

Graphentheoretische Konzepte und Algorithmen

**Sven Oliver Krumke, Hartmut Noltemeier,
Stefan Schwarz, Hans-Christoph Wirth**
Letzte Änderung: 13. Januar 2000, 9:26

Vorlesung Graphentheoretische Konzepte und
Algorithmen
Lehrstuhl für Informatik I
Bayerischen Julius-Maximilians-Universität Würzburg
S. O. Krumke, H. Noltemeier, S. Schwarz, H. C. Wirth

Inhaltsverzeichnis

1	Einleitung	1
1.1	Das Königsberger Brückenproblem	1
1.2	Das Haus von Nikolaus	2
1.3	Labyrinth	3
1.4	Informationen über WWW	3
2	Grundbegriffe	5
2.1	Gerichtete Graphen	5
2.2	Teilgraphen und Obergraphen	7
2.3	Ungerichtete Graphen	7
2.4	Graphen mit einer speziellen Struktur	8
2.5	Graphentheoretische Algorithmen	8
	Übungsaufgaben	12
3	Wege, Kreise und Zusammenhang	13
3.1	Wege	13
3.2	Zusammenhang	15
3.3	Der Satz von Euler	16
3.4	Hamiltonsche Kreise	19
	Übungsaufgaben	19
4	Bestimmung von Zusammenhangskomponenten	25
4.1	Transitive Hülle	25
4.2	Der Tripelalgorithmus	26
4.3	Der Reduzierte Graph	29
4.4	Irreduzible Kerne	31
	Übungsaufgaben	34
5	Bäume, Wälder und Matroide	37
5.1	Bäume und Wälder	37
5.2	Minimale spannende Bäume	41
5.3	Steinerbäume und andere Dilemmas	51
5.4	Spannende Wurzelbäume in gerichteten Graphen	51
	Übungsaufgaben	54
6	Suchstrategien	59
6.1	Untersuchung von Graphen mit Tiefensuche	59
6.2	Anwendungen von DFS	64
6.3	Tiefensuche für ungerichtete Graphen	67
6.4	Breitensuche	69
	Übungsaufgaben	69

7	Bestimmung kürzester Wege	71
7.1	Motivation	71
7.2	Kürzeste Wege	72
7.3	Kürzeste-Wege-Probleme	72
7.4	Kürzeste-Wege-Bäume	73
7.5	Der Algorithmus von Dijkstra	76
7.6	Der Algorithmus von Bellman und Ford	77
7.7	Die Bellmanschen Gleichungen und kreisfreie Graphen	79
7.8	Längste Wege	80
	Übungsaufgaben	81
8	Strömungen und Flüsse	83
8.1	Strömungen und Schnitte	83
8.2	Zulässige und maximale Strömungen	84
8.3	Das Max-Flow-Min-Cut-Theorem	87
8.4	Der Algorithmus von Ford und Fulkerson	89
8.5	Kombinatorische Anwendungen	97
8.6	Kostenminimale Strömungen	100
	Übungsaufgaben	103
9	Planare Graphen	105
9.1	Einbettungen	105
9.2	Die Eulersche Polyederformel	107
9.3	Triangulationen	107
9.4	Grade in planaren Graphen	109
9.5	Kriterien für Planarität	111
9.6	Duale Graphen	113
10	Homomorphismen	117
10.1	Tripeldarstellung von Graphen	117
10.2	Homomorphismen	117
10.3	Das Graphenisomorphieproblem	119
10.4	Automorphismen	120
A	Lösungen zu den Übungsaufgaben	123
B	Notationen	151

Vorwort

Das vorliegende Material wurde für einen Selbststudienkurs „Graphentheoretische Konzepte und Algorithmen“ im Rahmen des MEILE-Programmes für die Virtuelle Hochschule Bayern zusammengestellt.

Das zugrundeliegende Skript basiert auf Mitschriften vorausgegangener regelmäßiger Vorlesungen von Prof. Dr. H. Noltemeier und auf dem Skript zur Vorlesung „Graphentheoretische Konzepte und Algorithmen“, die von Dr. S. O. Krumke im Sommersemester 1997 an der Bayerischen Julius-Maximilians-Universität Würzburg gehalten wurde.

Das Skript wurde im Sommersemester 1999 ergänzt durch Inhalte aus der Vorlesung „Graphentheoretische Methoden und Algorithmen“, gehalten von Prof. Dr. H. Noltemeier, durch Übungen zur Vorlesung von H.-C. Wirth und durch Java-Applets von S. Schwarz.

Würzburg, 13. Januar 2000

Sven Oliver Krumke,
Hartmut Noltemeier, Stefan Schwarz, Hans-Christoph Wirth

Kapitel 1

Einleitung

1.1 Das Königsberger Brückenproblem

Die Geschichte der Graphentheorie beginnt mit einer Arbeit von Euler aus dem Jahr 1736, in der er das Königsberger Brückenproblem löste.

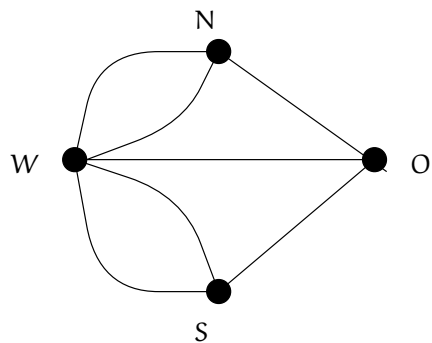
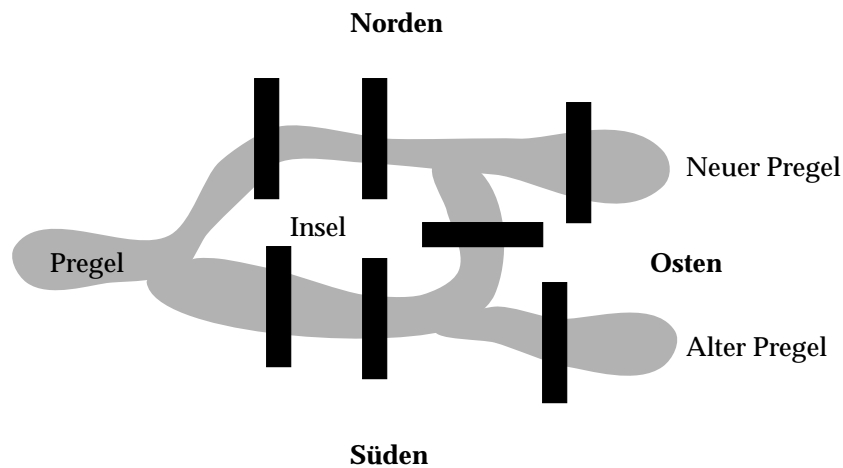


Abbildung 1.1:
Skizzierter Stadtplan von
Königsberg und zugehöriger
Graph.

Dieses Problem bestand darin zu entscheiden, ob es einen Rundweg durch Königsberg gibt, der jede der sieben Brücken *genau einmal* über-

quert. Abbildung 1.1 zeigt einen Stadtplan von Königsberg mit der Lage der Brücken.

Euler erkannte, daß man die Situation von der genauen Form der Ufer und Brücken abstrahieren kann. Man stellt die einzelnen Ufer durch Punkte (Ecken) dar, die durch Linien (Kanten) verbunden sind, die die einzelnen Brücken repräsentieren. Dabei erhält man den ebenfalls in Abbildung 1.1 gezeichneten *Graphen*. Das Königsberger Brückenproblem reduziert sich nun darauf zu entscheiden, ob es in dem entstandenen Graphen einen Rundweg gibt, der jede Linie (Kante) genau einmal durchläuft. Eulers Antwort auf das Brückenproblem war dann, daß es keinen Rundweg durch den Graphen (oder durch Königsberg) mit den gewünschten Eigenschaften gibt.

Bevor wir das Ergebnis Eulers einfach hinnehmen, sollten wir kurz darüber nachdenken, warum der obige Graph keinen Eulerschen Kreis enthält. Dazu starten wir eine gedachte Rundtour im Punkt O des Graphen, der den Osten der Stadt repräsentiert. Wir verlassen O über eine der drei einmündenden Kanten. Wenn wir das erste mal wieder nach O zurückkehren (und dies müssen wir, da wir alle Brücken genau einmal durchlaufen wollen), haben wir zu diesem Zeitpunkt zwei der drei in O einmündenden Brücken überquert. Wenn wir nun O wieder verlassen, besteht keine Möglichkeit mehr, zu O zurückzukehren, ohne eine Brücke mindestens zweimal durchlaufen zu haben.

Das Problem bei der Erstellung eines Rundweges, der alle Brücken genau einmal durchläuft, besteht offenbar darin, daß im Punkt O (und in allen anderen Ecken des Graphen) eine ungerade Anzahl von Kanten mündet. Man zeigt nun leicht, daß in einem Graphen höchstens dann ein Eulerscher Kreis, d.h. ein Kreis, der jede Kante eines Graphen genau einmal durchläuft, existiert, wenn in jeder Ecke eine gerade Anzahl von (ungerichteten) Kanten mündet.

Euler war darüberhinaus in der Lage zu zeigen, daß die obige Bedingung nicht nur notwendig sondern auch hinreichend für die Existenz eines Eulerschen Kreises ist. Den berühmten Satz von Euler werden wir in Kapitel 3 vorstellen und beweisen.

1.2 Das Haus von Nikolaus

Eine ähnliche Fragestellung wie beim Königsberger Brückenproblem erhält man beim „Haus des Nikolaus“, welches in Abbildung 1.2 dargestellt ist. Das Problem besteht darin, zu entscheiden, ob man das Bild aus Abbildung 1.2 zeichnen kann, ohne den Stift abzusetzen.

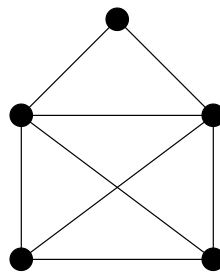


Abbildung 1.2:
Das Haus vom Nikolaus.

Aus graphentheoretischer Sicht fragt man hier nach einem Weg, der

jede Kante genau einmal durchläuft. Im Gegensatz zum Königsberger Brückenproblem muß dieser Weg aber kein Kreis sein, da wir mit dem Zeichnen an einer beliebigen Ecke anfangen und möglicherweise an einer anderen Ecke enden können. Wege in Graphen, die jede Kante genau einmal durchlaufen, nennt man wegen Ihrer engen Beziehung zu den Eulerschen Kreisen *Eulersche Wege*.

1.3 Labyrinth

Als abschließendes Beispiel betrachten wir das Problem, einen Weg durch ein Labyrinth von einem gegebenen Startpunkt zu einem Endpunkt zu finden.

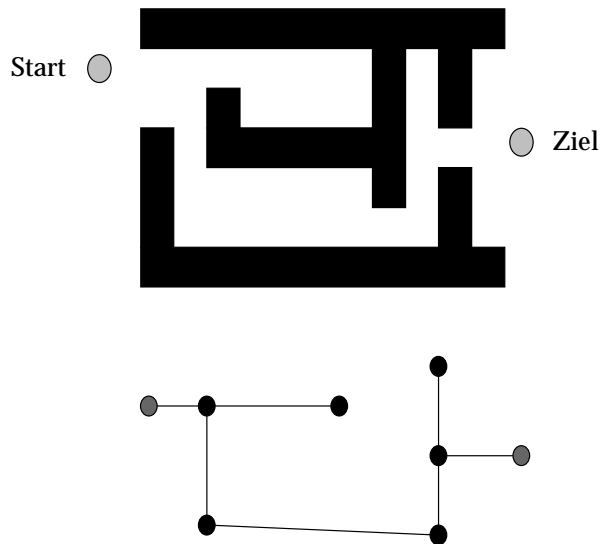


Abbildung 1.3:
Labyrinth und zugehöriger Graph.

Das Labyrinth aus Abbildung 1.3 wird in einen Graphen umgesetzt, indem man für Start, Ziel und jede Kreuzung Ecken im Graphen einführt und diese dann gemäß der Labyrinthstruktur verbindet. Man sucht dann im entstandenen Graphen einen Weg vom Start zum Ziel.

Fragen nach der Existenz von Wegen werden in den Kapiteln 4 und 6 behandelt. In Kapitel 7 wird die Problemstellung dahingehend erweitert, daß man nicht nach einem beliebigen Weg sondern nach einem kürzesten Weg fragt.

1.4 Informationen über WWW

Informationen sind unter folgender URL erhältlich:

<http://www-info1.informatik.uni-wuerzburg.de/>

Kapitel 2

Grundbegriffe

2.1 Gerichtete Graphen

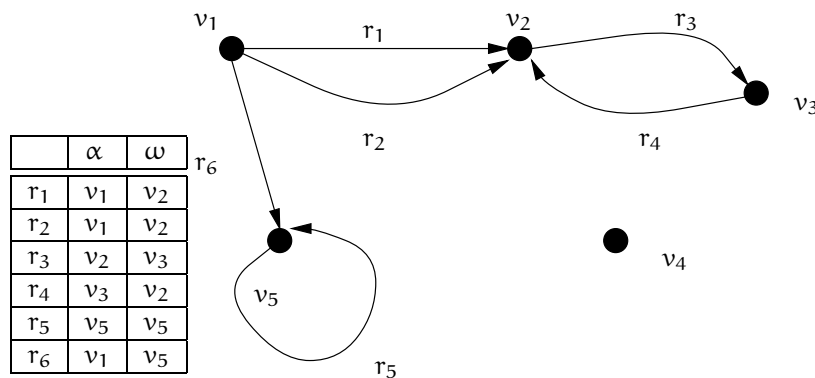
Definition 2.1 (Gerichteter Graph)

Ein **gerichteter Graph** (oder einfach **Graph**) ist ein Quadrupel $G = (V, R, \alpha, \omega)$ mit folgenden Eigenschaften:

1. V ist eine nicht leere Menge, die **Eckenmenge** des Graphen.
2. R ist eine Menge, die **Pfeilmenge** des Graphen.
3. Es gilt $V \cap R = \emptyset$.
4. $\alpha : R \rightarrow V$ und $\omega : R \rightarrow V$ sind Abbildungen ($\alpha(r)$: **Anfangsecke**, $\omega(r)$: **Endecke** des Pfeils r).

Der Graph G heißt **endlich**, wenn $|V \cup R| < +\infty$.

Beispiel 2.2 Gegeben sei der Graph $G = (V, R, \alpha, \omega)$ mit $V = \{v_1, \dots, v_5\}$ und $R = \{r_1, \dots, r_6\}$, sowie α und ω gemäß der folgenden Tabelle:



◇

Definition 2.3

Sei $G = (V, R, \alpha, \omega)$ ein Graph. Ein Pfeil $r \in R$ heißt **Schlinge**, wenn $\alpha(r) = \omega(r)$. Der Graph G heißt **schlingenfrei**, wenn er keine Schlingen enthält.

Zwei Pfeile r und r' heißen **parallel**, wenn $r \neq r'$ und $\alpha(r) = \alpha(r')$ sowie $\omega(r) = \omega(r')$.

Die Pfeile τ und τ' heißen **invers**, wenn $\alpha(\tau) = \omega(\tau')$ sowie $\omega(\tau) = \alpha(\tau')$.

Der Graph G heißt **einfach**, wenn er keine Schlingen und Parallelen enthält. In diesem Fall ist jeder Pfeil $\tau \in R$ eindeutig durch das Paar $(\alpha(\tau), \omega(\tau))$ charakterisiert. Man schreibt dann auch $G = (V, R)$ für den gerichteten Graphen G , wobei $R \subseteq V \times V$.

Definition 2.4

Sei $G = (V, R, \alpha, \omega)$ ein Graph. Zwei Ecken u und v heißen **adjazent** oder **benachbart**, wenn es einen Pfeil $\tau \in R$ gibt mit $\alpha(\tau) = u$ und $\omega(\tau) = v$ oder $\alpha(\tau) = v$ und $\omega(\tau) = u$.

Eine Ecke u und ein Pfeil τ heißen **inzident**, wenn $u \in \{\alpha(\tau), \omega(\tau)\}$. Zwei Pfeile τ und τ' heißen **inzident**, wenn es eine Ecke v gibt, die mit τ und τ' inzidiert.

Für eine Ecke $v \in V$ heißt

$$B^+(v) := \{\tau \in R : \alpha(\tau) = v\}$$

das von v
ausgehende
Pfeilbündel,

$$B^-(v) := \{\tau \in R : \omega(\tau) = v\}$$

das in v
mündende
Pfeilbündel,
die Nachfolger-
menge von

$$N(v) := \{u \in V : \exists \tau \in R \text{ mit } \alpha(\tau) = v, \omega(\tau) = u\}$$

v ,
die
Vorgängermenge

$$V(v) := \{u \in V : \exists \tau \in R \text{ mit } \alpha(\tau) = u, \omega(\tau) = v\}$$

von v ,

$$g^+(v) := |B^+(v)|$$

der Außengrad

$$g^-(v) := |B^-(v)|$$

von v ,

der Innengrad

$$g(v) := g^+(v) + g^-(v)$$

von v ,

der Grad von v .

Sei nun $|R| < +\infty$. Dann gilt

1. $\sum_{v \in V} g^+(v) = \sum_{v \in V} g^-(v) = |R|$
2. $\sum_{v \in V} g(v) = 2|R|$.

Lemma 2.5 Sei G ein endlicher Graph. Die Anzahl der Ecken in G mit ungeradem Grad ist gerade.

Beweis: Sei $U \subseteq V$ die Menge der Ecken in G mit ungeradem Grad. Dann gilt

$$2|R| = \sum_{v \in V} g(v) = \sum_{v \in V \setminus U} g(v) + \sum_{v \in U} g(v),$$

also

$$\sum_{v \in U} \overbrace{g(v)}^{\text{ungerade}} = 2|R| - \underbrace{\sum_{v \in V \setminus U} \overbrace{g(v)}^{\text{gerade}}}_{\text{gerade}} = 2M.$$

Für eine Ecke $v \in U$ ist ihr Grad $g(v) = 2k_v + 1$ ungerade ($k_v \in \mathbb{N}$ geeignet). Daher ergibt sich aus der letzten Gleichung:

$$2M = \sum_{v \in U} (2k_v + 1) = 2 \sum_{v \in U} k_v + |U|.$$

Somit ist $|U| = 2M - 2 \sum_{v \in U} k_v$ gerade. \square

2.2 Teilgraphen und Obergraphen

Definition 2.6 (Teilgraph, Obergraph)

Ein Graph $G' = (V', R', \alpha', \omega')$ heißt **Teilgraph** von $G = (V, R, \alpha, \omega)$ (i.Z. $G' \sqsubseteq G$), wenn

1. $V' \subseteq V$ und $R' \subseteq R$ sowie
2. $\alpha|_{R'} = \alpha'$ sowie $\omega|_{R'} = \omega'$.

G heißt dann **Obergraph** von G' ,

Die Relation „ \sqsubseteq “ ist

1. reflexiv ($G \sqsubseteq G$),
2. antisymmetrisch ($G \sqsubseteq G' \wedge G' \sqsubseteq G \Rightarrow G = G'$)
3. und transitiv ($G \sqsubseteq G' \wedge G' \sqsubseteq G'' \Rightarrow G \sqsubseteq G''$).

Definition 2.7 (Partialgraph, Subgraph)

Sei $R' \subseteq R$. Der **durch R' induzierte Partialgraph** $G_{R'}$ ist definiert durch $G_{R'} := (V, R', \alpha|_{R'}, \omega|_{R'})$.

Für eine Menge $V' \subseteq V$ definiert man den **durch V' induzierten Subgraphen** $G[V']$ durch $G[V'] := (V', R_{V'}, \alpha|_{R_{V'}}, \omega|_{R_{V'}})$ mit $R_{V'} := \{r \in R : \alpha(r) \in V' \wedge \omega(r) \in V'\}$.

Ein Graph H heißt **Partialgraph (Subgraph)** von G , wenn es eine Teilmenge $R' \subseteq R$ (Teilmenge $V' \subseteq V$) gibt, so daß $H = G_{R'}$ ($H = G[V']$) gilt. Ein Partial- oder Subgraph heißt **echt**, wenn die entsprechenden Teilmengen echte Teilmengen sind.

2.3 Ungerichtete Graphen

Definition 2.8 (Ungerichteter Graph)

Ein **ungerichteter Graph** ist ein Tripel $G = (V, E, \gamma)$ aus einer nichtleeren Menge V , einer Menge E und einer Abbildung $\gamma : E \rightarrow \{X : X \subseteq V \text{ mit } 1 \leq |X| \leq 2\}$.

Die Elemente von V heißen wieder **Ecken** von G , die Elemente von E heißen **Kanten**. Für eine Kante e heißen die Elemente von $\gamma(e)$ die **Endpunkte** von e .

Begriffe wie *Inzidenz*, *Adjazenz*, *Grad* etc. sind analog zu den gerichteten Graphen erklärt. Begriffe wie *Matching*, *Clique*, *Komplementgraph* etc. werden definiert, indem man durch Ersetzen jeder Kante durch ein Paar von inversen Pfeilen den ungerichteten Graphen zu einem (symmetrischen) gerichteten Graphen macht.

Der ungerichtete Graph G heißt *einfach*, wenn er keine Schlingen und Parallelen enthält. In diesem Fall kann man jede Kante $e \in E$ als

eine zweielementige Menge $e = \{u, v\}$ auffassen. Man schreibt dann auch $G = (V, E)$ für den ungerichteten Graphen G .

Bemerkung 2.9 Oftmals werden wir bei einfachen ungerichteten Graphen in leichtem Mißbrauch der Notation auch (u, v) anstelle von $\{u, v\}$ schreiben.

2.4 Graphen mit einer speziellen Struktur

In diesem Abschnitt werden Begriffe für Graphen eingeführt, die eine spezielle Struktur aufweisen. Die Definitionen erfolgen zunächst für gerichtete Graphen, lassen sich jedoch in naheliegender Weise über symmetrische Graphen auch auf ungerichtete Graphen übertragen.

Definition 2.10 (Matching)

Sei $G = (V, R)$ ein einfacher Graph. Ein **Matching** ist eine Teilmenge $M \subseteq R$ der Pfeile, so daß keine zwei Kanten aus M inzidieren.

Definition 2.11 (Maximales Matching)

Sei $G = (V, R)$ ein einfacher Graph. Ein Matching $M \subseteq R$ heißt **maximal**, falls für jede echte Obermenge $M \subset M' \subseteq R$ gilt: M' ist kein Matching in G .

Ein gerichteter Graph heißt **symmetrisch**, wenn er zu jedem Pfeil auch den inversen Pfeil enthält.

Im folgenden werden ungerichtete Graphen betrachtet.

Sei V eine beliebige Eckenmenge. Wir bezeichnen mit \hat{E}_V die Menge aller Kanten $\{\{v, w\} \mid v \neq w\}$ zwischen den Ecken in V .

Zu einer Eckenmenge V mit $n := |V|$ Elementen heißt der Graph $G = (V, \hat{E}_V)$ **vollständiger Graph der Ordnung n** . Er wird üblicherweise mit K_n bezeichnet. Es ist zu bemerken, daß K_n alle möglichen Kanten zwischen seinen n Ecken aufweist, aber schlingenfremd ist.

Definition 2.12 (Clique)

Sei $G = (V, E)$ ein einfacher Graph. Eine Teilmenge $C \subseteq V$ der Ecken heißt **Clique** in G , falls der induzierte Subgraph $G[C]$ vollständig ist.

Definition 2.13 (Komplementgraph)

Sei $G = (V, E)$ ein einfacher Graph. Der **Komplementgraph** zu G , bezeichnet mit $\bar{G} = (V, \bar{E})$, ist gegeben durch $\bar{E} := \hat{E}_V \setminus E$.

2.5 Graphentheoretische Algorithmen

Größenordnung von Funktionen

Sei M die Menge aller reellwertigen Funktionen $f: \mathbb{N} \rightarrow \mathbb{R}$ auf den natürlichen Zahlen. Jede Funktion $g \in M$ legt dann drei Klassen von Funktionen wie folgt fest:

- $\mathcal{O}(g) := \{f \in M \mid \exists c, n_0 \in \mathbb{N}: \forall n \geq n_0: f(n) \leq c \cdot g(n)\}$
- $\Omega(g) := \{f \in M \mid \exists c, n_0 \in \mathbb{N}: \forall n \geq n_0: f(n) \geq c \cdot g(n)\}$
- $\Theta(g) := \mathcal{O}(g) \cap \Omega(g)$

Man nennt eine Funktion f von *polynomieller Größenordnung* oder einfach *polynomiell*, wenn es ein Polynom g gibt, so daß $f \in \mathcal{O}(g)$ gilt.

Berechnungsmodell

Das bei der Laufzeit-Analyse verwendete Maschinenmodell ist das der *Unit-Cost RAM* (Random Access Machine). Diese Maschine besitzt abzählbar viele Register, die jeweils eine ganze Zahl beliebiger Größe aufnehmen können. Folgende Operationen sind jeweils in einem Takt der Maschine durchführbar: Ein- oder Ausgabe eines Registers, Übertragen eines Wertes zwischen Register und Hauptspeicher (evtl. mit indirekter Adressierung), Vergleich zweier Register und bedingte Verzweigung, sowie die arithmetischen Operationen Addition, Subtraktion, Multiplikation und Division [Pap94].

Dieses Modell erscheint für die Analyse der Laufzeit von Algorithmen besser geeignet als das Modell der Turing-Maschine, denn es kommt der Arbeitsweise realer Rechner näher. Allerdings ist zu beachten, daß die Unit-Cost RAM in einem Takt Zahlen beliebiger Größe verarbeiten kann. Durch geeignete Codierungen können damit ausgedehnte Berechnungen in einem einzigen Takt versteckt werden, ferner sind beliebig lange Daten in einem Takt zu bewegen. Damit ist das Modell echt mächtiger als das der Turing-Maschine. Es gibt keine Simulation einer Unit-Cost RAM auf einer (deterministischen) Turing-Maschine, die mit einem polynomiell beschränkten Mehraufwand auskommt.

Um diesem Problem der zu großen Zahlen vorzubeugen, kann man auf das Modell der *Log-Cost RAM* [Pap94] zurückgreifen. Bei einer solchen Maschine wird für jede Operation ein Zeitbedarf angesetzt, der proportional zum Logarithmus der Operanden, also proportional zur Codierungslänge ist. Eine andere Möglichkeit, das Problem auszuschließen, besteht darin, sicherzustellen, daß die auftretenden Zahlen nicht zu groß werden, also daß ihre Codierungslänge polynomiell beschränkt bleibt. Diese Voraussetzung ist bei den hier vorgestellten Algorithmen stets erfüllt. Der einfacheren Analyse wegen wird daher das Modell der Unit-Cost RAM zugrundegelegt.

Komplexitätsklassen

Die Komplexität eines Algorithmus ist ein Maß dafür, welchen Aufwand an Ressourcen ein Algorithmus bei seiner Ausführung braucht. Man unterscheidet die Zeitkomplexität, die die benötigte Laufzeit beschreibt, und die Raumkomplexität, die Aussagen über die Größe des benutzten Speichers macht. Raumkomplexitäten werden in diesem Skript nicht untersucht.

Die Komplexität wird in der Regel als Funktion über der Länge der Eingabe angegeben. Man nennt einen Algorithmus von der (*worst-case*-)Komplexität T , wenn die Laufzeit für alle Eingaben der Länge n durch die Funktion $T(n)$ nach oben beschränkt ist.

Die Komplexität von Algorithmen wird in dieser Arbeit als Funktion der Eckenzahl n und Kantenzahl m des eingegebenen Graphen angegeben. Diese Angabe ist detaillierter als die Abhängigkeit der Komplexität von der Eingabelänge: bei ecken- und kantenbewerteten Graphen ist deren Codierungslänge mindestens von der Größenordnung $\Omega(n + m)$.

Besonders wichtig sind in diesem Zusammenhang die Klassen P und NP. Die Klasse P ist die Menge aller Entscheidungsprobleme, die auf einer deterministischen Turing-Maschine in polynomieller Zeit gelöst werden können. Entsprechend ist die Klasse NP definiert als die Menge aller Probleme, deren Lösung auf einer nichtdeterministischen Turing-Maschine in Polynomialzeit möglich ist. Man vergleiche dazu etwa [GJ79].

Eine Transformation zwischen NP-Problemen heißt *polynomielle Reduktion*, wenn sie in polynomieller Zeit Instanzen zweier Probleme aus NP so ineinander überführt, daß die Antwort des Ausgangsproblems auf die Ausgangsinstanz dieselbe ist wie die Antwort des zweiten Problems auf die transformierte Instanz. Ein Problem heißt *NP-vollständig*, wenn jedes andere Problem aus NP polynomiell darauf reduziert werden kann. Der Reduktionsbegriff wird durch Einführen der *Turing-Reduktion* so erweitert, daß Reduktionen zwischen Optimierungsproblemen und Entscheidungsproblemen in NP erfaßt werden. Ein NP-Optimierungsproblem heißt dann *NP-hart*, wenn es von einem NP-vollständigen Entscheidungsproblem turing-reduziert werden kann.

Ein wesentliches Resultat der Komplexitätstheorie besagt, daß NP-harte Optimierungsprobleme nicht in polynomieller Zeit gelöst werden können, es sei denn, es gelte $P = NP$. Dies ist der Grund, warum bei der Untersuchung von NP-harten Optimierungsproblemen auf exakte Lösungen verzichtet wird und stattdessen Näherungen in Betracht gezogen werden.

Speicherung von Graphen

Sei $G = (V, R, \alpha, \omega)$ mit $V = \{v_1, \dots, v_n\}$ und $R = \{r_1, \dots, r_m\}$.

Adjazenzmatrixspeicherung

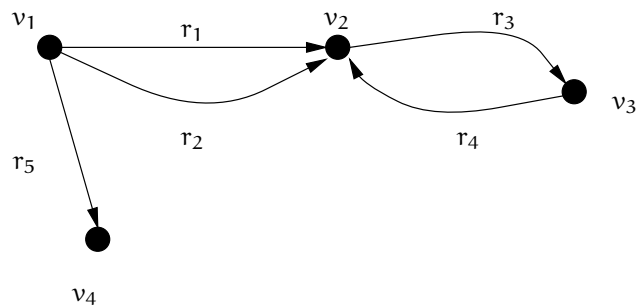


Abbildung 2.1:
Ein Graph.

Definition 2.14 (Adjazenzmatrix)

Die *Adjazenzmatrix* $A(G)$ ist eine $n \times n$ Matrix mit

$$a_{ij} = |\{r \in R : \alpha(r) = v_i \wedge \omega(r) = v_j\}|.$$

Beispiel 2.15 Für den Graphen G aus Bild 2.1 gilt

$$A(G) = \begin{pmatrix} 0 & 2 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Speicheraufwand für die Adjazenzmatrix: $\Theta(|V|^2)$.
Adjazenzmatrixspeicherung: $n, m, A(G)$

Inzidenzmatrixspeicherung

Definition 2.16 (Inzidenzmatrix)

Die **Inzidenzmatrix** $I(G)$ eines schlingenfreien Graphen G ist eine $n \times m$ Matrix mit

$$i_{kl} := \begin{cases} 1 & \text{falls } \alpha(r_l) = v_k \\ -1 & \text{falls } \omega(r_l) = v_k \\ 0 & \text{sonst} \end{cases}$$

Beispiel 2.17 Für den Graphen G aus Bild 2.1 gilt

$$I(G) = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 \\ -1 & -1 & 1 & -1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 \end{pmatrix}$$

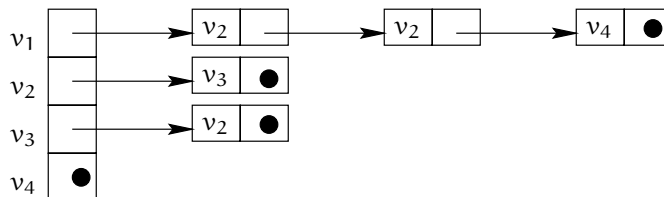
Speicheraufwand für die Inzidenzmatrix: $\Theta(|V| \cdot |R|)$.
Inzidenzmatrixspeicherung: $n, m, I(G)$

Adjazenzlistenspeicherung

Die *Adjazenzlistenrepräsentation* von G besteht dann aus den Zahlen n und m , sowie einem Array Adj von n Listen, für jede Ecke eine.

Die Liste $Adj[u]$ enthält (Pointer auf) alle Ecken v mit $v \in N_G(u)$. Falls es mehrere Pfeile r mit $\alpha(r) = u$ und $\omega(r) = v$ gibt, so wird v mehrmals aufgeführt.

Beispiel 2.18 Für den Graphen G aus Bild 2.1 sieht das Array Adj wie folgt aus:



Speicheraufwand für die Adjazenzlisten: $\Theta(|V| + |R|)$.

Dies ist ein Vorteil bei *dünnen* Graphen, d.h. falls $|R| \ll |V|^2$.

Eventuell speichert man bei der *Adjazenzlistenrepräsentation* neben der Liste $Adj[u]$ auch noch den *Außen-* und *Innengrad* der Ecke u .

Weitere Speicherungstechniken

Weitere Speicherungstechniken sind (siehe z.B. [Nol75]):

- Standardliste
- Eckenorientierte Liste (Variante der Adjazenzlistenspeicherung)

Übungsaufgaben

Aufgabe 2.1 – Speicherung von Graphen

In dieser Aufgabe sei $G = (V, R, \alpha, \omega)$ ein endlicher gerichteter Graph.

- (a) Der *inverse Graph* G^{-1} zu G ist definiert durch $G^{-1} = (V, R^{-1}, \alpha', \omega')$ mit $R^{-1} := \{r^{-1} : r \in R\}$ wobei $\alpha'(r^{-1}) = \omega(r)$ und $\omega'(r^{-1}) = \alpha(r)$. Der Graph G^{-1} entsteht aus G also dadurch, daß die Richtung aller Pfeile „umgedreht“ wird.

Geben Sie sowohl für die Adjazenzmatrixspeicherung als auch für die Adjazenzlistenspeicherung von G effiziente Algorithmen an, um G^{-1} aus G zu berechnen. Bestimmen Sie die Laufzeit Ihrer Algorithmen.

- (b) Geben Sie einen Algorithmus an, der in $\mathcal{O}(|V|)$ Zeit feststellt, ob ein einfacher Graph G in Adjazenzmatrixspeicherung eine Ecke v mit $g^+(v) = 0$ und $g^-(v) = |V| - 1$ enthält. (Hinweis: G kann höchstens *eine* solche Ecke v enthalten.)

Kapitel 3

Wege, Kreise und Zusammenhang

Im folgenden sei wieder $G = (V, R, \alpha, \omega)$ ein gerichteter Graph.

3.1 Wege

Definition 3.1 (Weg, Kreis)

Ein **Weg** w im Graphen G ist eine endliche Folge $w = (r_1, \dots, r_k)$ ($k \geq 1$) mit

1. $r_j \in R$ für $j = 1, \dots, k$ und
2. $\omega(r_j) = \alpha(r_{j+1})$ für $j = 1, \dots, k-1$.

Wir definieren die **Startecke** des Weges durch $\alpha(w) := \alpha(r_1)$ und die **Endecke** des Weges durch $\omega(w) := \omega(r_k)$. Die **Länge** $|w|$ des Weges w ist die Anzahl k der durchlaufenen Pfeile.

Ein Weg heißt ein **Kreis**, wenn $\alpha(w) = \omega(w)$.

Mit $s(w) := (\alpha(r_1), \dots, \alpha(r_k), \omega(r_k))$ bezeichnen wir die **Spur** des Weges w und sagen kurz, daß eine Ecke v von w **berührt** wird, wenn sie Element der Spur ist.

Ein Weg heißt **einfach**, wenn $r_j \neq r_l$ für $j \neq l$, d.h. wenn er keinen Pfeil mehr als einmal durchläuft.

Ein Weg heißt **elementar**, wenn $\alpha(r_j) \neq \alpha(r_l)$ und $\omega(r_j) \neq \omega(r_l)$ für $j \neq l$, d.h. wenn er — bis auf den Fall, daß Anfangs- und Endecke übereinstimmen — keine Ecke mehr als einmal berührt.

Definition 3.2 (Weg in einem ungerichteten Graphen)

Sei $G = (V, E, \gamma)$ ein ungerichteter Graph. Man nennt eine Folge $w = (e_1, \dots, e_k)$ von Kanten $e_i \in E$ einen **Weg** von v_0 nach v_k , wenn es $v_0, \dots, v_k \in V$ gibt, so daß $\gamma(e_i) = \{v_{i-1}, v_i\}$ für $i = 1, \dots, k$ gilt.

Wir definieren analog zum gerichteten Fall $\alpha(w) := v_0$ und $\omega(w) := v_k$.

Die Begriffe **elementar** und **einfach** sind analog zum gerichteten Fall definiert.

Bemerkung 3.3 Ist ein (gerichteter oder ungerichteter) Graph G parallelfrei, so ist jeder Weg $w = (r_1, \dots, r_k)$ in G eindeutig durch die Spur $s(w) = (v_1, \dots, v_{k+1})$ gekennzeichnet. In diesem Fall schreiben wir auch kurz $w = [v_1, \dots, v_{k+1}]$.

Lemma 3.4 *Jeder elementare Weg ist einfach.*

Beweis: Sei w elementarer Weg. Wäre $r_j = r_l$ für $j \neq l$, so wäre auch $\alpha(r_j) = \alpha(r_l)$ im Widerspruch zur Voraussetzung, daß w elementar ist. \square

Definition 3.5 (Zusammensetzen von Wegen)

Seien $w = (r_1, \dots, r_k)$ und $w' = (r'_1, \dots, r'_{k'})$ Wege in G mit $\omega(w) = \alpha(w')$. Dann definieren wir die **Zusammensetzung** von w und w' durch

$$w \circ w' := (r_1, \dots, r_k, r'_1, \dots, r'_{k'}).$$

Lemma 3.6 *Sei G ein endlicher Graph mit $g^+(v) > 0$ für alle $v \in V$. Dann besitzt G einen elementaren Kreis.*

Beweis: Wenn G eine Schlinge r besitzt, ist $w = (r)$ bereits ein elementarer Kreis in G . Sei daher im folgenden G schlingenfrei.

Sei $w = (r_1, \dots, r_k)$ ein längster elementarer Weg in G (so ein Weg existiert, da G elementare Wege enthält und jeder elementare Weg höchstens Länge $|V|$ besitzt).

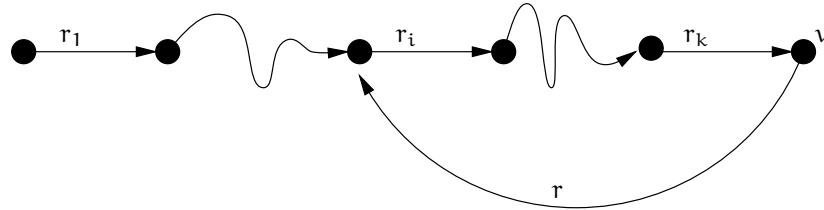


Abbildung 3.1:
 G besitzt einen einfachen Kreis.

Sei $v := \omega(w)$. Da $g^+(v) > 0$, gibt es einen Pfeil r mit $\alpha(r) = v$. Dann muß $\omega(r) \in s(w)$ gelten (sonst wäre $w \circ r$ ein längerer elementarer Weg in G), etwa $\omega(r) = \alpha(r_i)$ (siehe Bild 3.1).

Dann ist aber $(r_i, r_{i+1}, \dots, r_k, r)$ ein elementarer Kreis in G . \square
Analog zeigt man folgendes Lemma:

Lemma 3.7 *Sei G ein endlicher Graph mit $g^-(v) > 0$ für alle $v \in V$. Dann besitzt G einen einfachen Kreis.*

Ein weiteres hilfreiches Lemma ist das folgende:

Lemma 3.8 *Sei $w = (r_1, \dots, r_k)$ ein Kreis in G mit Spur $s(w) = (v_1, \dots, v_{k+1} = v_1)$ und sei $1 \leq i \leq k$ beliebig. Dann existiert in G ein Kreis w' mit Spur*

$$s(w') = (v_i, v_{i+1}, \dots, v_k, v_1, \dots, v_{i-1}, v_i).$$

Beweis: Setze $w' := (r_i, r_{i+1}, \dots, r_k, r_1, \dots, r_{i-1})$. \square

Definition 3.9 (Erreichbarkeit)

Die Ecke v' heißt (in G) von v **erreichbar**, wenn entweder $v = v'$ gilt oder es einen Weg w mit $\alpha(w) = v$ und $\omega(w) = v'$ gibt. Mit $E_G(v)$ bezeichnen wir die von v aus (in G) erreichbaren Ecken.

Oftmals schreibt man einfacher $E(v)$ statt $E_G(v)$, wenn klar ist, um welchen Graphen es sich handelt.

3.2 Zusammenhang

Definition 3.10 (Zusammenhang, Zusammenhangskomponente)

Zwei Ecken v und v' heißen **(stark) zusammenhängend** (i.Z. $v \sim v'$), wenn $v' \in E_G(v)$ und $v \in E_G(v')$ gilt. Die Menge aller Ecken, die mit v zusammenhängen, nennt man die **(starke) Zusammenhangskomponente** von v , i.Z. $ZK(v)$.

Definition 3.11 (Äquivalenzrelation, Äquivalenzklasse)

Sei U eine Menge. Eine Menge $R \subseteq U \times U$ heißt **Äquivalenzrelation** auf U , wenn gilt:

1. $(u, u) \in R$ für alle $u \in U$ (Reflexivität)
2. $(u, v) \in R \Rightarrow (v, u) \in R$ (Symmetrie)
3. $(u, v) \in R \wedge (v, w) \in R \Rightarrow (u, w) \in R$ (Transitivität).

Man schreibt auch uRv statt $(u, v) \in R$.

Ist R eine Äquivalenzrelation auf U und $u \in U$, so bezeichnet man mit $[u] := \{v \in U : uRv\}$ die **Äquivalenzklasse** von u .

Satz 3.12 Sei R eine Äquivalenzrelation auf U . Die Äquivalenzklassen von R bilden eine Partition von U .

Beweis: Zu zeigen:

1. Jedes $u \in U$ ist in (mindestens) einer Äquivalenzklasse enthalten.
2. Zwei Äquivalenzklassen sind entweder disjunkt oder identisch.

Zu 1: Nach Definition gilt $u \in [u]$.

Zu 2: Seien $[u]$ und $[u']$ Äquivalenzklassen. Es gelte $[u] \cap [u'] \neq \emptyset$, etwa $z \in [u] \cap [u']$. Zu zeigen: $[u] = [u']$.

Sei $v \in [u]$. Dann: uRv . Wegen $z \in [u]$ gilt uRz . Aus Symmetrie und Transitivität von R folgt: vRz .

Wegen $z \in [u']$ und der Symmetrie von R gilt auch zRu' .

Aus vRz und zRu' folgt nun $v \in [u']$.

Da $v \in [u]$ beliebig, gilt $[u] \subseteq [u']$. Analog zeigt man $[u'] \subseteq [u]$.
Insgesamt: $[u] = [u']$. \square

Lemma 3.13 Die Relation \sim ist eine Äquivalenzrelation auf V .

Beweis: Übung. \square

Nach Lemma 3.13 können wir die Zusammenhangskomponenten eines Graphen (siehe Definition 3.10) äquivalent auch wie folgt einführen:

Definition 3.14 (Zusammenhangskomponenten eines Graphen)

Die Äquivalenzklassen bezüglich \sim nennt man die **(starken) Zusammenhangskomponenten** von G . Falls G nur eine Zusammenhangskomponente besitzt, so ist G **(stark) zusammenhängend**.

Korollar 3.15 Die Zusammenhangskomponenten eines gerichteten Graphen $G = (V, R, \alpha, \omega)$ bilden eine Partition der Eckenmenge.

Beweis: Folgt aus Lemma 3.13 und Satz 3.12. \square

Bemerkung 3.16 Ist G ein (endlicher) Graph mit Zusammenhangskomponenten ZK_1, \dots, ZK_p , so bezeichnen wir oft auch den durch ZK_i induzierten Subgraphen $G[ZK_i]$ als Zusammenhangskomponente.

Definition 3.17 (Symmetrische Hülle)

Sei $G = (V, R, \alpha, \omega)$ ein gerichteter Graph. Die **symmetrische Hülle** von G ist dann der Graph $G^{sym} = (V, R \cup R^{-1}, \alpha', \omega')$ mit $R^{-1} := \{r^{-1} : r \in R\}$ wobei $\alpha'(r^{-1}) = \omega(r)$, $\omega'(r^{-1}) = \alpha(r)$, $\alpha'|_R = \alpha$ und $\omega'|_R = \omega$.

G heißt **schwach zusammenhängend**, wenn G^{sym} stark zusammenhängend ist.

Bemerkung 3.18 Die Begriffe „zusammenhängend“ und „Zusammenhangskomponenten“ sind analog für ungerichtete Graphen definiert. Beispielsweise sind u und v zusammenhängend, i.Z. $u \sim v$, wenn es einen Weg zwischen u und v gibt. Die Unterscheidung zwischen starkem und schwachem Zusammenhang gibt es im ungerichteten Fall natürlich nicht.

3.3 Der Satz von Euler

Definition 3.19 (Eulerscher Weg, Eulerscher Kreis)

Ein Weg w heißt **Eulerscher Weg**, wenn w jeden Pfeil aus R genau einmal durchläuft. Ist w zusätzlich ein Kreis, so nennt man w auch **Eulerschen Kreis**.

Einen Graphen G nennt man **Eulersch**, wenn er einen Eulerschen Kreis enthält.

Satz 3.20 (Satz von Euler) Ein gerichteter endlicher schwach zusammenhängender Graph $G = (V, R, \alpha, \omega)$ mit $R \neq \emptyset$ ist genau dann Eulersch, wenn $g^+(v) = g^-(v)$ für alle $v \in V$ gilt.

Beweis:

„ \Rightarrow “

Sei G Eulersch und schwach zusammenhängend. Sei $w = (r_1, \dots, r_k)$ ein Eulerscher Kreis. Da G (schwach) zusammenhängend ist, berührt w alle Ecken.

Bewegt man sich entlang von w , so liefert jeder Durchgang durch eine Ecke v den Beitrag 1 zum Innengrad und den Beitrag 1 zum Außengrad von v . Da w alle Pfeile enthält (und alle Ecken berührt werden), folgt $g^+(v) = g^-(v)$ für alle $v \in V$.

„ \Leftarrow “

Induktion nach $|R|$.

Induktionsanfang: $|R| = 1$.

In diesem Fall muß $V = \{v\}$ und $R = \{r\}$ mit $\alpha(r) = \omega(r) = v$ gelten. Dann ist $w = (r)$ ein Eulerscher Kreis in G .

Induktionsschritt: Es sei G schwach zusammenhängend mit $g^+(v) = g^-(v)$ für alle $v \in V$ und $|R| = k > 1$.

Induktionsvoraussetzung: Jeder schwach zusammenhängende Graph $H = (V_H, R_H, \alpha_H, \omega_H)$ mit $g^+(v) = g^-(v)$ für alle $v \in V_H$ und $0 < |R_H| < k$ besitzt einen Eulerschen Kreis.

Da G schwach zusammenhängend ist, gilt $g^+(v) > 0$ für alle $v \in V$ (sonst wäre $g^+(v) = g^-(v) = 0$ für ein v und v wäre isolierte Ecke). Nach Lemma 3.6 besitzt G einen einfachen Kreis $w = (r_1, \dots, r_k)$.

1. Fall: $R = \{r_1, \dots, r_k\}$

Dann ist w ein Eulerscher Kreis.

2. Fall: $R \neq \{r_1, \dots, r_k\}$

Betrachte $G' := G_{R \setminus \{r_1, \dots, r_k\}}$. In G' gilt für jede Ecke $g^+(v) = g^-(v)$. Seien ZK_1, \dots, ZK_p die schwachen Zusammenhangskomponenten von G' .

Falls $p = 1$, d.h. falls G' schwach zusammenhängend ist, können wir die Induktionsvoraussetzung auf G' anwenden. Diese liefert nun einen Eulerschen Kreis w' in G' . Da G schwach zusammenhängend ist, müssen die Kreise w und w' eine gemeinsame Ecke v in ihrer Spur besitzen. Nach Lemma 3.8 können wir o.B.d.A. voraussetzen, daß $\alpha(w) = \omega(w) = \alpha(w') = \omega(w') = v$ ist. Dann ist $w \circ w'$ ein Eulerscher Kreis in G .

Im folgenden sei daher $p > 1$, d.h. G' nicht schwach zusammenhängend. Nach Induktionsvoraussetzung besitzt jede $ZK_i = (V_i, R_i)$ von G' mit $R_i \neq \emptyset$ einen Eulerschen Kreis w_i , $i = 1, \dots, p$ (falls $R_i = \emptyset$, so ist V_i einelementig).

Da G schwach zusammenhängend ist, berührt der Kreis w jede ZK . Wir konstruieren nun einen Eulerschen Kreis in G wie folgt:

Sei $s(w) = (v_1, \dots, v_{k+1} = v_1)$ die Spur von $w = (r_1, \dots, r_k)$. O.B.d.A. gelte $v_1 \in ZK_1$ mit $ZK_1 = (V_1, R_1)$.

Sei $i_2 := \min\{i : v_i \notin ZK_1\} > 1$.

Falls $R_1 \neq \emptyset$, so sei w_1 der nach IV existierende Eulersche Kreis in ZK_1 . Nach Lemma 3.8 können wir o.B.d.A. voraussetzen, daß $\alpha(w_1) = \omega(w_1) = v_1$ ist. Wir setzen dann $w' := w_1 \circ (r_1, \dots, r_{i_2-1})$. Falls $R_1 = \emptyset$, so sei $w' := (r_1, \dots, r_{i_2-1})$.

Dann ist w' ein einfacher Weg in G , mit folgenden Eigenschaften:

1. $\alpha(w') = \alpha(w) = v_1$.
2. $\omega(w') = v_{i_2} \notin ZK_1$.
3. Jeder Pfeil aus R_1 wird von w' genau einmal durchlaufen.

Sei nun o.B.d.A. $v_{i_2} \in ZK_2$. Sei $i_3 := \min\{i \geq i_2 : v_i \notin ZK_1 \cup ZK_2\} > i_2$.

Falls $R_2 \neq \emptyset$, so setzen wir $w' := w' \circ w_2 \circ (r_{i_2}, \dots, r_{i_3-1})$, wobei w_2 der Eulersche Kreis in ZK_2 ist, der nach IV existiert und von dem wir wieder o.B.d.A. annehmen, daß $\alpha(w_2) = \omega(w_2) = v_{i_2}$ gilt.

Falls $R_2 = \emptyset$, so setzen wir $w' := w' \circ (r_{i_2}, \dots, r_{i_3-1})$.

Dann ist w' ein einfacher Weg mit :

1. $\alpha(w') = \alpha(w) = v_1$.
2. $\omega(w') = v_{i_3}$.
3. Jeder Pfeil aus $R_1 \cup R_2$ wird von w' genau einmal durchlaufen.

Fortsetzung des obigen Verfahrens liefert einen Eulerschen Kreis in G . In Abbildung 3.2 ist der Beweis veranschaulicht. \square

Korollar 3.21 *Jeder endliche schwach zusammenhängende Graph G mit $g^+(v) = g^-(v)$ für alle $v \in V$ ist stark zusammenhängend.* \square

Zur Erinnerung: Königsberger Brückenproblem

Frage: Gibt es einen Eulerschen Kreis im (ungerichteten) Graphen K ? (siehe Bild 3.3)

Analog zu Satz 3.20 beweist man den folgenden Satz:

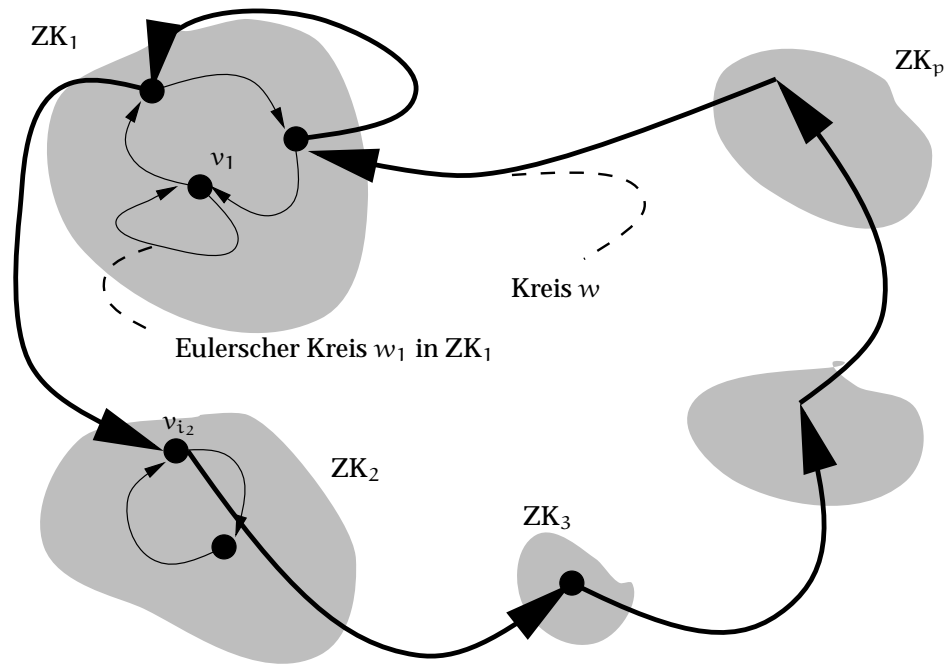


Abbildung 3.2:
Beweis des Satzes von Euler.

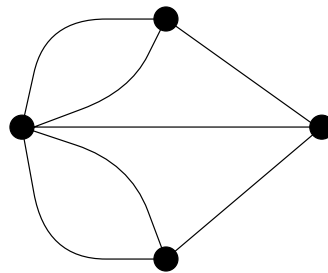


Abbildung 3.3:
Graph K des Königsberger
Brückenproblems.

Satz 3.22 (Satz von Euler für ungerichtete Graphen) *Ein ungerichteter endlicher und zusammenhängender Graph $G = (V, E, \gamma)$ mit $E \neq \emptyset$ ist genau dann Eulersch, wenn der Grad jeder Ecke von G gerade ist.*

Damit: Der Graph K aus Bild 3.3 ist nicht Eulersch.

Bemerkung 3.23 1. Offenbar läßt sich (mit Hilfe von Satz 3.20) in Polynomialzeit entscheiden, ob ein vorgegebener zusammenhängender Graph Eulersch ist.

2. Der Beweis von Satz 3.20 liefert auch schon einen einfachen Algorithmus zur Bestimmung eines Eulerschen Kreises, wenn ein solcher existiert.

3.4 Hamiltonsche Kreise

Definition 3.24 (Hamiltonscher Weg, Hamiltonscher Kreis)

Ein Weg w heißt **Hamiltonscher Weg**, wenn w jede Ecke aus V genau einmal durchläuft. Ist w zusätzlich ein Kreis, so nennt man w **Hamiltonschen Kreis**.

Einen Graphen G nennt man **Hamiltonsch**, wenn er einen Hamiltonschen Kreis enthält.

Satz 3.25 Die Frage, ob ein zusammenhängender (gerichteter oder ungerichteter) Graph G Hamiltonsch ist, ist NP-vollständig.

Beweis: Siehe beispielsweise [GJ79]. □

Das Problem zu entscheiden, ob ein Graph Hamiltonsch ist, nennen wir HAMILTONSCHER KREIS.

Übungsaufgaben

Aufgabe 3.1 – Starker Zusammenhang

- (a) Beweisen Sie, daß die in Definition 3.10 definierte Relation „stark zusammenhängend (\sim)“ eine Äquivalenzrelation ist.
- (b) Sei $G = (V, R, \alpha, \omega)$ ein gerichteter endlicher stark zusammenhängender Graph mit $|V| \geq 2$. Zeigen Sie, daß dann $|R| \geq |V|$ gilt.

Aufgabe 3.2 – Wege

Sei $G = (V, R, \alpha, \omega)$ ein gerichteter Graph und w ein Weg in G mit $\alpha(w) = u$ und $\omega(w) = v$. Zeigen Sie, daß es dann einen *elementaren* Weg w' in G mit $\alpha(w') = u$ und $\omega(w') = v$ gibt.

Aufgabe 3.3 – Zusammenhang ungerichteter Graphen

Sei $G = (V, E, \gamma)$ ein ungerichteter endlicher Graph. Mit $k(G)$ bezeichnen wir die *Anzahl der Zusammenhangskomponenten* von G . Sei $e \in E$ eine Kante von G und $G - e := (V, E \setminus \{e\}, \gamma)$ der Graph, den man erhält, wenn man aus G die Kante e entfernt. Beweisen Sie, daß dann die Ungleichung

$$k(G) \leq k(G - e) \leq k(G) + 1$$

gilt.

Aufgabe 3.4 – Bipartite Graphen

In dieser Aufgabe sei $G = (V, E)$ ein endlicher einfacher ungerichteter und zusammenhängender Graph. Man nennt G *bipartit*, wenn es eine Partition $V = A \cup B$, $A \cap B = \emptyset$ der Eckenmenge gibt, so daß $(u, v) \in E$ impliziert, daß entweder $u \in A$ und $v \in B$ oder $u \in B$ und $v \in A$ gilt.

Beweisen Sie, daß G genau dann bipartit ist, wenn es in G keinen elementaren Kreis ungerader Länge gibt.

