = A = A \cdot \mathrm{I}$$

Man beachte, dass die Einheitsmatrix auf der linken Seite Größe $m \times m$ hat und die auf der rechten Seite Größe $n \times n$.

Algorithmus 16.1: Einfachster Algorithmus, um zwei Matrizen zu multiplizieren

```
for i ← 0 to ℓ − 1 do
  for j ← 0 to m − 1 do
    C_ij ← 0
    for k ← 0 to n − 1 do
      C_ij ← C_ij + A_ik · B_kj
    od
  od
od
```

Ist eine Matrix quadratisch (wie z. B. die Adjazenzmatrix A eines Graphen), dann kann man A mit sich selbst multiplizieren. Dafür benutzt man wie schon an mehreren anderen Stellen in dieser Vorlesung die Potenzschreibweise:

$$A^0 = \mathrm{I}$$
$$\forall n \in \mathbb{N}_0 : A^{n+1} = A^n \cdot A$$

16.2.2 Matrixaddition

Matrixaddition

Für zwei Matrizen A und B der gleichen Größe $m \times n$ ist für uns in Kürze auch noch die Summe $A + B$ von Interesse. Die *Matrixaddition* von A und B liefert die $m \times n$-Matrix C, bei der für alle i und alle j gilt:

$$C_{ij} = A_{ij} + B_{ij} \; .$$

Nullmatrix

Das neutrale Element ist die sogenannte *Nullmatrix* (genauer gesagt die Nullmatrizen; je nach Größe). Sie enthält überall nur Nullen. Wir schreiben für Nullmatrizen einfach 0.

Zwei Matrizen zu addieren, ist leicht:

```
for i ← 0 to m − 1 do
  for j ← 0 to n − 1 do
    C_ij ← A_ij + B_ij
  od
od
```


16.3 BERECHNUNG DER ERREICHBARKEITSRELATION

Eine naheliegende Idee für die Berechnung von E^* ist natürlich, auf die Definition

$$E^* = \bigcup_{i=0}^{\infty} E^i$$

zurückzugreifen. Allerdings stellen sich sofort drei Probleme:

- Was kann man tun, um nicht unendlich viele Matrizen berechnen zu müssen? D. h., kann man das ∞ durch eine natürliche Zahl ersetzen?
- Woher bekommt man die Matrizen für die Relationen E^i, d. h. welcher Operation bei Matrizen entspricht das Berechnen von Potenzen bei Relationen?
- Wenn man die Matrizen hat, welcher Operation bei Matrizen entspricht die Vereinigung bei Relationen?

Beginnen wir mit dem ersten Punkt. Was ist bei Graphen spezieller als bei allgemeinen Relationen? Richtig: Es gibt nur *endlich* viele Knoten. Und das ist in der Tat eine große Hilfe: Wir interessieren uns für die Frage, ob für gegebene Knoten i und j ein Pfad in G von i nach j existiert. Sei $G = (V, E)$ mit $|V| = n$. Nehmen wir an, es existiert ein Pfad: $p = (i_0, i_1, \ldots, i_k)$ mit $i_0 = i$ und $i_k = j$. Was dann? Nun, wenn k „groß“ ist, genauer gesagt, $k \geq n$, dann kommen in der Liste p also $k+1 \geq n+1$ „Knotennamen“ vor. Aber G hat nur n verschiedene Knoten. Also muss mindestens ein Knoten x doppelt in der Liste p vorkommen. Das bedeutet, dass man auf dem Pfad von i nach j einen Zyklus von x nach x geht. Wenn man den weglässt, ergibt sich ein kürzerer Pfad, der immer noch von i nach j führt. Indem man dieses Verfahren wiederholt, solange im Pfad mindestens $n+1$ Knoten vorkommen, gelangt man schließlich zu einem Pfad, in dem höchstens noch n Knoten, und damit höchstens $n-1$ Kanten, vorkommen, und der auch immer noch von i nach j führt.

