Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

BRIGITTE LUTZ-WESTPHAL[1]

Die Mathematik der kürzesten Wege. Inhalte und Methoden für den Unterricht

[1]Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB), Takustraße 7, D-14195 Berlin, lutz-westphal@zib.de. *Arbeitsplatzadresse:* Technische Universität Berlin, Fakultät II - Mathematik und Naturwissenschaften, Institut für Mathematik, Sekr. MA 6-2, Straße des 17. Juni 136, D-10623 Berlin, westphal@math.tu-berlin.de.

ZIB-Report 04-36 (August 2004)

Brigitte Lutz-Westphal
(TU Berlin/ZIB/DFG-Forschungszentrum „Mathematik für Schlüsseltechnologien“)

Die Mathematik der kürzesten Wege. Inhalte und Methoden für den Unterricht

1 Wege mit der U-Bahn

Welches ist der optimale Weg? Diese Frage begleitet uns alltäglich: Beispielsweise haben Sie sicherlich versucht, den Weg zum Kauf dieses Buches optimal zu wählen. Sind sie zum Buchladen gegangen oder gefahren? Welchen Buchladen haben Sie ausgewählt? Denjenigen der die kürzeste Luftlinienentfernung hat oder den, der direkt an der Bushaltestelle liegt? Welches Verkehrsmittel haben Sie benutzt? Sind Sie sicher, die beste Wegroute zu kennen? Oder haben Sie das Buch im Internet bestellt? War das wirklich weniger zeitaufwändig und kostengünstiger, als in den Buchladen zu gehen? Haben Sie gewusst, dass auch die Datenströme im Internet mit Hilfe von kürzeste-Wege-Berechnungen gelenkt werden?

Fragen über Fragen tauchen auf. Das ist typisch für Optimierungsprobleme. Je länger man über solche Probleme nachdenkt, um so mehr Varianten fallen einem ein, und um so deutlicher wird es, dass diese Fragestellungen alles andere als eindeutig sind. Beobachten Sie sich selbst einmal, wie oft und in welchen Varianten Sie über gute oder optimale Wege nachdenken. Eine der folgenden Fragen haben Sie sich sicherlich in letzter Zeit auch gestellt: Wie komme ich morgens am schnellsten zur Schule oder zum Arbeitsplatz? Welche Zugverbindung ist die günstigste? Meint „günstig“ hier „nicht teuer“ oder „kurze Fahrzeit“? Wie plane ich die Route für die Autofahrt in den Urlaub? Möchte ich schöne, schnelle oder kurze Strecken fahren? Welche U-Bahn-Verbindung nehme ich? Die mit wenig Umsteigen oder die mit der etwas kürzeren Fahrzeit?

Um die Suche nach optimalen Wegen mathematisch fassen zu können, müssen wir uns zunächst eine präzise Fragestellung überlegen. Dann können wir ein mathematisches Modell erstellen und Lösungswege suchen.

Schauen wir uns einmal einen U-Bahn-Netzplan an (Abb. 1). Wie kommt man am besten von „Richard-Wagner-Platz“ nach „Köpenick“? Versuchen Sie eine gute Verbindung zu finden! Wie gehen Sie dabei vor?

Vermutlich haben Sie angefangen, Stationen zu zählen und haben sich dann mit einer Verbindung, die Ihnen recht kurz („kurz“ im Sinne von „kleine Anzahl der durchfahrenen Stationen“) erschien, zufrieden gegeben. Woher haben wir die Garantie, dass es nicht eine noch kürzere Verbindung gibt, wenn man z. B. erst einmal in die entgegengesetzte Richtung fährt? Der Weg mit den wenigsten durchfahrenen Stationen zwischen A und B führt im zweiten Beispiel (Abb. 2) ganz außenherum. Haben Sie bei der Suche nach einer guten Verbindung zwischen Richard-Wagner-Platz und Adlershof auch solche (Um-) Wege in Betracht gezogen?

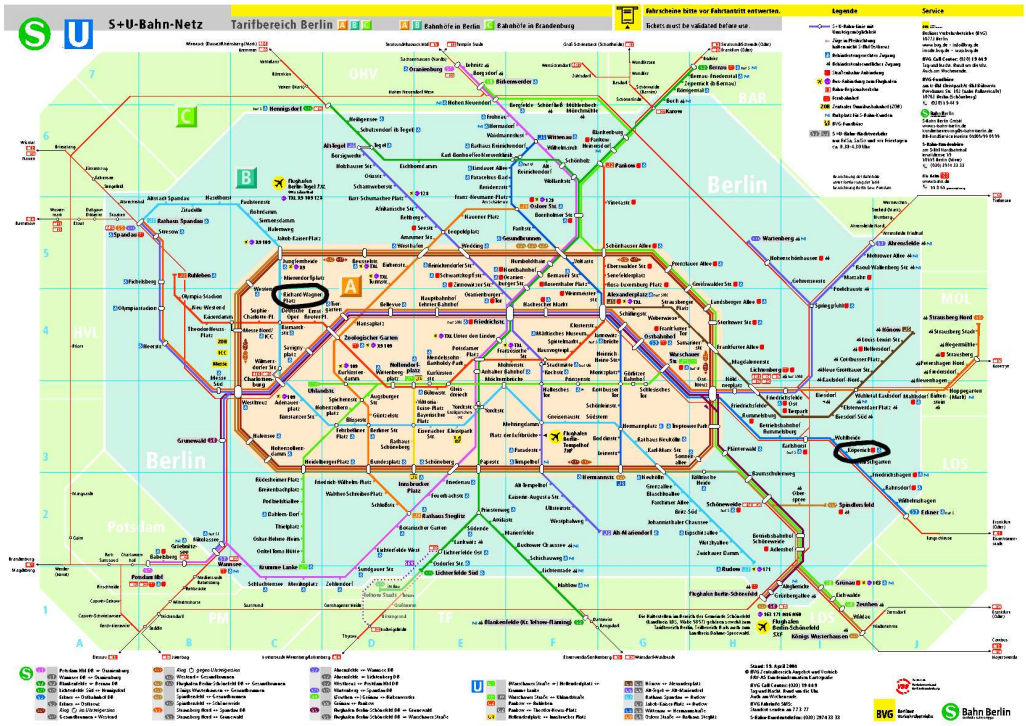


Abbildung 1: Das Berliner S- und U-Bahnnetz

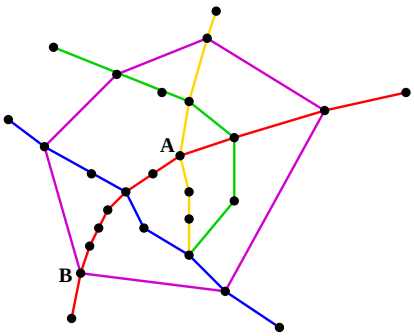


Abbildung 2: Ein anderer Liniennetzplan

2 Enumeration

Um sicher zu gehen, dass man keine günstige Verbindung übersieht, müsste man also alle möglichen Wege zwischen Start und Ziel anschauen und dann einen kürzesten (es kann mehrere gleich kurze geben) auswählen.

Wie viele verschiedene A(Start)-B(Ziel)-Wege gibt es denn? Leider lässt sich diese Zahl nicht immer einfach bestimmen. In unserem Beispiel mit dem U- und S-Bahn-Plan wäre selbst eine Abschätzung sehr kompliziert.

An dem Beispiel[2] in Abb. 3 kann man sehen, dass schon bei wenigen „Stationen“ eine gigantisch große Anzahl von A-B-Wegen entstehen kann.

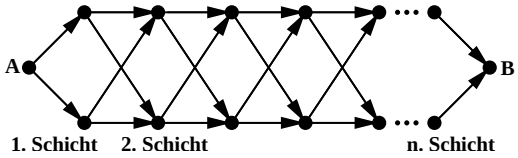


Abbildung 3: Ein Beispiel, das sich gut abzählen lässt

Von A aus hat man zwei Möglichkeiten loszufahren. Von beiden Knoten der zweiten Schicht aus wieder jeweils zwei, also insgesamt 4 verschiedene Wege. Mit jeder weiteren Schicht verdoppelt sich die Anzahl der Wege. Bei 10 Schichten, also 22 Knoten, hat man $2^{10} = 1024$ Wege, bei 20 Schichten sind es 42 Knoten, aber bereits mehr als 1 Million A-B-Wege!