Aufgabe 3.5 – Artikulationspunkte

In dieser Aufgabe sei $G = (V, E)$ ein endlicher einfacher ungerichteter und zusammenhängender Graph. Eine Ecke $v \in V$ heißt *Artikulationspunkt* von G , wenn $G[V \setminus \{v\}]$ nicht mehr zusammenhängend ist. Beweisen Sie, daß die folgenden zwei Aussagen äquivalent sind:

1. Die Ecke v ist ein Artikulationspunkt von G .
2. Es existieren zwei von v verschiedene Ecken x und y , so daß v Element der Spur jedes Weges von x nach y ist.

Aufgabe 3.6 – Topologische Sortierung

In dieser Aufgabe sei $G = (V, R, \alpha, \omega)$ ein endlicher gerichteter Graph. Eine *topologische Sortierung* von G ist eine bijektive Abbildung $\sigma: V \rightarrow \{1, 2, \dots, |V|\}$, mit folgender Eigenschaft:

$$\sigma(\alpha(r)) < \sigma(\omega(r)) \quad \text{für alle } r \in R.$$

- (a) Zeigen Sie, daß für G genau dann eine topologische Sortierung existiert, wenn G kreisfrei ist.
- (b) Geben Sie einen Algorithmus an, der für einen gerichteten Graphen G in Adjazenzlistendarstellung in $\mathcal{O}(|V| + |R|)$ Zeit prüft, ob G kreisfrei ist.

Aufgabe 3.7 – Graphfärbungen

Eine *k-Färbung* eines (gerichteten oder ungerichteten) Graphen G mit Eckenmenge V ist eine surjektive Abbildung $f: V \rightarrow \{1, \dots, k\}$, so daß für alle adjazenten Ecken $u, v \in V$ gilt: $f(u) \neq f(v)$. Zwei Ecken dürfen also höchstens dann die gleiche Farbe erhalten, wenn sie nicht durch eine Kante verbunden sind.

- (a) Sei $G = (V, R)$ ein gerichteter endlicher Graph mit maximalem Grad $\Delta := \max_{v \in V} g(v)$. Zeigen Sie, daß man G mit $\Delta + 1$ Farben färben kann.
- (b) Sei $G = (V, R)$ wieder ein gerichteter endlicher Graph, der keinen Kreis besitzt. Sei ℓ die maximale Länge eines elementaren Weges in G . Beweisen Sie, daß man G mit $\ell + 1$ Farben färben kann.

Aufgabe 3.8 – Graphfärbungen und chromatische Zahl

Sei $G = (V, E)$ ein einfacher ungerichteter Graph. Eine Funktion $f: V \rightarrow \{1, \dots, k\}$ heißt *k-Färbung des Graphen*, wenn für jede Kante $e = (v, w) \in E$ gilt: $f(v) \neq f(w)$. Die Eckenteilmenge $\{v \in V \mid f(v) = i\}$, heißt *Farbklasse* zur Farbe i , $i = 1, \dots, k$.

Die *chromatische Zahl* $\chi(G)$ eines Graphen ist definiert durch

$$\chi(G) := \min\{k \in \mathbb{N} \mid \text{es gibt eine } k\text{-Färbung von } G\}.$$

Für zwei Graphen $G_1 = (V, E_1)$ und $G_2 = (V, E_2)$ mit gleicher Eckenmenge wird durch $G_1 \cup G_2 := (V, E_1 \cup E_2)$ die *Vereinigung* der beiden Graphen definiert.

- a. Zeigen Sie: Hat jede Ecke $v \in V$ des Graphen $G = (V, E)$ einen Grad $g(v) \leq k$, dann gilt:

$$\chi(G) \leq k + 1.$$

Geben Sie dazu einen iterativen Algorithmus an, der eine gültige Färbung mit höchstens $k + 1$ Farben ermittelt.

- b. Zeigen Sie:

$$\chi(G_1 \cup G_2) \leq \chi(G_1) \cdot \chi(G_2).$$

Aufgabe 3.9 – Kritische Graphfärbungen

Ein Graph $G = (V, E)$ mit chromatischer Zahl $\chi = \chi(G)$ heißt *kritisch χ -chromatisch*, wenn gilt:

$$\forall e \in E: \chi((V, E \setminus e)) < \chi,$$

das heißt, durch Entfernen einer beliebigen Kante aus dem Graphen nimmt die chromatische Zahl ab.

- a. Geben Sie für $k = 2, 3, 4, \dots$ eine Familie von kritisch k -chromatischen Graphen an.
- b. Geben Sie für $n = 3, 5, 7, \dots$ eine Familie von kritisch 3-chromatischen Graphen mit n Ecken an.

Aufgabe 3.10 – Eulersche Graphen

Man gebe einen Algorithmus an, der für einen schwach zusammenhängenden Graphen G in $\mathcal{O}(|V| + |R|)$ Schritten bestimmt, ob dieser Eulersch ist und, falls G Eulersch ist, einen Eulerschen Kreis bestimmt.

Aufgabe 3.11 – Hamiltonsche Kreise

Im folgenden sei $G = (V, E)$ ein endlicher einfacher ungerichteter Graph. Seien u und v nicht-adjazente Ecken von G mit $g(u) + g(v) \geq |V|$ und $G' = (V, E \cup \{(u, v)\})$ der Graph, den man aus G erhält, wenn man die Kante (u, v) einfügt. Zeigen Sie, daß G genau dann Hamiltonsch ist, wenn G' Hamiltonsch ist.

Hinweis: Benutzen für die Rückrichtung (G' Hamiltonsch $\Rightarrow G$ Hamiltonsch) einen elementaren Weg w in G' , der jede Ecke genau einmal berührt, etwa $w = [u = v_1, \dots, v_n = v]$. Betrachten Sie dann die Mengen

$$A = \{v_i : 3 \leq i \leq n - 1 \text{ und } (u, v_i) \in E\}$$

$$B = \{v_i : 3 \leq i \leq n - 1 \text{ und } (v, v_{i-1}) \in E\}$$

Aufgabe 3.12 – Hamiltonsche Wege

Sei $G = (V, E)$ ein einfacher ungerichteter Graph mit n Ecken. Ein Weg w in G heißt *hamiltonsch*, falls w jede Ecke $v \in V$ genau einmal berührt. Ein Graph G heißt *hamiltonsch zusammenhängend*, falls es für jedes Eckenpaar $v, w \in V, v \neq w$, einen hamiltonschen Weg in G von v nach w gibt.

- a. Zeigen Sie: Für jedes $n \in \mathbb{N}$ gibt es einen Graphen G_n mit n Ecken und höchstens $2n - 2$ Kanten, der hamiltonsch zusammenhängend ist.
- b. Zeigen Sie: Jeder hamiltonsch zusammenhängende Graph mit mehr als zwei Ecken enthält einen hamiltonschen Kreis.

Aufgabe 3.13 – Starker Zusammenhang

- a. Ist $G = (V, R, \alpha, \omega)$ stark zusammenhängend und $|V| \geq 2$, so gilt:

$$|R| \geq |V|.$$

- b. Ist $G = (V, R, \alpha, \omega)$ schwach zusammenhängend, und sind seine starken Zusammenhangskomponenten $Z_1, Z_2, \dots, Z_p, \dots, Z_q$ so numeriert, daß $|Z_i| \geq 2 \iff i \leq p$, so gilt:

$$|R| \geq |V| + p - 1.$$

Aufgabe 3.14 – Distanzen in Graphen

Sei $G = (V, R, \alpha, \omega)$ ein gerichteter Graph, ferner sei $l: R \rightarrow \mathbb{Q}$ eine Pfeilbewertung. Für einen Weg $w = (r_1, \dots, r_k)$ ist dann $l(w) := \sum_{i=1}^k l(r_i)$ die *Länge* des Weges. Falls $l \equiv 1$ („*Einheitsbewertung*“), ist die Länge des Weges identisch mit der Anzahl seiner Pfeile.

Die *Distanz* $d_G(v_1, v_2)$ zweier Ecken im Graphen wird dann definiert durch

$$d_G(v_1, v_2) := \begin{cases} 0, & \text{falls } v_1 = v_2, \\ \inf\{l(w) \mid w \in W_{v_1, v_2}\}, & \text{falls } W_{v_1, v_2} \neq \emptyset, \\ +\infty, & \text{falls } W_{v_1, v_2} = \emptyset, \end{cases}$$

wobei mit W_{v_1, v_2} die Menge aller Wege von v_1 nach v_2 bezeichnet ist. Es ist anzumerken, daß sowohl Weglängen als auch Distanzen negativ sein können. Der Fall $d_G(v_1, v_2) = -\infty$ tritt auf, wenn auf einem Weg von v_1 nach v_2 ein Kreis negativer Länge liegt.

Weisen Sie nach, daß d_G die *Dreiecksungleichung*

$$d_G(v_1, v_3) \leq d_G(v_1, v_2) + d_G(v_2, v_3) \quad \text{für alle } v_1, v_2, v_3 \in V, v_1 \neq v_3$$

erfüllt. Die Ungleichung „ $+\infty \leq a+b$ “ sei bereits dann erfüllt, wenn $a = +\infty$ oder $b = +\infty$ gilt.

Aufgabe 3.15 – Zusammenhang

Ein ungerichteter Graph heißt *2-fach zusammenhängend*, wenn der durch Entfernen je einer beliebigen Kante entstehende Graph noch zusammenhängend ist.

Zeigen Sie: Ein zusammenhängender Graph $G = (V, E)$ ist genau dann 2-fach zusammenhängend, wenn jede Kante $e \in E$ auf einem Kreis in G liegt.

Aufgabe 3.16 – Graphfärbungen und Partitionen

Sei $G = (V, E)$ ein beliebiger Graph. Unter einer *Partition* $V = V_1 \dot{\cup} V_2$ der Eckenmenge versteht man eine Zerlegung $V = V_1 \cup V_2$ mit $V_1 \cap V_2 = \emptyset$ und $V_i \neq \emptyset$.

Zeigen Sie für den Fall $E \neq \emptyset$:

- a. Es gibt eine Partition $V = V_1 \dot{\cup} V_2$ der Eckenmenge, so daß

$$\chi(G[V_1]) + \chi(G[V_2]) = \chi(G).$$

- b. Wenn G nicht vollständig ist, dann gibt es eine Partition $V = V_1 \dot{\cup} V_2$ der Eckenmenge, so daß

$$\chi(G[V_1]) + \chi(G[V_2]) > \chi(G).$$

Hinweis: Wählen Sie V_1 als eine maximale Clique im Graphen.

Kapitel 4

Bestimmung von Zusammenhangskomponenten

4.1 Transitive Hülle

Definition 4.1 (Transitivität)

Ein Graph $G = (V, R, \alpha, \omega)$ heißt **transitiv**, wenn gilt: Sind $r, r' \in R$ mit $\omega(r) = \alpha(r')$, so existiert ein Pfeil $r'' \in R$ mit $\alpha(r'') = \alpha(r)$ und $\omega(r'') = \omega(r')$.

Definition 4.2 (Transitive Hülle)

Sei $G = (V, R)$ ein endlicher Graph ohne Parallelen. Ein Graph $G^* = (V, R^*)$ heißt **transitive Hülle** von G , wenn

1. $G \subseteq G^*$
2. G^* transitiv und
3. $G' \text{ transitiv} \wedge G \subseteq G' \Rightarrow G^* \subseteq G'$.

Anschaulich: G^* ist ein kleinster transitiver Obergraph von G .

Lemma 4.3 Sei G ein endlicher Graph ohne Parallelen. Dann ist die transitive Hülle G^* von G eindeutig definiert.

Beweis: Eindeutigkeit: Sind G^* und H^* transitive Hüllen von G , so folgt nach Eigenschaft 3 $G^* \subseteq H^*$ und $H^* \subseteq G^*$, also $G^* = H^*$.

Existenz: Sei \mathcal{G} die Menge der parallelenfreien transitiven Obergraphen von G mit gleicher Eckenmenge wie G . Die Menge \mathcal{G} ist nichtleer, denn $G_0 := (V, V \times V) \in \mathcal{G}$. Setze $G^* := (V, R^*)$ mit

$$R^* := \bigcap_{G'=(V,R') \in \mathcal{G}} R'. \quad (4.1)$$

Wir prüfen die Eigenschaften 1 bis 3 nach. Eigenschaft 1 folgt aus $G \subseteq G'$ für alle $G' \in \mathcal{G}$. Eigenschaft 3 wird durch die Schnittbildung gesichert.

Transitivität: Seien $r, r' \in R$ mit $\omega(r) = \alpha(r')$, etwa $r = (u, v)$ und $r' = (v, w)$. Da jedes $G' \in \mathcal{G}$ transitiv ist, gilt $(u, w) \in R'$ für alle $G' \in \mathcal{G}$. Also ist $(u, w) \in R^*$. \square

Lemma 4.4 Sei $G = (V, R)$ ein gerichteter endlicher Graph und G^* die transitive Hülle von G . Dann gilt für $u \neq v$ die Beziehung $v \in E_G(u)$ genau dann, wenn $(u, v) \in R^*$.

Beweis:

„ \Rightarrow “

Sei $v \in E_G(u)$ mit $u \neq v$. Dann existiert ein Weg $w = (r_1, \dots, r_k)$ von u nach v in G . Durch Induktion nach k folgt nun aus der Transitivität von G^* : $(u, v) \in R^*$.

„ \Leftarrow “

Sei $(u, v) \in R^*$ mit $u \neq v$. Annahme: $v \notin E_G(u)$. Betrachte $G' := (V, R')$ mit

$$R' := \{(x, y) : x \in V \wedge y \in E_G(x)\}.$$

Nach dem obigen Beweis der Hinrichtung gilt $R' \subseteq R^*$. Nach der Widerspruchsanahme gilt: $R' \subset R^*$, also $G \sqsubset G' \sqsubset G^*$ und $G' \neq G^*$.

G' ist aber transitiv:

$$\begin{aligned} (x, y) \in R' \wedge (y, z) \in R' &\Leftrightarrow y \in E_G(x) \wedge z \in E_G(y) \\ &\stackrel{\text{Transitivität der Erreichbarkeit}}{\Rightarrow} z \in E_G(x) \\ &\Leftrightarrow (x, z) \in R' \end{aligned}$$

Dies ist ein Widerspruch zur Minimalität von G^* (Eigenschaft 3). \square

Definition 4.5 (Transitive reflexive Hülle)

Sei $G = (V, R)$ ein endlicher Graph ohne Parallelen und $G^* = (V, R^*)$ die transitive Hülle von G . Der Graph

$$G_{\text{refl}}^* := (V, R^* \cup \{(v, v) : v \in V\})$$

heißt dann **transitive reflexive Hülle** von G .

Korollar 4.6 Sei $G = (V, R)$ ein gerichteter endlicher Graph mit $V = \{v_1, \dots, v_n\}$ und $A(G_{\text{refl}}^*) = (a_{ij})$ die Adjazenzmatrix der transitiven reflexiven Hülle von G . Dann gilt $v_i \sim v_j$ genau dann, wenn $a_{ik} = a_{jk}$ für $k = 1, \dots, n$, d.h. wenn die i te und die j te Zeile von $A(G_{\text{refl}}^*)$ übereinstimmen.

Beweis: Aus Lemma 4.4 und der Definition der Adjazenzmatrix. \square

4.2 Der Tripelalgorithmus

Im folgenden sei $G = (V, R)$ ein endlicher gerichteter Graph ohne Parallelen.

Ziel: Bestimmung von $A(G_{\text{refl}}^*) = (a_{ij})$

Definiere:

$$\mathcal{G}^{\text{op}} := \{G : G = (V, R) \text{ ist endlicher Graph ohne Parallelen mit Eckenmenge } V\}$$

Definition 4.7 (Tripeloperator)

Der **Tripeloperator** T_v zur Umwegecke v ist eine Abbildung $T_v : \mathcal{G}^{\circ P} \rightarrow \mathcal{G}^{\circ P}$ mit $T_v \circ G := T_v(G) := (V, R')$ mit

$$R' := R \cup \{(v, v)\} \cup \{(v_i, v_j) : (v_i, v) \in R \wedge (v, v_j) \in R\}$$

Die Anwendung des Tripeloperators kann wie in Abb. 4.1 veranschaulicht werden.

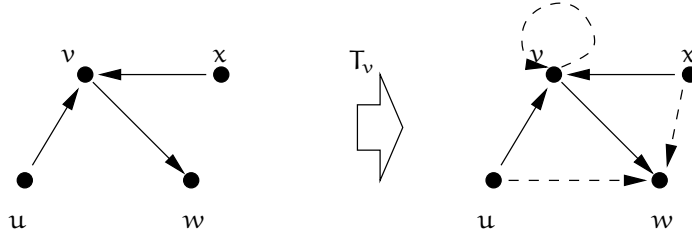


Abbildung 4.1: Ein Beispiel für die Anwendung eines Tripeloperators. Die neu hinzugekommenen Pfeile sind gestrichelt gezeichnet.

Für die sukzessive Anwendung von k Tripeloperatoren T_{v_1}, \dots, T_{v_k} schreiben wir auch

$$T_{v_k} \circ \dots \circ T_{v_1} \circ G := \prod_{i=1}^k T_{v_i} \circ G.$$

Bemerkung 4.8 Die Tripeloperatoren besitzen folgende „Monotonieeigenschaft“. Sind $G = (V, R)$ und $G' = (V, R')$ Graphen mit $G \sqsubseteq G'$ und $v \in V$, so gilt $T_v \circ G \sqsubseteq T_v \circ G'$.

Satz 4.9 Sei $G = (V, R)$ mit $V = \{v_1, \dots, v_n\}$ und $\pi : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$ eine beliebige Permutation. Dann gilt

$$G_{\text{refl}}^* = \prod_{i=1}^n T_{v_{\pi(i)}} \circ G \quad (= T_{v_{\pi(n)}} \circ \dots \circ T_{v_{\pi(1)}} \circ G).$$

Der Beweis von Satz 4.9 benötigt einige Hilfssätze.

Lemma 4.10 Sind $v_1, \dots, v_k \in V$, so gilt

$$\prod_{i=1}^k T_{v_i} \circ G \sqsubseteq G_{\text{refl}}^*.$$

Aus Lemma 4.10 folgt dann sofort:

$$\prod_{i=1}^n T_{v_{\pi(i)}} \circ G \sqsubseteq G_{\text{refl}}^*$$

für jede Permutation $\pi : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$.

Beweis von Lemma 4.10: Induktionsanfang: $k = 1$

Zu betrachten: $G' := (V, R') := T_v \circ G$.

Es gilt

$$\begin{aligned} R' &= R \cup \{(v, v)\} \cup \{(u, w) : (u, v) \in R \wedge (v, w) \in R\} \\ &= R \cup \{(v, v)\} \cup R_v. \end{aligned}$$

Da G_{refl}^* transitiv, folgt $R_v \subseteq R_{\text{refl}}^*$. Somit gilt $G' \subseteq G_{\text{refl}}^*$ nach Definition von G_{refl}^* .

Induktionsschritt: $k \rightarrow k+1$

$$G' := \prod_{i=1}^{k+1} T_{v_i} \circ G = T_{v_{k+1}} \circ G''$$

mit

$$G'' := \prod_{i=1}^k T_{v_i} \circ G.$$

Nach Induktionsvoraussetzung gilt $G'' \subseteq G_{\text{refl}}^*$.

Es gilt

$$\begin{aligned} R' &= R'' \cup \{(v, v)\} \cup \{(u, w) : (u, v) \in R'' \wedge (v, w) \in R''\} \\ &= R'' \cup \{(v, v)\} \cup R_{v''}. \end{aligned}$$

Aus $R'' \subseteq R_{\text{refl}}^*$ und der Transitivität von G_{refl}^* folgt wieder $R'_v \subseteq R_{\text{refl}}^*$. \square

Lemma 4.11 *Ist $w = [v_{i_0}, \dots, v_{i_k}]$ ein elementarer Weg der Länge $k \geq 2$ in einem parallelenfreien Graphen G , und $\pi : \{1, \dots, k-1\} \rightarrow \{1, \dots, k-1\}$ eine beliebige Permutation. Dann gilt*

$$(v_{i_0}, v_{i_k}) \in \prod_{j=1}^{k-1} T_{v_{i_{\pi(j)}}} \circ G.$$

Beweis: Induktion nach k .

Induktionsanfang: $k = 2$. Es sei $w = [v_{i_0}, v_{i_1}, v_{i_2}]$. Die einzig mögliche Permutation $\pi : \{1\} \rightarrow \{1\}$ ist $\pi(1) = 1$. Es gilt aber $(v_{i_0}, v_{i_2}) \in T_{v_{i_{\pi(1)}}} \circ G$.

Induktionsschritt: $k \rightarrow k+1$

Sei $w = [v_{i_0}, \dots, v_{i_{k+1}}]$. Es gilt

$$\prod_{j=1}^k T_{v_{i_{\pi(j)}}} \circ G = \prod_{j=2}^k T_{v_{i_{\pi(j)}}} \circ (T_{v_{i_{\pi(1)}}} \circ G).$$

Der Weg $w' = [v_{i_0}, \dots, v_{i_{\pi(1)-1}}, v_{i_{\pi(1)+1}}, \dots, v_{i_{k+1}}]$ ist ein elementarer Weg der Länge k im Graphen $G' := T_{v_{i_{\pi(1)}}} \circ G$ (siehe Bild 4.2).

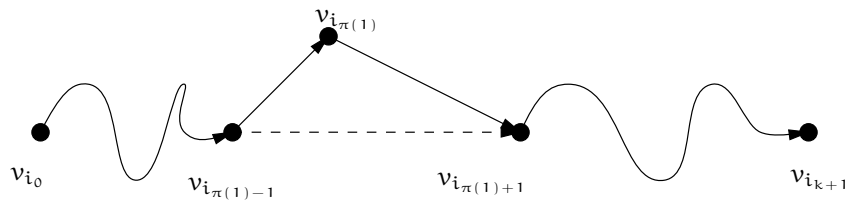


Abbildung 4.2:
Beweis von Lemma 4.11.

Nach Induktionsvoraussetzung gilt dann:

$$(v_{i_0}, v_{i_{k+1}}) \in \prod_{j=2}^k T_{v_{i_{\pi(j)}}} \circ G' = \prod_{j=2}^k T_{v_{i_{\pi(j)}}} \circ (T_{v_{i_{\pi(1)}}} \circ G).$$

□

Beweis von Satz 4.9: Wie bereits bemerkt, folgt aus Lemma 4.10:

$$G' := (V, R') := \prod_{i=1}^n T_{v_{\pi(i)}} \circ G \subseteq G_{\text{refl}}^* \quad (4.2)$$

für jede Permutation $\pi: \{1, \dots, n\} \rightarrow \{1, \dots, n\}$.

Wir zeigen nun $G_{\text{refl}}^* \subseteq G'$, was dann zusammen mit (4.2) die Gleichheit $G' = G_{\text{refl}}^*$ und damit die Behauptung des Satzes zeigt.

Sei $(v_i, v_j) \in R_{\text{refl}}^*$. Wir müssen zeigen, daß $(v_i, v_j) \in R'$ gilt.

Falls $v_i = v_j$ folgt $(v_i, v_i) \in R'$ aus $v_i \in T_{v_i} \circ G$ und der Monotonieeigenschaft der Tripeloperatoren (siehe Bemerkung 4.8).

Sei daher $v_i \neq v_j$. Nach Lemma 4.4 gilt für $v_i \neq v_j$ die Äquivalenz: $(v_i, v_j) \in R^* \Leftrightarrow v_j \in E_G(v_i)$. Sei daher w ein Weg in G mit $\alpha(w) = v_i$ und $\omega(w) = v_j$. Nach Aufgabe 3.2 können wir o.B.d.A. annehmen, daß w elementar ist.

Aus Lemma 4.11 folgt dann mit der Monotonieeigenschaft (Bemerkung 4.8) $(v_i, v_j) \in R'$. □

Algorithmus 4.1 Tripelalgorithmus

Input: Ein Graph $G = (V, R)$ ohne Parallelen, der durch seine Adjazenzmatrix $A(G) = (a_{ij})$ gegeben ist

```

1 for i ← 1, ..., n do {Betrachte Umwegecke v_i}
2   a_ii ← 1
3   for j ← 1, ..., n do
4     for k ← 1, ..., n do
5       a_jk ← max{a_jk, a_ji · a_ik}
6     end for
7   end for
8 end for
9 return A
```

Satz 4.12 Der Tripelalgorithmus bestimmt in $\Theta(|V|^3)$ Zeit die Adjazenzmatrix der transitiven reflexiven Hülle G_{refl}^* . □

Korollar 4.13 Mit dem Tripelalgorithmus kann man in $\Theta(|V|^3)$ Zeit die starken Zusammenhangskomponenten eines Graphen G bestimmen.

Beweis: Nach Korollar 4.6 gilt $v_i \sim v_j$ genau dann, wenn die i te und die j te Zeile von $A(G_{\text{refl}}^*)$ übereinstimmen. Nach Satz 4.12 ist $A(G_{\text{refl}}^*)$ in $\mathcal{O}(|V|^3)$ Zeit bestimmbar. Das „Ordnen“ der Ecken gemäß gleicher Zeilen ist ebenfalls in $\mathcal{O}(|V|^3)$ Zeit durchführbar. □

Wir werden in Kapitel 6 ein effizienteres Verfahren kennenlernen, mit dem man die Komponenten bestimmen kann.

4.3 Der Reduzierte Graph

Definition 4.14 (Reduzierter Graph)

Sei $G = (V, R, \alpha, \omega)$ ein gerichteter Graph mit den Zusammenhangskomponenten ZK_1, \dots, ZK_p . Der **reduzierte Graph** $\hat{G} = (\hat{V}, \hat{R})$ ist dann definiert durch

$$1. \hat{V} := \{ZK_1, \dots, ZK_p\}$$

$$2. \hat{R} := \{(ZK_i, ZK_j) : i \neq j \text{ und es gibt ein } r \in R \text{ mit } \alpha(r) \in ZK_i \text{ und } \omega(r) \in ZK_j\}.$$

Satz 4.15 *Der reduzierte Graph ist ein einfacher kreisfreier Graph.*

Beweis: Per Definition ist \hat{G} parallelen- und schlingenfrei, also einfach.

Widerspruchsannahme: w ist ein Kreis in \hat{G} . Da \hat{G} einfach ist, ist w durch seine Spur eindeutig charakterisiert. O.B.d.A. seien die Komponenten so numeriert, daß $s(w) = (ZK_1, ZK_2, \dots, ZK_k, ZK_1)$. Da \hat{G} schlingenfrei ist, besitzt w mindestens Länge 2.

Da $(ZK_j, ZK_{j+1}) \in \hat{R}$, gibt es $v'_j \in ZK_j$, $v_{j+1} \in ZK_{j+1}$ und $r_j \in R$ mit $\alpha(r_j) = v'_j$ und $\omega(r_j) = v_{j+1}$ ($j = 1, \dots, k-1$) (siehe Bild 4.3). Analog gibt es wegen $(ZK_k, ZK_1) \in \hat{R}$ Ecken $v'_k \in ZK_k$ und $v_1 \in ZK_1$ und einen Pfeil r_k mit $\alpha(r_k) = v'_k$ und $\omega(r_k) = v_1$.

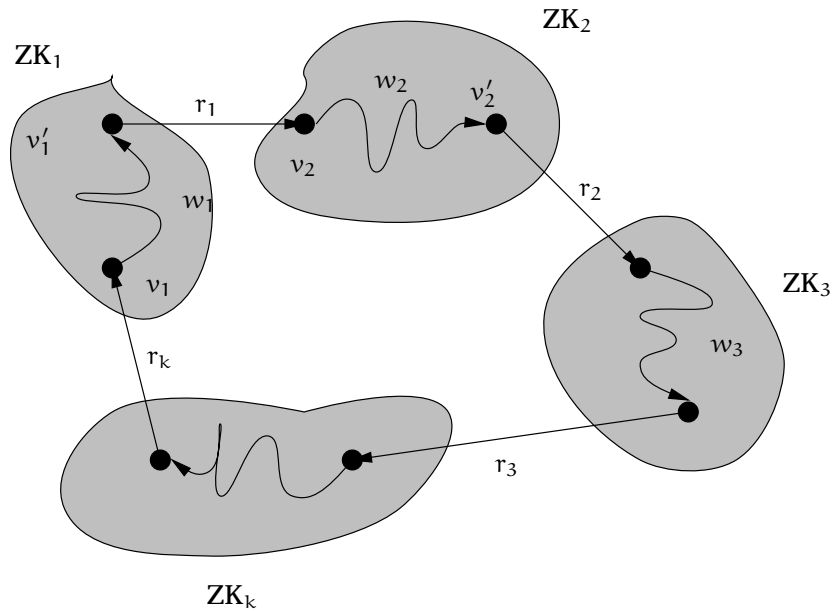


Abbildung 4.3:
Beweis von Satz 4.15.

Da $v_j, v'_j \in ZK_j$ gilt entweder $v_j = v'_j$ oder es existiert ein Weg w_j mit $\alpha(w_j) = v_j$ und $\omega(w_j) = v'_j$ für $j = 1, \dots, k$.

Dann ist aber $r_1 \circ w_2 \circ r_2 \circ \dots \circ w_k \circ r_k \circ w_1$ (wobei wir w_j weglassen, wenn $v_j = v'_j$ gilt) ein Kreis in G , der die Komponenten ZK_1, \dots, ZK_k berührt.

Insbesondere: $v_2 \in E_G(v'_1)$ wegen $(v'_1, v_2) \in R$. Außerdem: $v'_1 \in E_G(v_2)$, da $w_2 \circ r_2 \circ w_3 \circ \dots \circ w_k \circ r_k \circ w_1$ ein Weg in G von v_2 nach v'_1 ist. Somit $v'_1 \sim v_2$ im Widerspruch zu $v'_1 \in ZK_1 \neq ZK_2 \ni v_2$. \square

Aus Korollar 4.13 folgt, daß man \hat{G} in $\mathcal{O}(|V|^3)$ Zeit bestimmen kann.

Satz 4.16 *Sei $G = (V, R)$ ein parallelenfreier Graph. Dann kann der reduzierte Graph \hat{G} mit dem Tripelalgorithmus bestimmt werden, die Laufzeit liegt in $\Theta(|V|^3)$.* \square

4.4 Irreduzible Kerne

Für die transitive Hülle eines Graphen $G = (V, R)$ mußten alle „Transitivitätsbeziehungen“, die aus den Pfeilen von G folgen, eingefügt werden.

Im folgenden betrachten wir eine gewisse „Umkehrung“ dieses Problems.

Gegeben:

1. Menge $P = \{p_1, \dots, p_n\}$ von Prozessen
2. Prozeßlaufzeiten $t(p_i)$, $i = 1, \dots, n$
3. Präzedenzbeziehungen B der Form: p_i kann erst dann begonnen werden, wenn p_j beendet wurde.

Gesucht: Ein kürzester „Schedule“ d.h. eine Bijektion $s : P \rightarrow \{1, \dots, n\}$, wobei $s(p_i)$ der Anfangszeitpunkt von p_i ist.

Darstellung der Präzedenzstruktur im Präzedenzgraphen $G = (P, R_B)$ mit

$R_B = \{(p_j, p_i) : p_i \text{ kann erst dann begonnen werden, wenn } p_j \text{ beendet wurde}\}.$

Beispiel 4.17 Bild 4.4 zeigt ein Beispiel für einen Präzedenzgraphen mit Prozessen $P = \{p_1, \dots, p_5\}$.

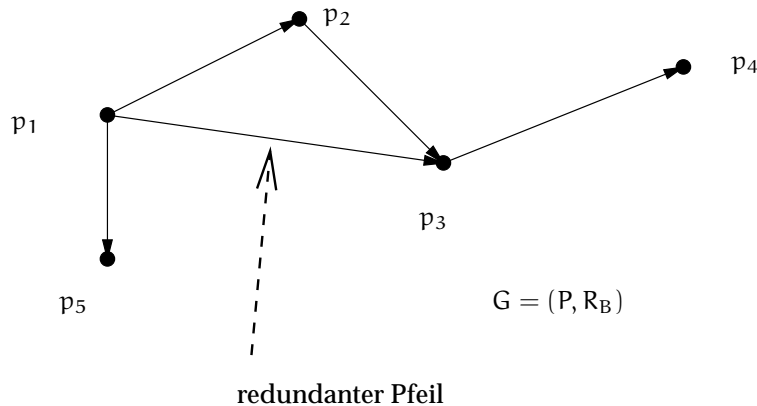


Abbildung 4.4:
Ein Präzedenzgraph.

Ein Schedule ist eine *topologische Sortierung* von G (siehe Übung 3.6).

Satz 4.18 *Ein endlicher Graph $G = (V, R, \alpha, \omega)$ besitzt genau dann eine topologische Sortierung, wenn G kreisfrei ist.*

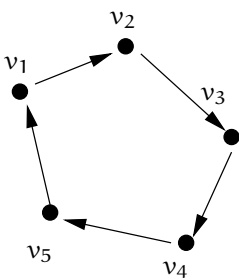
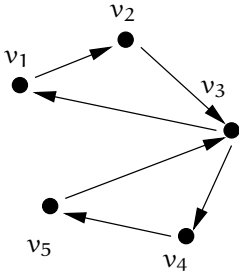
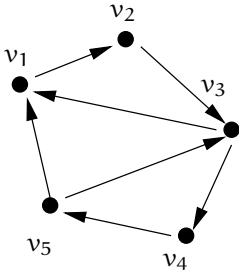
Beweis: Übung 3.6. □

Folge: Notwendig und hinreichend für die Existenz eines Schedules ist, daß der Präzedenzgraph G kreisfrei ist.

Aufgabe: Eliminieren von „redundanten Präzedenzbedingungen“

Definition 4.19 (Irreduzibler Kern)

Sei $G = (V, R)$ ein Graph. Ein Graph $G_* = (V, R_*)$ heißt **transitiv irreduzibler Kern** von G , falls gilt:



Ein Graph mit zwei verschiedenen irreduziblen Kernen.

1. $G_* \sqsubseteq G$,
2. $(G_*)^* = G^*$, d.h. G_* besitzt die gleiche transitive Hülle wie G , und
3. ist $G' \sqsubseteq G_*$ mit $G' \neq G_*$, so folgt $(G')^* \neq G^*$.

In der letzten Definition haben wir den Begriff des irreduziblen Kerns nur für parallelenfreie Graphen eingeführt. Wir werden uns ausschließlich mit diesem Fall beschäftigen, da Erreichbarkeitsbeziehungen im Graphen nicht durch die Existenz von Parallelen (oder Schlingen) beeinflusst wird. Der Vollständigkeit halber sei hier noch eine allgemeinere Definition eines irreduziblen Kerns gegeben, die auch für Graphen mit Parallelen gültig ist. Sie fällt für parallelenfreie Graphen mit Definition 4.19 zusammen.

Definition 4.20 (Irreduzibler Kern (alternative Definition))

Sei $G = (V, R, \alpha, \omega)$ ein Graph. Ein Graph $G_* = (V, R_*, \alpha|_{R_*}, \omega|_{R_*})$ heißt **transitiv irreduzibler Kern** von G , falls gilt:

1. $G_* \sqsubseteq G$,
2. In G und G_* gelten die gleichen Erreichbarkeitsbeziehungen, d.h. für Ecken $v, w \in V$ gilt: $v \in E_G(w) \Leftrightarrow v \in E_{G_*}(w)$, und
3. ist $G' \sqsubseteq G_*$ mit $G' \neq G_*$, so gelten in G' und G nicht die gleichen Erreichbarkeitsbeziehungen, d.h. es gibt $v, w \in G$ mit $v \in E_G(w)$, aber $v \notin E_{G'}(w)$.

Frage: Ist G_* eindeutig? Existiert G_* ? (vergleiche: transitive Hülle)

Beispiel 4.21 Wie nebenstehende Abbildung zeigt, ist ein irreduzibler Kern und sogar die Anzahl seiner Pfeile i. a. nicht eindeutig bestimmt.

Frage: Wie viele Pfeile besitzt ein „dünnster“ irreduzibler Kern?

Definition 4.22 (IRREDUZIBLER KERN)

Eine Instanz von IRREDUZIBLER KERN ist durch einen gerichteten Graphen $G = (V, R)$ und eine natürliche Zahl K gegeben. Das Problem ist zu entscheiden, ob G einen irreduziblen Kern $G_* = (V, R_*)$ mit $|R_*| \leq K$ besitzt.

Satz 4.23 IRREDUZIBLER KERN ist NP-vollständig.

Beweis: IRREDUZIBLER KERN liegt offenbar in NP: eine nichtdeterministische Maschine muß nur die Pfeilmenge $|R_*|$ „raten“ und dann in Polynomialzeit verifizieren, daß (V, R_*) ein irreduzibler Kern mit $|R_*| \leq K$ ist.

Wir benutzen nun eine Reduktion von HAMILTONSCHER KREIS. Wir wissen, daß HAMILTONSCHER KREIS NP-vollständig ist (Satz 3.25).

Gegeben sei eine beliebige Instanz $G = (V, R)$ von HAMILTONSCHER KREIS. O.B.d.A. sei G stark zusammenhängend (testbar in Polynomialzeit, etwa mit Tripelalgorithmus) und $|V| \geq 2$. Falls G nicht stark zusammenhängend ist, kann G nicht Hamiltonsch sein.

Wir konstruieren (in Polynomialzeit) eine Instanz $G' = (V', R')$, K' von IRREDUZIBLER KERN mit folgender Eigenschaft: G ist genau dann Hamiltonsch, wenn G' einen irreduziblen Kern mit Kardinalität höchstens K' besitzt.

Die Konstruktion ist wie folgt: $G' := G$ und $K' := |V|$. Offenbar ist sie in Polynomialzeit durchführbar.

Ist G Hamiltonsch, so besitzt G einen Hamiltonschen Kreis $w = (r_1, \dots, r_{|V|})$. Offenbar ist dann $G_* := (V, R_*)$ mit $R_* = \{r_1, \dots, r_{|V|}\}$ ein irreduzibler Kern mit $|R_*| \leq K' = |V|$.

Umgekehrt besitze G' einen irreduziblen Kern $G_* = (V, R_*)$ mit $|R_*| \leq K' = |V|$. Da G und damit auch G_* stark zusammenhängend ist, gilt nach Übung 3.1 $|R_*| \geq |V|$, also insgesamt $|R_*| = |V|$.

Da G_* stark zusammenhängend ist, gilt in G_* : $g^-(v) > 0$ und $g^+(v) > 0$ für alle $v \in V$. Aus $\sum_{v \in V} g^+(v) = \sum_{v \in V} g^-(v) = |R_*| = |V|$ folgt: $g^+(v) = g^-(v) = 1$ für alle $v \in V$.

Nach Satz 3.20 besitzt G_* einen Eulerschen Kreis $w = (r_1, \dots, r_{|V|})$ (der wegen des starken Zusammenhangs von G_* alle Ecken $v \in V$ berührt). Wegen $g^+(v) = g^-(v) = 1$ für alle $v \in V$ muß w elementarer Kreis sein. Also ist w ein Hamiltonscher Kreis. \square

Satz 4.24 Sei $G = (V, R)$ ein endlicher gerichteter Graph ohne Parallelen, der keinen Kreis besitzt. Dann gelten folgende Aussagen:

1. G besitzt einen eindeutigen irreduziblen Kern $G_* = (V, R_*)$.
2. Für G_* gilt die Abschätzung: $|R_*| \leq \lfloor |V|^2/4 \rfloor$.

Beweis: Übung. \square

Definition 4.25 (Wesentliche und unwesentliche Pfeile)

Sei $G = (V, R)$ und $r \in R$. Der Pfeil r heißt **wesentlich**, wenn $(V, R \setminus \{r\})^* \neq G^*$. Ansonsten heißt r **unwesentlich** oder **redundant**. Der Graph G heißt **irreduzibel**, falls R nur wesentliche Pfeile enthält.

Das folgende Lemma zeigt, daß die letzte Definition mit der Definition von irreduziblen Kernen kompatibel ist.

Lemma 4.26 Sei G ein endlicher gerichteter Graph und G_* ein irreduzibler Kern von G . Dann ist G_* irreduzibel nach Definition 4.25.

Beweis: Besäße G_* einen reduzierbaren Pfeil r , so erhielten wir durch Entfernen des Pfeils r aus G_* einen echten Teilgraphen G' von G_* mit gleicher transitiver Hülle wie G_* (also auch gleicher transitiver Hülle wie G). Dies widerspricht der Minimalitätseigenschaft eines irreduziblen Kernes (Eigenschaft 3 in Definition 4.19). \square

Nun: Einfacher „Wegwerf-Algorithmus“ zur Konstruktion eines irreduziblen Kernes (Algorithmus 4.2).

Algorithmus 4.2 Wegwerf-Algorithmus zur Konstruktion eines irreduziblen Kernes.

Input: Ein Graph $G = (V, R)$ ohne Parallelen

```

1  $R_* \leftarrow R$ 
2 while  $(V, R_*)$  enthält einen redundanten Pfeil  $r \in R_*$  do
3    $R_* \leftarrow R_* \setminus \{r\}$ 
4 end while
return  $(V, R_*)$ 

```

Satz 4.27 Algorithmus 4.2 konstruiert einen irreduziblen Kern von G . Er kann so implementiert werden, daß er in $\mathcal{O}(|V|^3 \cdot |R|^2)$ Zeit läuft.

Beweis: Die Korrektheit folgt unmittelbar aus der Konstruktion. Jeder Test in Schritt 2 kann durch $|R|$ Anwendungen des Tripelalgorithmus (für $(V, R \setminus \{r\})$, $r \in R$) erfolgen. Die **while**-Schleife wird höchstens $|R|$ mal durchlaufen. \square

Übungsaufgaben

Aufgabe 4.1 – Tripeloperatoren

Sei $G = (V, R)$ ein endlicher Graph ohne Parallelen. Beweisen oder widerlegen Sie, daß für zwei Tripeloperatoren T_u und T_v ($u, v \in V$) gilt:

$$T_u \circ T_v \circ G = T_v \circ T_u \circ G.$$

Aufgabe 4.2 – Irreduzible Kerne

Sei $G = (V, R)$ ein endlicher gerichteter Graph ohne Parallelen, der keinen Kreis besitzt. Beweisen Sie, daß G einen eindeutigen irreduziblen Kern $G_* = (V, R_*)$ besitzt.

Aufgabe 4.3 – Transitiv Hülle und reduzierter Graph

Bestimmen Sie mit Hilfe des Tripelalgorithmus die transitive reflexive Hülle G_{refl}^* und den reduzierten Graphen \hat{G} für den in Abbildung 4.5 gezeichneten Graphen G .

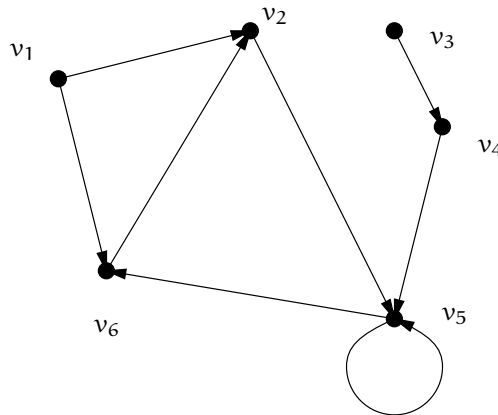


Abbildung 4.5:
Graph zu Übung 4.3

Aufgabe 4.4 – Irreduzible Kerne

Sei $G = (V, R)$ ein stark zusammenhängender Graph, und sei $G_* = (V, R_*)$ ein irreduzibler Kern. Zeigen Sie:

$$|V| \leq |R_*| \leq 2 \cdot (|V| - 1).$$

Hinweis: Wählen Sie eine Ecke $v \in V$ und Teilgraphen $G_{\text{out}}, G_{\text{in}}$ von G mit $E_{G_{\text{out}}}(v) = V$ und $v \in E_{G_{\text{in}}}(v')$ für alle $v' \in V$. Kann v beliebig gewählt werden?

Kapitel 5

Bäume, Wälder und Matroide

5.1 Bäume und Wälder

Definition 5.1 (Wald, Baum (ungerichteter Fall))

Ein ungerichteter Graph $G = (V, E, \gamma)$ heißt **Wald**, wenn $V \neq \emptyset$ und G keinen einfachen Kreis besitzt. Falls G zusätzlich zusammenhängend ist, so heißt G **Baum**.

Im gerichteten Fall muß man zur Definition von Bäumen und Wäldern den „Umweg“ über *Ketten* und *Zykel* machen.

Definition 5.2 (Kette, Zykel)

Ist $G = (V, R, \alpha, \omega)$ ein gerichteter Graph, so heißt

$$\kappa = (\delta_1 r_1, \dots, \delta_k r_k)$$

eine **Kette**, falls gilt:

1. $r_j \in R$ für $j = 1, \dots, k$
2. $\delta_j \in \{-1, +1\}$ für $j = 1, \dots, k$
3. Mit

$$\alpha(\delta_j r_j) := \begin{cases} \alpha(r_j) & \text{falls } \delta_j = 1 \\ \omega(r_j) & \text{falls } \delta_j = -1 \end{cases} \quad \text{und} \quad \omega(\delta_j r_j) := \begin{cases} \omega(r_j) & \text{falls } \delta_j = 1 \\ \alpha(r_j) & \text{falls } \delta_j = -1 \end{cases}$$

gilt $\omega(\delta_j r_j) = \alpha(\delta_{j+1} r_{j+1})$ für $j = 1, \dots, k-1$.

Wir nennen $\alpha(\kappa) := \alpha(\delta_1 r_1)$ die **Anfangsecke** und $\omega(\kappa) := \omega(\delta_k r_k)$ die **Endecke** von κ . Falls $\alpha(\kappa) = \omega(\kappa)$, so heißt κ **Zykel**.

Die **Spur** der Kette κ ist $s(\kappa) := (\alpha(\delta_1 r_1), \dots, \alpha(\delta_k r_k), \omega(\delta_k r_k))$. Die Kette κ nennen wir **einfach**, wenn $r_i \neq r_j$ für $i \neq j$. Ferner nennen wir κ **elementar**, wenn $\alpha(\delta_j r_j) \neq \alpha(\delta_l r_l)$ und $\omega(\delta_j r_j) \neq \omega(\delta_l r_l)$ für $j \neq l$ gilt.

Definition 5.3 (Wald, Baum (gerichteter Fall))

Ein gerichteter Graph $G = (V, R, \alpha, \omega)$ heißt **Wald**, wenn $V \neq \emptyset$ und G keinen einfachen Zykel besitzt. Falls G zusätzlich noch schwach zusammenhängend ist, so heißt G **Baum**.

Bemerkung: Wir sagen oft, ein Graph $G = (V, R, \alpha, \omega)$ sei *zyklenfrei*, und meinen damit, daß G keinen *einfachen* Zykel besitzt.

Definition 5.4 (Spannender Baum, spannender Wald)

Sei G ein (gerichteter oder ungerichteter) Graph. Ein Partialgraph H von G ist ein spannender Baum, wenn H ein Baum ist. Ein Partialgraph H von G ist ein spannender Wald, wenn jede schwache Zusammenhangskomponente von H ein spannender Baum einer schwachen Zusammenhangskomponente von G ist.

Lemma 5.5 Ein gerichteter Graph $G = (V, R, \alpha, \omega)$ ist genau dann schwach zusammenhängend, wenn es für jedes Paar von Ecken $u \neq v$ eine einfache Kette κ in G gibt mit $\alpha(\kappa) = u$ und $\omega(\kappa) = v$.

Beweisskizze: Nach Definition ist G genau dann schwach zusammenhängend, wenn G^{sym} stark zusammenhängend ist.

Man benutze nun die Äquivalenz: G^{sym} enthält einen Pfeil $r \iff G$ enthält einen Pfeil r' mit $\{\alpha(r'), \omega(r')\} = \{\alpha(r), \omega(r)\}$. \square

Ausführlicher Beweis von Lemma 5.5:

„ \Rightarrow “

Ist G schwach zusammenhängend, so ist G^{sym} stark zusammenhängend. Folglich existiert zu $u \neq v$ ein o.B.d.A. elementarer Weg $w = (r_1, \dots, r_k)$ in G^{sym} mit $\alpha(w) = u$ und $\omega(w) = v$. Falls $r_i \in R$, so wähle $r'_i := r_i$ und $\delta_i := 1$. Ansonsten existiert $r'_i \in R$ mit $\alpha(r'_i) = \omega(r_i)$ und $\omega(r'_i) = \alpha(r_i)$. In diesem Fall setzen wir $\delta_i := -1$. Die Kette $\kappa := (\delta_1 r'_1, \dots, \delta_k r'_k)$ besitzt die gewünschten Eigenschaften.

„ \Leftarrow “

Sei $\kappa = (\delta_1 r_1, \dots, \delta_k r_k)$ eine Kette mit $\alpha(\kappa) = u$ und $\omega(\kappa) = v$. Wähle $r'_i \in G^{\text{sym}}$ mit

$$r'_i := \begin{cases} r_i & \text{falls } \delta_i = 1 \\ r_i^{-1} & \text{falls } \delta_i = -1. \end{cases}$$

Dann ist $w = (r'_1, \dots, r'_k)$ ein Weg von u nach v in G^{sym} . Analog findet man einen Weg von v nach u . Somit ist $u \sim v$. Da u, v beliebig, ist G^{sym} stark zusammenhängend. \square

Lemma 5.6 Sei $G = (V, R, \alpha, \omega)$ ein endlicher schwach zusammenhängender Graph. Dann gilt $|R| \geq |V| - 1$.

Beweis: Induktion nach $n := |V|$.

Die Aussage ist richtig für $n = 1$. Sei sie bereits für alle $k < n$ bewiesen und $G = (V, R, \alpha, \omega)$ schwach zusammenhängend mit $|V| = n$.

Da G schwach zusammenhängend ist, folgt $g(v) \geq 1$ für alle $v \in V$. Wenn $g(v) \geq 2$ für alle $v \in V$, so gilt

$$|R| = \frac{1}{2} \sum_{v \in V} g(v) \geq \frac{1}{2} \cdot 2 \cdot |V| = |V|.$$

Also gilt insbesondere $|R| \geq |V|$. Ansonsten sei $g(v) = 1$ mit o.B.d.A. $g^+(v) = 0$. Der Graph $G' := G[V \setminus \{v\}]$ ist dann schwach zusammenhängend und besitzt nach Induktionsvoraussetzung mindestens $(|V| - 1) - 1$ Pfeile. Zählen wir den einen mit v inzidenten Pfeil hinzu, so erhalten wir, daß G mindestens $|V| - 1$ Pfeile enthält. \square

Satz 5.7 (Äquivalente Definitionen für Bäume) Sei $G = (V, R, \alpha, \omega)$ ein endlicher Graph. Dann sind folgende Aussagen äquivalent:

1. G ist ein Baum.
2. G enthält keinen einfachen Zykel, aber jeder echte Obergraph von G mit gleicher Eckenmenge besitzt einen einfachen Zykel.
3. Für jedes Paar von Ecken $u \neq v$ existiert genau eine einfache Kette κ in G mit $\alpha(\kappa) = u$ und $\omega(\kappa) = v$.
4. G ist schwach zusammenhängend und für jeden Pfeil $r \in R$ ist der durch $R \setminus \{r\}$ induzierte Partialgraph nicht schwach zusammenhängend.
5. G ist schwach zusammenhängend und $|R| = |V| - 1$.
6. G besitzt keinen einfachen Zykel und $|R| = |V| - 1$.

Beweis:

1 \Rightarrow 2 Sei $G' = (V, R', \alpha', \omega')$ echter Obergraph von G und $r \in R' \setminus R$.

Da G schwach zusammenhängend ist, existiert nach Lemma 5.5 eine einfache Kette $\kappa = (\delta_1 r_1, \dots, \delta_k r_k)$ mit $\alpha(\kappa) = \omega(r)$ und $\omega(\kappa) = \alpha(r)$. Dann ist $(\delta_1 r_1, \dots, \delta_k r_k, r)$ ein einfacher Zykel in G' .

2 \Rightarrow 3 Seien $u \neq v$ Ecken. Sei r ein neuer Pfeil mit $\alpha(r) := u$ und $\omega(r) := v$.

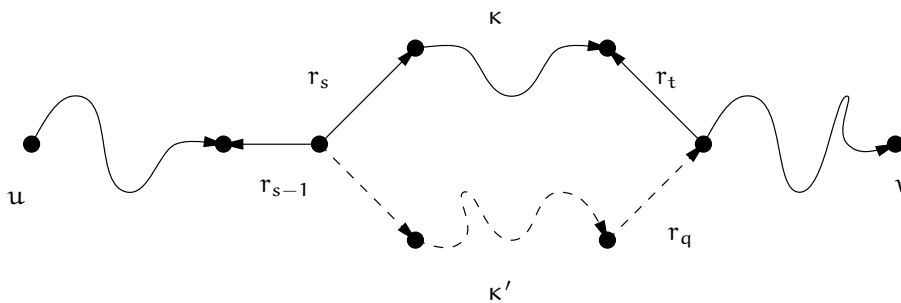
Nach Voraussetzung besitzt $G' := (V, R \cup \{r\}, \alpha, \omega)$ einen einfachen Zykel κ . Da G zyklentfrei, ist $r \in \kappa$, o.B.d.A. $\kappa = (\delta_1 r_1, \dots, \delta_k r_k, -r)$.

Dann ist $(\delta_1 r_1, \dots, \delta_k r_k)$ eine einfache Kette von u nach v .

Annahme: Die Ketten $\kappa = (\delta_1 r_1, \dots, \delta_k r_k)$ und $\kappa' = (\delta'_1 r'_1, \dots, \delta'_p r'_p)$ erfüllen beide $\alpha(\kappa) = \alpha(\kappa') = u$ und $\omega(\kappa) = \omega(\kappa') = v$.

Wähle s minimal, so daß $\delta_s r_s \neq \delta'_s r'_s$ (siehe Bild 5.1). Es muß $r_s \neq r'_s$ gelten, denn sonst besäße G eine Schlinge.

Abbildung 5.1: Konstruktion eines einfachen Zyklens aus den zwei Ketten κ und κ' .



Wähle nun $t \geq s$ und $q \geq s$ minimal, so daß $\omega(\delta_t r_t) = \omega(\delta'_q r'_q)$. Dann ist $(\delta_s r_s, \dots, \delta_t r_t, -\delta'_q r'_q, \dots, -\delta'_s r'_s)$ ein einfacher Zykel in G . Widerspruch!

3 \Rightarrow **4** Nach Lemma 5.5 ist G schwach zusammenhängend.

Wäre $G_{R \setminus \{r\}}$ noch schwach zusammenhängend, so gäbe es eine einfache Kette κ in $G_{R \setminus \{r\}}$ von $\alpha(r)$ nach $\omega(r)$. Dann sind aber κ und $\kappa' := (+r)$ zwei verschiedene einfache Ketten in G von $\alpha(r)$ nach $\omega(r)$. Widerspruch!

4 \Rightarrow **5** Nach Lemma 5.6 gilt $|R| \geq |V| - 1$. Wir zeigen nun $|R| \leq |V| - 1$ durch Induktion nach $n := |V|$.

Die Aussage ist offenbar richtig für $n = 1$.

Sei sie bereits für alle $1 \leq k < n$ bewiesen.

Wähle $r \in R$ beliebig und betrachte $G' := G_{R \setminus \{r\}}$. Nach Voraussetzung besitzt G' die schwachen Zusammenhangskomponenten G_1, \dots, G_p mit $p > 1$.

Nach Induktionsvoraussetzung gilt mit $G_i = (V_i, R_i)$: $|R_i| \leq |V_i| - 1$ für $i = 1, \dots, p$. Wegen $R = \{r\} \cup \bigcup_{i=1}^p R_i$ und $V = \bigcup_{i=1}^p V_i$ folgt durch Summation: $|R| \leq |V| - p + 1 \leq |V| - 1$ (wegen $p > 1$).

5 \Rightarrow **6** Annahme: $\kappa = (\delta_1 r_1, \dots, \delta_k r_k)$ sei ein einfacher Zykel.

Dann ist $G = (V, R \setminus \{r_j\}, \alpha, \omega)$ immer noch schwach zusammenhängend und besitzt $|V| - 2$ Pfeile im Widerspruch zu Lemma 5.6.

6 \Rightarrow **1** Es besitze G die schwachen Zusammenhangskomponenten G_1, \dots, G_p mit $G_i = (V_i, R_i)$.

Jede Komponente ist dann schwach zusammenhängend und zyklensfrei, also ein Baum (Eigenschaft 1). Nach dem bisher Bewiesenen ($1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 4 \Rightarrow 5$) gilt $|R_i| = |V_i| - 1$ für $i = 1, \dots, p$. Somit folgt

$$|V| - 1 = |R| = \sum_{i=1}^p |R_i| = |V| - p,$$

also $p = 1$. Folglich ist G schwach zusammenhängend. \square

Für ungerichtete Graphen zeigt man analog den folgenden Satz, den man aus Satz 5.7 erhält, wenn man die Begriffe für gerichtete Graphen (Zykel, Kette, schwach zusammenhängend) durch die entsprechenden Begriffe für ungerichtete Graphen (Kreis, Weg, zusammenhängend) ersetzt:

Satz 5.8 (Äquivalente Definitionen für Bäume) Sei $G = (V, E, \gamma)$ ein endlicher ungerichteter Graph. Dann sind folgende Aussagen äquivalent:

1. G ist ein Baum.
2. G enthält keinen einfachen Kreis, aber jeder echte Obergraph von G mit gleicher Eckenmenge besitzt einen einfachen Kreis.
3. Für jedes Paar von Ecken $u \neq v$ existiert genau ein einfacher Weg w in G mit $\alpha(w) = u$ und $\omega(w) = v$.
4. G ist zusammenhängend und für jede Kante $e \in E$ ist der durch $E \setminus \{e\}$ induzierte Partialgraph nicht zusammenhängend.
5. G ist zusammenhängend und $|E| = |V| - 1$.
6. G besitzt keinen einfachen Kreis und $|E| = |V| - 1$.

