Vorlesungsskript

Graphalgorithmen

Wintersemester 2018/19

Prof. Dr. Johannes Köbler
Humboldt-Universität zu Berlin
Lehrstuhl Komplexität und Kryptografie

9. Januar 2019

Inhaltsverzeichnis

1 Graphentheoretische Grundlagen

Definition 1.1. *Ein* ***(ungerichteter) Graph*** *ist ein Paar* $G = (V, E)$*, wobei*

V *- eine endliche Menge von* ***Knoten/Ecken*** *und*

E *- die Menge der* ***Kanten*** *ist.*

Hierbei gilt

$$E \subseteq \binom{V}{2} = \Big\{\{u, v\} \subseteq V \mid u \neq v\Big\}.$$

Sei $v \in V$ *ein Knoten.*

a) Die ***Nachbarschaft*** *von* v *ist* $N_G(v) = \{u \in V \mid \{u, v\} \in E\}$.

b) Der ***Grad*** *von* v *ist* $\deg_G(v) = |N_G(v)|$.

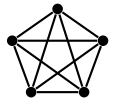
c) Der ***Minimalgrad*** *von* G *ist* $\delta(G) = \min_{v \in V} \deg_G(v)$ *und der* ***Maximalgrad*** *von* G *ist* $\Delta(G) = \max_{v \in V} \deg_G(v)$.

d) Jeder Knoten $u \in V$ *vom Grad* ≤ 1 *heißt* ***Blatt*** *und die übrigen Knoten (vom Grad* ≥ 2*) heißen* ***innere Knoten*** *von* G.

Falls G aus dem Kontext ersichtlich ist, schreiben wir auch einfach $N(v)$, $\deg(v)$, δ usw.

Beispiel 1.2.

- *Der* ***vollständige Graph*** (V, E) *auf* n *Knoten, d.h.* $|V| = n$ *und* $E = \binom{V}{2}$*, wird mit* K_n *und der* ***leere Graph*** $(V, \emptyset)$ *auf* n *Knoten wird mit* E_n *bezeichnet.*

K_1:  K_2: K_3:  K_4:  K_5: 

- *Der* ***vollständige bipartite Graph*** (A, B, E) *auf* $a + b$ *Knoten, d.h.* $A \cap B = \emptyset$*,* $|A| = a$*,* $|B| = b$ *und* $E = \{\{u, v\} \mid u \in A, v \in B\}$ *wird mit* $\boldsymbol{K_{a,b}}$ *bezeichnet.*

$K_{1,1}$: $K_{1,2}$:  $K_{2,2}$:  $K_{2,3}$:  $K_{3,3}$: 

- *Der* ***Pfad*** *mit* n *Knoten wird mit* P_n *bezeichnet.*

P_2: P_3: P_4: P_5:

- *Der* ***Kreis*** *mit* n *Knoten wird mit* C_n *bezeichnet.*

C_3:  C_4:  C_5:  C_6: 

Definition 1.3. *Sei* $G = (V, E)$ *ein Graph.*

a) Eine Knotenmenge $U \subseteq V$ *heißt* ***unabhängig*** *oder* ***stabil****, wenn es keine Kante von* G *mit beiden Endpunkten in* U *gibt, d.h. es gilt* $E \cap \binom{U}{2} = \emptyset$*. Die* ***Stabilitätszahl*** *ist*

$$\alpha(G) = \max\{|U| \mid U \text{ ist stabile Menge in } G\}.$$

b) Eine Knotenmenge $U \subseteq V$ *heißt* ***Clique****, wenn jede Kante mit beiden Endpunkten in* U *in* E *ist, d.h. es gilt* $\binom{U}{2} \subseteq E$*. Die* ***Cliquenzahl*** *ist*

$$\omega(G) = \max\{|U| \mid U \text{ ist Clique in } G\}.$$

c) Ein Graph $G' = (V', E')$ *heißt* ***Sub-/Teil-/Untergraph*** *von* G*, falls* $V' \subseteq V$ *und* $E' \subseteq E$ *ist. Im Fall* $V' = V$ *wird* G' *auch ein* ***(auf)spannender*** *Teilgraph von* G *genannt und wir schreiben für* G' *auch* $G - E''$ *(bzw.* $G = G' \cup E''$*), wobei* $E'' = E - E'$ *die Menge der aus* G *entfernten Kanten ist. Im Fall* $E'' = \{e\}$ *schreiben wir für* G' *auch einfach* $G - e$ *(bzw.* $G = G' \cup e$*).*

d) *Ein k-regulärer spannender Teilgraph von G wird auch als* ***k-Faktor*** *von G bezeichnet. Ein d-regulärer Graph G heißt* ***k-faktorisierbar****, wenn sich G in $l = d/k$ kantendisjunkte k-Faktoren $G_1, \ldots, G_l$ zerlegen lässt.*

e) *Ein Subgraph $G' = (V', E')$ heißt (****durch V'****) **induziert**, falls $E' = E \cap \binom{V'}{2}$ ist. Für G' schreiben wir dann auch $G[V']$ oder $G - V''$, wobei $V'' = V - V'$ die Menge der aus G entfernten Knoten ist. Ist $V'' = \{v\}$, so schreiben wir für G' auch einfach $G - v$ und im Fall $V' = \{v_1, \ldots, v_k\}$ auch $G[v_1, \ldots, v_k]$.*

f) *Ein* ***Weg*** *ist eine Folge von (nicht notwendig verschiedenen) Knoten $v_0, \ldots, v_\ell$ mit $\{v_i, v_{i+1}\} \in E$ für $i = 0, \ldots, \ell - 1$. Die* ***Länge*** *des Weges ist die Anzahl der durchlaufenen Kanten, also ℓ. Im Fall $\ell = 0$ heißt der Weg* ***trivial****. Ein Weg $(v_0, \ldots, v_\ell)$ heißt auch* ***v_0-v_ℓ-Weg****.*

g) *G heißt* ***zusammenhängend****, falls es für alle Paare $\{u, v\} \in \binom{V}{2}$ einen u-v-Weg gibt.*

h) *Die durch die Äquivalenzklassen $V_i \subseteq V$ der Relation*

$$Z = \{(u, v) \in V \times V \mid \text{ es gibt in } G \text{ einen } u\text{-}v\text{-Weg}\}$$

induzierten Teilgraphen $G[V_i]$ heißen ***Zusammenhangskomponenten*** *(engl.* connected components*) oder einfach* ***Komponenten*** *von G.*

i) *Ein u-v-Weg heißt* ***einfach*** *oder* ***u-v-Pfad****, falls alle durchlaufenen Knoten verschieden sind.*

j) *Ein* ***Zyklus*** *ist ein u-v-Weg mit $u = v$.*

k) *Eine Menge von Pfaden heißt* ***disjunkt****, wenn je zwei Pfade in der Menge keine gemeinsamen Knoten haben,* ***kantendisjunkt****, wenn je zwei Pfade in der Menge keine gemeinsamen Kanten haben, und* ***knotendisjunkt****, wenn je zwei Pfade in der Menge höchstens gemeinsame Endpunkte haben.*

l) *Ein* ***Kreis*** *ist ein Zyklus $(v_1 \ldots, v_\ell, v_1)$ der Länge $\ell \geq 3$, für den $v_1, \ldots, v_\ell$ paarweise verschieden sind.*

m) *Ein Graph heißt* ***kreisfrei****,* ***azyklisch*** *oder* ***Wald****, falls er keinen Kreis enthält. Ein* ***Baum*** *ist ein zusammenhängender Wald.*

Definition 1.4. *Ein* ***gerichteter Graph*** *oder* ***Digraph*** *ist ein Paar $G = (V, E)$, wobei*

V - eine endliche Menge von ***Knoten/Ecken*** *und*

E - die Menge der ***Kanten*** *ist.*

Hierbei gilt

$$E \subseteq V \times V = \Big\{(u, v) \mid u, v \in V\Big\},$$

wobei E auch Schlingen (u, u) enthalten kann. Sei $v \in V$ ein Knoten.

a) *Die* ***Nachfolgermenge*** *von v ist $N^+(v) = \{u \in V \mid (v, u) \in E\}$.*

b) *Die* ***Vorgängermenge*** *von v ist $N^-(v) = \{u \in V \mid (u, v) \in E\}$.*

c) *Die* ***Nachbarmenge*** *von v ist $N(v) = N^+(v) \cup N^-(v)$.*

d) *Der* ***Ausgangsgrad*** *von v ist $\deg^+(v) = |N^+(v)|$ und der* ***Eingangsgrad*** *von v ist $\deg^-(v) = |N^-(v)|$. Der* ***Grad*** *von v ist $\deg(v) = \deg^+(v) + \deg^-(v)$.*

e) *Ein* ***(gerichteter) v_0-v_ℓ-Weg*** *ist eine Folge von Knoten $v_0, \ldots, v_\ell$ mit $(v_i, v_{i+1}) \in E$ für $i = 0, \ldots, \ell - 1$.*

f) *Ein* ***(gerichteter) Zyklus*** *ist ein gerichteter u-v-Weg mit $u = v$.*

g) *Ein gerichteter Weg heißt* ***einfach*** *oder* ***(gerichteter) Pfad****, falls alle durchlaufenen Knoten verschieden sind.*

h) *Ein* ***(gerichteter) Kreis*** *in G ist ein gerichteter Zyklus $(v_1 \ldots, v_\ell, v_1)$ der Länge $\ell \geq 1$, für den $v_1, \ldots, v_\ell$ paarweise verschieden sind.*

i) *G heißt* ***kreisfrei*** *oder* ***azyklisch****, wenn es in G keinen gerichteten Kreis gibt.*

j) *G heißt* ***stark zusammenhängend****, wenn es in G für jedes Knotenpaar $u \neq v \in V$ sowohl einen u-v-Pfad als auch einen v-u-Pfad gibt.*

k) *G heißt* ***gerichteter Wald****, wenn G kreisfrei ist und jeder Knoten $v \in V$ Eingangsgrad $\deg^-(v) \leq 1$ hat.*

l) Ein Knoten $w \in V$ vom Eingangsgrad $\deg^-(w) = 0$ heißt ***Wurzel*** *von G, und ein Knoten $u \in V$ vom Ausgangsgrad $\deg^+(u) = 0$ heißt* ***Blatt*** *von G.*

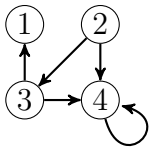
Die ***Adjazenzmatrix*** eines Graphen bzw. Digraphen $G = (V, E)$ mit (geordneter) Knotenmenge $V = \{v_1, \ldots, v_n\}$ ist die $(n \times n)$-Matrix $A = (a_{ij})$ mit den Einträgen

$$a_{ij} = \begin{cases} 1, & \{v_i, v_j\} \in E \\ 0, & \text{sonst} \end{cases} \qquad \text{bzw.} \qquad a_{ij} = \begin{cases} 1, & (v_i, v_j) \in E \\ 0, & \text{sonst.} \end{cases}$$

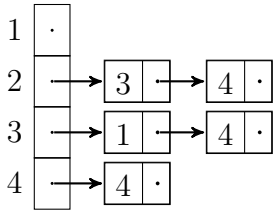
Für ungerichtete Graphen ist die Adjazenzmatrix symmetrisch mit $a_{ii} = 0$ für $i = 1, \ldots, n$.