Definition: Das Phänomen der ins Unermessliche wachsenden Kombinationen bei recht kleinen Grundmengen nennt man die KOMBINATORISCHE EXPLOSION.

(Bild: Klaus Staeck)

Nimmt man nun an, dass ein Computer 1 000 000 solcher Wege in einer Sekunde berechnen kann, so braucht er bei 25 Schichten etwa eine halbe Minute, bei 30 Schichten knappe 18 Minuten, bei 40 Schichten fast 13 Tage und bei 50 Schichten bereits mehr als 35 Jahre!

[2] aus Gritzmann/Brandenberg: Das Geheimnis des kürzesten Weges, Springer 2002

Das Verfahren der Enumeration:

1. Bestimme sämtliche verschiedene Möglichkeiten.
2. Wähle eine optimale Lösung.

Die Enumeration arbeitet korrekt, ist aber in der Praxis aufgrund der kombinatorischen Explosion und den damit verbundenen unglaublich langen Rechenzeiten außer bei sehr kleinen Beispielen nicht anwendbar.

Das Beispiel in Abb. 3 sieht einem Verkehrsnetzplan nicht gerade ähnlich. Es hat aber gegenüber realistischeren Beispielen den Vorteil, dass darin leicht abgezählt werden kann. Im allgemeinen Fall ist es nämlich nicht möglich, eine so kurze Formel für die Anzahl der A-B-Wege aufzustellen.

3 Breitensuche

3.1 Entwicklung des Algorithmus

Am nächsten Beispiel (Abb. 4) wollen wir nun ein geschickteres Verfahren entwickeln.

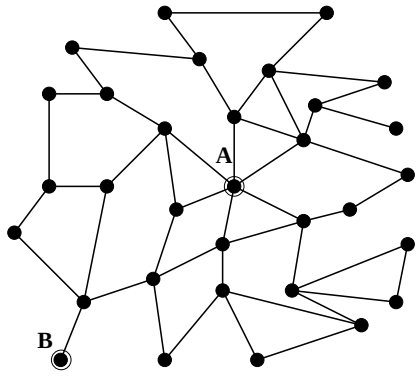


Abbildung 4: Ein Graph

Zur präziseren Verständigung definieren wir:

Definitionen:

- Ein GRAPH besteht aus KNOTEN und KANTEN, wobei eine Kante stets in einem Knoten beginnt und in einem (nicht notwendigerweise anderen) Knoten endet.
- Zwei Knoten, die durch eine Kante verbunden sind, heißen NACHBARN oder BENACHBART.
- Ein geschlossener Kantenzug heißt KREIS.
- Ein WEG ist ein Kantenzug ohne Kreise.

Nehmen wir also an, wir wollen von A nach B gelangen. Wir suchen einen Weg, der eine minimale Anzahl von Kanten benötigt und ein Verfahren, das solche Wege in beliebigen Beispielen konstruieren kann. Solch ein Verfahren nennt man einen ALGORITHMUS.[3]

Wir befinden uns nun also auf dem Knoten A. Wie würden Sie nun anfangen, einen kürzesten Weg nach B zu suchen, wenn Sie keine Ahnung haben, in welcher Richtung sich B befindet? Einfach in irgendeine Richtung losfahren und hoffen, dass man B erreicht, ist sicherlich keine gute Lösung. Zwar findet man evtl. einen Weg nach B, aber höchstwahrscheinlich nicht einen kürzesten.

Eine sinnvoller Ansatz ist, herauszufinden, welche Knoten um eine Kantenlänge vom Startknoten entfernt sind, welche zwei Kantenlängen usw. Da uns nur irgendein kürzester Weg interessiert, nehmen wir bei verschiedenen gleichlangen Wegen zum selben Knoten immer die zuerst aufgefundene Möglichkeit. Außerdem sollten wir es ausschließen, dass Kreise entstehen. Ein kürzester Weg kann keinen Kreis enthalten, da man einen Kreis einfach aus solch einem Weg herausschneiden könnte und so einen kürzeren Weg als den bisherigen erhalten würde.

Führt man die eben skizzierte Idee mit Papier und Farbstiften aus, so entsteht ein Bild wie in Abbildung 5.

Wie aber formuliert man diese Idee nun als Schritt-für-Schritt-Anweisung? Die Vorstellung, dass jemand, der keine Ahnung hat und gerade zur Türe hereinkommt, die Anweisung durchführen können soll, hilft hier enorm. Zudem ist es nützlich, sich klarzumachen, dass ein Computer den Graphen nicht „sehen“ kann. Er „weiß“ lediglich für jeden Knoten, welche Nachbarknoten er hat (siehe Abschnitt „Froschperspektive“).

Ein erster Versuch könnte so aussehen: Gehe vom Startknoten zu allen seinen Nachbarn. Gehe von diesen Nachbarn weiter zu deren Nachbarn usw. Leider funktioniert es so noch nicht. Was geht hier schief? Insbesondere werden Kreise entstehen, da nicht zwischen bereits besuchten und unbesuchten Knoten unterschieden wird. Zudem sollte man sich

[3] Beispiele für Algorithmen sind z. B. Bedienungsanleitungen oder Kochrezepte

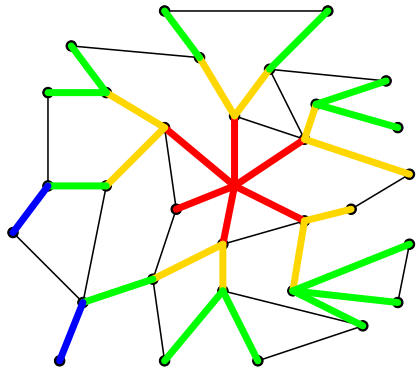


Abbildung 5: Die Idee zum Algorithmus

überlegen, in welcher Reihenfolge die Knoten, deren Nachbarn neu aufgesucht werden, betrachtet werden sollen.

Für beide Probleme gibt es eine elegante Lösung. Bereits besuchte Knoten werden notiert, so dass man bei neu hinzukommenden Nachbarn durch einen Vergleich mit der Liste der notierten Knoten erfährt, ob sie bereits besucht wurden. Nur Knoten, die bisher noch nicht notiert wurden, werden neu in die Liste aufgenommen. Wählt man als Reihenfolge für die Knoten, für die jeweils die Nachbarn betrachtet werden sollen, genau die Reihenfolge der Liste, so entsteht tatsächlich das gewünschte Resultat.

Wir schreiben den Startknoten in ein Notizbuch[4] und kringeln ihn dort ein. Welche Knoten sind zum Startknoten benachbart? Diese Knoten werden hinter den Startknoten in das Notizbuch eingetragen und zwar in der Reihenfolge des Auffindens. Damit kennen wir alle Knoten, die nur eine Kante vom Startknoten entfernt sind. Ist das Ziel einer der soeben notierten Knoten, so sind wir fertig.

Ist aber der Zielknoten nicht direkt mit dem Startknoten benachbart, so müssen wir weitersuchen. Wir markieren den ersten nicht umkringelten Knoten aus der Liste im Notizbuch mit einem Kringel und betrachten dessen Nachbarknoten. Alle die Nachbarknoten, die noch nicht im Notizbuch stehen, werden wiederum in der Reihenfolge des Auffindens eingetragen.

Den vorigen Schritt wiederholen wir mit dem jeweils ersten noch nicht eingekringelten Knoten solange, bis der Zielknoten eingetragen wurde. Will man die Distanzen vom Startknoten zu allen anderen Knoten erfahren, läuft der Algorithmus so lange, bis alle Knoten des Graphen in der Liste stehen oder bis alle Knoten der Liste eingekringelt wurden (Die zweite Abbruchbedingung braucht man, falls der Graph nicht zusammenhängend ist).

Diese Liste im Notizbuch arbeitet nach dem Prinzip „First in first out (FIFO)“. Die passende Datenstruktur ist eine sogenannte „Queue“ oder Warteschlange (Abb. 6).

