Relationen und Graphentheorie

Grundbegriffe. Relationen- und Graphentheorie gehören zu den wichtigsten Hilfsmitteln der Informatik, die aus der diskretenMathematik stammen.
Ein Graph setzt sich aus einer Menge von *Ecken E* und einer Menge von *Pfeilen* (*Kanten*) *K* zusammen. Jede Kante ist dabei durch zwei Ecken ("Endpunkte") definiert. Die Objekte, die in der Ecken- und Pfeilmenge enthalten sind, müssen nichts mit geometrischen Objekten zu tun haben. Vielmehr ist die Graphentheorie ein Hilfsmittel zur Beschreibung beliebiger Beziehungen zwischen Objekten beliebigen Typs. Aufgaben, die mit der Graphentheorie gelöst werden können, sind z.B.:

- Ist in einem System von vernetzten Rechnern Rechner *B* auch dann noch von Rechner *A* aus erreichbar, wenn eine der Netzwerkverbindungen gekappt wird (Erreichbarkeitsproblem)?
- Suche in einem Straßennetz die kürzeste Verbindung von *A* nach *B* (Wegsuche).
- Ermittle die minimale Bauzeit eines Bauprojektes (Terminplanung).

Im einzelnen unterscheidet man sogenannte *ungerichtete* Graphen, d.h. Graphen, in denen die Richtung der Pfeile (z.B. von Ecke *a* nach Ecke *b* oder umgekehrt) uninteressant ist, von *gerichteten* Graphen (*Digraphen=directed graphs*). Die Richtung der Pfeile spielt z.B. bei der Beschreibung eines Straßennetzes, das auch Einbahnstraßen enthält, eine Rolle. Als Beispiele gerichteter Graphen haben wir bereits Binärbäume, Quad- und Octrees kennengelernt.

Graphen werden *anschaulich* durch Linienzüge dargestellt. In vielen Anwendungen ist es notwendig, die *Kanten* eines Graphen zu *bewerten*. Beispielsweise setzt sich ein Rohrleitungsnetz im allgemeinen nicht aus lauter Rohren gleichen Durchmessers zusammen. Der Rohrdurchmesser spielt also eine wichtige Rolle als *Attribut* der Kanten. Viele Graphenalgorithmen verwenden solche *Attribute* zu Kanten. Man spricht von *bewerteten* Graphen.

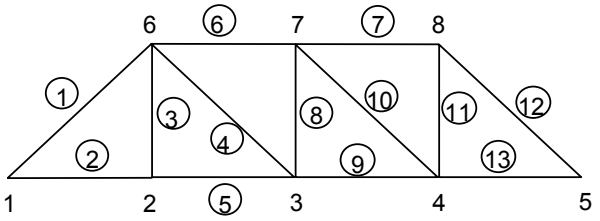


Bild 2.16: Beispiel eines Graphen. Die Knoten sind durch einfache Nummern, die Pfeile durch Nummern in Kreisrahmung gekennzeichnet.

Im *Rechner* können Graphen durch verschiedene *Datenstrukturen* beschrieben werden. Im einfachsten Fall läßt sich ein Graph durch eine *Matrix* (sogenannte *Adjazenzmatrix*) beschreiben, in der jedem Knoten eine Spalte und eine Zeile zugeordnet ist.

In Bild 2.16 ist als Beispiel ein Graph abgebildet. Seine Darstellung als *Adjazenzmatrix* sieht aus wie folgt:

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

3		1	1	1		1	1	
4			1	1	1		1	1
5				1	1			1
6	1	1	1			1	1	
7			1	1		1	1	1
8				1	1		1	1

Alle Felder der Matrix, die leer gelassen sind, sind mit einer Null besetzt. Alle Felder, die eine 1 enthalten, stellen eine Verknüpfung zwischen den beiden zugehörigen Knoten her. Der in Bild 2.16 dargestellte Graph ist ungerichtet, so daß jede Kante zweimal auftaucht, jeweils in beiden Richtungen. Die Adjazenzmatrix eines ungerichteten Graphen ist, wie man hier sieht, stets symmetrisch. Es würde also ausreichen, nur die obere oder untere Hälfte der Matrix zu speichern.