Bei der ***Adjazenzlisten-Darstellung*** wird für jeden Knoten v_i eine Liste mit seinen Nachbarn verwaltet. Im gerichteten Fall verwaltet man entweder nur die Liste der Nachfolger oder zusätzlich eine weitere für die Vorgänger. Falls die Anzahl der Knoten statisch ist, organisiert man die Adjazenzlisten in einem Feld, d.h. das Feldelement mit Index i verweist auf die Adjazenzliste von Knoten v_i. Falls sich die Anzahl der Knoten dynamisch ändert, so werden die Adjazenzlisten typischerweise ebenfalls in einer doppelt verketteten Liste verwaltet.

Beispiel 1.5.
Betrachte den gerichteten Graphen $G = (V, E)$ mit $V = \{1, 2, 3, 4\}$ und $E = \{(2, 3), (2, 4), (3, 1), (3, 4), (4, 4)\}$. Dieser hat folgende Adjazenzmatrix- und Adjazenzlisten-Darstellung:



	1	2	3	4
1	0	0	0	0
2	0	0	1	1
3	1	0	0	1
4	0	0	0	1



◁

2 Färben von Graphen

Definition 2.1. *Sei $G = (V, E)$ ein Graph und sei $k \in \mathbb{N}$.*

a) Eine Abbildung $f\colon V \to \mathbb{N}$ heißt ***Färbung*** *von G, wenn $f(u) \neq f(v)$ für alle $\{u, v\} \in E$ gilt.*

b) G heißt ***k-färbbar****, falls eine Färbung $f\colon V \to \{1, \ldots, k\}$ existiert.*

c) Die ***chromatische Zahl*** *ist*

$$\chi(G) = \min\{k \in \mathbb{N} \mid G \text{ ist } k\text{-färbbar}\}.$$

Beispiel 2.2.

$$\chi(E_n) = 1,\ \chi(K_{n,m}) = 2,\ \chi(K_n) = n,$$

$$\chi(C_n) = \begin{cases} 2, & n \text{ gerade} \\ 3, & \text{sonst.} \end{cases}$$

Ein wichtiges Entscheidungsproblem ist, ob ein gegebener Graph k-färbbar ist. Dieses Problem ist für jedes feste $k \geq 3$ schwierig.

k-Färbbarkeit (k-Coloring):
Gegeben: Ein Graph G.
Gefragt: Ist G k-färbbar?

Satz 2.3. *k-Coloring ist für $k \geq 3$ NP-vollständig.*

Das folgende Lemma setzt die chromatische Zahl $\chi(G)$ in Beziehung zur Stabilitätszahl $\alpha(G)$.

Lemma 2.4. $n/\alpha(G) \leq \chi(G) \leq n - \alpha(G) + 1$.

Beweis. Sei G ein Graph und sei c eine $\chi(G)$-Färbung von G. Da dann die Mengen $S_i = \{u \in V \mid c(u) = i\}$, $i = 1, \ldots, \chi(G)$, stabil sind, folgt $|S_i| \leq \alpha(G)$ und somit gilt

$$n \;=\; \sum_{i=1}^{\chi(G)} |S_i| \;\leq\; \chi(G)\alpha(G).$$

Für den Beweis von $\chi(G) \leq n - \alpha(G) + 1$ sei S eine stabile Menge in G mit $|S| = \alpha(G)$. Dann ist $G - S$ k-färbbar für ein $k \leq n - |S|$. Da wir alle Knoten in S mit der Farbe $k + 1$ färben können, folgt $\chi(G) \leq k + 1 \leq n - \alpha(G) + 1$. ■

Beide Abschätzungen sind scharf, können andererseits aber auch beliebig schlecht werden.

Lemma 2.5. $\binom{\chi(G)}{2} \leq m$ *und somit* $\chi(G) \leq \frac{1}{2} + \sqrt{2m + \frac{1}{4}}$.

Beweis. Zwischen je zwei Farbklassen einer optimalen Färbung muss es mindestens eine Kante geben. ■

Die chromatische Zahl steht auch in Beziehung zur Cliquenzahl $\omega(G)$ und zum Maximalgrad $\Delta(G)$:

Lemma 2.6. $\omega(G) \leq \chi(G) \leq \Delta(G) + 1$.

Beweis. Die erste Ungleichung folgt daraus, dass die Knoten einer maximal großen Clique unterschiedliche Farben erhalten müssen.

Um die zweite Ungleichung zu erhalten, betrachten wir folgenden Färbungsalgorithmus:

Algorithmus `greedy-color`

```
1  input ein Graph G = (V, E) mit V = {v_1, ..., v_n}
2  c(v_1) := 1
3  for i := 2 to n do
4     F_i := {c(v_j) | j < i, v_j ∈ N(v_i)}
5     c(v_i) := min{k ≥ 1 | k ∉ F_i}
```


Da für die Farbe $c(v_i)$ von v_i nur $|F_i| \leq \Delta(G)$ Farben verboten sind, gilt $c(v_i) \leq \Delta(G) + 1$. ■

2.1 Färben von planaren Graphen

Ein Graph G heißt ***planar***, wenn er so in die Ebene einbettbar ist, dass sich zwei verschiedene Kanten höchstens in ihren Endpunkten berühren. Dabei werden die Knoten von G als Punkte und die Kanten von G als Verbindungslinien (genauer: Jordankurven) zwischen den zugehörigen Endpunkten dargestellt.

Bereits im 19. Jahrhundert wurde die Frage aufgeworfen, wie viele Farben höchstens benötigt werden, um eine Landkarte so zu färben, dass aneinander grenzende Länder unterschiedliche Farben erhalten. Offensichtlich lässt sich eine Landkarte in einen planaren Graphen transformieren, indem man für jedes Land einen Knoten zeichnet und benachbarte Länder durch eine Kante verbindet. Länder, die sich nur in einem Punkt berühren, gelten dabei nicht als benachbart.

Die Vermutung, dass 4 Farben ausreichen, wurde 1878 von Kempe „bewiesen“ und erst 1890 entdeckte Heawood einen Fehler in Kempes „Beweis“. Übrig blieb der *5-Farben-Satz*. Der *4-Farben-Satz* wurde erst 1976 von Appel und Haken bewiesen. Hierbei handelt es sich jedoch nicht um einen Beweis im klassischen Sinne, da zur Überprüfung der vielen auftretenden Spezialfälle Computer benötigt werden.

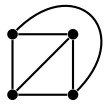
Satz 2.7 (Appel, Haken 1976)**.**
Jeder planare Graph ist 4-färbbar.

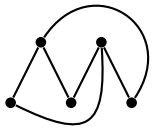
Aus dem Beweis des 4-Farben-Satzes von Appel und Haken lässt sich ein 4-Färbungsalgorithmus für planare Graphen mit einer Laufzeit von $\mathcal{O}(n^4)$ gewinnen.

In 1997 fanden Robertson, Sanders, Seymour und Thomas einen einfacheren Beweis für den 4-Farben-Satz, welcher zwar einen deut-

lich schnelleren $\mathcal{O}(n^2)$ Algorithmus liefert, aber ebenfalls nur mit Computer-Unterstützung verifizierbar ist.

Beispiel 2.8. *Wie die folgenden Einbettungen von K_4 und $K_{2,3}$ in die Ebene zeigen, sind K_4 und $K_{2,3}$ planar.*

K_4: 

$K_{2,3}$: 

◁

Zur Beantwortung der Frage, ob auch K_5 und $K_{3,3}$ planar sind, betrachten wir die ***Gebiete***, die bei der Einbettung von (zusammenhängenden) Graphen in die Ebene entstehen. Dabei gehören 2 Punkte zum selben Gebiet, falls es zwischen ihnen eine Verbindungslinie gibt, die keine Kante des eingebetten Graphen kreuzt oder berührt. Nur eines dieser Gebiete ist unbeschränkt und dieses wird als ***äußeres Gebiet*** bezeichnet. Die Anzahl der Gebiete von G bezeichnen wir mit $r(G)$ oder kurz mit r. Die begrenzenden Kanten eines Gebietes g bilden seinen ***Rand*** $rand(g)$. Ihre Anzahl bezeichnen wir mit $d(g)$, wobei Kanten $\{u, v\}$, an die g von beiden Seiten grenzt, doppelt gezählt werden.

Der ***Rand*** $rand(g)$ eines Gebiets g ist die (zirkuläre) Folge aller Kanten, die an g grenzen, wobei man jede Kante so durchläuft, dass g „in Fahrtrichtung links“ liegt bzw. jeden Knoten u, den man über eine Kante e erreicht, über die im Uhrzeigersinn nächste Kante e' wieder verlässt. Auf diese Weise erhält jede Kante auf dem Rand von g eine Richtung (oder Orientierung).

Da jede Kante zur Gesamtlänge $\sum_g d(g)$ aller Ränder den Wert 2 beiträgt (sie wird genau einmal in jeder Richtung durchlaufen), folgt

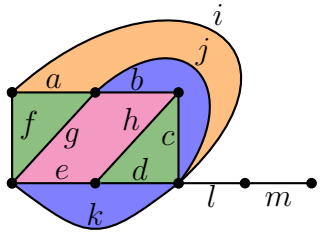
$$\sum_g d(g) = 2m(G).$$

Wir nennen das Tripel $G' = (V, E, R)$ eine ***ebene Realisierung*** des Graphen $G = (V, E)$, falls es eine Einbettung von G in die Ebene

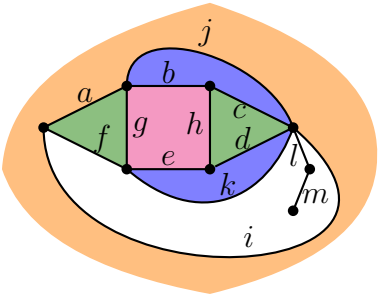
gibt, deren Gebiete die Ränder in R haben. In diesem Fall nennen wir $G' = (V, E, R)$ auch einen ***ebenen Graphen***. Ist G nicht zusammenhängend, so betten wir die Komponenten von G in die Ebene ein und fassen alle Ränder, die bei diesen Einbettungen entstehen, zu einer Randmenge R zusammen.