[4] Die Idee mit dem Notizbuch stammt von Volker Heun: Grundlegende Algorithmen, Vieweg 2000

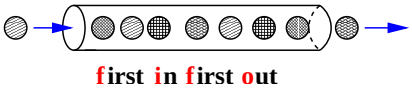


Abbildung 6: Eine Queue oder Warteschlange

Die Vermeidung von Kreisen und die Reihenfolge für die Bearbeitung der Knoten haben wir nun präzise festgelegt. Es fehlt allerdings noch eine Angabe, wie man sich die konstruierten Wege merkt. Eine Möglichkeit ist, sich für jeden Knoten den Vorgängerknoten zu notieren. D. h. für jeden Knoten, der neu in das Notizbuch eingetragen wird, schreibt man sich als Vorgänger den zu dem Zeitpunkt letzten eingekringelten Knoten auf, z. B. mit Hilfe einer Variablen $vor(X) = Y$. Man kann sich auch Kanten merken, z. B. in der Form (X,Y) für die Kante zwischen X und Y.[5] In beiden Fällen kann der gesuchte Weg aus diesen Angaben rekonstruiert werden.

Der beschriebene Algorithmus heißt „Breitensuche“ oder „Breadth-First-Search (BFS)“ und gehört zu den grundlegenden Graphenalgorithmen. Die Breitensuche kann z. B. auch testen, ob ein Graph zusammenhängend ist. In diesem Fall bricht der Algorithmus ab, ohne alle Knoten in das Notizbuch geschrieben zu haben.

Die nachfolgende Formulierung der Breitensuche verwendet der Übersichtlichkeit halber die Sprechweise „eine Kante markieren“. Dahinter steht die Vorstellung, dass man mit einem Stift Kanten anmalt, wie in der Abbildung 5. Gemeint ist aber die Speicherung dieser Kante bzw. von Vorgängerknoten. Da es noch allgemer zu formulieren ist und nur unwesentlich mehr Zeit kostet, wird die Breitensuche zumeist über alle Knoten einer Zusammenhangskomponente durchgeführt und nicht beim Erreichen des Zielknotens abgebrochen.

Ein Beispiel für eine Formulierung der Breitensuche:

Gegeben: ein Graph mit Knotennamen, ein Startknoten S
Gesucht: kürzeste Wege (bzgl. Kantenzahl) vom Startknoten zu allen anderen Knoten.

1. Wähle einen Startknoten.
2. Markiere alle Kanten, die von diesem Knoten ausgehen.
3. Schreibe alle Nachbarknoten des Startknotens in eine Liste.
4. Streiche den ersten Knoten aus der Liste.
5. Markiere alle von diesem Knoten ausgehenden Kanten, sofern sie noch nicht markiert sind, und nicht im Endknoten auf bereits markierte Kanten treffen.
6. Schreibe die Knoten, die am Ende dieser neu markierten Kanten sind, hinten in die Liste.
7. Wiederhole 4., 5. und 6. so lange, bis die Liste leer ist.

[5] In einigen Graphen haben die Kanten Namen, z.,B. Straßennamen im Stadtplan oder auch abstrakte Benennungen, etwa Kleinbuchstaben.

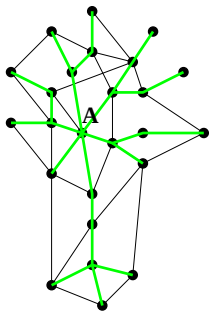


Abbildung 7: Die Breitensuche erzeugt einen aufspannenden Baum

Es gibt viele unterschiedliche Möglichkeiten, die Breitensuche zu formulieren. Diese Möglichkeiten können sich im Detaillierungsgrad stark unterscheiden. Es kommt nur darauf an, dass das Resultat das erwünschte ist.

3.2 Aufspannende Bäume

Schaut man sich das „Produkt“ der Breitensuche an, z. B. in der Abb. 7, so fallen zwei Dinge auf: Der entstandene Teilgraph ist kreisfrei und zusammenhängend. Er ist also ein BAUM. Da dieser Baum alle Knoten des ursprünglichen Graphen erreicht, ist es ein AUFSPANNENDER BAUM.

Definitionen:

- Ein BAUM ist ein kreisfreier zusammenhängender Graph.
- Ein AUFSPANNENDER BAUM ist ein kreisfreier zusammenhängender Teilgraph, der alle Knoten erreicht.

3.3 Korrektheitsbeweis

Zu beweisen ist nun noch, dass die Breitensuche tatsächlich immer kürzeste Wege bezüglich der Kantenzahl konstruiert. Augenscheinlich ist es so, aber wir benötigen die Garantie, dass es nicht doch irgendeinen „verrückten“ Graphen geben kann, in dem man mit der Breitensuche einen kürzesten Weg nicht finden kann. Der durch die Breitensuche erzeugte aufspannende Baum wird im folgenden „BFS-Baum“ genannt.

Beweis durch Widerspruch.

Annahme: Es existieren Knoten, für die man außerhalb des BFS-Baumes kürzere Wege im Graphen finden kann. Nennen wir die Distanzen im BFS-Baum „Distanz“ und die Distanzen außerhalb des BFS-Baumes „Weglänge“.
Sei der Knoten V der „kleinste Verbrecher“ unter ihnen, d.h. der Knoten mit der kleinsten Distanz, für den man einen kürzeren Weg W außerhalb des BFS-Baumes gefunden hat.
Also Weglänge(V) $<$ Distanz(V)

Aus der Annahme folgt:
(1) der Knoten, der V im Weg W vorausgeht, nennen wir ihn U, hat gleiche Weglänge wie Distanz, da V als kleinster Verbrecher gewählt wurde.
Distanz(U) = Weglänge(U).

(2) Weglänge(V) = Weglänge(U)+1, da V nach U im Weg kommt.

(3) Im BFS-Baum muss Distanz(V) > Distanz (U) sein, da ja sonst der Weg über U zu V nicht kürzer sein könnte als der Weg im BFS-Baum. Also war U vor V in der Queue (im Notizbuch).

Da U vor V in der Queue war (siehe 3)) und V und U im Graphen benachbart sind, folgt, dass V spätestens von U aus besucht wurde, eventuell aber auch schon in der gleichen „BFS-Schicht“ wie U war.
(4) Also gilt: Distanz(V) $\leq$ Distanz(U)+1.

(5) Aus (1) und (2) folgt: Distanz(U)+1 = Weglänge(U)+1 = Weglänge(V).

(6) Und es gilt auch noch: Weglänge(V) < Distanz(V) (siehe Annahme).

Die drei Aussagen (4), (5) und (6) zusammen ergeben:
Distanz(V) $\leq$ Distanz(U)+1 = Weglänge(U)+1 = Weglänge(V) < Distanz(V)
Das kann man verkürzt ausdrücken als: Distanz(V) < Distanz(V).
Dies ist ein Widerspruch, also muss die Annahme falsch sein!

qed

3.4 Nichtbaumkanten

Da die Korrektheit des Algorithmus nun bewiesen ist, können wir eine Eigenschaft der Nichtbaumkanten zeigen. Nichtbaumkanten sind alle Kanten des Graphen, die nicht zum BFS-Baum gehören.

Bei der Durchführung der Breitensuche kann man den Knoten einen Wert zuordnen, der die Distanz zum Startknoten angibt (Abb. 8)[6]. Diese Distanzen teilen den Graphen in „BFS-Schichten“ ein. Knoten mit der gleichen Distanz gehören zur selben Schicht.

Was für eine Regelmäßigkeit fällt auf, wenn Sie die Nichtbaumkanten in Abb. 8 betrachten? Die Nichtbaumkanten verbinden stets Knoten derselben oder benachbarter Schichten. Dass das nicht anders sein kann, wird schnell deutlich, wenn man eine Kante einzeichnet, die Knoten von nicht benachbarten Schichten verbindet (Abb. 9). Dadurch entsteht eine Abkürzung eines Weges aus dem BFS-Baum. Da der BFS-Baum bewiesenermaßen bereits kürzeste Wege enthält, kann es solche Abkürzungen nicht geben.

[6] Übungsaufgabe: Bauen Sie in die Breitensuche eine Anweisung ein, die diese Distanzen berechnet und speichert.