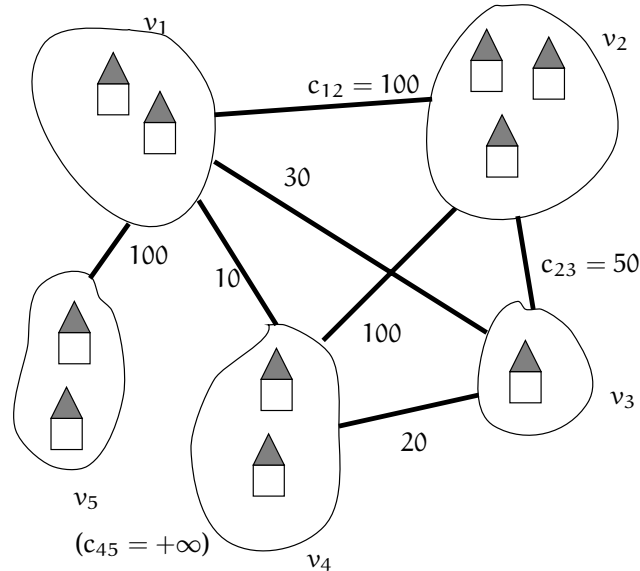
5.2 Minimale spannende Bäume

Ein Netzwerkdesignproblem

Gegeben sei die Situation von Abbildung 5.2:

1. Orte v_1, \dots, v_n , die durch ein Leitungsnetz verbunden werden sollen.
2. Mögliche Verbindungen der Form (v_i, v_j) mit Baukosten c_{ij} .

Abbildung 5.2:
Ein kostengünstigstes Netzwerk soll gebaut werden.



Gesucht: Ein kostengünstigstes Leitungsnetz, welches alle Orte miteinander verbindet.

Graphentheoretische Formulierung des Problems

Gegeben: Ein (ungerichteter) Graph $G = (V, E)$ und eine Kantenbewertungsfunktion $c: E \rightarrow \mathbb{R}$.

Gesucht: Ein spannender Baum $T = (V, E_T)$ von G mit minimalen Kosten (minimalem Gewicht) $c(T) := \sum_{e \in E_T} c(e)$.

Im folgenden sei $G = (V, R, \alpha, \omega)$ ein endlicher gerichteter Graph und $c: R \rightarrow \mathbb{R}$ eine Pfeilbewertungsfunktion. Da ein Wald nie inverse Pfeile oder Schlingen enthalten kann, werden wir uns auf den Fall beschränken, daß G *einfach* ist, also $G = (V, R)$.

Ziel: Bestimmung eines minimalen spannenden Baumes (Waldes) von G

Wir werden in diesem Kapitel mehrere Algorithmen zur Bestimmung eines minimalen spannenden Baumes kennenlernen.

Notationen:

- MST = „minimal spannender Baum“ (Minimum Spanning Tree)
- MSF = „minimal spannender Wald“ (Minimum Spanning Forest)

Der Algorithmus von Kruskal

Der Kruskal-Algorithmus (Algorithmus 5.1) benutzt folgende einfache Strategie: Starte mit einer leeren Pfeilmenge R_T . Füge dann iterativ den leichtesten Pfeil zu R_T hinzu, der keinen Zykel induziert.

Algorithmus 5.1 Algorithmus von Kruskal zur Konstruktion eines MSF.

Input: Ein Graph $G = (V, R)$ mit $n := |V|$ Ecken und $m := |R|$ Pfeilen, eine Pfeilbewertungsfunktion $c: R \rightarrow \mathbb{R}$

```

1 Sortiere die Pfeile nach ihrem Gewicht:  $c(r_1) \leq \dots \leq c(r_m)$ 
2  $R_T \leftarrow \emptyset$ 
3 for  $i \leftarrow 1, \dots, m$  do
4   if  $(V, R_T \cup \{r_i\})$  ist zyklensfrei then
5      $R_T \leftarrow R_T \cup \{r_i\}$ 
6   end if
7 end for
8 return  $R_T$ 

```

Falls G schwach zusammenhängend ist, so liefert der Kruskal-Algorithmus einen spannenden Baum, sonst einen spannenden Wald. Zur Korrektheit des Kruskal-Algorithmus werden wir ein allgemeineres Resultat zeigen.

Matroide und Unabhängigkeitssysteme

Definition 5.9 (Unabhängigkeitssystem, Matroid)

Ein **Unabhängigkeitssystem** ist ein Paar (S, \mathcal{F}) , wobei S eine endliche Menge und $\mathcal{F} \subseteq 2^S$ unter Inklusion abgeschlossen ist, d.h.

$$A \in \mathcal{F} \wedge B \subseteq A \Rightarrow B \in \mathcal{F}. \quad (5.1)$$

Die Mengen in \mathcal{F} nennen wir **unabhängige Mengen**.

Eine Menge $M \in \mathcal{F}$ heißt **maximal bezüglich** $A \subseteq S$, wenn $M \subseteq A$ und aus $M' \in \mathcal{F}$ und $M \subseteq M' \subseteq A$ folgt, daß $M = M'$ gilt. Eine **maximale unabhängige Menge** ist eine bezüglich S maximale Menge $M \in \mathcal{F}$.

Ein Unabhängigkeitssystem heißt **Matroid**, wenn gilt:

$$A, B \in \mathcal{F} \wedge |B| < |A| \Rightarrow \exists a \in A \setminus B : B \cup \{a\} \in \mathcal{F}. \quad (5.2)$$

Kurzschreibweisen:

$$A + x := A \cup \{x\}$$

$$A - x := A \setminus \{x\}$$

Lemma 5.10 Sei $\mathcal{U} = (S, \mathcal{F})$ ein Unabhängigkeitssystem. Dann ist \mathcal{U} genau dann ein Matroid, wenn gilt:

$$\text{Für jedes } A \subseteq S \text{ gilt: } M, M' \text{ maximal bzgl. } A \Rightarrow |M| = |M'|. \quad (5.3)$$

Beweis:

„ \Rightarrow “

Wäre $|M| < |M'|$, so folgt aus (5.2), daß $M + e \in \mathcal{F}$ für ein $e \in M' \setminus M$ im Widerspruch zur Maximalität von M .

„ \Leftarrow “

Seien $M, M' \in \mathcal{F}$ mit $|M| < |M'|$. Dann ist M nicht maximal bezüglich $A := M \cup M'$ (es gibt eine bezüglich A maximale Menge M'' , die M' enthält. Diese erfüllt $|M''| \geq |M'| > |M|$. Da nach Voraussetzung alle bezüglich A maximalen Mengen die gleiche Kardinalität besitzen und $|M| < |M''|$ ist, kann M nicht maximal sein). Also existiert ein $e \in A \setminus M = M' \setminus M$ mit $M + e \in \mathcal{F}$ (Die Menge M ist wieder in einer maximalen Menge M''' bezüglich A enthalten. Jedes Element aus M''' kann zu M hinzugefügt werden, ohne die Unabhängigkeit von M zu verletzen. Da M''' nur Elemente aus $M \cup M'$ enthält, folgt die letzte Aussage). \square

Beispiele für Matroide und Unabhängigkeitssysteme

Beispiel 5.11 Sei $E = \{1, 3, 5, 9, 11\}$ und $\mathcal{F} := \{A \subseteq E : \sum_{e \in A} e \leq 20\}$. Wir zeigen, daß (E, \mathcal{F}) zwar ein Unabhängigkeitssystem ist, aber kein Matroid.

Sei dazu $A \in \mathcal{F}$ und $B \subseteq A$. Wir müssen zeigen, daß auch $B \in \mathcal{F}$ gilt.

Es gilt $\sum_{e \in B} e \leq \sum_{e \in A} e \leq 20$, wobei die letzte Ungleichung aus der Definition von \mathcal{F} und $A \in \mathcal{F}$ folgt. Also ist auch $B \in \mathcal{F}$. Somit haben wir gezeigt, daß (E, \mathcal{F}) ein Unabhängigkeitssystem ist.

(E, \mathcal{F}) ist jedoch kein Matroid, denn $B := \{9, 11\}$ und $A := \{1, 3, 5, 9\}$ sind unabhängige Mengen mit $|B| < |A|$. Jedoch kann man zu B kein Element a aus $A \setminus B$ hinzufügen, so daß noch $B + a \in \mathcal{F}$ gilt.

Satz 5.12 Sei $G = (V, R, \alpha, \omega)$ ein endlicher Graph und

$$\mathcal{F} := \{R' \subseteq R : G_{R'} \text{ besitzt keinen einfachen Zykel}\}.$$

Dann ist (R, \mathcal{F}) ein Matroid. Die maximalen unabhängigen Mengen sind die spannenden Wälder von G .

Beweis: Besitzt $G_{R'}$ keinen einfachen Zykel, so ist auch $G_{R''}$ für alle $R'' \subseteq R'$ zyklensfrei. Somit ist (R, \mathcal{F}) ein Unabhängigkeitssystem.

Wir zeigen nun, daß die Bedingung (5.3) aus Lemma 5.10 gilt.

Sei $A \subseteq R$ und $R' \in \mathcal{F}$ maximal bezüglich A . Der Graph G_A besitze die p schwachen Zusammenhangskomponenten ZK_1, \dots, ZK_p .

Die Restriktion $G'[ZK_j]$ von $G' := G_{R'}$ auf ZK_j ist zyklensfrei. Da R' maximal ist, muß $G'[ZK_j]$ zusammenhängend sein (sonst könnte man R' vergrößern, ohne Zyklen zu induzieren). Nach Definition 5.4 ist damit $G'[ZK_j]$ ein spannender Baum von ZK_j , also ist G' ein spannender Wald von G . Es folgt nun mit Satz 5.7, daß $|R'| = |V| - p$.

Somit besitzen alle bezüglich A maximalen Mengen die Kardinalität $|V| - p$. \square

Der Greedy-Algorithmus für Matroide

Wir kommen nun zum Greedy-Algorithmus für Matroide, der in Algorithmus 5.2 notiert ist.

Man sieht, daß der Kruskal-Algorithmus ein Spezialfall von Algorithmus 5.2 ist, wobei der zugrundeliegende Matroid in Satz 5.12 definiert ist.

Algorithmus 5.2 Greedy-Algorithmus.

Input: Ein Matroid (S, \mathcal{F}) mit $m := |S|$, eine Gewichtsfunktion $c: S \rightarrow \mathbb{R}$

```

1 Sortiere die Elemente aus  $S$  nach ihrem Gewicht:  $c(e_1) \leq \dots \leq c(e_m)$ 
2  $M \leftarrow \emptyset$ 
3 for  $i \leftarrow 1, \dots, m$  do
4   if  $M + e_i \in \mathcal{F}$  then
5      $M \leftarrow M + e_i$ 
6   end if
7 end for
8 return  $M$ 

```

Satz 5.13 (Edmonds) *Der Greedy-Algorithmus findet eine maximale unabhängige Menge mit minimalem Gewicht.*

Beweis: Sei M_k die Menge M nach dem Hinzufügen des k ten Elements, d.h. $|M_k| = k$. Da S endlich, bricht der Greedy-Algorithmus nach endlich vielen Schritten mit einer Menge $M_{k'} \in \mathcal{F}$ ab.

Diese Menge ist maximal. Wäre nämlich $M_{k'} \subsetneq A$ mit $M_{k'} \neq A$, so existiert ein $e \in A \setminus M_{k'}$ mit $M_{k'} + e \in \mathcal{F}$. In einem Testschritt in Zeile 4 ist dann e abgelehnt worden, weil für die zu diesem Zeitpunkt aktuelle Menge M galt: $M_k + e \notin \mathcal{F}$. Da $M_k + e \subseteq M_{k'} + e$ folgt aber $M_k + e \in \mathcal{F}$. Widerspruch.

Wir zeigen nun durch Induktion nach k , daß gilt:

$$c(M_k) = \min\{c(I) : I \in \mathcal{F} \wedge |I| = k\}. \quad (5.4)$$

Dies zeigt dann, daß der Greedy-Algorithmus eine maximale Menge mit minimalem Gewicht liefert.

Für $k = 0$ ist die Aussage offenbar richtig. Sei sie bereits für k bewiesen und wir betrachten $M_{k+1} = M_k + e^*$.

Annahme: $I \in \mathcal{F}$ erfüllt: $|I| = k + 1$ und $c(I) < c(M_{k+1})$.

Wähle $e' \in I$ mit $c(e') = \max\{c(e) : e \in I\}$. Nach Induktionsvoraussetzung gilt $c(M_k) \leq c(I - e')$. Wegen

$$c(M_{k+1}) = c(M_k) + c(e^*) > c(I) = c(I - e') + c(e')$$

folgt:

$$c(e^*) > c(e') = \max\{c(e) : e \in I\}. \quad (5.5)$$

Da $|M_k| = k$ und $|I| = k + 1$ existiert ein $e \in I$ mit $M_k + e \in \mathcal{F}$. Wegen (5.5) gilt $c(M_k + e) < c(M_{k+1})$.

Da $c(e) < c(e^*)$ und $M_k + e \in \mathcal{F}$, muß e in einem früheren Testschritt in Zeile 4 wegen $M_l + e \notin \mathcal{F}$ ($l < k$) abgelehnt worden sein. Da $M_l + e \subseteq M_k + e$, folgt $M_l + e \in \mathcal{F}$. Widerspruch! \square

Korollar 5.14 *Der Kruskal-Algorithmus findet einen spannenden Wald mit minimalem Gewicht.*

Beweis: Unmittelbar aus Satz 5.12 und Satz 5.13. \square

Das Ergebnis aus Satz 5.13 besitzt eine interessante Umkehrung:

Satz 5.15 Sei $\mathcal{U} = (S, \mathcal{F})$ ein Unabhängigkeitssystem. Falls der Greedy-Algorithmus für jede Gewichtsfunktion $c: S \rightarrow \mathbb{R}$ eine maximale unabhängige Menge $M^* \in \mathcal{F}$ mit minimalem Gewicht $c(M^*)$ findet, dann ist \mathcal{U} ein Matroid.

Beweis: Siehe Übung 5.3. □

Laufzeit des Kruskal-Algorithmus

Die Laufzeit des Kruskal-Algorithmus hängt stark von der Effizienz des Zyklentests in Schritt 4 ab. Im folgenden skizzieren wir, wie man den Algorithmus effizient implementieren kann.

Wann erzeugt die Hinzunahme von (u, v) zu T einen einfachen Zykel? Dies ist offenbar genau dann der Fall, wenn u und v bereits in der gleichen Zusammenhangskomponente von T liegen. Diese Beobachtung benutzt die Implementierung, welche in Algorithmus 5.3 gezeigt ist.

Algorithmus 5.3 Implementierung des Algorithmus von Kruskal.

Input: Ein Graph $G = (V, R)$ mit $n := |V|$ Ecken und $m := |R|$ Pfeilen, eine Pfeilbewertungsfunktion $c: R \rightarrow \mathbb{R}$

- 1 Sortiere die Pfeile nach ihrem Gewicht: $c(r_1) \leq \dots \leq c(r_m)$
- 2 $R_T \leftarrow \emptyset$
- 3 **for all** $u \in V$ **do**
- 4 $V_u \leftarrow \text{MAKE-SET}(u)$
- 5 **end for**
- 6 **for** $i \leftarrow 1, \dots, m$ **do**
- 7 Finde die Endpunkte u und v von r_i
- 8 Finde die Komponenten $V_u = \text{FIND-SET}(u)$ und $V_v = \text{FIND-SET}(v)$, die u bzw. v enthalten.
- 9 **if** $V_u \neq V_v$ **then**
- 10 $\text{UNION}(V_u, V_v)$ und $R_T \leftarrow R_T \cup \{r_i\}$
- 11 **end if**
- 12 **end for**
- 13 **return** R_T

Man benutzt nun schnelle UNION-FIND Datenstrukturen (siehe z.B. [CLR90, Tar83, Kapitel 22]). Eine Sequenz von $m + n$ MAKE-SET, UNION und FIND-SET Operationen, von denen n MAKE-SET Operationen sind, kann man damit in $\mathcal{O}(m\alpha(m, n))$ Zeit ausführen, wobei α die inverse Ackermann-Funktion ist.

Folge: Der Kruskal-Algorithmus läuft in Zeit $\mathcal{O}(m \log m)$.

Dabei ist die große Ackermann-Funktion \mathcal{A} für natürliche Zahlen $i, j \geq 1$ wie folgt definiert:

$$\begin{aligned} \mathcal{A}(1, j) &= 2^j && \text{falls } j \geq 1, \\ \mathcal{A}(i, 1) &= \mathcal{A}(i-1, 2) && \text{falls } i \geq 2, \\ \mathcal{A}(i, j) &= \mathcal{A}(i-1, \mathcal{A}(i, j-1)) && \text{falls } i, j \geq 2, \end{aligned}$$

Die *inverse Ackerman-Funktion* ist durch

$$\alpha(m, n) = \min \{ i \geq 1 : \mathcal{A}(i, \lfloor m/n \rfloor) > \log_2 n \}$$

gegeben. Sie ist keine inverse Funktion im eigentlichen mathematischen Sinne, stellt aber gewissermaßen das inverse Wachstum der Ackermann-Funktion \mathcal{A} dar: Die Funktion α wächst so langsam, wie \mathcal{A} schnell wächst.

Man sieht leicht, daß die Ackermann-Funktion \mathcal{A} in jedem Argument monoton wachsend ist. Da

$$A(4, 1) = A(3, 2) = 2^{2^{\dots^2}} \} 16 \text{ mal}$$

weitaus größer als die geschätzte Anzahl der Atome im Universum (etwa 10^{80}) ist, gilt $\alpha(m, n) \leq 4$ für alle „in der Praxis vorkommenden Werte“, d.h. für $n \leq 10^{80}$.

Der Algorithmus von Prim

Wir formulieren den Algorithmus von Prim für *ungerichtete* zusammenhängende Graphen. Im folgenden sei daher $G = (V, E)$ ein ungerichteter Graph in Adjazenzlistendarstellung (dabei „erscheint“ jede ungerichtete Kante (u, v) zweimal, einmal via $v \in \text{Adj}[u]$ und einmal via $u \in \text{Adj}[v]$).

Wie der Algorithmus von Kruskal startet der Algorithmus von Prim mit einer leeren Kantenmenge E_T . Der Algorithmus besitzt die Eigenschaft, daß die Kantenmenge E_T zu jedem Zeitpunkt zusammenhängend ist. In jedem Schritt fügt er die jeweils billigste Kante hinzu, die E_T mit dem Restgraphen verbindet.

Definition 5.16 (Sichere Kante)

Sei $E' \subseteq E$ eine Teilmenge der Kanten von G mit folgender Eigenschaft:

$$\text{Es gibt einen MST } T^* = (V, E_{T^*}) \text{ von } G \text{ mit } E' \subseteq E_{T^*}. \quad (5.6)$$

Eine Kante $e \in E$ heißt **sicher für** E' , wenn $E' \cup \{e\}$ ebenfalls die Eigenschaft (5.6) besitzt.

Definition 5.17 (Schnitt)

Sei $G = (V, E, \gamma)$ ein ungerichteter Graph und $A \cup B = V$ eine Partition von V . Dann nennen wir

$$\sigma(A, B) := \{e \in E : \gamma(e) = \{a, b\} \text{ mit } a \in A \text{ und } b \in B\}$$

den **von A und B erzeugten Schnitt**.

Falls $G = (V, R, \alpha, \omega)$ ein gerichteter Graph und $A \cup B = V$ eine Partition von V ist, so ist

$$\sigma(A, B) := \{r \in R : (\alpha(r) \in A \wedge \omega(r) \in B) \vee (\alpha(r) \in B \wedge \omega(r) \in A)\}$$

den **von A und B erzeugte Schnitt**.

Oftmals nennen wir auch (A, B) einen Schnitt und meinen damit eigentlich $\sigma(A, B)$.

Satz 5.18 Sei $G = (V, E)$ ein zusammenhängender ungerichteter Graph und $c: E \rightarrow \mathbb{R}$ eine Gewichtsfunktion. Sei $E' \subseteq E$ mit der Eigenschaft (5.6). Sei (A, B) ein Schnitt von V mit der Eigenschaft: $\sigma(A, B) \cap E' = \emptyset$. Sei nun e eine leichteste Kante aus $\sigma(A, B)$. Dann ist e sicher für E' .

Beweis: Sei T^* ein MST mit $E' \subseteq T^*$. Ist $e \in T^*$, so ist nichts zu zeigen.

Ansonsten erzeugt die Hinzunahme von e zu T^* einen einfachen Kreis $w = (e_1, \dots, e_k)$ mit o.B.d.A. $e_1 = e$. Der Kreis w muß eine Kante $e_i \in \sigma(A, B)$ mit $e_i \neq e$ enthalten. Nach Voraussetzung gilt $c(e_i) \geq c(e)$. Dann erfüllt $T := T^* \setminus \{e_i\} \cup \{e\}$:

$$c(T) \leq c(T^*).$$

Ferner ist T zusammenhängend und besitzt $|V| - 1$ Kanten, ist also nach Satz 5.8 ein Baum. Somit ist T ein MST, der alle Kanten aus $E' \cup \{e\}$ enthält. \square

Algorithmus 5.4 Algorithmus von Prim.

Input: Ein zusammenhängender ungerichteter Graph $G = (V, E)$ mit $n := |V|$ und $m := |E|$, eine Kantenbewertungsfunktion $c: E \rightarrow \mathbb{R}$ und eine Startecke $s \in V$

```

1  $E_T \leftarrow \emptyset$ 
2 for all  $v \in V$  do
3    $d[v] \leftarrow +\infty$ 
4 end for
5  $Q \leftarrow \emptyset$  {erzeuge eine leere Prioritätsschlange  $Q$ }
6  $d[s] \leftarrow 0$ 
7 INSERT( $Q, s$ ) {füge  $s$  mit Schlüsselwert  $d[s] = 0$  in die Schlange ein}
8 while  $Q \neq \emptyset$  do
9    $u \leftarrow$  EXTRACT-MIN( $Q$ )
10   $d[u] \leftarrow -\infty$ 
11  if  $u \neq s$  then
12     $E_T \leftarrow E_T \cup \{e[u]\}$ 
13  end if
14  for all  $v \in$  Adj[ $u$ ] do
15    if  $c(u, v) < d[v]$  then
16      if  $d[v] = +\infty$  then
17         $d[v] \leftarrow c(u, v)$ 
18        INSERT( $Q, v$ )
19      else
20        DECREASE-KEY( $Q, v, c(u, v)$ )
21      end if
22       $e[v] \leftarrow (u, v)$ 
23    end if
24  end for
25 end while
26 return  $E_T$ 

```

Die Korrektheit des Algorithmus von Prim folgt aus Satz 5.18.

Laufzeit

Falls man die Prioritätsschlange Q als binären Heap verwaltet, so benötigen INSERT, EXTRACT-MIN und DECREASE-KEY jeweils $\mathcal{O}(\log |Q|)$ Zeit. Somit ist die Gesamtlaufzeit des Algorithmus von Prim $\mathcal{O}(m \log n)$.

Eine Verbesserung der Laufzeit ist durch *Fibonacci-Heaps* (siehe z.B. [CLR90, Kapitel 21]) möglich:

Satz 5.19 *Beginnt man mit einem leeren Fibonacci-Heap Q und führt dann eine beliebige Folge von INSERT, EXTRACT-MIN und DECREASE-KEY Operationen aus, so ist die dafür benötigte gesamte Zeit höchstens die Summe der amortisierten Kosten der einzelnen Operationen. Dabei sind die amortisierten Kosten für jede EXTRACT-MIN Operation $\mathcal{O}(\log|Q|)$ und $\mathcal{O}(1)$ für alle anderen Operationen.*

Aus Satz 5.19 folgt, daß man den Algorithmus von Prim mit Hilfe von Fibonacci-Heaps so implementieren kann, daß er in $\mathcal{O}(n \log n + m)$ Zeit läuft.

Der Algorithmus von Fredman und Tarjan

Der Algorithmus von Fredman und Tarjan baut auf dem Algorithmus von Prim auf. Er läuft in Zeit $\mathcal{O}(n + m\beta(m, n))$, wobei

$$\beta(m, n) := \min\{i : \log^{(i)} n \leq m/n\}. \quad (5.7)$$

Die Funktion β wächst extrem langsam! Es gilt $\beta(m, n) \leq \log^* n$, wobei

$$\log^* n := \min\{i : \log^{(i)} n \leq 1\}. \quad (5.8)$$

Man beachte, daß $\log^* 16 = 3$, $\log^* 65536 = 4$, $\log^* 2^{65536} = 5$. Zur Erinnerung: Die geschätzte Zahl der Atome im Universum ist etwa 10^{80} !

Die Idee des Algorithmus von Fredman und Tarjan ist es, die Prioritätsschlange Q in ihrer Größe geschickt zu beschränken (denn EXTRACT-MIN benötigt $\mathcal{O}(\log|Q|)$ amortisierte Zeit).

Grundidee des Algorithmus

1. Lasse einen einzelnen Baum T wie im Algorithmus von Prim wachsen, bis die Schlange Q , welche die „Nachbarecken“ zu T enthält, eine gewisse Größe überschreitet.
2. Starte dann von einer neuen Ecke und stoppe wieder, falls Q zu groß wird.
3. Die ersten Schritte werden ausgeführt, bis jede Ecke in einem Baum enthalten ist. Dann wird jeder Baum zu einer „Superecke“ kontrahiert und der Algorithmus fährt mit dem geschrumpften Graphen fort.
4. Nach einer genügenden Anzahl von Durchläufen bleibt nur noch eine Superecke übrig. Expandieren liefert dann den MST.

Der Algorithmus

Die Implementierung führt das Kontrahieren implizit aus. In jedem Durchlauf beginnt man mit einem Wald von bisher gewachsenen alten Bäumen. Im Durchlauf verbindet man dann die Bäume zu neuen Bäumen, die dann die alten Bäume für den nächsten Durchlauf werden.

Start eines Durchgangs

1. Numeriere die alten Bäume und gib jeder Ecke die Nummer seines Baumes. Damit kann man für jede Ecke v den Baum, dem sie zugehört, direkt aus $\text{tree}[v]$ ablesen. Aufwand für diesen Schritt: $\mathcal{O}(n + m)$.
2. Aufräumen: Lösche aus dem Graphen alle Kanten, die zwei Ecken im gleichen Baum verbinden. Behalte auch nur die jeweils billigsten Kanten zwischen verschiedenen Bäumen.
Das Aufräumen kann in $\mathcal{O}(n + m)$ Zeit erfolgen: Sortiere die Kanten lexikographisch nach den Nummern ihrer Endpunkte mittels zweier Durchgänge von Counting-Sort (siehe z.B. [CLR90, Kapitel 9]). Danach laufe die sortierte Liste einmal von vorne nach hinten durch.
3. Nach dem Aufräumen erstellt man für jeden alten Baum T eine Liste mit den Kanten, die einen Endpunkt in T haben.
4. Jeder alte Baum T erhält den Schlüssel $d[T] := +\infty$. Seine Markierung wird gelöscht.

Wachsen eines neuen Baumes

1. Wähle irgendeinen unmarkierten alten Baum T_0 und füge ihn in Q mit Schlüssel $d[T_0] = -\infty$ ein.
2. Wiederhole den folgenden Schritt, bis Q leer ist oder $|Q| > 2^{2m/t}$, wobei t die Anzahl der alten Bäume zu Beginn des Durchgangs ist.
 - (a) Lösche einen alten Baum T mit minimalem Schlüssel aus Q und setze $d[T] := -\infty$
 - (b) Wenn $T \neq T_0$, dann füge $e[T]$ zum Wald hinzu ($e[T]$ verbindet den alten Baum T mit dem aktuellen Baum, der T_0 enthält).
 - (c) Wenn T markiert ist, dann stoppe und beende den Wachstumsschritt wie unten geschildert.
 - (d) Sonst, markiere T . Für jede Kante (u, v) mit $u \in T$ und $c(u, v) < d[\text{tree}[v]]$ setze $e[\text{tree}[v]] := (u, v)$. Wenn $d[\text{tree}[v]] = +\infty$, dann füge $\text{tree}[v]$ in Q mit Schlüssel $c(u, v)$ ein. Ansonsten erniedrige den Schlüsselwert von T in Q auf $c(u, v)$.
3. Zum Beenden des Wachstumsschrittes leere Q und setze $d[T] := +\infty$ für jeden alten Baum T mit endlichem Schlüsselwert (diese sind die Bäume, die während des Durchgangs in Q eingefügt wurden).

Laufzeit

Die Zeit für Aufräumen und Initialisieren ist $\mathcal{O}(m)$. Sei t die Anzahl der alten Bäume, dann ist die Zeit für den Wachstumsschritt

$$\mathcal{O}(t \log 2^{2m/t} + m) = \mathcal{O}(m), \quad (5.9)$$

denn wir benötigen höchstens t EXTRACT-MIN Operationen auf einem Heap der Größe höchstens $2^{2m/t}$ und $\mathcal{O}(m)$ andere Heap-Operationen, von denen jede nur $\mathcal{O}(1)$ amortisierte Zeit benötigt.

Insgesamt: Ein Durchgang benötigt $\mathcal{O}(m)$ Zeit.

Es bleibt die Frage, wie viele Durchgänge notwendig sind. Seien zu Beginn eines Durchganges t alte Bäume und $m' \leq m$ Kanten vorhanden (einige Kanten sind möglicherweise gelöscht worden). Nach dem Durchgang besitzt jeder Baum T , der am Ende des Durchgangs übrig bleibt, mehr als $2^{2^{m/t}}$ Kanten, die *mindestens einen* Endpunkt in T haben (Wenn T_0 der erste Baum war, aus dem T entstanden ist, dann wuchs T_0 , bis daß der Heap die Größe $2^{2^{m/t}}$ überschritt. Zu diesem Zeitpunkt besaß der aktuelle Baum T' mehr als $2^{2^{m/t}}$ inzidente Kanten. Nachher sind möglicherweise noch weitere Bäume mit T' verbunden worden, was zur Folge hatte, daß jetzt von diesen inzidenten Kanten einige *beide* Endpunkte im Endbaum T besitzen).

Da jede der m' Kanten nur zwei Endpunkte besitzt, erfüllt die Anzahl t' der Bäume nach Ende des Durchlaufs

$$t' \leq \frac{2m'}{2^{2^{m/t}}}. \quad (5.10)$$

Die Schranke für die Heap-Größe im nächsten Durchlauf ist dann

$$2^{2^{m/t'}} \geq 2^{2^{2^{m/t}}}. \quad (5.11)$$

Da die Startschranke für die Heap-Größe $2m/n$ ist und eine Heap-Größe von n nur im letzten Durchgang möglich ist, haben wir höchstens

$$\min\{i : \log^{(i)} n \leq m/n\} + 1 = \beta(m, n) + \mathcal{O}(1)$$

Durchläufe.

Abschließende Bemerkungen zu den MST-Algorithmen

Wir haben in diesem Kapitel die folgenden MST-Algorithmen kennengelernt:

Algorithmus	Laufzeit	Bemerkungen
Kruskal	$\mathcal{O}(m \log m)$ $\mathcal{O}(m\alpha(m, n))$	im „Normalfall“ falls die Kanten bereits nach Gewicht sortiert sind oder sich in $\mathcal{O}(m)$ Zeit sortieren lassen (z.B. mit Counting-Sort oder Radix-Sort, siehe z.B. [CLR90, Kapitel 9])
Prim	$\mathcal{O}(m \log m)$ $\mathcal{O}(n \log n + m)$	mit binären Heaps mit Fibonacci-Heaps. Schnellster Algorithmus, falls G „dicht“ ist, d.h. $m \in \Omega(n \log n)$
Fredman & Tarjan	$\mathcal{O}(m\beta(m, n))$	Extrem schnell für „dünne“ Graphen

Da die Algorithmen für beliebige Gewichtsfunktionen $c: \mathbb{R} \rightarrow \mathbb{R}$ bzw. $c: E \rightarrow \mathbb{R}$ einen MST finden, können wir mit ihnen auch spannende Bäume mit *maximalem Gewicht* bestimmen.

5.3 Steinerbäume und andere Dilemmas

Eine Erweiterung des Problems, einen MST zu finden, ist das Steinerbaum-Problem. Im Gegensatz zu einem spannenden Baum muß ein Steinerbaum nur eine Teilmenge der Ecken aufspannen.

Definition 5.20 (Steinerbaum)

Sei G ein (gerichteter oder ungerichteter) Graph mit Eckenmenge V und $K \subseteq V$ eine beliebige Teilmenge der Eckenmenge. Ein **Steinerbaum** in G für die Menge K ist ein Teilgraph $T = (V_T, E_T)$ von G , der ein Baum ist und dessen Eckenmenge K umfaßt: $K \subseteq V_T$.

Die Elemente von K nennt man **Terminals**, die Ecken aus $V_T \setminus K$ **Steinerpunkte**.

Folge: Jeder spannende Baum von G ist Steinerbaum für $K := V$.

Definition 5.21 (STEINERBAUM, MINSTEINERBAUM)

Eine Instanz von STEINERBAUM ist durch einen (gerichteten oder ungerichteten) Graphen $G = (V, R)$ mit einer Bewertungsfunktion $c: R \rightarrow \mathbb{R}$, eine Teilmenge $K \subseteq V$ und eine natürliche Zahl k gegeben. Das Problem ist zu entscheiden, ob G einen Steinerbaum T mit Kosten $c(T) \leq k$ besitzt.

Das Problem MINSTEINERBAUM besteht darin, einen Steinerbaum mit minimalem Gewicht zu bestimmen.

Satz 5.22 (Karp 1972) STEINERBAUM ist NP-vollständig. Dies gilt sogar, wenn $c(r) = 1$ für alle $r \in R$ ist.

Beweis: Siehe auch [GJ79, Problem ND12] □

Eine weitere Variante des MST-Problems ist es, einen spannenden Baum zu finden, in dem der maximale Grad beschränkt ist.

Definition 5.23 (GRADBESCHRÄNKTER BAUM)

Eine Instanz von GRADBESCHRÄNKTER BAUM ist durch einen Graphen $G = (V, R)$ und eine natürliche Zahl D gegeben. Das Problem ist zu entscheiden, ob G einen spannenden Baum besitzt, in dem jede Ecke Grad höchstens D besitzt.

Satz 5.24 GRADBESCHRÄNKTER BAUM ist NP-vollständig.

Beweis: Reduktion von HAMILTONSCHER KREIS. Siehe z.B. [GJ79, Problem ND1] □

5.4 Spannende Wurzelbäume in gerichteten Graphen

Definition 5.25 (Wurzel, Wurzelbaum)

Sei $G = (V, R, \alpha, \omega)$ ein gerichteter Graph. Die Ecke $s \in V$ heißt **Wurzel** von G , wenn $E_G(s) = V$, d.h. wenn alle Ecken $v \in V$ von s aus erreichbar sind.

Ein **Wurzelbaum mit Wurzel s** (kürzer auch **s -Wurzelbaum**) ist ein Baum G , der eine Wurzel $s \in V$ besitzt. Die Ecken $v \in V$ mit $g^+(v) = 0$ nennt man die **Blätter** des Wurzelbaumes.

Ist $v \in V$, so heißt jedes $u \in V$ auf dem eindeutigen Weg von der Wurzel s zu v ein **Vorfahre** von v . Wenn u Vorfahre von v ist, so ist v ein **Nachkomme** von u .

Wir betrachten nun das Problem, bei gegebenem gerichteten Graphen $G = (V, R, \alpha, \omega)$ mit Pfeilbewertung $c: R \rightarrow \mathbb{R}$ und ausgezeichnete Ecke $s \in V$ einen s -Wurzelbaum minimalen Gewichts zu finden. Dieses Problem läßt sich durch den Algorithmus von Edmonds (Algorithmus 5.5) in Polynomialzeit lösen.

Algorithmus 5.5 Algorithmus von Edmonds.

Input: Graph $G = (V, R, \alpha, \omega)$,
Pfeilbewertung $c: R \rightarrow \mathbb{R}$,
Wurzel $s \in V$

- 1 wähle aus jedem Parallelenbüschel einen leichtesten Pfeil, entferne die übrigen
- 2 entferne alle Schlingen und die Pfeile aus $B^-(s)$
- 3 **for** $v \in V, v \neq s$ **do**
- 4 Abbruch mit Fehler, falls $B^-(v) = \emptyset$
- 5 $c_0(v) \leftarrow \min_{r \in B^-(v)} c(r)$
- 6 **end for**
- 7 **for** alle verbliebenen Pfeile $r \in R$ **do**
- 8 $c'(r) := c(r) - c_0(\omega(r))$
- 9 **end for**
- 10 $R' \leftarrow \emptyset$
- 11 **for** $v \in V, v \neq s$ **do**
- 12 wähle einen Pfeil $r \in B^-(v)$ mit $c'(r) = 0$
- 13 $R' \leftarrow R' \cup \{r\}$
- 14 **end for**
- 15 **if** $G' = (V, R')$ ist kein Baum **then**
- 16 ermittle Kreis k in G' ,
- 17 sei \tilde{V} die Menge der Ecken von k
- 17 kontrahiere Ecken in \tilde{V} zu einer Superecke \tilde{v}
- 18 rufe Algorithmus rekursiv für den kontrahierten Graphen auf
- 18 sei (V, \tilde{R}) die Lösung auf dem kontrahierten Graphen
- 19 setze $R' \leftarrow \tilde{R}$
- 20 ersetze in R' den eingehenden Pfeil (v_1, \tilde{v}) durch einen leichtesten Pfeil $r_1 \in R$ mit $\alpha(r_1) = v_1$ und $\omega(r_1) \in \tilde{V}$
- 21 ersetze in R' jeden ausgehenden Pfeil (\tilde{v}, v_p) durch einen leichtesten Pfeil $r_p \in R$ mit $\alpha(r_p) \in \tilde{V}$ und $\omega(r_p) = v_p$
- 22 füge zu R' alle Pfeile aus k mit Ausnahme des Pfeils mit Endecke $\omega(r_1)$ hinzu
- 23 **end if**

Lemma 5.26 Sei $G = (V, R, \alpha, \omega)$ ein endlicher Graph, $v \in V$ eine ausgezeichnete Ecke. Dann sind äquivalent:

- (i) G ist ein s -Wurzelbaum.
- (ii) $E_G(s) = V$ und G ist ein Baum.
- (iii) G ist ein Baum und $g^-(s) = 0$ und $g^-(v) = 1$ für alle $v \neq s$

(iv) $g^-(s) = 0$ und $g^-(v) = 1$ für alle $v \neq s$ und $E_G(s) = V$

Beweis:

„(i) \iff (ii)“ Gilt nach Definition.

„(ii) \implies (iii)“ Sei G ein Baum, $E_G(s) = V$. Wegen $E_G(s) = V$ gilt $g^-(v) \geq 1$ für alle $v \in V \setminus \{s\}$ und damit

$$|V| - 1 = |R| = g^-(s) + \sum_{v \in V \setminus \{s\}} g^-(v) \geq g^-(s) + |V| - 1. \quad (5.12)$$

Daraus folgt $g^-(s) = 0$ und schließlich $g^-(v) = 1$ für alle $v \in V \setminus \{s\}$.

„(iii) \implies (iv)“ Sei G ein Baum mit $g^-(s) = 0$ und $g^-(v) = 1$ für alle $v \neq s$. Sei $v \in V \setminus \{s\}$ beliebig. Da G ein Baum ist, gibt es eine Kette in G zwischen s und v . Wegen der Gradbedingungen ist diese Kette ein Weg von s nach v , also ist $v \in E_G(s)$.

„(iv) \implies (ii)“ Sei $E_G(s) = V$, $g^-(s) = 0$ und $g^-(v) = 1$ für alle $v \neq s$. Es ist $|R| = \sum_{v \in V} g^-(v) = |V| - 1$. Wäre G kein Baum, so enthielte G einen Zykel. Sei K die Menge der Ecken des Zyklus. Wegen $g^-(v) \leq 1$ ist dieser Zykel ein Kreis, und es gibt keinen Pfeil r mit $\alpha(r) \notin K$ und $\omega(r) \in K$. Wegen $g^-(s) = 0$ kann der Kreis nicht die Ecke s enthalten. Somit gilt $K \not\subseteq E_G(s)$. Widerspruch. \square

Satz 5.27 (Korrektheit des Algorithmus von Edmonds) Algorithmus 5.5 ermittelt bei Eingabe eines Graphen $G = (V, R, \alpha, \omega)$ mit Pfeilbewertung $c: R \rightarrow \mathbb{R}_0^+$ und Wurzel $s \in V$ einen s -Wurzelbaum mit geringstem Gewicht unter allen s -Wurzelbäumen oder gibt die Information, daß in G kein s -Wurzelbaum existiert.

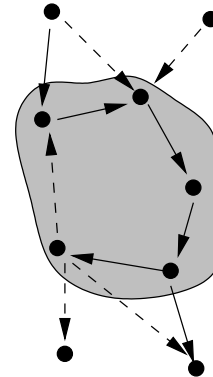
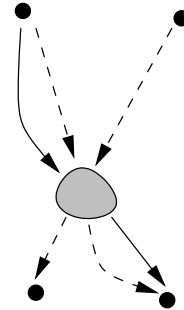
Beweis: Da ein Wurzelbaum keine Parallelen oder Schlingen enthält, beeinflußt das Entfernen von teureren Parallelen und von Schlingen sicher nicht das Gewicht eines Wurzelbaumes. In jedem s -Wurzelbaum gilt nach Lemma 5.26 $g^-(s) = 0$, also kann auch das Pfeilbüschel $B^-(s)$ o.B.d.A. entfernt werden.

Da $g^-(v) > 0$ für alle $v \in V$, $v \neq s$, nach Lemma 5.26 notwendig für die Existenz eines s -Wurzelbaumes ist, erfolgt in Zeile 4 der Abbruch, falls diese Bedingung nicht erfüllt ist.

Jeder s -Wurzelbaum T enthält für jede Ecke $v \neq s$ genau einen Pfeil aus dem eingehenden Pfeilbüschel $B^-(v)$. Bezeichnen wir diesen Pfeil mit r_v , so gilt

$$\begin{aligned} c(T) &= \sum_{v \neq s} c(r_v) = \sum_{v \neq s} (c(r_v) - c_0(v)) + \sum_{v \neq s} c_0(v) = \\ &= \sum_{v \neq s} c'(r_v) + \sum_{v \neq s} c_0(v) = \\ &= c'(T) + \sum_{v \neq s} c_0(v). \end{aligned}$$

Die Summe $\sum_{v \neq s} c_0(v)$ ist eine Konstante für den Graphen, daher ist ein s -Wurzelbaum, der minimal bezüglich c ist, auch minimal bezüglich c' und umgekehrt.



Konstruktion eines Wurzelbaumes im Ausgangsgraphen (unten) aus dem optimalen Wurzelbaum im kontrahierten Graphen (oben). Durchgezogene Pfeile sind jeweils Teil des Baumes, die übrigen Pfeile des Graphen sind gestrichelt. Vergleiche Algorithmus 5.5, Zeilen 20ff.

Offenbar gilt $c' \geq 0$. Ferner gibt es für jede Ecke $v \neq s$ einen Pfeil $r_v \in B^-(v)$ mit $c'(r_v) = 0$. Damit hat jeder c' -minimale Wurzelbaum Gesamtgewicht 0.

In den Zeilen 11 wird eine Menge R' von $|V| - 1$ vielen Pfeilen vom Gesamtgewicht 0 ausgewählt. Bildet (V, R') einen Baum, dann wegen der Gradbedingungen und Lemma 5.26 auch einen s -Wurzelbaum, und sein Gewicht ist minimal.

Andernfalls besteht ein Zykel, der wegen der Gradeigenschaften ein Kreis sein muß; sei mit Z die Menge der Ecken des Kreises bezeichnet. Nach dem rekursiven Aufruf gilt: Der gefundene Graph mit Pfeilmengen \tilde{R} ist ein minimaler s -Wurzelbaum auf dem reduzierten Graphen. Durch die Pfeilersetzungen wird die Baumeigenschaft beibehalten und $E_{R'}(s) = V$ sichergestellt, also ist der entstehende Baum ein s -Wurzelbaum.

Es bleibt zu zeigen, daß der Wurzelbaum $T = (V, R')$ gewichtsminimal ist. Betrachte dazu einen beliebigen gewichtsminimalen s -Wurzelbaum T^* . Durch Kontraktion entsteht der Graph T^*/Z . Kreise oder Schlingen in T^*/Z beginnen und enden in Knoten aus Z . Da T^* optimal war und alle Ecken aus Z auf einem Kreis aus Pfeilen vom Gewicht 0 liegen, haben alle Kreise/Schlingen Gewicht 0 und bestehen wegen $c' \geq 0$ nur aus Pfeilen vom Gewicht 0. Wenn aus jedem Kreis/Schlinge jeweils der Pfeil entfernt wird, dessen Endecke in Z liegt, dann ergibt sich ein s -Wurzelbaum auf G/Z . Wäre dieser nicht gewichtsminimal, so könnte durch die Konstruktion in den Zeilen 20ff. ein Wurzelbaum in G konstruiert werden, der leichter als T^* ist, im Widerspruch zur Wahl von T^* . – Da bei der Konstruktion von T in den Zeilen 20ff. nur Pfeile vom Gewicht 0 zu einem auf G/Z optimalen Wurzelbaum hinzugefügt werden, folgt $c(T) = c(T^*)$ und damit ist der gefundene Baum gewichtsminimal. \square

Das vorgestellte Verfahren hat sicher polynomielle Laufzeit: In jeder Iteration ist der Aufwand für das Entfernen von Parallelen, für die Berechnung von c' und für das Ermitteln von R' jeweils in $O(|R|)$. Der Test, ob ein Baum vorliegt, ist mittels DFS ebenfalls in $O(|R|)$ Aufwand durchzuführen. Mit gleichem Aufwand ist die Eckenkontraktion und die abschließende dazu inverse Operation zu implementieren. Da bei jedem rekursiven Aufruf die Anzahl der Ecken strikt abnimmt, bleibt die Tiefe der Rekursion in $O(|V|)$. Damit ist der gesamte Zeitaufwand in $O(|V| \cdot |R|)$.

Übungsaufgaben

Aufgabe 5.1 – Bäume und unizyklische Graphen

Im folgenden sei $G = (V, E, \gamma)$ ein ungerichteter Graph. Eine Kante $e \in E$ heißt *Brücke*, wenn $k(G - e) > k(G)$ gilt (vgl. Übung 3.3). Wir nennen einen zusammenhängenden Graphen G *unizyklisch*, wenn G genau einen elementaren Kreis enthält. Beweisen Sie, daß für endliche Graphen die folgenden Aussagen äquivalent sind:

1. G ist unizyklisch.
2. Für eine geeignete Kante $e \in E$ ist $G - e$ ein Baum.
3. G ist zusammenhängend mit $|V| = |E|$.

4. G ist zusammenhängend und die Menge aller Kanten von G , die keine Brücken sind, bildet einen elementaren Kreis.

Aufgabe 5.2 – Minimale Spannende Bäume

Im folgenden sei $G = (V, R, \alpha, \omega)$ ein endlicher schwach zusammenhängender Graph und $c: R \rightarrow \mathbb{R}$ eine Gewichtsfunktion.

- (a) Sei $A \cup B = V$ eine Partition von V und $\sigma(A, B)$ der zugehörige Schnitt. Sei $r^* \in \sigma(A, B)$ ein Pfeil mit $c(r^*) = \min\{c(r) : r \in \sigma(A, B)\}$. Beweisen oder widerlegen Sie, daß es dann einen minimalen spannenden Baum T^* von G mit $r^* \in T^*$ gibt.
- (b) Sei wiederum $\sigma(A, B)$ ein Schnitt und T_A bzw. T_B minimale spannende Bäume von $G[A]$ bzw. $G[B]$. Beweisen oder widerlegen Sie, daß es einen minimalen spannenden Baum T^* von G mit $T_A \subseteq T^*$ und $T_B \subseteq T^*$ gibt.

Aufgabe 5.3 – Matroide und der Greedy-Algorithmus

Sei $\mathcal{U} = (S, \mathcal{F})$ ein Unabhängigkeitssystem und $c: S \rightarrow \mathbb{R}$ eine Gewichtsfunktion.

In Satz 5.13 wurde gezeigt, daß der Greedy-Algorithmus immer eine maximale unabhängige Menge M^* mit minimalem Gewicht $c(M^*)$ findet, falls \mathcal{U} nicht nur ein Unabhängigkeitssystem sondern sogar ein Matroid ist. Beweisen Sie nun Umkehrung dieses Ergebnisses aus Satz 5.15.

Aufgabe 5.4 – Schnelle MST- und Komponenten-Berechnungen

Sei im folgenden wieder $G = (V, R, \alpha, \omega)$ ein endlicher schwach zusammenhängender Graph mit $n := |V|$ Ecken und $m := |R|$ Pfeilen.

- (a) Zeigen Sie, daß man die schwachen Zusammenhangskomponenten von G in $\mathcal{O}(n + m\alpha(m, n))$ Zeit berechnen kann.
- (b) Sei $c: R \rightarrow \{1, \dots, n\}$ eine Pfeilgewichtungsfunktion. Geben Sie einen Algorithmus an, der einen MST von G bezüglich c in Zeit $\mathcal{O}(n + m\alpha(m, n))$ berechnet.

Aufgabe 5.5 – Radius und Diameter

Sei $G = (V, R, \alpha, \omega)$ ein gerichteter Graph, und d_G die Distanzfunktion auf G . Wir bezeichnen mit

$$e(v) := \sup_{u \in V} d_G(u, v)$$

die *Exzentrizität* der Ecke v im Graphen. Damit sind durch

$$r(G) := \inf_{v \in V} e(v) \quad \text{und} \quad d(G) := \sup_{v \in V} e(v)$$

der *Radius* und der *Diameter* des Graphen definiert. Wie üblich werden die Definitionen über die symmetrische Hülle auch auf ungerichtete Graphen ausgedehnt.

- a. Zeigen Sie: Ist G ein ungerichteter zusammenhängender Graph, so gilt

$$r(G) \leq d(G) \leq 2 \cdot r(G).$$

- b. Zeigen Sie: Für beliebige Zahlen $r, d \in \mathbb{N}$ mit $1 \leq r \leq d \leq 2r$ gibt es einen ungerichteten Graphen G mit Einheitsbewertung, der Radius $r(G) = r$ und Diameter $d(G) = d$ aufweist.

Aufgabe 5.6 – Zentrum in Bäumen

Sei $G = (V, R, \alpha, \omega)$ ein Graph mit einer beliebigen Distanzfunktion. Die Menge

$$C(G) := \{v \in V \mid e(v) = r(G)\}$$

wird *Zentrum* des Graphen G genannt.¹

Eine Ecke $v \in V$ eines Baumes heißt *Blatt*, falls $g(v) = 1$, und *innere Ecke*, falls $g(v) > 1$.

- a. Sei G ein gerichteter Baum mit endlichem Radius $r(G) < \infty$. Weisen Sie nach, daß das Zentrum $C(G)$ aus genau einer Ecke besteht.
- b. Sei $T = (V, E)$ ein ungerichteter Baum mit Einheitsbewertung, $|V| \geq 3$. Es sei T' der Baum, der aus T durch Entfernen aller Blätter (und der inzidenten Kanten) entsteht. Zeigen Sie:

$$C(T') = C(T).$$

- c. Zeigen Sie: Das Zentrum eines Baumes mit Einheitsbewertung besteht entweder aus einer Ecke oder aus zwei adjazenten Ecken. Gilt dies auch für allgemeine Bewertungen?

Aufgabe 5.7 – Minimale spannende Bäume (MST)

- a. Für einen spannenden Baum $T = (V, E_T)$ eines Graphen $G = (V, E)$ mit Kantengewichtsfunktion $c: E \rightarrow \mathbb{R}$ sei $L(T)$ die sortierte Liste der Kantengewichte, d. h. $L(T) = (e_1, \dots, e_{n-1})$ mit $c(e_1) \leq \dots \leq c(e_{n-1})$ und $E_T = \{e_1, \dots, e_{n-1}\}$.

Zeigen Sie: Sind T_1 und T_2 MSTs desselben Graphen, so gilt $L(T_1) = L(T_2)$.

Hinweis: Verwenden Sie das Ergebnis aus Aufgabe 5.8.

- b. Folgern Sie: Ist die Kantengewichtsfunktion injektiv, so ist der MST eindeutig bestimmt.
- c. Der »Wegwerf-Algorithmus« zur Bestimmung eines MST startet mit dem zusammenhängenden Ausgangsgraphen. Er identifiziert einen beliebigen Kreis, dort eine schwerste Kante, und entfernt die Kante aus dem Graphen. Die Iteration wird wiederholt, bis der entstehende Graph keinen Kreis mehr enthält.

Zeigen Sie die Korrektheit des Verfahrens.

¹Es bezeichnen $e(v)$ die Exzentrizität einer Ecke und $r(G)$ den Radius des Graphen.

Aufgabe 5.8 – Minimale Flaschenhals-Bäume (MBT)

Sei $G = (V, E, \alpha, \omega)$ ein endlicher Graph, $c: E \rightarrow \mathbb{R}$ eine Gewichtsfunktion. Das *Flaschenhals-Gewicht* (engl. *bottleneck weight*) $c_{\max}(G')$ eines Partialgraphen $G' = (V, E')$ ist definiert durch $c_{\max}(G') := \max\{c(e) \mid e \in E'\}$. Ein minimaler Flaschenhalsbaum (MBT) ist ein spannender Baum von minimalem Flaschenhalsgewicht.

Zeigen Sie: Jeder MST ist ein MBT.

Kapitel 6

Suchstrategien

6.1 Untersuchung von Graphen mit Tiefensuche

In diesem Kapitel werden wir uns mit der sogenannten Tiefensuche oder Depth-First-Search (DFS) beschäftigen. DFS ist ein effizientes Verfahren, um Informationen über einen Graphen, etwa über seine Zusammenhangskomponenten, zu gewinnen.

Grundideen von DFS

Der Graph wird durch DFS nach der Strategie „zunächst in die Tiefe gehen“ erforscht. Pfeile werden ausgehend von der zuletzt entdeckten Ecke v aus erforscht, von der noch unerforschte Pfeile starten. Wenn alle von v ausgehenden Pfeile erforscht sind, dann erfolgt ein „Backtracking“ zu der Ecke, von der aus v entdeckt wurde.

Jede Ecke ist am Anfang weiß, d.h. noch nicht entdeckt. Wenn eine Ecke entdeckt wird, wird sie grau gefärbt. Wenn die Ecke komplett abgearbeitet ist, d.h. wenn alle von ihr ausgehenden Pfeile erforscht sind, wird sie schwarz gefärbt.

Wenn eine Ecke v neu entdeckt wird, während wir die Adjazenzliste von u durchlaufen, so merken wir uns u als den „Vorgänger“ $\pi[v]$ von v .

Algorithmus 6.1 zeigt die DFS-Hauptprozedur, welche die Prozedur DFS-VISIT aus Algorithmus 6.2 benutzt.

Der Vorgängergraph $G_\pi := (V, R_\pi, \alpha|_{R_\pi}, \omega|_{R_\pi})$ bildet einen DFS-Wald, der aus mehreren DFS-Wurzelbäumen besteht.¹ Die Pfeile in R_π nennen wir *Tree Edges*. Falls G keine Parallelen besitzt, so gilt

$$R_\pi := \{ (\pi[v], v) : v \in V \wedge \pi[v] \neq \text{NIL} \} \quad (6.1)$$

DFS versieht die Ecken auch mit Zeitmarken. Jede Ecke $v \in V$ hat zwei Zeitmarken:

1. $d[v]$: Zeitpunkt, zu dem v entdeckt wurde
2. $f[v]$: Zeitpunkt, zu dem die Adjazenzliste von v komplett erforscht wurde, d.h. zu dem alle von v ausgehenden Pfeile erforscht wurden.

¹Diese Aussage werden wir in Satz 6.1 beweisen.

Nach Konstruktion von DFS gilt für jede Ecke $v \in V$:

$$d[v] < f[v]. \quad (6.2)$$

Der DFS-Algorithmus

Algorithmus 6.1 Depth First Search Hauptprozedur.

DFS(G)

Input: Ein Graph $G = (V, R, \alpha, \omega)$ in
Adjazenzlistendarstellung mit $n := |V|$ und $m := |R|$

```

1 for all  $v \in V$  do
2   farbe[ $v$ ]  $\leftarrow$  weiß
3    $\pi[v] \leftarrow$  NIL
4 end for
5 zeit  $\leftarrow$  0
6 for all  $u \in V$  do
7   if farbe[ $u$ ] = weiß then
8     DFS-VISIT( $u$ )
9   end if
10 end for

```

Algorithmus 6.2 Depth First Search.

DFS-VISIT(u)

```

1 farbe[ $u$ ]  $\leftarrow$  grau {Die weiße Ecke  $u$  wurde gerade entdeckt}
2  $d[u] \leftarrow$  zeit
3 zeit  $\leftarrow$  zeit + 1
4 for all  $v \in \text{Adj}[u]$  do {Erforsche Pfeil von  $u$  nach  $v$ }
5   if farbe[ $v$ ] = weiß then
6      $\pi[v] \leftarrow u$ 
7      $R_\pi \leftarrow R_\pi \cup \{r_{uv}\}$ , wobei  $r_{uv}$  der Pfeil von  $u$  nach  $v$  ist, der dem
      Eintrag von  $v \in \text{Adj}[u]$  entspricht.
8     DFS-VISIT( $v$ )
9   end if
10 end for
11 farbe[ $u$ ]  $\leftarrow$  schwarz {von  $u$  gehen keine unerforschten Pfeile mehr
    aus}
12  $f[u] \leftarrow$  zeit
13 zeit  $\leftarrow$  zeit + 1

```

Laufzeit

Die Laufzeit des Algorithmus 6.1 ohne die Zeit für die Aufrufe von DFS-VISIT ist offenbar in $\mathcal{O}(n)$.