Führen zwei Einbettungen von G in die Ebene auf dieselbe Randmenge R, so werden sie als ***äquivalent*** angesehen. Eine andere Möglichkeit, Einbettungen bis auf Äquivalenz kombinatorisch zu beschreiben, besteht darin, für jeden Knoten u die (zirkuläre) Ordnung π_u aller mit u inzidenten Kanten anzugeben. Man nennt $\pi = \{\pi_u \mid u \in V\}$ ein ***Rotationssystem*** für G, falls es eine entsprechende Einbettung gibt. Rotationssysteme haben den Vorteil, dass sie bei Verwendung der Adjazenzlistendarstellung ohne zusätzlichen Platzaufwand gespeichert werden können, indem man die zu u adjazenten Knoten gemäß π_u anordnet.

Beispiel 2.9. *Die beiden nebenstehenden Einbettungen eines Graphen $G = (V, E)$ in die Ebene haben jeweils 7 Gebiete und führen beide auf den ebenen Graphen $G' = (V, E, R)$ mit den 7 Rändern*



$$R = \{(a, f, g), (a, j, i), (b, g, e, h), (b, c, j), (c, h, d), (d, e, k), (f, i, l, m, m, l, k)\}.$$



Das zugehörige Rotationssystem ist

$$\pi = \{(a, f, i), (a, j, b, g), (b, c, h), (e, k, f, g), (d, e, h), (c, j, i, l, k, d), (l, m), (m)\}.$$

Man beachte, dass sowohl in R als auch in π jede Kante genau zweimal vorkommt. Anstelle von (zirkulären) Kantenfolgen kann man die Elemente von R und π natürlich auch durch entsprechende Knotenfolgen beschreiben. ◁

Satz 2.10 (Polyederformel von Euler, 1750)**.**
Für einen zusammenhängenden ebenen Graphen $G = (V, E, R)$ gilt

$$n(G) - m(G) + r(G) = 2. \qquad (*)$$

Beweis. Wir führen den Beweis durch Induktion über die Kantenzahl $m(G) = m$.

$m = 0$: Da G zusammenhängend ist, muss dann $n = 1$ sein.
Somit ist auch $r = 1$, also $(*)$ erfüllt.

$m - 1 \rightsquigarrow m$: Sei G ein zusammenhängender ebener Graph mit m Kanten.

Ist G ein Baum, so entfernen wir ein Blatt und erhalten einen zusammenhängenden ebenen Graphen G' mit $n' = n - 1$ Knoten, $m' = m - 1$ Kanten und $r' = r$ Gebieten. Nach IV folgt $n - m + r = (n - 1) - (m - 1) + r = n' - m' + r' = 2$.

Falls G kein Baum ist, entfernen wir eine Kante auf einem Kreis in G und erhalten einen zusammenhängenden ebenen Graphen G' mit $n' = n$ Knoten, $m' = m - 1$ Kanten und $r' = r - 1$ Gebieten. Nach IV folgt $n - m + r = n - (m - 1) + (r - 1) = n' - m' + r' = 2$. ■

Korollar 2.11. *Sei $G = (V, E)$ ein planarer Graph mit $n \geq 3$ Knoten. Dann ist $m \leq 3n - 6$. Falls G dreiecksfrei ist, gilt sogar $m \leq 2n - 4$.*

Beweis. O.B.d.A. sei G zusammenhängend. Wir betrachten eine beliebige planare Einbettung von G. Da $n \geq 3$ ist, ist jedes Gebiet g von $d(g) \geq 3$ Kanten umgeben. Daher ist $2m = i = \sum_g d(g) \geq 3r$ bzw. $r \leq 2m/3$. Eulers Formel liefert

$$m = n + r - 2 \leq n + 2m/3 - 2,$$

was $(1 - 2/3)m \leq n - 2$ und somit $m \leq 3n - 6$ impliziert.
Wenn G dreiecksfrei ist, ist jedes Gebiet von $d(g) \geq 4$ Kanten umgeben. Daher ist $2m = i = \sum_g d(g) \geq 4r$ bzw. $r \leq m/2$. Eulers Formel liefert daher $m = n + r - 2 \leq n + m/2 - 2$, was $m/2 \leq n - 2$ und somit $m \leq 2n - 4$ impliziert. ■

Korollar 2.12. *Die Graphen K_5 und $K_{3,3}$ sind nicht planar.*

Beweis. Wegen $n(K_5) = 5$, also $3n(K_5) - 6 = 9$, und wegen $m(K_5) = \binom{5}{2} = 10$ gilt $m(K_5) \not\leq 3n(K_5) - 6$.
Wegen $n(K_{3,3}) = 6$, also $2n(K_{3,3}) - 4 = 8$, und wegen $m(K_{3,3}) = 3 \cdot 3 = 9$ gilt $m(K_{3,3}) \not\leq 2n(K_{3,3}) - 4$. ■

Als weitere interessante Folgerung aus der Polyederformel können wir zeigen, dass jeder planare Graph einen Knoten v vom Grad $\deg(v) \leq 5$ hat.

Korollar 2.13. *Jeder planare Graph hat einen Minimalgrad $\delta \leq 5$.*

Beweis. Für $n \leq 6$ ist die Behauptung klar. Für $n > 6$ impliziert die Annahme $\delta \geq 6$ die Ungleichung

$$m = \tfrac{1}{2} \textstyle\sum_{u \in V} \deg(u) \geq \tfrac{1}{2} \sum_{u \in V} 6 = 3n,$$

was im Widerspruch zu $m \leq 3n - 6$ steht. ■

Definition 2.14. *Seien $G = (V, E)$ und H Graphen und seien $u, v \in V$.*

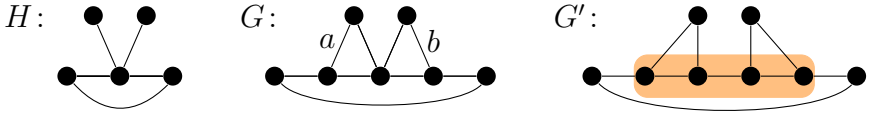
- *Durch **Fusion** von u und v entsteht aus G der Graph $G_{uv} = (V - \{v\}, E')$ mit*

$$E' = \{e \in E \mid v \notin e\} \cup \{\{u, v'\} \mid \{v, v'\} \in E - \{u, v\}\}.$$

*Ist $e = \{u, v\}$ eine Kante von G (also $e \in E$), so sagen wir auch, G_{uv} entsteht aus G durch **Kontraktion** der Kante e. Hat zudem v den Grad 2 mit $N_G(v) = \{u, w\}$, so sagen wir auch, G_{uv} entsteht aus G durch **Überbrückung** des Knotens v bzw. G aus G_{uv} durch **Unterteilung** der Kante $\{u, w\}$.*

- *G heißt zu H **kontrahierbar**, falls H aus einer isomorphen Kopie von G durch wiederholte Kontraktionen gewonnen werden kann.*
- *G heißt **Unterteilung** von H, falls G aus einer isomorphen Kopie von H durch wiederholte Unterteilungen gewonnen werden kann.*
- *H heißt **Minor** von G, wenn ein Teilgraph von G zu H kontrahierbar ist, und **topologischer Minor**, wenn ein Teilgraph von G eine Unterteilung von H ist.*
- *G heißt **H-frei**, falls H kein Minor von G ist. Für eine Menge $\mathcal{H}$ von Graphen heißt G **$\mathcal{H}$-frei**, falls G für alle $H \in \mathcal{H}$ H-frei ist.*

Beispiel 2.15. *Betrachte folgende Graphen:*



G ist keine Unterteilung von H, da G Knoten vom Grad 3 hat, aber H nicht. Entfernen wir jedoch die beiden Kanten a und b aus G, so ist der resultierende Teilgraph eine Unterteilung von H, d.h. H ist ein topologischer Minor von G. H ist aber kein topologischer Minor von G', da H einen Knoten vom Grad 4 hat und G' nur Knoten vom Grad ≤ 3. Da durch Kontraktion der drei umrandeten Kanten ein zu H isomorpher Graph entsteht, ist H aber ein Minor von G'. ◁

Es ist klar, dass die Klasse $\mathcal{K}$ der planaren Graphen zwar unter Unterteilung und (topologischer) Minorenbildung abgeschlossen ist (d.h. wenn $G \in \mathcal{K}$ und H ein Minor oder eine Unterteilung von G ist, dann folgt $H \in \mathcal{K}$), aber nicht unter Fusion.

Nach Definition lässt sich jeder (topologische) Minor H von G aus einem zu G isomorphen Graphen durch wiederholte Anwendung folgender Operationen gewinnen:

- Entfernen einer Kante oder eines Knotens,
- Kontraktion einer Kante (bzw. Überbrückung eines Knotens).

Da die Kontraktionen (bzw. Überbrückungen) o.B.d.A. auch zuletzt ausgeführt werden können, gilt hiervon auch die Umkehrung. Zudem ist leicht zu sehen, dass G und H genau dann (topologische) Minoren voneinander sind, wenn sie isomorph sind.

Satz 2.16 (Kempe 1878, Heawood 1890)**.**
Jeder planare Graph ist 5-färbbar.

Beweis. Wir beweisen den Satz durch Induktion über n.

$n = 1$: Klar.

$n - 1 \rightsquigarrow n$: Da G planar ist, existiert ein Knoten u mit $\deg(u) \leq 5$. Im Fall $\deg(u) \leq 4$ entfernen wir u aus G. Andernfalls hat u zwei Nachbarn v und w, die nicht durch eine Kante verbunden sind (andernfalls wäre K_5 ein Teilgraph von G). In diesem Fall entfernen wir alle mit u inzidenten Kanten außer $\{u, v\}$ und $\{u, w\}$ und kontrahieren diese beiden Kanten zum Knoten v.