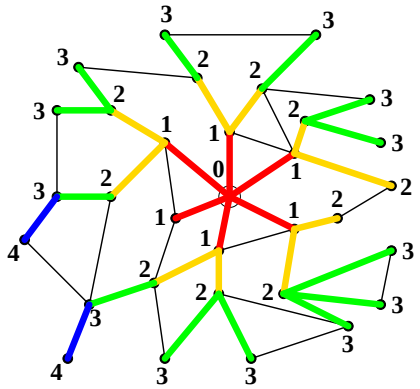


Abbildung 8: Die Schichten der Breitesuche

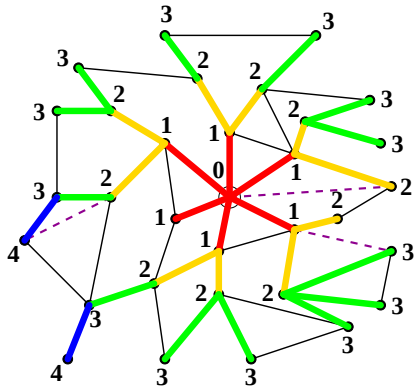


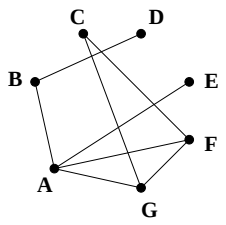
Abbildung 9: Solche Nichtbaumkanten kann es nicht geben.

4 Exkurs: Graphenisomorphie

Computer können kürzeste Wege bestimmen, nachdem man ihnen einen Algorithmus einprogrammiert hat. Wie aber wird der Graph, auf dem der Computer arbeiten soll, computertauglich dargestellt? Es gibt dafür mehrere bewährte Datenstrukturen, eine ist die Adjazenzmatrix (Abb. 10). Die Einträge bedeuten dabei Existenz oder Nichtexistenz einer Kante zwischen zwei Knoten.

Versucht man eine gegebene Matrix in eine Zeichnung umzusetzen, so stellt man fest, dass die Lage der Knoten, die Form und Länge der Kanten nicht durch die Matrix festgelegt sind (Abb. 11).

Durch diese Überlegungen wird deutlich, dass ein Graph zunächst einmal nichts weiter darstellt als Nachbarschaften von Knoten. Mit welcher Bedeutung die Knoten belegt werden und was die (Äquivalenz-) Relation „ist benachbart mit“ darstellen soll, kommt auf den jeweiligen Kontext an.



	A	B	C	D	E	F	G
A	**0**	**1**	**0**	**0**	**1**	**1**	**1**
B	**1**	**0**	**0**	**1**	**0**	**0**	**0**
C	**0**	**0**	**0**	**0**	**0**	**1**	**1**
D	**0**	**1**	**0**	**0**	**0**	**0**	**0**
E	**1**	**0**	**0**	**0**	**0**	**0**	**0**
F	**1**	**0**	**1**	**0**	**0**	**0**	**1**
G	**1**	**0**	**1**	**0**	**0**	**1**	**0**

Abbildung 10: Ein Graph und seine Adjazenzmatrix

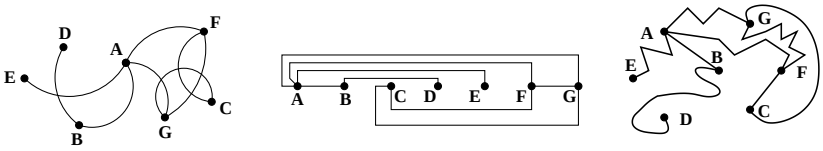


Abbildung 11: Verschiedene Darstellungen desselben Graphen

Die verschiedenen Darstellungen desselben Graphen führen uns zu einem weiteren graphentheoretischen Begriff.

Definition: Zwei Graphen heißen ISOMORPH, wenn man die Knoten beider Graphen so benennen kann, dass die Adjazenzmatrizen beider Graphen gleich sind.

5 Der Algorithmus von Dijkstra

5.1 Gewichtete Graphen

Bislang haben wir uns Gedanken gemacht, wie man Wege findet, die eine minimale Anzahl von Kanten benötigen. Diese Vorgabe reichte für das Auffinden von guten U-Bahnverbindungen aus. Für die Planung von Bahn- oder Autofahrten interessieren allerdings andere Größen. Die Reiseauskunft der Deutschen Bahn AG gibt, falls man nichts weiter angibt, die Verbindung aus, die am wenigsten Zeit benötigt. Routenplaner für Autofahrten können die schnellste (Abb. 12), kürzeste oder billigste Fahrtsrecke ermitteln.

Die Graphen, die bisher betrachtet wurden, beinhalten allerdings keinerlei Informationen über Kantenlängen oder -kosten . Die gezeichnete Länge einer Kante sagt nichts darüber aus (vgl. Abschnitt 4). Daher muss man solche Zusatzinformationen als KANTENGEWICHTE angeben (Abb. 13).

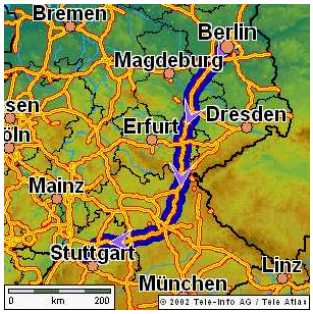


Abbildung 12: Routenplaner im Internet

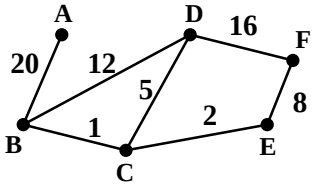


Abbildung 13: Ein gewichteter Graph

Die Adjazenzmatrix dieses Graphen sieht so aus:

	A	B	C	D	E	F
A	-	20	-	-	-	-
B	20	-	1	12	-	-
C	-	1	-	5	2	-
D	-	12	5	-	-	16
E	-	-	2	-	-	8
F	-	-	-	16	8	-

Dabei muss man sich im Klaren sein, dass diese Matrix keine eindeutige Information enthält. Die Einträge könnten auch die Anzahl (ungewichteter) Kanten zwischen zwei Knoten darstellen. Man kann auch vereinbaren, dass der Eintrag 0 bedeutet, dass dort keine Kante ist. Allerdings verliert man dann die Möglichkeit, Kanten mit dem Gewicht 0 zu belegen, was gelegentlich sinnvoll sein kann.

5.2 Der Algorithmus von Dijkstra

Die Breitensuche konstruiert bezüglich der Summe der Kantengewichte keine kürzesten Wege. Dennoch kann die Grundidee der Breitensuche auch hier genutzt werden. Der Beginn des Algorithmus bleibt fast gleich: Vom Startknoten aus werden alle Nachbarn betrachtet, jetzt allerdings mit ihren Kantengewichten (als Distanzen zum Startknoten).

In der nächsten Iteration des Algorithmus geht man nicht vom zuerst aufgefundenen Knoten weiter, sondern von dem mit der kleinsten Distanz. Diese Distanz wird festge-

schrieben (permanent). Alle noch nicht permanent markierten Nachbarn dieses Knotens werden nun betrachtet. Sie bekommen als Distanz jeweils die Summe der Distanz des Vorgängerknotens und des Gewichts der verbindenen Kante. An dieser Stelle kann es vorkommen, dass nun ein Knoten zwei Distanzen hat. Es wird die kleinere ausgewählt und die andere (mit zugehöriger Kante) aus dem Speicher gelöscht. Sehr wichtig ist dabei eine saubere Speicherung der jeweiligen verwendeten Kanten. Mit Stift und Papier können die entsprechenden Kanten einfach eingefärbt und ggf. durchgestrichen werden.

Dieser Mehrfachschritt wird nun sooft wiederholt, bis alle Knoten permanente Distanzen erhalten haben. Abb. 14 zeigt den Ablauf des Algorithmus. Es ist zu beachten, dass der Algorithmus nur dann sinnvoll arbeitet, wenn alle Kantengewichte nicht-negativ sind. Gäbe es negative Kantengewichte, so könnte die Distanz eines bereits berechnete Weges durch die Addition eines solgen negativen Gewichts plötzlich kürzer werden. Somit könnten sich nachträglich kürzere Um-Wege zu den Knoten mit bereits permanent bestimmten Distanzen ergeben.

Zu Beginn des Algorithmus wird allen Knoten die Distanz „unendlich“ zugeordnet. Dadurch muss im Algorithmus keine Fallunterscheidung zwischen erstmalig besuchten Knoten und bereits betrachteten Knoten gemacht werden. In jedem Fall wird eine neue Distanz berechnet und mit der bereits vorhandenen verglichen. Beim ersten Besuch eines Knotens verschwindet so die Distanz „unendlich“ und wird durch die eben berechnete ersetzt. Ist ein Knoten vom Startknoten aus gar nicht erreichbar, weil der Graph nicht zusammenhängend ist, so bleibt die Distanz „unendlich“ stehen, was auch anschaulich sinnvoll ist.