Mit anderen Worten: Was die Erreichbarkeit in einem endlichen Graphen mit n Knoten angeht, gilt:

$$E^* = \bigcup_{i=0}^{n-1} E^i$$

Aber höhere Potenzen schaden natürlich nicht. Das heißt, es gilt sogar:

16.1 Lemma. Für jeden gerichteten Graphen $G = (V, E)$ mit n Knoten gilt:

$$\forall k \geq n-1 : E^* = \bigcup_{i=0}^{k} E^i$$

16.3.1 Potenzen der Adjazenzmatrix

Wenn man die Adjazenzmatrix A eines Graphen quadriert, erhält man als Eintrag in Zeile i und Spalte j

$$(A^2)_{ij} = \sum_{k=0}^{n-1} A_{ik} A_{kj} \,.$$

Jeder der Summanden ist genau dann 1, wenn $A_{ik} = A_{kj} = 1$ ist, also genau dann, wenn (i,k,j) ein Pfad der Länge 2 von i nach j ist. Und für verschiedene k sind das auch verschiedene Pfade. Also ist

$$(A^2)_{ij} = \sum_{k=0}^{n-1} A_{ik} A_{kj}$$

gleich der Anzahl der Pfade der Länge 2 von i nach j.

Überlegen Sie sich, dass analoge Aussagen für $(A^1)_{ij}$ und Pfade der Länge 1 von i nach j, sowie $(A^0)_{ij}$ und Pfade der Länge 0 von i nach j richtig sind. Tatsächlich gilt:

16.2 Lemma. Es sei G ein gerichteter Graph mit Adjazenzmatrix A. Für alle $k \in \mathbb{N}_0$ gilt: $(A^k)_{ij}$ ist die Anzahl der Pfade der Länge k in G von i nach j.

Der Beweis wäre eine recht einfache vollständige Induktion. Der einzige gegenüber dem Fall $k = 2$ zusätzlich zu beachtende Punkt besteht darin, dass die Verlängerung verschiedener Wege um die gleiche Kante (falls das überhaupt möglich ist), wieder zu verschiedenen Wegen führt.

Signum-Funktion

Wir bezeichnen nun mit sgn die sogenannte *Signum-Funktion*

$$\operatorname{sgn} : \mathbb{R} \to \mathbb{R} : \operatorname{sgn}(x) = \begin{cases} 1 & \text{falls } x > 0 \\ 0 & \text{falls } x = 0 \\ -1 & \text{falls } x < 0 \end{cases}$$

Für die Erweiterung auf Matrizen durch komponentenweise Anwendung schreiben wir auch wieder sgn:

$$\operatorname{sgn} : \mathbb{R}^{m \times n} \to \mathbb{R}^{m \times n} : (\operatorname{sgn}(M))_{ij} = \operatorname{sgn}(M_{ij})$$

Unter Verwendung dieser Hilfsfunktion ergibt sich aus Lemma 16.2:

16.3 Korollar. Es sei G ein gerichteter Graph mit Adjazenzmatrix A.

1. Für alle $k \in \mathbb{N}_0$ gilt:

$$\operatorname{sgn}((A^k)_{ij}) = \begin{cases} 1 & \text{falls in } G \text{ ein Pfad der Länge } k \\ & \quad \text{von } i \text{ nach } j \text{ existiert} \\ 0 & \text{falls in } G \text{ kein Pfad der Länge } k \\ & \quad \text{von } i \text{ nach } j \text{ existiert} \end{cases}$$

2. Für alle $k \in \mathbb{N}_0$ gilt: $\operatorname{sgn}(A^k)$ ist die Matrix, die die Relation E^k repräsentiert.

16.3.2 Erste Möglichkeit für die Berechnung der Wegematrix

Um zu einem ersten Algorithmus zur Berechnung der Wegematrix zu kommen, müssen wir als letztes noch klären, welche Operation auf Matrizen „zur Vereinigung von Relationen passt“. Seien dazu auf der gleichen Menge M zwei binäre Relationen $R \subseteq M \times M$ und $R' \subseteq M \times M$ gegeben, repräsentiert durch Matrizen A und A'. Dann gilt:

$$\begin{aligned} (i,j) \in R \cup R' &\iff (i,j) \in R \vee (i,j) \in R' \\ &\iff A_{ij} = 1 \vee A'_{ij} = 1 \\ &\iff A_{ij} + A'_{ij} \geq 1 \\ &\iff (A + A')_{ij} \geq 1 \\ &\iff \operatorname{sgn}(A + A')_{ij} = 1 \end{aligned}$$