In beiden Fällen ist der resultierende Graph G' ein Minor von G und daher planar. Da G' zudem höchstens $n - 1$ Knoten hat, existiert nach IV eine 5-Färbung c' für G'. Da wir im 2. Fall dem Knoten w die Farbe $c'(v)$ geben können, haben die Nachbarn von u höchstens 4 verschiedene Farben und wir können G 5-färben. ■

Kuratowski konnte 1930 beweisen, dass jeder nichtplanare Graph G den $K_{3,3}$ oder den K_5 als topologischen Minor enthält. Für den Beweis benötigen wir noch folgende Notationen.

Definition 2.17. *Sei $G = (V, E)$ ein Graph.*

- *Eine Menge $S \subseteq V$ heißt **Separator** in G, wenn es zwei Knoten $u, v \in V \setminus S$ gibt, zwischen denen in $G - S$ kein u-v-Weg existiert. Ist $|S| = k$, so nennen wir S auch einen **k-Separator** zwischen u und v oder auch einen **u-v-Separator** der Größe k. Ein 1-Separator wird auch **Artikulation** oder **Schnittknoten** von G genannt.*
- *Ein Graph G heißt **k-zusammenhängend**, $0 \leq k \leq n - 1$, falls G keinen $(k-1)$-Separator hat. Die größte Zahl k, für die G k-*

zusammenhängend ist, heißt ***Zusammenhangszahl*** *von G und wird mit $\kappa(G)$ bezeichnet.*

Ein Graph G mit $n \geq 2$ Knoten ist also genau dann zusammenhängend, wenn $\kappa(G) \geq 1$ ist.

Lemma 2.18. *Ist ein Graph $G = (V, E)$ nicht planar, so hat er einen*

- *2-zusammenhängenden Untergraphen $U = (V', E')$ und einen*
- *3-zusammenhängenden topologischen Minor $M = (V'', E'')$,*

die ***minimal nicht planar*** *sind, d.h. U und M sind nicht planar und für alle $e' \in E'$ und $e'' \in E''$ sind die Graphen $U - e'$ und $M - e''$ planar.*

Beweis. Wir entfernen zuerst solange Kanten und Knoten aus G, bis wir aus dem verbliebenen Teilgraphen $U = (V', E')$ keine weiteren Kanten oder Knoten entfernen können, ohne dass U planar wird.

U ist zusammenhängend, da andernfalls mindestens eine Komponente von U nicht planar ist und wir alle übrigen Komponenten entfernen könnten, ohne dass U planar wird.

U ist sogar 2-zusammenhängend, da U sonst einen Schnittknoten s enthält und $U - s$ in $k \geq 2$ Komponenten $U[V_1], \ldots, U[V_k]$ zerfällt. Dann ist aber mindestens ein Teilgraph $T_i = U[V_i \cup \{s\}]$ nicht planar und wir können alle Knoten außerhalb von T_i entfernen, ohne dass U planar wird.

Um einen topologischen Minor M von G mit den behaupteten Eigenschaften zu erhalten, konstruieren wir zu U einen topologischen Minor U', der minimal nicht planar ist und zudem 3-zusammenhängend ist oder weniger Knoten als U hat. Indem wir diese Konstruktion wiederholen, erhalten wir schließlich M.

Falls U 3-zusammenhängend ist, ist $U' = U$. Andernfalls gibt es in U einen 2-Separator $S = \{u, v\}$, d.h. $U - S$ zerfällt in $k \geq 2$ Komponenten $U[V_1], \ldots, U[V_k]$. Betrachte die (2-zusammenhängenden) Graphen $G_i = U[V_i \cup \{u, v\}] \cup \{u, v\}$. Dann ist mindestens ein G_i nicht planar (z.B. G_1), da sonst auch U planar wäre. Da $k \geq 2$ ist, erhalten wir einen zu G_1 isomorphen Graphen U' als topologischen Minor von $H = U[V_1 \cup V_2 \cup \{u, v\}]$ (und damit von U), indem wir in $U[V_2 \cup \{u, v\}]$ einen beliebigen u-v-Pfad P wählen und aus H alle Knoten und Kanten entfernen, die nicht auf P liegen und danach P überbrücken. Dann hat U' weniger Knoten als U und ist wie U minimal nicht planar. ■

Definition 2.19. *Sei G ein Graph und sei K ein Kreis in G. Ein Teilgraph B von G heißt* ***Brücke*** *von K in G, falls*

- *B nur aus einer Kante besteht, die zwei Knoten von K verbindet, aber nicht auf K liegt (solche Brücken werden auch als* ***Sehnen*** *von K bezeichnet), oder*
- *$B - K$ eine Komponente von $G - K$ ist und B aus $B - K$ durch Hinzufügen aller Kanten zwischen $B - K$ und K (und der zugehörigen Endpunkte auf K) entsteht.*

Die Knoten von B, die auf K liegen, heißen ***Kontaktpunkte*** *von B. Zwei Brücken B und B' von K heißen* ***inkompatibel****, falls*

- *B Kontaktpunkte u, v und B' Kontaktpunkte u', v' hat, so dass diese vier Punkte in der Reihenfolge u, u', v, v' auf K liegen, oder*
- *B und B' mindestens 3 gemeinsame Kontaktpunkte haben.*

Es ist leicht zu sehen, dass in einem planaren Graphen kein Kreis mehr als zwei inkompatible Brücken haben kann.

Satz 2.20 (Kuratowski 1930)**.**
Für einen Graphen G sind folgende Aussagen äquivalent:

(i) G ist planar.

(ii) G enthält weder den $K_{3,3}$ noch den K_5 als topologischen Minor.

Beweis. Die Implikation von *i*) nach *ii*) folgt aus der Abgeschlossenheit der planaren Graphen unter (topologischer) Minorenbildung.

Die Implikation von *ii)* nach *i)* zeigen wir durch Kontraposition. Sei also $G = (V, E)$ nicht planar. Dann hat G nach Lemma 2.18 einen 3-zusammenhängenden nicht planaren topologischen Minor $M = (V', E')$, so dass $M - e'$ für jede Kante $e' \in E'$ planar ist. Wir entfernen eine beliebige Kante $e_0 = \{a_0, b_0\}$ aus M. Dann ist $M - e_0$ planar. Da $M - e_0$ 2-zusammenhängend ist, gibt es in $M - e_0$ einen Kreis K durch die beiden Knoten a_0 und b_0 (siehe Übungen). Wir wählen K zusammen mit einer ebenen Realisierung H' von $M - e_0$ so, dass K möglichst viele Gebiete in H' einschließt.

Für zwei Knoten a, b auf K bezeichnen wir mit $K[a, b]$ die Menge aller Knoten, die auf dem Bogen von a nach b (im Uhrzeigersinn) auf K liegen. Zudem sei $K[a, b) = K[a, b] \setminus \{b\}$. Die Mengen $K(a, b)$ und $K(a, b]$ sind analog definiert.

Die Kanten jeder Brücke B von K in $M - e_0$ verlaufen in H' entweder alle innerhalb oder alle außerhalb von K. Im ersten Fall nennen wir B eine ***innere Brücke*** und im zweiten eine ***äußere Brücke***.

Es ist klar, dass K in H' mindestens eine innere und mindestens eine äußere Brücke haben muss. Zudem muss jede äußere Brücke B aus einer Kante $\{u, v\}$ bestehen, die zwei Knoten $u \in K(a_0, b_0)$ und $v \in K(b_0, a_0)$ verbindet. Andernfalls hätte B nämlich mindestens 2 Kontaktpunkte auf $K[a_0, b_0]$ oder auf $K[b_0, a_0]$. Daher könnte K zu einem Kreis K' erweitert werden, der in H' mehr Gebiete einschließt (bzw. ausschließt) als K, was der Wahl von K und H' widerspricht.

K hat in M außer den Brücken in $M - e_0$ noch zusätzlich die Brücke e_0. Wir wählen nun eine innere Brücke B, die sowohl zu e_0 als auch zu mindestens einer äußeren Brücke $e_1 = \{a_1, b_1\}$ inkompatibel ist. Eine solche Brücke muss es geben, da wir sonst alle mit e_0 inkompatiblen inneren Brücken nach außen klappen und e_0 als innere Brücke hinzunehmen könnten, ohne die Planarität zu verletzen.

Wir benutzen K und die drei Brücken e_0, e_1 und B, um eine Unterteilung des $K_{3,3}$ oder des K_5 in M zu finden. Hierzu geben wir entweder zwei disjunkte Mengen $A_1, A_2 \subseteq V'$ mit jeweils 3 Knoten an, so dass

9 knotendisjunkte Pfade zwischen allen Knoten $a \in A_1$ und $b \in A_2$ existieren. Oder wir geben eine Menge $A \subseteq V'$ mit fünf Knoten an, so dass 10 knotendisjunkte Pfade zwischen je zwei Knoten $a, b \in A$ existieren. Da e_0 und e_1 inkompatibel sind, können wir annehmen, dass die vier Knoten a_0, a_1, b_0, b_1 in dieser Reihenfolge auf K liegen.

Fall 1: B hat einen Kontaktpunkt $k_1 \notin \{a_0, a_1, b_0, b_1\}$. Aus Symmetriegründen können wir $k_1 \in K(a_0, a_1)$ annehmen. Da B weder zu e_0 noch zu e_1 kompatibel ist, hat B weitere Kontaktpunkte $k_2 \in K(b_0, a_0)$ und $k_3 \in K(a_1, b_1)$, wobei $k_2 = k_3$ sein kann.

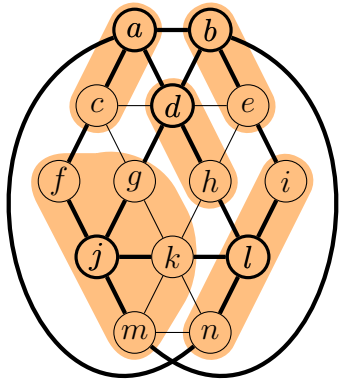
- **Fall 1a:** Ein Knoten $k_i \in \{k_2, k_3\}$ liegt auf dem Bogen $K(b_0, b_1)$. In diesem Fall existieren 9 knotendisjunkte Pfade zwischen $\{a_0, a_1, k_i\}$ und $\{b_0, b_1, k_1\}$.
- **Fall 1b:** $K(b_0, b_1) \cap \{k_2, k_3\} = \emptyset$. In diesem Fall ist $k_2 \in K[b_1, a_0)$ und $k_3 \in K(a_1, b_0]$. Dann gibt es in B einen Knoten u, von dem aus 3 knotendisjunkte Pfade zu $\{k_1, k_2, k_3\}$ existieren. Folglich gibt es 9 knotendisjunkte Pfade zwischen $\{a_0, a_1, u\}$ und $\{k_1, k_2, k_3\}$.