Ein Beispiel für eine Formulierung des Algorithmus von Dijkstra:

Gegeben: ein gewichteter Graph mit nicht-negativen Gewichten, ein Startknoten
Gesucht: Kürzeste Wege vom Startknoten zu allen anderen Knoten
(„Distanz“ bedeutet hier stets Weglänge zum Startknoten.)

1. Zu Beginn bekommen alle Knoten die Distanz unendlich.
2. Wähle einen Startknoten. Der Startknoten bekommt die permanente Distanz 0. Er ist nun der aktive Knoten.
3. Berechne die Distanzen (temporäre Distanzen) aller noch nicht mit permanenten Distanzen versehenen Nachbarknoten des aktiven Knotens:
Kantengewicht der verbindenden Kante + Distanz des aktiven Knotens.
4. „Update“: Ist diese neu berechnete Distanz für einen Knoten kleiner als die bereits vorhandene, so wird die vorherige Distanz gelöscht, die kleinere Distanz notiert, und der aktive Knoten wird als Vorgänger dieses Knotens gespeichert. Ein vorher gespeicherter Vorgänger dieses Knotens wird gelöscht (Mit dieser Vorgängerliste lässt sich am Ende der Weg rekonstruieren).
Ist die neu berechnete Distanz größer als die bereits vorhandene, ändert sich nichts an Distanz und Vorgänger.
5. Wähle einen Knoten mit minimaler temporärer Distanz, diese Distanz wird nun unveränderlich (permanente Distanz). Dieser Knoten wird nun zum aktiven Knoten.
6. Wiederhole 3.-6. so lange, bis alle Knoten permanente Distanzmarken haben.

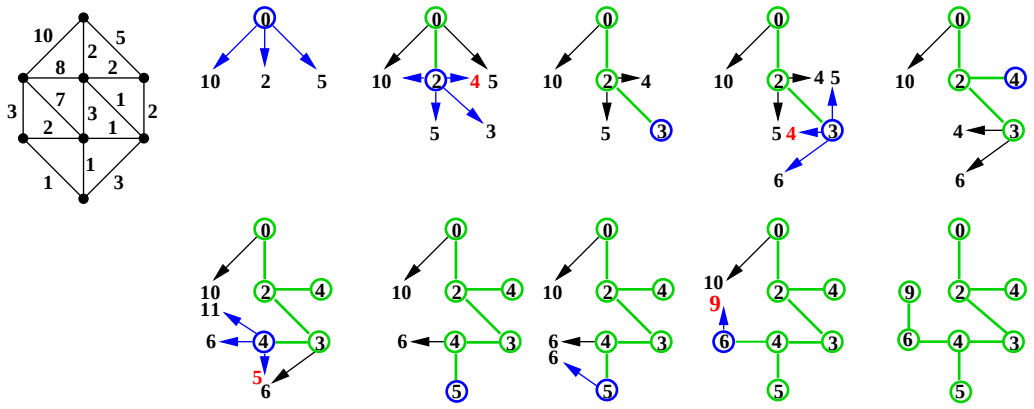


Abbildung 14: Der Algorithmus von Dijkstra. Blau ist der jeweilige aktive Knoten, grün sind die permanent markierten Knoten und Kanten, rot sind Distanzen, die kürzer als bereits vorhandene sind. Der „Update“-Schritt und die Wahl des neuen aktiven Knoten sind jeweils in einem Bild dargestellt.

Statt einer Queue wird hier eine „Priority-Queue“ verwendet, d. h. eine Datenstruktur, die stets das kleinste Element herausgibt. Man kann sich diese Datenstruktur als Warteschlange an der Supermarktkasse vorstellen, bei der immer die Person, die die kleinste Summe zu bezahlen hat, nach vorne darf. Das funktioniert natürlich nur bei endlich vielen Wartenden. Die Sortierung innerhalb dieser Priority-Queue erfolgt üblicherweise mittels des Verfahrens „Heapsort“.

5.3 Korrektheitsbeweis

Es bleibt nun noch zu beweisen, dass der Algorithmus von Dijkstra die kürzesten Wege von einem Startknoten A zu allen anderen korrekt berechnet. Der Korrektheitsbeweis per vollständiger Induktion ist einer der handlicheren Beweise und ohne einen großen zusätzlichen Begriffsapparat verständlich.

Wir bezeichnen die Menge der permanent markierten Knoten mit P. In Schritt 5 des Algorithmus kommt bei jeder Iteration ein Knoten zu dieser Menge hinzu. Die vollständige Induktion wird über die Kardinalität der Menge P geführt. Mit $dist(X)$ bezeichnen wir die vom Algorithmus berechnete Länge eines kürzesten Weges zum Startknoten.

Induktionsannahme: Befindet sich ein Knoten X in P, so ist $dist(X)$ korrekt berechnet.

Induktionsanfang: Zu Beginn des Algorithmus ist nur der Startknoten A in P. $dist(A) = 0$. Da wir nur nicht-negative Kantengewichte zulassen, ist dies die Länge eines kürzesten Weges von A nach A.

Induktionsschritt: Wir betrachten nun die Menge P zu irgendeinem Zeitpunkt im Ablauf des Algorithmus. Sei Z der zuletzt in die Menge P aufgenommene Knoten. Wir nehmen nun an, dass $dist(Z)$ nicht korrekt berechnet worden ist, und führen diese Annahme zum Widerspruch.

Sei k ein kürzester Weg von A nach Z. Da Z bisher nicht in P war, muss es eine Kante in k geben, die P verlässt. Nennen wir ihre Endknoten U und W. Sei U in P und W nicht in P.

W≠Z. Warum? Nach Induktionsvoraussetzung wurde $dist(U)$ korrekt berechnet. Zudem ist der in k enthaltene Weg von A nach W ein kürzester Weg, da man anderenfalls den Weg k, der ja ein kürzester Weg ist, noch verkürzen könnte. Daraus folgt, dass auch $dist(W)$ korrekt berechnet wurde. Nach Annahme ist $dist(Z)$ nicht korrekt berechnet worden, also muss W≠Z sein.

Sei $d(Z)$ die Länge eines kürzesten Weges von A nach Z. Nach Annahme ist $dist(Z)$ nicht korrekt berechnet worden, also muss $d(Z) < dist(Z)$. sein. Außerdem gilt $d(W) \leq d(Z)$, da alle Kanten nicht-negatives Gewicht haben. Somit gilt: $dist(W) = d(W) \leq d(Z) < dist(Z)$. Dann aber hätte der Algorithmus in dem betrachteten Moment nicht Z, sondern W in die Menge P der permanent markierten Knoten aufgenommen. Damit haben wir einen Widerspruch zur Annahme gefunden und die Korrektheit des Algorithmus bewiesen.

6 Unterricht

6.1 Einstieg

Eine Unterrichtsreihe über kürzeste Wege beginnt am besten mit dem Blick auf das lokale Bus- oder U-Bahnnetz (falls vorhanden) oder auch auf den Stadtplan. Die Suche nach einem kürzesten Weg von zu Hause zur Schule ist eine attraktive Aufgabe, da die Lösung im Alltag tatsächlich eine Rolle spielen kann. Durch die unterschiedlichen Wohnorte ergibt sich eine Individualisierung der Aufgabe. Nebenbei beschäftigen die Lernenden sich mit ihrem Wohnort, dessen Infrastruktur und den geographischen Gegebenheiten.

Ausgehend von diesem konkreten Beispiel können die Lernenden weitgehend selbständig zur Verallgemeinerung des Problems kommen und Ideen für kürzeste-Wege-Algorithmen entwickeln. Von zentraler Bedeutung ist dabei, dass die mathematische Theorie anhand der aus dem Anwendungsbeispiel erwachsenden Fragen entwickelt wird.

Der Übergang von den ersten spontanen Ideen für das Auffinden kürzester Wege zum allgemeingültigen Algorithmus wird von den Lernenden vermutlich nicht allein vollzogen werden können. Hier können Hilfestellungen von Seiten der Lehrkraft erforderlich werden. Anregungen dazu finden Sie im Abschnitt 6.6.