Die Prozedur DFS-VISIT wird insgesamt für jede Ecke $v \in V$ genau einmal aufgerufen, da sie nur für weiße Ecken aufgerufen wird und als erstes die übergebene Ecke grau färbt. Während eines Aufrufs von

DFS-VISIT wird die Schleife in den Zeilen 4 bis 10 $|\text{Adj}[u]|$ mal durchlaufen. Es folgt nun, daß die Gesamtzeit für alle DFS-VISIT Aufrufe in

$$\mathcal{O}\left(\sum_{v \in V} |\text{Adj}[v]|\right) = \mathcal{O}\left(\sum_{v \in V} g^+(v)\right) = \mathcal{O}(m)$$

ist. Daher ist die Gesamtzeit für DFS in $\mathcal{O}(n + m)$.

Eigenschaften von DFS

Satz 6.1 *Der Algorithmus DFS werde auf den gerichteten Graphen $G = (V, R, \alpha, \omega)$ angewandt. Dann ist G_π ein Wald. Jede schwache Zusammenhangskomponente von G_π ist ein Wurzelbaum.*

Beweis: Wenn zu R_π ein Pfeil r_{uv} hinzugenommen wird, so war v zu diesem Zeitpunkt weiß, also unentdeckt. Danach wird v durch den Aufruf von DFS-VISIT(v) grau (und danach schwarz) gefärbt. Es kann also kein Pfeil mit Endecke v zu R_π hinzugefügt werden. Somit gilt in G_π :

$$g^-(v) \leq 1 \quad \text{für alle } v \in V. \quad (6.3)$$

Ferner gilt nach Konstruktion von DFS

$$d[\alpha(r)] < d[\omega(r)] \quad \text{für jeden Pfeil } r \in R_\pi. \quad (6.4)$$

Aus (6.3) folgt, daß jeder Zykel in G_π ein Kreis sein müßte. Wegen (6.4) kann jedoch G_π keinen einfachen Kreis enthalten. Somit ist G_π zyklensfrei, also ein Wald.

Sei G' eine beliebige schwache Zusammenhangskomponente von G_π . Dann gibt es wegen der Kreisfreiheit von G' eine Ecke $s \in G'$ mit $g^-(s) = 0$. Sei $v \in G'$ mit $v \neq s$ beliebig. Da G' schwach zusammenhängend ist, gibt es eine Kette κ in G' mit $\alpha(\kappa) = s$ und $\omega(\kappa) = v$. Die Kette κ muß ein Weg von s nach v sein, da sonst entweder $g^-(s) > 0$ oder $g^-(u) > 1$ für ein $u \in G'$ wäre. Somit ist jede Ecke $v \in G'$ von s aus erreichbar.

Insbesondere muß $g^-(v) \geq 1$ für jedes $v \neq s$ gelten. Wegen (6.3) folgt $g^-(v) = 1$ für alle $v \neq s$. Nach Lemma 5.26 ist damit G' ein Wurzelbaum mit Wurzel s . \square

Satz 6.2 (Intervallsatz für DFS) *Der Algorithmus DFS werde auf den gerichteten Graphen $G = (V, R, \alpha, \omega)$ angewandt. Für jedes Paar u, v von Ecken aus V gilt dann bei Abbruch genau eine der folgenden Aussagen:*

1. Die Intervalle $[d[u], f[u]]$ und $[d[v], f[v]]$ sind disjunkt.
2. Das Intervall $[d[u], f[u]]$ ist vollständig in $[d[v], f[v]]$ enthalten. Die Ecke u ist ein Nachkomme von v in einem DFS-Baum.
3. Das Intervall $[d[v], f[v]]$ ist vollständig in $[d[u], f[u]]$ enthalten. Die Ecke v ist ein Nachkomme von u in einem DFS-Baum.

Beweis:

1. Fall: $d[u] < d[v]$

Unterfall (a): $d[v] < f[u]$

Dann wurde v entdeckt, als u grau war. Die Ecke v ist dann Nachkomme von u . Da v später als u entdeckt wurde, werden alle von v ausgehenden Pfeile erforscht, bevor die Suche zu u zurückkehrt. Also gilt $f[v] < f[u]$. Wegen $d[v] < f[v]$ haben wir $d[u] < d[v] < f[v] < f[u]$.

Unterfall (b): $d[v] > f[u]$

Dann gilt: $d[u] < f[u] < d[v] < f[v]$ und die Intervalle sind disjunkt.

2. Fall: $d[u] > d[v]$

Analog zum 1. Fall. □

Korollar 6.3 Die Ecke v ist genau dann ein echter Nachkomme von u im DFS-Wald G_π , wenn $d[u] < d[v] < f[v] < f[u]$.

Beweis: Direkt aus Satz 6.2. □

Lemma 6.4 Seien $u, v \in ZK_i$ in der gleichen starken Zusammenhangskomponente ZK_i von $G = (V, R, \alpha, \omega)$. Jeder Weg von u nach v in G berührt nur Ecken aus ZK_i .

Beweis: Ist x auf einem Weg w von u nach v , so gilt $v \in E_G(x)$. Sei w' der Teilweg von w , der von u nach x führt. Da $u \sim v$, existiert w'' mit $\alpha(w'') = v$ und $\omega(w'') = u$. Dann ist $w'' \circ w'$ ein Weg von v nach x . Also gilt $x \in E_G(v)$ und damit $x \sim v$ (Abbildung 6.1 veranschaulicht den Beweis). □

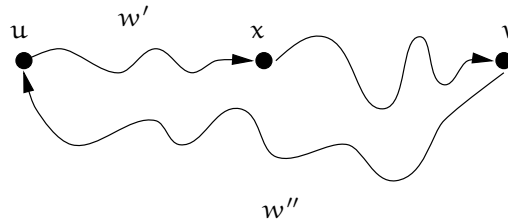


Abbildung 6.1:
Beweis von Lemma 6.4

Satz 6.5 (Satz vom weißen Weg) Die Ecke v ist genau dann ein Nachkomme von u im DFS-Wald G_π , wenn zu dem Zeitpunkt $d[u]$, zu dem u entdeckt wird, die Ecke v von u durch einen Weg erreicht werden kann, der nur weiße Ecken berührt.

Beweis:

„ \Rightarrow “

Sei v Nachkomme von u und x auf dem Weg von u nach v in G_π . Nach Korollar 6.3 gilt $d[u] < d[x]$. Also ist zum Zeitpunkt $d[u]$ die Ecke x noch weiß.

„ \Leftarrow “

Annahme: v ist von u zum Zeitpunkt $d[u]$ durch einen weißen Weg w erreichbar, wird aber kein Nachkomme von u im DFS-Wald.

O.B.d.A. enthält w außer v sonst nur Nachkommen von u (wähle sonst die zu u nächste Ecke auf w , die kein Nachkomme wird). Sei $s(w) = [u = v_0, v_1, \dots, v_k, v]$.

Dann ist v_k ein Nachkomme von u (falls $k = 0$, so gilt $v_k = u$). Nach Korollar 6.3 folgt daher: $f[v_k] \leq f[u]$.

Wir wissen $d[u] < d[v]$, da v zum Zeitpunkt $d[u]$ durch den weißen Weg erreichbar ist (und insbesondere selbst zum Zeitpunkt $d[u]$ weiß ist). Ferner gilt $d[v] < f[v_k]$, denn spätestens beim Durchlauf der Adjazenzliste von v_k wird v entdeckt.

Also ist $d[u] < d[v] < f[v_k] \leq f[u]$. Nach Satz 6.2 ist $[d[v], f[v]]$ dann vollständig in $[d[u], f[u]]$ enthalten. Nach Korollar 6.3 ist dann aber v echter Nachkomme von u . Widerspruch! \square

Satz 6.6 DFS werde auf den Graphen $G = (V, R, \alpha, \omega)$ angewandt. Alle Ecken aus einer Zusammenhangskomponente von G sind dann im gleichen DFS-Wurzelbaum von G_π .

Beweis: Sei u die erste von DFS entdeckte Ecke der Komponente ZK. Dann sind zu diesem Zeitpunkt alle anderen Ecken von ZK noch weiß (weil u die erste entdeckte Ecke ist). Sei v eine beliebige andere Ecke der ZK. Dann gibt es einen Weg von u nach v , welcher nur Ecken aus ZK berührt. Dieser ist zum Zeitpunkt $d[u]$ ein weißer Weg. Nach dem Satz von weißem Weg (Satz 6.5) wird v ein Nachkomme von u in G_π . \square

Klassifizierung der Pfeile durch DFS

Tree Edges Dies sind die Pfeile aus R_π . Ein Pfeil r ist eine Tree Edge, wenn $\omega(r)$ zuerst beim Erforschen von r entdeckt wurde.

Back Edges Pfeile $r \in R$, bei denen $\alpha(r)$ ein Nachkomme von $\omega(r)$ in G_π ist. Schlingen werden als Back Edges angesehen.

Forward Edges Pfeile $r \in R$, bei denen $\omega(r)$ ein Nachfahre von $\alpha(r)$ in G_π ist.

Cross Edges Alle anderen Pfeile.

Man kann DFS so modifizieren, daß der Algorithmus die Pfeile beim Entdecken klassifiziert. Die Idee ist die folgende: Jeder Pfeil r kann durch die Farbe von $\omega(r)$ klassifiziert werden, wenn r (zum ersten Mal) erforscht wird:

$\omega(r)$ **ist weiß:** Dann ist r eine Tree Edge.

Dieser Fall ist klar nach Konstruktion des Algorithmus.

$\omega(r)$ **ist grau:** Der Pfeil r ist eine Back Edge.

Dies sieht man daraus, daß die grauen Ecken zu jedem Zeitpunkt einen Weg von Nachkommen bilden.

$\omega(r)$ **ist schwarz:** r ist eine Forward oder Cross Edge.

Lemma 6.7 Sei r ein Pfeil im Graphen $G = (V, R, \alpha, \omega)$ und $\alpha(r) = u$ und $\omega(r) = v$. Als r das erste Mal erforscht wird, sei v schwarz. Dann ist r eine Forward Edge, wenn $d[u] < d[v]$ und eine Cross Edge, wenn $d[u] > d[v]$.

Beweis: Wenn $d[u] < d[v]$, so gilt $d[u] < d[v] < f[v] < f[u]$, da u noch grau ist, wenn v bereits schwarz geworden ist. Nach dem Intervallsatz ist dann v Nachkomme von u . Somit ist r eine Forward Edge.

Sei nun $d[u] > d[v]$. Wäre r eine Forward Edge, so wäre v Nachkomme von u im DFS-Wald. Dann folgt aber nach dem Intervallsatz, daß $d[u] < d[v] < f[v] < f[u]$ gilt. Dies ist ein Widerspruch zu $d[u] > d[v]$. \square

6.2 Anwendungen von DFS

Test auf Kreise

Satz 6.8 *Ein Graph G ist genau dann kreisfrei, wenn DFS keine Back Edges produziert.*

Beweis:

„ \Rightarrow “

Sei r eine Back Edge. Dann ist $\omega(r)$ ein Vorfahre von $\alpha(r)$ in G_π . Also existiert ein Weg w in G von $\omega(r)$ zu $\alpha(r)$. Dann ist $w \circ (r)$ ein Kreis in G .

„ \Leftarrow “

Sei w ein Kreis in G mit Spur $s(w) = [v_0, v_1, \dots, v_k = v_0]$ und sei v_i die Ecke des Kreises, welche von DFS zuerst entdeckt wird. Zum Zeitpunkt $d[v_i]$ existiert dann ein weißer Weg zu v_{i-1} . Nach dem Satz von weißem Weg (Satz 6.5) wird dann v_{i-1} ein Nachkomme von v_i in G_π . Daher ist der Pfeil von v_{i-1} zu v_i eine Back Edge. \square

Korollar 6.9 *Mit Hilfe von DFS kann man in Zeit $\mathcal{O}(|V| + |R|)$ testen, ob ein gegebener Graph $G = (V, R, \alpha, \omega)$ kreisfrei ist.* \square

Topologische Sortierung

Algorithmus 6.3 Topologische Sortierung.

Input: Ein Graph $G = (V, R, \alpha, \omega)$ in Adjazenzlistendarstellung mit $n := |V|$ und $m := |R|$

- 1 $L \leftarrow \emptyset$ { L ist eine leere lineare Liste}
- 2 Rufe $\text{DFS}(G)$ auf, um die Zeiten $f[v]$ für $v \in V$ zu berechnen.
- 3 Sobald eine Ecke $v \in V$ abgearbeitet ist (d.h. schwarz gefärbt wird), füge v an den Anfang von L an.
- 4 **return** L

Satz 6.10 *Algorithmus 6.3 erzeugt in $\mathcal{O}(n + m)$ Zeit eine topologische Sortierung eines kreisfreien gerichteten Graphen.*

Beweis: Sei G kreisfrei. Es genügt zu zeigen, daß für jeden Pfeil $r \in R$ gilt: $f[\alpha(r)] > f[\omega(r)]$.

Betrachte den Zeitpunkt, zu dem r durch DFS erforscht wird. Zu diesem Zeitpunkt kann $\omega(r)$ nicht grau sein, da r sonst eine Back-Edge wäre (Widerspruch zu Satz 6.8).

Wenn $\omega(r)$ weiß ist, so wird nach dem Satz vom weißen Weg $\omega(r)$ ein Nachfahre von $\alpha(r)$. Nach dem Intervallsatz gilt dann $f[\omega(r)] < f[\alpha(r)]$. Wenn $\omega(r)$ bereits schwarz ist, so ist $\omega(r)$ bereits abgearbeitet, während $\alpha(r)$ noch grau ist. Somit gilt dann ebenfalls $f[\alpha(r)] > f[\omega(r)]$.

Die Laufzeit folgt aus der linearen Laufzeit von DFS und der Tatsache, daß die Listenoperationen in konstanter Zeit durchführbar sind. \square

Bestimmung von starken Zusammenhangskomponenten

Algorithmus 6.4 Berechnung der starken Zusammenhangskomponenten.

- Input:** Ein Graph $G = (V, R, \alpha, \omega)$ in Adjazenzlistendarstellung mit $n := |V|$ und $m := |R|$
- 1 Rufe DFS(G) auf, um die Zeiten $f[v]$ für $v \in V$ zu berechnen.
 - 2 Berechne den inversen Graphen G^{-1} .
 - 3 Rufe DFS(G^{-1}) auf. Dabei betrachte in der Hauptschleife von DFS die Ecken gemäß absteigendem Wert $f[u]$ (wobei $f[u]$ im ersten Schritt berechnet wurde).
 - 4 Seien T_1, \dots, T_p die im letzten Schritt erzeugten DFS-Bäume.
 - 5 Gib jeden DFS-Baum T_i aus dem letzten Schritt als einzelne starke Zusammenhangskomponente aus.
-

Im Rest dieses Abschnittes bezeichnen $d[u]$ und $f[u]$ die Entdeckungs- und Beendigungszeiten für eine Ecke u , die durch DFS(G) im Schritt 1 von Algorithmus 6.4 berechnet wurden.

Definition 6.11 (Ahnherren)

Für eine Ecke $u \in V$ definieren wir den **Ahnherren** $\phi(u)$ von u durch

$$\phi(u) := x \quad \text{wobei } x \in E_G(u) \text{ mit maximalem } f[x]. \quad (6.5)$$

Da u von sich selbst aus erreichbar ist, gilt

$$f[u] \leq f[\phi(u)]. \quad (6.6)$$

Ferner folgt aus $v \in E_G(u)$, daß $f[\phi(v)] \leq f[\phi(u)]$. Insbesondere folgt mit $v = \phi(u)$: $f[\phi(\phi(u))] \leq f[\phi(u)]$. Nach (6.6) gilt andererseits $f[\phi(u)] \leq f[\phi(\phi(u))]$. Also ist $f[\phi(\phi(u))] = f[\phi(u)]$ und daher

$$\phi(\phi(u)) = \phi(u) \quad \text{für alle } u \in V. \quad (6.7)$$

Satz 6.12 Die Ecke $\phi(u)$ ist ein Vorfahre von u in G_π .

Beweis: Falls $\phi(u) = u$, so ist die Aussage trivial. Sei daher $\phi(u) \neq u$.

Betrachte die Farben der Ecken zum Zeitpunkt $d[u]$. Wenn $\phi(u)$ grau ist, so ist $\phi(u)$ ein Vorfahre von u und der Beweis ist vollständig. Wir müssen jetzt nur noch zeigen, daß $\phi(u)$ weder schwarz noch weiß ist.

$\phi(u)$ ist schwarz: Dann ist $f[\phi(u)] < f[u]$ im Widerspruch zu (6.6).

$\phi(u)$ ist weiß: Sei w der Weg von u zu $\phi(u)$.

- 1. Fall: Alle Ecken auf w sind weiß:** Dann wird $\phi(u)$ nach dem Satz vom weißen Weg ein Nachkomme von u in G_π . Dann ist aber nach Satz 6.2 $f[\phi(u)] < f[u]$ im Widerspruch zu (6.6).
- 2. Fall: w enthält graue oder schwarze Ecken:** Sei die Spur von w gegeben durch $s(w) = [u = v_0, v_1, \dots, v_{k-1}, v_k = \phi(u)]$ und t maximal, so daß v_t nicht weiß ist (siehe Abbildung 6.2).

Abbildung 6.2:
Beweis von Satz 6.12



Die Ecke v_t muß grau sein, da sonst der Pfeil von v_t nach v_{t+1} bei der Abarbeitung der Adjazenzliste von v_t nicht untersucht worden wäre.

Nach dem Satz vom weißen Weg wird dann aber $\phi(u)$ Nachkomme von v_t in G_π und es gilt: $f[\phi(u)] < f[v_t]$ im Widerspruch zur Wahl von $\phi(u)$ (Gleichung (6.5)). \square

Korollar 6.13 Die Ecken u und $\phi(u)$ liegen in der gleichen starken Zusammenhangskomponente.

Beweis: Nach Definition ist $\phi(u) \in E_G(u)$. Nach Satz 6.12 ist $\phi(u)$ ein Vorfahre von u in G_π . Also ist auch $u \in E_G(\phi(u))$. \square

Satz 6.14 Es gilt $u \sim v$ genau dann, wenn $\phi(u) = \phi(v)$.

Beweis:

„ \Rightarrow “

Sei $x = \phi(u)$. Dann ist x auch von v erreichbar und somit $f[\phi(v)] \geq f[\phi(u)]$. Die gleiche Argumentation für $y = \phi(v)$ zeigt, daß $f[\phi(u)] \geq f[\phi(v)]$, somit insgesamt also $\phi(u) = \phi(v)$ gilt.

„ \Leftarrow “

Wenn $x = \phi(u) = \phi(v)$, so gilt nach Korollar 6.13 $x \sim u$ und $x \sim v$. Wegen der Transitivität von \sim folgt $v \sim u$. \square

Der Beweis des folgenden Lemmas ist einfach und wird hier deshalb nicht ausgeführt.

Lemma 6.15 Die Graphen G und G^{-1} besitzen die gleichen starken Zusammenhangskomponenten. \square

Satz 6.16 Algorithmus 6.4 findet korrekt in $\mathcal{O}(n + m)$ Zeit die starken Zusammenhangskomponenten eines gerichteten Graphen $G = (V, R, \alpha, \omega)$.

Beweis: Nach Lemma 6.15 besitzen G und G^{-1} die gleichen starken Zusammenhangskomponenten. Nach Satz 6.6, angewandt auf $\text{DFS}(G^{-1})$ in Schritt 3, sind zwei Ecken aus der gleichen Komponente auch im gleichen DFS-Baum enthalten, der in Schritt 5 ausgegeben wird.

Wir müssen jetzt noch zeigen, daß die Bäume T_1, \dots, T_p aus Schritt 4 nicht zu groß sind, d.h. daß $u, v \in T_i$ impliziert, daß $u \sim_G v$.

Seien T_1, \dots, T_p mit $T_i = (V_i, R_i)$ die Bäume, die in Schritt 4 durch $\text{DFS}(G^{-1})$ ausgegeben werden, und seien s_1, \dots, s_p ihre Wurzeln. Definiere für $1 \leq i \leq p$

$$C(s_i) := \{v \in V : \phi(v) = s_i\}.$$

Wir zeigen durch Induktion nach p , daß für $1 \leq i \leq p$ gilt:

- (i) $\phi(s_i) = s_i$
- (ii) $V_i = C(s_i)$

Nach Satz 6.14 sind alle Ecken aus $C(v_p)$ in der gleichen starken Zusammenhangskomponente. Dies zeigt dann, daß jeder Baum eine starke Zusammenhangskomponente bildet und beendet den Beweis.

Induktionsanfang: $p = 1$.

- (i) Klar, da s_1 die Ecke v mit maximalem $f[v]$ ist.
- (ii) Sei $u \in C(s_1)$. Insbesondere ist dann $s_1 \in E_G(u)$, also $u \in E_{G^{-1}}(s_1)$. Da s_1 die erste Ecke ist, die in der Hauptschleife von $\text{DFS}(G^{-1})$ untersucht wird, ist zum Zeitpunkt des Entdeckens von s_1 die Ecke u durch einen weißen Weg (in G^{-1}) erreichbar. Also wird nach dem Satz vom weißen Weg u Nachkomme von s_1 , also insbesondere $u \in T_1$. Somit ist $C(s_1) \subseteq V_1$ gezeigt.

Sei nun $u \in T_1$. Dann ist u von s_1 in G^{-1} erreichbar. Also ist $s_1 \in E_G(u)$ für jedes $u \in T_1$. Insbesondere ist daher $f[\phi(u)] \geq f[s_1]$ für jedes $u \in T_1$. Da s_1 aber als erste Ecke in der Hauptschleife von $\text{DFS}(G^{-1})$ untersucht wurde, besitzt s_1 die größte Abarbeitungszeit $f[s_1]$. Also gilt $f[\phi(u)] = f[s_1]$, also $s_1 = \phi(u)$ für alle $u \in T_1$. Somit ist $V_1 \subseteq C(v_1)$ und insgesamt $C(v_1) = V_1$.

Induktionsschritt: $p - 1 \rightarrow p$.

- (i) Wäre $f[\phi(s_p)] > f[s_p]$, so würde $\phi(s_p)$ beim Algorithmus $\text{DFS}(G^{-1})$ vor s_p betrachtet. Sei T_j der Baum, der $\phi(s_p)$ enthält. Nach Induktionsvoraussetzung ist $V_j = \{v : \phi(v) = s_j\}$. Wegen $\phi(s_p) \in V_j$ und $\phi(\phi(s_p)) = \phi(s_p)$ wäre dann $s_j = \phi(s_p)$ und $s_p \in T_j$.
- (ii) Nach Satz 6.6 landen alle Ecken aus $C(s_p)$ im gleichen DFS-Baum, da sie stark zusammenhängen. Da $s_p \in C(s_p)$ und s_p Wurzel von T_p ist, folgt, daß jede Ecke aus $C(s_p)$ in T_p landet.

Wir zeigen, daß jede Ecke u mit $f[\phi(u)] < f[s_p]$ oder $f[\phi(u)] > f[s_p]$ nicht in T_p plaziert wird.

Sei $f[\phi(u)] > f[s_p]$. Zum Zeitpunkt, zu dem s_p in der Hauptschleife selektiert wird, wurde $\phi(u)$ bereits betrachtet und bereits in einen Baum T_j eingeordnet. Die Eckenmenge dieses Baumes ist dann nach Induktionsvoraussetzung $C(\phi(\phi(u))) = C(\phi(u))$. Nach Induktionsvoraussetzung ist dann $u \in T_j$.

Sei nun $f[\phi(u)] < f[s_p]$. Dann kann u nicht in T_p eingeordnet werden, denn $u \in T_p$ impliziert $s_p \in E_G(u)$ und somit $f[\phi(u)] \geq f[s_p]$.

Dies beendet den Beweis. □

6.3 Tiefensuche für ungerichtete Graphen

Der Algorithmus DFS (Algorithmus 6.1) kann auch für ungerichtete Graphen eingesetzt werden. Im Schritt 7 wird dann nicht der Pfeil $r_{u,v}$ zu R_π hinzugefügt, der von u nach v verläuft, sondern die entsprechende Kante zwischen u und v . Ansonsten bleibt der Algorithmus unverändert.

Ungerichtete Wurzelbäume

Definition 6.17 (Wurzel, Wurzelbaum (ungerichtete Graphen))

Sei $G = (V, E, \gamma)$ ein ungerichteter Graph. Die Ecke $s \in V$ heißt **Wurzel** von G , wenn $E_G(s) = V$, d.h. wenn alle Ecken $v \in V$ von s aus erreichbar sind.

Ein **Wurzelbaum mit Wurzel** s ist ein Baum G , in dem eine Ecke s als Wurzel ausgezeichnet ist. Die Begriffe **Vorfahre** und **Nachkomme** sind dann analog zum gerichteten Fall definiert.

Analog zu Satz 6.1 beweist man den folgenden Satz:

Satz 6.18 Der Algorithmus DFS werde auf den ungerichteten Graphen $G = (V, E, \gamma)$ angewandt. Dann ist G_π ein Wald. Jede Zusammenhangskomponente G' von G_π ist ein ungerichteter Wurzelbaum, wobei die Wurzel von G' diejenige Ecke v mit $\pi[v] = \text{NIL}$ ist.

Mit den Definitionen aus Definition 6.17 bleiben die folgenden Ergebnisse auch für den ungerichteten Fall richtig:

1. Satz 6.2 (Intervallsatz)
2. Korollar 6.3
3. Satz 6.5 (Satz vom weißen Weg)

Die Beweise der oben genannten Ergebnisse sind bereits so formuliert, daß sie *ohne Änderung* auch die Argumentation im ungerichteten Fall ergeben.

Klassifizierung der Kanten

Die Klassifizierung der Kanten nach dem Schema für den gerichteten Fall (man ersetze „Pfeil“ durch „Kante“) ist nicht ganz eindeutig: Eine Kante zwischen dem Vorfahren u und dem Nachkomme v , die nicht zu G_π gehört, könnte man sowohl als Back Edge als auch als Forward Edge ansehen. Wir lösen diese Zweideutigkeiten dadurch auf, daß wir eine Kante als ersten passenden Typ in unserer Liste klassifizieren.

Lemma 6.19 DFS werde auf den ungerichteten Graphen $G = (V, E, \gamma)$ angewandt. Dann ist jede Kante entweder eine Tree Edge oder eine Back Edge.

Beweis: Sei $e \in E$ eine Kante mit $\gamma(e) = \{u, v\}$. Falls $u = v$, so ist e per Definition eine Back Edge. Sei daher $u \neq v$ mit o.B.d.A. $d[u] < d[v]$. Es folgt $d[v] < f[u]$, da $v \in \text{Adj}[u]$. Also ist $d[u] < d[v] < f[u]$ und somit muß nach dem Intervallsatz $d[u] < d[v] < f[v] < f[u]$ gelten. Dann ist v Nachkomme von u in G_π .

Wenn e zuerst von u ausgehend erforscht wird, dann wird e eine Tree Edge. Ansonsten ist e eine Back Edge, da u zu dem Zeitpunkt, zu dem e zum ersten Mal erforscht wird, dann immer noch grau ist. \square

Bestimmung der Zusammenhangskomponenten

Satz 6.20 DFS werde auf den ungerichteten Graphen $G = (V, E, \gamma)$ angewandt. Die Eckenmenge jedes DFS-Wurzelbaums ist dann eine Zusammenhangskomponente von G . Folglich kann man mit DFS die Zusammenhangskomponenten eines ungerichteten Graphen in $\mathcal{O}(n + m)$ Zeit bestimmen.

Beweis: Wie im Beweis von Satz 6.6 zeigt man durch Anwendung des Satzes vom weißen Weg, daß zwei Ecken u und v die zusammenhängen, im gleichen DFS-Baum landen. Sind umgekehrt u und v im gleichen Wurzelbaum, so sind u und v offenbar zusammenhängend, da G_π nur Kanten aus G enthält. \square

6.4 Breitensuche

Unter Breitensuche versteht man eine Strategie zum Durchsuchen eines Graphen, die – im Gegensatz zur Tiefensuche – die Suchrichtung zuerst nach der Breite ausdehnt. Der prototypische Algorithmus soll hier nicht behandelt werden. Im nächsten Kapitel wird die wichtigste Form der Breitensuche, der Algorithmus von Dijkstra, vorgestellt.

Übungsaufgaben

Aufgabe 6.1 – Wurzelbäume in stark zusammenhängenden Graphen

Sei $G = (V, R, \alpha, \omega)$ ein endlicher stark zusammenhängender Graph.

- Sei $s \in V$ eine beliebige Ecke. Beweisen Sie, daß G einen spannenden Wurzelbaum mit Wurzel s besitzt, d.h. einen Wurzelbaum $T = (V, R', \alpha', \omega')$ mit Wurzel s und $T \sqsubseteq G$.
- Sei $G = (V, R)$ ein endlicher stark zusammenhängender Graph. Beweisen Sie, daß es einen irreduziblen Kern G_* von G gibt, dessen Pfeilmenge R_* die Ungleichung $|R_*| \leq 2(|V| - 1)$ erfüllt. (Hinweis: Verwenden Sie Teil (a) und die Tatsache, daß mit G auch G^{-1} stark zusammenhängend ist)

Aufgabe 6.2 – Satz vom grauen Weg

Beweisen Sie, daß bei Ausführung des Algorithmus DFS folgendes gilt: Zum Zeitpunkt $d[v]$ sind die grauen Ecken genau die Vorfahren von v in G_π . Diese bilden einen Weg von der Wurzel s des entsprechenden DFS-Wurzelbaums zur Ecke v .

Aufgabe 6.3 – Eigenschaften der Tiefensuche

Sei $G = (V, R, \alpha, \omega)$ ein endlicher Graph, auf den der Algorithmus DFS aus diesem Kapitel angewandt werde. Beweisen oder widerlegen Sie folgende Aussage: Sei $v \in E_G(u)$ und $d[u] < d[v]$. Dann ist v ein Nachfahre von u im DFS-Wald G_π .

Aufgabe 6.4 – Berechnung des reduzierten Graphen

Geben Sie einen Algorithmus an, der zu einem endlichen gerichteten Graphen $G = (V, R, \alpha, \omega)$ den reduzierten Graphen \hat{G} in Zeit $\mathcal{O}(|V| + |R|)$ berechnet.

Kapitel 7

Bestimmung kürzester Wege

In diesem Kapitel beschäftigen wir uns damit, *kürzeste Wege* in einem gerichteten Graphen zu finden.

7.1 Motivation

Ein Autofahrer möchte den kürzesten Weg von Würzburg nach Hamburg finden. Er hat eine Straßenkarte, auf der die möglichen Strecken und Entfernungen eingezeichnet sind (siehe Abbildung 7.1).

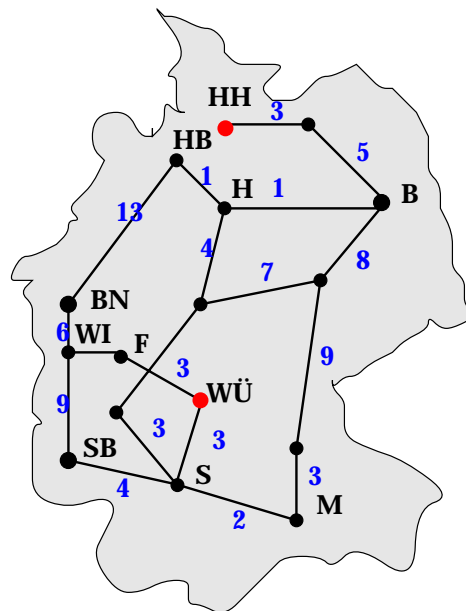
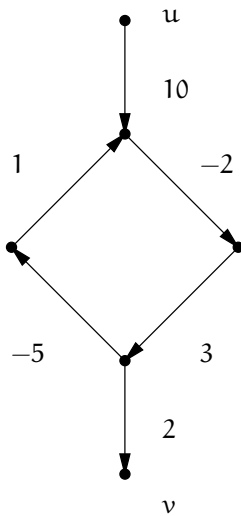


Abbildung 7.1:
Motivation für das Kürzeste-Wege-Problem.

In diesem Kapitel werden wir uns damit beschäftigen, wie „kürzeste-Wege-Probleme“ ähnlich dem des Autofahrers effizient mit graphentheoretischen Methoden gelöst werden können.

7.2 Kürzeste Wege



Für die Ecken u und v gilt:
 $\delta(u, v) = -\infty$.

Als Grundvoraussetzung für den Rest dieses Kapitels sei $G = (V, R)$ ein endlicher gerichteter Graph ohne Parallelen¹ und $c: R \rightarrow \mathbb{R}$ eine Gewichtsfunktion. Wir bezeichnen wie üblich mit $n := |V|$ die Anzahl der Ecken und mit $m := |R|$ die Anzahl der Pfeile von G .

Definition 7.1 (Länge eines Weges in einem gewichteten Graphen, Abstand)

Sei $G = (V, R)$ ein Graph und $c: R \rightarrow \mathbb{R}$ eine Gewichtsfunktion. Die **Länge** $c(w)$ eines Weges $w = (r_1, \dots, r_k)$ in G ist dann $c(w) := \sum_{i=1}^k c(r_i)$.

Für zwei (nicht notwendigerweise verschiedene) Ecken $u, v \in V$ definieren wir mit $W_G(u, v)$ die **Menge aller Wege in G von u nach v** . Der **Abstand** $\delta(u, v)$ ist dann wie folgt definiert:

$$\delta(u, v) := \begin{cases} \inf\{c(w) : w \in W_G(u, v)\} & \text{falls } u \neq v \text{ und } v \in E_G(u) \\ \min(0, \inf\{c(w) : w \in W_G(u, u)\}) & \text{falls } u = v \text{ und } W_G(u, u) \neq \emptyset \\ 0 & \text{falls } u = v \text{ und } W_G(u, u) = \emptyset \\ +\infty & \text{falls } v \notin E_G(u). \end{cases} \quad (7.1)$$

Man beachte, daß wir in obiger Definition das Infimum anstelle eines Minimums benutzen. In der Tat kann $\delta(u, v) = -\infty$ auftreten, wie das nebenstehende Beispiel zeigt.

Lemma 7.2 Die folgenden zwei Aussagen sind äquivalent:

- (i) Es gilt $\delta(u, v) = -\infty$.
- (ii) Es gibt einen Weg w von u nach v und eine Ecke $x \in s(w)$, so daß x auf einem Kreis negativer Länge liegt.

Beweis: „(i) \Rightarrow (ii)“

Annahme: Es gibt keinen Kreis w mit den Eigenschaften aus (ii).

Jeder Weg von u nach v wird dann nicht länger, wenn man Teilwege aus ihm entfernt, die Kreise sind (da jeder solche Kreis eine nicht negative Länge besitzt). Da es nur endlich viele Wege in G gibt, die keinen Kreis enthalten, folgt, daß das Infimum in (7.1) endlich sein muß. Dies widerspricht der Voraussetzung, daß $\delta(u, v) = -\infty$.

„(ii) \Rightarrow (i)“ Trivial. \square

7.3 Kürzeste-Wege-Probleme

Man unterscheidet folgende Typen von Kürzeste-Wege-Problemen

Ein-Paar-kürzeste-Wege-Problem Finde einen kürzesten Weg (die Länge eines kürzesten Weges) von $s \in V$ zu $t \in V$.

Eine-Quelle-kürzeste-Wege-Problem Finde kürzeste Wege (die Längen der kürzesten Wege) von $s \in V$ zu allen $v \in V \setminus \{s\}$.

Alle-Paare-kürzeste-Wege-Problem Bestimme für jedes Paar u, v von Ecken einen kürzesten Weg.

Wir werden uns im folgenden auf das *Eine-Quelle-kürzeste-Wege-Problem* konzentrieren.

¹Parallelen spielen bei der Berechnung kürzester Wege keine Rolle. Wir können jeweils die billigste der Parallelen im Graphen behalten und alle anderen vorab eliminieren.

7.4 Kürzeste-Wege-Bäume

Die Repräsentation von kürzesten Wegen beim Eine-Quelle-kürzeste-Wege-Problem erfolgt durch Bäume kürzester Wege, die wir in der folgenden Definition einführen.

Definition 7.3 (Baum kürzester Wege)

Sei $G = (V, R)$ ein gerichteter Graph, $c: R \rightarrow \mathbb{R}$ und $s \in V$. Ein **Baum kürzester Wege bezüglich s** ist dann ein Wurzelbaum $T = (V', R') \sqsubseteq G$ mit Wurzel s und folgenden Eigenschaften:

- (i) $V' = E_G(s)$
- (ii) Für alle $v \in V' \setminus \{s\}$ ist der eindeutige Weg von s zu v in T ein kürzester Weg in G von s zu v .

Ein Baum kürzester Wege muß nicht eindeutig sein, wie das folgende Beispiel zeigt.

Beispiel 7.4 Betrachte den Graphen $G = (V, R)$ mit $V = \{s, a_1, a_2, v\}$ und $R = \{(s, a_1), (s, a_2), (a_1, v), (a_2, v)\}$. Die Längen auf den Pfeilen seien identisch 1 gewählt. Der Graph G ist in Abbildung 7.2 dargestellt. Im Baum kürzester Wege bezüglich s hat man die Wahl, genau einen der Pfeile (a_1, v) oder (a_2, v) aufzunehmen.

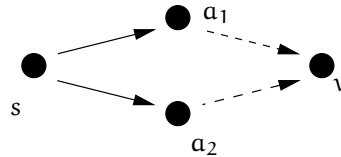


Abbildung 7.2:
Bäume kürzester Wege sind nicht immer eindeutig.

Das folgende Lemma nennt eine einfache, aber wichtige Eigenschaft der Abstände von s .

Lemma 7.5 Sei s eine ausgezeichnete Ecke in G . Dann gilt

$$\delta(s, v) \leq \delta(s, u) + c(u, v) \quad \text{für alle } (u, v) \in R.$$

Beweisskizze: Ein kürzester Weg von s nach v ist höchstens so lang wie ein beliebiger Weg w von s nach v . Insbesondere gilt dies auch für alle kürzesten Wege von s nach u , die um den Pfeil (u, v) zu einem Weg von s nach v verlängert werden. \square

Bei den Algorithmen, die im folgenden vorgestellt werden, tritt ähnlich wie bei DFS jeweils ein *Vorgängergraph* G_π auf, den wir wie folgt definieren: $G_\pi = (V_\pi, R_\pi)$ mit

$$\begin{aligned} V_\pi &:= \{v \in V : \pi[v] \neq \text{NIL}\} \cup \{s\} \\ R_\pi &:= \{(\pi[v], v) \in R : v \in V_\pi \wedge v \neq s\} \end{aligned}$$

Wir werden dann zeigen, daß G_π ein Baum kürzester Wege bezüglich s ist.

Algorithmus 7.1 Initialisierung für kürzeste Wege Algorithmen

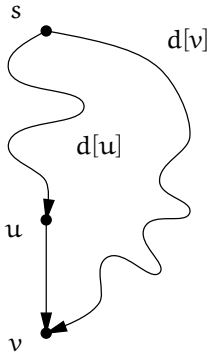
```

INIT( $G, s$ )
1 for all  $v \in V$  do
2    $d[v] \leftarrow +\infty$ 
3    $\pi[v] \leftarrow \text{NIL}$ 
4 end for
5  $d[s] \leftarrow 0$ 

```

Unsere Algorithmen benutzen alle die gleiche Initialisierung INIT aus Algorithmus 7.1. Die Werte $d[v]$ ($v \in V$) sind dabei obere Schranken für $\delta(s, v)$, die im Lauf des Algorithmus angepaßt werden.

Ferner benutzen wir sogenannte „Testschritte“ TESTE(u, v) für Pfeile (u, v), die in Algorithmus 7.2 dargestellt sind. Ein solcher Testschritt prüft, ob wir über u und den Pfeil (u, v) einen kürzeren Weg von s nach v als die aktuelle obere Schranke $d[v]$ finden können.

TESTE(u, v)**Algorithmus 7.2**

```

TESTE( $u, v$ )
1 if  $d[v] > d[u] + c(u, v)$  then
2    $d[v] \leftarrow d[u] + c(u, v)$ 
3    $\pi[v] \leftarrow u$ 
4 end if

```

Lemma 7.6 In G sei von s aus kein Kreis negativer Länge erreichbar. Ein Algorithmus initialisiere mit Hilfe von Algorithmus 7.1 und führe dann eine beliebige Anzahl von Testschritten (Algorithmus 7.2) aus. Dann gelten während des Algorithmus folgende Bedingungen:

- (i) $d[v] \geq \delta(s, v)$ für jedes $v \in V$.
- (ii) $G_\pi = (V_\pi, R_\pi)$ ist ein Wurzelbaum mit Wurzel s .
- (iii) G_π enthält für $v \in V_\pi \setminus \{s\}$ einen Weg von s nach v der Länge höchstens $d[v]$, dessen Spur durch $[s = \pi^{(k)}[v], \dots, \pi[v], v]$ gegeben ist.

Beweis: Wir beweisen die Behauptung durch Induktion nach der Anzahl p der Testschritte.

Falls $p = 0$, so ist $V_\pi = \{s\}$, $R_\pi = \emptyset$, $d[s] = 0$ und $d[v] = +\infty$ für $v \neq s$. Die Aussagen sind somit alle offenbar richtig.

Die Aussagen gelten nach $p - 1$ Testschritten und es werde ein weiterer Schritt TESTE(u, v) ausgeführt. Wenn $d[v] \leq d[u] + c(u, v)$ war, so wird nichts an G_π oder den Werten π und d geändert. Somit ist in diesem Fall nichts zu zeigen.

Wenn $d[v] > d[u] + c(u, v)$ war, so wird nun $d[v]$ auf $d[u] + c(u, v)$ vermindert, $\pi[v] = u$ gesetzt, die Kante (u, v) zu G_π hinzugenommen und eventuell die alte Kante (x, v) vom alten Vorgänger x von v entfernt.

Zu (i) Es gilt nach Induktionsvoraussetzung

$$d[u] + c(u, v) \stackrel{\text{IV}}{\geq} \delta(s, u) + c(u, v) \stackrel{\text{Lemma 7.5}}{\geq} \delta(s, v)$$

Zu (ii) G_π besitzt immer $|V_\pi| - 1$ Pfeile, jede Ecke in G_π ungleich s besitzt Innengrad 1, während s Innengrad 0 besitzt. Zeigen wir, daß alle Ecken in V_π von s aus erreichbar sind, so folgt, daß $E_{G_\pi}(s) = V_\pi$ (und damit G_π insbesondere schwach zusammenhängend) und somit nach Lemma 5.26 die Behauptung.

War vorher $\pi[v] = \text{NIL}$, so wird G_π um die Ecke v und den Pfeil (u, v) erweitert. Nach Induktionsvoraussetzung ist u von s in G_π aus erreichbar, also damit dann auch v .

Sei nun $\pi[v] = x \neq \text{NIL}$. Dann wird der Pfeil (x, v) in G_π durch (u, v) ersetzt. Nach Induktionsvoraussetzung war G_π vor dem Tausch ein Wurzelbaum. Es genügt daher zu zeigen, daß nach dem Tausch die Ecke v von s in G_π erreichbar ist.

Ist nämlich y ein Nachkomme von v in G_π , so ist in diesem Fall y auch weiterhin von s aus erreichbar. Ansonsten benutzte der Weg von s nach y vor dem Tausch den Pfeil (x, v) nicht und ist damit immer noch vorhanden.

Wir betrachten die Folge v_0, v_1, \dots mit $v_i := \pi^{(i)}[v]$.² Wenn diese Folge irgendwann mit $v_k = s$ abbricht, so liefert sie uns in umgekehrter Reihenfolge einen Weg von s nach v .

Ansonsten gilt irgendwann $v_j = v_{j+t}$ und (falls j und t minimal gewählt wurden) $[v_{j+t}, v_{j+t-1}, \dots, v_j]$ ist die Spur eines elementaren Kreises in G_π . Da G_π nach Induktionsvoraussetzung vor dem Austausch von (x, v) durch (u, v) kreisfrei war, muß er den Pfeil (u, v) enthalten, also muß $j = 0$ gelten. Dann ist $v_1 = \pi[v] = u$ und unser Kreis besitzt die Form $[v_t = v, \dots, v_1 = u, v_0 = v]$

Nach Konstruktion des Algorithmus gilt

$$d[v_i] \geq d[v_{i+1}] + c(v_{i+1}, v_i), \quad \text{für } i = 0, \dots, t-1 \quad (7.2)$$

denn nach Setzen von $\pi[v_i] := v_{i+1}$ gilt Gleichheit und eventuell wurde $d[v_{i+1}]$ nachher noch verringert. Ferner gilt

$$d[v_0] = d[v] > d[u] + c(u, v) = d[v_1] + c(v_1, v_0). \quad (7.3)$$

Summation von (7.2) und (7.3) ergibt

$$\sum_{i=0}^{t-1} c(v_{i+1}, v_i) < d[v_0] - d[v_t] = 0.$$

Somit haben wir einen Kreis negativer Länge gefunden, der von s aus erreichbar ist (da beispielsweise u nach Induktionsvoraussetzung von s aus erreichbar war). Widerspruch!

Zu (iii) Nach (ii) ist G_π ein Wurzelbaum mit Wurzel s . Wir haben in (ii) auch gezeigt, daß für $v \in V_\pi \setminus \{s\}$ die Spur des eindeutigen Weges von s nach v durch $[s = \pi^{(k)}[v], \dots, \pi[v], v]$ gegeben ist. Nach Konstruktion gilt nun wieder

$$d[\pi^{(i)}[v]] \geq d[\pi^{(i+1)}[v]] + c(\pi^{(i+1)}[v], \pi^{(i)}[v]), \quad \text{für } i = 0, \dots, k-1 \quad (7.4)$$

²Dabei ist $\pi^{(0)}[v] := v$ und $\pi^{(i)}[v] := \pi[\pi^{(i-1)}[v]]$ für $i > 0$, sofern $\pi^{(i-1)}[v] \neq \text{NIL}$.

Summation von (7.4) liefert dann:

$$\sum_{i=0}^{k-1} c(\pi^{(i+1)}[v], \pi^{(i)}[v]) \leq d[\pi^{(0)}[v]] - d[\pi^{(k)}[v]] = d[v] - d[s] = d[v].$$

Somit besitzt der oben angegebene Weg in G_π Länge höchstens $d[v]$. \square

Korollar 7.7 Erzeugt ein Algorithmus nach Initialisierung durch eine Anzahl von Testschritten $d[v] = \delta(s, v)$ für jedes $v \in V$, so ist G_π ein Baum kürzester Wege bezüglich s .

Beweis: Nach Lemma 7.6 enthält der Wurzelbaum G_π für jedes $v \in V_\pi$ einen Weg der Länge höchstens $d[v] = \delta(s, v)$. Da G_π nur Pfeile aus G enthält, kann dieser Weg auch nicht kürzer als ein kürzester Weg sein. \square

7.5 Der Algorithmus von Dijkstra

In diesem Abschnitt setzen wir voraus, daß $c(r) \geq 0$ für alle $r \in R$ ist. Wir lösen das Eine-Quelle-Kürzeste-Wege-Problem in diesem Fall mit dem Algorithmus von Dijkstra, der in Algorithmus 7.3 dargestellt ist.

Algorithmus 7.3 Algorithmus von Dijkstra

Input: Ein gerichteter Graph $G = (V, R)$ in Adjazenzlistendarstellung; eine nichtnegative Gewichtsfunktion $c: R \rightarrow \mathbb{R}_{\geq 0}$ und ein Knoten $s \in V$

- 1 INIT(G, s)
- 2 $Q \leftarrow \emptyset$
- 3 **for all** $v \in V$ **do**
- 4 INSERT(Q, v) {Füge v mit Schlüssel $d[v]$ in die Prioritätsschlange Q ein}
- 5 **end for**
- 6 DECREASE-KEY($Q, s, 0$) {Vermindere den Schlüsselwert von s in Q auf 0}
- 7 $S \leftarrow \emptyset$
- 8 **while** $Q \neq \emptyset$ **do**
- 9 $u \leftarrow$ EXTRACT-MIN(Q)
- 10 $S \leftarrow S \cup \{u\}$
- 11 **for all** $v \in$ Adj[u] **do**
- 12 **if** $d[v] > d[u] + c(u, v)$ **then**
- 13 DECREASE-KEY($Q, v, d[u] + c(u, v)$) {Vermindere den Schlüssel $d[v]$ von v in Q auf $d[u] + c(u, v)$ }
- 14 $\pi[v] \leftarrow u$
- 15 **end if**
- 16 **end for**
- 17 **end while**

Lemma 7.8 Bei Abbruch vom Algorithmus 7.3 gilt $d[v] = \delta(s, v)$ für alle $v \in V$. Der Graph G_π ist ein Baum kürzester Wege bezüglich s .

Beweis: Der erste Teil wird in Übung 7.2 gezeigt. Der zweite Teil folgt unmittelbar aus Korollar 7.7. \square

Satz 7.9 *Algorithmus 7.3 kann so implementiert werden, daß seine Laufzeit in $\mathcal{O}(n \log n + m)$ liegt.*

Beweis: Siehe Übung 7.2. \square

7.6 Der Algorithmus von Bellman und Ford

Während der Algorithmus von Dijkstra (Algorithmus 7.3) nur dann anwendbar ist, wenn die Gewichte $c(r)$ nicht negativ sind, so kann der Algorithmus von Bellman und Ford, den wir in diesem Abschnitt vorstellen werden, auch für negative Gewichte eingesetzt werden. Natürlich tritt bei negativen Gewichten möglicherweise der Fall ein, daß ein negativer Kreis von s aus erreichbar ist, d.h. daß $\delta(s, v) = -\infty$ für Ecken $v \in V$ gilt. Wir werden sehen, daß der Algorithmus von Bellman und Ford benutzt werden kann, um diesen Fall zu entdecken.

Algorithmus 7.4 Algorithmus Bellman und Ford

Input: Ein gerichteter Graph $G = (V, R)$ in Adjazenzlistendarstellung; eine Gewichtsfunktion $c: R \rightarrow \mathbb{R}$ und ein Knoten $s \in V$

```

1 INIT( $G, s$ )
2 for  $i \leftarrow 1, \dots, |V| - 1$  do
3   for all  $(u, v) \in R$  do
4     TESTE( $u, v$ )
5   end for
6 end for
7 for all  $(u, v) \in R$  do
8   if  $d[v] > d[u] + c(u, v)$  then
9     return „Ein negativer Kreis ist von  $s$  erreichbar.“
10  end if
11 end for

```

Lemma 7.10 *Wenn G keinen negativen Kreis enthält, der von s aus erreichbar ist, dann gilt bei Abbruch des Algorithmus 7.4: $d[v] = \delta(s, v)$ für alle $v \in V$. Ferner ist G_π ein Baum kürzester Wege bezüglich s .*

Beweis: Falls v nicht von s aus erreichbar ist, so folgt bereits aus Lemma 7.6 (i), daß bei Abbruch $d[v] = +\infty = \delta(s, v)$ gilt.

Sei also $v \in E_G(s)$ und w ein kürzester Weg von s nach v . Da in G kein Kreis negativer Länge von s aus erreichbar ist, können wir o.B.d.A. annehmen, daß w elementar ist, $w = [s = v_0, v_1, \dots, v_k = v]$. Da w elementar ist, gilt $k \leq |V| - 1$.

Wir zeigen nun, daß nach dem i ten Durchlauf der **for**-Schleife in Zeile 2 gilt:

$$d[v_j] = \delta(s, v_j) \quad \text{für } j = 0, \dots, i. \quad (7.5)$$

Da $k \leq |V| - 1$ und wir $|V| - 1$ Durchläufe haben, folgt damit dann die Korrektheit des Algorithmus.

Für $i = 0$ sind die Aussagen trivial. Seien sie bereits für $i > 0$ bewiesen und wir betrachten den $(i + 1)$ -ten Durchlauf der Schleife. Im $(i + 1)$ ten Durchlauf wird in Schritt 3 der Pfeil (v_i, v_{i+1}) betrachtet. Danach gilt

$$d[v_{i+1}] \leq d[v_i] + c(v_i, v_{i+1}) \stackrel{IV}{=} \delta(s, v_i) + c(v_i, v_{i+1}) = \delta(s, v_{i+1}).$$

Die letzte Gleichheit folgt dabei aus der Tatsache, daß $[v_0, \dots, v_{i+1}]$ ein kürzester Weg von $s = v_0$ nach v_{i+1} ist.

Mit Lemma 7.6 (i) folgt nun, daß nach dem $(i + 1)$ ten Durchlauf der Schleife gilt: $d[v_{i+1}] = \delta(s, v_{i+1})$. Für die Ecken v_0, \dots, v_i galt bereits vor dem $(i + 1)$ ten Durchlauf die Gleichheit $d[v_j] = \delta(s, v_j)$. Da sich die Werte $d[v]$ nach Konstruktion beim Durchlauf nicht erhöhen können, folgt, daß bei Abbruch gilt: $d[v] = \delta(s, v)$ für alle $v \in V$.

Daß G_π ein Baum kürzester Wege bezüglich s ist, folgt wiederum aus Korollar 7.7. \square

Lemma 7.11 *Es gilt $v \in E_G(s)$ genau dann, wenn bei Abbruch des Algorithmus 7.4 gilt: $d[v] < +\infty$.*

Beweis:

„ \Rightarrow “

Sei $v \in E_G(s)$ und $w = [v_0 = s, \dots, v_k = v]$ ein elementarer Weg von s nach v in G . Man zeigt nun ähnlich wie im Lemma 7.10 durch Induktion nach i , daß nach dem i ten Durchlauf der **for**-Schleife in Zeile 2 gilt: $d[v_i] \leq \sum_{j=0}^{i-1} c(v_j, v_{j+1})$. Damit folgt dann, $d[v] < +\infty$ bei Abbruch.

„ \Leftarrow “

Annahme: $d[v] < +\infty$ aber $v \notin E_G(s)$. Nach Lemma 7.6 (i) impliziert $v \notin E_G(s)$, daß $d[v] = +\infty$, Widerspruch! \square

Lemma 7.12 *Der Algorithmus 7.4 entscheidet korrekt, ob es einen Kreis negativer Länge in G gibt, der von s aus erreichbar ist.*

Beweis: G besitze keinen solchen Kreis. Für jeden Pfeil $(u, v) \in R$ gilt nach Lemma 7.5 die Ungleichung: $\delta(s, v) \leq \delta(s, u) + c(u, v)$. Nach Lemma 7.10 gilt bei Abbruch von Algorithmus 7.4: $\delta(s, v) = d[v]$. Somit folgt $d[v] \leq d[u] + c(u, v)$ für alle $(u, v) \in R$ und der Algorithmus informiert nicht über einen Kreis negativer Länge.

Sei nun w ein o.B.d.A. elementarer Kreis negativer Länge in G , der von s aus erreichbar ist, etwa $w = [v_0, \dots, v_{k+1} = v_0]$ mit $v_0 \in E_G(s)$. Dann gilt:

$$\sum_{i=0}^k c(v_i, v_{i+1}) < 0. \quad (7.6)$$

Annahme: Der Algorithmus informiert *nicht* über den negativen Kreis. Dann gilt bei Abbruch $d[v_{i+1}] \leq d[v_i] + c(v_i, v_{i+1})$ für $i = 0, \dots, k$. Nach Lemma 7.11 ist $d[v_i]$ für $i = 0, \dots, k$ endlich. Also haben wir

$$d[v_{i+1}] - d[v_i] \leq c(v_i, v_{i+1}) \quad \text{für } i = 0, \dots, k \quad (7.7)$$

Summation von (7.7) für $i = 0, \dots, k$ ergibt

$$\sum_{i=0}^k c(v_i, v_{i+1}) \geq d[v_{k+1}] - d[v_0] = 0,$$

da $v_0 = v_{k+1}$, im Widerspruch zu (7.6). \square

Satz 7.13 *Algorithmus 7.4 entscheidet korrekt, ob es einen Kreis negativer Länge in G gibt, der von s aus erreichbar ist. Falls es keinen solchen Kreis gibt, werden die Distanzen $\delta(s, v)$ ($v \in V$) korrekt berechnet. Der Algorithmus kann so implementiert werden, daß er in $\mathcal{O}(|V| \cdot |R|)$ Zeit läuft.*

Beweis: Die Korrektheit wurde bereits in den Lemmata 7.10 und 7.12 gezeigt. Es verbleibt die Laufzeitanalyse. Die Initialisierung benötigt $\mathcal{O}(n)$ Zeit. Die Hauptschleife wird $\mathcal{O}(n)$ mal durchlaufen, wobei die innere Schleife jeweils $\mathcal{O}(m)$ Zeit benötigt. Der abschließende Test benötigt $\mathcal{O}(m)$ Zeit. \square

7.7 Die Bellmanschen Gleichungen und kreisfreie Graphen

Wir betrachten wieder das Ein-Quellen-Kürzeste-Wege-Problem. Wir nehmen an, daß G keinen Kreis negativer Länge besitzt, der von s aus erreichbar ist.