Fall 2: Alle Kontaktpunkte von B liegen in der Menge $\{a_0, a_1, b_0, b_1\}$. Da B inkompatibel zu e_0 und e_1 ist, müssen in diesem Fall alle vier Punkte zu B gehören. Sei P_0 ein a_0-b_0-Pfad in B und sei P_1 ein a_1-b_1-Pfad in B. Sei u der erste Knoten auf P_0, der auch auf P_1 liegt und sei v der letzte solche Knoten.

- **Fall 2a:** $u = v$. Dann gibt es in B vier knotendisjunkte Pfade von u zu $\{a_0, a_1, b_0, b_1\}$ und somit existieren in M 10 knotendisjunkte Pfade zwischen den Knoten u, a_0, a_1, b_0, b_1.
- **Fall 2b:** $u \neq v$. Durch u und v wird der Pfad P_1 in drei Teilpfade P_{xu}, P_{uv} und P_{vy} unterteilt, wobei die Indizes die Endpunkte bezeichnen und $\{x, y\} = \{a_1, b_1\}$ ist.

  Somit gibt es in B drei Pfade zwischen u und jedem Knoten in $\{a_0, v, x\}$ und zwei Pfade zwischen v und jedem Knoten in $\{b_0, y\}$, die alle 5 knotendisjunkt sind. Folglich gibt es in M 9 knotendisjunkte Pfade zwischen $\{a_0, v, x\}$ und $\{b_0, y, u\}$. ■

Beispiel 2.21. *Der nebenstehende Graph ist nicht planar, da wir den K_5 durch Kontraktion der farblich unterlegten Teilgraphen als Minor von G erhalten.*
Alternativ lässt sich der K_5 auch als ein topologischer Minor von G erhalten, indem wir die dünnen Kanten entfernen und in dem resultierenden Teilgraphen alle Knoten vom Grad 2 überbrücken. ◁



Eine unmittelbare Folgerung aus dem Satz von Kuratowski ist folgende Charakterisierung der Klasse der planaren Graphen.

Korollar 2.22 (Wagner 1937)**.** *Ein Graph ist genau dann planar, wenn er $\{K_{3,3}, K_5\}$-frei ist.*

Satz 2.23 (Satz von Robertson und Seymour, 1983-2004)**.** *Sei $\mathcal{K}$ eine Graphklasse, die unter Minorenbildung abgeschlossen ist. Dann gibt es eine endliche Menge $\mathcal{H}$ von Graphen mit*

$$\mathcal{K} = \{G \mid G \text{ ist } \mathcal{H}\text{-frei}\}.$$

Die Graphen in $\mathcal{H}$ sind bis auf Isomorphie eindeutig bestimmt und heißen ***verbotene Minoren*** *für die Klasse $\mathcal{K}$.*

Eine interessante Folgerung aus diesem Satz ist, dass jede unendliche Graphklasse zwei Graphen G und H enthält, so dass H ein Minor von G ist. Das Problem, für zwei gegebene Graphen G und H zu entscheiden, ob H ein Minor von G ist, ist zwar NP-vollständig (da sich das Hamiltonkreisproblem darauf reduzieren lässt). Für einen festen Graphen H ist das Problem dagegen effizient entscheidbar.

Satz 2.24 (Robertson und Seymour, 1995)**.** *Für jeden Graphen H gibt es einen $O(n^3)$-zeitbeschränkten Algorithmus, der für einen gegebenen Graphen G entscheidet, ob er H-frei ist.*

Korollar 2.25. *Die Zugehörigkeit zu jeder unter Minorenbildung abgeschlossenen Graphklasse $\mathcal{K}$ ist in* P *entscheidbar.*

Der Entscheidungsalgorithmus für $\mathcal{K}$ lässt sich allerdings nur angeben, wenn wir die verbotenen Minoren für $\mathcal{K}$ kennen. Leider ist der Beweis von Theorem 2.23 in dieser Hinsicht nicht konstruktiv, so dass der Nachweis, dass $\mathcal{K}$ unter Minorenbildung abgeschlossen ist, nicht automatisch zu einem effizienten Erkennungsalgorithmus für $\mathcal{K}$ führt.

2.2 Färben von chordalen Graphen

Chordalen Graphen treten in vielen Anwendungen auf, z.B. sind alle Intervall- und alle Komparabilitätsgraphen (auch transitiv orientierbare Graphen genannt) chordal. Wir werden sehen, dass sich für chordale Graphen effizient eine optimale Knotenfärbung berechnen lässt.

Definition 2.26. *Ein Graph $G = (V, E)$ heißt* ***chordal*** *oder* ***triangguliert****, wenn jeder Kreis $K = (u_1, \ldots, u_l, u_1)$ der Länge $l \geq 4$ in G mindestens eine Sehne hat.*

G ist also genau dann chordal, wenn er keinen induzierten Kreis der Länge $l \geq 4$ enthält (ein induzierter Kreis ist ein induzierter Teilgraph $G[V']$, $V' \subseteq V$, der ein Kreis ist). Dies zeigt, dass die Klasse der chordalen Graphen unter induzierter Teilgraphbildung abgeschlossen ist (aber nicht unter Teilgraphbildung). Jede solche Graphklasse $\mathcal{G}$ ist durch eine Familie von minimalen ***verbotenen induzierten Teilgraphen*** H_i charakterisiert, die bis auf Isomorphie eindeutig bestimmt sind. Die Graphen H_i gehören also nicht zu $\mathcal{G}$, aber sobald wir einen Knoten daraus entfernen, erhalten wir einen Graphen in $\mathcal{G}$. Die Klasse der chordalen Graphen hat die Familie der Kreise C_n der Länge $n \geq 4$ als verbotene induzierte Teilgraphen.

Lemma 2.27. *Für einen Graphen G sind folgende Aussagen äquivalent.*

(i) G ist chordal.

(ii) Jeder inklusionsminimale x-y-Separator S in G ist eine Clique.

(iii) Jedes Paar von nicht adjazenten Knoten x und y in G hat einen x-y-Separator S, der eine Clique ist.

Beweis. Um zu zeigen, dass die zweite Aussage aus der ersten folgt, nehmen wir an, dass G einen minimalen x-y-Separator S hat (d.h. $S \setminus \{s\}$ ist für jedes $s \in S$ kein x-y-Separator), der zwei nicht adjazente Knoten u und v enthält. Seien $G[V_1]$ und $G[V_2]$ die beiden Komponenten in $G - S$ mit $x \in V_1$ und $y \in V_2$. Da S minimal ist, haben die beiden Knoten u und v sowohl einen Nachbarn in V_1 als auch in V_2. Betrachte die beiden Teilgraphen $G_i = G[V_i \cup \{u, v\}]$ $(i = 1, 2)$ und wähle jeweils einen kürzesten u-v-Pfad P_i in G_i. Da deren Länge ≥ 2 ist, ist $K = P_1 \cup P_2$ ein Kreis der Länge ≥ 4. Aufgrund der Konstruktion ist zudem klar, dass K keine Sehnen in G hat.

Dass die zweite Aussage die dritte impliziert, ist klar, da jedes Paar von nicht adjazenten Knoten x und y einen x-y-Separator S hat, und S eine Clique sein muss, wenn wir S inklusionsminimal wählen.

Um zu zeigen, dass die erste Aussage aus der dritten folgt, nehmen wir an, dass G nicht chordal ist. Dann gibt es in G einen induzierten Kreis K der Länge ≥ 4. Seien x und y zwei beliebige nicht adjazente Knoten auf K und sei S ein x-y-Separator in G. Dann muss S mindestens zwei nicht adjazente Knoten aus K enthalten. ■

Definition 2.28. *Sei $G = (V, E)$ ein Graph und sei $k \geq 0$. Ein Knoten $u \in V$ vom Grad k heißt **k-simplizial**, wenn alle Nachbarn von u paarweise adjazent sind. Jeder k-simpliziale Knoten wird auch als **simplizial** bezeichnet.*

Zusammenhängende chordale Graphen können als eine Verallgemeinerung von Bäumen aufgefasst werden. Ein Graph G ist ein Baum, wenn er aus K_1 durch sukzessives Hinzufügen von 1-simplizialen Knoten erzeugt werden kann. Entsprechend heißt G **k-Baum**, wenn G aus K_k durch sukzessives Hinzufügen von k-simplizialen Knoten erzeugt werden kann. Wir werden sehen, dass ein zusammenhängender Graph G genau dann chordal ist, wenn er aus einem isolierten Knoten (also aus einer 1-Clique) durch sukzessives Hinzufügen von simplizialen Knoten erzeugt werden kann. Äquivalent hierzu ist, dass G durch sukzessives Entfernen von simplizialen Knoten auf einen isolierten Knoten reduziert werden kann.

Definition 2.29. *Sei $G = (V, E)$ ein Graph. Eine lineare Ordnung $(u_1, \ldots, u_n)$ auf V heißt **perfekte Eliminationsordnung (PEO)** von G, wenn u_i simplizial in $G[u_1, \ldots, u_i]$ für $i = 2, \ldots, n$ ist.*

Es ist klar dass alle Knoten eines vollständigen Graphen simplizial sind. Das folgende Lemma verallgemeinert die bekannte Tatsache, dass jeder nicht vollständige Baum T (also $T \notin \{K_1, K_2\}$) mindestens zwei nicht adjazente Blätter hat.

Lemma 2.30. *Jeder nicht vollständige chordale Graph G besitzt mindestens zwei simpliziale Knoten, die nicht adjazent sind.*

Beweis. Wir führen Induktion über n. Für $n \leq 2$ ist die Behauptung klar. Sei $G = (V, E)$ ein Graph mit $n \geq 3$ Knoten. Da G nicht vollständig ist, enthält G zwei nichtadjazente Knoten x_1 und x_2. Sei S ein minimaler x_1-x_2-Separator der Größe $k \geq 0$. Im Fall $k > 0$ ist S nach Lemma 2.27 eine Clique in G. Seien $G[V_1]$ und $G[V_2]$ die beiden Komponenten von $G - S$ mit $x_i \in V_i$. Wir zeigen die Existenz zweier simplizialer Knoten $s_i \in V_i$, $i = 1, 2$.

Betrachte die Teilgraphen $G_i = G[V_i \cup S]$. Da G_i chordal ist und weniger als n Knoten hat, ist G_i nach IV entweder eine Clique oder G_i enthält mindestens zwei nicht adjazente simpliziale Knoten y_i, z_i.

