

## 2

# Günstig verbunden: Minimale aufspannende Bäume

Brigitte Lutz-Westphal

### 1 Leitungsnetze planen, Straßen erneuern und Computer verkabeln

Dieses Kapitel handelt von Algorithmen, die zur Planung von Leitungsnetzen eingesetzt werden können, die aber auch in anderen Anwendungen wie z. B. dem Chipdesign eine Rolle spielen. Die Algorithmen basieren auf einer verblüffend einfachen Idee und sind daher auch schon gut für den Unterricht in der Mittelstufe geeignet. Darüberhinaus birgt diese Art der Netzplanung reizvolle graphentheoretische Aspekte in sich. Es geht hier um spezielle Graphen, so genannte *Bäume*, die viele interessante Eigenschaften haben. Einige dieser Eigenschaften können Sie oder Ihre Schülerinnen und Schülern bei der Arbeit an den Anwendungsproblemen selber entdecken und sich damit ein spannendes Gebiet der Graphentheorie selbst erschließen.

Die vier im Folgenden vorgestellten Probleme zielen auf vergleichbare Lösungen und unterscheiden sich mathematisch gesehen nur wenig voneinander. Die Fragen zu den verschiedenen Problemen zeigen aber unterschiedliche Richtungen der Bearbeitung auf. Das Kapitel basiert auf Problem 1, die Inhalte sind problemlos auf die anderen Anwendungssituationen übertragbar. Nehmen Sie sich Zeit, eigene Lösungen oder Lösungsansätze zu finden! Dann sind Sie bestens auf die Lektüre dieses Kapitels vorbereitet und können im Laufe des Kapitels weiter an Ihren Ideen feilen.

#### *Problem 1 – Leitungen erneuern*

In die vorhandenen Leitungsrohre einer Telefongesellschaft sollen neue Glasfaserkabel für eine besonders schnelle Datenübertragung verlegt werden. Dabei sollen

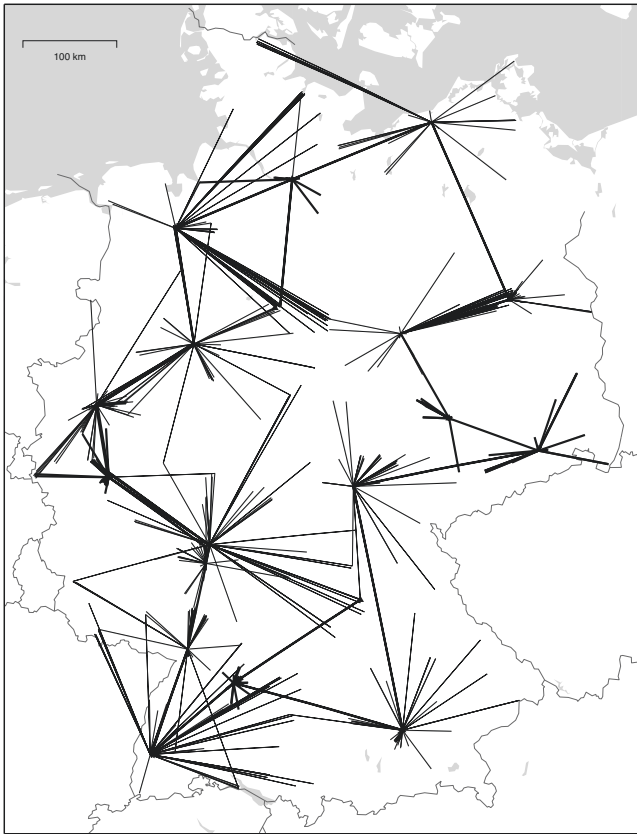


Abbildung 1. »Datenauto-  
bahnen« für die Wissen-  
schaft (reale Daten aus der  
Planung für das Wissen-  
schaftsnetz G-WiN, Quelle:  
ZIB).

natürlich die Kosten so gering wie möglich gehalten werden und dennoch alle Kunden von der besseren Übertragungsqualität profitieren.

- Müssen alle Leitungen erneuert werden oder können einige alte Leitungen be-  
lassen werden?
- Wie wählt man die zu erneuernden Leitungsabschnitte aus?
- Gibt es eine eindeutige Lösung?

Probieren Sie anhand verschiedener Beispiele für Leitungsnetze (Abbildungen 1 und 2) aus, wie Sie an das Problem herangehen würden. Am Besten zeichnen Sie sich mehrere eigene Beispiele und färben mit einem bunten Stift die Leitungen ein, die erneuert werden sollen. Wann haben Sie genügend Leitungen eingefärbt? Was ist der entscheidende Unterschied zwischen den beiden Telefonnetzen in Abbildung 2? Warum fällt es Ihnen im zweiten Telefonnetz leichter, eine Lösung zu finden? Gibt es verschiedene gleich gute Lösungen? Versuchen Sie, Ihre Ideen in Worte zu fassen. Dann haben Sie bereits eine erste Beschreibung dessen, was wir in diesem Kapitel suchen.

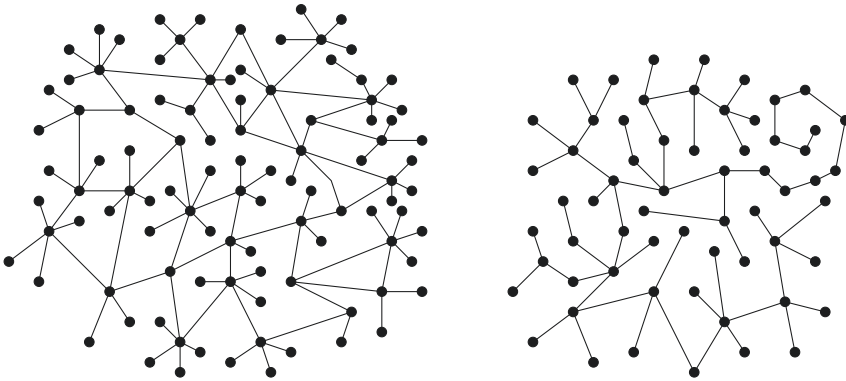


Abbildung 2. Zwei Telefonleitungsnetze

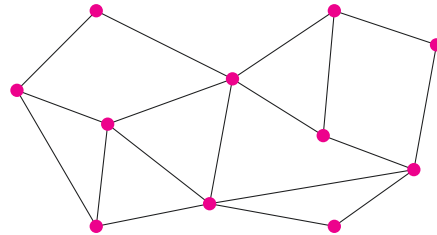


Abbildung 3. Welche Straßen sollen erneuert werden?

### Problem 2 – Straßenbeläge kostengünstig verbessern

In einem Straßennetz zwischen mehreren Dörfern (z. B. Abbildung 3) sollen die Straßenbeläge erneuert werden. Um Geld zu sparen, sollen die Bauarbeiten an möglichst wenigen Straßen durchgeführt werden. Trotzdem muss jeder Ort auf neuen Straßen zu erreichen sein.

- Nach welchen Kriterien wählen Sie die Straßen aus?
- Wieviele Straßen müssen erneuert werden?

### Problem 3 – Telefonleitungen mieten

Eine neue Mobilfunkgesellschaft will Leitungen von der Telekom mieten. Zwischen den Städten nutzt auch der Mobilfunk das Festnetz.

- Welche Leitungen soll die neue Firma mieten?
- Wie sieht es mit der Ausfallsicherheit aus?

In den Abbildungen 1 und 2 sind verschiedene Leitungsnetze zu sehen, die Sie der Bearbeitung des Problems zugrunde legen können.

### Problem 4 – Computernetzwerke verkabeln

In einer Schule wird ein neuer Computerraum eingerichtet. Die Computer sollen ein Netzwerk bilden. Für die Kabel sind schon Kabelkanäle montiert worden. Nun bleibt nur noch zu entscheiden, welche der möglichen Kabelverbindungen genutzt werden, um alle Computer an das Netzwerk anzuschließen. Je weniger Meter Kabel dabei verbraucht werden, umso einfacher werden sowohl die Montagearbeiten als auch die Wartung.

- Suchen Sie Methoden, mit denen das Problem gelöst werden kann, ohne dass Sie wissen, wie der konkrete Raum aussieht.

## 2 Das Problem modellieren

Leitungsnetze wie in den Abbildungen 1, 2 oder 3 können als Graphen interpretiert werden (vgl. Kapitel 1). Daher werden im Folgenden meist die entsprechenden Fachbegriffe verwendet, auch wenn Sie oder Ihre Schülerinnen und Schüler vermutlich zunächst von Knotenpunkten und Leitungen sprechen werden. Für Ihre eigene Erarbeitung und im Unterricht hat es mit der Fachsprache noch Zeit.

Welche Kriterien soll das Netz der neuen Leitungen erfüllen? Sie haben bereits darüber nachgedacht und haben vermutlich ähnliche Ideen gehabt wie die hier ausgeführten.

(1) Alle Kunden sollen von den besseren Kabeln profitieren, also müssen alle Knoten an das neue Netz angeschlossen werden. In Abbildung 4 ist dieses Kriterium durch die rot eingezeichneten Kanten erfüllt. Trotzdem ist dies noch keine gute Lösung. Warum?

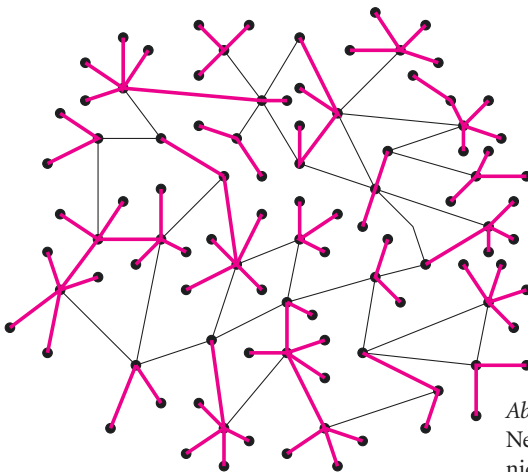


Abbildung 4. Jeder Knoten wurde an das rote Netz angeschlossen, doch reicht das so noch nicht aus.

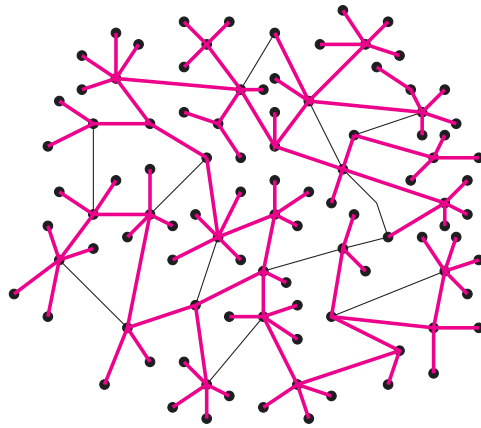


Abbildung 5. Alle Knoten sind miteinander verbunden

(2) Jeder soll mit jedem über das neue Netz kommunizieren können. Also müssen wir noch weitere Leitungen hinzufügen (wie in Abbildung 5), um ein zusammenhängendes rotes Netz zu bekommen.