Sei $w = (r_1, \dots, r_k)$ ein kürzester Weg von s zu einer Ecke v . Man beachte, daß unter unseren Voraussetzungen an G ein solcher Weg existiert. Dann muß $w' = (r_1, \dots, r_{k-1})$ ein kürzester Weg von s zu $w(r_{k-1})$ sein (wobei wir $w(r_{k-1}) := s$ setzen, falls $w = (r_1)$ nur aus einem Pfeil besteht). Somit erfüllen die Abstände $\delta(s, v)$ in G die folgenden sogenannten *Bellmanschen Gleichungen*³

$$\delta(s, s) = 0 \tag{7.8}$$

$$\delta(s, v) = \min(\{+\infty\} \cup \{\delta(s, u) + c(u, v) : (u, v) \in R\}), \quad \text{für } v \neq s \tag{7.9}$$

Ist G kreisfrei, so sind insbesondere keine Kreise negativer Länge von der Ecke s aus erreichbar. In diesem Fall kann man die Bellmanschen Gleichungen (7.8) und (7.9) benutzen, um die Abstände $\delta(s, v)$ ($v \in V$) in Zeit $\mathcal{O}(n + m)$ zu berechnen.

Ist nämlich σ eine topologische Sortierung von G , so folgt aus $(u, v) \in R$, daß $\sigma(u) < \sigma(v)$. Somit genügt es, die Ecken $v \in V$ in (7.9) gemäß der topologischen Sortierung zu betrachten. In diesem Fall ist dann $\delta(s, u)$ für alle Vorgänger u von v bereits berechnet, bevor wir das Minimum

³Manche Autoren definieren auch

$$c(v_i, v_j) := \begin{cases} c(r) & \text{falls } r = (v_i, v_j) \in R \\ +\infty & \text{falls } r = (v_i, v_j) \notin R. \end{cases}$$

und formulieren die Bellmanschen Gleichungen dann wie folgt:

$$\delta(s, s) = 0$$

$$\delta(s, v) = \min\{\delta(s, u) + c(u, v) : u \neq v\}, \quad \text{für } v \neq s$$

in (7.9) bestimmen. Algorithmus 7.5 setzt diese Idee um. Es ergibt sich damit der folgende Satz:

Satz 7.14 *Algorithmus 7.5 berechnet korrekt in $\mathcal{O}(n + m)$ Zeit die Abstände in einem kreisfreien Graphen von einem ausgezeichneten Knoten. Bei Abbruch ist G_π ein Baum kürzester Wege bezüglich s . \square*

Algorithmus 7.5 Kürzeste Wege Berechnungen in einem kreisfreien Graphen.

Input: Ein gerichteter kreisfreier Graph $G = (V, R)$ in Adjazenzlistendarstellung; eine Gewichtsfunktion $c: R \rightarrow \mathbb{R}$ und ein Knoten $s \in V$

```

1 INIT( $G, s$ )
2 Erstelle aus den Adjazenzlisten Adj neue Listen, die nach Endpunkten
  sortiert sind, d.h.  $u \in \text{Adj}'[v]$  gdw.  $(u, v) \in R$ 
3 Berechne eine topologische Sortierung von  $G$ . Sei  $L$  die Liste der
  topologisch sortierten Ecken.
4 for all  $v \in L$  in sortierter Reihenfolge do
5   for all  $u \in \text{Adj}'[v]$  do
6     TESTE( $u, v$ )
7   end for
8 end for

```

7.8 Längste Wege

Bisher haben wir uns damit beschäftigt *kürzeste Wege* von einer Ecke s zu allen anderen Ecken $v \in V \setminus \{s\}$ zu finden. Falls ein negativer Kreis von s aus erreichbar war, so haben wir dies (etwa mit dem Algorithmus von Ford und Bellman) erkannt und den Algorithmus abgebrochen.

Im Fall von negativen Kreisen könnten wir auch nach dem kürzesten *elementaren* Weg von s zu den Ecken $v \in V \setminus \{s\}$ fragen. Dies ist äquivalent dazu, bezüglich der Gewichtsfunktion $-c$ nach einem längsten elementaren Weg zu fragen.

Es zeigt sich nun, daß das Problem, längste elementare Wege zu bestimmen, deutlich „schwerer“ ist, als nach kürzesten Wegen zu fragen.

Satz 7.15 *Das Problem, zu entscheiden, ob in einem (gerichteten oder ungerichteten) Graphen G mit nichtnegativer Pfeil- bzw. Kantengewichtungsfunktion c ein elementarer Weg der Länge mindestens L von $s \in V$ nach $t \in V$ ($s \neq t$) existiert, ist NP-vollständig.*

Beweis: Das Problem beinhaltet als Teilproblem die Frage nach einem *Hamiltonschen Weg* von s nach t in G ($c \equiv 1$ und $L := |V| - 1$).

Man kann das Problem HAMILTONSCHER KREIS auf das Hamiltonschen Weg Problem in Polynomialzeit transformieren (siehe z.B. [GJ79]). \square

Übungsaufgaben

Aufgabe 7.1 – Baum kürzester Wege

Sei $G = (V, R, \alpha, \omega)$ ein Graph, $c: R \rightarrow \mathbb{R}_0^+$ eine nichtnegative Pfeilbewertung.

- Ein Weg w heißt *kürzester Weg (in G)*, falls $c(w) = \text{dist}_G(\alpha(w), \omega(w))$. Zeigen Sie: Jeder Teilweg eines kürzesten Weges ist seinerseits ein kürzester Weg.
- Sei $s \in V$ ein Knoten in G . Zeigen Sie: Es gibt einen Wurzelbaum $T_s = (V, R_T)$ mit Wurzel s , so daß

$$\text{dist}_T(s, v) = \text{dist}_G(s, v) \quad \text{für alle } v \in V.$$

Er wird *Baum kürzester Wege* (engl. *shortest path tree*) genannt.

Aufgabe 7.2 – Algorithmus von Dijkstra

- Beweisen Sie Lemma 7.8.
- Beweisen Sie Lemma 7.9.

Aufgabe 7.3 – Bäume kürzester Wege

In dieser Aufgabe sei $G = (V, R)$ ein endlicher Graph und $c: R \rightarrow \mathbb{R}$ eine Gewichtsfunktion. Es sei kein Kreis negativer Länge von der Ecke $s \in V$ erreichbar. Zeigen Sie, daß ein Baum kürzester Wege bezüglich s in G existiert.

Aufgabe 7.4 – Zuverlässigste Wege

Sei $G = (V, R)$ ein endlicher gerichteter Graph, der ein Kommunikationsnetz modelliert: die Ecken sind Verbindungsknoten, die Pfeile sind unidirektionale Kommunikationslinks.

Für jeden Pfeil $(u, v) \in R$ ist eine Wahrscheinlichkeit $0 \leq f(u, v) \leq 1$ bekannt, mit der die Verbindung (u, v) ausfällt. Dabei nehmen wir an, daß diese Wahrscheinlichkeiten voneinander unabhängig sind. Daher ist dann für einen Weg $w = (r_1, \dots, r_k)$ seine *Zuverlässigkeit*, d.h. die Wahrscheinlichkeit, daß er *nicht* ausfällt, durch

$$z(w) := \prod_{i=1}^k (1 - f(r_i))$$

gegeben.

Geben Sie einen effizienten Algorithmus an, der zu zwei gegebenen Ecken $s, t \in V$ einen *zuverlässigsten Weg* w von s nach t bestimmt, d.h. einen Weg w , für den $z(w)$ maximal ist.

Aufgabe 7.5 – Kürzeste Wege bei Einheitskosten (Breitensuche)

In dieser Aufgabe sei $G = (V, R)$ ein endlicher Graph und $s \in V$ eine Ecke. Wir betrachten das Problem, kürzeste Wege von s zu allen Ecken $v \in V$ zu bestimmen, wenn alle Pfeile $r \in R$ die Länge 1 besitzen. Dazu sei der Algorithmus 7.6 gegeben. Dieser Algorithmus ist als *Breitensuche* oder Breadth-First-Search bekannt.

Algorithmus 7.6 Breitensuche (Breadth-First-Search)BFS(G)

Input: Ein gerichteter Graph $G = (V, R)$ in Adjazenzlistendarstellung; ein Knoten $s \in V$

```

1 INIT( $G, s$ ) {Initialisierung durch Algorithmus 7.1}
2  $L \leftarrow \{s\}$  {Eine Liste, die nur  $s$  enthält}
3 while  $L \neq \emptyset$  do
4   Entferne das erste Element  $u$  aus  $L$ 
5   for all  $(u, v) \in R$  do
6     if  $d[v] > d[u] + 1$  then
7        $d[v] \leftarrow d[u] + 1$ 
8        $\pi[v] \leftarrow u$ 
9       Füge  $v$  an das Ende von  $L$  an.
10  end if
11  end for
12 end while

```

- (a) Beweisen Sie, daß bei Abbruch des Algorithmus $d[v] = \delta(s, v)$ für alle $v \in V$ gilt und G_π ein Baum kürzester Wege bezüglich s ist.
- (b) Zeigen Sie, daß man den obigen Algorithmus so implementieren kann, daß er in Zeit $\mathcal{O}(|V| + |R|)$ läuft.

Aufgabe 7.6 – Eigenschaften von Abständen in Graphen

In dieser Aufgabe sei $G = (V, R)$ ein endlicher Graph und $c: R \rightarrow \mathbb{R}$ eine Gewichtsfunktion. Lemma 7.5 der Vorlesung besagt, daß

$$\delta(s, v) \leq \delta(s, u) + c(u, v) \quad \text{für alle } (u, v) \in R.$$

Der skizzierte Beweis behandelte nur den Fall, daß $\delta(s, v)$ und $\delta(s, u)$ endlich sind. Vervollständigen Sie den Beweis für den Fall, daß auch $+\infty$ und $-\infty$ als Abstände auftreten können.

Kapitel 8

Strömungen und Flüsse

8.1 Strömungen und Schnitte

Definition 8.1 (Strömung)

Sei $G = (V, R, \alpha, \omega)$ ein gerichteter Graph. Eine Funktion $\beta: R \rightarrow \mathbb{R}$ heißt **Strömung** in G , wenn mit

$$\beta^+(v) := \sum_{r: \alpha(r)=v} \beta(r) \quad \text{und} \quad \beta^-(v) := \sum_{r: \omega(r)=v} \beta(r) \quad (8.1)$$

gilt:

$$\beta^+(v) = \beta^-(v) \quad \text{für alle } v \in V. \quad (8.2)$$

Dabei bedeutet für unendliche Graphen die Bedingung (8.2), daß die Reihen aus (8.1) konvergent sind und ihre Grenzwerte gleich sind.

Wir werden uns im folgenden auf *endliche* Graphen beschränken. Es ergibt sich nun nahezu unmittelbar das folgende Lemma, dessen Beweis wir nicht explizit führen.

Lemma 8.2 Es seien β und β' Strömungen in G . Dann sind auch $\beta + \beta'$ und $t \cdot \beta$ für jedes $t \in \mathbb{R}$ Strömungen. Folglich bildet die Menge $\mathfrak{S}(G)$ der Strömungen in G einen linearen Raum. \square

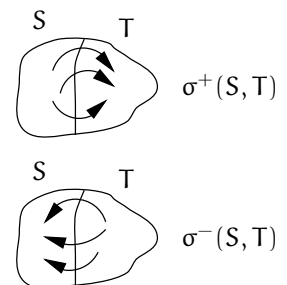
Definition 8.3 (Vorwärts- und Rückwärtsteil eines Schnittes)

Sei (S, T) ein Schnitt in G . Dann definieren wir den **Vorwärtsteil** $\sigma^+(S, T)$ des Schnittes durch

$$\sigma^+(S, T) := \{r \in R: \alpha(r) \in S \wedge \omega(r) \in T\} \quad (8.3)$$

und den **Rückwärtsteil** $\sigma^-(S, T)$ des Schnittes (S, T) durch

$$\sigma^-(S, T) := \{r \in R: \omega(r) \in S \wedge \alpha(r) \in T\} \quad (8.4)$$



Bemerkung 8.4 Manche Autoren (z.B. [Nol75]) definieren für eine Partition $S \cup T$ den Vorwärts- und den Rückwärtsteil des durch S induzierten Schnittes durch

$$\sigma^+(S) := \{+r: \alpha(r) \in S \wedge \omega(r) \in T\}$$

$$\sigma^-(S) := \{-r: \alpha(r) \in T \wedge \omega(r) \in S\}$$

und bezeichnen mit $\sigma(S) := \sigma^+(S) \cup \sigma^-(S)$ den durch S induzierten Schnitt.

Bis auf die Markierungen „+“ und „-“ der Pfeile im Vorwärts- und Rückwärtsteil entsprechen dann $\sigma(S)$, $\sigma^+(S)$ und $\sigma^-(S)$ unseren Definitionen $\sigma(S, T)$, $\sigma^+(S, T)$ und $\sigma^-(S, T)$.

Definition 8.5

Sei (S, T) ein Schnitt im Graphen G und $w: R \rightarrow \mathbb{R}$ eine beliebige Funktion. Dann definieren wir

$$w(S, T) := \sum_{r \in \sigma^+(S, T)} w(r).$$

Satz 8.6 Ist β eine Strömung in G und (S, T) ein Schnitt, so gilt:

$$\beta(S, T) = \beta(T, S).$$

Bevor wir den Satz beweisen, soll kurz die anschauliche Bedeutung seines Ergebnisses erläutert werden. Satz 8.6 besagt, daß aus der Menge S durch eine Strömung so viel „herausläuft“, wie in S hineinläuft. Dies entspricht unserer physikalischen Vorstellung einer Strömung.

Beweis von Satz 8.6: Es gilt:

$$\begin{aligned} \beta(S, T) &= \sum_{\substack{r: \alpha(r) \in S \\ \omega(r) \in T}} \beta(r) = \sum_{v \in S} \beta^+(v) - \sum_{\substack{r: \alpha(r) \in S \\ \omega(r) \in S}} \beta(r) \\ &\stackrel{(8.2)}{=} \sum_{v \in S} \beta^-(v) - \sum_{\substack{r: \alpha(r) \in S \\ \omega(r) \in S}} \beta(r) \\ &= \sum_{\substack{r: \omega(r) \in S \\ \alpha(r) \in T}} \beta(r) \\ &= \beta(T, S) \end{aligned}$$

Dies zeigt die behauptete Gleichheit. \square

8.2 Zulässige und maximale Strömungen

Voraussetzung 8.7 Im folgenden sei $G = (V, R, \alpha, \omega)$ ein endlicher gerichteter Graph und $l, c: R \rightarrow \mathbb{R}$ Funktionen, die den Pfeilen in G Kapazitäten zuweisen, wobei $0 \leq l(r) \leq c(r)$ für alle $r \in R$ gelte.

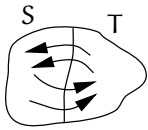
Unter der Voraussetzung 8.7 nennen wir für einen Schnitt (S, T) den Wert $c(S, T) - l(T, S)$ die *Kapazität des Schnittes* (S, T) . Anschaulich bezeichnet die Kapazität von (S, T) den maximalen Nettofluß aus S heraus: Maximal $c(S, T)$ Einheiten können aus S herausfließen, während mindestens $l(T, S)$ Einheiten hineinfließen müssen.

Definition 8.8 (Zulässige Strömung)

Sei G wie in Voraussetzung 8.7. Eine Strömung $\beta \in \mathfrak{S}(G)$ heißt **zulässig bezüglich der Kapazitätsfunktionen l und c** , wenn

$$l(r) \leq \beta(r) \leq c(r) \quad \text{für alle } r \in R$$

gilt.



$$\beta(S, T) = \beta(T, S)$$

Es stellt sich die Frage, ob in einem gegebenen Graphen überhaupt eine bezüglich der Kapazitätsschranken l und c zulässige Strömung existiert. Falls $l \equiv 0$ gilt, so ist die *Nullströmung* $\beta \equiv 0$ offenbar eine zulässige Strömung. Falls $l \not\equiv 0$, so ist die Existenz einer zulässigen Strömung nicht trivial.

Abbildung 8.1 zeigt einen Graphen, in dem keine zulässige Strömung existiert.

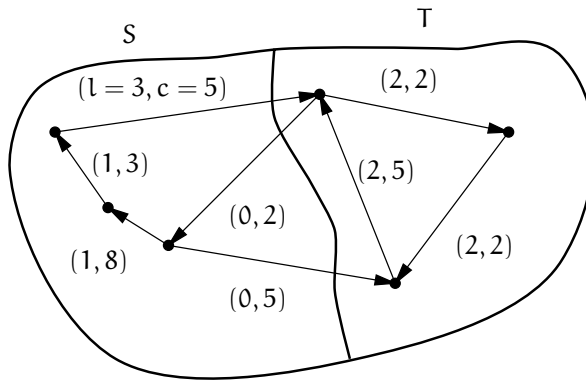


Abbildung 8.1:

Für zulässiges $\beta \in \mathfrak{C}(G)$ und den Schnitt (S, T) müßte $\beta(S, T) \geq 3$ und $\beta(T, S) \leq 2$ gelten. Dies widerspricht Satz 8.6.

Lemma 8.9 Eine notwendige Bedingung für die Existenz einer zulässigen Strömung in G ist, daß

$$c(S, T) \geq l(T, S) \quad (8.5)$$

für alle Schnitte (S, T) in G gilt.

Vor dem Beweis wiederum kurz die anschauliche Bedeutung der Aussage des Lemmas: Das Lemma besagt, daß für jeden Schnitt (S, T) mindestens so viel aus S herauslaufen können darf, wie mindestens nach S hineinlaufen muß.

Beweis von Lemma 8.9: Sei β eine zulässige Strömung in G . Dann gilt:

$$0 \stackrel{\text{Satz 8.6}}{=} \beta(S, T) - \beta(T, S) \stackrel{l \leq \beta \leq c}{\leq} c(S, T) - l(T, S)$$

Somit ist die Behauptung bewiesen. \square

Wir werden nachher (Satz von Hoffman) zeigen, daß die nahezu trivialerweise notwendige Bedingung (8.5) auch hinreichend für die Existenz einer zulässigen Strömung in G ist.

Definition 8.10 (Maximalströmungsproblem)

Gegeben ist ein Graph G wie in Voraussetzung 8.7. Ferner sei $r_0 \in R$ ein ausgezeichneter Pfeil des Graphen mit $\alpha(r_0) \neq \omega(r_0)$ und $l(r_0) = 0$. Bestimmt werden soll eine bezüglich l und c zulässige Strömung β^* mit maximalem Flußwert $\beta^*(r_0)$. Falls keine zulässige Strömung existiert, soll darüber informiert werden.

In der obigen Definition haben wir vorausgesetzt, daß der ausgezeichnete Pfeil r_0 keine Schlinge ist. In der Tat ist im Fall, daß $\alpha(r_0) = \omega(r_0)$ gilt, das Maximalströmungsproblem kaum interessant. Sofern

nur eine zulässige Strömung $\beta \in \mathfrak{S}(G)$ existiert, können wir den Strömungswert $\beta(r_0)$ auf dem Pfeil r_0 dann auf die obere Kapazitätsschranke $c(r_0)$ erhöhen und erhalten damit eine maximale Strömung in G .

Weiterhin haben wir in der letzten Definition vorausgesetzt, daß für den ausgezeichneten Pfeil r_0 gilt: $l(r_0) = 0$. Dies stellt keine Einschränkung dar, da wir den Flußwert auf r_0 *maximieren* wollen.

Satz 8.11 *Es gelten die Voraussetzungen wie in Definition 8.10. Ist β eine bezüglich l und c zulässige Strömung in G , so gilt*

$$\beta(r_0) \leq c(S, T) - l(T, S) \quad (8.6)$$

für jeden Schnitt (S, T) in G mit $r_0 \in \sigma^-(S, T)$. Für den Fall, daß $l \equiv 0$ gilt, haben wir

$$\beta(r_0) \leq c(S, T) \quad (8.7)$$

für jeden Schnitt (S, T) in G mit $r_0 \in \sigma^-(S, T)$.

Beweis: Es gilt

$$\beta(T, S) = \beta(r_0) + \sum_{\substack{r \in \sigma^+(T, S) \\ r \neq r_0}} \beta(r).$$

Nach Satz 8.6 ist $\beta(T, S) = \beta(S, T)$. Also haben wir

$$\begin{aligned} \beta(r_0) &= \beta(T, S) - \sum_{\substack{r \in \sigma^+(T, S) \\ r \neq r_0}} \beta(r) \stackrel{\text{Satz 8.6}}{=} \beta(S, T) - \sum_{\substack{r \in \sigma^+(T, S) \\ r \neq r_0}} \beta(r) \\ &\stackrel{l \leq \beta \leq c}{\leq} c(S, T) - \sum_{\substack{r \in \sigma^+(T, S) \\ r \neq r_0}} l(r) \stackrel{l(r_0) = 0}{=} c(S, T) - l(T, S). \end{aligned}$$

Dies zeigt die Behauptung. \square

Korollar 8.12 *Es gelten die Voraussetzungen wie in Definition 8.10. Es gebe in G eine bezüglich l und c zulässige Strömung. Dann gilt:*

$$\begin{aligned} &\max\{\beta(r_0) : \beta \text{ ist zulässige Strömung in } G\} \\ &\leq \min(c(r_0), \min\{c(S, T) - l(T, S) : r_0 \in \sigma^-(S, T)\}) \end{aligned}$$

Wir werden nachher zeigen, daß in der obigen Ungleichung tatsächlich sogar Gleichheit gilt.

Bemerkung 8.13 *Wir haben bisher stillschweigend vorausgesetzt, daß eine maximale Strömung existiert. Dies ist aber nicht vollkommen trivial, da für jede zulässige Strömung β der Wert $\beta(r_0)$ nach Satz 8.11 beschränkt ist, eventuell aber nur ein Supremum und kein Maximum existieren könnte.*

Wir zeigen daher kurz, daß tatsächlich eine maximale Strömung existiert. Wir können jede Strömung β in G als Vektor des \mathbb{R}^m ($m := |R|$) auffassen: $\beta = (\beta(r_1), \dots, \beta(r_m))$. Die Menge F der zulässigen Strömungen läßt sich dann wie folgt schreiben:

$$F = \left\{ \beta : \begin{array}{ll} l(r_i) \leq \beta(r_i) \leq c(r_i), & \text{für } i = 1, \dots, m \\ \sum_{r: \alpha(r)=v} \beta(r) - \sum_{r: \omega(r)=v} \beta(r) = 0, & \text{für alle } v \in V \end{array} \right\}$$

Wegen der ersten Bedingung $l(r_i) \leq \beta(r_i) \leq c(r_i)$ für $i = 1, \dots, m$ ist F beschränkt. Die Abgeschlossenheit von F folgt unmittelbar aus der obigen Definition. Folglich ist F kompakt. Nach bekannten Sätzen der Analysis nimmt daher die stetige Funktion $f: F \rightarrow \mathbb{R}$ mit $f(\beta) := \beta(r_0)$ ihr Maximum auf F an, d.h. es existiert ein $\beta^* \in F$ mit

$$f(\beta^*) = \max\{\beta(r_0) : \beta \in F\}.$$

Somit existiert eine maximale Strömung β^* . □

8.3 Das Max-Flow-Min-Cut-Theorem

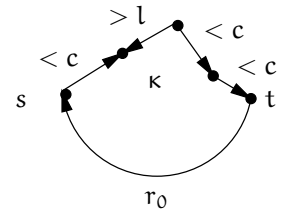
Wir gehen zunächst davon aus, daß in G eine zulässige Strömung β bekannt ist. Im Fall, daß die unteren Kapazitätsschranken $l(r)$ alle gleich 0 sind, kann dies etwa die Nullströmung $\beta_0 \equiv 0$ sein.

Ansonsten seien die Voraussetzungen wie beim Maximalströmungsproblem (Definition 8.10) gegeben.

Definition 8.14 (Strömungsvergrößernde Kette)

Eine elementare Kette $\kappa = (\delta_1 r_1, \dots, \delta_k r_k)$ in $G_{R \setminus \{r_0\}}$ heißt **strömungsvergrößernde Kette**, wenn sie folgende Eigenschaften besitzt:

- (i) $\alpha(\kappa) = s = \omega(r_0)$ und $\omega(\kappa) = t = \alpha(r_0)$,
- (ii) $\beta(r_i) < c(r_i)$ für alle i mit $\delta_i = 1$ und
- (iii) $\beta(r_i) > l(r_i)$ für alle i mit $\delta_i = -1$.



Sei β eine zulässige Strömung in G und $\kappa = (\delta_1 r_1, \dots, \delta_k r_k)$ eine strömungsvergrößernde Kette. Wir setzen

$$\varepsilon_1 := \min\{c(r_i) - \beta(r_i) : \delta_i = 1\} > 0 \tag{8.8}$$

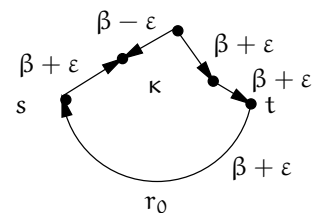
$$\varepsilon_2 := \min\{\beta(r_i) - l(r_i) : \delta_i = -1\} > 0. \tag{8.9}$$

Dabei folgt die Tatsache, daß $\varepsilon_1, \varepsilon_2$ strikt positiv sind, direkt aus der Definition einer strömungsvergrößernden Kette. Wir definieren nun noch

$$\varepsilon_3 := c(r_0) - \beta(r_0) \geq 0. \tag{8.10}$$

Gilt sogar $\varepsilon_3 > 0$, so ist für $0 < \varepsilon \leq \min\{\varepsilon_1, \varepsilon_2, \varepsilon_3\}$ die Strömung $\beta' := \beta + \beta_\varepsilon$ mit

$$\beta_\varepsilon(r) := \begin{cases} \delta_i \cdot \varepsilon & \text{falls } r = r_i \text{ für ein } i \\ \varepsilon & \text{falls } r = r_0 \\ 0 & \text{sonst} \end{cases} \tag{8.11}$$



eine zulässige Strömung¹ in G mit $\beta'(r_0) = \beta(r_0) + \varepsilon > \beta(r_0)$.

Durch die obige Argumentation haben wir somit folgendes Lemma bewiesen:

Lemma 8.15 Sei $\beta \in \mathfrak{S}(G)$ eine zulässige Strömung mit $\beta(r_0) < c(r_0)$ und κ eine strömungsvergrößernde Kette. Dann kann man eine zulässige Strömung $\beta' \in \mathfrak{S}(G)$ finden, die $\beta'(r_0) > \beta(r_0)$ erfüllt. □

¹Die Zulässigkeit von β' folgt sofort aus der Definition. Daß β' tatsächlich eine Strömung ist, verifiziert man leicht.

Bemerkung 8.16 Sind die Ausgangsströmung β sowie die Kapazitätsschranken l und c ganzzahlig, so kann auch die Strömung β' aus dem letzten Lemma ganzzahlig bestimmt werden, wenn wir $\beta' := \beta + \beta_\varepsilon$ (siehe (8.11)) mit

$$\varepsilon := \min\{\varepsilon_1, \varepsilon_2, \varepsilon_3\}$$

wählen, da die Werte ε_i aus (8.8)–(8.10) dann ganzzahlig sind.

Lemma 8.17 Es gelten die Voraussetzungen von Definition 8.10. Sei $\beta \in \mathfrak{C}(G)$ eine zulässige Strömung. Es existiere keine strömungsvergrößernde Kette von s nach t . Dann ist β eine maximale Strömung und es existiert ein Schnitt (S, T) mit

$$\beta(r_0) = c(S, T) - l(T, S).$$

Beweis: Wir nennen eine elementare Kette $\kappa = (\delta_1 r_1, \dots, \delta_k r_k)$ mit $\alpha(\kappa) = s$ und $\omega(\kappa) = v$ eine *vergrößernde Kette*, wenn sie die Eigenschaften (ii) und (iii) aus Definition 8.14 besitzt. Dann ist jede vergrößernde Kette κ mit $\omega(\kappa) = t$ eine strömungsvergrößernde Kette.

Setze nun

$$S := \{s\} \cup \{v \in V : \text{es gibt eine vergrößernde Kette nach } v\} \quad (8.12)$$

und $T := V \setminus S$. Nach Voraussetzung ist $t \notin S$, also $t \in T$. Daher ist (S, T) ein Schnitt mit $r_0 \in \sigma^-(S, T)$.

Sei $r \in \sigma^+(S, T)$ mit $u := \alpha(r)$ und $v := \omega(r)$. Nach Definition gilt $v \in T$, und es existiert nach Definition von S und T daher keine vergrößernde Kette nach v . Wir behaupten, daß $\beta(r) = c(r)$ gelten muß.

Annahme: $\beta(r) < c(r)$. Ist $u = s$, so ist $(+r)$ eine vergrößernde Kette im Widerspruch zu $v \in T$. Ist $u \neq s$, so sei $\kappa = (\delta_1 r_1, \dots, \delta_k r_k)$ eine vergrößernde Kette nach u . Diese Kette kann nicht v berühren, da nach Voraussetzung $v \in T$ gilt (und sonst die Teilkette von s nach v bereits eine vergrößernde Kette nach v wäre). Dann ist aber $(\delta_1 r_1, \dots, \delta_k r_k, +r)$ eine einfache Kette, die vergrößernde Kette nach v ist, im Widerspruch zu $v \in T$.

Somit haben wir gezeigt, daß für den Schnitt (S, T) die Gleichung

$$\beta(S, T) = c(S, T) \quad (8.13)$$

gilt. Wir betrachten nun die Pfeile in $\sigma^-(S, T) \setminus \{r_0\}$. Wir behaupten, daß für $r \in \sigma^-(S, T)$ die Beziehung $\beta(r) = l(r)$ gelten muß.

Wäre nämlich $\beta(r) > l(r)$, so zeigt man durch Fallunterscheidung nach $\omega(r) = s$ bzw. $\omega(r) \neq s$ ähnlich wie oben, daß in beiden Fällen eine vergrößernde Kette nach $\alpha(r)$ existiert, was der Voraussetzung $\alpha(r) \in T$ widerspricht.

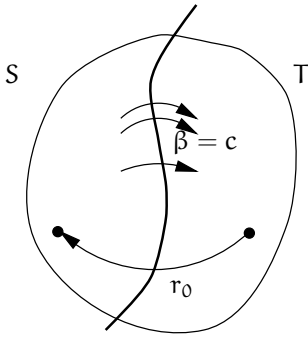
Daher gilt für jeden Pfeil $r \in \sigma^-(S, T) \setminus \{r_0\}$ die Beziehung $\beta(r) = l(r)$ und wir haben für (S, T)

$$\begin{aligned} \beta(T, S) &= \beta(r_0) + \sum_{r \in \sigma^-(S, T) \setminus \{r_0\}} \beta(r) \\ &= \beta(r_0) + \sum_{r \in \sigma^-(S, T) \setminus \{r_0\}} l(r) \stackrel{l(r_0)=0}{=} \beta(r_0) + l(T, S). \end{aligned} \quad (8.14)$$

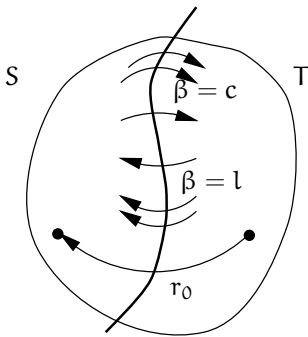
Wegen $\beta(T, S) = \beta(S, T)$ (Satz 8.6) ergibt sich aus (8.14):

$$\beta(r_0) = \beta(S, T) - l(T, S) \stackrel{(8.13)}{=} c(S, T) - l(T, S).$$

Nach Satz 8.11 ist dann aber β eine maximale Strömung. \square



$$\beta(S, T) = c(S, T)$$



Satz 8.18 (Max-Flow-Min-Cut-Theorem) *Es gelten die Voraussetzungen wie in Definition 8.10. Es gebe in G eine bezüglich l und c zulässige Strömung. Dann gilt:*

$$\begin{aligned} & \max\{ \beta(r_0) : \beta \text{ ist zulässige Strömung in } G \} \\ & = \min\{ c(r_0), \min\{ c(S, T) - l(T, S) : r_0 \in \sigma^-(S, T) \} \} \end{aligned}$$

Beweis: Sei β^* eine maximale Strömung. Korollar 8.12 liefert bereits, daß $\beta^*(r_0) \leq M$, wobei

$$M := \min\{ c(r_0), \min\{ c(S, T) - l(T, S) : r_0 \in \sigma^-(S, T) \} \}.$$

Es genügt also, $\beta^*(r_0) \geq M$ zu zeigen.

Wenn $\beta^*(r_0) = c(r_0)$ ist, so gilt $\beta^*(r_0) = c(r_0) \geq M$ und der Beweis ist vollständig.

Ist $\beta^*(r_0) < c(r_0)$, so kann es nach Lemma 8.15 keine strömungsvergrößernde Kette von s nach t geben. Nach Lemma 8.17 existiert ein Schnitt (S, T) mit

$$\beta^*(r_0) = c(S, T) - l(T, S) \geq M.$$

Somit folgt wieder $\beta^*(r_0) = M$ und der Beweis ist vollständig. \square

8.4 Der Algorithmus von Ford und Fulkerson

Der Algorithmus von Ford und Fulkerson startet mit einer beliebigen zulässigen Strömung β in G (gilt $l \equiv 0$, so können wir mit der Nullströmung $\beta \equiv 0$ starten). Sofern noch $\beta(r_0) < c(r_0)$ gilt, erhöht er die Strömung entlang einer strömungsvergrößernden Kette. Wenn es keine solche strömungsvergrößernde Kette mehr gibt, muß die aktuelle Strömung nach Lemma 8.17 maximal sein.

Algorithmus 8.1 Maximalströmungs-Alg. von Ford und Fulkerson

Input: Ein gerichteter schwach zsh. Graph $G = (V, R, \alpha, \omega)$;
Kapazitätsschranken $0 \leq l(r) \leq c(r)$ ($r \in R$), ein
ausgezeichneter Pfeil $r_0 \in R$;
eine zulässige Ausgangsströmung β in G

```

1  $\beta' \leftarrow \beta$ 
2 while es gibt eine strömungsvergr. Kette  $\kappa$  und  $\beta(r_0) < c(r_0)$  do
3   Sei  $\kappa = (\delta_1 r_1, \dots, \delta_k r_k)$ 
4   Berechne  $\varepsilon_1, \varepsilon_2$  und  $\varepsilon_3$  wie in (8.8)–(8.10)
5    $\varepsilon \leftarrow \min\{\varepsilon_1, \varepsilon_2, \varepsilon_3\}$ 
6   Setze  $\beta' := \beta' + \beta_\varepsilon$  mit  $\beta_\varepsilon$  wie in (8.11)
7 end while

```

Der Algorithmus von Ford und Fulkerson ist in Algorithmus 8.1 dargestellt. Er besitzt eine wichtige Eigenschaft. Sofern die Ausgangsströmung β und die Funktionen l und c ganzzahlig sind, sind alle zwischenzeitlich vom Algorithmus berechneten Strömungen β' ganzzahlig.

Für diesen Fall ergibt sich auch die Korrektheit mit Hilfe der Lemmata 8.15 und 8.17: In jedem Schritt wird der Strömungswert $\beta'(r_0)$

um mindestens eine Einheit erhöht. Somit bricht der Algorithmus nach maximal $\beta^*(r_0) - \beta(r_0)$ Iterationen mit einer Strömung β' ab, für die entweder $\beta'(r_0) = c(r_0)$ gilt oder für die keine strömungsvergrößernde Kette existiert. Nach Lemma 8.17 ist diese Strömung β' dann eine maximale Strömung.

Damit ergibt sich als Nebenprodukt der folgende wichtige Satz:

Satz 8.19 (Ganzzahligkeitssatz) *Sei G wie in Definition 8.10 und seien die Funktionen l und c ganzzahlig mit $l \equiv 0$. Dann existiert in G eine maximale Strömung β^* , die ganzzahlig ist.* \square

Für rationale Werte β , l und c ergibt sich die Korrektheit des Algorithmus von Ford und Fulkerson wie folgt. Sei N ein gemeinsamer Hauptnenner aller auftretenden rationalen Werte $\beta(r)$, $l(r)$, $c(r)$ ($r \in R$). Dann folgt für jedes ε , welches in Schritt 5 bestimmt wird, daß $\varepsilon \geq 1/N$. Somit terminiert der Algorithmus nach maximal $N \cdot (\beta^*(r_0) - \beta(r_0))$ Iterationen. Wie oben folgt dann, daß die Strömung β' bei Abbruch eine maximale Strömung ist.

Somit haben wir folgenden Satz gezeigt:

Satz 8.20 *Ist G wie in Definition 8.10 mit ganzzahligen (rationalen) Kapazitäten l und c und β eine zulässige ganzzahlige (rationale) Strömung in G , so bestimmt der Algorithmus von Ford und Fulkerson eine maximale Strömung.* \square

Der Algorithmus von Ford und Fulkerson kann im Fall von reellen Kapazitäten (bei ungeschickter Wahl der strömungsvergrößernden Ketten) versagen. Ford und Fulkerson haben ein Beispiel angegeben, bei dem das Verfahren nicht nur nicht abbricht, sondern auch gegen eine Strömung β' konvergiert, deren Strömungswert $\beta'(r_0)$ nur ein Viertel des maximalen Strömungswertes beträgt. In der Praxis spielen reelle Kapazitäten natürlich keine Rolle, da man im Computer sowieso nur rationale Zahlen darstellen kann.

Effizienz des Algorithmus von Ford und Fulkerson

Wir haben bereits argumentiert, daß der Algorithmus von Ford und Fulkerson maximal $\beta^*(r_0) - \beta(r_0) \leq \beta^*(r_0) - l(r_0)$ Iterationen benötigt, in denen jeweils eine strömungsvergrößernde Kette bestimmt wird. Abbildung 8.2 zeigt einen Graphen $G = (V, R)$ mit Kapazitäten $c(r)$ ($r \in R$) und $l \equiv 0$, bei dem der Algorithmus tatsächlich $M = \beta^*(r_0)$ Iterationen benötigt, wenn er mit der Nullströmung $\beta \equiv 0$ startet und als strömungsvergrößernde Ketten jeweils abwechselnd $(+r_1, +r_2, +r_5)$ und $(+r_4, -r_2, +r_3)$ gewählt werden.

Wir werden weiter unten zeigen, daß man eine strömungsvergrößernde Kette in $\mathcal{O}(m)$ Zeit bestimmen kann. Damit ergibt sich eine Laufzeit des Algorithmus von Ford und Fulkerson von $\mathcal{O}(m \cdot \beta^*(r_0))$. Diese Laufzeit ist nicht polynomial² und auch nur für kleine Werte $\beta^*(r_0)$ tragbar.

Wir werden aber zeigen, daß man den Algorithmus von Ford und Fulkerson derart modifizieren kann (Algorithmus von Edmonds und Karp), daß er nur $\mathcal{O}(nm)$ Iterationen benötigt, so daß wir mit $\mathcal{O}(nm^2)$ eine polynomiale Gesamtkomplexität erhalten.

²Wir nehmen an, daß die Daten wie üblich binär codiert werden.

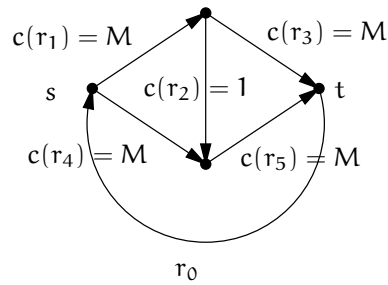


Abbildung 8.2:
Der Algorithmus von Ford und Fulkerson benötigt $\Theta(\beta^*(r_0))$ Iterationen.

Bestimmung strömungsvergrößernder Ketten

Definition 8.21 (Inkrementgraph)

Sei G wie in Definition 8.10 und β eine zulässige Strömung in G . Dann definieren wir den **Inkrementgraphen** $G_\beta = (V, R_\beta, \alpha, \omega)$ bezüglich β wie folgt:

$$R_\beta := \{ +r : r \in R \setminus \{r_0\} \wedge \beta(r) < c(r) \} \cup \{ -r : r \in R \setminus \{r_0\} \wedge \beta(r) > l(r) \},$$

wobei wir wie bei den Ketten $\alpha(+r) := \alpha(r)$ und $\omega(+r) := \omega(r)$ sowie $\alpha(-r) := \omega(r)$ und $\omega(-r) := \alpha(r)$ setzen.

Abbildung 8.3 zeigt ein Beispiel für einen Inkrementgraphen G_β .

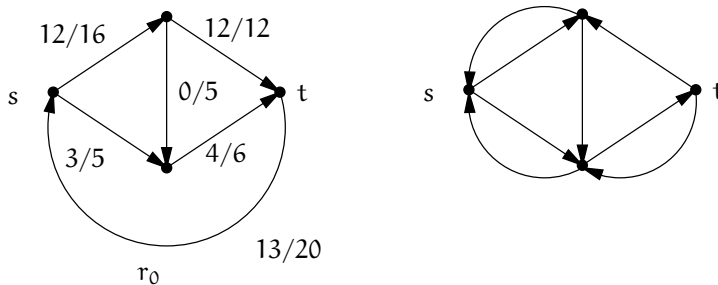


Abbildung 8.3:
Ein Graph G mit Flußwerten $\beta(r)$ und Kapazitäten $c(r)$ ($l \equiv 0$), sowie der zugehörige Inkrementgraph G_β .

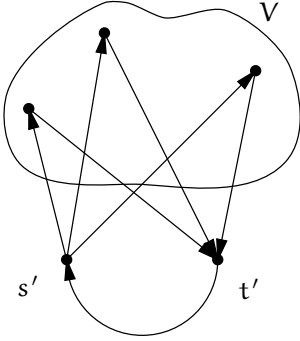
Offenbar entspricht jeder elementare Weg in G_β von s nach t einer strömungsvergrößernden Kette in G . Zur Bestimmung strömungsvergrößernder Ketten können wir daher die kürzeste Wege Algorithmen aus Kapitel 7 benutzen.

Der Graph G_β ist aus G und β in $\mathcal{O}(n + m)$ Zeit berechenbar. Mit Hilfe der Breitensuche (siehe Aufgabe 7.5) können wir daher eine kürzeste³ strömungsvergrößernde Kette von s nach t in $\mathcal{O}(n + m)$ Zeit berechnen oder feststellen, daß keine solche Kette existiert.

Bestimmung einer zulässigen Ausgangsströmung

Im Algorithmus 8.1 haben wir die Kenntnis einer zulässigen Ausgangsströmung β vorausgesetzt. Im Fall $l \equiv 0$ können wir einfach mit der Nullströmung $\beta \equiv 0$ starten.

³d.h. mit minimaler Pfeilanzahl



Der Algorithmus bricht dann nach maximal $\beta^*(r_0)$ Iterationen mit einer maximalen Strömung ab. Wir werden im folgenden zeigen, daß wir das Problem, eine zulässige Strömung im allgemeinen Fall $l \neq 0$ zu finden, mit Hilfe eines „Bootstrapping-Verfahrens“ auf den Fall $l \equiv 0$ zurückführen können.

Sei G wie in Definition 8.10. Wir konstruieren einen Obergraphen $G' = (V', R', \alpha', \omega')$ von G in folgender Weise:

$$\begin{aligned} V' &:= V \cup \{t', s'\} \\ R' &:= R \cup \{(t', s')\} \cup \{(s', v) : v \in V\} \cup \{(v, t') : v \in V\} \end{aligned}$$

Wir setzen $l'(r') := 0$ für alle $r' \in R'$ und definieren obere Kapazitätsschranken c' wie folgt:

$$\begin{aligned} c'(r) &:= c(r) - l(r) && \text{für alle } r \in R \\ c'(s', v) &:= \sum_{r: \omega(r)=v} l(r) && \text{für alle } v \in V \quad (\text{„Mindestzufuß zu } v\text{“}) \\ c'(v, t') &:= \sum_{r: \alpha(r)=v} l(r) && \text{für alle } v \in V \quad (\text{„Mindestabfluß von } v\text{“}) \\ c'(t', s') &:= M := \sum_{v \in V} \sum_{r: \omega(r)=v} l(r) = \sum_{r \in R} l(r) \end{aligned}$$

Lemma 8.22 Sei β eine bezüglich l und c zulässige Strömung in G . Dann ist β' mit

$$\beta'|_R = \beta - l \quad (8.15)$$

$$\beta'|_{R' \setminus R} = c'_{R' \setminus R}, \quad (8.16)$$

eine bezüglich $l' \equiv 0$ und c' zulässige Strömung in G' .

Ist umgekehrt β' eine zulässige Strömung in G' mit den Eigenschaften (8.15) und (8.16), so ist $\beta := \beta'|_R + l$ eine zulässige Strömung in G .

Beweis: Sei $v \in V$. Dann gilt für jede Funktion β' , welche (8.15) und (8.16) erfüllt:

$$\begin{aligned} (\beta')^+(v) &= \beta'(v, t') + \sum_{\substack{r \in R \\ \alpha(r)=v}} \beta'(r) \\ &= c'(v, t') + \sum_{\substack{r \in R \\ \alpha(r)=v}} (\beta(r) - l(r)) \\ &= \sum_{\substack{r \in R \\ \alpha(r)=v}} l(r) + \sum_{\substack{r \in R \\ \alpha(r)=v}} (\beta(r) - l(r)) \\ &= \beta^+(v) \end{aligned}$$

Ferner gilt:

$$\begin{aligned}
 (\beta')^-(v) &= \beta'(s', v) + \sum_{\substack{r \in R \\ \omega(r)=v}} \beta'(r) \\
 &= c'(s', v) + \sum_{\substack{r \in R \\ \omega(r)=v}} \beta'(r) \\
 &= \sum_{\substack{r \in R \\ \omega(r)=v}} l(r) + \sum_{\substack{r \in R \\ \omega(r)=v}} (\beta(r) - l(r)) \\
 &= \beta^-(v)
 \end{aligned}$$

Ist β eine zulässige Strömung in G , so folgt aus $\beta^+(v) = \beta^-(v)$, daß $(\beta')^+(v) = (\beta')^-(v)$ für alle $v \in V$. Ferner ergibt sich

$$(\beta')^+(s') = \sum_{v \in V} \beta'(s', v) = \sum_{v \in V} \sum_{\substack{r \in R \\ \omega(r)=v}} l(r) = \beta'(t', s') = (\beta')^-(s')$$

und

$$\begin{aligned}
 (\beta')^+(t') &= \beta'(t', s') = \sum_{v \in V} \sum_{\substack{r \in R \\ \omega(r)=v}} l(r) = \sum_{v \in V} \sum_{\substack{r \in R \\ \alpha(r)=v}} l(r) \\
 &= \sum_{v \in V} \beta'(v, t') = (\beta')^-(t')
 \end{aligned}$$

Somit ist β' als Strömung nachgewiesen. Wir betrachten nun die Zulässigkeit von β' bezüglich der Kapazitätsschranken $l' \equiv 0$ und c' .

Aus $l \leq \beta \leq c$ folgt $0 \leq \beta - l = \beta'|_R \leq c'|_R$. Da $\beta'|_{R' \setminus R} = c'_{R' \setminus R}$, muß β' eine zulässige Strömung in G' sein.

Sei nun umgekehrt β' eine zulässige Strömung in G' mit den Eigenschaften (8.15) und (8.16). Dann folgt nach den obigen Rechnungen für $v \in V$:

$$\beta^+(v) = (\beta')^+(v) \stackrel{\beta' \text{ Strömung}}{=} (\beta')^-(v) = \beta^-(v).$$

Ergo ist β eine Strömung in G . Aus $0 \leq \beta'(r) \leq c'(r) = c(r) - l(r)$ für alle $r \in R$, folgt dann auch $l(r) \leq \beta(r) \leq c(r)$ für alle $r \in R$, also die Zulässigkeit von β bezüglich l und c . \square

Korollar 8.23 Sei G wie in Voraussetzung 8.7. Dann gibt es in G genau dann eine bezüglich l und c zulässige Strömung β , wenn im Obergraphen G' eine Strömung β' mit maximalem Strömungswert $\beta'(t', s')$ die Gleichung $\beta'(t', s') = \sum_{r \in R} l(r)$ erfüllt.

Beweis: Ist β eine zulässige Strömung, so ist β' wie in Lemma 8.22 eine Strömung in G' mit den gewünschten Eigenschaften.

Ist umgekehrt β' eine Strömung in G' mit $\beta'(t', s') = \sum_{r \in R} l(r)$, so folgt $\beta'|_{R' \setminus R} = c'_{R' \setminus R}$. Nach Lemma 8.22 ist dann $\beta := \beta'|_R + l$ eine bezüglich l und c zulässige Strömung in G . \square

Die Ergebnisse von Korollar 8.23 ermöglichen uns den in Algorithmus 8.2 gezeigten „Bootstrapping-Ansatz“ zur Bestimmung einer maximalen Strömung im Fall, daß $l \not\equiv 0$.

Algorithmus 8.2 Algorithmus zur Gewinnung einer zulässigen Ausgangsströmung und einer maximalen Strömung für den Fall $l \neq 0$.

Input: Ein gerichteter schwach zsh. Graph $G = (V, R, \alpha, \omega)$; Kapazitätsschranken $0 \leq l(r) \leq c(r)$ ($r \in R$), ein ausgezeichnete Pfeil $r_0 \in R$

- 1 $M \leftarrow \sum_{r \in R} l(r)$
 - 2 Konstruiere aus G den Obergraphen G' mit ausgezeichnetem Pfeil $r'_0 = (t', s')$ und Kapazitätsschranken $l' \equiv 0, c'$.
 - 3 Bestimme ausgehend von der Nullströmung β' (etwa mit Hilfe von Algorithmus 8.1) eine Strömung β' in G' mit maximalem Strömungswert $\beta'(t', s')$.
 - 4 **if** $\beta'(t', s') < M$ **then**
 - 5 **return** „Es gibt keine zulässige Strömung in G .“
 - 6 **else**
 - 7 $\beta := \beta'|_R + l$
 - 8 Bestimme ausgehend von der zulässigen Strömung β eine maximale Strömung in G .
 - 9 **end if**
-

Die Korrektheit der Vorgehensweise ergibt sich unmittelbar aus Korollar 8.23 und der Korrektheit des Algorithmus von Ford und Fulkerson (für ganzzahlige bzw. rationale Daten). Als Nebenprodukt erhalten wir auch eine Verallgemeinerung des Ganzzahligkeitssatzes (Satz 8.19), bei dem wir nicht mehr $l \equiv 0$ fordern müssen.

Satz 8.24 (Ganzzahligkeitssatz, zweite Version) Sei G wie in Definition 8.10 und seien die Funktionen l und c ganzzahlig. Falls in G eine zulässige Strömung existiert, dann auch eine maximale Strömung β^* , die ganzzahlig ist. \square

Weiterhin sind wir nun in der Lage, zu zeigen, daß die notwendige Bedingung aus Lemma 8.9 für die Existenz einer zulässigen Strömung tatsächlich auch hinreichend ist.

Satz 8.25 (Hoffman) Sei G wie in Voraussetzung 8.7. Eine notwendige und hinreichende Bedingung für die Existenz einer zulässigen Strömung in G ist, daß

$$c(S, T) \geq l(T, S)$$

für alle Schnitte (S, T) in G gilt.

Beweis: Sei die Bedingung erfüllt. Wir konstruieren aus G den Obergraphen G' wie im Bootstrapping-Ansatz. Wir werden zeigen, daß unter der Voraussetzung des Satzes eine maximale Strömung β' in G' mit

$$\beta'(t', s') = \sum_{v \in V} \sum_{\substack{r \in R \\ \omega(r) = v}} l(r) =: M$$

existiert, d.h. welche die Eigenschaften (8.15) und (8.16) besitzt. Nach Lemma 8.22 folgt damit dann die Existenz einer zulässigen Strömung in G .

Dazu genügt es nach dem Max-Flow-Min-Cut-Theorem zu zeigen, daß

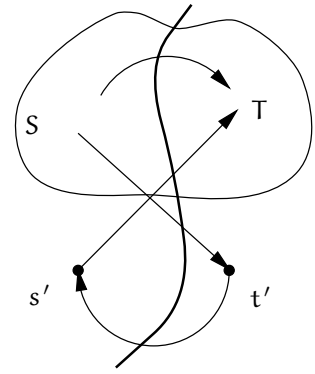
$$c'(S', T') \geq M$$

für jeden Schnitt (S', T') in G' mit $(t', s') \in \sigma^-(S', T')$ gilt. Sei (S', T') ein solcher Schnitt. Wir setzen $S := S' \setminus \{s'\}$, $T := T' \setminus \{t'\}$. Dann ist (S, T) ein Schnitt in G .

Die Kapazität des Schnittes (S', T') berechnet sich wie folgt:

$$\begin{aligned} c'(S', T') &= (c - l)(S, T) + \sum_{v \in S} c'(v, t') + \sum_{v \in T} c'(s', v) \\ &= c(S, T) - l(S, T) + \sum_{v \in S} \sum_{\substack{r \in R \\ \alpha(r) = v}} l(r) + \sum_{v \in T} \sum_{\substack{r \in R \\ \omega(r) = v}} l(r) \\ &= c(S, T) - \sum_{\substack{r \in R \\ \alpha(r) \in S \\ \omega(r) \in T}} l(r) + \sum_{\substack{r \in R \\ \alpha(r) \in S \\ \omega(r) \in T}} l(r) + \sum_{\substack{r \in R \\ \alpha(r) \in S \\ \omega(r) \in S}} l(r) + \sum_{\substack{r \in R \\ \alpha(r) \in S \\ \omega(r) \in T}} l(r) + \sum_{\substack{r \in R \\ \alpha(r) \in T \\ \omega(r) \in T}} l(r) \\ &= c(S, T) + \sum_{\substack{r \in R \\ \alpha(r) \in S \\ \omega(r) \in S}} l(r) + \sum_{\substack{r \in R \\ \alpha(r) \in S \\ \omega(r) \in T}} l(r) + \sum_{\substack{r \in R \\ \alpha(r) \in T \\ \omega(r) \in T}} l(r) + \sum_{\substack{r \in R \\ \alpha(r) \in T \\ \omega(r) \in S}} l(r) - \sum_{\substack{r \in R \\ \alpha(r) \in T \\ \omega(r) \in S}} l(r) \\ &= c(S, T) + \sum_{r \in R} l(r) - l(T, S) \\ &= M + \underbrace{c(S, T) - l(T, S)}_{\geq 0} \\ &\geq M. \end{aligned}$$

Dies beendet den Beweis. \square



Der Algorithmus von Edmonds und Karp

Wir betrachten nun die folgende Variante des Algorithmus von Ford und Fulkerson: Wir erhöhen in jedem Schritt die Strömung entlang einer kürzesten strömungsvergrößernden Kette. Wie bereits bemerkt, können wir in jeder Iteration eine solche Kette in $\mathcal{O}(n + m)$ Zeit finden. Den entstehenden Algorithmus nennen wir den Algorithmus von Edmonds und Karp.

Lemma 8.26 Für eine Strömung β bezeichne $\delta_\beta(s, v)$ den Abstand von s nach v in G_β . Für jedes $v \in V \setminus \{s, t\}$ ist dann $\delta_\beta(s, v)$ während des Algorithmus von Edmonds und Karp monoton wachsend.

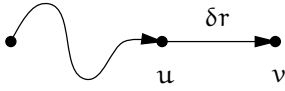
Beweis: Wir nehmen an, daß für eine Ecke $v \in V \setminus \{s, t\}$ der Abstand $\delta_\beta(s, v)$ bei einer Flußerhöhung abnimmt, und führen diese Annahme zum Widerspruch.

Sei β der Fluß vor der Erhöhung und β' der Fluß unmittelbar nachher. Unsere Widerspruchsannahme besagt dann, daß

$$\delta_{\beta'}(s, v) < \delta_\beta(s, v). \quad (8.17)$$

O.B.d.A. sei v bereits so gewählt, daß $\delta_{\beta'}(s, v)$ minimal ist unter allen Ecken v , welche (8.17) erfüllen, d.h

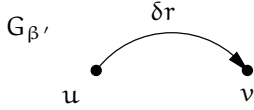
$$\delta_{\beta'}(s, u) < \delta_{\beta'}(s, v) \Rightarrow \delta_\beta(s, u) \leq \delta_{\beta'}(s, u). \quad (8.18)$$



Sei $w = (\delta_1 r_1, \dots, \delta_k r_k, \delta r)$ ein kürzester Weg von s nach v in $G_{\beta'}$ mit Spur $s(w) = [s, \dots, u, v]$, wobei $\delta_i, \delta \in \{+, -\}$. Da wir als Gewichtsfunktion auf den Pfeilen die Funktion betrachten, die allen Pfeilen den Wert 1 zuweist, ist w elementar. Es gilt $\alpha(\delta r) = u$ und $\omega(\delta r) = v$.

Wir haben $\delta_{\beta'}(s, u) = \delta_{\beta}(s, v) - 1$ und folglich nach (8.18)

$$\delta_{\beta}(s, u) \leq \delta_{\beta'}(s, u). \tag{8.19}$$



1. Fall: $\delta r \in G_{\beta}$.

Dann ist $\delta r \in G_{\beta}$ mit $\alpha(\delta r) = u$ und $\omega(\delta r) = v$. Nach Lemma 7.5 gilt dann:

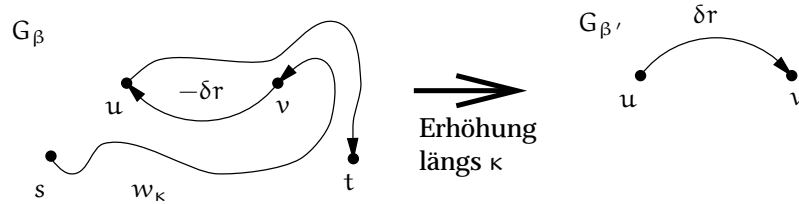
$$\delta_{\beta}(s, v) \leq \delta_{\beta}(s, u) + 1 \leq \delta_{\beta'}(s, u) + 1 = \delta_{\beta'}(s, v)$$

im Widerspruch zu (8.17).