Also wird die Relation $R \cup R'$ durch die Matrix $\operatorname{sgn}(A + A')$ repräsentiert.

Aus Lemma 16.1 und dem eben aufgeführten Korollar 16.3 folgt recht schnell eine erste Formel für die Berechnung der Wegematrix:

16.4 Lemma. Es sei G ein gerichteter Graph mit Adjazenzmatrix A. Dann gilt für alle $k \geq n - 1$:

- Die Matrix $\operatorname{sgn}(\sum_{i=0}^{k} A^i)$ repräsentiert die Relation E^*.
- Mit anderen Worten:

$$W = \operatorname{sgn}\left(\sum_{i=0}^{k} A^i\right)$$

  ist die Wegematrix des Graphen G.

16.5 Beweis. Angesichts der schon erarbeiteten Ergebnisse ist es ausreichend, sich noch die beiden folgenden eher technischen Dinge zu überlegen:

- Die Vereiniung $\bigcup_{i=0}^{n-1} E^i$ wird repräsentiert durch die Matrix $\operatorname{sgn}(\sum_{i=0}^{k} \operatorname{sgn}(A^i))$.
- In dieser Formel darf man die „inneren“ Anwendungen von sgn weglassen.

Das sieht man so:

- Zum ersten Punkt genügt es, die obige Überlegung zur Matrixrepräsentation von $R \cup R'$ per Induktion auf beliebig endliche viele Relationen zu übertragen.
- Mit einer ähnlichen Heransgehensweise wie oben ergibt sich

$$\begin{aligned}
\operatorname{sgn}(\operatorname{sgn}(A) + \operatorname{sgn}(A'))_{ij} = 1 &\iff (\operatorname{sgn}(A) + \operatorname{sgn}(A'))_{ij} \geq 1 \\
&\iff \operatorname{sgn}(A)_{ij} + \operatorname{sgn}(A')_{ij} \geq 1 \\
&\iff \operatorname{sgn}(A)_{ij} = 1 \vee \operatorname{sgn}(A')_{ij} = 1 \\
&\iff A_{ij} \geq 1 \vee A'_{ij} \geq 1 \\
&\iff A_{ij} + A'_{ij} \geq 1 \\
&\iff (A + A')_{ij} \geq 1 \\
&\iff \operatorname{sgn}(A + A')_{ij} = 1
\end{aligned}$$

  Dabei haben wir in beiden Punkten benutzt, dass die Einträge in den interessierenden Matrizen nie negativ sind.

■

Das sich ergebende Verfahren ist in Algorithmus 16.2 dargestellt.

16.3.3 Zählen durchzuführender arithmetischer Operationen

Wir stellen nun ein erstes Mal die Frage danach wie aufwändig es ist, die Wegematrix eines Graphen auszurechnen. Unter Aufwand wollen hier zunächst einmal der Einfachheit halber die Anzahl benötigter arithmetischer Operationen verstehen. (Wir werden das im Laufe weiterer Einheiten noch genauer diskutieren.)

Wir beginnen mit der wohl naivsten Herangehensweise, wollen aber darauf hinweisen, dass Sie schon in diesem Fall sehen werden, dass man manchmal durch Nachdenken oder hübsche Ideen signifikante Verbesserungen erreichen kann.