Der Begriff *zusammenhängend* wird auch in der Graphentheorie verwendet und entspricht unserer intuitiven Vorstellung. Durch die Überlegung, dass jeder Knoten mit jedem anderen (nicht immer direkt, sondern auch über Umwege) verbunden sein soll, bekommen wir unmittelbar eine graphentheoretische Beschreibung von Zusammenhang geliefert.

### ■ Definitionen

Ein Graph heißt *zusammenhängend*, wenn jeder Knoten mit jedem anderen durch einen Kantenzug verbunden ist. Besteht ein Graph aus mehreren untereinander nicht verbundenen Teilen, so nennt man diese Teile *Zusammenhangskomponenten*.

Dabei ist zu beachten, dass ein isolierter Knoten als eigene Zusammenhangskomponente aufgefasst wird.

(3) Das rote Netz bzw. der rote Teilgraph soll zusammenhängend sein und alle Knoten erreichen. Das setzt natürlich voraus, dass der ursprüngliche Graph selbst zusammenhängend ist, wovon wir im Folgenden stets ausgehen werden. Zudem wollen wir nur so wenige rote Kanten wie möglich verwenden. Welches Kriterium gibt an, wann wir genug rote Kanten haben? Aus Sparsamkeitsgründen wollen wir vermeiden, dass das rote Netz doppelte Verbindungen zwischen zwei Knoten enthält. Solche doppelten Verbindungen erzeugen Kreise (Definition vgl. Kapitel 1, S. 17) in dem roten Graphen (siehe Abbildung 6). Wir suchen also einen Teilgraphen, der zusammenhängend ist, alle Knoten erreicht und kreisfrei ist. Genau das ist die Definition eines *aufspannenden Baumes*.

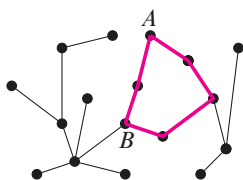


Abbildung 6. A und B sind durch zwei verschiedene Wege miteinander verbunden, es entsteht ein Kreis.

### Definitionen

Ein *aufspannender Baum* eines zusammenhängenden Graphen ist ein kreisfreier und zusammenhängender Teilgraph, der alle Knoten erreicht.

Ein Graph, der selbst sein aufspannender Baum ist, heißt einfach *Baum*.

Anders ausgedrückt: Ein *Baum* ist ein kreisfreier zusammenhängender Graph.

Ist der ursprüngliche Graph nicht zusammenhängend, so kann man darin keinen aufspannenden Baum finden, aber einen aufspannenden *Wald*, der aus mehreren Bäumen besteht

Diese Baumdefinition ist nicht die einzig mögliche. Sie ergibt sich in dieser Form aus den Anwendungsbeispielen. Die im nächsten Abschnitt daraus hergeleiteten Eigenschaften von Bäumen eignen sich zum Teil zur Formulierung von gleichwertigen Baumdefinitionen, die Ihnen bei der Arbeit in der Schule oder auch in der Fachliteratur begegnen können.

## 3 Bäume

Bäume sind sympathische Objekte, das zeigt sich auch im Unterricht. Vielleicht liegt es am (positiven) Assoziationsreichtum des Begriffes, aber wohl auch an der Vielgestaltigkeit und ästhetischen Qualität von (graphentheoretischen) Bäumen. Die Definition ist sehr handlich, und daher muss nicht mit einer großen Anzahl von Voraussetzungen jongliert werden. Beim Nachdenken über die Problemstellungen vom Anfang sind Ihnen sicher schon spezielle Eigenschaften von Bäumen aufgefallen. Einigen Baumeigenschaften werden wir in diesem Abschnitt nachspüren. Damit begeben wir uns wieder experimentierfreudig und neugierig in das »Graphenlabor« (vgl. Kapitel 1, S. 8).

Im Unterricht oder in Ihrer eigenen Arbeit an diesem Thema kommt die Theorie über Bäume natürlich nicht so geballt vor wie in diesem Kapitel, sondern entwickelt sich aus dem Nachdenken über die Problemstellungen. Es bietet sich für die Buchform trotzdem an, die graphentheoretischen Aspekte zu bündeln, um die entsprechenden Resultate zur Hand zu haben, wenn sie z. B. für die Unterrichtsvorbereitung benötigt werden.

Sind Sie im Moment an einem schnellen Überblick über die Thematik interessiert, so sehen Sie sich die Aussagen, die in diesem Abschnitt erarbeitet und bewiesen werden, an, ohne sich in die Beweise zu vertiefen, und lesen Sie dann bei Abschnitt 4 weiter.

### *Eindeutigkeit der Wege*

Müssen Eichhörnchen nachdenken, wenn sie von einem Platz auf einem Baum an einen anderen Platz auf demselben Baum wechseln wollen? Das wissen wir natürlich nicht, aber mathematisch betrachtet gibt es eine Antwort auf diese Frage.

- Zeichnen Sie einen Baum. Wie viele verschiedene Wegmöglichkeiten hat ein Eichhörnchen, das von einem Astende oder einer Astgabel zu einem anderen Astende oder einer anderen Astgabel klettern möchte?

Sie haben sich vermutlich gefragt, welche Sorte von Baum nun gemeint war. Was sind die Unterschiede zwischen dem echten Baum und dem graphentheoretischen Modell? Findet sich die oben gesuchte Eigenschaft des Graphen-Baumes auch in realen Bäumen oder in Stammbäumen wieder? In welchen Fällen würde der Stammbaum diese Eigenschaft verlieren? Wie wirken sich verschiedene Gesellschaftsformen auf die Form der Stammbäume aus? Im Gegensatz zu beliebigen Graphen, in denen keine allgemeine Aussage über die Anzahl der Wege zwischen je zwei Knoten gemacht werden können (siehe Kapitel 1, Abschnitt 3), herrscht in Graphen-Bäumen absolute Klarheit:

#### ■ Satz

In einem Baum sind je zwei Knoten durch genau einen Weg verbunden.

Die Begründung ist relativ kurz, und daher ein gutes Übungsbeispiel für logisch stringentes Argumentieren. Nicht nur hier, sondern auch anderswo hat man ja oft das Gefühl, dass nichts zu beweisen sei, weil die zu beweisende Vermutung scheinbar offensichtlich stimmt. Eine Beweismöglichkeit ist, zu zeigen, dass es sowohl mindestens als auch höchstens einen Weg zwischen je zwei Knoten gibt. Wir haben Bäume so definiert, dass sie zusammenhängend sind. Also gibt es mindestens einen Weg zwischen je zwei Knoten. Andererseits gibt es in einem aufspannenden Baum keine doppelten Verbindungen zwischen je zwei Knoten, weil das ja einen Kreis ergeben würde. Also gibt es höchstens einen Weg zwischen je zwei Knoten. »Mindestens« und »höchstens« ergibt zusammen »genau einen«.

Diese zentrale Eigenschaft von Bäumen wird beispielsweise genutzt, wenn mittels der Breitensuche oder des Algorithmus von Dijkstra (siehe Kapitel 1) kürzeste Wege gesucht werden. Beide Algorithmen erzeugen aufspannende Bäume und geben für jeden Knoten einen eindeutigen kürzesten Weg zum Startknoten an.

### Die Anzahl der Baumkanten

Für Bäume können Aussagen getroffen werden, die für allgemeine Graphen undenkbar sind. Anhand von Überlegungen zur Anzahl der Baumkanten können Sie in diesem Abschnitt ein ganzes Bündel von Baumeigenschaften entdecken.

- Was passiert, wenn man eine Kante aus einem Baum entfernt oder ihm eine Kante hinzufügt? Wie kann der Baum dann wieder »repariert« werden? Zeichnen Sie Beispiele!
- Wieviele Kanten hat ein Baum mit  $n$  Knoten? Experimentieren Sie mit verschiedenen (aufspannenden) Bäumen!

Entfernt man eine Kante aus einem Baum, und zwar nur die Kante und nicht die zugehörigen Knoten, wird er stets in zwei Zusammenhangskomponenten zerfallen (dabei kann es natürlich passieren, dass eine Zusammenhangskomponente nur aus einem Knoten besteht). Das entspricht auch der Erfahrung: Stellen Sie sich einmal vor, Sie würden einen Ast eines Baumes durchsägen und er würde nicht zu Boden fallen, sondern wäre trotzdem noch an anderer Stelle festgewachsen. Sie würden sich doch sehr wundern! Also: nimmt man eine Kante weg, so erhält man – mathematisch gesprochen – einen Wald, der aus zwei Bäumen besteht. Eine Erfahrung, die man auch immer wieder macht, wenn man einen Ast absägt? Hier hört die Analogie zu echten Bäumen und Wäldern wieder auf. Leider entsteht beim Absägen eines Astes nicht gleich ein neuer Baum! Der mathematische *Baum* hat einiges mit seinem Namenspaten gemeinsam, aber eben nicht alles. Warum sind es immer genau zwei Zusammenhangskomponenten beim Entfernen einer Kante? Oder anders gefragt: Wie viele Zusammenhangskomponenten entstehen, wenn man mehrere Kanten aus dem Baum entfernt? Suchen Sie eine stichhaltige Begründung für Ihre Vermutung.

Sehen wir uns für eine weitere Überlegung nochmals den Wald an, der durch Wegnehmen einer Kante entstanden ist. Auf wie viele verschiedene Arten kann man daraus wieder einen Baum machen? Jede Kante, die einen Endknoten in der einen Komponente und den anderen in der zweiten Komponente hat, repariert den auseinandergefallenen Baum wieder.

Fügt man einem Baum eine Kante hinzu (siehe Abbildung 7), so entsteht ein Kreis. Um wieder einen Baum zu erhalten, kann irgendeine Kante aus dem Kreis wieder herausgenommen werden. So lassen sich Bäume nach Bedarf umbauen. Ist das immer so? Hängt es nicht vom jeweiligen Baum ab? Es scheint so zu sein, dass ein Baum seine Eigenschaften verliert, sobald Kanten hinzugefügt oder weggenommen werden. Ein Baum hat also gerade so viele Kanten, dass er zusammenhängend ist und nicht in mehrere Teile zerfällt. Und er hat die maximale Anzahl von Kanten, so dass kein Kreis entsteht. Oder etwas kürzer ausgedrückt:



### ■ Satz

Ein Baum ist minimal zusammenhängend und maximal kreisfrei.