2. Fall: $\delta r \notin G_{\beta}$.

Wir wissen, daß $\delta r \in G_{\beta'}$. Also muß wegen $\delta r \notin G_{\beta}$ dann $-\delta r \in G_{\beta}$ gewesen sein und die letzte strömungsvergrößernde Kette κ muß $-\delta r$ benutzt haben. Der entsprechende Weg w_{κ} in G_{β} besitzt also die Form $w_{\kappa} = (\delta'_1 r'_1, \dots, -\delta r, \delta'_p r'_p)$. Er ist nach unserer Wahl ein kürzester Weg von s nach t in G_{β} . Abbildung 8.4 veranschaulicht die Situation.

Abbildung 8.4: Der Weg w_{κ} in G_{β} benutzt den Pfeil $-\delta r \in G_{\beta}$. Durch Erhöhen der Strömung längs κ wird $-\delta r$ durch δr ersetzt.



Da $\alpha(-\delta r) = v$ und $\omega(-\delta r) = u$ ist, folgt nach Lemma 7.5, daß

$$\delta_{\beta}(s, v) = \delta_{\beta}(s, u) - 1 \leq \delta_{\beta'}(s, u) - 1 = \delta_{\beta'}(s, v) - 2 < \delta_{\beta'}(s, v).$$

Dies ist ein Widerspruch. □

Satz 8.27 Sei G wie in Definition 8.10 mit ganzzahligen, rationalen oder reellen⁴ Kapazitäten. Der Algorithmus von Edmonds und Karp terminiert ausgehend von einer zulässigen Strömung nach $\mathcal{O}(nm)$ Iterationen mit einer maximalen Strömung. Die Gesamtkomplexität des Algorithmus ist $\mathcal{O}(nm^2)$.

Beweis: Bei jeder Änderung der Strömung wird die Strömung auf einem Pfeil r entweder auf $c(r)$ erhöht oder auf $l(r)$ erniedrigt. Wir nennen einen Pfeil δr in einem Inkrementgraphen G_{β} kritisch, wenn δr auf der strömungsvergrößernden Kette κ liegt und seine Residualkapazität gleich der von κ ist. Nach der Erhöhung längs κ verschwindet δr aus dem Inkrementgraphen und wird durch $-\delta r$ ersetzt. In jeder Iteration ist mindestens ein Pfeil des Inkrementgraphen kritisch.

Sei $\delta r \in R$ mit $\alpha(\delta r) = u$ und $\omega(\delta r) = v$. Wir schätzen nun ab, wie oft δr in einer Iteration kritisch sein kann.

⁴Im Fall von reellen Kapazitäten nehmen wir an, daß wir arithmetische Operationen auf reellen Zahlen ebenfalls in konstanter Zeit ausführen können

Da wir die Strömung immer entlang kürzester Wege in G_β erhöhen, gilt, wenn δr das erste Mal kritisch ist, nach Lemma 7.5

$$\delta_\beta(s, v) = \delta_\beta(s, u) + 1.$$

Danach verschwindet δr aus dem Inkrementgraphen, während $-\delta r$ erscheint. Wir betrachten nun den Fluß β' , nach dessen Erhöhung längs einer strömungsvergrößernden Kette κ' wieder δr im Inkrementgraphen erscheint. Die Kette κ' muß $-\delta r$ benutzen und, da wir immer entlang kürzester Ketten erhöhen, folgt mit Lemma 7.5 daher

$$\delta_{\beta'}(s, u) = \delta_{\beta'}(s, v) + 1.$$

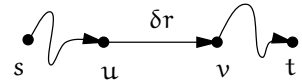
Nach Lemma 8.26 gilt $\delta_\beta(s, v) \leq \delta_{\beta'}(s, v)$ und somit

$$\delta_{\beta'}(s, u) = \delta_{\beta'}(s, v) + 1 \geq \delta_\beta(s, v) + 1 = \delta_\beta(s, u) + 2.$$

Zwischen zwei Zeitpunkten, zu denen δr kritisch wird, erhöht sich somit der Abstand von s zu u im Inkrementgraphen um mindestens 2. Am Anfang ist dieser Abstand mindestens 1. Er kann jedoch niemals mehr als $n - 1$ betragen. Also ist δr höchstens $\mathcal{O}(n)$ mal kritisch. Da wir $\mathcal{O}(m)$ Pfeile im Inkrementgraphen haben, von denen jede höchstens $\mathcal{O}(n)$ mal kritisch werden kann, erfolgen also höchstens $\mathcal{O}(nm)$ Iterationen.

Der Abbruch des Algorithmus kann nur dann erfolgen, wenn für die aktuelle Strömung β entweder keine strömungsvergrößernde Kette existiert oder $\beta(r_0) = c(r_0)$ gilt. Nach Lemma 8.17 ist dann β eine maximale Strömung. Folglich terminiert der Algorithmus von Edmonds und Karp nach $\mathcal{O}(nm)$ Iterationen mit einer maximalen Strömung.

Wir haben bereits festgestellt, daß wir in jeder Iteration eine kürzeste strömungsvergrößernde Kette in $\mathcal{O}(n + m)$ Zeit durch Breitensuche finden können. Somit ist die Gesamtkomplexität des Algorithmus $\mathcal{O}(nm^2)$. \square



Der Pfeil δr liegt auf einem kürzestem Weg von s nach t in G_β .

8.5 Kombinatorische Anwendungen

Der Heiratssatz

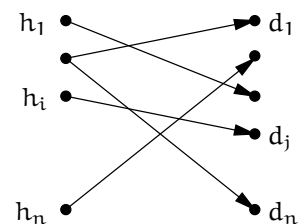
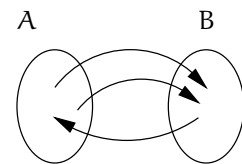
Definition 8.28 (Bipartiter Graph)

Ein gerichteter Graph $G = (V, R, \alpha, \omega)$ heißt **bipartit**, wenn es eine Partition $V = A \cup B$, $A \cap B = \emptyset$ der Eckenmenge V gibt, so daß für jeden Pfeil $r \in R$ gilt:

$$(\alpha(r) \in A \wedge \omega(r) \in B) \vee (\omega(r) \in A \wedge \alpha(r) \in B)$$

Analog bezeichnet man einen ungerichteten Graphen $G = (V, E, \gamma)$ als **bipartit**, wenn es eine Partition $V = A \cup B$, $A \cap B = \emptyset$ gibt, so daß für $e \in E$ gilt: $\gamma(e) = \{a, b\}$, wobei $a \in A$ und $b \in B$.

Gegeben sei eine Menge von Herren $H = \{h_1, \dots, h_n\}$ und eine Menge $D = \{d_1, \dots, d_n\}$ von Damen ($H \cap D = \emptyset$). Ferner sei ein bipartiter „Sympatiegraph“ $G = (H \cup D, R)$ gegeben, wobei $(h_i, d_j) \in R$ bedeutet, daß die Dame d_j dem Herren h_i gefällt. Die Frage ist nun, ob es eine Paarung der Damen und Herren gibt, so daß jeder Herr genau eine Dame heiraten kann, die ihm auch gefällt.



Definition 8.29

Ein **Matching** in einem Graphen $G = (V, R, \alpha, \omega)$ ist eine Teilmenge $M \subseteq R$ der Pfeilmenge, so daß keine zwei Pfeile $r, r' \in M$ mit $r \neq r'$ inzident sind.

Ein Matching M heißt **perfekt**, wenn jede Ecke $v \in V$ mit einem Pfeil $r \in M$ inzidiert.

Wir können das Heiratsproblem somit als das Problem formulieren zu bestimmen, ob ein bipartiter Graph ein perfektes Matching besitzt. Dies ist in unserem Fall ein Matching M mit $|M| = n$.

Satz 8.30 (Heiratsatz, Hall 1935) Das Heiratsproblem ist genau dann lösbar, wenn

$$|V(D')| \geq |D'| \quad \text{für alle } D' \subseteq D, \tag{8.20}$$

wobei

$$V(D') := \{h \in H : (h, d) \in R \text{ für ein } d \in D'\}.$$

Beweis: Offenbar ist die Bedingung in (8.20) notwendig. Wir zeigen nun die Umkehrung.

Wir erweitern den Sympatiegraphen in folgender Weise. Sei $G' = (V', R')$ mit

$$V' := V \cup \{s, t\}$$

$$R' := R \cup \{r_0 := (t, s)\} \cup \{(s, h) : h \in H\} \cup \{(d, t) : d \in D\}.$$

Wir definieren Kapazitäten $l' \equiv 0$ sowie $c'(r) = 1$ für alle $r \neq r_0$ sowie $c'(r_0) = n$. Abbildung 8.5 veranschaulicht die Konstruktion.

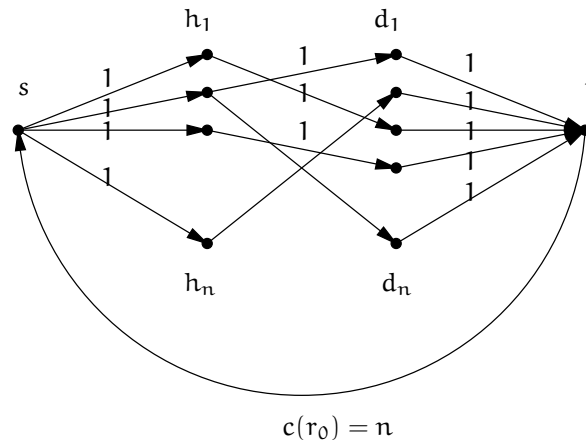


Abbildung 8.5: Konstruktion eines Flußnetzwerkes für das Heiratsproblem.

Jede ganzzahlige zulässige Strömung β in G' induziert ein Matching M_β in G durch

$$M_\beta := \{r \in R : \beta(r) = 1\}.$$

Weitere Anwendungen

Wir nennen im folgenden noch ohne Beweis weitere wichtige Sätze der Graphentheorie, die sich mit Hilfe des Max-Flow-Min-Cut-Theorems beweisen lassen.

Satz 8.31 (Menger, 1927) Sei $G = (V, R, \alpha, \omega)$ ein endlicher gerichteter Graph und $s, t \in V$ nichtadjazente Ecken von G . Dann ist die Maximalzahl eckendisjunkter Wege von s nach t gleich der minimalen Mächtigkeit einer s und t trennenden Eckenmenge. \square

Definition 8.32 (Eckenüberdeckung)

Sei G ein (gerichteter oder ungerichteter) Graph mit Eckenmenge V . Eine Teilmenge $C \subseteq V$ heißt **Eckenüberdeckung** (vertex cover) in G , wenn jeder Pfeil (jede Kante) in G mit mindestens einer Ecke aus C inzidiert.

Satz 8.33 (König, Egerváry 1931) Sei G ein endlicher bipartiter Graph. Dann ist die maximale Mächtigkeit eines Matchings in G gleich der minimalen Mächtigkeit einer Eckenüberdeckung in G . \square

8.6 Kostenminimale Strömungen

In diesem Abschnitt wenden wir uns Strömungen zu, bei denen Kosten ins Spiel kommen. Besonders wichtig ist dabei die Suche nach einer kostenminimalen Strömung. Am Ende stehen einige Anwendungen, die mit Hilfe von kostenminimalen Strömungen gelöst werden können.

Definition 8.34 (Strömungskosten) Sei $G = (V, R, \alpha, \omega)$ ein endlicher Graph, $\beta: R \rightarrow \mathbb{R}_0^+$ eine (nicht notwendig zulässige) Strömung auf G , und $k: R \rightarrow \mathbb{R}_0^+$ eine Pfeilbewertung, die sogenannte **Kostenfunktion**.

Dann sind durch

$$k(\beta) := \sum_{r \in R} k(r) \cdot \beta(r)$$

die **Kosten der Strömung** β gegeben.

Haben wir einen Graphen mit einem ausgezeichneten Pfeil r_0 , und geben auf r_0 einen Strömungswert F vor, so gibt es im allgemeinen mehrere Strömungen β mit $\beta(r_0) = F$, die also den Strömungswert F auf r_0 realisieren. Wir untersuchen nun das Problem, unter allen zulässigen Strömungen β mit $\beta(r_0) = F$ eine Strömung β^* mit geringsten Strömungskosten $k(\beta^*)$ zu finden.

Der Einfachheit halber setzen wir im folgenden voraus, daß die unteren Kapazitätsschranken $l \equiv 0$ erfüllen. Für den allgemeinen Fall $l \neq 0$ wird die Darstellung komplizierter, der gezeigte Rechengang ist jedoch prinzipiell derselbe. Weiter wollen wir annehmen, daß zum ausgezeichneten Pfeil $r_0 = (t, s)$ kein paralleler oder inverser Pfeil existiert. Dies wird erreicht, indem ein solcher paralleler oder inverser Pfeil ersetzt wird durch zwei hintereinanderhängende Pfeile, die über eine neu eingeführt Ecke laufen. Die Kapazitäten der neuen Pfeile entsprechen denen des ersetzten, und die Kosten werden so gewählt, daß die Kostensumme der neuen Pfeile gleich den Kosten des ersetzten Pfeiles ist.

Definition 8.35 (Inkrementgraph, assoziierte Strömung) Sei $G = (V, R)$ ein einfacher Graph, $r_0 = (t, s) \in R$ ein ausgezeichneter Pfeil, weiter $c: R \rightarrow \mathbb{R}_0^+$ obere Kapazitätsfunktion, $k: R \rightarrow \mathbb{R}$ eine Kostenfunktion, $\beta: R \rightarrow \mathbb{R}_0^+$ eine zulässige Strömung in G .

Für jedes $r \in R$ sei r^+ ein Pfeil mit $\alpha(r^+) = \alpha(r)$ und $\omega(r^+) = \omega(r)$; r^- sei ein zu r^+ inverser Pfeil. Setze die Kapazitäten $c_\beta(r^+) := c(r) - \beta(r)$ und $c_\beta(r^-) := \beta(r)$. Der **Inkrementgraph** $G_\beta = (V, R_\beta, \alpha, \omega)$ ist dann durch

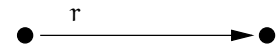
$$R_\beta := \{r^+ \mid r \in R \setminus \{r_0\} \wedge c_\beta(r^+) > 0\} \cup \{r^- \mid r \in R \setminus \{r_0\} \wedge c_\beta(r^-) > 0\}$$

definiert, wobei die Kosten $k_\beta(r^+) = k(r)$ und $k_\beta(r^-) := -k(r)$ gewählt werden.

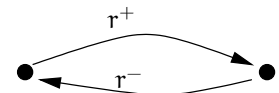
Sei γ eine zweite zulässige Strömung in G . Die zu γ assoziierte Strömung γ_β im Inkrementgraphen G_β ist dann definiert durch

$$\begin{aligned} & \gamma_\beta(r^+) := \gamma(r) - \beta(r) \\ \text{und} & \gamma_\beta(r^-) := 0, & \text{falls } \gamma(r) - \beta(r) \geq 0 \\ & \gamma_\beta(r^+) := 0 \\ \text{und} & \gamma_\beta(r^-) := \beta(r) - \gamma(r), & \text{falls } \gamma(r) - \beta(r) < 0 \end{aligned}$$

Zwischen einem Graphen mit zulässiger Strömung und dem entsprechenden Inkrementgraphen besteht ein wichtiger Zusammenhang. Wo der Ausgangsgraph Änderungen an der Strömung nach oben oder nach unten zuläßt, finden sich im Inkrementgraphen Pfeile entsprechender Kapazität. In diesem Sinn wird die Dynamik des Ausgangsgraphen mit vorhandener Strömung abgebildet auf den statischen Fall des Inkrementgraphen.



$$\begin{aligned} c(r) &= 4 \\ k(r) &= 7 \\ \beta(r) &= 3 \end{aligned}$$



$$\begin{aligned} c_\beta(r^+) &= 1 \\ k_\beta(r^+) &= +7 \\ c_\beta(r^-) &= 3 \\ k_\beta(r^-) &= -7 \end{aligned}$$

Ausgangsgraph mit Strömung (oben) und dazugehöriges Pfeilpaar des Inkrementgraphen (unten)

Lemma 8.36 Sei $G = (V, R)$ einfacher Graph mit oberen Kapazitäten c . Sei β eine zulässige Strömung in G . Dann gilt: γ ist genau dann eine zulässige Strömung in G , wenn γ_β zulässig in G_β ist.

Beweis: Übung 8.2a. □

Lemma 8.37 Sei $G = (V, R)$ einfacher Graph mit oberen Kapazitäten c und Strömungskosten k , β eine zulässige Strömung in G , G_β der Inkrementgraph mit Strömungskosten k_β . Dann gilt $k_\beta \gamma_\beta = k\gamma - k\beta$, d. h.

$$k_\beta(r^+) \cdot \gamma_\beta(r^+) + k_\beta(r^-) \cdot \gamma_\beta(r^-) = k(r) \cdot \gamma(r) - k(r) \cdot \beta(r).$$

Beweis: Übung 8.2b. □

Diesem Lemma kann man entnehmen, daß die Kosten einer Strömung γ_β im Inkrementgraphen identisch sind mit der Änderung der Kosten, die im Ausgangsgraphen beim Übergang von der Strömung β zu γ auftritt.

Satz 8.38 (Kriterium für kostenminimale Strömungen) Sei $G = (V, R)$ einfacher Graph mit oberen Kapazitäten c und Strömungskosten k , r_0 ein ausgezeichneter Pfeil, $F \geq 0$ ein vorgegebener Strömungswert, β eine zulässige Strömung mit $\beta(r_0) = F$.

Dann gilt: β ist genau dann kostenminimal, wenn der Inkrementgraph G_β keinen Kreis negativer Länge bezüglich der Kostenfunktion k_β aufweist.

Beweis: Der Beweis des Satzes erfolgt in Übung 8.2c. □

Mit Satz 8.38 ergibt sich eine einfache Möglichkeit, eine kostenminimale Strömung zu ermitteln. Der Algorithmus von Klein (vgl. Algorithmus 8.3) startet mit einer zulässigen Strömung, die den Wert F auf r_0 realisiert. Dann werden (durch Addition von Kreisströmungen mit negativen Kosten) die Strömungskosten solange erniedrigt, bis im Inkrementgraphen kein Kreis negativer Länge mehr existiert. Die dann aktuelle Strömung ist kostenminimal, also ist der Algorithmus korrekt.

Das Verfahren terminiert: In jeder Iteration wird im Inkrementgraphen mindestens ein Pfeil negativer Kosten gesättigt. Dieser Pfeil wird somit aus dem Inkrementgraphen entfernt. Da keine Pfeile negativer Kosten neu entstehen, kann es höchstens $O(|R|)$ viele Iterationen geben.

In Algorithmus 8.3 ist nicht näher spezifiziert, auf welche Weise Ausgangsströmung und negative Kreise gefunden werden sollen. Dazu werden in [AMO93] verschiedene Verfahren vorgestellt und hinsichtlich ihrer Laufzeit verglichen.

Algorithmus 8.3 Algorithmus von Klein zur Bestimmung einer kostenminimalen Strömung.

Input: Ein gerichteter einfacher Graph $G = (V, R)$;
 Kapazitätsschranken $c(r) \geq 0$ und
 Strömungskosten $k(r)$ ($r \in R$),
 ein ausgezeichneter Pfeil $r_0 \in R$;
 ein Strömungswert F mit $0 \leq F \leq c(r_0)$

- 1 wähle zulässige Strömung β in G mit $\beta(r_0) = F$
 Abbruch, falls keine solche Strömung existiert
- 2 **loop**
- 3 bestimme Inkrementgraph G_β mit Kostenfunktion k_β
- 4 ermittle einen Kreis z negativer k_β -Länge in G_β
 verlasse die Schleife, falls kein solcher Kreis existiert
- 5 setze $\varepsilon := \min_{r \in z} c_\beta(r)$
 sei $\beta^{(\varepsilon)}$ die Kreisströmung entlang z
- 6 setze $\beta \leftarrow \beta + \beta^{(\varepsilon)}$
- 7 **end loop**

Output: β ist kostenminimale Strömung mit $\beta(r_0) = F$

Anwendungen

Im folgenden wird erläutert, wie sich ein Transportproblem mit Hilfe einer kostenminimalen Strömung in einem kapazitierten Graphen lösen lässt.

Wir betrachten ein Gut, das an m Produktionsstätten Q_1, \dots, Q_m hergestellt und an n Orten/Regionen S_1, \dots, S_n nachgefragt wird. Die Produktionskapazität der Stätte Q_i ist gegeben durch a_i (Einheit Menge/Zeit), der Preis für die Herstellung beträgt p_i (Einheit Kosten/Menge),

die geforderte Nachfrage in einer Region S_j beträgt b_j (Einheit Menge/Zeit). Weiter ist ein Transportnetz, etwa ein Straßennetz, in Form eines gerichteten Graphen gegeben, auf dessen Pfeilen die Werte c die maximale Transportkapazität (Einheit Menge/Zeit), und die Werte k die Transportkosten auf dem betreffenden Straßenstück angeben. Produktions- und Nachfrageorte finden sich als Ecken in dem Graphen wieder.

Wir führen nun als neue Ecken eine Superquelle Q und eine Supersenke S in den Transportgraphen ein. Für jede Produktionsstätte Q_i wird ein Pfeil (Q, Q_i) der Kapazität a_i und der Kosten p_i hinzugefügt. Jede Nachfrageregion S_j wird mit einem Pfeil (S_j, S) der unteren Kapazität $l = b_j$ mit der Supersenke verbunden. Ein weiterer Pfeil $r_0 = (S, Q)$ verbindet Senke mit Quelle, seine obere Kapazität sei genügend groß gewählt, etwa $c(r_0) = \sum_i a_i$.

Eine zulässige Strömung korrespondiert dann mit einem Transportplan, der die Nachfrage befriedigt, ohne Produktionsstätten und Transportwege zu überlasten. Die Kosten der Strömung geben dabei die Produktions- und Transportkosten des Planes an. Offenbar besteht die optimale Lösung des Transportproblem in einer kostenminimalen Strömung.

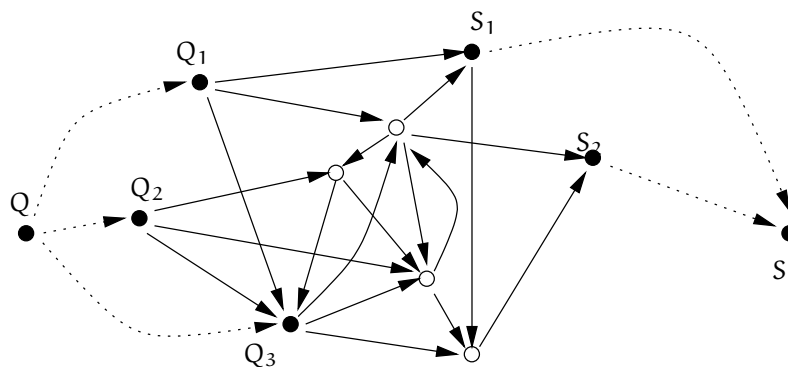


Abbildung 8.7: Beispiel für ein Transportproblem

Das Modell ist noch erweiterbar. So läßt sich etwa ein Lager modellieren durch zwei Ecken l_{ein} und l_{aus} (Einfahrt und Ausfahrt), die durch einen Pfeil $(l_{\text{ein}}, l_{\text{aus}})$ verbunden sind. Kapazität und Kosten auf dem Pfeil spiegeln dabei Lagerkapazität und Lagerkosten (Einheit Kosten/Menge) wieder.

Übungsaufgaben

Aufgabe 8.1 – Zerlegung von Strömungen

Sei $G = (V, R, \alpha, \omega)$ ein Graph, $c: R \rightarrow \mathbb{R}_0^+$ obere Kapazitätsfunktion. Es sei $\beta: R \rightarrow \mathbb{R}_0^+$ eine beliebige zulässige Strömung.

Zeigen Sie: Es gibt eine Zerlegung $\beta = \beta_1 + \dots + \beta_k$ mit $k \leq |R|$, so daß jedes β_i eine zulässige Strömung entlang eines Kreises in G ist.

Hinweis: Eine solche Zerlegung kann konstruktiv gefunden werden.

Aufgabe 8.2 – Kostenminimale Strömung

Sei $G = (V, R)$ ein einfacher Graph, $r_0 = (t, s) \in R$ ein ausgezeichneter Pfeil, $c: R \rightarrow \mathbb{R}_0^+$ obere Kapazitätsfunktion, $k: R \rightarrow \mathbb{R}$ eine Kostenfunktion. Es sei $\beta: R \rightarrow \mathbb{R}_0^+$ eine zulässige Strömung in G . Mit G_β sei der Inkrementgraph bezeichnet.

Sei γ eine zweite Strömung in G , und sei γ_β die assoziierte Strömung im Inkrementgraphen.

- a. Zeigen Sie: γ ist genau dann zulässige Strömung in G , wenn γ_β zulässige Strömung in G_β ist. Geben Sie an, wie eine Strömung γ im Ausgangsgraphen G aus β und der Strömung γ_β im Inkrementgraphen G_β berechnet wird.

- b. Zeigen Sie die Beziehung $k_\beta \gamma_\beta = k\gamma - k\beta$, d. h. genauer

$$k_\beta(r^+) \cdot \gamma_\beta(r^+) + k_\beta(r^-) \cdot \gamma_\beta(r^-) = k(r) \cdot \gamma(r) - k(r) \cdot \beta(r).$$

Sei $F > 0$ ein vorgegebener Flußwert. Eine Strömung β heißt *Strömung zum Flußwert F* , wenn $\beta(r_0) = F$ gilt.

- c. Sei β eine Strömung zum Flußwert F . Zeigen Sie: β ist genau dann kostenminimal, wenn in G_β kein negativer Kreis bezüglich der Kosten besteht.

Aufgabe 8.3 – Potentialdifferenz, Topologische Sortierung

Sei $G = (V, R, \alpha, \omega)$ ein Graph. Eine Eckenbewertung $\pi: V \rightarrow \mathbb{R}$ wird auch *Potential* genannt. Eine Pfeilbewertung $\Delta: R \rightarrow \mathbb{R}$ heißt *Potentialdifferenz*, wenn es ein geeignetes Potential π gibt, so daß

$$\Delta(r) = \pi(\omega(r)) - \pi(\alpha(r)) \quad \text{für alle } r \in R.$$

- a. Sei β eine beliebige Strömung, Δ eine beliebige Potentialdifferenz in G . Zeigen Sie:

$$\beta \cdot \Delta := \sum_{r \in R} \beta(r) \cdot \Delta(r) = 0.$$

Hinweis: Nutzen Sie $\sum_{r \in R} = \sum_{v \in V} \sum_{r \in R: \alpha(r)=v} = \sum_{v \in V} \sum_{r \in R: \omega(r)=v}$.

- b. Zeigen Sie, daß die folgenden Aussagen äquivalent sind.

- (i) G ist kreisfrei.
- (ii) Es gibt eine Potentialdifferenz Δ in G mit $\Delta(r) > 0$ für alle $r \in R$. (Das entsprechende Potential wird dann auch *topologische Sortierung* der Eckenmenge genannt.)
- (iii) Ist β Strömung in G mit $\beta(r) \geq 0$ für alle $r \in R$, so ist β die Nullströmung.

Kapitel 9

Planare Graphen

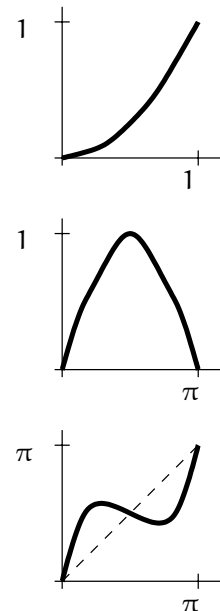
Gegenstand dieses Kapitels sind planare Graphen. Wir werden sehen, daß für die Eigenschaft der Planarität die Unterscheidung in gerichtete und ungerichtete Graphen nicht wesentlich ist. Um die Darstellung nicht zu komplizieren, werden die Ergebnisse dieses Kapitels daher nur für ungerichtete Graphen formuliert. Die Übertragung für gerichtete Graphen ist ohne weiteres möglich; wo dies nicht der Fall ist, wird im Text eigens darauf hingewiesen.

9.1 Einbettungen

Definition 9.1 (Planarer Graph) Sei G ein Graph. Dann heißt G **planar**, wenn seine Ecken als Punkte und seine Kanten als Linienstücke in die (euklidische) Ebene so eingebettet werden können, daß sich zwei Linienstücke nur in Punkten schneiden, die adjazente Ecken repräsentieren.

Unter einem **Linienstück** wird dabei eine Abbildung $\varphi: [0, 1] \rightarrow \mathbb{R}^2$ verstanden, die stetig und in jedem inneren Punkt differenzierbar ist. Oft wird hier auch zugelassen, daß die Abbildung in einer endlichen Menge von Punkten nicht differenzierbar ist, also das Linienstück endlich viele „Knicke“ enthalten darf. Farý hat 1948 gezeigt, daß allgemeine Linienstücke überhaupt nicht erforderlich sind. Vielmehr gibt es zu jedem planaren Graphen auch eine Einbettung, so daß alle Linienstücke gerade Stecken darstellen.

Für einen planaren Graphen gibt es in der Regel auch Einbettungen, die nicht überschneidungsfrei sind. In der Definition wird daher nur gefordert, daß eine überschneidungsfreie Einbettung existiert.



Beispiele für Linienstücke
 $\varphi_1(t) = (t, t^2)$,
 $\varphi_2(t) = (\pi t, \sin \pi t)$,
 $\varphi_3(t) = (\pi t, t + \sin 2\pi t)$.

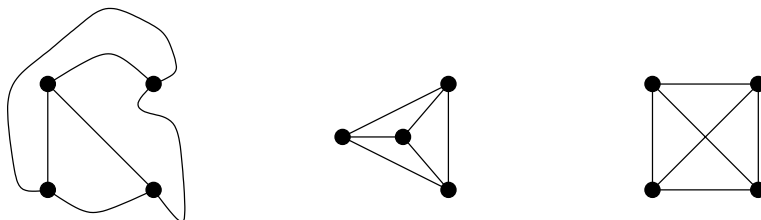
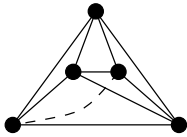
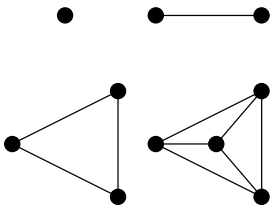
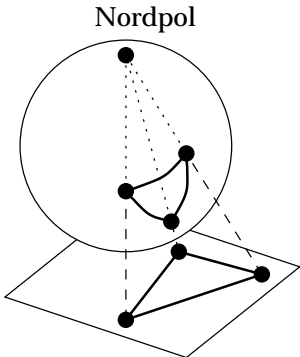


Abbildung 9.1: Der K_4 ist planar. Zum Beweis zwei überschneidungsfreie Einbettungen. Rechts eine nicht überschneidungsfreie Einbettung.



Einbettungen von K_1 bis K_5 . Beim K_5 kann die letzte (gestrichelte) Kante nicht mehr überschneidungsfrei gezeichnet werden.



Man erkennt durch Ausprobieren, daß die vollständigen Graphen K_1 bis K_4 planar sind. Da durch Weglassen von Linienstücken keine neuen Überschneidungen erzeugt werden, ist jeder Partialgraph eines planaren Graphen selbst wieder planar. Mit Sätzen der Mathematik läßt sich ferner zeigen, daß bei gegebener planarer Einbettung zu jedem Linienstück auch endlich viele Parallele überschneidungsfrei gezeichnet werden können, ebenso endlich viele Schlingen an einer Ecke. Damit folgt:

Lemma 9.2 *Alle Graphen mit $|V| \leq 4$ Ecken sind planar.*

Beim Versuch, eine überschneidungsfreie Einbettung des K_5 zu finden, stellen wir fest, daß die letzte Kante nicht mehr ohne Überschneidung gezeichnet werden kann. Ein wesentliches Ergebnis der folgenden Überlegungen wird sein, daß dies nicht durch eine speziell gewählte Einbettung behoben werden kann, sondern daß der K_5 tatsächlich nicht planar ist.

Die Linienstücke einer gegebenen planaren Einbettung bilden geschlossene Kurven (die einfachen Zykeln des Graphen entsprechen) und teilen die Zeichenebene in Regionen auf. Man nennt eine beschränkte Region auch ein **Gebiet**. Man kann zeigen, daß für einen zusammenhängenden Graphen jedes Gebiet einfach zusammenhängend ist, also kein „Loch“ enthält. Ferner ist für endliche Graphen genau eine der Regionen unbeschränkt.

Bei gegebenem planaren Graphen ist eine Einbettung nicht eindeutig festgelegt. In der Wahl, welcher der Zykeln im Graphen die unbeschränkte Region in einer Einbettung berandet, ist man sogar frei. Um dies einzusehen, betrachten wir eine beliebige Einbettung E_1 und einen Zykel z , der zu einer beschränkten Region in E_1 gehört. Dann kann daraus eine Einbettung E_2 gewonnen werden, bei der z zu der unbeschränkten Region gehört.

Dazu bedienen wir uns der *stereographischen Projektion*: Anschaulich wird auf die Zeichenebene (mit der planaren Einbettung) eine Kugel gelegt. Der Berührungspunkt mit der Zeichenebene heißt Südpol, der gegenüberliegende Punkt Nordpol. Ein Strahl vom Nordpol zu einem Punkt der Zeichenebene trifft die Kugeloberfläche in genau einem Punkt. Dadurch wird eine Bijektion zwischen der gesamten Zeichenebene und der Kugeloberfläche (mit Ausnahme des Nordpols) festgelegt.

Die beschriebene Bijektion hat „schöne Eigenschaften“. Sie verzerrt zwar das Urbild, aber topologische Eigenschaften (Indizes usw.) bleiben erhalten. Also ist das Bild einer planaren Einbettung auf der Kugel auch wieder eine überschneidungsfreie Einbettung. Das Bild der unbeschränkten Region ist auf der Kugeloberfläche die beschränkte Region, die um den Nordpol liegt.

Nun werde die Kugel so gerollt, daß die zu z gehörende Region den Nordpol enthält. Bei der Projektion von der Kugeloberfläche auf die Zeichenebene entsteht dann die gesuchte planare Einbettung, bei der z die unbeschränkte Region berandet. Zu beachten ist, daß die Projektion für den Nordpol nicht definiert ist, d. h. die Kugel ist immer so zu drehen, daß am Nordpol kein Punkt oder Linienstück der Einbettung zu liegen kommt.

9.2 Die Eulersche Polyederformel

Euler hat folgende Beziehung zwischen der Anzahl der Ecken, Kanten und Regionen einer Einbettung eines planaren Graphen gezeigt:

Satz 9.3 (Eulersche Polyederformel) *Jeder endliche zusammenhängende planare Graph mit n Ecken, m Kanten und f Regionen erfüllt bei jeder planaren Einbettung*

$$n - m + f = 2.$$

Bemerkung: Der Name „Polyederformel“ rührt daher, daß die Ecken und das Kantennetz jedes Polyeders im \mathbb{R}^3 einen planaren Graphen bilden.

Beweis: Man überlegt zunächst, daß jeder endliche Baum planar ist. Eine beschränkte Region in der Einbettung würde einen einfachen Zykel im Baum implizieren. Also gibt es in der Einbettung nur die unbeschränkte Region, und es gilt $f = 1$. Wegen $m = n - 1$ folgt dann die Gültigkeit der Eulerschen Formel für den Baum.

Sei nun G planar und eine überschneidungsfreie Einbettung gegeben. Bestimme zunächst einen spannenden Baum T von G . Füge nun iterativ die restlichen Kanten zum Baum hinzu. Jedes neue Linienstück verbindet zwei bereits vorhandene Ecken des Graphen, die auf dem Rand einer momentanen Region liegen. Diese Region wird durch das Linienstück in zwei neue Regionen zerteilt. Damit bleibt die Differenz $m - f$ bei jedem Schritt konstant. Da die Polyederformel für den Baum T erfüllt war, folgt ihre Gültigkeit auch für den gesamten Graphen G . \square

Wenn der Graph nicht zusammenhängend ist, geht die Anzahl k seiner Zusammenhangskomponenten ebenfalls in die Polyederformel ein. Es gilt dann:

Satz 9.4 (Eulersche Polyederformel II) *Jeder endliche planare Graph mit n Ecken, m Kanten, f Regionen und k Zusammenhangskomponenten erfüllt bei jeder planaren Einbettung*

$$n - m + f = 1 + k.$$

\square

Man beachte, daß in diesem Fall die Regionen auch nicht mehr notwendigerweise einfach zusammenhängend sind; vielmehr können Regionen nun auch „Löcher“ aufweisen.

9.3 Triangulationen

Bei gegebenem planaren Graphen können nach den Überlegungen in der Einleitung zu diesem Kapitel beliebig (endlich) viele parallele Kanten und Schlingen hinzugefügt werden, ohne die Planarität zu verletzen. Für diesen Abschnitt werden wir uns daher auf einfache Graphen beschränken, d. h. Parallelen und Schlingen ausschließen (bei gerichteten Graphen auch Inversen).

Definition 9.5 (Maximal planare Graphen) *Ein einfacher planarer Graph heißt **maximal planar**, wenn keine Kante hinzugefügt werden kann, ohne die Planarität oder Einfachheit zu verletzen.*

Bemerkung: Bei gerichteten Graphen ist zu den Bedingungen noch „inversenfrei“ aufzunehmen.

Lemma 9.6 Sei $G = (V, E)$ maximal planar mit $|V| \geq 3$. Dann ist in jeder Einbettung jede Kante mit genau zwei Regionen inzident.

Beweis: Sei eine Einbettung eines maximal planaren Graphen gegeben. Offenbar kann eine Kante nicht mit mehr als zwei Regionen inzidieren. Sei $e = (v_1, v_2) \in E$ eine Kante, die nur mit einer Region g inzidiert. Dann gibt es auf dem Rand von g noch eine dritte Ecke $v_3 \neq v_1, v_2$. Wäre v_3 gleichzeitig zu v_1 und zu v_2 adjazent, so bildeten die drei Ecken ein Dreieck, und e wäre mit mehr als einer Region inzident. Also kann eine der Kanten (v_1, v_3) oder (v_2, v_3) ohne Verletzung der Einfachheit zu G hinzugefügt werden, im Widerspruch zur Maximalität. \square

Unter einem **Dreieck** verstehen wir einen (Sub-)Graphen mit drei Ecken, von denen je zwei durch eine Kante verbunden sind. Damit gilt der folgende Satz:

Satz 9.7 Ist $G = (V, E)$ mit $|V| \geq 3$ maximal planar, so ist in jeder Einbettung von G jede Region durch ein Dreieck berandet.

Bemerkung: Die Aussage bezieht sich insbesondere auch auf die unbeschränkte Region.

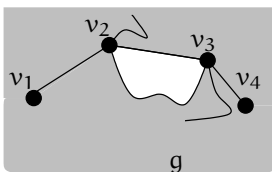
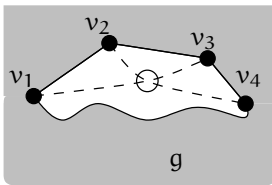
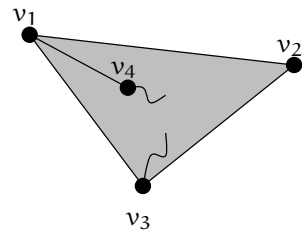
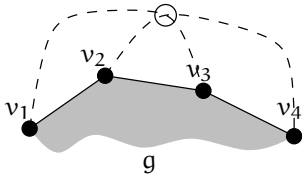
Beweis: Ohne Einschränkung kann zunächst angenommen werden, daß G zusammenhängend ist. Andernfalls ist G sicher nicht maximal planar, da die Komponenten in der Einbettung disjunkt liegen und noch mindestens einen verbindenden Linienzug zulassen.

Betrachte zunächst ein Gebiet g . Hätte g mehr als drei berandende Kanten, so sind zwei Fälle zu unterscheiden:

Im ersten Fall bilden keine drei der berandenden Kanten zusammen ein Dreieck. Dann gibt es eine Kette von vier paarweise verschiedenen Knoten v_1, \dots, v_4 auf dem Rand von g . Betrachte die beiden Kanten $a = (v_1, v_3)$ und $b = (v_2, v_4)$. Falls diese Kanten im Graphen enthalten sind, können die zugehörigen Linienstücke nicht im Inneren von g verlaufen, da sonst das Gebiet in zwei Teile getrennt würde; damit verlaufen beide Linienstücke im Äußeren von g . Es ist nun zu sehen, daß nicht beide Linienstücke ohne Schnitt im Äußeren geführt werden können. Also kann G nicht gleichzeitig a und b enthalten. Damit kann eine der beiden Kanten zu G hinzugenommen werden, ohne die Einfachheit zu verletzen. Da das zugehörige Linienstück im Inneren von g geführt werden kann, ist auch die Planarität garantiert. Damit war G nicht maximal planar.

Im zweiten Fall bilden drei Kanten ein Dreieck. Alle weiteren berandenden Kanten liegen dann im Inneren der durch das Dreieck begrenzten Fläche. Seien (v_1, v_2) , (v_2, v_3) und (v_3, v_1) die Kanten des Dreiecks, (v_1, v_4) eine weitere Kante des Randes von g . Dann kann (v_2, v_4) nicht Kante in G sein, da das zugehörige Linienstück das Gebiet zerschneiden würde. Somit war G nicht maximal planar.

Es bleibt die unbeschränkte Region g zu betrachten. Analog zum oben beschriebenen Fall existiert dann auf dem Rand von g eine Kette von vier paarweise verschiedenen Ecken v_1, \dots, v_4 . Die Kanten $a = (v_1, v_3)$ und $b = (v_2, v_4)$ können nicht gleichzeitig in G sein, da die gleichzeitige Führung beider Linienstücke außerhalb von g kreuzungsfrei nicht möglich ist. Diese Argumentation gilt auch dann, wenn eine



oder mehrere der berandenden Kanten (v_1, v_2) , (v_2, v_3) oder (v_3, v_4) neben g mit keiner anderen Region inzidieren, und insbesondere, falls g einzige Region ist. Somit kann eine der beiden Kanten a oder b zu G hinzugefügt werden, ohne die Einfachheit zu verletzen, wobei das zugehörige Liniensegment im Inneren von g geführt werden kann. Somit war der Graph nicht maximal planar. \square

Auch die Umkehrung dieses Satzes ist richtig:

Satz 9.8 Sei $G = (V, E)$ mit $|V| \geq 3$. Ist in einer Einbettung von G jede Region durch ein Dreieck berandet, so ist G maximal planar.

Beweis: Angenommen, G wäre nicht maximal planar. Dann gäbe es eine Kante e , die zu G hinzugefügt werden könnte, ohne die Planarität zu verletzen. In der gegebenen Einbettung verläuft die Kante im Inneren einer Region. Da diese ein Dreieck ist, muß e parallel zu einer der Kanten des Dreiecks laufen. Dieser Widerspruch zur Einfachheit zeigt die Behauptung. \square

Korollar 9.9 Kriterium für maximal planare Graphen Sei $G = (V, E)$ mit $|V| \geq 3$. G ist genau dann maximal planar, wenn in jeder Einbettung von G jede Region durch ein Dreieck berandet ist. \square

9.4 Grade in planaren Graphen

Auch in diesem Abschnitt wird vorausgesetzt, daß die Graphen einfach sind. Wir bezeichnen mit n die Anzahl der Ecken, mit m die Anzahl der Kanten und mit f die Anzahl der Regionen eines solchen Graphen.

Ein Hilfsmittel zum Beweis der folgenden Aussagen ist der „Face-Edge-Inzidenzgraph“ $H = (F \cup E, M)$ zur Einbettung eines planaren Graphen $G = (V, E)$. Die Eckenmenge von $H = F \cup E$ setzt sich zusammen aus der Menge F der Regionen von G und der Menge E der Kanten von G . Der Graph H ist bipartit und enthält eine Kante (f, e) genau dann, wenn in der planaren Einbettung von G die Kante e und die Region f inzidieren.

Lemma 9.10 Sei G maximal planar mit mindestens drei Ecken. Dann gilt

$$\begin{aligned} m &= 3n - 6 \\ f &= 2n - 4 \end{aligned}$$

Beweis: Im Face-Edge-Inzidenzgraphen hat jede Ecke $f_i \in F$ den Grad 3 (nach Korollar 9.9) und jede Ecke $e_i \in E$ den Grad 2 (nach Lemma 9.6). Durch Abzählen der Kanten des Face-Edge-Inzidenzgraphen ergibt sich $3f = 2m$. Aus der eulerschen Polyederformel (9.3) folgt $3f + 3n - 6 = 3m$. Damit gilt

$$m = 3m - 2m = (3f + 3n - 6) - 3f = 3n - 6.$$

Ebenso folgt aus (9.3) die Beziehung $2f = 4 + 2m - 2n$ und damit

$$f = 3f - 2f = 2m - (4 + 2m - 2n) = 2n - 4,$$

wie behauptet. \square

Satz 9.11 *In einem einfachen planaren Graphen mit mindestens 3 Ecken gilt*

$$\begin{aligned} m &\leq 3n - 6 \\ f &\leq 2n - 4. \end{aligned}$$

Beweis: Ein einfacher planarer Graph hat höchstens so viele Kanten und höchstens so viele Regionen wie ein maximal planarer Graph, der gleiche Eckenzahl besitzt. Mit Lemma 9.10 folgt die Behauptung. \square

Es ist bemerkenswert, daß für einfache planare Graphen die Anzahl der Kanten und Regionen *linear* von der Anzahl der Ecken des Graphen abhängt. Daher lassen sich für Algorithmen auf planaren Graphen häufig schärfere Abschätzungen für den Zeit- und Speicherplatzbedarf angeben als auf allgemeinen Graphen.

Im folgenden wird gezeigt, daß in einfachen planaren Graphen nicht nur die Anzahl der Kanten „überschaubar“ bleibt, sondern auch die Grade der Ecken in einem gewissen Sinn nicht zu groß werden.

Lemma 9.12 *Jeder einfache planare Graph $G = (V, E)$ hat eine Ecke $v \in V$ vom Grad $g(v) \leq 5$.*

Bemerkung: Für gerichtete Graphen muß die Voraussetzung „einfacher Graph“ durch „Graph ohne Schlingen, Parallelen und Inversen“ ersetzt werden.

Beweis: Sei o. E. G zusammenhängend, andernfalls beschränke die Betrachtung auf eine Zusammenhangskomponente. Betrachte nun eine beliebige planare Einbettung des Graphen.

Jede beschränkte Region wird von mindestens drei Kanten berandet: bestünde der Rand nur aus einer oder aus zwei Kanten, impliziert das die Existenz einer Schlinge oder eines Parallelenpaars. Weiter ist offenbar jede Kante Teil des Randes von höchstens zwei Regionen.

Durch Abzählen der Kanten im Face-Edge-Inzidenzgraphen folgt mit obigen Überlegungen

$$3(f - 1) \leq 2m, \quad \text{also} \quad f \leq \frac{2}{3}m + 1. \quad (9.1)$$

Wäre nun $g(v) \geq 6$ für jede Ecke $v \in V$, so folgte

$$2m = \sum_{v \in V} g(v) \geq 6n, \quad \text{also} \quad n \leq \frac{m}{3}. \quad (9.2)$$

Mit der Eulerschen Polyederformel und den Gleichungen (9.1) und (9.2) ergibt sich

$$2 = n - m + f \leq \frac{m}{3} - m + \frac{2}{3}m + 1 = 1.$$

Dieser Widerspruch zeigt die Behauptung. \square

Lemma 9.13 *In einem einfachen planaren Graphen ist der durchschnittliche Grad der Ecken kleiner als 6.*

Beweis: Für $n \geq 3$ folgt mit Satz 9.11:

$$\frac{\sum_{v \in V} g(v)}{|V|} = \frac{2m}{n} \leq \frac{6n - 12}{n} < 6.$$

Für $n = 1$ und $n = 2$ ist die Behauptung sofort klar. \square

Das folgende Lemma zeigt eine schärfere Aussage als Lemma 9.12.

Lemma 9.14 *Jeder endliche einfache zusammenhängende planare Graph hat mindestens 3 Ecken vom Grad kleiner als sechs.*

Beweis: Wir führen die Annahme, es gebe $n - 2$ Ecken mit Grad mindestens sechs, zum Widerspruch. Da der Graph zusammenhängend ist, gibt es keine isolierte Ecke, und es gilt

$$2m = \sum_{v \in V} g(v) \geq 6(n - 2) + 2 = 6n - 10.$$

Andererseits folgt mit Satz 9.11:

$$2m \leq 6n - 12.$$

Dies ist ein Widerspruch. \square

9.5 Kriterien für Planarität

Wir wenden uns nun der Charakterisierung von planaren bzw. nicht planaren Graphen zu. Zunächst weisen wir für zwei besonders einfache Graphen nach, daß sie nicht planar sind (und zeigen damit auch die Vermutung vom Beginn dieses Kapitels).

Satz 9.15 *Der Graph K_5 ist nicht planar.*

Beweis: Die Anzahl der Kanten des K_5 ist $m = 10$. Wäre K_5 planar, so folgte mit Satz 9.11:

$$10 = m \leq 3n - 6 = 9,$$

also war die Annahme, der K_5 sei planar, falsch. \square

Satz 9.16 *Der Graph $K_{3,3}$ ist nicht planar.*

Beweis: Der Graph $K_{3,3}$ hat $n = 6$ Ecken und $m = 9$ Kanten. Wir nehmen an, $K_{3,3}$ sei planar. In jeder Einbettung des Graphen ist jede Region von einem einfachen Zykel begrenzt. Da der Graph bipartit ist, ist die Länge jedes Zyklus gerade, und da der Graph parallelenfrei ist, ist die Länge nicht gleich zwei. Damit ist jede Region mit mindestens vier Kanten inzident. Durch Abzählen der Kanten des Face-Edge-Inzidenzgraphen ergibt sich

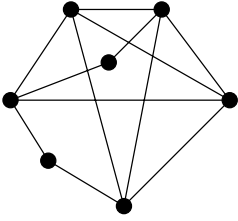
$$4f \leq 2m.$$

Mit der eulerschen Formel (9.3) folgt

$$\begin{aligned} 2 &= n - m + f \leq n - m + \frac{m}{2} \\ 9 &= m \leq 2n - 4 = 8. \end{aligned}$$

Dieser Widerspruch zeigt die Behauptung. \square

Offenbar muß in einem planaren Graphen auch jeder Teilgraph selbst planar sein. Damit halten wir fest:



Korollar 9.17 Ein Graph ist nicht planar, wenn er einen K_5 oder einen $K_{3,3}$ als Teilgraphen enthält. \square

Leider läßt sich aus diesem Korollar nicht ohne weiteres eine hinreichende Bedingung für Planarität eines Graphen ableiten. So ist etwa der am Rand abgebildete Graph nicht planar, obwohl er keinen der genannten Graphen als Teilgraphen enthält. Man erkennt aber sofort, daß in diesem Beispielgraphen die Struktur eines K_5 enthalten ist. Im folgenden werden wir Beobachtungen dieser Art präziser fassen.

Definition 9.18 (Pfeilkontraktion) Sei $G = (V, R)$ ein einfacher Graph, $r = (u, v) \in R$. Der Graph G/r entsteht aus G durch **Kontraktion** des Pfeiles r wie folgt: Die Ecken u und v werden durch eine neue Ecke uv ersetzt. Die Pfeile der Form (u, x) und (v, x) werden durch (uv, x) ersetzt, ebenso die Pfeile der Form (x, u) und (x, v) durch (x, uv) ; dies geschieht nur, soweit dadurch nicht Parallelen oder Schlingen erzeugt werden.

Bemerkung: Für eine Pfeilmenge $R' = \{r_1, \dots, r_k\} \subseteq R$ schreiben wir $G/R' = (\dots ((G/r_1)/r_2) \dots)/r_k$, wobei die Reihenfolge der Kontraktionen unerheblich ist. Die Operation ist in naheliegender Weise auch für ungerichtete Graphen erklärt.

Definition 9.19 (Graph-Minor) Ein Graph H ist **(Graph-)Minor** eines einfachen Graphen G , in Zeichen $H \leq G$, wenn H aus einem Teilgraphen von G durch Pfeilkontraktionen gewonnen werden kann.

Bemerkung: Offensichtlich gilt $G \leq G$, ferner auch $H \leq G$ für jeden Teilgraphen H von G .

In der Literatur findet sich auch eine andere Definition. Neben der Pfeilkontraktion hat man noch die Operationen »Löschen eines Pfeils« und »Löschen einer Ecke«, und man nennt H einen Minor von G , falls H durch Anwendung der drei Operationen (in beliebiger Reihenfolge) aus G hervorgeht. Man sieht, daß die Definitionen äquivalent sind.

Eine Graphklasse C heißt **minorenabgeschlossen**, wenn

$$G \in C \wedge H \leq G \Rightarrow H \in C$$

gilt. Ein Minor H in einer Graphklasse C heißt **minorenminimal**, wenn

$$G \in C \wedge G \leq H \Rightarrow G = H$$

gilt.

Für den Rest dieses Abschnittes beschränken wir uns ausdrücklich nur auf endliche Graphen und auf Klassen, die nur endliche Graphen enthalten.

Man kann zeigen, daß die Menge der einfachen planaren Graphen minorenabgeschlossen ist.

Das Entscheidungsproblem „Gegeben Graphen G und H ; gilt nun $H \leq G$?“ ist NP-vollständig. Bei fest vorgegebenem Minor H ist das Problem jedoch mit einer Laufzeit polynomiell in $|V_G|$ lösbar, wenn auch mit hohen Konstanten.

Nach Robertson und Seymour (1983–1988) gilt:

Satz 9.20 Jede Klasse von einfachen (endlichen) Graphen enthält nur endlich viele minorenminimale Elemente. \square

Für eine Klasse C von einfachen Graphen bezeichnen wir mit \bar{C} die Klasse aller übrigen einfachen Graphen; keine der Klassen enthalte unendliche Graphen.

Wählen wir C als die Menge der einfachen planaren Graphen, so besteht \bar{C} aus der Menge der nicht-planaren einfachen Graphen. Dann enthält \bar{C} nur endlich viele minorenminimale Elemente. Diese spielen eine wichtige Rolle:

Definition 9.21 (Obstruktionsmenge) Sei C eine Klasse von einfachen Graphen. Dann heißt

$$\text{Obstr}(C) := \{H \in \bar{C} \mid H \text{ ist minorenminimal in } \bar{C}\}$$

die **Obstruktionsmenge** von C .

Beispiel 9.22 Die Klasse der Wälder ist minorenabgeschlossen. Ihre Obstruktionsmenge ist $\{K_3\}$.

Lemma 9.23 Sei C Klasse von einfachen Graphen, minorenabgeschlossen. Dann gilt:

$$G \in C \iff \nexists H \in \text{Obstr}(C), H \leq G$$

Beweis: „ \Rightarrow “: Sei $H \in \text{Obstr}(C)$. Dann ist $H \in \bar{C}$. Wäre $H \leq G$, so folgte wegen der Abgeschlossenheit $H \in C$. Widerspruch.

„ \Leftarrow “: Sei $G \notin C$, also $G \in \bar{C}$. Nun ist entweder G selbst minorenminimal in \bar{C} , oder es gibt einen Graphen $H \in \bar{C}$, der minorenminimal ist und $H \leq G$ erfüllt. Dieser Graph ist dann per Definition in $\text{Obstr}(C)$. Widerspruch. \square

Ohne Beweis führen wir nun einen der berühmtesten Sätze der Graphentheorie an:

Satz 9.24 (Kuratowski, 1930) Sei C die Klasse der endlichen planaren einfachen Graphen. Dann gilt:

$$\text{Obstr}(C) = \{K_5, K_{3,3}\}.$$

Mit anderen Worten: Ein einfacher Graph G ist genau dann planar, wenn sich G und jeder Subgraph nicht auf einen der Graphen K_5 oder $K_{3,3}$ kontrahieren läßt.

Dieses notwendige und hinreichende Kriterium ist auch Basis verschiedener Algorithmen zum Test auf Planarität und zur Konstruktion einer Einbettung. Hopcroft und Tarjan [HT74] geben einen Algorithmus an, der in Laufzeit $O(|V|)$ überprüft, ob ein einfacher Graph planar ist. Hierbei wird unterstellt, daß der eingegebene Graph $G = (V, E)$ von vornherein $|E| \leq 3|V| - 6$ erfülle; andernfalls wäre der Graph ohnehin nicht planar, aber der Algorithmus benötigte möglicherweise schon allein zum Lesen der Eingabe eine Zeit, die nicht mehr in $O(|V|)$ läge.

9.6 Duale Graphen

Wir gehen davon aus, daß $G = (V, E)$ ein planarer Graph ist, und daß eine planare Einbettung von G vorliegt. Dann wird durch die folgende Konstruktion ein planarer Graph $G^* = (V^*, E^*)$ definiert, der ein **zu G dualer Graph** genannt wird.

Die Eckenmenge V^* wird gleich der Menge F der Regionen der Einbettung von G gewählt. Für jede Kante $e \in E$, die in der vorliegenden Einbettung von G mit den Regionen f_1, f_2 inzidiert, gibt es im Graphen G^* die Kante (f_1, f_2) . Diese Kante ist so zu zeichnen, daß sie keine anderen Kanten schneidet. Damit ist der entstehende Graph wieder planar.