Im Fall eines bewerteten Graphen könnte man anstelle der Einsen die Bewertung der Pfeile in die Adjazenzmatrix eintragen, z.B. die Längen der Wege zwischen den Knoten.

Bild 2.17: Beispiel zum Erreichbarkeitsproblem.

Die Adjazenzmatrix läßt eine sehr einfache Lösung des Erreichbarkeitsproblems zu: Wenn man für einen Graphen mit einer Knotenanzahl v die Adjazenzmatrix A aufstellt, so stellt die v-te Potenz dieser Matrix die Adjazenzmatrix der *transitiven Hülle* des Graphen dar. In der Adjazenzmatrix der transitiven Hülle sind alle Einträge ungleich Null, die einer direkten oder *indirekten* Verbindung zwischen zwei Knoten entsprechen. In unserem in Bild 2.16 gezeigten Graphen sind alle Knoten indirekt mit jedem anderen Knoten verbunden. Die Adjazenzmatrix der transitiven Hülle enthält daher keine Nullen mehr. Bei einem solch kleinen Beispiel kann man das noch recht schnell direkt sehen. Für die allgemeine Untersuchung solcher Fragestellungen bei Graphen mit einer großen Anzahl von Ecken und Pfeilen ist die schematische Berechnung der transitiven Hülle ein wichtiges Hilfsmittel. Die Berechnung über die v-te Potenz der Adjazenzmatrix ist allerdings nicht sehr effizient, da sie den Aufwand v^4 verursacht. Es gibt jedoch einen Algorithmus, der diese Aufgabe in weniger als v^3 Schritten löst.

Wir wollen diesen Algorithmus anhand des Beispiels in Bild 2.17 darstellen. Wir wollen alle Knoten j – von 1 bis 4 – der Reihe nach untersuchen. Wir schreiben jeweils die Knoten i an, von denen aus man zu Knoten j kommen kann, und alle Knoten k, die von j aus erreichbar sind. Offenkundig sind diese beiden Knotenmengen indirekt über Knoten j verbunden. Unsere Analyse läuft in folgenden Schritten ab: Zunächst untersuchen wir Knoten 1. Wir finden:

Knoten 1:

erreichbar von:	verbunden mit:

4	2

Also können wir nunmehr auch die Verbindung von 4 nach 2 in unseren Graphen aufnehmen. Er sieht damit aus wie folgt (Bild 2.18):

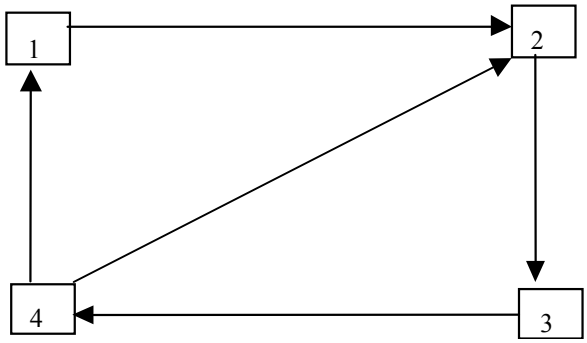


Bild 2.18: Schrittweise Ermittlung der transitiven Hülle des Graphen aus Bild 2.17. Zustand nach Untersuchung von Knoten 1.

Nun gehen wir weiter zu Knoten 2. Dort finden wir nunmehr:

Knoten 2:

erreichbar von:	verbunden mit:
1	3
4	

Wir können also auch die Pfeile 1-3 und 4-3 in unseren Graphen eintragen. Er erhält damit folgende Gestalt (Bild 2.19):

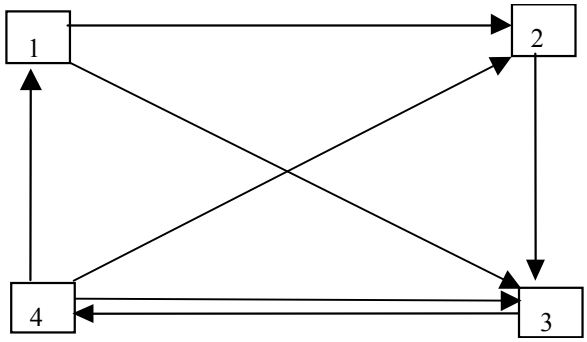


Bild 2.19: Schrittweise Ermittlung der transitiven Hülle des Graphen aus Bild 2.17. Zustand nach Untersuchung von Knoten 2.