Das ist ja doch eine recht starke Aussage. Haben Sie versucht, sie zu begründen? Für einen Beweis kann die Baumeigenschaft der eindeutigen Wege genutzt werden: In einem Baum sind je zwei Knoten durch einen eindeutigen Weg miteinander verbunden. Nimmt man eine Kante aus dem Baum heraus, so wird dadurch mindestens einer dieser eindeutigen Wege unterbrochen. Dann kann der Restgraph nicht mehr zusammenhängend sein. Fügt man dem Baum eine Kante hinzu – wir nehmen an, sie verbindet die Knoten  $a$  und  $b$  – schafft man dadurch neben den im Baum bereits vorhandenen Weg einen weiteren (nur eine Kante langen) Weg zwischen  $a$  und  $b$ . Beide  $a$ - $b$ -Wege zusammen ergeben einen Kreis.

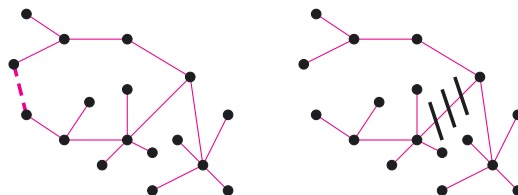
Bei der Suche nach aufspannenden Bäumen wäre es interessant zu wissen, wie überprüft werden kann, ob man tatsächlich einen aufspannenden Baum gezeichnet hat, und nicht etwa eine Kante zu viel oder zu wenig ausgewählt hat. Kreise zu suchen oder Zusammenhangskomponenten zu zählen, kann unter Umständen sehr zeitaufwändig sein, wenn der Graph groß und unübersichtlich ist. Glücklicherweise gibt es ein leichter zu handhabendes Kriterium, das Sie bei der Bearbeitung der obigen Aufgaben sicher durch Ausprobieren und Abzählen schnell gefunden haben: Hat der Graph  $n$  Knoten, so hat der aufspannende Baum  $n - 1$  Kanten. Etwas verallgemeinert kann man sagen:

### ■ Satz

Ein Baum mit  $n$  Knoten hat  $n - 1$  Kanten.

Eine erste Beweisidee, die auch Schülern spontan einfällt, könnte sein, einen Baum Schritt für Schritt zu konstruieren. Der Baum mit einem Knoten hat keine Kante. Will man diesem Baum einen weiteren Knoten hinzufügen, so bindet man den Knoten mit genau einer Kante an. Auch jeder weitere hinzukommende Knoten wird mit genau einer Kante angebunden. Bindet man nämlich einen Knoten mit zwei Kanten an, so muss ein Kreis entstehen, da der schon vorhandene Graph ein Baum ist. Die zwei neuen Kanten müssten an zwei (nicht notwendigerweise verschiedene) Knoten angebunden werden und es ergibt sich dann die gleiche Situation wie beim Hinzufügen einer zusätzlichen Kante in einen fertigen Baum (s. o.).

Abbildung 7. Fügt man einem Baum an irgendeiner Stelle eine weitere Kante hinzu, entsteht ein Kreis (Ausschnitt aus dem Telefonnetz). Wird eine Kante herausgenommen, so zerfällt der Baum.



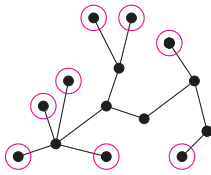


Abbildung 8. Dieser Baum hat 8 Blätter

Daher muss der schrittweise aufgebaute Baum auf  $n$  Knoten  $n - 1$  Kanten haben.

Dieser Beweis taucht vereinzelt auch in der Literatur auf. Er hat neben dem Vorteil, dass er direkt einleuchtet, aber den Nachteil, dass wir für bereits vorhandene Bäume nicht einfach so garantieren können, dass sie auf diese Weise gebaut werden können. Das muss bewiesen werden und ist etwas umständlich (aber schön aufgeschrieben in [14, S. 180]).

Ein recht anschaulicher Beweis kann mit Hilfe der Technik des »sukzessiven Abpflückens von Blättern« geführt werden. Es ist die reziproke Vorgehensweise zur schrittweisen Konstruktion. Wir machen den Baum solange schrittweise kleiner, bis uns etwas Bekanntes begegnet. Ein *Blatt* ist – wie man es sich auch anschaulich vorstellt – ein Knoten in einem Baum, von dem nur eine Kante ausgeht (siehe Abbildung 8). Auch in anderen Zusammenhängen spielt die Anzahl der Kantenenden, die in einen Knoten münden eine Rolle, daher gibt es dafür einen Begriff, den *Grad* eines Knotens.

### ■ Definitionen

Die Anzahl der in einen Knoten mündenden Kantenenden heißt der *Grad* des Knotens.

Die Knoten eines Baumes mit Grad 1 heißen *Blätter*.

Zunächst müssen wir beweisen, dass ein Baum überhaupt Blätter hat, damit man auch etwas abpflücken kann. Auch hier könnte es ja wieder sein, dass es ein unerwartetes untypisches Beispiel gibt, also irgendeinen merkwürdigen Baum, der kein Blatt hat. Die Existenz eines einzigen solchen Gegenbeispiels, auch wenn es noch niemand je gesehen hat, würde genügen, um die Vermutung zu widerlegen. Versuchen Sie eine Argumentation zu finden, dass ein Baum immer Blätter hat.

Der Beweis dieser Vermutung ist schon etwas komplizierter: Wir betrachten einen Baum  $B$ .  $B$  soll mindestens zwei Knoten haben. Sei  $W$  ein längster Weg in diesem Baum bezüglich der Anzahl der Kanten. Wie man einen solchen längsten Weg findet, interessiert uns im Moment nicht. Wir wissen, dass es in dem Baum Wege geben muss, da der Baum alle seine Knoten miteinander verbindet. Unter diesen Wegen gibt es einen oder auch mehrere längste Wege, von denen wir uns einen aussuchen.

Wir benennen die beiden Endknoten dieses längsten Weges mit  $a$  und  $z$ . Nun kann man zeigen, dass  $a$  und  $z$  beide Grad 1 haben. Hätte etwa  $a$  mehr als einen Nachbarknoten im Weg, beispielsweise die Knoten  $b$  und  $c$ , so könnte man den Weg  $W$  um mindestens eine Kante verlängern.

Dies widerspricht aber der Annahme, dass  $W$  ein längster Weg in  $B$  ist. Die gleiche Argumentation kann man für  $z$  führen. Also gibt es in dem Baum  $B$  mindestens zwei Blätter:  $a$  und  $z$ .

### ■ Satz

Jeder Baum (mit mindestens 2 Knoten) besitzt mindestens zwei Blätter.

Wir wissen also nun, dass ein Baum stets Blätter besitzt. Wir müssen uns jetzt überlegen, dass das, was übrig bleibt, wenn man von einem Baum ein Blatt mit zugehöriger (eindeutiger) Kante abpflückt, immer noch ein Baum ist. Da nichts hinzugefügt wird, bleibt der Graph kreisfrei. Zusammenhängend bleibt er auch, da die Kante, die abgepflückt wird, nicht mitten in einem Weg zwischen zwei Knoten des restlichen Graphen liegen kann. Der abgepflückte Knoten hatte ja Grad 1. Daher bleiben die eindeutigen Wege zwischen je zwei Knoten des Restgraphen erhalten. Nach dem Abpflücken eines Blattes bleibt also ein Graph übrig, der wieder ein Baum ist.

Nach dieser ganzen Vorarbeit kann die Vermutung, dass ein Baum mit  $n$  Knoten  $n - 1$  Kanten besitzt, sehr elegant bewiesen werden. Sie pflücken Blatt um Blatt von einem Baum ab. Beobachten Sie dabei die Veränderung der Anzahl der Knoten und der Kanten. Suchen Sie eine Argumentation, die den Satz über die Anzahl der Kanten im Baum beweist. (Einen weiteren Beweis für diesen Satz finden Sie in Kapitel 3 auf Seite 87. Dort wird mit Knotengraden argumentiert.)

Die Anzahl der Kanten in dem nun betrachteten Baum  $B$  wird mit  $|E|$  bezeichnet, die Anzahl der Knoten mit  $|V|$  ( $E$  von engl. *edges* und  $V$  von engl. *vertices*). Der Baum hat  $n$  Knoten, anders ausgedrückt:  $|V| = n$ . Pflücken Sie ein Blatt und die dazugehörige Kante (wenn man so will, den Stiel) ab. Es entsteht ein neuer Baum  $B_1$  mit  $|V| - 1$  Knoten und  $|E| - 1$  Kanten. Die Differenz zwischen der Anzahl Knoten und der Anzahl Kanten im alten Baum und die entsprechende Differenz im neuen Baum sind gleich, da genau ein Knoten und eine Kante entfernt wurden:  $|V| - |E| = |V_1| - |E_1|$ . Diesen Vorgang des Abpflückens von einzelnen Blättern wiederholen Sie so lange, bis Sie nach  $n - 1$  Schritten den Baum  $B_{n-1}$  auf einem Knoten erhalten. Dieser Baum hat keine Kante, d. h. es gilt:  $|V_{n-1}| - |E_{n-1}| = 1 - 0 = 1$ . Außerdem ist  $|V| - |E| = |V_{n-1}| - |E_{n-1}|$ , weil in jedem Schritt je eine Kante und ein Knoten entfernt wurden. Also gilt  $|V| - |E| = 1$ , bzw.  $n - |E| = 1$  (die Anzahl der Knoten ist  $n$ ), was äquivalent ist zu »Die Anzahl der Kanten im Baum  $B$  ist  $n - 1$ «.

### Die Anzahl der aufspannenden Bäume

Sicherlich haben Sie bei der Arbeit an den eingangs geschilderten Problemsituationen bemerkt, dass es zu einem Graphen mehrere verschiedene aufspannende Bäume geben kann. Da liegt es doch nahe, einmal nachzuforschen, wie viele verschiedene aufspannende Bäume ein Graph hat. Leider ist es im Allgemeinen schwierig, eine solche Angabe zu machen. Für den Spezialfall der vollständigen Graphen (d. h. jeder Knoten ist mit jedem anderen durch eine Kante verbunden) ist es aber möglich, eine Formel zu finden. Schätzen Sie zunächst, wie viele aufspannende Bäume es auf dem vollständigen Graphen mit 4, 5 oder 10 Knoten gibt.

- Zeichnen Sie 4 Knoten auf ein Blatt Papier. Bezeichnen Sie die Knoten mit Buchstaben oder Zahlen. Auf wie viele verschiedene Arten können Sie einen aufspannenden Baum einzeichnen?
- Wiederholen Sie das Experiment mit 2, 3 und 5 bezeichneten Knoten. Formulieren Sie eine Vermutung für die Anzahl der aufspannenden Bäume auf  $n$  Knoten. Suchen Sie Beweisideen!