Die einfachste Methode besteht darin, Algorithmus 16.2 zu benutzen und ohne viel Nachdenken zu implementieren. Das bedeutet, dass wir zur Berechung von $\operatorname{sgn}\left(\sum_{i=0}^{n-1} A^i\right)$ folgende Berechnungen (nicht unbedingt in dieser Reihenfolge) durchzuführen haben:

- n^2 Berechnungen der sgn Funktion;
- n Matrix-Additionen;
- $0 + 1 + \cdots + n - 1 = \sum_{i=1}^{n-1} i = n(n-1)/2$ Matrix-Multiplikationen;

Algorithmus 16.2: einfachster Algorithmus, um die Wegematrix zu berechnen

```
{ Matrix A sei die Adjazenzmatrix }
{ Matrix W wird am Ende die Wegematrix enthalten }
{ Matrix M wird benutzt, um A^i zu berechnen }
W ← 0                          { Nullmatrix }
for i ← 0 to n − 1 do
  M ← I                        { Einheitsmatrix }
  for j ← 1 to i do
    M ← M · A                  { Matrixmultiplikation }
  od
  W ← W + M                    { Matrixaddition }
od
W ← sgn(W)
```

Alle Matrizen haben Größe $n \times n$.

Für jeden der n^2 Einträge in einer Summenmatrix muss man eine Addition durchführen, also benötigt eine Matrixaddition n^2 arithmetische Operationen.

Für jeden der n^2 Einträge in einer Produktmatrix besteht jedenfalls die naheliegende Methode darin, eine Formel der Form $\sum_{k=0}^{n-1} a_{ik} b_{kj}$ auszuwerten. (Tatsächlich gibt es andere Möglichkeiten, wie wir in nachfolgenden Abschnitten sehen werden!) Das sind jeweils n Multiplikationen und $n-1$ Additionen. Insgesamt ergeben sich so $2n^3 - n^2$ Operationen.

Für die Berechnung der Wegematrix nach dieser Methode kommt man so auf

$$\begin{aligned} & n^2 + n^2 \cdot n + (2n^3 - n^2)n(n-1)/2 \\ =& n^2 + n^3 + (2n^3 - n^2)(n^2 - n)/2 \\ =& n^2 + n^3 + (2n^5 - 2n^4 - n^4 + n^3)/2 \\ =& n^5 - \frac{3}{2}n^4 + \frac{3}{2}n^3 + n^2 \end{aligned}$$

Operationen. Wenn z. B. $n = 1000$ ist, dann sind das immerhin 998 501 501 000 000 Operationen, also „fast“ $n^5 = 10^{15}$.

16.3.4 Weitere Möglichkeiten für die Berechnung der Wegematrix

Kann man die Wegematrix auch mit „deutlich“ weniger Operationen berechnen? Vielleicht haben Sie eine Möglichkeit schon gesehen: Wir haben weiter vorne so getan, als müsste man für die Berechnung von A^i immer $i-1$ Matrixmultiplikationen durchführen. Da aber der Reihe nach *alle* Potenzen A^i berechnet werden, ist das nicht so. Man merkt sich einfach immer das alte A^{i-1} und braucht dann nur *eine* Matrixmultiplikation, um zu A^i zu gelangen. Diese Vorgehensweise ist in Algorithmus 16.3 dargestellt. Damit ergeben sich insgesamt nur n Matrixmultiplikationen statt $n(n-1)/2$ und die Gesamtzahl arithmetischer Operationen sinkt von $n^5 - (3/2)n^4 + (3/2)n^3 + n^2$ auf $2n^4 + n^2$. Für $n = 1000$ sind das 2 000 001 000 000, also „ungefähr“ 500 mal weniger als mit Algorithmus 16.2.

Algorithmus 16.3: verbesserter Algorithmus, um die Wegematrix zu berechnen

```
{ Matrix A sei die Adjazenzmatrix }
{ Matrix W wird am Ende die Wegematrix enthalten }
{ Matrix M wird benutzt um A^i zu berechnen }
W ← 0                          { Nullmatrix }
M ← I                          { Einheitsmatrix }
for i ← 0 to n − 1 do
  W ← W + M                    { Matrixaddition }
  M ← M · A                    { Matrixmultiplikation }
od
W ← sgn(W)
```

Wenn man einmal unterstellt, dass jede Operation gleich lange dauert, dann erhält man also Ergebnisse um etwa einen Faktor $n/2$ schneller.