6.2 Weiterer Unterrichtsverlauf

Eine Unterrichtseinheit, die die im ersten Teil dieses Kapitels vorgestellten Inhalte behandelt, sollte ca. 12 Stunden umfassen. Hat man weniger Zeit zur Verfügung, so können die Überlegungen zur Graphenisomorphie wegfallen. Bleibt man bei der Ausformulierung der Algorithmen auf der Ebene des „Anmalens von Kanten“, so entfallen die Betrachtungen zu Datenstrukturen und zu speichernden Daten. Insbesondere in jüngeren Jahrgangsstufen empfiehlt sich das. In der Oberstufe ist die Behandlung dieser Aspekte eine sehr reizvolle Möglichkeit die enge Verknüpfung zwischen (theoretischer) Informatik und Mathematik deutlich zu machen.

Ob die Korrektheitsbeweise im Unterricht behandelt werden, bleibt Ermessenssache. Erfahrungsgemäß fällt selbst das Nachvollziehen solcher Beweise auch guten Schülerinnen und Schülern schwer. Wichtig ist aber, dass die Lernenden sich bewusst werden, dass die Intuition allein nicht ausreicht, sondern die Korrektheit eines Algorithmus zu beweisen ist.

Für den gesamten Verlauf der Unterrichtseinheit empfiehlt es sich, stets darauf zu achten, welche Fragestellungen bei den Lernenden von selbst auftauchen. Diese Fragen können den inhaltlichen Ablauf des Unterrichts mit lenken. Das ist deshalb gut möglich, weil der Stoff nicht induktiv aufgebaut ist, sondern eher sternförmig (Abb. 15).

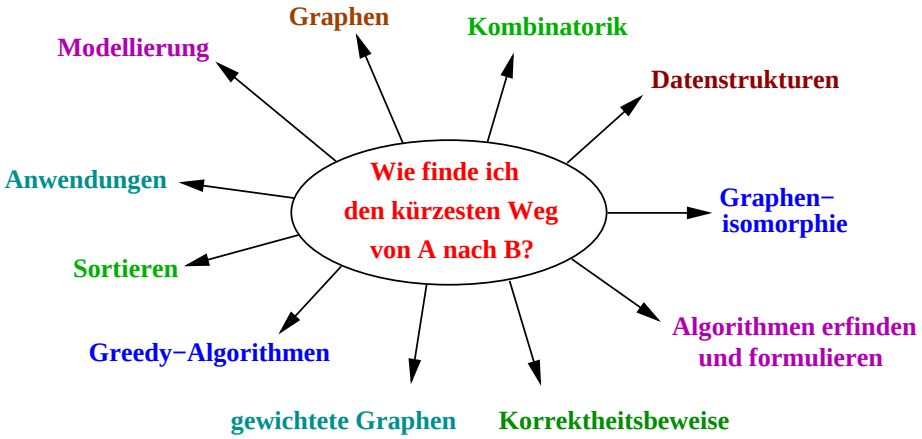


Abbildung 15: Die Stoffabfolge kann nach den Bedürfnissen der Lerngruppe gewählt werden

Wichtig ist, dass die Aufgabenstellungen stets so weit tragen, dass die Schüler sich über einen längeren Zeitraum damit befassen können. Hier einige Beispiele für Aufgabenstellungen, die den Lern- und Erforschungsprozess anregen und lenken können:

- Finde einen kürzesten Weg in einem Nahverkehrsnetz. Entwickle eine Strategie, die auch in anderen Netzen kürzeste Wege findet. Entwirf eigene U-Bahnnetze und probiere auch daran deine Strategie, kürzeste Wege zu finden, aus. Probiere aus, ob die Strategie auch in ganz verrückten Beispielen richtig arbeitet.
- Stelle dir vor, du willst einem kleinen Roboter beibringen, für dich herauszufinden, wie man am schnellsten von zu Hause zur Schule kommt. Er kann immer nur eine Anweisung nach der anderen befolgen. Schreibe ihm eine Liste von einfachen Anweisungen auf. Kann es mit deinen Anweisungen passieren, dass der Roboter sich irrt?

6.3 Das „Graphenlabor“

Man kann kürzeste-Wege-Algorithmen im Unterricht behandeln, ohne je von Graphen zu sprechen. Andererseits biete die Behandlung dieser Thematik eine gute Möglichkeit,

eine neue mathematische Objektklasse einzuführen. Dass mit dem Begriff „Graph“ in der Schule häufig Funktionsgraphen gemeint sind, sollte kein Hindernis sein. Aus dem Kontext heraus sieht man stets, welche Art von Graph jeweils gemeint ist. Die Arbeit mit Graphen eröffnet eine einzigartige „Spielwiese“ zum Experimentieren. Man kann es sich als ein „Graphenlabor“ vorstellen, in dem wie in einer Alchemistenküche die verschiedensten Graphen kreiert werden.

Möchte man also etwas tiefer in die Graphentheorie eindringen, so gibt es dafür mannigfache Möglichkeiten. So taucht bei der Arbeit mit Graphen immer wieder die Frage nach guten Beispielgraphen auf. Diese Frage lässt sich gut für den Unterricht nutzbar machen. Sogenannte „worst case“-Beispiele zu konstruieren hilft oft, den gerade betrachteten Sachverhalt besser zu verstehen und ist nicht immer so einfach, wie man denkt! Oder man begibt sich auf die Suche nach generellen Eigenschaften von Graphen.

Besonders schön ist es aber, sich Graphen mit bestimmten Eigenschaften zu bauen und sich somit ganz in das Labor der Graphenkonstrukteure zu begeben. Eine sehr fruchtbare Experimentieraufgabe im Zusammenhang mit kürzesten Wegen ist die Konstruktion von U-Bahnnetzen (Graphen) mit ganz bestimmten Eigenschaften. Wie sehen Graphen aus, die folgende Anforderungen erfüllen? Kann man allgemeingültige Aussagen dazu machen?

- Jede Station ist von jeder anderen aus direkt erreichbar (vollständige Graphen)
- Die Wege zwischen je zwei Stationen sind höchstens zwei Kanten lang. (z. B. Sterngraphen, aber auch viele andere)
- Die Wege zwischen je zwei Stationen sind höchstens n Kanten lang. (Graphen mit Durchmesser n).
- Die Länge eines kürzesten Weges zwischen je zwei Stationen ist direkt ablesbar (Bäume)

Die Lernenden sollten auch selbst solche Aufgaben erfinden. Auf diese Weise stoßen sie auch auf generelle Grapheneigenschaften und haben die Chance sich tiefgehend und nachhaltig mit Graphen vertraut zu machen. Man kann mehrere Schulstunden mit solcher selbstbestimmter graphentheoretischer Forschung verbringen, wenn der Zeitplan es zulässt. Mit zu dieser Arbeit im Graphenlabor gehört auch der Versuch, die gewonnenen Resultate zu begründen und angemessen schriftlich zu fixieren, etwa in einem Lerntagebuch.

6.4 Graphenisomorphie

Das Phänomen der Graphenisomorphie ist ein für den Unterricht sehr lohnendes Thema. Es werden neben dem Modellieren und Mathematisieren weitere Dimensionen des Denkens angesprochen wie z. B. das räumliche Vorstellungsvermögen und das Kategorisieren. Eine spielerische Gestaltung des Unterrichts durch kreatives Ausprobieren und das Finden von eigenen Graphen mit interessanten verschiedenen Darstellungen spricht insbesondere auch jüngere Altersstufen an. Der Übergang zur Mathematisierung geschieht ganz natürlich, wenn man sich auf die Suche nach Kriterien zum Überprüfen der Isomorphie begibt.

Welche Eigenschaften müssen überprüft werden, um herauszufinden, ob zwei Graphen isomorph sind?