Es ist für das Abzählen ganz entscheidend, den Knoten Namen zu geben, da alle kombinatorischen Möglichkeiten berücksichtigt werden sollen (siehe Abbildung 9). Bei 2 Knoten gibt es nur einen aufspannenden Baum, bei 3 Knoten gibt es 3, bei 4 Knoten 16 und bei 5 Knoten 125. Haben Sie die Formel gefunden?

#### ■ Die Cayley-Formel

Die Anzahl der aufspannenden Bäume für den vollständigen Graphen mit  $n$  Knoten ist  $n^{n-2}$ .

Für diesen Satz gibt es viele verschiedene Beweise, die sich der unterschiedlichsten Methoden bedienen (Eine schöne Zusammenstellung findet man in [15]). Ein Klassiker ist der Beweis mit dem so genannten *Prüfer-Code*. Er ist aus zweierlei Gründen für die Schule gut geeignet: Er gibt einen leicht nachvollziehbaren Algorithmus an, der teilweise durch selbstständiges Nachdenken gefunden werden kann. Zweitens gibt er einen ganz kleinen Ausblick in den Bereich der Verschlüsselungen, der nicht nur wegen seines erstaunlich starken Einflusses auf den Verlauf der Geschichte sehr faszinierend ist.

Der Beweis dieses Satzes zeigt die Gleichmächtigkeit von zwei Mengen durch eine Bijektion. Die eine Menge ist die Menge der aufspannenden Bäume auf dem

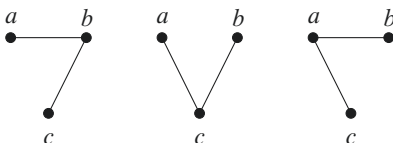


Abbildung 9. Die drei Bäume auf drei Knoten. Lässt man keine Umbenennung der Knoten zu, so sind diese drei Bäume nicht isomorph.

vollständigen Graphen mit  $n$  Knoten, von der wir vermuten, dass sie die Mächtigkeit  $n^{n-2}$  hat. Nun braucht man noch eine zweite Menge, von der man sicher weiß, dass sie die Mächtigkeit  $n^{n-2}$  hat. Dafür bietet sich die Menge der  $(n-2)$ -Tupel mit Einträgen aus  $\{1, \dots, n\}$  an, denn es gibt  $n^{n-2}$  viele  $n-2$ -Tupel, das kann man sich leicht überlegen. Jetzt gilt es zu zeigen, dass jedem aufspannenden Baum genau ein solches Tupel (der Code) zweifelsfrei zugeordnet werden kann und dass aus jedem  $(n-2)$ -Tupel ein aufspannender Baum konstruiert werden kann.

Der zu betrachtende Baum bekommt als erstes Knotennummern. Versuchen Sie zunächst selbst, aus einem Baum mit  $n$  durchnummerierten Knoten einen  $(n-2)$ -stelligen Code zu gewinnen, also eine Reihen aus  $n-2$  Zahlen. Vielleicht hilft dabei die Vorstellung, dass man jemandem am Telefon einen Baum beschreiben möchte, und daher nicht irgendetwas zeichnen oder zeigen kann. Experimentieren Sie mit verschiedenen Nummerierungen und vielen verschiedenen Bäumen, um ein Verfahren zu finden, das jedem Baum einen eindeutigen Code zuordnet. Versuchen Sie auch umgekehrt, aus einem vorgegebenen Code einen Baum zu erzeugen. Bekommen Sie wieder den gleichen Baum heraus, wenn sie einen Baum erst in einen Code umwandeln und dann wieder zurückverwandeln?

Folgende Tipps können noch hilfreich sein: Beginnen Sie Ihre Untersuchung mit Bäumen mit einer geringen Knotenzahl. Aber welcher Knotenanzahl wird eine Codierung überhaupt erst benötigt? Vergrößern Sie kleine Bäume schrittweise. Verwenden Sie Verfahren der Baumerweiterung um aus den »kleinen« Bäumen große zu erzeugen. Berücksichtigen Sie die Ihnen bekannten Eigenschaften von Bäumen.

Diese Aufgabe ist zugegebenermaßen sehr knifflig, aber auch lohnend. Wenn Sie genug geknobelt haben, sehen Sie sich die folgende klassische Lösung an.

Zuerst beschreiben wir eine Abbildung von der Menge der aufspannenden Bäume für den vollständigen Graphen mit  $n$  Knoten in die Menge der  $(n-2)$ -Tupel. Oder anders ausgedrückt: Jedem Baum wird ein  $(n-2)$ -stelliger Code zugeordnet, der so genannte *Prüfer-Code*.

1. Zunächst werden die Knoten des Baumes auf beliebige Weise mit den Zahlen  $1, \dots, n$  durchnummeriert.
2. Dann sucht man das Blatt mit der kleinsten Nummer. Die Nummer des Nachbarknotens wird notiert.
3. Das Blatt mit zugehöriger Kante wird entfernt.
4. Die Schritte 2 und 3 werden so lange wiederholt, bis noch zwei Knoten mit einer sie verbindenden Kante übrig bleiben.

Zwei Eigenschaften von Bäumen werden hier genutzt: 1. Nach dem Abpflücken eines Blattes von einem Baum bleibt ein Baum übrig, und 2. Ein Baum mit mindestens 2 Knoten hat immer Blätter. Ohne diese beiden weiter oben bewiesenen Aussagen könnte es passieren, dass man irgendwann plötzlich kein Blatt mehr findet, also die Codierung nicht durchführen könnte.

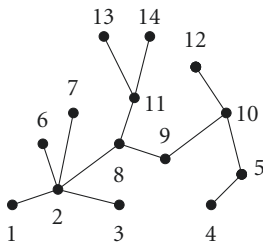


Abbildung 10. Der Prüfer-Code für diesen Baum lautet (2, 2, 5, 10, 2, 2, 8, 10, 9, 8, 11, 11). Versuchen Sie, aus dem Code den Baum zu rekonstruieren.

Für jeden Baum erhält man so eine  $(n - 2)$ -stellige Codenummer (vgl. Abbildung 10). Wir brauchen jetzt noch eine Vorschrift, die aus einem gegebenen Tupel in eindeutiger Art und Weise einen Baum erzeugt. Damit wird eine Abbildung von der Menge der  $(n - 2)$ -Tupel in die Menge der Bäume auf  $n$  Knoten definiert. Falls Sie es nicht schon getan haben, suchen Sie eine solche Vorschrift, die aus dem Prüfer-Code wieder einen Baum macht!

Eine Möglichkeit, aus dem Prüfer-Code einen Graphen zu konstruieren, besteht darin, dass man sich zwei Zahlenfolgen aufschreibt: die Zahlen  $1, \dots, n$  in eine obere Zeile und den Prüfer-Code darunter. Außerdem kann man sich schon einen leeren (kantenlosen) Graphen mit  $n$  durchnummerierten Knoten aufzeichnen. Das kleinste Blatt steht nicht in dem Code, da nur dessen Nachbarknoten notiert wurde, und das Blatt dann entfernt wurde. Das heißt, im ersten Schritt müssen Sie in der oberen Zeile die kleinste nicht in der Codenummer enthaltene Zahl suchen. Sie streichen diese durch, und verbinden diesen Knoten mit dem Knoten, der als erster im Code steht. Die erste Zahl aus dem Code wird ebenfalls durchgestrichen. Dieses Vorgehen wird wiederholt, bis  $n - 2$  Kanten gezeichnet sind. Die letzte Kante ergibt sich aus den beiden in der oberen Zeile übriggebliebenen Zahlen.

Zeichnen die Schülerinnen und Schüler nach dieser Vorschrift Bäume, so werden sie, je nach dem, was im Unterricht vorausgegangen ist, wieder oder zum ersten Mal feststellen, dass derselbe Graph sehr unterschiedlich aussehen kann. Das Phänomen der Graphenisomorphie wird hier unmittelbar deutlich (vgl. Kapitel 1, Abschnitt »Graphenisomorphie«, S. 10).

Zwischen beiden Mengen haben wir nun Abbildungen definiert. Es bleibt aber zu beweisen, dass ein aus einem Code erzeugter Graph wirklich immer ein Baum ist, und dass der Prüfer-Code eines so erzeugten Graphen gleich dem ursprünglichen Code ist. Erst dann ist wirklich klar, dass die Konstruktionsvorschriften tatsächlich eine Bijektion definieren. Um den Rahmen dieses Buches nicht zu sprengen, wird hier auf diesen Teil des Beweises verzichtet und die Lektüre des entsprechenden Abschnittes in [15, S. 255 ff.] empfohlen.

## 4 Die Tiefensuche

### *Der Algorithmus*

Sie wissen nun, was ein aufspannender Baum ist, und kennen einige interessante Eigenschaften von Bäumen. Wie aber konstruiert man aufspannende Bäume? Die Graphen, die in der Praxis vorkommen, sind häufig sehr groß und daher nicht mehr gut von Hand zu bearbeiten. Wie kann man Computer dazu bringen, aufspannende Bäume zu erzeugen bzw. zu erkennen? Eine Möglichkeit kennen Sie schon aus dem ersten Kapitel: die Breitensuche. Das Erzeugnis der Breitensuche ist ein aufspannender Baum. Spontan und ohne Vorwissen würden Sie allerdings vermutlich nicht die Breitensuche (nach-)erfinden, sondern einen anderen Algorithmus.

- Suchen und formulieren Sie einen Algorithmus, der einen aufspannenden Baum in einem zusammenhängenden Graphen findet. Spielen Sie verschiedene Szenarien mit Papier und Farbstiften durch. Was tun Sie, wenn es irgendwo nicht mehr weitergeht?

Hier hilft wieder die Vorstellung, dass der Computer stets nur einen einzelnen Knoten und die davon ausgehenden Kanten »sehen« kann, aber nicht den ganzen Graphen (vgl. Abschnitt »Froschperspektive« im ersten Kapitel). Basteln Sie sich eine Lochblende (Abbildung 17 in Kapitel 1) bzw. bitten Sie Ihre Schülerinnen und Schüler, das zu tun. Zeichnen Sie einen Graphen, wählen Sie einen Startknoten und legen Sie die Lochblende darauf. Und los geht's.

Wir werden eine der möglichen Vorgehensweise genauer analysieren und formulieren. Eine Idee, auf die Sie wahrscheinlich gerade eben auch gekommen sind, ist, nun einfach von Knoten zu Knoten zu laufen (bzw. zu malen). Damit erzeugen Sie einen Weg in dem Graphen (zur Erinnerung: ein Weg ist ein Kantenzug ohne Knotenwiederholungen). Irgendwann werden Sie an einen Knoten kommen, an dem es nicht mehr weitergeht. Wie viele verschiedene Arten von »nicht mehr Weiterkönnen« haben Sie gefunden? Wie verhalten Sie sich, wenn Sie steckenbleiben?