Falls G_i eine Clique ist, ist $s_i = x_i$ simplizial in G_i, und da x_i keine Nachbarn außerhalb von $V_i \cup S$ hat, ist s_i dann auch simplizial in G.

Ist G_i keine Clique, kann höchstens einer der beiden Knoten y_i, z_i zu S gehören (da S im Fall $S \neq \emptyset$ eine Clique und $\{y_i, z_i\} \notin E$ ist). O.B.d.A. sei $y_i \in V_i$. Dann hat $s_i = y_i$ keine Nachbarn außerhalb von $V_i \cup S$ und somit ist s_i auch simplizial in G. ■

Satz 2.31. *Ein Graph ist genau dann chordal, wenn er eine PEO hat.*

Beweis. Falls G chordal ist, lässt sich eine PEO gemäß Lemma 2.30 bestimmen, indem wir für $i = n, \ldots, 2$ sukzessive einen simplizialen Knoten u_i in $G - \{u_{i+1}, \ldots, u_n\}$ wählen.

Für die umgekehrte Richtung sei $(u_1, \ldots, u_n)$ eine PEO von G. Wir zeigen induktiv, dass $G_i = G[u_1, \ldots, u_i]$ chordal ist. Da u_{i+1} simplizial in G_{i+1} ist, enthält jeder Kreis K der Länge ≥ 4 in G_{i+1}, auf dem u_{i+1} liegt, eine Sehne zwischen den beiden Kreisnachbarn von u_{i+1}. Daher ist mit G_i auch G_{i+1} chordal. ■

Korollar 2.32. *Es gibt einen Polynomialzeitalgorithmus A, der für einen gegebenen Graphen G eine PEO berechnet, falls G chordal ist, und andernfalls einen induzierten Kreis der Länge ≥ 4 ausgibt.*

Beweis. A versucht wie im Beweis von Theorem 2.31 beschrieben, eine PEO zu bestimmen. Stellt sich heraus, dass $G_i = G - \{u_{i+1}, \ldots, u_n\}$ keinen simplizialen Knoten u_i hat, so ist G_i wegen Lemma 2.30 nicht chordal. Daher gibt es in G_i nach Lemma 2.27 *(iii)* ein Knotenpaar x, y, so dass kein x-y-Separator eine Clique ist. Berechnen wir für dieses Paar einen beliebigen minimalen x-y-Separator S, so ist S keine Clique und wir können wie im Beweis von $(i) \implies (ii)$ einen induzierten Kreis K der Länge ≥ 4 in G_i konstruieren. Da G_i ein induzierter Teilgraph von G ist, ist K auch ein induzierter Kreis in G. ■

Eine PEO kann verwendet werden, um einen chordalen Graphen zu färben:

Algorithmus `chordal-color`(V, E)

```
1 berechne eine PEO (u_1, ..., u_n) für G = (V, E)
2 starte greedy-color mit der Knotenfolge (u_1, ..., u_n)
```


Satz 2.33. *Für einen gegebenen chordalen Graphen $G = (V, E)$ berechnet der Algorithmus* `chordal-color` *eine k-Färbung c von G mit $k = \chi(G) = \omega(G)$.*

Beweis. Sei u_i ein beliebiger Knoten mit $c(u_i) = k$. Da $(u_1, \ldots, u_n)$ eine PEO von G ist, ist u_i simplizial in $G[u_1, \ldots, u_i]$. Somit bilden die Nachbarn u_j von u_i mit $j < i$ eine Clique und wegen $c(u_i) = k$ bilden sie zusammen mit u_i eine k-Clique. Daher gilt $\chi(G) \leq k \leq \omega(G)$, woraus wegen $\omega(G) \leq \chi(G)$ die Behauptung folgt. ■

Um `chordal-color` in Linearzeit zu implementieren, benötigen wir einen Linearzeit-Algorithmus zur Bestimmung einer PEO. Rose, Tarjan und Lueker haben 1976 einen solchen Algorithmus angegeben, der auf *lexikographischer Breitensuche* (kurz LexBFS oder LBFS, engl. *lexicographic breadth-first search*) basiert. Bevor wir diese Variante der Breitensuche vorstellen, gehen wir kurz auf verschiedene Ansätze zum Durchsuchen von Graphen ein.

Der folgende Algorithmus `GraphSearch`(V, E) startet eine Suche in einem beliebigen Knoten u und findet zunächst alle von u aus erreichbaren Knoten. Danach wird solange von einem noch nicht erreichten Knoten eine neue Suche gestartet, bis alle Knoten erreicht wurden. Die Menge der aktuellen Knoten wird dabei in einer Datenstruktur A gespeichert. Genauer enthält A alle bereits entdeckten Knoten, die noch nicht abgearbeitet sind.

Algorithmus `GraphSearch`(V, E)

```
1 R := ∅    // Menge der erreichten Knoten
2 L := ()   // Ausgabeliste
3 repeat
4   wähle u ∈ V \ R    // u wurde neu entdeckt
5   append(L, u)
6   parent(u) := ⊥
7   A := {u}    // Menge der aktuellen Knoten
8   R := R ∪ {u}
```

```
9    while A ≠ ∅ do
10     wähle u aus A
11     if ∃v ∈ N(u)\R then
12       A := A ∪ {v}     // v wurde neu entdeckt
13       R := R ∪ {v}
14       append(L, v)
15       parent(v) := u
16     else entferne u aus A     // u wurde abgearbeitet
17   until R = V
18   return(L)
```

Der Algorithmus `GraphSearch(`V, E`)` findet in jedem Durchlauf der repeat-Schleife eine neue Komponente des Eingabegraphen $G = (V, E)$. Dies bedeutet, dass alle Knoten, die zu einer Komponente gehören, konsekutiv in der Ausgabeliste $L = (u_1, \ldots, u_n)$ auftreten, wobei abgesehen vom ersten Knoten jeder Komponente jeder Knoten u_k einen Nachbarn u_i mit $i < k$ hat.

Die folgende Definition fasst diese Eigenschaften der Ausgabeliste zusammen.

Definition 2.34. *Sei $G = (V, E)$ ein Graph. Eine lineare Ordnung $(u_1, \ldots, u_n)$ auf V heißt* ***Suchordnung (SO)*** *von G, wenn für jedes Tripel $j < k < l$ gilt:*

$$u_j \in N(u_l) \setminus N(u_k) \Rightarrow \exists i < k : i \neq j \wedge u_i \in N(u_k).$$

Satz 2.35. *Für jeden Graphen $G = (V, E)$ gibt der Algorithmus* `GraphSearch(`V, E`)` *eine SO von G aus.*

Beweis. Ein Knoten u_k erhält nur dann den Wert $\texttt{parent}(u_k) = \perp$, wenn alle Knoten u_j mit $j < k$ bereits abgearbeitet sind und diese nur Nachbarn u_l mit $l < k$ hatten. Falls also ein Vorgänger u_j von u_k mit einem Nachfolger u_l von u_k verbunden ist, liefert die `parent`-Funktion einen Nachbarn $u_i = \texttt{parent}(u_k)$ von u_k mit $i < k$. Da $u_j \notin N(u_k)$ ist, gilt zusätzlich $i \neq j$. ■

Die `parent`-Funktion induziert einen gerichteten Wald $W = (V, E_{\texttt{parent}})$, dessen Kantenmenge aus allen Kanten der Form $(\texttt{parent}(v), v)$ mit $\texttt{parent}(v) \neq \perp$ besteht. Die Kanten von W werden auch als ***Baumkanten*** (kurz ***B-Kanten***) und W wird auch als ***Suchwald*** von $G = (V, E)$ bezeichnet. Für jeden Knoten $v \in V$ gibt es genau eine Wurzel w in W, von der aus v in W erreichbar ist. Der eindeutig bestimmte w-v-Pfad $P = (u_0, \ldots, u_l)$ in W mit $u_0 = w$ und $u_l = v$ lässt sich ausgehend von $u_l = v$ unter Verwendung der `parent`-Funktion mittels $u_{i-1} = \texttt{parent}(u_i)$ für $i = l, \ldots, 1$ berechnen. P wird auch als ***parent-Pfad*** von v bezeichnet. Es ist klar, dass 2 Knoten v und v' genau dann in einer Komponente von G liegen, wenn sie die gleiche Wurzel haben.

Realisieren wir die Menge der aktuellen Knoten als einen Keller S, so erhalten wir eine Suchstrategie, die als ***Tiefensuche*** (kurz *DFS*, engl. *depth first search*) bezeichnet wird. Die Benutzung eines Kellers bewirkt, dass nach der Entdeckung eines neuen Knotens v unter den Nachbarn des aktuellen Knotens u die Suche zuerst bei den Nachbarn von v fortgesetzt wird, bevor die anderen Nachbarn von u getestet werden.

Algorithmus `DFS(`V, E`)`

```
1   R := ∅     // Menge der erreichten Knoten
2   L := ()    // Ausgabeliste
3   repeat
4     wähle u ∈ V\R     // u wurde neu entdeckt
5     R := R ∪ {u}
6     append(L, u)
7     parent(u) := ⊥
8     S := (u)     // Keller der aktuellen Knoten
9     while S ≠ () do
10      u := top(S)
11      if ∃v ∈ N(u)\R then
12        push(S, v)     // v wurde neu entdeckt
```

```
13        R := R ∪ {v}
14        append(L, v)
15        parent(v) := u
16      else pop(S)    // u wurde abgearbeitet
17  until R = V
18  return(L)
```

Definition 2.36. *Sei $G = (V, E)$ ein Graph. Eine lineare Ordnung $(u_1, \ldots, u_n)$ auf V heißt* ***DFS-Ordnung (DO)*** *von G, wenn für jedes Tripel $j < k < l$ gilt:*

$$u_j \in N(u_l) \setminus N(u_k) \Rightarrow \exists i : j < i < k \wedge u_i \in N(u_k).$$

Satz 2.37. *Für jeden Graphen $G = (V, E)$ gibt der Algorithmus* `DFS(`V, E`)` *eine DO von G aus.*

Beweis. Siehe Übungen. ■

Realisieren wir die Menge der abzuarbeitenden Knoten als eine Warteschlange Q, so findet der resultierende Algorithmus `BFS(`V, E`)` einen kürzesten Weg vom Startknoten u zu allen von u aus erreichbaren Knoten. Diese Suchstrategie wird als ***Breitensuche*** (kurz *BFS*, engl. *breadth first search*) bezeichnet. Die Benutzung einer Warteschlange Q zur Speicherung der noch abzuarbeitenden Knoten bewirkt, dass alle Nachbarknoten v des aktuellen Knotens u vor den bisher noch nicht erreichten Nachbarn von v ausgegeben werden.