Und es geht noch schneller: statt n Matrixmultiplikationen wollen wir nun mit $\log_2 n$ von ihnen auskommen. Hierfür nutzen wir die Beobachtung aus, deren explizite Erwähnung in Lemma 16.1 Sie vielleicht ein bisschen gewundert hat:

$$\forall k \geq n-1 : E^* = \bigcup_{i=0}^{k} E^i$$

Statt $n-1$ wählen wir nun die nächstgrößere Zweierpotenz $k = 2^{\lceil \log_2 n \rceil}$. Außerdem benutzen wir noch einen Trick, der es uns erlaubt, statt $\bigcup_{i=0}^{k} E^i$ etwas ohne viele

Vereinigungen hinzuschreiben. Dazu betrachten wir $F = E^0 \cup E^1 = \mathrm{I}_V \cup E$. Unter Verwendung der Rechenregel (die Sie sich bitte klar machen) für Relationen

$$(A \cup B) \circ (C \cup D) = (A \circ C) \cup (A \circ D) \cup (B \circ C) \cup (B \circ D)$$

ergibt sich

$$F^2 = (E^0 \cup E^1) \circ (E^0 \cup E^1) = E^0 \cup E^1 \cup E^1 \cup E^2 = E^0 \cup E^1 \cup E^2$$

Daraus folgt

$$\begin{aligned} F^4 = (F^2)^2 &= (E^0 \cup E^1 \cup E^2) \circ (E^0 \cup E^1 \cup E^2) \\ &= \dots \\ &= E^0 \cup E^1 \cup E^2 \cup E^3 \cup E^4 \end{aligned}$$

und durch Induktion sieht man, dass für alle $m \in \mathbb{N}_0$ gilt:

$$F^{2^m} = \bigcup_{i=0}^{2^m} E^i$$

Wenn man einfach zu Beginn die Matrix für $F = E^0 + E$ berechnet und sie dann so oft quadriert, dass nach m-maligen Durchführen $2^m \geq n - 1$ ist, hat man das gewünschte Ergebnis. Offensichtlich genügt $m = \lceil \log_2 n \rceil$.

Im Matrizenrechnung übersetzt ergeben sich

$$2n^2 + \lceil \log_2 n \rceil (2n^3 - n^2)$$

arithmetische Operationen, was gegenüber $2n^4 + n^2$ wieder eine beträchtliche Verbesserung ist, nämlich ungefähr um einen Faktor $2n/\lceil \log_2 n \rceil$.

16.4 ALGORITHMUS VON WARSHALL

Wir kommen nun zu einem Algorithmus zur Berechnung der Wegematrix eines Graphen, bei dem gegenüber der eben zuletzt behandelten Methode der Faktor $\log_2 n$ sogar auf eine (kleine) Konstante sinkt. Er stammt von Warshall (1962) und ist in Algorithmus 16.4 dargestellt.

Zum besseren Verständnis sei als erstes darauf hingewiesen, dass für zwei Bits x und y das Maximum $\max(x, y)$ dem logischen Oder entspricht, wenn man 1 als wahr und 0 als falsch interpretiert. Analog entspricht das Minimum $\min(x, y)$ dem logischen Und.