- gleiche Anzahl Knoten
- gleiche Anzahl Kanten
- gleiche Knotengrade (Definition siehe Anhang)
- Nachbarschaft von Knoten entsprechender Grade (gleiche Gradfolgen)
- Existenz von Mehrfachkanten zwischen entsprechenden Knoten (Abb. 16)

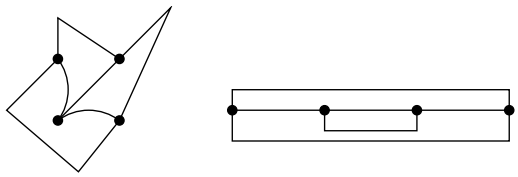


Abbildung 16: gleiche Anzahl Knoten, Kanten, gleiche Gradfolge, aber dennoch verschiedene Graphen

Durch die Beschäftigung mit isomorphen Graphen bekommen die Schülerinnen und Schüler ein Gefühl für das „Wesen“ von Graphen. Graphen sind keine starren geometrischen Gebilde, sondern eine Repräsentationsform für Konfigurationen von Beziehungen im weitesten Sinne. Das Objekt Graph kann beliebig verformt werden, ohne dass sich der Aussagegehalt ändert. Diese Tatsache kann besonders gut durch ein Knopf-Faden-Modell (siehe Abb. 17) veranschaulicht werden, das man auf den Folienprojektor legen kann, in die Hosentasche stecken oder an den Kleiderhaken hängen oder was auch immer. Die Nachbarschaftsbeziehungen der Knöpfe (repräsentiert durch die Fadenverbindungen), also die in dem Graphen beinhalteten Informationen, ändern sich dadurch nicht.

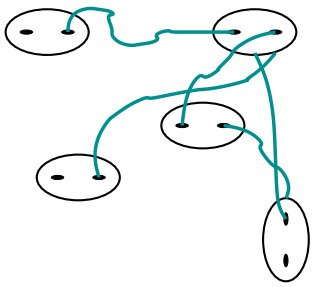


Abbildung 17: Der „Knopfgraph“: ein Knopf-Faden-Modell

Arbeitet man mit Fadenmodellen von Graphen, so gibt es einen sehr einfachen Algorithmus zum Auffinden kürzester Wege. Man knüpft den gegebenen Graphen so zusammen, dass die Fadenlänge zwischen je zwei Knoten (die man in diesem Fall am besten

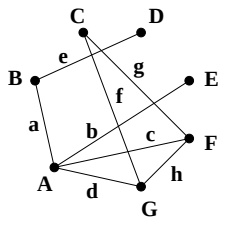
tatsächlich als Knoten realisiert) proportional zum jeweiligen Kantengewicht ist. Dann nimmt man den Anfangsknoten mit den Fingern der einen Hand und den Endknoten mit der anderen Hand und zieht das Modell straff. Der straff gespannte Weg zwischen den beiden Knoten ist der kürzeste!

Nachteil dieser Methode ist, dass das Basteln des Fadenmodells sehr zeitaufwändig ist. Hat man aber Zeit, z. B. im Rahmen von Projekttagen, so kann dieses Experiment sich lohnen, weil es deutlich macht, dass die Problematik an sich nicht schwierig ist. Die Schwierigkeit liegt darin, das Vorgehen zu abstrahieren und automatisieren.

6.5 Matrizen

Die Überlegungen zu geeigneten Datenstrukturen für Graphen führen auf Matrizen, die in der Schule üblicherweise im Zusammenhang mit linearen Gleichungssystemen und linearen Abbildungen vorkommen. Hier ist ein Anknüpfungspunkt zur klassischen Schulmathematik gegeben.

Neben der quadratischen symmetrischen Adjazenzmatrix die für jeden Knoten je eine Zeile und eine Spalte besitzt, gibt es die Darstellungsform der Inzidenzmatrix (Abb. 18). Für Graphen, die wenig „dicht“ sind, d. h. die relativ wenige Kanten besitzen, ist die Inzidenzmatrix eine Speicherplatz sparende Darstellung.



	A	B	C	D	E	F	G
a	**1**	**1**	**0**	**0**	**0**	**0**	**0**
b	**1**	**0**	**0**	**0**	**1**	**0**	**0**
c	**1**	**0**	**0**	**0**	**0**	**0**	**1**
d	**1**	**0**	**0**	**0**	**0**	**0**	**1**
e	**0**	**1**	**0**	**1**	**0**	**0**	**0**
f	**0**	**0**	**1**	**0**	**0**	**0**	**1**
g	**0**	**0**	**1**	**0**	**0**	**1**	**0**
h	**0**	**0**	**0**	**0**	**0**	**1**	**1**

Abbildung 18: Inzidenzmatrix

Im Rahmen des Nachdenkens über Matrizen kann mit Aufgaben wie der in Abb 19 abgedruckten[7] die Datenanalyse geübt werden. Dabei entstehen Fragestellungen wie: Welche Rolle spielen Einträge auf der Diagonalen? Wie wirken sich verschiedene symmetrische oder asymmetrische Anordnungen der Einträge aus? Was lässt sich über die Summe der Einträge der Inzidenzmatrix eines ungewichteten Graphen aussagen? Was bedeuten Einträge, die nicht 0 oder 1 sind? Mit Hilfe solcher Aufgabenstellungen kann man den in der Schule sonst eher kalkülorientierten Umgang mit Matrizen bereichern.

[7] Diese Aufgabe basiert auf einer Idee von StDir Arne Madincea (Herder-Oberschule Berlin-Charlottenburg).

$$\begin{matrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{matrix} \qquad \begin{matrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{matrix} \qquad \begin{matrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$

$$\begin{matrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{matrix} \qquad \begin{matrix} 0 & 0 & 0 & 3 & 0 \\ 0 & 2 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 3 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{matrix} \qquad \begin{matrix} 0 & 1 & 1 & 0 & 0 \\ -1 & 0 & -1 & 0 & 1 \\ -1 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & -1 \\ 0 & -1 & 1 & 1 & 0 \end{matrix}$$

$$\begin{matrix} 0 & 2 & 1 & 0 & 1 \\ 2 & 0 & 1{,}3 & 4 & 2 \\ 1 & 1{,}3 & 2{,}5 & 0 & 34 \\ 0 & 4 & 0 & 0 & 6{,}7 \\ 1 & 2 & 34 & 6{,}7 & 0 \end{matrix} \qquad \begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix} \qquad \begin{matrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{matrix}$$

Abbildung 19: Aufgabe: Interpretieren Sie diese Matrizen! Wie könnten die Graphen aussehen?

6.6 Algorithmen erfinden

Im Mathematikunterricht kann und soll es nicht darum gehen, Programmiersprachen zu erlernen. Im Rahmen des Unterrichts über kombinatorische Optimierung werden Algorithmen entworfen und ausformuliert. Es ist vollkommen ausreichend und bereits anspruchsvoll genug, Umgangssprache zu schreiben. Zur Erleichterung der einfachen Lesbarkeit der umgangssprachlichen Texte kann die Verwendung von Schlüsselwörtern wie „solange“, „wenn - dann“ im Sinne eines Pseudocode vereinbart werden, notwendig ist es aber nicht. In den seltenen Ausnahmefällen, in denen die ganze Lerngruppe bereits eine Programmiersprache beherrscht, spricht natürlich nichts dagegen, dieses Können auch im Mathematikunterricht einzusetzen.

Eine Schwierigkeit beim Niederschreiben von Algorithmen liegt darin, dass man sehr viele Dinge intuitiv und durch „Draufgucken“ löst. Jemandem, der ganz ohne Vorkenntnisse mit Hilfe des Algorithmus an die Lösung des Problems geht, dem Computer z. B., muss man aber jeden kleinsten Schritt vorgeben. Ein wichtiger Bestandteil des Unterrichts ist daher die Ausarbeitung der Grundidee des Algorithmus zu einer präzisen Formulierung. Dafür ist eine genaue Analyse dessen, was man tun will, notwendig.

Um den Blick erst einmal grundsätzlich zu schärfen, kann man mit der Klasse Algorithmen für alltägliche Tätigkeiten wie Spaghettikochen oder den Weg vom Klassenzimmer zum Ausgang der Schule aufschreiben. Dabei stellt sich genauso wie bei Graphenalgorithmen die Frage: Wie genau müssen die Anweisungen für einen Algorithmus werden? Muss ich dazusagen, dass eine Türe vor dem Durchqueren geöffnet werden muss oder setze ich das als selbstverständlich voraus? Ist es wirklich notwendig, dass in den Anleitungen zum Aufwärmen von Tiefkühlpizza (die auch Algorithmen sind) als Arbeitsschritt „Folie

entfernen“ steht? Diese Beispiele zeigen, dass Entscheidungsspielräume bleiben. Wichtig ist nur, dass man sich bewusst ist, was man als selbstverständlich voraussetzt.