Nun gehen wir weiter zu Knoten 3. An dieser Stelle können wir für unseren erweiterten Graphen nun feststellen:

Knoten 3:

erreichbar von:	verbunden mit:
1	4
2	
4	

Das erbringt uns die neuen Pfeile 1-4 und 2-4. Der Pfeil 4-4 entspricht der 1 auf der Diagonale der zugehörigen Adjazenzmatrix. Unser Graph hat nun folgende Gestalt (Bild 2.20):

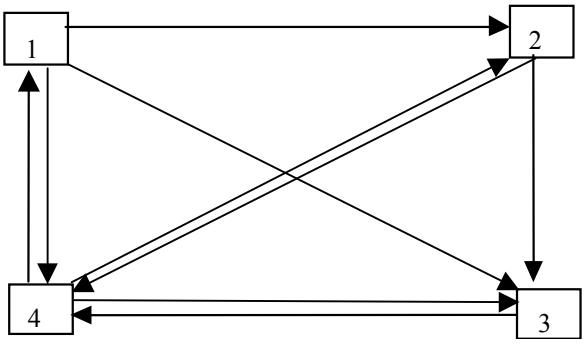


Bild 2.20: Schrittweise Ermittlung der transitiven Hülle des Graphen aus Bild 2.17. Zustand nach Untersuchung von Knoten 3.

Schließlich müssen wir auch noch Knoten 4 untersuchen. Dort finden wir:

Knoten 4:

erreichbar von:	verbunden mit:
1	1
2	2
3	3

Damit ergeben sich noch die neuen Pfeile 2-1, 3-1 und 3-2, so daß sich die transitive Hülle unseres Graphen als voll besetzter Graph, in dem jeder Knoten mit jedem anderen verbunden ist, herausstellt.

Die Vorgehensweise, die wir dargestellt haben, läßt sich mit Hilfe der Adjazenzmatrix *A* in folgendem Algorithmus wiedergeben:

```
for j=1 to Knotenanzahl
   for i=1 to Knotenanzahl
     if A(i,j)=1 then
       for k=1 to Knotenanzahl
         if A(j,k)=1 then A(i,k)=1
       next k
     end if
   next i
next j
```

Wie man leicht einsieht, benötigt dieser Algorithmus infolge der drei geschachtelten Schleifen, deren innerste aber nicht jedesmal durchlaufen wird, maximal (`Knotenanzahl`)³ Operationen. Die äußerste Schleife entspricht der Untersuchung aller Knoten. Die nächste Schleife sucht alle jene Knoten heraus, die von Knoten *j* aus erreichbar sind, ermittelt also jeweils die 1. Spalte unserer Tabellen. Die innerste Schleife schließlich ermittelt die rechte Spalte unserer Tabellen und führt die neuen Verbindungen ("Abkürzungen" von *i* nach *k*) ein.

Für Graphen, die zwar viele Knoten, aber nur wenige Pfeile enthalten, ist die Adjazenzmatrix keine besonders effiziente Datenstruktur. Die Darstellung eines Graphen als Adjazenzmatrix hat aber den Vorteil, daß man sehr einfach alle von einem Knoten ausgehenden und vor allem auch die an einem Knoten endenden ("*inzidenten*") Pfeile ermitteln kann. Letzteres ist für unseren Algorithmus "transitive Hülle" entscheidend.

Einige andere wichtige Datenstrukturen für Graphen sind zwar wesentlich weniger speicherintensiv, lassen aber dafür keine so einfache Ermittlung inzidenter Pfeile zu.

Sehen wir uns zunächst die Darstellung eines Graphen als *Kantenliste* an (Bild 2.21). Der Graph hat dieselbe Anzahl von Knoten und Pfeilen wie der in Bild 2.16, ist aber nun ein *gerichteter* Graph.