Korollar 9.25 Sei G ein planarer Graph mit einer überschneidungsfreien Einbettung in die Ebene, n, m, f die Anzahl seiner Ecken, Kanten und Regionen. Sei G^* ein zu G dualer Graph, n^*, m^*, f^* die entsprechenden Zahlen im dualen Graphen. Dann gilt

$$m^* = m, \quad n^* = f, \quad f^* = n.$$

Beweis: Die ersten beiden Gleichungen folgen unmittelbar aus der Konstruktionsweise des dualen Graphen. Die letzte Gleichung folgt aus der eulerschen Polyederformel, die anwendbar ist, weil der duale Graph planar ist. \square

Es ist zu betonen, daß ein dualer Graph von der ursprünglich gewählten Einbettung abhängig ist. In Abb. 9.2 ist derselbe Graph in zwei verschiedenen Einbettungen zu sehen. Die entstehenden dualen Graphen sind nicht isomorph¹ zueinander, wie ein Vergleich der Grade der Ecken zeigt.

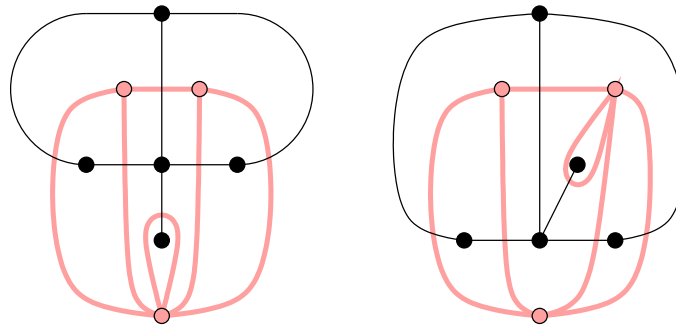


Abbildung 9.2:

Zwei verschiedene Einbettungen desselben Graphen (schwarz). Die entstehenden dualen Graphen (grau) sind nicht isomorph zueinander.

Da ein dualer Graph eines planaren Graphen selbst wieder planar ist, kann auf diesen erneut die Konstruktion eines dualen Graphen angewandt werden.

Dabei ist es möglich, die Punkte und Linienstücke in der Einbettung des neuen Graphen so zu wählen, daß sie mit den Punkten und Linienstücken des Ausgangsgraphen zusammenfallen. Das entstehende Bild des doppelt-dualen Graphen ist dann identisch dem Bild des Ausgangsgraphen, d. h. die beiden Graphen sind gleich.

Mit dem erst im nächsten Kapitel eingeführten Begriff der Isomorphie von Graphen gilt allgemeiner der folgende Satz:

Satz 9.26 Sei G ein planarer Graph, G^* ein zu G dualer Graph, G^{**} ein zu G^* dualer Graph. Dann sind G und G^{**} isomorph. \square

¹Der Begriff der Isomorphie wird erst im nächsten Kapitel genau eingeführt. Informell kann man zwei Graphen isomorph nennen, wenn sie so eingebettet werden können, daß die Bilder der Einbettungen durch „Wackeln“ an den Punkten und Linienstücken zur Deckung gebracht werden können.

Zykel und Schnitte

Auch planare Graphen lassen sich mit einer Kantenbewertung versehen, die Kosten, Längen oder Kapazitäten modelliert. Beim Übergang zum Dualen eines Graphen ergibt sich auf natürliche Weise wieder ein kantenbewerteter Graph, wobei die Kantenbewertung durch die 1:1-Korrespondenz zwischen den Kantenmengen der beiden Graphen vom Ausgangsgraphen auf den dualen übernommen wird.

In Abb. 9.3 wird ein Paar von dualen Graphen dargestellt. (Die Kanten, die aus dem Bild ragen, sind alle mit derselben Ecke v_∞ inzident, die die unbeschränkte Region vertritt.) Im linken Teil ist ein Zykel ausgezeichnet, im rechten Teil sind die entsprechenden Kanten im dualen Graphen markiert. Wie leicht zu erkennen ist, bilden diese Kanten einen Schnitt. Wo der Zykel die Regionen in seinem Inneren von den außenliegenden trennt, trennt der Schnitt die entsprechenden Eckenmengen voneinander.

Wenn die Kantenbewertung im dualen Graphen als Kapazität aufgefaßt wird, so erkennt man, daß die Länge eines Zyklus in einem planaren Graphen korrespondiert mit der Kapazität eines Schnittes in einem dualen Graphen. Diese Beziehung läßt sich auch algorithmisch nutzen, wobei zugute kommt, daß ein dualer Graph in Konstruktion und Speicherplatzaufwand von vergleichbarer Größenordnung des Ausgangsgraphen ist.

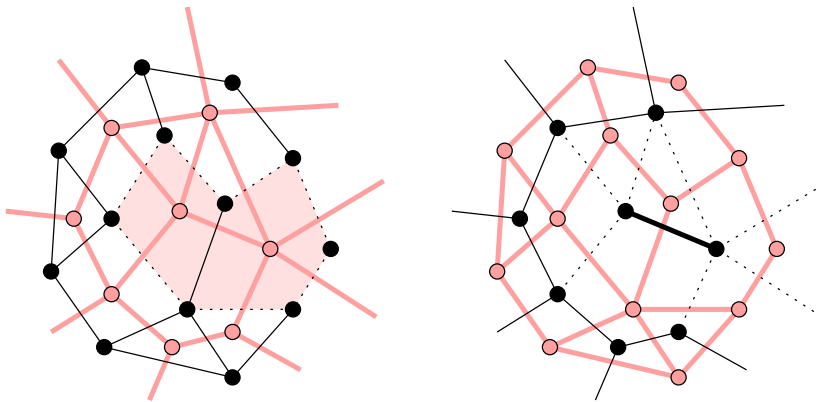


Abbildung 9.3: Zusammenhang zwischen Länge eines Zyklus (links, gepunktet) und Kapazität eines Schnittes im dualen Graphen (rechts, gepunktet). Die aus dem Bild ragenden Kanten sind alle mit der Ecke v_∞ der unbeschränkten Region inzident.

Kapitel 10

Homomorphismen

In diesem Kapitel beschäftigen wir uns mit Frage, wann zwei Graphen „ähnlich“ sind.

10.1 Tripeldarstellung von Graphen

Sei $G = (V, R, \alpha, \omega)$ ein Graph.¹ Wir definieren die *Grundmenge* von G durch $\underline{G} := V \cup R$ und erweitern die Abbildungen α, ω auf \underline{G} in folgender Weise:

$$\begin{aligned}\alpha(v) &:= v \text{ und} \\ \omega(v) &:= v \text{ für alle } v \in V.\end{aligned}$$

Somit läßt sich ein Graph $G = (V, R, \alpha, \omega)$ auch in *Tripeldarstellung* $G = (\underline{G}, \alpha, \omega)$ notieren. Ein Graph ist genau dann endlich, wenn \underline{G} endlich ist.

Für einen Graphen $G = (\underline{G}, \alpha, \omega)$ in Tripeldarstellung gilt dann:

$$V := \alpha(\underline{G}) = \omega(\underline{G})$$

und ein Element $x \in \underline{G}$ ist genau dann eine Ecke von G , wenn $\alpha(x) = x$ (oder $\omega(x) = x$) gilt.

Im folgenden sei $G = (\underline{G}, \alpha, \omega)$ ein Graph in Tripeldarstellung, d.h. $V := \alpha(\underline{G}) = \omega(\underline{G})$.

10.2 Homomorphismen

Definition 10.1 (Homomorphismus)

Seien $G_1 = (\underline{G}_1, \alpha_1, \omega_1)$ und $G_2 = (\underline{G}_2, \alpha_2, \omega_2)$ zwei Graphen. Eine Abbildung $\tau: \underline{G}_1 \rightarrow \underline{G}_2$ heißt (**Graph-**) **Homomorphismus**, wenn gilt:

1. τ ist surjektiv und
2. $\tau(\alpha_1(e)) = \alpha_2(\tau(e))$ sowie $\tau(\omega_1(e)) = \omega_2(\tau(e))$ für alle $e \in \underline{G}_1$.
(Kurz: $\tau \circ \alpha_1 = \alpha_2 \circ \tau$ und $\tau \circ \omega_1 = \omega_2 \circ \tau$)

¹Wie in Definition 2.1 setzen wir voraus, daß $V \cap R = \emptyset$ gilt.

Der Homomorphismus τ ist ein **Isomorphismus**, falls τ bijektiv ist. Falls es einen Isomorphismus von G_1 nach G_2 gibt, so nennen wir G_1 und G_2 **isomorph** und schreiben $G_1 \cong G_2$.

Ein Isomorphismus ist ein **Automorphismus**, wenn $G_1 = G_2$ gilt. Mit $\mathcal{A}(G)$ bezeichnen wir die Menge aller Automorphismen des Graphen G .

Beispiel 10.2 Abbildung 10.1 zeigt einen Graphen G_1 und sein homomorphes Bild G_2 unter dem Homomorphismus τ , der gemäß Tabelle 10.1 gegeben ist.

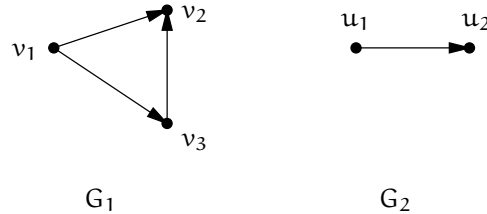


Abbildung 10.1:
Ein Graph G_1 und ein homomorphes Bild G_2 .

x	$\tau(x)$
v_1	u_1
v_2	u_2
v_3	u_1
(v_1, v_2)	(u_1, u_2)
(v_1, v_3)	v_1
(v_3, v_2)	(u_1, u_2)

Tabelle 10.1: Beispiel für einen Homomorphismus τ .

Lemma 10.3 Ist $\tau : \underline{G}_1 \rightarrow \underline{G}_2$ ein Homomorphismus, so gilt $\tau(V_1) = V_2$.

Beweis:

„ $\tau(V_1) \subseteq V_2$ “

Sei $v \in V_1$. Dann ist $\alpha_1(v_1) = v_1$ und folglich

$$\tau(v_1) = \tau(\alpha_1(v_1)) = \underbrace{\alpha_2(\tau(v_1))}_{\in V_2}.$$

Somit ist $\tau(v_1) \in V_2$.

„ $\tau(V_1) \supseteq V_2$ “

Sei $v_2 \in V_2$. Da τ surjektiv ist, gibt es $x \in \underline{G}_1$ mit $\tau(x) = v_2$. Somit

$$v_2 = \alpha_2(v_2) = \alpha_2(\tau(x)) = \tau(\underbrace{\alpha_1(x)}_{=: y \in V_1}).$$

Somit gilt $\tau(y) = v_2$ für ein $y \in V_1$, also $v_2 \in \tau(V_1)$. □

Korollar 10.4 Gilt $G_1 \cong G_2$, so folgt $|V_1| = |V_2|$. □

Korollar 10.5 Ist $\tau : \underline{G}_1 \rightarrow \underline{G}_2$ ein Isomorphismus, so ist $\tau(V_1) = V_2$ und $\tau(R_1) = R_2$.

Beweis: Nach Lemma 10.3 gilt $\tau(V_1) = V_2$. Wäre $\tau(r) \in V_2$ für ein $r \in R$, so wäre τ nicht bijektiv. Da τ nach Definition eines Homomorphismus surjektiv ist, folgt, daß für jedes $r' \in R_2$ auch ein $r \in R_1$ mit $\tau(r) = r'$ existiert. \square

Lemma 10.6 Ist $\tau : \underline{G}_1 \rightarrow \underline{G}_2$ ein Isomorphismus, so gilt für jedes $v \in V_1$: $g_{G_1}^+(v) = g_{G_2}^+(\tau(v))$, und $g_{G_1}^-(v) = g_{G_2}^-(\tau(v))$, d.h. der Außen- und Innen-grad der Ecke $\tau(v)$ in G_2 ist gleich dem Außen- bzw. Innen-grad von v in G_1 .

Beweis: Sei $v \in G_1$. Sei $r \in B_{G_1}^+(v)$ beliebig. Dann gilt

$$\alpha_2(\tau(r)) = \tau(\alpha_1(r)) = \tau(v).$$

Somit wird jeder Pfeil $r \in B_{G_1}^+(v)$ auf ein $r' \in G_2$ mit $r' \in B_{G_2}^+(\tau(v))$ abgebildet. Da nach Korollar 10.5 Pfeile auf Pfeile abgebildet werden, folgt, daß $g_{G_1}^+(v) = g_{G_2}^+(\tau(v))$ gilt. Analog zeigt man die Behauptung für den Innen-grad. \square

10.3 Das Graphenisomorphieproblem

Definition 10.7 (Graphenisomorphieproblem)

Eine Instanz des **Graphenisomorphieproblems** ist durch zwei Graphen $G_1 = (V_1, R_1)$ und $G_2 = (V_2, R_2)$ gegeben. Das Problem ist zu entscheiden, ob $G_1 \cong G_2$ gilt.

Das Graphenisomorphieproblem liegt in NP, seine genaue Komplexität ist bisher noch unbekannt. Es konnte bisher weder gezeigt werden, daß das Problem NP-vollständig ist, noch daß es in polynomialer Zeit lösbar ist.

Ein einfacher Ansatz führt zu einem exponentiellen Algorithmus.

Lemma 10.8 Sind $G_1 = (V_1, R_1)$ und $G_2 = (V_2, R_2)$ einfache Graphen, so ist jeder Isomorphismus $\tau: \underline{G}_1 \rightarrow \underline{G}_2$ durch die Bilder $\tau(v_1)$ ($v_1 \in V_1$) eindeutig festgelegt.

Beweis: Ist $r \in R_1$ mit $r = (u, v)$ so gilt dann

$$\begin{aligned}\alpha_2(\tau(r)) &= \tau(\alpha_1(r)) = \tau(u) \\ \omega_2(\tau(r)) &= \tau(\omega_1(r)) = \tau(v),\end{aligned}$$

also $\tau(r) = (\tau(u), \tau(v))$. \square

Nach Korollar 10.4 gilt für isomorphe Graphen $G_1 \cong G_2$: $|V_1| = |V_2|$. Sei o.B.d.A. $V_1 = V_2 = \{1, \dots, n\}$. Dann können wir jeden Isomorphismus $\tau: \underline{G}_1 \rightarrow \underline{G}_2$ mit einer Permutation π von $\{1, \dots, n\}$ identifizieren.

Also genügt es² für den Test, ob $G_1 \cong G_2$ gilt, für jede der $n!$ Permutationen zu prüfen, ob diese Permutation einen Isomorphismus zwischen G_1 und G_2 induziert. Dies führt zu einem Aufwand von $\mathcal{O}(n!n^2)$.

²Falls $|V_1| \neq |V_2|$, so können wir die Frage nach der Isomorphie sofort mit „nein“ beantworten.

10.4 Automorphismen

Satz 10.9 Sei $G = (V, R, \alpha, \omega)$ ein Graph. Dann ist $(\mathcal{A}(G), \circ)$ eine (im allgemeinen nicht kommutative) Gruppe.

Beweis: Jeder Automorphismus ist eine Bijektion von \underline{G} nach \underline{G} . Da die Menge B der Bijektionen von \underline{G} nach \underline{G} eine Gruppe bezüglich der Hintereinanderausführung bilden, genügt es zu zeigen, daß $\mathcal{A}(G)$ eine Untergruppe von B ist.

Dafür müssen wir zeigen, daß für alle $\tau, \varphi \in \mathcal{A}(G)$ gilt: $\tau \circ \varphi \in \mathcal{A}(G)$ und $\tau^{-1} \in \mathcal{A}(G)$.

1. $\tau \circ \varphi \in \mathcal{A}(G)$:

Es gilt für beliebiges $x \in \underline{G}$

$$\begin{aligned} \alpha((\tau \circ \varphi)(x)) &= \alpha(\tau(\underbrace{\varphi(x)}_{=: y \in \underline{G}})) = \alpha(\tau(y)) = \tau(\alpha(y)) \\ &= \tau(\alpha(\varphi(x))) = \tau(\varphi(\alpha(x))) = (\tau \circ \varphi)(\alpha(x)). \end{aligned}$$

Analog folgt

$$\omega((\tau \circ \varphi)(x)) = (\tau \circ \varphi)(\omega(x)).$$

Somit ist $\tau \circ \varphi \in \mathcal{A}(G)$.

2. $\tau^{-1} \in \mathcal{A}(G)$:

Für beliebiges $x \in \underline{G}$ gilt mit $y := \tau^{-1}(x)$:

$$\tau^{-1}(\alpha(x)) = \tau^{-1}(\alpha(\tau(y))) = \tau^{-1}(\tau(\alpha(y))) = \alpha(y) = \alpha(\tau^{-1}(x)).$$

Somit ist auch $\tau^{-1} \in \mathcal{A}(G)$.

□

Definition 10.10 (Zyklische Gruppe)

Eine Gruppe (\mathcal{G}, \circ) heißt **zyklisch**, wenn ein $e \in \mathcal{G}$ existiert, so daß $\mathcal{G} = \{e^0, e^1, e^2, \dots\}$ gilt. Dabei sei $e^0 := 1_{\mathcal{G}}$, $e^1 := e$ und $e^t := e^{t-1} \circ e$ für $t > 1$.

Beispiel 10.11 Wir betrachten die Automorphismengruppe $\mathcal{A}(G_n)$ des Graphen $G_n = (V_n, R_n, \alpha_n, \omega_n)$ mit

$$\begin{aligned} V_n &:= \{v_0, v_1, \dots, v_{n-1}\} \\ R_n &:= \{r_0, r_1, \dots, r_{n-1}\} \end{aligned}$$

und für alle $k \in \{0, 1, \dots, n-1\}$ gelte

$$\alpha_n(r_k) := v_k \quad \text{und} \quad \omega_n(r_k) := v_{(k+1) \bmod n}.$$

Der Graph G_n ist in Abbildung 10.2 dargestellt. Wir zeigen nun, daß $\mathcal{A}(G_n)$ zyklisch ist und $|\mathcal{A}(G_n)| = |V_n| = n$ gilt.

Dazu beweisen wir zunächst, daß ein Automorphismus φ von G_n bereits durch $\varphi(v_0)$ eindeutig festgelegt ist. Seien φ und ϑ Automorphismen von G_n mit $\varphi(v_0) = \vartheta(v_0) = v_t$. Wir zeigen durch Induktion

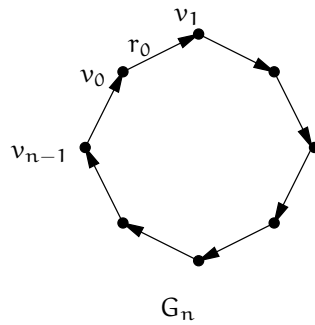


Abbildung 10.2:
Ein Graph G_n mit zyklischer
Automorphismengruppe
der Ordnung n .

nach k , daß $\varphi(v_i) = \vartheta(v_i)$ und $\varphi(r_i) = \vartheta(r_i)$ für $i = 0, \dots, k-1$ gelten muß.

Für $k = 0$ ist nur noch $\varphi(r_0) = \vartheta(r_0)$ zu zeigen. Es gilt

$$\alpha_n(\varphi(r_0)) = \varphi(\alpha_n(r_0)) = \varphi(v_0) = \vartheta(v_0) = \vartheta(\alpha_n(r_0)) = \alpha_n(\vartheta(r_0)).$$

Somit wird r_0 sowohl von φ als auch von ϑ auf einen Pfeil abgebildet, der in $v_t = \varphi(v_0) = \vartheta(v_0)$ startet. Da jede Ecke in G_n nur einen ausgehenden Pfeil besitzt, folgt, daß $\varphi(r_0) = \vartheta(r_0) = r_t$ gelten muß.

Es sei bereits $\varphi(v_i) = \vartheta(v_i)$ und $\varphi(r_i) = \vartheta(r_i)$ für $i = 0, \dots, k$ bewiesen. Dann folgt

$$\varphi(v_{k+1}) = \varphi(\omega_n(r_k)) = \omega_n(\varphi(r_k)) \stackrel{\text{IV}}{=} \omega_n(\vartheta(r_k)) = \vartheta(\omega_n(r_k)) = \vartheta(v_{k+1}).$$

Weiterhin folgt damit

$$\alpha_n(\varphi(r_{k+1})) = \varphi(\alpha_n(r_{k+1})) = \varphi(v_{k+1}) = \vartheta(v_{k+1}) = \vartheta(\alpha_n(r_{k+1})) = \alpha_n(\vartheta(r_{k+1})).$$

Da α_n injektiv ist, folgt $\varphi(r_{k+1}) = \vartheta(r_{k+1})$. Dies beendet die Induktion.

Da $\varphi(v_0)$ nur n verschiedene Werte annehmen kann, gilt $|\mathcal{A}(G_n)| \leq n$. Sei nun $\tau \in \mathcal{A}(G_n)$ für $k \in \{0, 1, \dots, n-1\}$ definiert durch

$$\tau(v_k) := v_{(k+1) \bmod n} \quad \text{und} \quad \tau(r_k) := r_{(k+1) \bmod n}.$$

Dann gilt für $k \in \{0, 1, \dots, n-1\}$: $\tau^k(v_0) = v_k$. Insbesondere sind so also n verschiedene Automorphismen angegeben, es gilt $|\mathcal{A}(G_n)| \geq n$ und $\mathcal{A}(G_n)$ ist zyklisch.

Anhang A

Lösungen zu den Übungsaufgaben

Kapitel 2

Lösung zu Aufgabe 2.1

- (a) Für die Adjazenzmatrix des inversen Graphen G^{-1} gilt $A(G^{-1}) = A^T(G)$, wobei wir mit $A^T(G)$ die Transponierte der Matrix $A(G)$ bezeichnen.

Die Transponierte einer Matrix läßt sich durch Vertauschen der Zeilen- und Spalten-Indizes berechnen. Sei $A(G) = (a_{ij})$, dann liefert Algorithmus A.1 die Matrix $A(G^{-1}) = (a'_{ij})$.

Algorithmus A.1 Berechnung von G^{-1} bei Adjazenzmatrixdarstellung

Input: Ein gerichteter einfacher Graph $G = (V, R)$ in Adjazenzmatrixdarstellung

```
1 for  $1 \leq i \leq |V|$  do
2   for  $1 \leq j \leq |V|$  do
3      $a'_{ij} \leftarrow a_{ji}$ 
4   end for
5 end for
```

Algorithmus A.1 besitzt eine Laufzeit von $\Theta(|V|^2)$. Diese Laufzeit ist offenbar Worst-Case optimal, da man zur Erstellung der Adjazenzmatrix des inversen Graphen jeden der $\Theta(|V|^2)$ Einträge der Adjazenzmatrix von G betrachten muß.

Zur Berechnung der Adjazenzliste Adj' des inversen Graphen durchläuft man alle Pfeile des Ursprungsgraphen, d.h. alle Listenelemente seiner Adjazenzliste Adj , und fügt die Pfeile mit vertauschtem Anfangs- und Endknoten in die Liste des inversen Graphen ein. Dieses Verfahren ist in Algorithmus A.2 dargestellt.

Da das Einfügen an den Anfang einer Liste in konstanter Zeit möglich ist, beträgt die Laufzeit $\Theta(|V|+|R|)$. Auch dieser Algorithmus ist von der Laufzeit her Worst-Case optimal, da die Eingabegröße für einen Graphen in Adjazenzlistendarstellung $\Theta(|V|+|R|)$

Algorithmus A.2 Berechnung von G^{-1} bei Adjazenzlistendarstellung

Input: Ein gerichteter einfacher Graph $G = (V, R)$ in Adjazenzlistendarstellung

```

1 for all  $u \in V$  do
2   for all  $v \in \text{Adj}[u]$  do
3     Füge den Endknoten  $u$  an den Anfang der Liste  $\text{Adj}'[v]$  ein
4   end for
5 end for

```

ist und man in einem korrekten Algorithmus die Eingabe komplett betrachten muß.

(b) Sei $G = (V, R)$ ein gerichteter einfacher Graph mit $n := |V|$ Ecken und $A(G) = (a_{ij})$ seine Adjazenzmatrix.

Behauptung 1: Es gibt höchstens eine Ecke $v \in V$ mit $g^+(v) = 0$ und $g^-(v) = n - 1$.

Beweis: Da G keine Schlingen enthält, folgt aus $g^-(v) = n - 1$, daß für alle $w \neq v$ der Pfeil (w, v) in R enthalten ist. Damit kann außer v keine Ecke Innengrad $n-1$ und Außengrad 0 besitzen. \square

Folgerung 2: Sei $i \neq j$. Dann gilt:

- falls $a_{ij} = 1$: die Ecke i kommt nicht in Betracht.
- falls $a_{ij} = 0$: die Ecke j kommt nicht in Betracht.

In Algorithmus A.3 ist ein Verfahren angegeben, daß in $\mathcal{O}(n)$ Zeit testet, ob eine Ecke v mit $g^-(v) = n - 1$ und $g^+(v) = 0$ in G existiert.

Algorithmus A.3 Reduzierung der Suche auf eine Ecke.

Input: Ein gerichteter einfacher Graph $G = (V, R)$ in Adjazenzmatrixdarstellung mit $n := |V|$ Ecken

```

1 Setze  $i \leftarrow 2$  (Zeilen-) und  $j \leftarrow 1$  (Spalten-Zähler)
2 while  $i \leq n$  do
3   if  $a_{ij} = 0$  then
4      $j \leftarrow i$ 
5   end if
6    $i \leftarrow i + 1$ 
7 end while

```

Behauptung 3: Nach Durchlauf des Algorithmus kommt nur die Ecke j als Kandidat in Betracht.

Beweis: Wir zeigen das Ergebnis mit Induktion nach den Schritten des Algorithmus. Sei

$$M(i, j) := \{1, 2, \dots, i-1\} \setminus \{j\}.$$

Dann gilt: Steht der Algorithmus an Position (i, j) , dann können alle Ecken in $M(i, j)$ als Kandidaten ausgeschlossen werden.

Induktionsanfang: $(i, j) = (2, 1)$. Die Behauptung gilt, weil $M(2, 1) = \emptyset$.

Induktionsschluß: Befinde sich der Algorithmus an Position (i, j) , und sei die Menge $M := M(i, j)$ markiert.

1. Fall: $a_{ij} = 0$. Wegen Folgerung 2 ist die Ecke j auszuschließen, d. h. es muß gelten $M' = M \cup \{j\}$. Der Algorithmus setzt $i' := i + 1$ und $j' := i$. Damit ist

$$\begin{aligned} M' &= M(i', j') = M(i + 1, i) = \\ &= \{1, 2, \dots, i\} \setminus \{i\} = \{1, \dots, i - 1\} = \\ &= (\{1, \dots, i - 1\} \setminus \{j\}) \cup \{j\} = \\ &= M \cup \{j\} \end{aligned}$$

2. Fall: $a_{ij} = 1$. Wegen Folgerung 2 ist die Ecke i auszuschließen, d. h. es muß gelten $M' = M \cup \{i\}$. Der Algorithmus setzt $i' := i + 1$ und $j' := j$. Damit ist

$$\begin{aligned} M' &= M(i', j') = M(i + 1, j) = \\ &= \{1, 2, \dots, i\} \setminus \{j\} = \\ &= (\{1, \dots, i - 1\} \setminus \{j\}) \cup \{i\} = \\ &= M \cup \{i\} \end{aligned}$$

Am Ende des Algorithmus gilt $i = n + 1$, also sind die Ecken

$$M(n + 1, j) = \{1, \dots, n\} \setminus \{j\}$$

markiert, und es verbleibt nur die Ecke j als Kandidat. \square

Offenbar ist die Laufzeit von Algorithmus A.3 in $\mathcal{O}(n)$. Nach Ende des Algorithmus ist nur noch eine Ecke j potentieller Kandidat. Ob j dann $g^-(j) = n - 1$ und $g^+(j) = 0$ erfüllt, kann mit Algorithmus A.4 in $\mathcal{O}(n)$ Zeit getestet werden.

Algorithmus A.4 Testen der (einzigen) Kandidatenecke j .

```

1 Kandidat  $\leftarrow$  true
2 for  $k \in \{1, 2, \dots, n\} \setminus \{j\}$  do
3   if  $a_{jk} = 1 \vee a_{kj} = 0$  then
4     Kandidat  $\leftarrow$  false
5   stop
6   end if
7 end for
```

Kapitel 3

Lösung zu Aufgabe 3.1

- (a) **Reflexivität** Für alle $v \in V$ gilt nach Definition 3.9: $v \in E_G(v)$, also ist $v \sim v$ nach Definition 3.10.

Symmetrie Sei $u \sim v$. Dann ist $v \in E_G(u)$ und $u \in E_G(v)$. Nach Definition 3.10 ist daher dann auch $v \sim u$.

Transitivität Sei $u \sim v$ und $v \sim x$. Dann ist $v \in E_G(u)$. Entweder ist $u = v$ oder es existiert ein Weg w mit $\alpha(w) = u$ und $\omega(w) = v$.

Ist $v = x$, so ist dann entweder $u = v = x$ oder w ein Weg von u nach x , also $x \in E_G(u)$. Falls $v \neq x$, so muß wegen $x \in E_G(v)$ ein Weg w' mit $\alpha(w') = v$ und $\omega(w') = x$ existieren. Wenn $v = u$, so ist dieser Weg w' ein Weg von u nach x . Ansonsten ist $w \circ w'$ ein Weg, der von u nach x führt. Insgesamt ist damit also $x \in E_G(u)$ gezeigt. Die Umkehrung $u \in E_G(x)$ folgt analog. \square

(b) Da G stark zusammenhängend und $|V| \geq 2$ ist, muß für jedes $v \in V$ gelten: $g^+(v) \geq 1$. Somit gilt

$$|R| = \sum_{v \in V} g^+(v) \geq \sum_{v \in V} 1 = |V|. \quad \square$$

Lösung zu Aufgabe 3.2

Sei $w = (r_1, \dots, r_k)$ und $s(w) = (u = v_1, \dots, v_{k+1} = v)$. Falls w bereits elementar ist, so ist nichts zu zeigen. Ansonsten sei $v_i = v_{i+p}$ und i minimal mit der Eigenschaft, daß v_i von w mehr als einmal berührt wird. Dann ist $w' := (r_1, \dots, r_{i-1}, r_{i+p}, \dots, r_k)$ ein Weg von u nach v . Falls w' elementar ist, so sind wir fertig. Ansonsten setzen wir das obige Verfahren mit w' fort. Da die Länge der Wege in jedem Schritt jeweils strikt abnimmt, muß das Verfahren mit einem elementaren Weg terminieren. \square

Lösung zu Aufgabe 3.3

Die erste Ungleichung ist trivial. Wir zeigen daher nur den zweiten Teil. Seien G_1, \dots, G_p die Komponenten von G und G'_1, \dots, G'_k die Komponenten von $G - e$. Sei ferner $\gamma(e) = \{u, v\}$. Offenbar sind dann u und v in der gleichen Komponenten von G enthalten. Sei dies O.B.d.A. G_1 .

Wir zeigen nun, daß für $i = 2, \dots, p$ folgendes gilt: Sind $a, b \in G_i$, so existiert eine Komponente von $G - e$ die beide Ecken a und b enthält. Danach zeigen wir, daß G_1 durch Entfernen von e in höchstens zwei Komponenten zerfällt. Daraus folgt dann $k \leq p + 1$.

Seien $a, b \in G_i$ für ein $i \geq 2$. Dann existiert nach Definition des Zusammenhangs ein Weg $w = (e_1, \dots, e_s)$ zwischen a und b in G . Dieser Weg kann e nicht enthalten, denn sonst wären $u, v \in G_i$ im Widerspruch dazu, daß $u, v \in G_1$ und der Tatsache, daß die Komponenten eine Partition von V bilden. Dann ist w aber auch ein Weg in $G - e$ zwischen a und b , d.h. a und b müssen in der gleichen Komponente von $G - e$ enthalten sein.

Wir betrachten nun $G_1 = (V_1, E_1)$, d.h. diejenige Komponente von G , welche u und v enthält. Wir definieren eine Partition $V_1 = U \cup (V_1 \setminus U)$ von V_1 durch

$$U := \{u\} \cup \{x \in V_1 : \text{es gibt einen Weg von } x \text{ nach } u, \text{ der } e \text{ nicht benutzt}\}.$$

Offenbar sind dann alle $x \in U$ in der gleichen Zusammenhangskomponente von $G - e$ wie u . Wir zeigen, daß alle Ecken aus $V_1 \setminus U$ in der gleichen Zusammenhangskomponente wie v sind.

Sei dazu $x \in V_1 \setminus U$ beliebig. Nach Voraussetzung existiert ein Weg $w = (e_1, \dots, e_s)$ von x nach u in G . Dieser Weg benutzt e . O.B.d.A. können wir annehmen, daß $e = e_s$ und e sonst nicht in w vorkommt. Dann ist (e_1, \dots, e_{s-1}) ein Weg von x nach v in $G - e$.

Lösung zu Aufgabe 3.4

Sei zunächst $G = (V, E)$ bipartit mit der Partition $V = A \cup B$ und sei w ein elementarer Kreis in G . Wir zeigen, daß w gerade Länge haben muß.

Sei $w = [v_1, \dots, v_k = v_1]$ und o. E. sei $v_1 \in A$. Da G bipartit ist, muß $v_2 \in B$ sein, u. s. w. Induktiv folgt, daß $v_i \in A$ für alle ungeraden i und $v_i \in B$ für alle geraden i . Wegen $v_k = v_1 \in A$ muß k ungerade sein. Also hat w gerade Länge. Dies zeigt die eine Folgerungsrichtung.

Umgekehrt besitze G nun keine elementaren Kreise ungerader Länge.

Sei $v \in V$. Da G zusammenhängend ist, gibt es von jeder Ecke in V einen Weg zu v . Daher bilden die Mengen $A = \{x \in V : \text{der kürzeste Weg von } v \text{ nach } x \text{ hat gerade Länge}\}$ und $B = \{x \in V : \text{der kürzeste Weg von } v \text{ nach } x \text{ hat ungerade Länge}\}$ eine Partition von V .

Wir zeigen nun, daß es in A und B keine adjazenten Ecken gibt:

Annahme: $a, a' \in A$ seien adjazent. Seien $w = [v = w_1, \dots, w_k = a]$ und $w' = [v = w'_1, \dots, w'_p = a']$ jeweils kürzeste Wege von v nach a und a' . Wähle i, j maximal mit $w_i = w'_j$. Da w und w' minimale Wege sind, muß $i = j$ sein. Daher haben die Teilwege $[w_i, \dots, a]$ und $[w'_j, \dots, a']$ entweder beide gerade, oder beide ungerade Länge.

Da a und a' adjazent sind, läßt sich damit ein Kreis $[w_i, \dots, a, a', \dots, w'_j = w_i]$ ungerader Länge konstruieren im Widerspruch dazu, daß G keine ungeraden Kreise besitzt.

Analog folgt, daß B keine adjazenten Ecken enthält.

Da A und B eine Partition bilden, muß jede Kante in E eine Ecke in beiden Mengen haben. \square

Lösung zu Aufgabe 3.5

Sei $v \in V$ Artikulationspunkt von G . Da $G[V \setminus \{v\}]$ nicht zusammenhängend ist, muß es in diesem Graphen zwei Ecken x und y geben, die durch keinen Weg verbunden sind.

Da G aber zusammenhängend war, muß es in G Wege von x nach y geben. Diese müssen die Ecke v enthalten.

Für die umgekehrte Richtung seien $x, y \in V$ zwei von v verschiedene Ecken, so daß jeder Weg zwischen x und y die Ecke v berührt.

Sei w nun ein Weg in $G[V \setminus \{v\}]$ mit $\gamma(w) = (x, y)$. Wegen $G[V \setminus \{v\}] \sqsubseteq G$ ist w auch in G ein Weg zwischen x und y , der aber v nicht enthält – im Widerspruch zur Voraussetzung.

Daher gibt es in $G[V \setminus \{v\}]$ keinen Weg zwischen x und y , d.h. x und y befinden sich in unterschiedlichen Zusammenhangskomponenten von $G[V \setminus \{v\}]$. G ist also nach Entfernen von v nicht mehr zusammenhängend und damit ist v nach Def. ein Artikulationspunkt von G . \square

Lösung zu Aufgabe 3.6

- (a) „ \Rightarrow “ z.z. Es existiert eine topologische Sortierung des Graphen $G \Rightarrow G$ ist kreisfrei.

Widerspruchannahme: Es ex. eine topologische Sortierung σ und G enthält einen Kreis $w = (r_1, r_2, \dots, r_p)$, mit $\sigma(\alpha(r_i)) < \sigma(\omega(r_i)) = \sigma(\alpha(r_{i+1}))$ $i \in \{1, \dots, p-1\}$ folgt dann induktiv: $\sigma(\alpha(r_1)) < \sigma(\alpha(r_p)) < \sigma(\omega(r_p)) = \sigma(\alpha(r_1))$ Widerspruch! \square

„ \Leftarrow “ Beweis durch Induktion nach $|V|$. Falls $|V| = 1$, so ist die Aussage klar. Sei sie für $|V| = n$ bewiesen und $|V| = n + 1$.

Es existiert eine Ecke $v \in V$ mit $g^-(v) = 0$, sonst besäße G nach Lemma 3.6 einen Kreis.

Der Graph $G[V \setminus \{v\}]$ ist kreisfrei und besitzt nach Induktionsvoraussetzung eine topologische Sortierung $\sigma: V \setminus \{v\} \rightarrow \{1, \dots, n\}$. Dann ist aber σ' mit $\sigma'(v)' := 1$ und $\sigma'(u) := \sigma(u) + 1$ für $u \neq v$ eine topologische Sortierung von G (da $g^-(v) = 0$). \square

(b) TOPSORT

Input: Ein gerichteter Graph $G = (V, R)$ in Adjazenzlistendarstellung mit $|V| = n$

```

1 k ← 1
2 L ← ∅ {leere Liste}
3 for all v ∈ V do
4   indeg[v] ← 0
5 end for
6 for all v ∈ V do
7   for all w ∈ Adj[v] do
8     indeg[w] ← indeg[w] + 1
9   end for
10 end for
11 for all v ∈ V do
12   if indeg[v] = 0 then
13     füge v an L an
14   end if
15 end for
16 while L ≠ ∅ do
17   entferne das erste Element v von L
18   σ[v] ← k
19   k ← k + 1
20   for all w ∈ Adj[v] do
21     indeg[w] ← indeg[w] - 1
22     if indeg[w] = 0 then
23       füge w an L an
24     end if
25   end for
26 end while
27 if k = |V| + 1 then
28   return true
29 else
30   return false
31 end if
```

Die Schritte 1 bis 5 benötigen $\mathcal{O}(|V|)$ Zeit. Jeder Pfeil $r \in R$ wird in Schritt 7 und 20 höchstens einmal erfaßt. Somit ist die Komplexität des Algorithmus $\mathcal{O}(|V| + |R|)$.

Behauptung:

- (i) Zu Beginn des k ten Durchlaufs der **while**-Schleife gilt für jedes $w \in V$:

$$\text{indeg}[w] = |\{u \in V : u \in V_G(w) \wedge \sigma[u] \text{ ist noch nicht definiert}\}| \quad (\text{A.1})$$

- (ii) Ist v der im k ten Durchlauf gewählte Knoten, so gilt $k > \sigma[x]$ für alle Vorgänger von v .

Induktion nach k : $k = 1$: In den Zeilen 6 bis 10 werden offenbar korrekt die Innengrade der Ecken berechnet und in indeg gespeichert.

$k \rightarrow k + 1$: Sei v der im k ten Durchlauf gewählte Knoten. Dann wird $\text{indeg}[w]$ genau für die Nachfolger von v um eins vermindert und v numeriert. Dies zeigt, daß Gleichung (A.1) zu Beginn des $(k + 1)$ ten Durchlauf gilt.

Sei nun v der im $(k + 1)$ ten Durchlauf gewählte Knoten. Wir müssen zeigen, daß $k + 1 > \sigma[x]$ für alle Vorgänger von v gilt. Wir haben bereits gezeigt, daß zu Beginn des $(k + 1)$ ten Durchlaufs $\text{indeg}[v]$ die Anzahl der noch nicht numerierten Vorgänger von v . Da $\text{indeg}[v] = 0$, sind alle Vorgänger von v bereits numeriert. \diamond

Nun die Korrektheit: Gibt der Algorithmus **true** aus, so wurden $|V|$ Knoten numeriert, d.h. alle Ecken. Nach der obigen Behauptung, Teil (ii) ist σ eine topologische Sortierung, also ist G kreisfrei.

Sei umgekehrt G kreisfrei. Dann gibt es eine Ecke mit $g^-(v) = 0$. Somit ist die Liste L beim ersten Durchlauf nicht leer. Es genügt nun zu zeigen, daß L beim k ten Durchlauf ($1 \leq k \leq n$) nicht leer ist.

Sei $k \leq n$. Nach der obigen Behauptung, Teil (a) gibt zu Beginn des k ten Durchlaufs $\text{indeg}[v]$ für $v \in V$ die Anzahl der noch nicht numerierten Vorgänger an. Da $k \leq n$, ist mindestens eine Ecke v_0 noch nicht numeriert.

Ist $\text{indeg}[v_0] > 0$, so gibt es einen noch nicht numerierten Vorgänger v_1 von v_0 . Ist $\text{indeg}[v_1] = 0$, so ist $v_1 \in L$, insbesondere $L \neq \emptyset$. Ansonsten gibt es einen noch nicht numerierten Vorgänger v_2 von v_1 . Da G kreisfrei ist, muß die Folge v_0, v_1, v_2, \dots mit einem v_i mit $\text{indeg}[v_i] = 0$ abbrechen. \square

Lösung zu Aufgabe 3.7

- (a) Ist der maximale Grad eines Graphen gleich Δ , dann kann jede Zusammenhangskomponente (die Richtung der Pfeile ist für diese Aufgabe unerheblich) des Graphen mit $\Delta + 1$ Farben durch einen Wellenfront-Algorithmus gefärbt werden. Die erste Ecke bekommt eine beliebige Farbe. Alle folgenden Ecken färbt man so, daß die Färbungseigenschaft nicht verletzt wird. Da die Ecke höchstens Δ viele Nachbarn hat, die zusammen nicht mehr als Δ verschiedene Farben besitzen, gibt es immer eine verbleibende Farbe, mit der die neue Ecke gefärbt werden kann.

- (b) Sei $G = (V, R)$. Sei l die Länge des längsten elementaren Weges in G . Wir definieren für alle $v \in V$:

$$f(v) := \max \left\{ |w| \mid \begin{array}{l} w \text{ ist elementarer Weg,} \\ \alpha(w) = v \end{array} \right\}$$

Behauptung: f ist eine $(l + 1)$ -Färbung auf G .

Beweis: Offenbar ist $f(V) \subset \{0, 1, \dots, l\}$. Es werden also nur $l + 1$ Farben vergeben. Nachzuweisen bleibt, daß benachbarte Ecken nie dieselbe Farbe erhalten.

Seien v, v' benachbart, o. E. $(v, v') \in R$. Sei $f(v) = f(v') = k$. Seien

$$\begin{aligned} w &= (v = v_0, v_1, \dots, v_k) \\ w' &= (v' = v'_0, v'_1, \dots, v'_k) \end{aligned}$$

die zugeordneten elementaren Wege, notiert durch ihre Spur. Wäre v nicht in der Spur von w' , so wäre

$$(v, v') \circ w'$$

ein elementarer Weg der Länge $k + 1$ von v aus im Widerspruch zur Definition. Also gilt $v = v'_j$ für ein j . Insbesondere ist v von v' durch den Weg $(v' = v'_0, v'_1, \dots, v'_j = v)$ erreichbar. Dann ist aber $(v' = v'_0, v'_1, \dots, v'_j = v) \circ (v, v')$ ein Kreis in G . Widerspruch. \square

Lösung zu Aufgabe 3.8

- Färbe die Ecken in beliebiger Reihenfolge mit den Farben $\{1, 2, \dots, k+1\}$. Da jede Ecke höchstens k Nachbarn hat, die höchstens k verschiedene Farben belegen, steht jeweils mindestens eine gültige Farbe zur Verfügung.
- Zu Färbungen f_1 von G_1 und f_2 von G_2 ist die Färbung $f: v \mapsto (f_1(v), f_2(v))$ eine gültige Färbung für den Vereinigungsgraphen.

Lösung zu Aufgabe 3.9

- Für $k \geq 2$ ist der K_k kritisch k -chromatisch.
- Die Kreise ungerader Länge sind kritisch 3-chromatisch.

Lösung zu Aufgabe 3.10

Betrachte zunächst Algorithmus A.5, der auf einem schwach zusammenhängenden Graphen $G = (V, R)$ mit einer beliebigen Ecke $v_0 \in V$ als zuerst zu besuchende Ecke aufgerufen werde.

(i) Behauptung: Algorithmus A.5 hat eine Laufzeit in $O(|R|)$.

Beweis: Jeder Pfeil (v, w) wird nur beim Pfeilbüschel $B^-(w)$ erforscht, und die Prozedur wird für jede Ecke höchstens ein Mal aufgerufen.

(ii) Behauptung: Auf einem stark zusammenhängenden Graphen wird jede Ecke als erreicht markiert. Jede Ecke hat genau einen ausgehenden roten Pfeil.

Algorithmus A.5 Algorithmus zur Markierung der Pfeile.

```

ERFORSCHEN(w)
1  for alle  $r \in B^-(w)$  do
2    setze  $v \leftarrow \alpha(r)$ 
3    if  $v$  ist noch nicht erreicht then
4      färbe  $(v, w)$  rot
5      markiere  $v$  als erreicht
6      ERFORSCHEN(v)
7    else
8      färbe  $(v, w)$  blau
9    end if
10 end for

```

Beweis: Angenommen, eine Ecke v sei nicht erreicht. Dann gibt es keine Ecke $w \in N(v)$, die erreicht ist, andernfalls wäre beim Erforschen von w auch über den Pfeil (v, w) die Ecke v erreicht worden. Induktiv folgt, daß keine Ecke in $E_G(v)$ erreicht sein kann. Wegen des starken Zusammenhangs gilt $E_G(v) = V$, also ist gar keine Ecke erreicht, Widerspruch. – Da beim Besuch der Ecke ein ausgehender Pfeil rot markiert wird, folgt die zweite Behauptung.

(iii) Für jede Ecke $v \in V$ gibt es einen Pfad von v nach v_0 , der nur rote Pfeile nutzt.

Beweis: Starte in v und laufe von der gerade aktuellen Ecke jeweils dem ausgehenden roten Pfeil nach weiter zur nächsten Ecke. Dieser Lauf endet entweder in v_0 , womit die Behauptung gezeigt wäre, oder er endet mit einem Kreis aus roten Pfeilen, der v_0 nicht enthält.

Sei nun w die Ecke des Kreises, die von Algorithmus ERFORSCHEN zuerst erforscht wird, r der entsprechende Pfeil. Dann wird r rot gefärbt, liegt aber nicht im Kreis, folglich gehen von w zwei verschiedene rot gefärbte Pfeile aus. Widerspruch.

Nach Ermittlung der Markierungen werde auf den Graphen der Algorithmus A.6 angewandt.

Algorithmus A.6 Algorithmus zum Durchlaufen

```

DURCHLAUFE-EULER-KREIS
Input: durch ERFORSCHEN markierter Graph  $G$ 
1  markiere alle Pfeile als nicht durchlaufen
2  wähle aktuelle Ecke  $v \leftarrow v_0$ 
3  while es gibt ungenutzte Pfeile in  $B^+(v)$  do
4    if es gibt ungenutzten blauen Pfeil  $r \in B^+(v)$  then
5      durchlaufe  $r$ 
6    else
7      durchlaufe roten Pfeil  $r \in B^+(v)$ 
8    end if
9    setze  $v \leftarrow \omega(r)$ 
10 end while

```

Es bleibt zu zeigen: Wenn der Graph Eulersch ist, dann durchläuft Algorithmus A.6 jeden Pfeil genau einmal. (Umgekehrt ist klar, wenn

der Algorithmus alle Pfeile durchläuft und mit dem Durchlauf im Ausgangspunkt endet, ist der Graph Eulersch.)

Wir bezeichnen mit L den Graphen, der durch die roten Pfeile induziert wird. Für $v \in V$ werde mit d_v die Distanz zu v_0 bezeichnet, d.h. die Anzahl der roten Pfeile auf dem Pfad von v nach v_0 .

Es ist klar, daß kein Pfeil zweimal durchlaufen wird. Wegen der Gradbedingung $g^-(v) = g^+(v)$ für einen Eulerschen Graphen kann jede besuchte Ecke auch wieder verlassen werden; der Durchlauf endet daher in der Ecke v_0 .

Wir zeigen: Für jedes $v \in V$ wird jeder Pfeil in $B^+(v)$ durchlaufen.

Induktion nach d_v . Für $d_v = 0$ ist $v = v_0$. Da der Algorithmus endete, wurden alle Pfeile in $B^+(v)$ durchlaufen. – Sei die Behauptung für d gezeigt, und sei $d_v = d + 1$. Betrachte den roten von v ausgehenden Pfeil $r = (v, w)$. Dann ist $d_w = d$. Alle von w ausgehenden Pfeile wurden nach I.V. durchlaufen, wegen der Gradgleichheit also auch alle eingehenden Pfeile, insbesondere auch r . Da der rote Pfeil zuletzt gewählt wird, folgt, daß alle Pfeile in $B^+(v)$ durchlaufen sein müssen.

Lösung zu Aufgabe 3.11

„ \Rightarrow “ Jeder Hamiltonsche Kreis in G ist auch einer in G' .

„ \Leftarrow “ Sei G' Hamiltonsch.

Annahme, G sei nicht Hamiltonsch.

Da G' Hamiltonsch ist, gibt es einen elementaren Weg w in G' , der jede Ecke genau einmal berührt, etwa $w = [v_1, \dots, v_n]$ mit O.B.d.A. $v_1 = u$ und $v_n = v$.

Betrachte die Mengen

$$A = \{v_i : 3 \leq i \leq n-1 \text{ und } (u, v_i) \in E\}$$

$$B = \{v_i : 3 \leq i \leq n-1 \text{ und } (v, v_{i-1}) \in E\}$$

Da $(u, v) \notin E$ gilt $|A| + |B| \geq d(u) + d(v) - 2 \geq n - 2$. Wegen $|A \cup B| \leq n - 3$ gibt es $v_i \in A \cap B$, also $(u, v_i) \in E$ und $(v, v_{i-1}) \in E$. Dann ist aber $w' := [u = v_1, \dots, v_{i-1}, v_n = v, v_{n-1}, \dots, v_i, v_1 = u]$ ein Hamiltonscher Kreis in G . Widerspruch! \square

Lösung zu Aufgabe 3.12

- a. Für $1 \leq n \leq 4$ ist der K_n eine gültige Lösung. Abbildung A.1 zeigt den Graphen G_n für den Fall, daß $n = 2m$ gerade ist. Falls $n = 2m - 1$ ungerade ist, ist die Ecke w_m zu entfernen, und die Kante (v_1, w_m) durch (v_1, v_m) zu ersetzen. Man überzeuge sich, daß die Anzahl der Kanten gleich $2n - 2$ ist.

Seien $s, t \in V$ gegeben. Es ist zu zeigen, daß es einen hamiltonschen Weg von s nach t gibt.

Im ersten Fall gelte $\{s, t\} = \{v_i, w_i\}$ oder $\{w_i, v_{i+1}\}$ für ein i . Dann liegen s und t benachbart auf dem in der Abb. A.1 unten dargestellten hamiltonschen Kreis $(v_1, w_1, v_2, w_2, \dots, v_m, w_m, v_1)$. Die Lösung ergibt sich durch Entfernen der Kante (s, t) aus dem Kreis.

Nun seien i und j so gewählt, daß gilt $s \in \{v_i, w_i\}$, $t \in \{v_j, w_j\}$ und $j > i$. Das ist durch Vertauschen von s und t immer möglich. Durch i und j sind folgende drei Teilwege im Graphen festgelegt

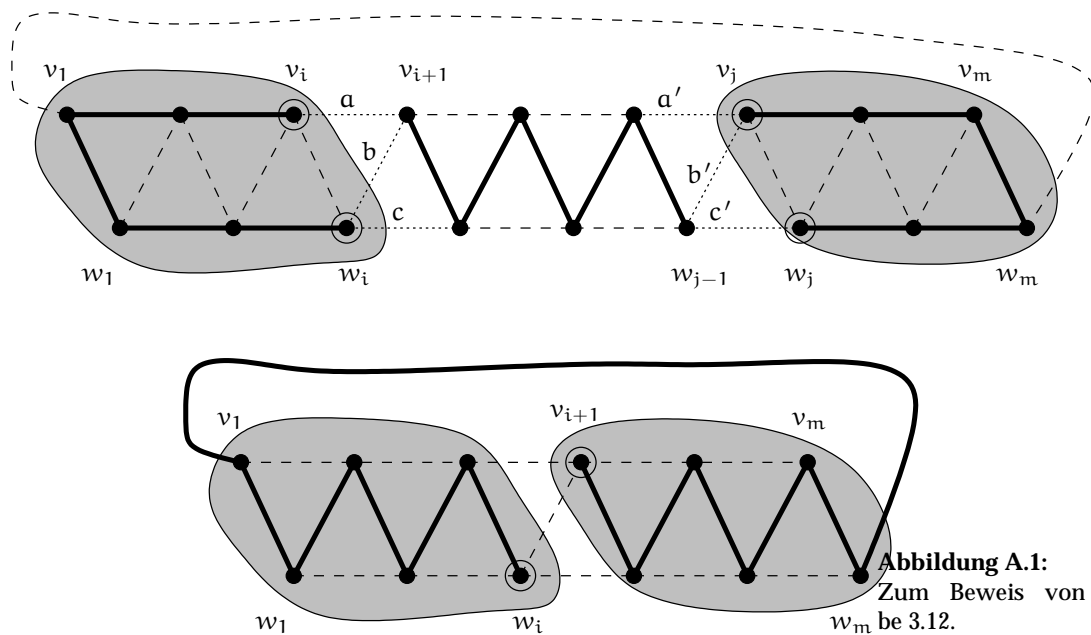


Abbildung A.1:
Zum Beweis von Aufga-
be 3.12.

(die Wege sind in Abb. A.1 oben mit dicken Linien ausgezeichnet):

- der Weg $l = (v_i, v_{i-1}, \dots, v_1, w_1, w_2, \dots, w_i)$ von v_i nach w_i , der genau die Knoten mit Indizes im Bereich $\{1, \dots, i\}$ besucht,
- der Weg $m = (v_{i+1}, w_{i+1}, v_{i+2}, w_{i+2}, \dots, v_{j-1}, w_{j-1})$ von v_{i+1} nach w_{j-1} , der genau die Knoten mit Indizes im Bereich $\{i+1, \dots, j-1\}$ besucht (für den Fall $j = i+1$ ist dieser Weg leer),
- der Weg $r = (v_j, v_{j+1}, \dots, v_m, w_m, w_{m-1}, \dots, w_j)$ von v_j nach w_j , der genau die Knoten mit Indizes im Bereich $\{j, \dots, m\}$ besucht (falls $n = 2m - 1$ ungerade ist, entfällt der Knoten w_m in r).

In der Abbildung sind ferner die Kanten $a = (v_i, v_{i+1})$, $b = (w_i, v_{i+1})$, $b' = (w_{j-1}, v_j)$ und $c' = (w_{j-1}, w_j)$ dargestellt. Für die Lage von s und t verbleiben folgende Fälle (falls m der leere Weg ist, entfallen die eingeklammerten Teilwege in der Lösung):

- $s = v_i, t = v_j$: Der Weg $l \circ (b \circ m) \circ c' \circ r^{-1}$ von s nach t ist hamiltonsch.
- $s = v_i, t = w_j$: Der Weg $l \circ (b \circ m) \circ b' \circ r$ von s nach t ist hamiltonsch.
- $s = w_i, t = v_j, j > i + 1$: Der Weg $l^{-1} \circ a \circ m \circ c' \circ r^{-1}$ von s nach t ist hamiltonsch.
- $s = w_i, t = w_j$: Der Weg $l^{-1} \circ a \circ (m \circ b') \circ r$ von s nach t ist hamiltonsch.

- b. Sei $G = (V, E)$ ein hamiltonsch zusammenhängender Graph. Wähle eine beliebige Kante $e = (v_1, v_2) \in E$. Da G hamiltonsch zusammenhängend ist, gibt es einen hamiltonschen Weg w in G von v_1

nach v_2 , und da G mehr als zwei Ecken enthält, benutzt w nicht die Kante e . Der Weg $w \circ e$ ist dann ein hamiltonscher Kreis.

Lösung zu Aufgabe 3.13

- a. Ist eine Folgerung aus Teilaufgabe b.
- b. Betrachte den reduzierten Graphen. Er besteht aus q Ecken und ist schwach zusammenhängend, nach Lemma 5.6 der Vorlesung enthält er mindestens $q - 1$ Pfeile. Diese Pfeilmenge kann injektiv auf Pfeile zwischen starken ZK in G abgebildet werden.

Betrachte eine starke Zusammenhangskomponente Z mit $|Z| \geq 2$. Da alle Ecken gegenseitig erreichbar sind, hat jede Ecke mindestens einen einlaufenden Pfeil. Dieser entspringt in einer anderen Ecke von Z , sonst trüge er nichts zur Erreichbarkeit innerhalb von Z bei. Somit finden sich in jeder starken Zusammenhangskomponente Z mindestens $|Z|$ viele weitere Pfeile.

Summierung liefert als untere Schranke für die Anzahl der Pfeile in G :

$$|R| \geq (q - 1) + \sum_{i=1}^p |Z_i| = q - 1 + |V| - (q - p) = |V| - 1 + p.$$

Lösung zu Aufgabe 3.14

Fallunterscheidung:

Für $d_G(v_1, v_3) = -\infty$ ist nichts zu zeigen.