Algorithmus 16.4: Berechnung der Wegematrix nach Warshall

for $i \leftarrow 0$ **to** $n-1$ **do**
  for $j \leftarrow 0$ **to** $n-1$ **do**
    $W_{ij} \leftarrow \begin{cases} 1 & \text{falls } i = j \\ A_{ij} & \text{falls } i \neq j \end{cases}$
  od
od
for $k \leftarrow 0$ **to** $n-1$ **do**
  for $i \leftarrow 0$ **to** $n-1$ **do**
    for $j \leftarrow 0$ **to** $n-1$ **do**
      $W_{ij} \leftarrow \max(\, W_{ij}, \min(W_{ik}, W_{kj})\,)$
    od
  od
od

Den Aufwand dieses Algorithmus sieht man schnell. Für die Initialisierung der Matrix W im ersten Teil werden n^2 Operationen benötigt. Die weitere Rechnung besteht aus drei ineinander geschachtelten **for**-Schleifen, von denen jede jedes Mal n mal durchlaufen wird. Das ergibt n^3-malige Ausführung des Schleifenrumpfes, bei der jedes Mal zwei Operationen durchgeführt werden.

Weitaus weniger klar dürfte es für Sie sein, einzusehen, warum der Algorithmus tatsächlich die Wegematrix berechnet. Es stellt sich hier mit anderen Worten wieder einmal die Frage nach der *Korrektheit* des Algorithmus.

Die algorithmische Idee, die hier im Algorithmus von Warshall benutzt wird, geht auf eine fundamentale Arbeit von Stephen Kleene (1956) zurück, der sie im Zusammenhang mit *endlichen Automaten* und *regulären Ausdrücken* benutzt hat. (Unter anderem wird in dieser Arbeit die Schreibweise mit dem hochgestellten Stern * für den Konkatenationsabschluss eingeführt, der deswegen auch Kleene-Stern heißt.) Auf diese Themen werden wir in einer späteren Einheit eingehen.

Für den Nachweis der Korrektheit des Algorithmus von Warshall besteht die Hauptaufgabe darin, eine Schleifeninvariante für die äußerste (Laufvariable k) der drei ineinander geschachtelten Schleifen zu finden. Für die Formulierung ist es hilfreich, bei einem Pfad $p = (v_0, v_1, \ldots, v_{m-1}, v_m)$ der Länge $m \geq 2$ über die Knoten $v_1, \ldots, v_{m-1}$ reden zu können. Wir nennen sie im folgenden die *Zwischenknoten* des Pfades. Pfade der Längen 0 und 1 besitzen keine Zwischenknoten. Hier ist nun die Schleifeninvariante:

16.6 Lemma. Für alle $i, j \in \mathbb{Z}_n$ gilt: Nach k Durchläufen der äußeren Schleife des Algorithmus von Warshall ist $W[i,j]$ genau dann 1, wenn es einen wiederholungsfreien Pfad von i nach j gibt, bei dem alle Zwischenknoten Nummern in $\mathbb{Z}_k$ haben (also Nummern, die echt kleiner als k sind).

Hat man erst einmal nachgewiesen, dass das tatsächlich Schleifeninvariante ist, ist die Korrektheit des gesamten Algorithmus schnell bewiesen. Denn dann gilt insbesondere nach Beendigung der Schleife, also nach n Schleifendurchläufen:

- Für alle $i, j \in \mathbb{Z}_n$ gilt: Nach n Schleifendurchläufen ist W_{ij} genau dann 1, wenn es einen wiederholungsfreien Pfad von i nach j gibt, bei dem alle Zwischenknoten Nummern in $\mathbb{Z}_n$ haben, wenn also überhaupt ein Pfad existiert (denn andere Knotennummern gibt es gar nicht).

16.7 Beweis. (von Lemma 16.6)

Induktionsanfang: Dass die Behauptung im Fall $k = 0$ gilt, ergibt sich aus der Initialisierung der Matrix W: Knoten mit Nummern echt kleiner als 0 gibt es nicht; in den zur Rede stehenden Pfaden kommen also keine Knoten außer dem ersten und dem letzten vor. Das bedeutet aber, dass die Pfade von einer der Formen (x) oder (x, y) sein müssen.

Für den Induktionsschritt sei $k > 0$ beliebig aber fest und wir treffen die

Induktionsvoraussetzung: Für alle $i, j \in \mathbb{Z}_n$ gilt: Nach $k-1$ Durchläufen der äußeren Schleife des Algorithmus von Warshall ist W_{ij} genau dann 1, wenn es einen wiederholungsfreien Pfad von i nach j gibt, bei dem alle Zwischenknoten Nummern haben, die in $\mathbb{Z}_{k-1}$ sind.