6.6.1 Die Froschperspektive und die Lochblende

Ein sehr wirksames Mittel, um den Blick vom Gesamtbild des Graphen zu lösen, ist die Vorstellung der Froschperspektive.

Der Computer weiß, wenn er einen Knoten betrachtet, nur, wie viele und welche Nachbarn dieser Knoten hat, weiter nichts. Diese Informationen stehen in der dem Knoten entsprechenden Zeile oder Spalte der Adjazenzmatrix. In welcher Richtung sich ein etwaiger Zielknoten befindet, kann aus der Matrix nicht herausgelesen werden. Geometrische bzw, geographische Informationen sind in der Struktur „Graph“ nicht enthalten (vgl. Abschnitt 4 über Graphenisomorphie).

Diese Froschperspektive lässt sich im Schulhof wunderbar erleben (Abb. 20). Zeichnen Sie einen großen Graphen mit Kreide auf den Schulhof und bitten Sie eine Person, sich auf den zuvor gewählten Startknoten zu setzen. Eine Gruppe von Mitschülern gibt dieser Person genaue Anweisungen, wie sie sich über den Graphen bewegen soll. Der „Witz“ dabei ist, dass die Anweisungsgeber den Graphen auch nicht überblicken, bzw. am besten gar nicht sehen. Die Person auf dem Graphen braucht farbige Kreiden, um Knoten oder Kanten markieren zu können, und Papier und Bleistift, falls sie sich besuchte Knoten o. ä. merken (also notieren) soll. Alle anderen stehen, falls die räumlichen Gegebenheiten es zulassen, in einem höheren Stockwerk am Fenster und kontrollieren das Resultat.

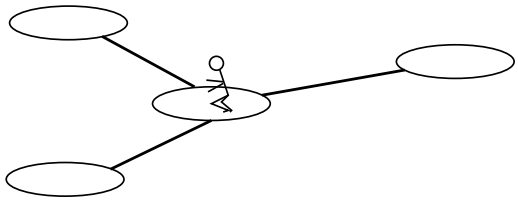


Abbildung 20: Die Froschperspektive des Computers

Nicht immer hat man die Möglichkeit, solche Freiluftexperimente zu machen. Dann hilft die Lochblende (Abb. 21) weiter. Sie wird entweder aus Pappe oder aus Folie gefertigt, wobei Folie praktischer ist, da das Festhalten und Verschieben der Pappe an der Tafel etwas mühsam sein kann. Mit einem Graphen, der Lochblende und einem Stift kann man sich ähnlich wie im Schulhof klarmachen, wie man den Computer trotz seiner Froschperspektive dazu bringt, kürzeste Wege zu finden.

6.6.2 Blättertausch

Eine Schwierigkeit, die insbesondere in der Mittelstufe auftaucht[8] ist ein mangelndes Fehlerbewusstsein bezüglich der selbst erfundenen und formulierten Algorithmen. Viele Lernende sind schon nach dem Durchtesten weniger Beispielgraphen davon überzeugt,

[8] vgl. Andreas Schuster: Habilitationsschrift, Universität Würzburg, 2004

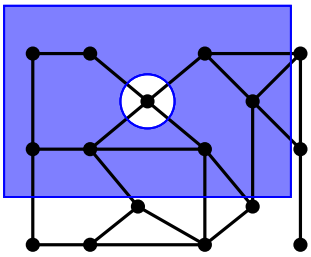


Abbildung 21: Die Lochblende

dass ihre Vorgehensweise und Formulierungen „wasserdicht“ sind. Im Unterricht sollte es daher ein Ziel sein, ein solches Fehlerbewusstsein entstehen zu lassen.

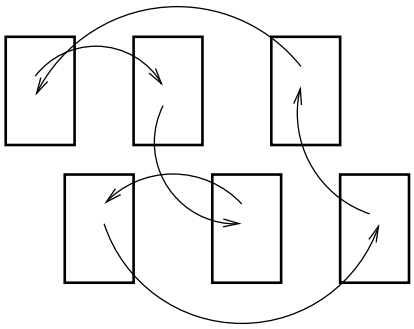


Abbildung 22: Hilfe bei der Fehlersuche: Blättertausch

Ein einfaches, aber auch sehr wirksames Mittel zur Fehlersuche in Algorithmen ist der Blättertausch. Jeder schreibt seinen Algorithmus auf ein loses Blatt Papier. Diese Blätter werden gemischt und neu in der Klasse verteilt. Nun führt jeder den gerade erhaltenen Algorithmus an einem eigenen Beispielgraphen aus. Dabei werden Schwächen in der Formulierung aber auch kozeptionelle Fehler meist problemlos erkannt. Wer Fehler oder Unklarheiten entdeckt, schreibt Kommentare und Vorschläge zur Verbesserung dazu. Die Autor/innen bekommen ihr eigenes Blatt zurück und arbeiten die Verbesserungsvorschläge nach kritischer Prüfung ein.

Führt man den Blättertausch in mehreren Iterationen durch, so erhält man gute Ergebnisse. Dabei werden sämtliche Mitglieder der Lerngruppe gleichermaßen gefordert. Das Eindenken in den Gedankengang von anderen wird geschult und auch die Schwächeren kommen zum Zuge, denn das Ausführen eines Algorithmus gelingt normalerweise immer und Fehler in der Formulierung werden dabei von allen rasch entdeckt.

6.6.3 Rollenspiel

Eine besonders reizvolle Methode, die bei der präzisen Analyse und Formulierung von Algorithmen helfen kann, ist das Rollenspiel. An der Tafel werden die benötigten Da-

tenstrukturen und Listen[9] angezeichnet. Für den Algorithmus von Dijkstra braucht man also einen gewichteten Graphen in Form seiner Adjazenzmatrix, eine Liste der permanent markierten Knoten, eine Liste der bereits besuchten, aber noch nicht permanent markierten Knoten (als Priority Queue, falls es im Unterricht behandelt wurde), eine Liste mit den Distanzen, eine Liste mit den Vorgängerknoten. Eine Tafelseite oder aber auch eine Folie auf dem OH-Projektor sollte zur Verfügung stehen, um die Schritte des Algorithmus zu notieren. Siehe dazu Abb. 23.

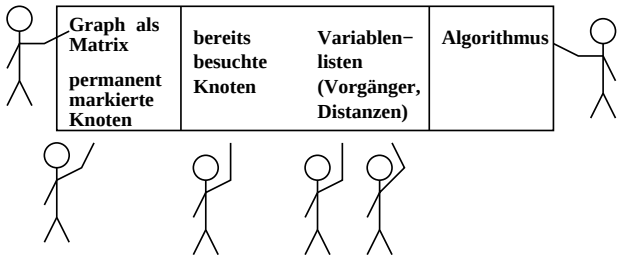


Abbildung 23: Das „Rollenspiel“

Nun wird für jede dieser Listen/Datenstrukturen eine Person gewählt, die dieses Objekt verwaltet. Alle anderen wachen darüber, ob an der Tafel korrekt gearbeitet wird. Dann fängt der Algorithmus an zu laufen. Eine Person sollte stets notieren, was gerade getan wird. Auf diese Weise werden Schleifen im Ablauf leicht erkannt und es entsteht eine erste Fassung einer formalisierten Schritt-für-Schritt-Anweisung.

Durch dieses Rollenspiel klären sich viele Fragen, z.B. über die zeitliche Abfolge von Tätigkeiten oder wann genau welche Information von Nöten ist. Nebenbei hat das Rollenspiel einen hohen Motivationsfaktor. Es bringt Bewegung in das Klassenzimmer, regt zu Diskussionen an und ist hervorragend geeignet, die Mechanik von Algorithmen zu veranschaulichen. Besonders reizvoll ist es, das Rollenspiel nur auf den abstarkten Datenstrukturen ablaufen zu lassen und erst hinterher zur Kontrolle den Graphen zu zeichnen und das berechnete Resultat einzutragen. Dieser Nervenkitzel, ob der Algorithmus tatsächlich das tut, was geplant war, ist selbst für Oberstufenschüler noch spannend.

[9] hier ist mit „Liste“ nicht eine spezielle Datenstruktur gemeint