Sei $d_G(v_1, v_3) \in \mathbb{R}$. Falls $v_2 \notin E_G(v_1)$ oder $v_3 \notin E_G(v_2)$, ist einer der Terme auf der rechten Seite gleich $+\infty$ und nichts zu zeigen. Andernfalls sind die Mengen W_{v_1, v_2} und W_{v_2, v_3} nicht leer. Da für zwei Wege $w_1 \in W_{v_1, v_2}$ und $w_2 \in W_{v_2, v_3}$ deren Konkatenation $w_1 \circ w_2 \in W_{v_1, v_3}$ erfüllt, und ferner $l(w_1 \circ w_2) = l(w_1) + l(w_2)$ gilt, folgt

$$\inf\{l(w) \mid w \in W_{v_1, v_3}\} \leq \inf\{l(w) \mid w \in W_{v_1, v_2}\} + \inf\{l(w) \mid w \in W_{v_2, v_3}\}.$$

Für $d_G(v_1, v_3) = +\infty$ ist $v_3 \notin E_G(v_1)$. Wäre $v_2 \in E_G(v_1)$ und gleichzeitig $v_3 \in E_G(v_2)$, dann auch $v_3 \in E_G(v_1)$ im Widerspruch zur Annahme. Also ist einer der Terme $d_G(v_1, v_2)$ oder $d_G(v_2, v_3)$ gleich $+\infty$ und die Ungleichung gilt.

Lösung zu Aufgabe 3.15

„ \Leftarrow “: Zu zeigen: für beliebige Ecken v, w und Kante e ist nach Entfernen der Kante e der Knoten w von v aus erreichbar. Wenn der Weg von v nach w im Ausgangsgraphen die Kante e nicht benutzt, ist nichts zu zeigen. Andernfalls kann in dem Weg e durch das Teilstück des Kreises, auf dem e liegt, ersetzt werden.

„ \Rightarrow “: Betrachte eine Kante $e = (v, w)$, die nicht auf einem Kreis liegt. Wäre nach Entfernen von e die Ecke w noch von v aus über einen Weg k erreichbar, so bildete k zusammen mit e einen Kreis. Widerspruch.

Lösung zu Aufgabe 3.16

- a. Sei $\chi := \chi(G)$. Wähle eine Teilmenge von $1 \leq k < \chi(G)$ Farbklassen. V_1 enthalte genau die Knoten, die mit den gewählten k Farben gefärbt sind. Dann ist offenbar $\chi(G[V_1]) \leq k$ und $\chi(G[V_2]) \leq \chi - k$. Gleichzeitig ist aber $\chi(G[V_1]) \geq k$: eine Färbung von $G[V_1]$ mit weniger als k Farben würde zusammen mit der $(\chi - k)$ -Färbung von $G[V_2]$ eine Färbung von G mit weniger als χ Farben induzieren. Ebenso ist $\chi(G[V_2]) \geq \chi - k$.
- b. Betrachte eine maximale Clique V_1 mit k Knoten. Angenommen, $G[V_2]$ wäre $(\chi - k)$ -färbbar. Dann induzierte diese Färbung zusammen mit der k -Färbung von V_1 eine gültige χ -Färbung von G . In dieser Färbung kommen die k Farben, die in der Clique verwendet werden, außerhalb der Clique nicht vor, die Farbklassen sind also einelementig.
- Wähle einen Knoten $x \in V_1$, eine Farbe f , die in V_2 vorkommt, und die (nichtleere) Menge T der Nachbarn von x in V_2 , die mit Farbe f gefärbt sind. Da die Clique maximal ist, gibt es zu jedem Knoten $t \in T$ einen Knoten $t' \in V_1$, der nicht zu t adjazent ist. Färbe t mit der Farbe von t' . Nun kann x mit Farbe f gefärbt werden, wodurch die ursprüngliche Farbklassse von x leer wird. Dies ergibt eine $(\chi - 1)$ -Färbung von G . Widerspruch.

Kapitel 4

Lösung zu Aufgabe 4.1

Es gilt stets $G' := T_u \circ T_v \circ G = T_v \circ T_u \circ G =: G''$.

Beweis: Es genügt zu zeigen, daß $(x, y) \in G' \Rightarrow (x, y) \in G''$. Sei daher $(x, y) \in G'$.

Falls $(x, y) \in T_v \circ G$, so folgt $(x, y) \in G''$ aus $G \subseteq T_u \circ G$ und der Monotonieeigenschaft der Tripeloperatoren:

$$G \subseteq T_u \circ G \Rightarrow T_v \circ G \subseteq T_v \circ T_u \circ G = G''.$$

Sonst ist $(x, y) \notin T_v \circ G$. Nach Definition des Tripeloperators T_u gilt dann entweder $(x, y) = (u, u)$ oder

$$(x, u) \in T_v \circ G \wedge (u, y) \in T_v \circ G.$$

Falls $(x, y) = (u, u)$, so gilt offenbar $(u, u) \in G''$. Ansonsten:

1. Fall: $(x, u) \in G \wedge (u, y) \in G$ Dann ist $(x, y) \in T_u \circ G$, also auch $(x, y) \in T_v \circ T_u \circ G = G''$.

2. Fall: $(x, u) \in G \wedge (u, y) \notin G$ Dann gilt nach Definition von T_v : $(u, v) \in G$ und $(v, y) \in G$. In diesem Fall gilt:

$$(x, u) \in G \wedge (u, v) \in G \Rightarrow (x, v) \in T_u \circ G$$

$$\begin{aligned} (x, v) \in T_u \circ G \wedge (v, y) \in G &\Rightarrow (x, v) \in T_u \circ G \wedge (v, y) \in T_u \circ G \\ &\Rightarrow (x, y) \in T_v \circ T_u \circ G = G''. \end{aligned}$$

3. Fall: $(x, u) \notin G \wedge (u, y) \in G$ Analog zum 2. Fall.

4. Fall: $(x, u) \notin G \wedge (u, y) \notin G$ Dann ist $(x, v) \in G$ und $(v, u) \in G$, sowie $(u, v) \in G$ und $(v, y) \in G$.

$$(x, v) \in G \wedge (v, y) \in G \Rightarrow (x, y) \in T_v \circ G \Rightarrow (x, y) \in T_v \circ T_u \circ G = G''.$$

Lösung zu Aufgabe 4.2

Sei ein gerichteter kreisfreier Graph $G = (V, R)$ gegeben. Da G kreisfrei ist, gibt es eine topologische Sortierung, d. h. wir können o. E. die Ecken so sortieren, daß gilt

$$r = (v_i, v_j) \in R \Rightarrow i < j. \quad (\text{A.2})$$

Seien nun $G_1 = (V, R_1)$ und $G_2 = (V, R_2)$ zwei verschiedene irreduzible Kerne von G . Dann ist die Menge $R_1 \setminus R_2$ nicht leer. Sei $r_1 = (v_i, v_j)$ beliebig gewählt.

Da $r_1 \notin R_2$, aber $r_1 \in R$, gibt es in G_2 einen Weg w_2 aus mindestens zwei Pfeilen mit $\alpha(w_2) = v_i$ und $\omega(w_2) = v_j$. Sei v_m eine beliebige Ecke auf diesem Weg. Aus Gleichung A.2 folgt mit Induktion

$$i < m < j. \quad (\text{A.3})$$

Betrachte nun den Teilweg w'_2 von w_2 mit $\alpha(w'_2) = v_i$ und $\omega(w'_2) = v_m$. Dieser Weg w'_2 ist auch ein Weg in G , also muß es im irreduziblen Kern G_1 einen Weg w'_1 geben, mit $\alpha(w'_1) = v_i$ und $\omega(w'_1) = v_m$. Der Weg w'_1 kann wegen Gleichung A.3 nicht über den Pfeil r_1 führen. Analog folgt die Existenz eines Weges w''_1 in G_1 mit $\alpha(w''_1) = v_m$ und $\omega(w''_1) = v_j$, der ebenfalls nicht über r_1 führen kann.

Somit besteht zu r_1 in G_1 der Umweg $w'_1 \circ w''_1$, und der Pfeil r_1 ist redundant im Widerspruch zur Annahme. \square

Lösung zu Aufgabe 4.4

Da G stark zusammenhängend ist, muß jede Ecke in G_* Innengrad mindestens 1 haben, also folgt $|R_*| \geq |V|$.

Wähle eine beliebige Ecke $v \in V$. Nun konstruiere $G_{\text{out}} = (V, R_{\text{out}})$ als spannenden Wurzelbaum von G_* mit Wurzel v , ebenso $G_{\text{in}} = (V, R_{\text{in}})$ als spannenden Wurzelbaum von G_*^{-1} mit Wurzel v .

Beide Bäume haben je $|V| - 1$ Pfeile. Mit $R' := R_{\text{out}} \cup R_{\text{in}}^{-1}$ folgt $|R'| \leq 2(|V| - 1)$.

Offenbar ist $G' = (V, R')$ Partialgraph von G_* . Wegen $E_{G_{\text{out}}}(v) = V$ einerseits und $v \in E_{G_{\text{in}}^{-1}}(v')$ für alle $v' \in V$ andererseits ist G' stark zusammenhängend. Somit gilt sogar Gleichheit $G' = G_*$, denn andernfalls fänden sich in $R_* \setminus R'$ noch redundante Pfeile.

Kapitel 5

Lösung zu Aufgabe 5.1

Wir nennen eine Kante, die keine Brücke ist, eine *innere Kante*.

„1 \Rightarrow 2“ Sei G unizyklisch. Dann ist G zusammenhängend. Wähle eine Kante e auf dem Kreis. Durch Entfernen von e entsteht ein kreisfreier Graph, der immer noch zusammenhängend ist. Dies ist ein Baum.

„2 \Rightarrow 3“ Sei $G - e$ ein Baum. Dann ist $G - e$ zusammenhängend und besteht aus $|V| - 1$ Kanten. Durch Hinzunahme von e bleibt der Zusammenhang erhalten, und es gilt $|E| = |V|$.

„3 \Rightarrow 4“ Da $|E| = |V|$ ist, ist G kein Baum. Also gibt es innere Kanten. Da der Zusammenhang durch Entfernen einer inneren Kante nicht zerstört wird, liegt sie auf einem Kreis. Gleichzeitig entsteht aber durch Wegnehmen einer beliebigen inneren Kante ein zusammenhängender Graph mit $|V| - 1$ Kanten, also ein Baum. Dieser ist kreisfrei. Folglich müssen alle inneren Kanten auf demselben elementaren Kreis liegen, da sonst nach Wegnehmen einer inneren Kante noch ein Kreis verbliebe. – Alle Kanten auf diesem Kreis sind innere Kanten, da eine Brücke nicht auf einem Kreis liegen kann.

„4 \Rightarrow 1“ Sei G ein zusammenhängender Graph, dessen innere Kanten auf einem elementaren Kreis liegen. In G kann es keinen zweiten Kreis geben, denn Brücken können nicht auf einem Kreis liegen. Somit hat G genau einen Kreis, ist also unizyklisch. \square

Lösung zu Aufgabe 5.2

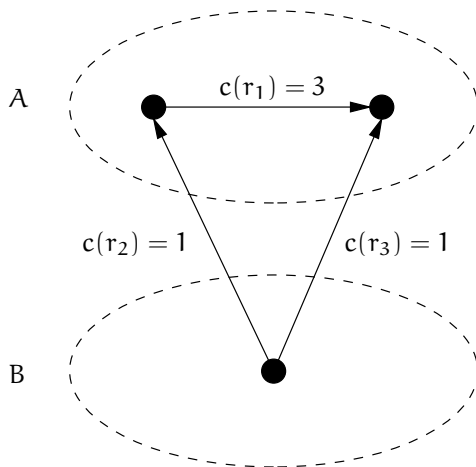
(a) Die Aussage ist richtig und folgt aus Satz 5.18:

Da bei der Definition des MST für gerichtete Graphen die Richtung der Pfeile „neutralisiert“ wird, gilt Satz 5.18 analog auch für gerichtete Graphen.

Sei nun $r^* \in \sigma(A, B)$ eine leichteste Kante im Schnitt. Mit $R' = \emptyset$ gilt $R' \subseteq R$ und $R' \cap \sigma(A, B) = \emptyset$

Damit folgt nach Satz 5.18, daß r^* sicher ist für R' , d.h. es gibt einen MST T mit $R' \cup r^* = r^* \subseteq T$. \square

(b) Die Behauptung ist falsch. Im folgenden Gegenbeispiel ist $T_A = (A, \{r_1\})$ MST von $G[A]$ und $T_B = (B, \emptyset)$ MST von $G[B]$. Der MST von G ist aber $T = (V, \{r_2, r_3\})$, also $T_A \not\subseteq T$.



Lösung zu Aufgabe 5.3

Widerspruchannahme: Der Greedy-Algorithmus liefere für jede Gewichtsfunktion $c: S \rightarrow \mathbb{R}$ eine maximale unabhängige Menge $M^* \in \mathcal{F}$

mit minimalem Gewicht $c(M^*)$, aber \mathcal{U} ist kein Matroid.

Wir konstruieren nun eine Gewichtsfunktion und zeigen, daß es für diese Funktion eine maximale unabhängige Menge M' gibt mit $c(M') < c(M^*)$.

Da \mathcal{U} kein Matroid ist gibt es nach Lemma 5.10 ein $A \subseteq S$, sowie $D, E \in \mathcal{F}$ mit D, E sind maximal bzgl. A und $n := |D| > |E| =: m$. Sei nun $c: S \rightarrow \mathbb{R}$ definiert durch

$$c(e) := \begin{cases} -(1 + \epsilon) & \text{für } e \in E, \text{ wobei } 0 \leq \epsilon < \frac{1}{m} \\ -1 & \text{für } e \in D - E \\ 0 & \text{sonst} \end{cases}$$

da $n \geq m + 1$ gilt:

$$c(E) = \sum_{e \in E} c(e) > m(-1 - \frac{1}{m}) = -(m + 1) > -n \geq \sum_{d \in D} c(d) = c(D)$$

Sei M_k die Menge nach dem Hinzufügen des k ten Elements durch den Algorithmus (vgl. Beweis zu Satz 5.13).

Nach der Sortierung der Elemente von S im ersten Schritt des Algorithmus gilt $e_1, \dots, e_m \in E$ aufgrund der Konstruktion von c . Folglich ist $M_1 = \{e_1\} \subseteq E$. Induktiv folgt weiter, daß $M_i = M_{i-1} + e_i$ für $i = 2, \dots, m$, da $M_{i-1} + e_i \subseteq E \in \mathcal{F}$ und mit Unabhängigkeitssystem \mathcal{U} gilt $M_{i-1} + e_i \in \mathcal{F}$.

Dies impliziert $E = M_m \subseteq M^*$ und somit $c(M^*) \leq c(E)$ nach Konstruktion von c . Gleichzeitig gilt: $M^* \cap (D - E) = \emptyset$, denn gäbe es ein $e \in M^* \cap (D - E)$, dann wäre $(E + e) \subseteq M^* \in \mathcal{F}$ und mit Definition von \mathcal{U} wäre $(E + e) \in \mathcal{F}$ im Widerspruch zur Maximalität von E bzgl. A .

Da $M^* \cap (D - E) = \emptyset$, folgt $c(M^* - E) = 0$ und somit $c(M^*) = c(E) > c(D)$.

Da $D \in \mathcal{F}$, existiert eine maximale unabhängige Menge $M' \supseteq D$ mit $c(M') = c(D) < c(E) = c(M^*)$ im Widerspruch zur Minimalität von $c(M^*)$. \square

Lösung zu Aufgabe 5.4

(a) Wir betrachten folgenden Algorithmus:

Zur Laufzeit: der Algorithmus verwendet die Union-Find-Datenstrukturen, wie sie in der Vorlesung erwähnt wurden. Es werden n MAKESET-Operationen und $3m$ UNION- und FINDSET-Operationen durchgeführt, was nach Vorlesung in einer Laufzeit von $O(3m \cdot \alpha(3m, n)) = O(m\alpha(m, n))$ möglich ist.

(b) Idee: Kruskal-Algorithmus. Wir sortieren die Pfeile in linearer Zeit nach ihrem Gewicht. Das ist möglich, weil die Gewichtsfunktion nur Werte in $\{1, \dots, n\}$ annimmt. Dann werden die Pfeile der Reihe nach in den Baum eingefügt. Der Test auf Kreisfreiheit wird wieder mit Union-Find-Datenstrukturen durchgeführt, denn offenbar schließt eine Kante genau dann einen Kreis, wenn Anfangs- und Endecke bereits in derselben Zusammenhangskomponente des Waldes liegen.

Laufzeit: es werden n MAKESET-Operationen und weniger als $3m$ andere Operationen durchgeführt, was die Behauptung zeigt. \square

Algorithmus A.7 Ermittlung von schwachen Zusammenhangskomponenten

BESTIMME KOMPONENTEN SCHWACHEN ZUSAMMENHANGS

Input: Ein Graph $G = (V, R)$

```

1  for all  $v \in V$  do {Initialisierung}
2    MAKESET( $v$ )
3    Korb( $v$ )  $\leftarrow \emptyset$ 
4    eingekorbt( $v$ )  $\leftarrow$  false
5  end for
6  for all  $\tau = (v, w) \in R$  do {Verschmelzung der ZK}
7    UNION( $v, w$ )
8  end for
9  for all  $\tau \in R$  do {Sammeln der nicht isolierten Ecken in Körben}
10   Korb(FINDSET( $\alpha(\tau)$ ))  $\leftarrow$  Korb(FINDSET( $\alpha(\tau)$ ))  $\cup$   $\{\alpha(\tau), \omega(\tau)\}$ 
11   eingekorbt( $\alpha(\tau)$ )  $\leftarrow$  eingekorbt( $\omega(\tau)$ )  $\leftarrow$  true
12 end for
13 for all  $v \in V$  do {Ausgabe}
14   if Korb( $v$ )  $\neq \emptyset$  then
15     Gib alle Ecken in Korb( $v$ ) als eine ZK aus
16   else if eingekorbt( $v$ ) = false then
17     Gib  $\{v\}$  als eine ZK aus
18   end if
19 end for

```

Algorithmus A.8 Routine zur MST-Berechnung

BERECHNE MST

Input: Ein Graph $G = (V, R, \alpha, \omega)$,
eine Pfeilbewertungsfunktion $c: R \rightarrow \{1, \dots, n\}$

```

1  for all  $v \in V$  do {Initialisierung}
2    MAKESET( $v$ )
3  end for
4  for all  $i = 1, \dots, n$  do
5    Korb( $i$ )  $\leftarrow \emptyset$ 
6  end for
7  for all  $\tau \in R$  do {Pfeile sortieren}
8    Korb( $c(\tau)$ )  $\leftarrow$  Korb( $c(\tau)$ )  $\cup$   $\{\tau\}$ 
9  end for
10 for all  $i = 1, \dots, n$  do
11   for all  $\tau \in$  Korb( $i$ ) do
12     if FINDSET( $\alpha(\tau)$ )  $\neq$  FINDSET( $\omega(\tau)$ ) then
13        $T \leftarrow T \cup \{\tau\}$  {Pfeil in MST aufnehmen}
14       UNION( $\alpha(\tau), \omega(\tau)$ ) {Zusammenhang updaten}
15     end if
16   end for
17 end for

```

Lösung zu Aufgabe 5.5

- a. Die erste Ungleichung ist klar nach Definition. Zur Ungleichung $d \leq 2r$: sei $z \in V$ eine Ecke des Zentrums, also $e(z) = r(G)$. Dann gilt für alle Eckenpaare $v, w \in V$: $d(v, w) \leq d(v, z) + d(z, w) \leq r + r \leq 2r$.
- b. Seien r und d gegeben. Konstruiere einen Graphen wie in Abb. A.2 gezeigt. Der Graph besteht aus einem Kreis der Länge $2b$, wobei $b := 2r - d \geq 0$ gewählt wird. An jeder Ecke des Graphen werde ein Pfad der Länge $a := d - r \geq 0$ angehängt.

Wegen der Symmetrie genügt es, die Exzentrizitäten nur für die Ecken auf einem der Pfade, etwa dem Pfad von w_0 nach v_0 , zu untersuchen. Offenbar ist für alle Ecken dieses Pfades die Ecke w_b die am weitesten entfernte. Damit ist

$$e(w_0) = d(w_0, w_b) = a + b + a = (d - r) + (2r - d) + (d - r) = d \quad \text{und}$$

$$e(v_0) = d(v_0, w_b) = b + a = (2r - d) + (d - r) = r.$$

Für alle anderen Ecken auf dem Pfad liegt die Exzentrizität zwischen diesen Extremen.

Sonderfälle:

- Für $d = 2r$ wird $a = r$ und $b = 0$. Der Graph degeneriert zu einem Pfad der Länge $2r$. Die zentrale Ecke hat Exzentrizität r , die beiden Endecken des Pfades haben Exzentrizität $2r = d$.
- Für $d = 2r - 1$ wird $a = r - 1$ und $b = 1$. Der Kreis degeneriert zu einer Kante. Damit ist der Graph ein Pfad der Länge $2r - 1$. Die beiden zentralen Ecken haben Exzentrizität r , die beiden Endecken des Pfades haben Exzentrizität $2r - 1 = d$.
- Für $d = r$ wird $a = 0$ und $b = r$. Der Graph degeneriert zu einem Kreis der Länge $2r$. Jede Ecke hat gleiche Exzentrizität r .

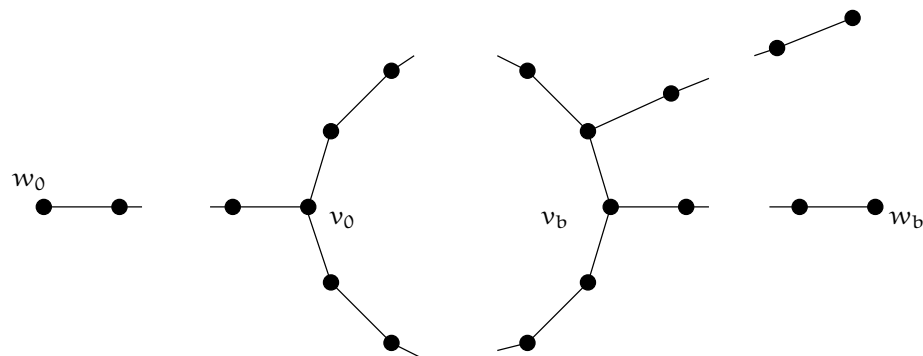


Abbildung A.2:
Zum Beweis von Aufgabe 5.5.

Lösung zu Aufgabe 5.6

- a. Da $r(G) < \infty$, gibt es eine Ecke v mit endlicher Exzentrizität $e(v)$. Diese Ecke hat Außengrad $g^+(v) = 0$ (andernfalls wäre v vom Endpunkt des ausgehenden Pfeiles nicht erreichbar im Widerspruch zur endlichen Exzentrizität). Damit gilt $E_G(v) = \emptyset$ und folglich $e(v') = \infty$ für alle $v' \neq v$.
- b. Sei v ein innerer Knoten von T . Da die Menge der Knoten $\{x \mid d_T(x, v) = e(v)\}$ nur Blätter enthält, erniedrigt sich die Exzentrizität von v beim Entfernen der Blätter. Andererseits liegt auf dem Pfad von x nach v in T nur ein Blatt (nämlich x), daher erniedrigt sich die Länge des Pfades um eins. Damit gilt: die Exzentrizität aller inneren Knoten von T wird beim Entfernen der Blätter gleichmäßig um 1 erniedrigt. Nach Definition nimmt auch der Radius um 1 ab. Das Zentrum bleibt also gleich, falls nicht Knoten daraus entnommen wurden.

Solange innere Knoten vorhanden sind (das ist äquivalent zu der Bedingung $|V| \geq 3$), enthält das Zentrum keine Blätter: die Exzentrizität des Nachbarn eines Blattes ist immer um 1 kleiner als die des Blattes selbst.

- c. Das Entfernen von Blättern zerstört den Zusammenhang im Restgraphen nicht, der Restgraph bleibt also ein Baum. Solange $|V| \geq 3$, kann daher das Entfernen wiederholt angewandt werden. Das Verfahren endet, wenn $|V| \leq 2$, also mit zwei adjazenten Ecken oder einer einzelnen Ecke, die dann das Zentrum darstellen.

Bei allgemeinen Bewertungen ist schon die Aussage in Aufgabe b nicht mehr gültig (betrachte etwa Weg der Kantengewichte 1, 1, 2). Die Aussage in Aufgabe c wird etwa bei $d_G \equiv 0$ falsch.

Lösung zu Aufgabe 5.7

- a. Bemerkung: Zum Beweis wird das Ergebnis von Aufgabe 5.8 vorausgesetzt.

Sei T ein Baum, der vom Kruskal-Algorithmus berechnet wird, und sei $L(T) = (c(e_1), \dots, c(e_m))$. Sei ferner T' ein weiterer MST mit $L(T') = (c(e'_1), \dots, c(e'_m))$. Wegen der Sortierung durch den Algorithmus ist $M_k = \{e_1, \dots, e_k\}$ für alle k . Mit Aufgabe 5.8 folgt dann

$$c(e_k) = c_{\max}(M_k) \leq c_{\max}(\{e'_1, \dots, e'_k\}) = c(e'_k) \quad \text{für alle } k = 1, \dots, m, \quad (\text{A.4})$$

und da beide Bäume gleiches Gewicht haben, folgt mit $\sum_k c(e_k) = \sum_k c(e'_k)$ die Gleichheit

$$c(e_k) = c(e'_k) \quad \text{für alle } k = 1, \dots, m, \quad (\text{A.5})$$

also auch $L(T) = L(T')$.

- b. Durch die Injektivität folgt aus der Eindeutigkeit der sortierten Gewichtslisten auch die Eindeutigkeit der Kantenmenge selbst.

- c. Sei $c_1 \leq \dots \leq c_p$ die sortierte Liste der im Graphen auftretenden Pfeilgewichte. Wir bezeichnen mit G_i den Bottleneck-Graphen zum Gewicht c_i .

Wir zeigen nun durch Induktion: Der Wegwerf-Algorithmus (WWA) findet einen minimalen spannenden Wald in G_i .

Induktionsanfang: Für $i = 1$ besteht der Graph nur aus gleichschweren Kanten. Der WWA endet mit einem Wald, und da er nur Kanten entfernt, falls sie einen Kreis bilden, erhält er die Spann-Eigenschaft, terminiert also mit einem spannenden Wald. Da nach den Matroid-Eigenschaften alle spannenden Wälder gleiche Kardinalität haben, folgt die Behauptung für $i = 1$.

Induktionsschluß: Sei die Behauptung für alle $j \leq i$ gezeigt. Betrachte G_{i+1} . Bezeichne mit C' die Menge der Kreise, deren schwerste Kanten Gewicht c_{i+1} tragen, und mit C die Menge der übrigen Kreise. Beim Betrachten eines Kreises aus C' ist für die Entscheidung, welche Kante entfernt werden soll, allein die Menge der involvierten Kanten vom Gewicht c_{i+1} ausschlaggebend. Diese ist unveränderlich, wenn leichtere Kanten aus dem Graphen entfernt werden (evtl. wird der betrachtete Kreis durch Umwege länger). Also können wir annehmen, daß der WWA in einer ersten Phase die Kreise in C betrachtet und in einer zweiten Phase die Kreise in C' .

Beim Betrachten der Kreise in C spielen Kanten vom Gewicht c_{i+1} keine Rolle; der WWA arbeitet während der ersten Phase in G_{i+1} genauso wie in G_i , akzeptiert also eine Menge von Kanten in G_i , die einen spannenden Wald in G_i ergeben; dieser ist gewichtsminimal nach Induktionsvoraussetzung. Nach Konstruktion terminiert der WWA nach der zweiten Phase mit einem spannenden Wald von G_{i+1} .

Der Kruskal-Algorithmus auf G_{i+1} findet zunächst einen MSF auf G_i und ergänzt diesen zu einem MSF in G_{i+1} . Durch Vergleich der Kantenzahlen ergibt sich, daß der vom WWA gefundene spannende Wald ebenfalls minimal ist.

Lösung zu Aufgabe 5.8

Wir zeigen, daß der vom Kruskal-Algorithmus im k -ten Schritt erzeugte Wald M_k kleinstes Flaschenhalsgewicht unter allen k -elementigen Kantenmengen hat, also

$$c_{\max}(M_k) = \min\{c_{\max}(M) \mid |M| = k\} \quad \text{für alle } k = 1, \dots, m. \quad (\text{A.6})$$

Für $k = 1$ ist die Behauptung richtig, da der Algorithmus das leichteste Element wählt.

Sei die Behauptung für k gezeigt. Sei $M_{k+1} = M_k + e$ die Kantenmenge, die der Algorithmus im nächsten Schritt wählt. Widerspruchannahme: Es gibt eine Menge $M \in F$ mit $|M| = k + 1$, aber $c_{\max}(M) < c_{\max}(M_{k+1}) = c(e)$.

Nach den Matroideigenschaften gibt es ein $e' \in M \setminus M_k$, so daß $M_k + e' \in F$. Offenbar ist $c(e') \leq c_{\max}(M)$.

Somit folgt $c(e') < c(e)$, also wurde e' vom Greedy-Algorithmus in einem Schritt $j < k$ bereits betrachtet, aber verworfen, d. h. es gilt $M_j + e' \notin F$.

Gleichzeitig ist aber $M_j + e' \subset M_k + e'$, und wegen $M_k + e' \in F$ folgt auch $M_j + e' \in F$. Dieser Widerspruch zeigt die Behauptung (A.6).

Kapitel 6

Lösung zu Aufgabe 6.1

- (a) Man starte in einer beliebigen Ecke s eine DFS auf G . Nach Satz 6.1 ist der Vorgängergraph G_π dann ein Wald, wobei jede schwache ZK von G_π ein Wurzelbaum ist. Nach Satz 6.6 sind alle Ecken aus einer ZK von G im gleichen Wurzelbaum von G_π . Da G stark zusammenhängt, ist G_π somit ein Wurzelbaum. Ferner ist G_π ein Partialgraph von G und mit Definition 5.4 folgt: G_π ist spannender Wurzelbaum von G mit Wurzel s , da $\pi(s) = \text{NIL}$. \square
- (b) Sei $G_\pi = (V, R_\pi)$ ein spannender Wurzelbaum von G mit Wurzel s gemäß Teil (a).

Da G_π spannender Baum ist, gilt: $|R_\pi| = |V| - 1$.

Wir betrachten nun den inversen Graphen G^{-1} zu G . Da G stark zusammenhängt, muß dies auch für G^{-1} gelten. In G^{-1} existiert dann ein spannender Wurzelbaum $T = (V, R')$ mit Wurzel s gemäß Aufgabenteil (a). Setze

$$R'' := \{(v, u) : (u, v) \in R'\}.$$

Der Partialgraph $H = (V, R_\pi \cup R'')$ von G ist dann stark zusammenhängend und besitzt höchstens $2(|V| - 1)$ Pfeile. Der starke Zusammenhang folgt, da in H jede Ecke von s aus erreichbar ist (wegen $G_\pi \sqsubseteq H$) und umgekehrt s von jeder Ecke aus erreichbar ist (da s in (V, R'') von jeder Ecke aus erreichbar war). Streicht man aus H alle redundanten Pfeile (etwa nach dem Wegwerfalgorithmus), so erhält man einen irreduziblen Kern von G mit höchstens $2|V| - 2$ Pfeilen. \square

Lösung zu Aufgabe 6.2

Wir führen eine Induktion nach der Anzahl k der grauen Ecken:

(Induktionsanfang) $k = 1$:

Gibt es zum Zeitpunkt $d[v]$ nur eine graue Ecke, so wurde DFS-VISIT(v) aus der DFS-Hauptprozedur aufgerufen. Also ist v Wurzel und besitzt keine Vorfahren.

(Induktionsschritt) $k \rightarrow k + 1$:

Betrachte den Zeitpunkt, zu dem die $(k + 1)$ -te Ecke grau gefärbt wird. DFS-VISIT(v) wurde dann rekursiv aufgerufen. Also wurde vor dem Aufruf $\pi(v) = u$ gesetzt, wobei u grau war.

Zum Zeitpunkt $d[u]$ gab es k graue Ecken, die nach (IV) genau die Vorfahren von u waren. Da u der einzige direkte Vorgänger von v ist, sind diese Ecken sowie u genau die Vorfahren von v . Da u zum Zeitpunkt $d[v]$ noch nicht vollständig erforscht ist, sind diese Ecken noch grau. \square

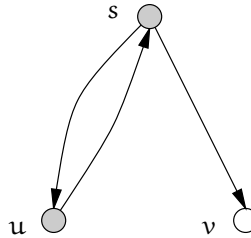
Nach Satz 6.1 sind die ZK von G_π Wurzelbäume. Daher gibt es genau einen Weg von der Wurzel s nach v und alle Ecken auf diesem Weg sind Vorfahren von v . \square

Lösung zu Aufgabe 6.3

Wenn $d[u] < d[v]$, so gilt $d[u] < d[v] < f[v] < f[u]$, da u noch grau ist, wenn v bereits schwarz geworden ist. Nach dem Intervallsatz ist dann v Nachkomme von u . Somit ist r eine Forward Edge.

Sei nun $d[u] > d[v]$. Wäre r eine Forward Edge, so wäre v Nachkomme von u im DFS-Wald. Dann folgt aber nach dem Intervallsatz, daß $d[u] < d[v] < f[v] < f[u]$ gilt. Dies ist ein Widerspruch zu $d[u] > d[v]$.

Die zweite Behauptung in der Aufgabe ist falsch: Wenn im folgenden Gegenbeispiel von s aus zuerst u und danach v entdeckt wird, ist der Pfeil von u nach s nicht in G_π enthalten, aber $d[s] < d[u] < d[v]$ und $v \in E_G(u)$.



Lösung zu Aufgabe 6.4

Idee: Wir nutzen den Algorithmus 6.4, der starke Zusammenhangskomponenten berechnet. Jede dieser Komponenten bildet eine Ecke im reduzierten Graphen. Bei der Ermittlung der Pfeile im reduzierten Graphen kann man für jeden Pfeil des ursprünglichen Graphen den entsprechenden Pfeil in den reduzierten einfügen. Dabei treten jedoch möglicherweise Parallelen auf. Wenn man die Pfeile vorher (in linearer Zeit) nach den Zusammenhangskomponenten sortiert hat, kann man diese Parallelen von vornherein ausschließen.

Betrachte Algorithmus A.9. Das Sortieren nach der Nummer der ZK der Zielecke der Pfeile sorgt dafür, daß nach Zeile 6 in der Liste L alle Pfeile hintereinanderstehen, deren Nummer der Ziel-ZK gleich ist. Das Sortierverfahren ist stabil. Nachdem dann nach der Nummer der Start-ZK sortiert wurde, ist also in Zeile 11 erreicht, daß alle Pfeile mit gleichen Anfangs- und End-ZK-Nummern hintereinander in der Liste L stehen. Damit können in Zeile 15 Parallelen dadurch ausgeschlossen werden, daß der aktuelle Pfeil nur mit dem zuletzt betrachteten Pfeil verglichen wird.

Zur Laufzeit: Zwei Listen können in konstanter Zeit konkateniert werden. Damit ist die Initialisierung der Listen in den Zeilen 2 und 7 und die Konkatenation in den Zeilen 6 und 11 in einer Zeit von $\mathcal{O}(p) \subseteq \mathcal{O}(n)$ möglich. Die übrigen **for**-Schleifen haben jeweils eine Laufzeit von $\mathcal{O}(m)$. \square

Kapitel 7

Lösung zu Aufgabe 7.1

- Sei w kürzester Weg, $w = w_1 \circ w_2$. Wäre w_1 kein kürzester Weg, dann existierte Weg w'_1 mit $\alpha(w_1) = \alpha(w'_1)$, $\omega(w'_1) = \omega(w_1)$ und $c(w'_1) < c(w_1)$. Dann wäre $w'_1 \circ w_2$ ein kürzerer Weg als w .

Algorithmus A.9 Routine zur Berechnung des reduzierten Graphen

ALGORITHMUS BERECHNE REDUZIERTEN GRAPHEN

Input: ein Graph $G = (V, R)$

- 1 rufe Algorithmus 6.4 zur Berechnung der starken Zusammenhangskomponenten auf. Sei p die Anzahl der ZKs. Anstelle der Ausgabe der ZK versieh jede Ecke v mit einer Nummer $z[v]$ ($1 \leq z[v] \leq p$), die der Nummer der ZK entspricht.
- 2 Initialisiere die Listen $L_1 \leftarrow \dots \leftarrow L_p = \emptyset$
- 3 **for all** $r = (v, w) \in R$ **do** {Sortieren nach der End-Ecke der Pfeile}
- 4 hänge r an die Liste $L_{z[w]}$ an
- 5 **end for**
- 6 Konkateniere die Listen $L \leftarrow L_1 + \dots + L_p$.
- 7 Initialisiere die Listen $L_1 \leftarrow \dots \leftarrow L_p = \emptyset$
- 8 **for all** $r = (v, w) \in L$ **do** {Sortieren nach der Anfangsecke der Pfeile}
- 9 hänge r an die Liste $L_{z[v]}$ an
- 10 **end for**
- 11 Konkateniere die Listen $L \leftarrow L_1 + \dots + L_p$.
- 12 Setze $\hat{V} \leftarrow \{1 \dots, p\}$ {Berechnen von $\hat{G} = (\hat{V}, \hat{R})$ }
- 13 Setze $\alpha \leftarrow \omega \leftarrow 0$
- 14 **for all** $r = (v, w) \in L$ **do**
- 15 **if** $z[v] \neq \alpha$ oder $z[w] \neq \omega$ **then**
- 16 füge den Pfeil $(z[v], z[w])$ in \hat{R} ein
- 17 **end if**
- 18 setze $\alpha \leftarrow z[v]$ und $\omega \leftarrow z[w]$
- 19 **end for**

Output: Der reduzierte Graph $\hat{G} = (\hat{V}, \hat{R})$.

- b. Baum T_s wird iterativ aufgebaut. Starte mit $T_s = (\{s\}, \emptyset)$. Iteration für $v \in V$, $v \neq s$: Sei (r_1, \dots, r_k) kürzester Weg von s nach v . Wähle j maximal, so daß für $v_j = \omega(r_j)$ gilt: $g^-(v_j) = 1$ (wenn kein solches j existiert, setze $j = 0$ und $v_0 = s$). Füge den Teilweg (r_{j+1}, \dots, r_k) dem Baum zu.

Man zeigt durch Induktion: Der entstehende Graph T_s ist ein Wurzelbaum mit Wurzel s , und es gilt $\text{dist}_T(s, v) = \text{dist}_G(s, v)$.

Lösung zu Aufgabe 7.2

- (a) Wir nennen im Folgenden einen Weg w mit $s(w) = (v_1, \dots, v_{n+1})$ einen *Grenzweg*, falls $v_1, \dots, v_n \in S$ und $v_{n+1} \in V \setminus S$ sind. Sei außerdem S_i die Menge S nach der i -ten Iteration der **while** Schleife. Es gelten folgende Invarianten:

- (i) Für alle $u \in S_i$ gilt $d[u] = \delta(s, u)$ und für den kürzesten Weg w von s nach u ist $s(w) \subseteq S_i$, falls er existiert.
- (ii) Für alle $u \in V \setminus S_i$ gilt

$$d[u] = \min\{\ell(w) : w \text{ ist Grenzweg mit } \alpha(w) = s \text{ und } \omega(w) = u\}$$

bzw. $d[u] = \infty$, falls kein solcher Grenzweg existiert.

Beweis: Induktion über $|S_k| = k$.

(Induktionsanfang): $k = 1$

Da $d[s] = 0$ folgt $S_1 = \{s\}$ und (i) gilt offensichtlich. Nach der for Schleife ab Zeile 12 gilt für alle $u \in V \setminus S_1$, daß

$$d[u] = \min\{\ell(r) : r \in \text{Adj}[s] \text{ und } \omega(r) = u\}$$

bzw. $d[u] = \infty$, falls es kein $r \in \text{Adj}[s]$ gibt mit $\omega(r) = u$. Da $\text{Adj}[s]$ alle Grenzwege enthält folgt Aussage (ii).

(Induktionsvoraussetzung): Es gelten die Invarianten (i) und (ii) für ein k .

(Induktionsschritt): $k \rightarrow k + 1$

Sei $S_{k+1} = S_k \cup \{v\}$. Falls $d[v] = \infty$, dann gilt für alle Heap-Elemente u , daß $d[u] = \infty$. Mit IV (ii) folgt, daß es für alle $u \in V \setminus S_k$ keinen Grenzweg von s nach u gibt. Somit ist $v \notin E_G(s)$ und (i) gilt. Desweiteren ist dann die Bedingung in Zeile 12 nie erfüllt. Es finden also keine DECREASE-KEY Operationen statt und somit folgt die Gültigkeit von (ii) nach IV.

Sei also im folgenden $d[v] \neq \infty$. Annahme: $\delta(s, v) < d[v]$, d.h. es existiert ein Weg w von s nach v mit $\ell(w) < d[v]$.

Nach IV (ii) hat für S_k der minimale Grenzweg von s nach v die Länge $d[v]$. Somit ist w kein Grenzweg für S_k .

Sei u das erste Element der Spur von w , das nicht in S_k liegt und sei w' der Teilweg von w mit $s(w') = (s, \dots, u)$. Dann ist w' ein Grenzweg von s nach v und mit IV (ii) folgt $\ell(w') \geq d[u]$.

Da im $(k + 1)$ ten Schritt v das minimale Heap-Element war, gilt $d[u] \geq d[v]$. Da ℓ eine nicht negative Gewichtsfunktion ist, folgt $\ell(w) \geq \ell(w') \geq d[v]$ im Widerspruch zur Annahme, daß $\ell(w) < d[v]$.

Nach Induktionsvoraussetzung, Teil (ii) existiert ein Grenzweg von s nach v der Länge $d[v]$, der somit gleichzeitig der kürzeste Weg von s nach v ist. Dies zeigt (i). Es verbleibt zu zeigen, daß (ii) gilt. Sei dazu $u \in V \setminus S_{k+1}$. Wir bezeichnen mit $d[u]$ und $d[v]$ die Werte d bei Entfernen von v aus dem Heap, aber vor den DECREASE-KEY Operationen.

Für die Länge $\ell(w)$ eines kürzesten S_{k+1} -Grenzweges w von s nach u gilt:

$$\ell(w) = \min\left(\min\{\ell(w') : w' \text{ ist } S_k\text{-Grenzweg von } s \text{ nach } u\}, \right. \tag{A.7}$$

$$\left. \delta(s, v) + \min\{c(r) : \alpha(r) = v \wedge \omega(r) = u\}\right) \tag{A.8}$$

Nach Induktionsvoraussetzung ist der Term in (A.8) gleich $d[u]$. Der zweite Term in (A.8) entspricht nach Induktionsvoraussetzung

$$d[v] + \min\{c(r) : \alpha(r) = v \wedge \omega(r) = v\}$$

Somit gilt:

$$\ell(w) = \min(d[u], d[v] + \min\{c(r) : \alpha(r) = v \wedge \omega(r) = v\}) \tag{A.9}$$

Nach den DECREASE-KEY Operationen, die auf das Entfernen von v aus Q folgen, wird der neue Wert $d[u]$ aber genau auf den Wert aus (A.9) gesetzt. Dies zeigt (ii). \square

- (b) Im Verlauf des Algorithmus werden genau $n := |V|$ Elemente in den Heap eingefügt, die alle wieder entfernt werden. Nur die Betrachtung eines Pfeiles gibt Anlass zu einer Schlüsselverminderung und jeder Pfeil wird höchstens einmal betrachtet. Es ergibt sich also eine Folge von $|V|$ INSERT Operationen, n EXTRACT-MIN Operationen, sowie höchstens $m := |R|$ DECREASE-KEY Operationen. Unter Verwendung eines Fibonacci-Heaps folgt mit Satz 5.19, dass man den Algorithmus so implementieren kann, dass er in Zeit $\mathcal{O}(n \log n + m)$ läuft.

Lösung zu Aufgabe 7.3

Der Beweis aus Aufgabe 7.1 bleibt auch gültig, falls statt nichtnegativer Gewichtsfunktionen beliebige zugelassen werden, solange vermieden wird, daß Kreise negativer Länge zu Distanzen $-\infty$ führen.

Lösung zu Aufgabe 7.4

Wir nutzen die Monotonie der Logarithmus-Funktion, um die Suche nach einem zuverlässigsten Weg zurückzuführen auf die Suche nach einem kürzesten Weg. Ein Weg (r_1, \dots, r_k) hat genau dann maximale Zuverlässigkeit

$$\prod_{i=1}^k (1 - f(r_i)),$$

wenn der Wert

$$\log \prod_{i=1}^k (1 - f(r_i)) = \sum_{i=1}^k \log(1 - f(r_i))$$

maximal, also

$$\sum_{i=1}^k -\log(1 - f(r_i))$$

minimal ist.

Lösung zu Aufgabe 7.5

- (a) Der Algorithmus ist identisch zum Algorithmus von Dijkstra (Algorithmus 7.3) für den Spezialfall, daß $c(v) = 1$ für alle $v \in V$ gilt. Um dies zu beweisen zeigen wir, daß das erste Element der Liste L jeweils ein Knoten mit minimalem $d[v]$ ist, dessen Nachfolger noch nicht betrachtet wurden. Das Entfernen des ersten Listenelements entspricht dann dem EXTRACT-MIN-Schritt im Dijkstra-Algorithmus.

Behauptung: Zu jedem Zeitpunkt gilt $L = \{v_1, \dots, v_k, \dots, v_l\}$ wobei $d[v_i] + 1 = d[v_j]$ für alle $0 < i \leq k < j \leq l$.

Für die Initialisierung $L = \{s\}$ ist die Behauptung trivial.

Bei jedem Durchlauf der while-Schleife bleibt diese Eigenschaft von L erhalten. Sei zu Beginn $L = \{u = v_1, \dots, v_k, \dots, v_l\}$, so gilt

nach Entfernen von u und eventuellem Hinzufügen von Nachfolgerknoten v_{l+1}, \dots, v_m $L = \{v_2, \dots, v_m\}$ mit $d[v_i] + 1 = d[v_j]$ für $1 < i \leq k < j \leq m$.

Hieraus folgt nun, daß die Knoten monoton wachsende Distanzwerte $d[\cdot]$ zugewiesen bekommen. Damit ist auch ausgeschlossen, daß ein Knoten ein zweites mal betrachtet wird, denn der zugewiesene Wert kann sich nicht verbessern und der Test in Zeile 6 wird nicht mehr erfüllt.

Zu betrachten ist noch der Fall der Ecken v mit $\delta(s, v) = \infty$. Diese werden vom Algorithmus nach der Initialisierung nicht mehr betrachtet. Dies ist aber kein wesentlicher Unterschied zum Dijkstra-Algorithmus, da auch dieser ihren Distanzwert nicht ändert.

Die übrige Behauptung folgt nun aus Lemma 7.8.

- (b) Wie in Teil (a) bewiesen, wird jede Ecke und damit auch jeder Pfeil höchstens einmal betrachtet.

Da sich die verwendeten Listenoperationen in konstanter Zeit ausführen lassen (z.B. Implementierung als Ringpuffer) ergibt sich eine Laufzeit von $O(|V| + |R|)$.

Lösung zu Aufgabe 7.6

1. Fall: $\delta(s, v) = -\infty$. Dann ist die Ungleichung offenbar immer erfüllt.

2. Fall: $\delta(s, v) = +\infty$. Dann ist $v \notin E_G(s)$. Sei $r = (u, v) \in R$. Wäre $u \in E_G(s)$, dann über r auch $v \in E_G(s)$, ein Widerspruch. Also ist $u \notin E_G(s)$, d. h. $\delta(s, u) = +\infty$, und die Ungleichung gilt wie behauptet.

Kapitel 8

Lösung zu Aufgabe 8.1

Sei $\beta \not\equiv 0$ die aktuelle Strömung in der i -ten Iteration. Ermittle einen Knoten $v \in V$ mit $\beta^+(v) > 0$. Durchlaufe einen von v ausgehenden Pfeil r mit $\beta(r) > 0$. Da β Strömung ist, gilt $\beta^+(\omega(r)) > 0$. Wiederhole das Durchlaufen, bis nach höchstens $|V|$ Schritten ein Kreis C gefunden wurde. Ermittle $\varepsilon := \min_{r \in C} \beta(r)$. Setze $\beta_i(r) := \varepsilon$ für die Pfeile $r \in C$ und gleich null sonst, ferner $\beta := \beta - \beta_i$. Iteriere, bis $\beta \equiv 0$.

Da in jeder Iteration die Strömung auf mindestens einem Pfeil null wird, bricht das Verfahren nach höchstens $k = |R|$ Iterationen ab. Offenbar gilt $\beta = \sum_{i=1}^k \beta_i$.

Lösung zu Aufgabe 8.2

- a. Es gilt:

$$\gamma(r) = \beta(r) + \gamma_\beta(r^+) - \gamma_\beta(r^-).$$

- b. Für $\gamma(r) > \beta(r)$ ist

$$\gamma_\beta(r^+) = \gamma(r) - \beta(r) > 0, \quad \gamma_\beta(r^-) = 0.$$

Damit sind die Kosten im Inkrementgraphen

$$k_{\beta}(r^+) \cdot \gamma_{\beta}(r^+) = k(r) \cdot (\gamma(r) - \beta(r)) = k(r)\gamma(r) - k(r)\beta(r).$$

Für $\gamma(r) < \beta(r)$ ist

$$\gamma_{\beta}(r^+) = 0 \quad \gamma_{\beta}(r^-) = \beta(r) - \gamma(r) > 0.$$

Damit sind die Kosten im Inkrementgraphen

$$k_{\beta}(r^-) \cdot \gamma_{\beta}(r^-) = -k(r) \cdot (\beta(r) - \gamma(r)) = k(r)\gamma(r) - k(r)\beta(r).$$

- c. „ \Rightarrow “: Widerspruchsannahme: Sei $C = (r_1^{\delta_1}, \dots, r_p^{\delta_p})$ ein Kreis in G_{β} mit $\delta_i \in \{+, -\}$ und $\sum_i k(r_i^{\delta_i}) < 0$. Wähle $\varepsilon := \min_i c(r_i^{\delta_i})$ als die kleinste Kapazität im Kreis und γ_{β} als die Kreisströmung vom Wert ε entlang C . Die Kosten sind dann $k_{\beta}(\gamma_{\beta}) = \varepsilon \cdot \sum_i k(r_i^{\delta_i}) < 0$.

Sei γ die induzierte Strömung im Ausgangsgraphen. Da γ_{β} zulässig in G_{β} ist, ist nach Teilaufgabe a auch γ zulässig in G . Weiter ist γ Strömung zum Wert F , weil der ausgezeichnete Pfeil nicht im Kreis enthalten ist.

Für die Kosten von γ gilt nach Teilaufgabe a:

$$k(\gamma) = k(\beta) + k_{\beta}(\gamma_{\beta}) < k(\beta),$$

also war β nicht kostenminimal.

„ \Leftarrow “: Sei β Strömung zum Flußwert F und G_{β} enthalte keinen Kreis negativer Kosten. Sei γ eine zweite zulässige Strömung zum Flußwert F . Dann ist γ_{β} zulässige Strömung in G_{β} . Diese läßt sich nach Aufgabe 8.1 in eine Summe von Kreisströmungen zerlegen, deren Kosten nach Voraussetzung nichtnegativ sind. Damit ist

$$k(\gamma) = k(\beta) + k(\gamma_{\beta}) \geq k(\beta),$$

also ist β kostenminimal.

Lösung zu Aufgabe 8.3

a. Es gilt

$$\begin{aligned}
2 \cdot \sum_{r \in R} \beta(r) \Delta(r) &= \sum_{v \in V} \sum_{\substack{r \in R \\ \alpha(r)=v}} \beta(r) \Delta(r) + \sum_{v \in V} \sum_{\substack{r \in R \\ \omega(r)=v}} \beta(r) \Delta(r) = \\
&= \sum_{v \in V} \sum_{\substack{r \in R \\ \alpha(r)=v}} \beta(r) \pi(\omega(r)) - \sum_{v \in V} \sum_{\substack{r \in R \\ \alpha(r)=v}} \beta(r) \pi(\alpha(r)) + \\
&\quad + \sum_{v \in V} \sum_{\substack{r \in R \\ \omega(r)=v}} \beta(r) \pi(\omega(r)) - \sum_{v \in V} \sum_{\substack{r \in R \\ \omega(r)=v}} \beta(r) \pi(\alpha(r)) = \\
&= \sum_{v \in V} \sum_{\substack{r \in R \\ \alpha(r)=v}} \beta(r) \pi(\omega(r)) - \sum_{v \in V} \pi(v) \beta^+(v) + \\
&\quad + \sum_{v \in V} \pi(v) \beta^-(v) - \sum_{v \in V} \sum_{\substack{r \in R \\ \omega(r)=v}} \beta(r) \pi(\alpha(r)) = \\
&= \sum_{v \in V} \sum_{\substack{r \in R \\ \alpha(r)=v}} \beta(r) \pi(\omega(r)) - \sum_{v \in V} \sum_{\substack{r \in R \\ \alpha(r)=v}} \beta(r) \pi(\alpha(r)) = \\
&= \sum_{r \in R} \beta(r) \Delta(r),
\end{aligned}$$

$$\text{also } \sum_{r \in R} \beta(r) \Delta(r) = 0.$$

b. „(i) \Rightarrow (ii)“: Induktive Konstruktion. Wähle im i -ten Schritt eine Ecke v im aktuellen Graphen mit $d^-(v) = 0$. Diese existiert wegen der Kreisfreiheit von G .

Setze $\pi(v) := i$, und entferne v mit allen inzidenten Pfeilen. Da die Zielecke $\omega(r)$ jedes hier entfernten Pfeiles r später gelabelt wird, folgt $\Delta(r) > 0$. Ferner ist der entstehende Graph wieder kreisfrei.

„(ii) \Rightarrow (iii)“: Nach Teilaufgabe a ist $\beta \Delta = 0$. Da $\Delta(r) > 0$ und $\beta(r) \geq 0$, ist jeder Summand nichtnegativ. Da die Summe null ist, muß jeder Summand gleich null sein. Wegen $\Delta(r) > 0$ folgt dann $\beta(r) = 0$ für alle $r \in R$.

„(iii) \Rightarrow (i)“: Sei G nicht kreisfrei, C ein Kreis, ε die Mindestkapazität auf C . Dann ist $\beta(r) := \varepsilon$ für $r \in C$ und $\beta(r) := 0$ sonst eine Strömung in G .

Kapitel 9**Kapitel 10**

Anhang B

Notationen

Schlußbemerkungen und Dank

Das Skript wurde in der Hoffnung erstellt, daß es Studentinnen oder Studenten eine Einführung in graphentheoretische Algorithmen ermöglicht. Vielleicht weckt es ja auch bei einigen die Freude an der Graphentheorie.

Das Skript wird weiterhin aktualisiert (und dabei hoffentlich verbessert) werden. Leider enthält es noch immer Tippfehler. Für Anregungen und Verbesserungsvorschläge (auch inhaltlicher Natur) bin ich jederzeit dankbar. Ich möchte mich an dieser Stelle nochmals für die hilfreichen Kommentare derjenigen Studenten bedanken, die sich die Mühe gemacht haben, mich auf Fehler und Ungereimtheiten hinzuweisen: Bernd Hirschfeld, Rouven Lepenis, Michael Menth. Ich entschuldige mich für die Namen derjenigen, die ich unabsichtlich vergessen haben sollte.

Würzburg, im Sommer 1997

Sven Oliver Krumke

Schlußbemerkung

Im Rahmen der Vorlesung »Graphentheoretische Methoden und Algorithmen«, die von Herrn Prof. Noltemeier im Sommersemester 1999 an der Universität Würzburg gehalten wurde, ist dieses Skript erneut zum Einsatz gekommen. Es wurde dabei um das Kapitel über planare Graphen und um Abschnitte zu kostenminimalen Strömungen und zu spannenden Wurzelbäumen ergänzt, zudem wurden neue Übungsaufgaben aufgenommen und die Lösungen vervollständigt. Ferner konnten eine Reihe von Fehlern berichtigt werden, für deren Aufspürung insbesondere Thomas Heilmann Dank gesagt sei.

Würzburg, im Dezember 1999

Hans-Christoph Wirth

Literaturverzeichnis

- [AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Networks flows*, Prentice Hall, Englewood Cliffs, New Jersey, 1993.
- [CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to algorithms*, MIT Press, 1990.
- [GJ79] M. R. Garey and D. S. Johnson, *Computers and intractability (a guide to the theory of NP-completeness)*, W.H. Freeman and Company, New York, 1979.
- [Har72] F. Harary, *Graph theory*, Addison-Wesley Publishing Company, Inc., 1972.
- [HT74] J. Hopcroft and P. Tarjan, *Efficient planarity testing*, Journal of the ACM **21** (1974), no. 4, 549–568.
- [Jun90] D. Jungnickel, *Graphen, Netzwerke und Algorithmen*, 2 ed., BI-Wissenschaftsverlag, 1990.
- [Nol75] H. Noltemeier, *Graphentheorie: mit Algorithmen und Anwendungen*, de Gruyter Lehrbuch, 1975.
- [Pap94] C. M. Papadimitriou, *Computational complexity*, Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1994.
- [Tar83] R. E. Tarjan, *Data structures and networks algorithms*, CBMS-NSF Regional Conference Series in Applied Mathematics, vol. 44, Society for Industrial and Applied Mathematics, 1983.