Kante	von	nach
1	1	6
2	1	2
3	2	6
4	3	6
5	2	3
6	6	7
7	7	8
8	3	7
9	3	4
10	4	7
11	4	8
12	5	8
13	4	5

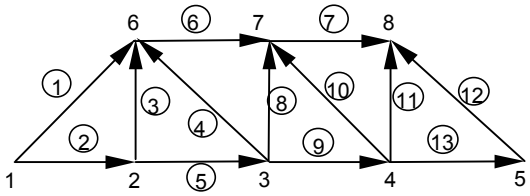
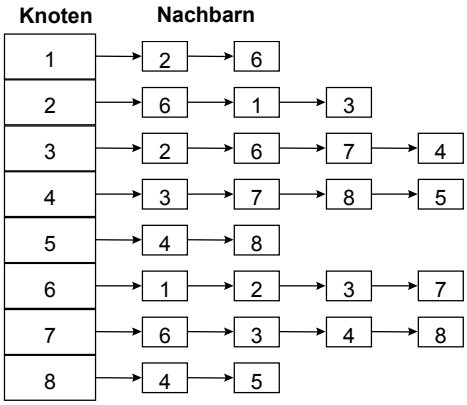


Bild 2.21: Darstellung eines Graphen durch eine Kantenliste.

Alle Kanten sind in einer Liste bzw. Tabelle aufgeführt. Jede Kante verweist per Nummer auf eine Anfangs- und Endecke. Die Nachbarschaftsrelationen des Graphen sind durch diese Darstellung in einer einzigen Liste vollständig beschrieben. Auch die geometrischen und sonstigen Zusatzattribute der Kante (Form, Strichstärke, Linienart, Farbe) können wir durch Hinzufügen weiterer Spalten zur Kantentabelle in dieser Datenstruktur unterbringen. Wir wollen jedoch auch noch geometrische Attribute zu den *Ecken* speichern, z.B. Koordinaten. Daher benötigen wir in der Regel wie bei der Datenstruktur Adjazenzliste noch ein zweites Feld, in dem diesmal die Knoten nach Nummer sortiert mit ihren Attributen aufgeführt sind.

Die Speicherung des Graphen als Kantenliste hat den Nachteil, daß es recht aufwendig ist, die von einem Knoten ausgehenden Pfeile bzw. die in einem Knoten inzidenten Pfeile zu ermitteln. Für Wegalgorithmen oder Erreichbarkeitsprobleme ist diese Datenstruktur also nicht besonders günstig.



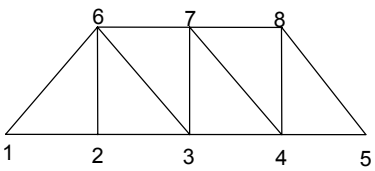


Bild 2.22: Darstellung eines Graphen durch eine Adjazenzliste (Nachbarschaftsliste).

Eine Datenstruktur, die den enormen Speicheraufwand der Adjazenzmatrix vermeidet und dennoch einen effizienten Zugriff auf einen Teil der Nachbarschaftsinformation gestattet, ist die *Nachbarschaftsliste* (*Adjazenzliste*, Bild 2.22). Dabei verwaltet jede Ecke des Graphen eine Liste aller Ecken, mit denen sie direkt durch eine Kante verbunden ist. In unserem Bild haben wir wiederum einen ungerichteten Graphen gewählt. In diesem Fall kommt jede Kante doppelt vor, einmal als Nachbarschaftsbeziehung von Ecke *a* nach Ecke *b* und einmal als Verbindung von *b* nach *a*.

Im Gegensatz zur Darstellung "Kantenliste" macht die Datenstruktur "Adjazenzliste" die Ermittlung der von einem Knoten ausstrahlenden Pfeile recht einfach. Die Ermittlung inzidenter Pfeile ist allerdings sogar noch aufwendiger als bei der Kantenliste.

Je nach Einsatzzweck des Graphen muß man eine geeignete Darstellungsform wählen; ein generelles Rezept läßt sich nicht angeben.