Algorithmus `BFS(`V, E`)`

```
1  R := ∅    // Menge der erreichten Knoten
2  L := ()   // Ausgabeliste
3  repeat
4    wähle u ∈ V\R    // u wurde neu entdeckt
5    R := R ∪ {u}
6    parent(u) := ⊥
```



```
7    Q := (u)    // Warteschlange der aktuellen Knoten
8    while Q ≠ () do
9      u := dequeue(Q)    // u wird komplett abgearbeitet
10     append(L, u)
11     for all v ∈ N(u)\R do
12       enqueue(Q, v)    // v wurde neu entdeckt
13       parent(v) := u
14     R := R ∪ N(u)
15  until R = V
16  return(L)
```

Definition 2.38. *Sei $G = (V, E)$ ein Graph. Eine lineare Ordnung $(u_1, \ldots, u_n)$ auf V heißt* ***BFS-Ordnung (BO)*** *von G, wenn für jedes Tripel $j < k < l$ gilt:*

$$u_j \in N(u_l) \setminus N(u_k) \Rightarrow \exists i < j : u_i \in N(u_k).$$

Satz 2.39. *Für jeden Graphen $G = (V, E)$ gibt der Algorithmus* `BFS(`V, E`)` *eine BO von G aus.*

Beweis. Existiert im Fall $k < l$ eine Position $j < k$ mit $u_j \in N(u_l) \setminus N(u_k)$, so muss es einen Knoten $u_i \in N(u_k)$ mit $i < j$ geben, der dafür gesorgt hat, dass der Knoten u_k vor dem Knoten u_l in die Warteschlange aufgenommen wurde. ■

BFS-Ordnungen lassen sich anschaulich anhand der Adjazenzmatrix charakterisieren. Sei $(u_1, \ldots, u_n)$ eine BO für $G = (V, E)$ und sei $A = (a_{ij})$ die Adjazenzmatrix von G mit $a_{ij} = 1 \Leftrightarrow \{u_i, u_j\} \in E$. Weiter seien $z_i = a_{i1} \ldots a_{i,i-1}$ die Präfixe der Zeilen von A, die unterhalb der Diagonale verlaufen. Sind nun die ersten j Einträge $a_{k1} \ldots a_{kj}$ einer Zeile s_k Null, so muss dies auch für jede Zeile s_l mit $l > k$ so sein, da im Fall $a_{lj} = 1$ der Knoten $u_j \in N(u_l) \setminus N(u_k)$ wäre und somit ein $i < j$ mit $a_{ki} = 1$ existieren müsste. Dies bedeutet, dass s_l mindestens so viele Nullen als Präfix hat wie z_k. Es ist aber möglich, dass z_k bspw. mit 00010... beginnt und z_l mit 00011....

Alternativ können wir Q auch als eine Warteschlange von Knotenmengen realisieren (siehe Algorithmus `BFS'`), um einen Überblick über alle möglichen Fortsetzungen der aktuellen Liste L zu einer BO zu erhalten. Die Prozedur `Dequeue`(Q) liefert ein beliebiges Element aus der ersten Menge in Q zurück und entfernt dieses aus Q.

Algorithmus `BFS'`(V, E)

```
1  R := ∅    // Menge der erreichten Knoten
2  L := ()   // Ausgabeliste
3  repeat
4    wähle u ∈ V \ R
5    R := R ∪ {u}
6    Q := ({u})    // Warteschlange von Knotenmengen
7    while Q ≠ () do
8      u := Dequeue(Q)    // u wird komplett abgearbeitet
9      append(L, u)
10     if N(u) ⊈ R then enqueue(Q, N(u) \ R)
11     R := R ∪ N(u)
12 until R = V
13 return(L)
```

Prozedur `Dequeue`(Q)

```
1  entferne u aus first(Q)
2  if first(Q) = ∅ then dequeue(Q)
3  return(u)
```

Fassen wir die Menge $V \setminus R$ der noch nicht erreichten Knoten als Nachfolgemenge der letzten Menge in Q auf, so wird von dieser Restmenge in jedem Durchlauf der `while`-Schleife von `BFS'` die Teilmenge $N(u) \setminus R$ abgetrennt und im Fall $N(u) \setminus R \neq \emptyset$ der Schlange Q hinzugefügt.

Der Unterschied von LexBFS zur normalen Breitensuche besteht darin, dass die zulässigen Ausgabefolgen gegenüber der BFS weiter eingeschränkt werden. Der Name von LexBFS rührt daher, dass die Knoten in einer Reihenfolge ausgegeben werden, die eine lexikographische Sortierung der Zeilenpräfixe z_i bewirkt, sofern man sie durch Anhängen von Einsen auf die gleiche Länge bringt. Eine solche Sortierung kann auch bei einer gewöhnlichen Breitensuche auftreten, ist bei dieser aber nicht garantiert. Bei einer Breitensuche werden die noch nicht besuchten Nachbarn des aktuellen Knotens in beliebiger Reihenfolge zur Warteschlange hinzugefügt und auch wieder in dieser Reihenfolge entfernt. Dagegen werden bei einer LexBFS die Knoten in der Warteschlange nachträglich umsortiert, falls dies notwendig ist, um eine LexBFS-Ordnung der Knoten zu erhalten (siehe Definition 2.40). Ähnlich wie bei `BFS'` wird hierzu die Menge der noch nicht abgearbeiteten Knoten in eine Folge von Knotenmengen zerlegt. Im Gegensatz zu `BFS'` kann `LexBFS` aber nicht nur die letzte Menge $V \setminus R$ splitten, sondern alle Mengen der Folge.

Algorithmus `LexBFS`(V, E, u)

```
1  L := ()    // Ausgabeliste
2  Q := (V)   // Warteschlange von Knotenmengen
3  while Q ≠ () do
4    u := Dequeue(Q)    // u wird komplett abgearbeitet
5    append(L, u)
6    Splitqueue(Q, N(u))
7  return(L)
```

Prozedur `Splitqueue`(Q, S)

```
1  for T in Q with T ∩ S ∉ {∅, T} do
2    ersetze die Teilfolge (T) in Q durch (T ∩ S, T \ S)
```

Für eine effiziente Implementierung sollte die Schlange $Q = (S_1, \ldots, S_k)$ von Knotenmengen $S_i \subseteq V$ als doppelt verkettete Liste realisiert werden und für jeden Knoten u in der Adjazenzliste ein Zeiger auf die Menge S_i, die u enthält und auf seinen Eintrag in

S_i gespeichert werden. Zudem sollte die for-Schleife in der Prozedur `Splitqueue` durch eine Schleife über die Knoten in $S = N(u)$ ersetzt werden.

Definition 2.40. *Sei $G = (V, E)$ ein Graph. Eine lineare Ordnung $(u_1, \ldots, u_n)$ auf V heißt **LexBFS-Ordnung (LBO)** von G, wenn für jedes Tripel $j < k < l$ gilt:*

$$u_j \in N(u_l) \setminus N(u_k) \Rightarrow \exists i < j : u_i \in N(u_k) \setminus N(u_l).$$

Ob eine Ordnung $(u_1, \ldots, u_n)$ eine LBO ist, lässt sich wie folgt an der gemäß $(u_1, \ldots, u_n)$ geordneten Adjazenzmatrix A ablesen: die verkürzten Zeilen $z_1, \ldots, z_n$ unter der Diagonalen müssen wie folgt sortiert sein: entweder ist z_i ein Präfix von z_{i+1} oder z_i hat an der ersten Position, wo sich die beiden Strings unterscheiden, eine Eins. Bringen wir also die verkürzten Zeilen durch Anhängen von Einsen auf dieselbe Länge, so sind sie lexikographisch sortiert. In den Übungen wird gezeigt, dass man sogar eine lexikographische Ordnung auf den *kompletten* Zeilen von A erhält, falls man die Diagonale auf 1 setzt und die Knoten in jeder Menge von Q nach absteigendem Knotengrad in G sortiert.

Satz 2.41. *Für jeden Graphen $G = (V, E)$ gibt der Algorithmus `LexBFS`(V, E) eine LBO $(u_1, \ldots, u_n)$ von G aus.*

Beweis. Sei $A = (a_{ij})$ die Adjazenzmatrix von G mit $a_{ij} = 1 \Leftrightarrow \{u_i, u_j\} \in E$. Wir zeigen, dass die Strings $z_i = a_{i1}, \ldots, a_{i,i-1}$ lexikalisch sortiert sind. Existiert nämlich im Fall $k < l$ eine Position $j < k$ mit $a_{kj} = 0$ und $a_{lj} = 1$, so muss es eine Position $i < j$ mit $a_{ki} = 1$ und $a_{li} = 0$ geben. Ansonsten wäre der Knoten u_l spätestens beim Abarbeiten von u_j in eine Menge vor dem Knoten u_k sortiert worden und könnte daher nicht nach dem Knoten u_k ausgegeben werden. ■

Satz 2.42. *Jede LBO für einen chordalen Graphen G ist eine PEO für G.*

Beweis. Sei $(u_1, \ldots, u_n)$ eine LBO für $G = (V, E)$ und sei $A = (a_{ij})$ die Adjazenzmatrix von G mit $a_{ij} = 1 \Leftrightarrow \{u_i, u_j\} \in E$, wobei wir für a_{ij} auch $A[i, j]$ schreiben. Wir zeigen, dass G nicht chordal ist, wenn u_i nicht simplizial in $G_i = G[u_1, \ldots, u_i]$ ist.

Falls u_i nicht simplizial in G_i ist, müssen Indizes $i_2 < i_1 < i =: i_0$ mit $A[i_0, i_1] = A[i_0, i_2] = 1$ und $A[i_1, i_2] = 0$ existieren. Wegen $A[i_1, i_2] = 0$ und $A[i_0, i_2] = 1$ muss es einen Index $i_3 < i_2$ geben mit $A[i_1, i_3] = 1$ und $A[i_0, i_3] = 0$, wobei wir i_3 möglichst klein wählen.