Zwei Situationen machen Probleme: Man kann in eine »Sackgasse« geraten sein. Das ist ein Knoten vom Grad 1. Oder man erzeugt gerade einen Kreis. Doch woran merkt man, ob gerade ein Kreis entsteht? Haben Sie sich das überlegt? Zum Glück ist das ganz einfach zu erkennen: Endet die gerade markierte Kante in einem Knoten, der schon zuvor in den gerade entstehenden Baum eingebunden wurde, so schließt diese Kante einen Kreis in dem aufspannenden Baum. Das muss vermieden werden.

Was tun in diesen beiden Fällen? Ist man in eine Sackgasse geraten, so muss man aus dieser wieder hinaus und soweit entlang des gerade eben eingefärbten Weges rückwärts gehen, bis ein Knoten kommt, der noch unbesuchte Kanten be-

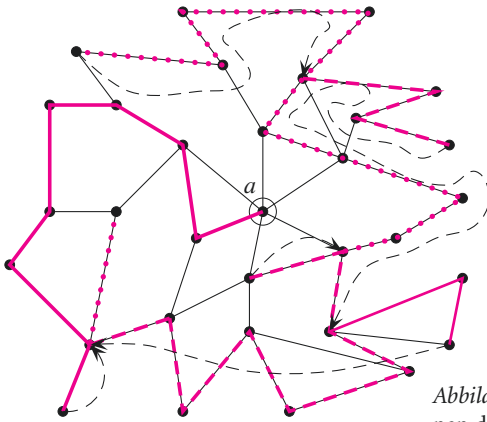


Abbildung 11. Die Tiefensuche: Die Pfeile bezeichnen das Backtracking.

sitzt. Dieses Rückwärtslaufen wird *Backtracking* genannt (vgl. Abbildung 11). Von dem so gefundenen Knoten geht man von Neuem los, bis es wieder nicht mehr weitergeht usw. Im zweiten Fall macht man beinahe das gleiche. Allerdings muss die Kante, die in einem bereits besuchten Knoten endet, ihre Markierung wieder verlieren. Arbeitet man mit Papier und Buntstiften müsste man also radieren.

Dieser Algorithmus wird *Tiefensuche* oder *Depth-First-Search (DFS)* genannt. Der Name kommt daher, dass man in jeder Iteration in die Tiefe des Graphen läuft, bis es auf diesem Weg nicht mehr weitergeht. Je nach dem, wie die jeweils nächste Kante ausgewählt wird, kann es vorkommen, dass in der ersten Iteration des Algorithmus bereits alle Knoten abgelaufen werden, wie etwa in Abbildung 12, in den meisten Fällen wird man jedoch mehrmals neu ansetzen müssen.

### ■ Die Tiefensuche ■

Eingabe: ein zusammenhängender Graph

Ausgabe: ein aufspannender Baum des Graphen

1. Wähle einen Startknoten. Markiere ihn.
2. Falls es noch unmarkierte Kanten gibt, die von dem Knoten ausgehen: Wähle eine dieser Kanten und prüfe, welche der beiden folgenden Anweisungen auszuführen ist. Anderenfalls gehe zu 3.
  - Ist der Endknoten dieser Kante noch unmarkiert, so markiere diese Kante und den Endknoten der Kante. Schreibe die Kante hinten in eine Liste der markierten Kanten. Wiederhole 2. für den neu markierten Knoten.
  - Ist der Endknoten bereits ein Baumknoten, so wird diese Kante aus dem Graphen gestrichen. Gehe zum Anfangsknoten der Kante zurück und wiederhole 2.
3. Prüfe, ob bereits alle Knoten markiert sind. Falls ja: Stopp, falls nein, weiter mit 4.



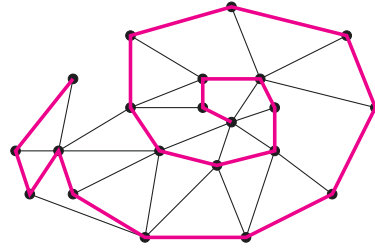


Abbildung 12. Die erste Iteration der Tiefensuche hat bereits den aufspannenden Baum konstruiert.

4. »Backtracking«: Gehe entlang der zuletzt markierten Kanten zurück, indem du die Liste der markierten Kanten rückwärts abarbeitest und die jeweils verwendete Kante daraus wieder streichst.
  - Triffst du auf einen Knoten, von dem noch unmarkierte Kanten ausgehen, gehe zu 2.
  - Anderenfalls: Stopp.

Auch diese Formulierung des Algorithmus ist wieder nur eine von unzählig vielen Möglichkeiten (eine weitere Formulierungsvariante findet sich auf S. 61). Die hier gewählte verwendet eine IF-THEN/ELSE-Verzweigung (»Falls ... tue dies. Anderenfalls ...«), die beim Programmieren sehr gebräuchlich ist.

Für den Mathematikunterricht sollte es vor allem darum gehen, überhaupt einen funktionierenden Algorithmus zu formulieren. Denn von einem mathematischen Blickwinkel aus betrachtet, steht das Verständnis der Funktionsweise des Algorithmus im Vordergrund, nicht seine Implementierung. Für die konkrete Implementierung muss man sich natürlich sehr viel mehr Gedanken machen, welche Variante des Algorithmus für die jeweilige Anwendung am sinnvollsten ist. Dazu gehören dann auch Laufzeitabschätzungen, so dass hier ein idealer Anknüpfungspunkt zum Informatikunterricht gegeben ist.

Die Abbruchbedingung, dass beim Backtracking keine unbesuchten Kanten mehr gefunden werden, ist wichtig für den Fall, dass man den Algorithmus auf beliebige Graphen anwenden möchte. Ist der Graph, auf dem der Algorithmus durchgeführt werden soll, nicht zusammenhängend, so bricht die Tiefensuche ab, sobald sie eine Zusammenhangskomponente erkundet hat und für diese Komponente einen aufspannenden Baum konstruiert hat. Würde man den Algorithmus laufen lassen wollen, bis alle Knoten Baumknoten geworden sind, was für zusammenhängende Graphen eine vernünftige Abbruchbedingung ist, so würde der Algorithmus in solch einem Fall unendlich lange weitersuchen.

Weil der Algorithmus abbricht, sobald eine Zusammenhangskomponente komplett exploriert wurde, ist er geeignet, die Zahl der Zusammenhangskomponenten eines Graphen festzustellen. Für jede Komponente muss die Tiefensuche auf dem noch nicht betrachteten Restgraphen neu gestartet werden. Analog geht das auch mit der Breitensuche.

### *Korrektheitsbeweis*

Die Tiefensuche gehört zu den einfachsten Graphenalgorithmien. Sie kommt mit wenigen Anweisungen aus, so dass der Korrektheitsbeweis übersichtlich ausfällt. Wir wollen zeigen, dass die Tiefensuche in zusammenhängenden Graphen tatsächlich einen aufspannenden Baum erzeugt. Dazu müssen wir drei Kriterien prüfen: Ist das Produkt der Tiefensuche zusammenhängend, ist es kreisfrei und beinhaltet es alle Knoten?

Dadurch, dass die Tiefensuche sich beim Backtracking stets an den bereits markierten Kanten entlanghangelt und in jeder Iteration neue Äste an den bereits konstruierten Baum anhängt, muss das Endprodukt zusammenhängend sein. Da wir per Konstruktionsvorschrift Kreise vermeiden, ist das Produkt am Ende auch kreisfrei. Ob tatsächlich alle Knoten eingebunden werden, müssen wir noch zeigen. Nehmen wir an, der Algorithmus bricht ab und es sind noch nicht alle Knoten des Graphen Baumknoten geworden. Da der Algorithmus stoppt, wurden beim Backtracking alle Kanten abgelaufen und die zu ihnen gehörigen Knoten besucht. Da es aber noch unbesuchte Knoten gibt, müssen diese zu anderen Komponenten des Graphen gehören. Damit besteht der Eingabegraph aus mehreren Komponenten, was im Widerspruch zu unseren Voraussetzungen steht.

### *Das Daumenkino und noch einmal die Lochblende*

Das Nacherfinden der Tiefensuche ist der Einsatzbereich par excellence für das methodische Werkzeug »Lochblende«. Geht es um die Entwicklung der Breiten-suche, so muss schon relativ bald von dem Bild der Froschperspektive abstrahiert werden, da man zum jeweils nächsten aktiven Knoten quasi springen muss. Die Tiefensuche hingegen verläuft ganz linear, so dass sie tatsächlich komplett mit der Lochblende durchgeführt werden kann.

Probieren Sie es aus: Legen Sie die Lochblende auf einen Graphen und führen Sie die Tiefensuche aus, ohne die Lochblende hochzuheben. Sie müssen sie nur verschieben. Wenn Sie das Backtracking durch »Rückwärtskanten« (wie in Abbildung 11) einzeichnen und gelöschte Kanten z. B. mit einer Wellenlinie durchstreichen, so müssen Sie während der Prozedur nicht einmal den Stift anheben. Er bleibt die ganze Zeit im Fenster der Lochblende.

Der Verlauf des Algorithmus wird hier besonders anschaulich und (be-)greifbar. Diese Thematik eignet sich daher gut als erste Annäherung an das Erfinden von Algorithmen in der Mittelstufe oder auch schon für jüngere Schülerinnen und Schüler. Arbeiten Sie mit der Lerngruppe heraus, was ein aufspannender Baum ist, lassen Sie Lochblenden basteln und Graphen bauen und dann kann das Forschen losgehen: Wie kann man aufspannende Bäume konstruieren?

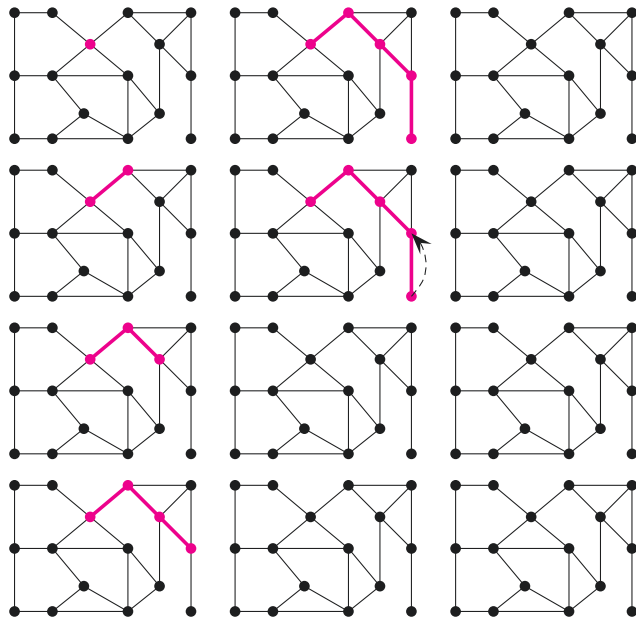


Abbildung 13. Daumenkinos zu basteln, hilft, Algorithmen besser zu verstehen.