Induktionsschluss: Wir bezeichnen mit $W^{[k]}$ die Matrix, wie sie nach k Schleifendurchläufen berechnet wird, und analog mit $W^{[k-1]}$ die Matrix nach $k-1$ Schleifendurchläufen. Die beiden Implikationen werden getrennt bewiesen:

$\Longrightarrow$: Es sei $W_{ij}^{[k]} = 1$. Dann hat also mindestens eine der folgenden Bedingungen zugetroffen:

- $W_{ij}^{[k-1]} = 1$: In diesem Fall existiert ein Pfad, dessen Zwischenknoten alle Nummern in $\mathbb{Z}_{k-1}$ haben, und das ist auch einer, dessen Zwischenknoten alle Nummern in $\mathbb{Z}_k$ haben.
- $W_{ik}^{[k-1]} = 1$ und $W_{kj}^{[k-1]} = 1$. Dann existieren Pfade von i nach k und von k nach j, deren Zwischenknoten alle Nummern in $\mathbb{Z}_{k-1}$ sind. Wenn man die Pfade zusammensetzt, erhält man einen Pfad von i nach j, dessen Zwischenknoten alle Nummern in $\mathbb{Z}_k$ haben. Durch Entfernen von Zyklen kann man auch einen entsprechenden Pfad konstruieren, der wiederholungsfrei ist.

$\Longleftarrow$: Es gebe einen wiederholungsfreien Pfad p von i nach j, dessen Zwischenknoten alle Nummern in $\mathbb{Z}_k$ haben. Dann sind zwei Fälle möglich:

- Ein Zwischenknoten in p hat Nummer $k-1$:
  Da p wiederholungsfrei ist, enthält das Anfangsstück von p, das von i nach $k-1$ führt, nicht $k-1$ als Zwischenknoten, also nur Knotennummern in $\mathbb{Z}_{k-1}$. Das gleiche gilt für das Endstück von p, das von $k-1$ nach j führt. Nach Induktionsvoraussetzung sind also $W_{i,k-1}^{[k-1]} = 1$ und $W_{k-1,j}^{[k-1]} = 1$. Folglich wird im k-ten Durchlauf $W_{ij}^{[k]} = 1$ gesetzt.
- Kein Zwischenknoten in p hat Nummer $k-1$:
  Dann sind die Nummern der Zwischenknoten alle in $\mathbb{Z}_{k-1}$. Nach Induktionsvoraussetzung ist folglich $W_{ij}^{[k-1]} = 1$ und daher auch $W_{ij}^{[k]} = 1$.

■

16.5 AUSBLICK

Wir sind in dieser Einheit davon ausgegangen, dass die Matrizen auf die naheliegende Weise miteinander multipliziert werden. In der nächsten Einheit werden wir sehen, dass es auch andere Möglichkeiten gibt, die in einem gewissen Sinne sogar besser sind. Dabei werden auch Möglichkeiten zur Quantifizierung von „gut“, „besser“, usw. Thema sein.

Effiziente Algorithmen für Problemstellungen bei Graphen sind nach wie vor Gegenstand intensiver Forschung. Erste weiterführende Aspekte werden Sie im kommenden Semester in der Vorlesung „Algorithmen 1“ zu sehen bekommen.

LITERATUR

Kleene, Stephen C. (1956). “Representation of Events in Nerve Nets and Finite Automata”. In: *Automata Studies*. Hrsg. von Claude E. Shannon und John McCarthy. Princeton University Press. Kap. 1, S. 3–40.
Eine Vorversion ist online verfügbar; siehe `http://www.rand.org/pubs/research_memoranda/2008/RM704.pdf` (8.12.08) (siehe S. 160).

Warshall, Stephen (1962). “A Theorem on Boolean Matrices”. In: *Journal of the ACM* 9, S. 11–12 (siehe S. 159).