Ist ein Algorithmus gefunden, so lohnt es sich, zum noch tieferen Verständnis eine Schulstunde zu investieren und Daumenkinos basteln zu lassen. Dazu benötigt man reichlich Kopien ein und desselben Graphen und eine Klammermaschine. Aufgabe ist, pro Kopie des Graphen einen Schritt des Algorithmus einzuzichnen, so dass man sich den gesamten Ablauf des Algorithmus nachher als Daumenkino ansehen kann (vgl. Abbildung 13).

Neben der schönen Auflockerung, im Mathematikunterricht zu basteln, bringt die Erstellung des Daumenkinos einen hohen Erkenntnisgewinn. Durch die Auffächerung des Ablaufes in Einzelbilder werden oft noch Unklarheiten entdeckt (z. B.: Was geschieht mit Sackgassen? Oder: Wie können Kreise vermieden werden?). Zudem wird deutlich, dass auf dem Graphen viele unterschiedliche aufspannende Bäume entstehen können, denn vermutlich wird die Tischnachbarin einen anderen Baum konstruieren als man selber. Aber alle aufspannenden Bäume sind gleichermaßen richtig. Diese Beobachtung führt direkt zur Frage nach der Anzahl der aufspannenden Bäume eines Graphen (vgl. S. 50).

Die Tatsache, dass bei der Ausführung desselben Algorithmus verschiedene Bäume herauskommen können, kann als Aufhänger für die Suche nach einer Festlegung des Ablaufes genutzt werden. Ab Seite 60 werden wir uns darüber Gedanken machen und diese Frage mittels einer speziellen Datenstruktur lösen.

*Exkurs: Ariadne – die erste Informatikerin*

Gelegentlich kann es sehr nützlich, ja sogar lebensrettend sein, einige Algorithmen zu kennen. Stellen Sie sich vor, Sie hätten sich in einer verzweigten Höhle verlaufen. Wie finden Sie zurück zum Ausgang? Oder, was etwas unwahrscheinlicher ist, Sie möchten sich in einem Labyrinth zurechtfinden. Zumindest Harry Potter befand sich schon in dieser Situation. Wie gehen Sie vor?

- Sie wollen in einer Ihnen unbekannten Höhle den Weg zum Ausgang finden oder in einem Labyrinth den Weg zum Ziel finden und dann auch wieder hinaus. Was tun Sie? Schreiben Sie einen Algorithmus.

Vielleicht haben Sie die Rechte-Hand-Regel befolgt: Bleibe mit der rechten Hand stets in Kontakt mit der Wand und laufe immer weiter. Eine noch bekanntere Beschreibung des gleichen Lösungsverfahrens findet man in der griechischen Mythologie.

Der früheste Algorithmus zur Lösung des Labyrinthproblems stammt von der kretischen Königstochter Ariadne. Sie war vermutlich die erste Informatikerin der Geschichte. Folgende Geschichte wurde überliefert:

Auf Kreta hatte man einst ein Problem. Der Stiefsohn von König Minos war ein Ungeheuer, halb Mensch, halb Stier: der Minotaurus. Der Minotaurus war immer sehr hungrig und hatte vor allem Appetit auf Menschenfleisch. Um die Kreter zu schützen, wurde für den Minotaurus ein Gebäude errichtet, aus dem er nicht mehr herauskommen konnte. Die Reste eines solchen Labyrinth-Bauwerks, das aus dem 17. Jahrhundert vor Christus stammt, findet man heute noch in Knossos. Jahr für Jahr wurden dem Minotaurus allerdings sieben junge Frauen und sieben junge Männer aus Athen zum Fraß vorgeworfen. Diese abscheuliche Verpflichtung hatte Athen nach einer Niederlage gegen Kreta eingehen müssen.

Theseus, der Sohn des Königs von Athen, hatte sich in den Kopf gesetzt, diesem grausamen Vertrag ein Ende zu machen, indem er den als unbesiegbar geltenden Minotaurus tötete. Er fuhr mit nach Kreta, wo er Ariadne, die Tochter des Minos traf. Die beiden verliebten sich ineinander und Ariadne war besorgt, dass Theseus niemals mehr aus dem Labyrinth herausfinden würde. Sie gab ihm ein Wollknäuel, hielt das Ende des Wollfadens am Eingang zum Labyrinth fest und gab folgende Anweisung: Theseus solle in das Labyrinth hineinlaufen und dabei stets den Faden abwickeln, so dass er eine Spur seines Weges hinterlasse. An jeder Kreuzung solle er den am weitesten rechts liegenden Gang wählen. Geriete er in eine Sackgasse, solle er umkehren, den Faden wieder aufwickeln und an der nächsten Kreuzung die nächstrechte Abzweigung nehmen, dort weitergehen, den Faden wieder abwickeln und so weiter.

Theseus folgte diesen Anweisungen, fand den Minotaurus, erschlug ihn und konnte dann – immer dem Faden nach – auf direktem Weg zu Ariadne zurück-



Abbildung 14. Der Ariadnefaden im Labyrinth

laufen. Leider konnten sie ihre Zweisamkeit nicht sehr lange genießen, da Theseus Ariadne auf dem Weg nach Athen aus etwas diffusen Gründen auf Naxos zurückließ, wo sie starb.

- Spielen Sie die Anweisungen Ariadnes auf Papier oder, falls möglich, in einem echten oder (z. B. mit Tischen und Stühlen) selbst gebauten Labyrinth nach!
- »Übersetzen« Sie das Labyrinth aus Abbildung 14 in einen Graphen, führen Sie den Ariadne-Algorithmus daran nochmals aus und fassen ihn in eigene Worte. Tipp: Nehmen Sie für das Abwickeln und für das Wiederaufwickeln des Fadens verschiedene Farben.

Was können Sie in ihren Skizzen erkennen? Es gibt einen Backtracking-Mechanismus, nämlich dann, wenn der Faden wieder aufgewickelt wird. Wenn Sie das ganze Labyrinth ablaufen, werden Sie am Ende wieder am Startpunkt herauskommen. Sie können zudem beobachten, dass (wenn die Backtracking-Kanten nicht betrachtet werden) ein Baum entsteht. Der Ariadne-Algorithmus ist eine Tiefensuche!

Meist wird der Ariadne-Algorithmus nicht den kompletten aufspannenden Baum konstruieren, da das Ziel ist, einen bestimmten Ort im Labyrinth zu erreichen und nicht, alle Wege abzusuchen. Daher muss man für das Labyrinth-Problem ggf. nur die Abbruchbedingung anders formulieren, als wir das für die Tiefensuche weiter oben getan haben.

Um die Analogie komplett zu machen, muss das Labyrinth als Graph modelliert werden. Dass Kreuzungen, der Eingang und Sackgassenenden als Knoten dargestellt werden und Verbindungswege als Kanten, ist schnell klar. Eine Adjazenzmatrix lässt sich leicht aufstellen. Die Schwierigkeit hier liegt darin, eine schöne und übersichtliche zeichnerische Darstellung zu finden. *Graphenzeichnen* ist ein Forschungsgebiet der diskreten Mathematik. Es ist und bleibt ein schwieriges und komplexes Problem, automatisiert gute Zeichnungen für Graphen erstellen zu lassen. In Abbildung 15 ist eine Darstellung des zum Labyrinth aus Abbildung 14



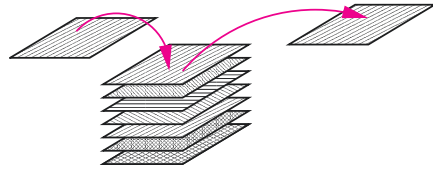


Abbildung 16. Ein »Stack« oder »Stapel«: damit lässt sich die Tiefensuche elegant formulieren.

Es liegt nahe, an dieser Stelle neugierig zu sein, und der Frage nachzugehen: Was passiert denn, wenn man statt FIFO einfach LIFO (Last In First Out) macht? Probieren Sie es aus!

Bei diesem Experiment werden Sie nicht mehr die Breitensuche herausbekommen, sondern etwas, das der Tiefensuche zumindest ähnelt. Wie müssen Sie den Algorithmus verändern, damit tatsächlich eine Tiefensuche dabei herauskommt? Die nun verwendete Datenstruktur heißt *Stack* oder *Stapel* und funktioniert tatsächlich wie ein Papierstapel. Das, was zuletzt obendrauf gelegt wurde, wird als erstes wieder heruntergenommen (siehe Abbildung 16).

- Suchen Sie eine gemeinsame Formulierung für die Breitensuche und die Tiefensuche, die sich nur darin unterscheidet, dass eine Queue bzw. ein Stack verwendet wird.

Diese Aufgabe stellt sich als unerwartet schwierig dar. Sie eignet sich für den Unterricht in der Oberstufe, falls man sich nicht scheut, auch im Mathematikunterricht Elemente der theoretischen Informatik zu behandeln. Geben Sie Ihren Schülerinnen und Schülern reichlich Zeit dafür. Mit dieser Aufgabe kann das algorithmische Denken ganz intensiv eingefordert und gefördert werden. Zur Überprüfung der gefundenen Formulierungen werden die in Kapitel 1, Seiten 24 ff., aufgezeigten Methoden auch hier nochmals empfohlen. Eine mögliche Lösung sieht so aus:

#### ■ Aufspannender-Baum-Algorithmus ■

Eingabe: ein Graph

Ausgabe: ein aufspannender Baum einer Zusammenhangskomponente

Benötigt wird: eine Datenstruktur  $D$  und eine Vorgängerliste

1. Wähle einen Startknoten. Er ist nun der aktive Knoten und wird als besucht markiert.
2. Schreibe alle noch unbesuchten Nachbarn des aktiven Knotens in der Reihenfolge ihres Auffindens in  $D$ . Der aktive Knoten wird als Vorgänger dieser Knoten notiert.
3. Falls  $D$  leer ist: Stopp, anderenfalls gehe zu 4.
4. Der nächste Knoten mitsamt Vorgänger wird von der Datenstruktur  $D$  ausgegeben.
  - Falls er bereits als besucht markiert ist: Lösche ihn aus  $D$ . Wiederhole 3.
  - Anderenfalls wird dieser Knoten zum aktiven Knoten und als besucht

markiert. Sein Vorgänger wird verbindlich festgeschrieben und er wird aus  $D$  gelöscht.

5. Gehe zu 2.

Nun können Sie nach Belieben, bzw. je nach Anforderungen des zu lösenden Problems, für die Datenstruktur  $D$  eine Queue oder einen Stack verwenden. Was immer noch bis zu einem gewissen Grad offen bleibt, ist, in welcher Reihenfolge die Nachbarn des aktiven Knotens in die Datenstruktur aufgenommen werden. Meist wird die für den Graphen verwendete Datenstruktur eine Reihenfolge naheliegen.

Spielen Sie diesen Aufspannender-Baum-Algorithmus im Unterricht als Rollenspiel (vgl. Kapitel 1, S. 24) durch. Lassen Sie Ihre Schülerinnen und Schüler erleben, wie sich Stack und Queue verschieden auswirken. Das Backtracking der Tiefensuche wird hier besonders schön sichtbar: Es ist das Rückwärtsgehen im Stack durch bereits besuchte Knoten, die aus dem Stack gestrichen werden, ohne dass eine weitere Aktion folgt (Schritt 4, erster Fall). Wollen Sie die Mechanik von Stack oder Queue noch plastischer darstellen, so verwenden Sie dafür tatsächlich einen Papierstapel bzw. eine Papprolle, in die beschriftete Tischtennisbälle geschoben werden. Auch hier geht es wieder nicht um konkrete Implementierungen, sondern darum, Vorgehensweisen ganz präzise zu analysieren.

Noch eine Beobachtung überrascht. Die Endprodukte von Breiten- und Tiefensuche sind qualitativ verschieden. Der Breitensuche-Baum liefert kürzeste Wege bezüglich der Anzahl der Kanten vom Startknoten zu allen anderen Knoten. Die Tiefensuche erzeugt einen aufspannenden Baum, weiter nichts. Über Distanzen zum Startknoten kann man keine allgemeingültigen Aussagen machen. Auch wenn es manchmal so wirken kann, konstruiert die Tiefensuche keine längsten Wege. So kann man hier im direkten Vergleich beider Algorithmen feststellen, dass die Breitensuche ein Ergebnis liefert, das mehr Informationen beinhaltet als das der Tiefensuche. Die Erkenntnis, wie eng verwandt und dennoch verschieden Breiten- und Tiefensuche sind, kann zu weiterem detailliertem Nachdenken über Lösungsstrategien anregen.

## 5 Die Algorithmen von Kruskal und Prim

### *Kosten kommen ins Spiel*

Am Anfang dieses Kapitels haben wir überlegt, wie ein möglichst günstiges Teilnetz eines Telefonnetzes aussehen muss. Sie haben sich sicherlich schon gefragt, warum wir uns keine Gedanken über die Kosten von Leitungsabschnitten gemacht haben. Das lag daran, dass in der ersten Hälfte des Kapitels über das grundlegende Konzept der Bäume und aufspannenden Bäume nachgedacht werden sollte.



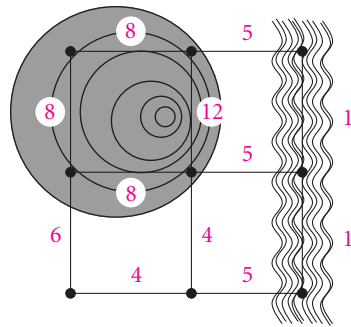


Abbildung 17. Unterschiedliche Umgebungen verursachen unterschiedliche Kosten für die Verlegung und Wartung von Kabeln

Darum wurde das Modell für die reale Situation zunächst so einfach wie möglich gehalten.

Nun aber wird es Zeit, über die Kosten nachzudenken. Dass die Länge der Leitungsabschnitte nicht unbedingt proportional zu den Kosten sein muss, zeigt das schematisierte Beispiel in Abbildung 17. Sowohl die Verlegung der Kabel als auch die Wartung ist billiger, wenn das Kabel in einem Flussbett verlegt werden kann, als wenn es durch einen Berg verläuft. Mit dieser Überlegung kann man Kantengewichte (siehe auch Kapitel 1, S. 26) motivieren. Im »echten Leben« bestimmt eine sehr große Zahl von Faktoren die Kosten, insbesondere auch die Kapazität der Leitungen. Für unsere Fragestellung liegt es nun nahe, nach aufspannenden Bäumen mit minimalen Kosten zu suchen.

### Definition

Ein *minimaler aufspannender Baum* in einem gewichteten Graphen ist ein aufspannender Baum mit einer minimalen Summe von Kantengewichten.

### Zwei »gierige« Vorgehensweisen

Wie aber geht man mit gewichteten Kanten um? Tiefensuche oder Breitensuche erzeugen in diesem Fall keine optimalen aufspannenden Bäume.

- Lösen Sie das Telefonleitungsproblem vom Anfang des Kapitels, indem Sie nun Kosten für die einzelnen Leitungsabschnitte berücksichtigen.

Für diese Aufgabe gibt es zwei recht naheliegende Lösungsstrategien, die von Schülerinnen und Schülern meist schnell gefunden werden. Sämtliche notwendigen Vorüberlegungen sind in diesem Kapitel bereits gemacht worden, daher verzichten wir an dieser Stelle auf eine ausführliche Herleitung.

Der erste Algorithmus »malt« immer die nächstbilligste Kante an, sofern dabei keine Kreise aus markierten Kanten entstehen. Im Verlauf des Algorithmus kön-

nen dabei unzusammenhängende Teilgraphen entstehen. Hier bietet es sich wieder an, Daumenkinos zu basteln, um den Ablauf des Algorithmus zu visualisieren.

#### ■ *Der Algorithmus von Kruskal* ■

Eingabe: ein gewichteter zusammenhängender Graph mit  $n$  Knoten

Ausgabe: ein minimaler aufspannender Baum des Graphen

1. Wähle eine »billigste« noch unmarkierte Kante.
2. Markiere sie, falls sie keinen Kreis mit anderen markierten Kanten schließt.
3. Stopp, falls  $n - 1$  Kanten markiert sind, anderenfalls gehe zu 1.

Der andere Algorithmus lässt einen Baum möglichst kostengünstig wachsen. Das »Produkt« des Algorithmus ist zu jedem Zeitpunkt zusammenhängend.

#### ■ *Der Algorithmus von Prim* ■

Eingabe: ein gewichteter zusammenhängender Graph

Ausgabe: ein minimaler aufspannender Baum des Graphen

1. Wähle einen Startknoten.
2. Markiere die »billigste« vom bereits konstruierten Baum ausgehende Kante, falls sie keinen Kreis schließt.
3. Stopp, falls  $n - 1$  Kanten markiert sind, anderenfalls gehe zu 2.

Es ist nicht ganz einfach, diese Algorithmen in so wenigen Schritten darzustellen. Ein erster Formulierungsversuch wird meist sehr viel mehr Schritte brauchen als die Endfassung. Der Trick bei der obigen Formulierung des Algorithmus von Prim besteht darin, den Startknoten bereits als Baum zu betrachten. Somit muss man nicht mehr zwischen der Wahl der ersten Kante und der Wahl der weiteren Kanten unterscheiden.

Beide Algorithmen gehen »gierig« vor: Es wird stets eine billigste zulässige Kante gewählt. So wird bei jedem Schritt lokal, quasi mit Tunnelblick optimiert, in der Hoffnung, dass damit auch die Gesamtlösung optimal ausfällt. Nach dem englischen Wort für gierig werden solche Vorgehensweisen *Greedy-Strategien* genannt. Ob dieses lokale optimale Vorgehen tatsächlich ein globales Optimum liefert, muss nachgewiesen werden. Hier ist das der Fall, was an der besonderen Struktur des Problems liegt (die Menge aller Bäume eines Graphen bildet ein so genanntes *Matroid*). Korrektheitsbeweise für die Algorithmen von Kruskal und Prim finden Sie am Ende dieses Kapitels. Für andere Probleme, z. B. das Travelling-Salesman-Problem (Kapitel 4) oder Graphenfärben (Kapitel 5), liefern Greedy-Strategien gerade einmal Näherungslösungen, die auch beliebig weit vom Optimum entfernt sein können.

## 6 Steinerbäume

Bäume tauchen vielerorts auf, insbesondere bei Versorgungsnetzen für z. B. Strom oder Wasser oder bei der Planung von Leiterbahnen für Chips. In manchen Fällen steht die geometrische Lage der Knoten fest, aber es können zusätzliche Knoten frei gewählt werden.

Ein Beispiel (aus [24]) für solch eine Situation ist die Frage nach kürzesten Verbindungswegen zwischen verschiedenen Angellöchern auf einem zugefrorenen See z. B. im Norden Kanadas. Es gibt keinerlei Vorgaben für den Verlauf der Wege, außer dass sie insgesamt möglichst kurz sein sollen, und daher nur geradlinig verlaufen können.

- Zeichnen Sie einige Knoten auf ein Blatt Papier und suchen Sie nach Möglichkeiten, diese Knoten durch einen möglichst kurzen Baum miteinander zu verbinden. »Hilfsknoten« sind erlaubt.
- Experimentieren Sie mit geometrischen Grundformen wie gleichseitigem Dreieck, Quadrat, Rechteck ....

Durch Ausprobieren sieht man schnell, dass die Gesamtlänge der Baumkanten durch das Hinzufügen von Zwischenknoten kleiner werden kann. Doch wo genau soll man den oder die zusätzlichen Knoten platzieren?

Die Frage nach dem minimalen Baum zwischen Knoten fester geometrischer Lage führt auf so genannte *Steinerbäume* (nach dem Schweizer Mathematiker Jacob Steiner (1796–1863)). Bei 4 Knoten, die als Rechteck angeordnet sind, werden beispielsweise zwei zusätzliche Knoten benötigt. (siehe Abbildung 18). Dass diese Fragestellung weitaus komplexer und schwieriger ist, als diejenigen, die diesem Kapitel zugrunde liegen, können Sie sich sicher leicht vorstellen. Das Problem besteht darin, zu entscheiden, wie viele zusätzliche Knoten benötigt werden, und wo genau sie liegen sollen.

Um ein Gefühl für die Formenvielfalt von Steinerbäumen zu entwickeln, können Sie mit Seifenblasen experimentieren (Idee aus [24]): Nehmen Sie zwei gleichgroße Plexiglasscheiben und befestigen Sie senkrecht dazwischen Stäbe, die die Knoten repräsentieren (vgl. Abbildung 19). Dann tauchen Sie das Ganze in Seifen-



Abbildung 18. Steinerbäume: Wo müssen die zusätzlichen (roten) Knoten liegen, damit die Gesamtlänge der Baumkanten möglichst klein wird?

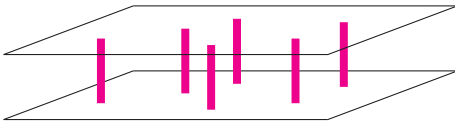


Abbildung 19. Bauen Sie sich solch ein Gerät und tauchen es in Seifenblasenflüssigkeit! Es entstehen Seifenblasen-Steinerbäume.

blasenflüssigkeit und nehmen es wieder heraus. Die Seifenhaut bildet einen Steinerbaum zwischen den Stäben! Allerdings ist es nicht unbedingt ein minimaler Steinerbaum. Die Seifenhäute bilden zwar Minimalflächen, doch muss dabei die Gesamtlänge des Baumes nicht unbedingt minimal sein. Tauchen Sie die gleiche Anordnung wiederholt ein, so werden Sie sehen, wie unterschiedlich die Steinerbäume aussehen können und Sie bekommen eine Ahnung von den unzähligen Möglichkeiten, die Steinerpunkte zu setzen.

## 7 Vertiefung: Korrektheitsbeweise für die Algorithmen von Kruskal und Prim

Gerade bei auf den ersten Blick richtig erscheinenden Algorithmen wie denen von Kruskal und Prim ist ein Korrektheitsbeweis unerlässlich. Die Konstruktionsmethoden sind so einleuchtend, dass es unvorstellbar ist, dass man damit auch einmal Schiffbruch erleiden könnte. So leicht, wie die Algorithmen erfunden sind, so schwer ist es, deren Korrektheit zu zeigen. Häufig wird die Korrektheit dieser Algorithmen mit Hilfe spezieller Strukturen, so genannten *Matroiden*, bewiesen, wozu man sich eine nicht ganz einfache Theorie aneignen muss. Es gibt aber auch einige Korrektheitsbeweise ohne Matroide, von denen jeder einzelne seine ganz eigenen Tücken hat. Jiří Matoušek macht in seinem schönen Lehrbuch ([15, S. 177]) sogar die Bemerkung, dass einem bei diesen Beweisen besonders leicht Fehler unterlaufen.

Die meisten Beweise arbeiten mit dem Austausch von einzelnen Baumkanten. Die grundsätzlichen Überlegungen dazu sind hier kurz skizziert:

**Kantentausch für Kreise:** Man kann aus einem aufspannenden Baum einen anderen aufspannenden Baum desselben Graphen machen, indem man eine neue Kante hinzufügt, die – wie Sie ja wissen – immer einen Kreis schließt, und anschließend eine andere Kante aus diesem Kreis herausnimmt.

**Kantentausch im Schnitt:** Jede der Kanten eines aufspannenden Baumes induziert einen so genannten *Schnitt* in dem Graphen: Stellen Sie sich vor, Sie nehmen die gewählte Kante aus dem Graphen heraus. Der aufspannende Baum zerfällt in zwei Teile. Die Knotenmenge des Graphen wird dadurch in zwei Teile geteilt. Nehmen Sie eine der beiden Mengen und färben alle Kanten, die einen Knoten in dieser Menge haben und einen Knoten außerhalb. Diese Kantenmenge ist der zu der

Knotenmenge gehörende Schnitt. Die ursprüngliche Kante kann durch eine andere Kante aus dem Schnitt ersetzt werden. Es entsteht wieder ein aufspannender Baum.

*Korrektheit des Algorithmus von Kruskal.* Ein Beweis mittels vollständiger Induktion. Es ist lediglich zu zeigen, dass der Baum  $B$ , den der Algorithmus produziert, minimal ist. Denn dass überhaupt ein Baum entsteht, sieht man bereits an der Konstruktionsvorschrift ( $n - 1$  Kanten kreisfrei in einem Graphen mit  $n$  Knoten unterzubringen geht nur als aufspannender Baum).

Die Grundidee ist, dass man eine Menge von Kanten, wie sie nach einigen Iterationen des Algorithmus vorliegt, zu einem minimalen aufspannenden Baum ergänzen kann. Das muss nicht unbedingt derselbe Baum sein wie der vom Algorithmus konstruierte. Wenn man das für alle Zwischenprodukte des Algorithmus zeigen kann (einschließlich des Endprodukts, das dann durch null Kanten zum minimalen aufspannenden Baum ergänzt wird), dann ist die Korrektheit bewiesen.

Die erste Kante, die vom Algorithmus gewählt wird, ist eine mit minimalem Gewicht. Sie kann auf mindestens eine Art zu einem minimalen aufspannenden Baum ergänzt werden. Das ist der Induktionsanfang. Nun wollen wir zeigen, dass auch die ersten  $2, 3, \dots, n - 1$  Kanten, die der Algorithmus ausgewählt hat, zu einem minimalen aufspannenden Baum ergänzt werden können. Wir nehmen nun an, dass es auch für die ersten  $k$  Kanten geht (das ist die Induktionsvoraussetzung), und zeigen, dass es dann auch für  $k + 1$  Kanten geht. Wenn das geschafft ist, können wir von einer Kante schrittweise auf  $n - 1$  Kanten schließen.

Wir gehen von den ersten  $k$  Kanten  $\{e_1, e_2, \dots, e_k\}$  aus, die der Algorithmus gewählt hat. Nach Induktionsvoraussetzung gibt es einen minimalen aufspannenden Baum  $B_k$ , der diese Kanten enthält. Nun nehmen wir die  $k + 1$ . Kante  $e_{k+1}$  hinzu. Entweder ist sie sowieso in  $B_k$  enthalten, dann ist der Beweis fertig. Oder sie ist nicht in  $B_k$  enthalten, dann konstruieren wir jetzt einen minimalen aufspannenden Baum  $B_{k+1}$ , der die  $k + 1$ . Kante enthält.

Nun folgt die oben beschriebene Technik: Durch Hinzufügen der Kante  $e_{k+1}$  zu  $B_k$  entsteht ein Kreis. Der (eindeutige!) Kreis, der durch Hinzufügen von  $e_{k+1}$  entstanden ist, muss mindestens eine Kante enthalten, die nicht in dem vom Algorithmus erzeugten aufspannenden Baum  $B$  vorkommt. Eine dieser Kanten wird entfernt, es entsteht wieder ein Baum. Sie kann kein kleineres Gewicht als  $e_{k+1}$  haben, sonst wäre sie vom Algorithmus vor  $e_{k+1}$  ausgewählt worden, also hat sie ein mindestens ebenso hohes Gewicht wie  $e_{k+1}$ . Daraus folgt, dass der neu entstandene Baum  $B_{k+1}$  kein höheres Gesamtgewicht haben kann als  $B_k$ .  $B_k$  war bereits schon minimal, also muss auch  $B_{k+1}$  minimal sein. Damit haben wir das Gewünschte konstruiert: einen minimalen aufspannenden Baum, der die Kanten  $\{e_1, e_2, \dots, e_{k+1}\}$  enthält. Der Schritt von  $k$  zu  $k + 1$  ist getan, und somit ist

auch für  $n - 1$  Kanten bewiesen, dass diese sich (durch Zufügen von Null Kanten) zu einem minimalen aufspannenden Baum ergänzen lassen. Der Algorithmus von Kruskal arbeitet korrekt.

*Korrektheit des Algorithmus von Prim.* Dieser (Widerspruchs-)Beweis arbeitet mit der Methode des »kleinsten Verbrechers« die auch in anderen Beweisen dieses Buches verwendet wird. Man stellt sich vor, dass es einen Graphen gibt, für den der Algorithmus einen nicht-minimalen aufspannenden Baum konstruiert. Das heißt, dass der Algorithmus falsche, also zu teure Kanten, wählt. Wir sehen uns den Zeitpunkt an, zu dem das erste Mal eine falsche Kante, genannt »kleinster Verbrecher«, gewählt wird. Der zuvor konstruierte Teilbaum  $T$  lässt sich also noch zu einem minimalen aufspannenden Baum  $B_T$  ergänzen, da bis dahin alles richtig gelaufen ist,  $T + \text{»kleinster Verbrecher«}$ , aber nicht mehr.

Jetzt sieht man sich den *Schnitt* des Teilbaumes  $T$  an (siehe Abbildung 20). Das sind genau die Kanten, die im Ablauf des Algorithmus als nächstes in Erwägung gezogen werden. »Kleinsten Verbrecher« ist eine Schnittkante, denn es ist die Kante, die zu  $T$  hinzugewählt wurde.  $B_T$ , der minimale aufspannende Baum, der  $T$  enthält, muss auch eine Kante aus dem Schnitt enthalten, sonst wäre  $B_T$  nicht zusammenhängend. Diese Kante  $e_{B_T}$  kann nicht billiger sein als »kleinster Verbrecher«, denn der Algorithmus hat »kleinster Verbrecher« als eine billigste Kante aus allen Schnittkanten gewählt. Die Kante  $e_{B_T}$  kann aber auch nicht teurer sein als »kleinster Verbrecher«, denn sonst könnte man durch Austausch beider Kanten einen aufspannenden Baum erzeugen, der billiger als der minimale aufspannende Baum  $B_T$  ist. So einen Baum kann es natürlich nicht geben.

Also ist »kleinster Verbrecher« gleich teuer wie  $e_{B_T}$ . Das bedeutet, dass man in  $B_T$  die Kante  $e_{B_T}$  durch »kleinster Verbrecher« ersetzen kann und wieder einen minimalen aufspannenden Baum erhält. Das bedeutet aber, dass die Wahl von »kleinster Verbrecher« gar kein Fehler war, denn  $T + \text{»kleinster Verbrecher«}$  lässt sich doch, wie eben gesehen, zu einem minimalen aufspannenden Baum ergänzen. Die Annahme war also falsch. Damit ist die Korrektheit des Algorithmus bewiesen.

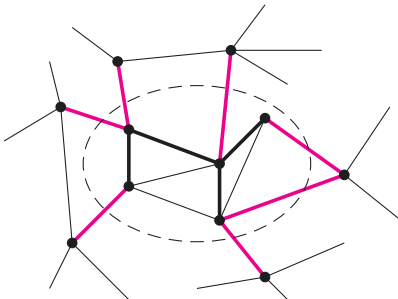


Abbildung 20. Der Schnitt (rote Kanten) für den bereits konstruierten Teilbaum (dicke Kanten) besteht aus allen Kanten, die einen Knoten im Teilbaum und den anderen Knoten außerhalb des Teilbaumes haben.

Diskrete Mathematik erleben

Anwendungsbasierte und verstehensorientierte  
Zugänge

Hußmann, S.; Lutz-Westphal, B. (Hrsg.)

2015, XVII, 347 S. 204 Abb., 100 Abb. in Farbe.,

Softcover

ISBN: 978-3-658-06992-6