Falls nun $A[i_2, i_3] = 1$ ist, haben wir einen induzierten Kreis $G[u_{i_0}, u_{i_1}, u_{i_2}, u_{i_3}] = (u_{i_0}, u_{i_1}, u_{i_3}, u_{i_2})$ der Länge 4 in G gefunden. Andernfalls muss es wegen $A[i_2, i_3] = 0$ und $A[i_1, i_3] = 1$ einen Index $i_4 < i_3$ geben mit $A[i_2, i_4] = 1$ und $A[i_1, i_4] = 0$, wobei wir i_4 wieder möglichst klein wählen. Da spätestens für $i_k = 1$ kein Index $i_{k+1} < i_k$ existiert, also $A[i_{k-1}, i_k] = 1$ sein muss, erhalten wir eine Indexfolge $1 \leq i_k < \cdots < i_1 < i_0$ mit

(a) $A[i_0, i_1] = A[i_j, i_{j+2}] = A[i_{k-1}, i_k] = 1$ für $j = 0, \ldots, k-2$ und

(b) $A[i_0, i_3] = A[i_j, i_{j+1}] = A[i_j, i_{j+3}] = A[i_{k-2}, i_{k-1}] = 0$ für $j = 1, \ldots, k-3$ und

(c) $A[i_j, l] = A[i_{j-1}, l]$ für $j = 1, \ldots, k-3$ und $l < i_{j+2}$.

Die Eigenschaften (a) und (b) ergeben sich direkt aus der Konstruktion der Folge. Eigenschaft (c) folgt aus der minimalen Wahl der Indizes $i_3, \ldots, i_k$ und impliziert für $r = 3, \ldots, k$ die Gleichungen $A[i_0, i_r] = A[i_1, i_r] = \cdots = A[i_{r-3}, i_r]$, indem wir $j = 1, \ldots, r-3$ und $l = i_r$ setzen. Da zudem $A[i_{r-3}, i_r]$ gemäß Eigenschaft (b) für $r = 3, \ldots, k$ den Wert 0 hat, folgt für alle Paare $0 \leq j < r \leq k$ die Äquivalenz

$$A[i_j, i_r] = 1 \Leftrightarrow r = j + 2 \text{ oder } j = 0 \wedge r = 1 \text{ oder } j = k - 1 \wedge r = k.$$

Folglich ist $G[u_{i_0}, \ldots, u_{i_k}]$ ein Kreis der Länge $k + 1 \geq 4$. ■

Damit haben wir einen Linearzeitalgorithmus, der für chordale Graphen eine PEO berechnet. Da auch `greedy-color` linear zeitbe-

schränkt ist, können wir den Algorithmus `chordal-color` in Linearzeit implementieren. Diesen Algorithmus können wir leicht noch so modifizieren, dass er zusammen mit der gefundenen k-Färbung entweder eine Clique C der Größe k (als Zertifikat, dass $\chi(G) = k = \omega(G)$ ist) oder einen induzierten Kreis der Länge ≥ 4 (als Zertifikat, dass G nicht chordal ist) ausgibt.

2.3 Der Satz von Brooks

Satz 2.43 (Brooks 1941)**.** *Für einen zusammenhängenden Graphen G gilt $\chi(G) = \Delta(G) + 1$ genau dann, wenn $G = K_n$ für ein $n \geq 1$ oder $G = C_n$ für ein ungerades $n \geq 3$ ist.*

Beweis. Es ist klar, dass die Graphen $G = K_n$ für $n \geq 1$ und $G = C_n$ für ungerades $n \geq 3$ die chromatische Zahl $\Delta(G) + 1$ haben. Für $\Delta(G) \leq 2$ ist leicht zu sehen, dass dies auch die einzigen zusammenhängenden Graphen mit dieser Eigenschaft sind.

Für zusammenhängende Graphen $G \neq K_n$ mit $\Delta(G) \geq 3$ zeigen wir induktiv über $n = n(G)$, dass $\chi(G) \leq \Delta(G)$ ist. Für $n \leq 4$ (IA) ist dies klar, da wir den K_4 ausgeschlossen haben. Für den IS sei also $G \neq K_n$ ein zusammenhängender Graph mit $n \geq 5$ Knoten und sei $d := \Delta(G) \geq 3$.

Falls $\delta(G) < d$ ist, hat $G' = G - u$ eine d-Färbung c', wobei u ein Knoten vom Grad $\deg(u) < d$ ist (man beachte, dass $\delta(G') < d$ und somit G' nicht d-regulär ist, was nach IV $\chi(G) \leq d$ impliziert). Da $\deg(u) < d$ ist, lässt sich c' zu einer d-Färbung c von G erweitern.

Falls $\kappa(G) \leq 1$ ist, hat G $k \geq 2$ Blöcke $B_1, \ldots B_k$, die nicht d-regulär und somit d-färbbar sind. Dies impliziert $\chi(G) \leq d$, da wir die d-Färbungen der Blöcke ausgehend von einem beliebigen Wurzelblock des BC-Baums hin zu den Blattblöcken in eine d-Färbung für G transformieren können.

Es bleibt also der Fall, dass G d-regulär und $\kappa(G) \geq 2$ ist.

Behauptung 2.44. *In G gibt es einen Knoten u_1, der zwei Nachbarn a und b mit $\{a, b\} \notin E$ hat, so dass $G - \{a, b\}$ zusammenhängend ist.*

Da $G \neq K_n$ ist, gibt es einen Knoten x, der zwei Nachbarn $y, z \in N(x)$ mit $\{y, z\} \notin E$ hat.

- Falls $G - y$ 2-zusammenhängend ist, ist $G - \{y, z\}$ zusammenhängend und die Behauptung folgt für $u_1 = x$.
- Ist $G - y$ nicht 2-zusammenhängend, d.h. $G - y$ hat mindestens zwei Blöcke, dann hat der BC-Baum T von $G - y$ mindestens zwei Blätter. Da $\kappa(G) \geq 2$ ist, ist y in G zu mindestens einem Knoten in jedem Blatt von T benachbart, der kein Schnittknoten ist. Wählen wir für a und b zwei dieser Knoten in verschiedenen Blättern, so ist $G - \{a, b\}$ zusammenhängend und somit die Behauptung für $u_1 = y$ bewiesen.

Sei also u_1 ein Knoten, der zwei Nachbarn a und b mit $\{a, b\} \notin E$ hat, so dass $G - \{a, b\}$ zusammenhängend ist. Durchsuchen wir den Graphen $G - \{a, b\}$ ausgehend vom Startknoten u_1, so erhalten wir eine Suchordnung $(u_1, \ldots, u_{n-2})$. Starten wir nun `greedy-color` mit der Reihenfolge $(a, b, u_{n-2}, \ldots, u_1)$, so erhalten wir eine d-Färbung c für G mit $c(a) = c(b) = 1$. Zudem hat Knoten u_i, $i > 1$, einen Nachbarn u_j mit $j < i$, weshalb $c(u_i) \leq \deg(u_i) \leq d$ ist. Zuletzt erhält auch u_1 eine Farbe $c(u_1) \leq d$, da die Nachbarn a und b von u_1 dieselbe Farbe haben. ■

In den Übungen wird folgende Folgerung aus dem Beweis des Satzes von Brooks gezeigt.

Korollar 2.45. *Es gibt einen Linearzeitalgorithmus, der für jeden Graphen G mit $\Delta(G) \leq 3$ eine $\chi(G)$-Färbung berechnet.*

2.4 Kantenfärbungen

Neben der Frage, mit wievielen Farben die Knoten eines Graphen gefärbt werden können, muss bei vielen Anwendungen auch eine Kantenfärbung mit möglichst wenigen Farben gefunden werden.

Definition 2.46. *Sei $G = (V, E)$ ein Graph und sei $k \in \mathbb{N}$.*

*a) Eine Abbildung $c\colon E \to \mathbb{N}$ heißt **Kantenfärbung** von G, wenn $c(e) \neq c(e')$ für alle $e \neq e' \in E$ mit $e \cap e' \neq \emptyset$ gilt.*

*b) G heißt **k-kantenfärbbar**, falls eine Kantenfärbung $c\colon E \to \{1, \ldots, k\}$ existiert.*

*c) Die **kantenchromatische Zahl** oder der **chromatische Index** von G ist*

$$\chi'(G) = \min\{k \in \mathbb{N} \mid G \text{ ist } k\text{-kantenfärbbar}\}.$$

Eine k-Kantenfärbung $c\colon E \to \mathbb{N}$ muss also zwei Kanten, die einen Knoten gemeinsam haben, verschiedene Farben zuweisen. Daher bildet jede ***Farbklasse*** $E_i = \{e \in E \mid f(e) = i\}$ ein Matching von G, d.h. c zerlegt E in k disjunkte Matchings. Umgekehrt liefert jede Zerlegung von E in k disjunkte Matchings eine k-Kantenfärbung von G.

Neben Graphen treten in manchen Anwendungen auch ***Multigraphen*** $G = (V, E)$ auf. Diese können mehr als eine Kante zwischen zwei Knoten haben, d.h. E ist eine Multimenge auf $\binom{V}{2}$.

Eine ***Multimenge*** A auf einer Grundmenge M lässt sich durch eine Funktion $v_A\colon M \to \mathbb{N}$ beschreiben, wobei $v_A(a)$ die Anzahl der Vorkommen des Elements a in A angibt. Die Mächtigkeit von A ist $|A| = \sum_{a \in A} v_A(a)$.

Wie bei Graphen gehen wir davon aus, dass jede Kante $e = \{u, v\}$ eines Multigraphen $G = (V, E)$ zwei verschiedene Endpunkte $u \neq v$ hat, d.h. G ist schlingenfrei. In G gibt es genau $v_E(e) = v_E(u, v)$ Kanten zwischen den beiden Knoten u und v. Die Zahl $v_E(e)$ wird auch als ***(Kanten-)Vielfachheit*** von e bezeichnet. Ein wichtiger Parameter von Multigraphen ist die maximale Kantenvielfachheit

$$v(G) = \max_{e \in E} v_E(e),$$

die auch als ***(Graph-)Vielfachheit*** von G bezeichnet wird. Der ***Grad*** eines Knotens $\in V$ ist $\deg_G(u) = \sum_{v \in N(u)} v_E(u, v)$ und der ***Maximalgrad*** von G ist wie üblich $\Delta(G) = \max_{u \in V} \deg_G(u)$.

Eine k-Kantenfärbung für einen Multigraphen $G = (V, E)$ lässt sich durch eine Funktion c beschreiben, die jeder Kante $e \in \binom{V}{2}$ eine Menge $c(e) \subseteq \{1, \ldots, k\}$ von $|c(e)| = v_E(e)$ Farben zuordnet, so dass $c(e) \cap c(e') = \emptyset$ für alle $e \neq e' \in \binom{V}{2}$ mit $e \cap e' \neq \emptyset$ gilt.

Beispiel 2.47.

$$\chi'(C_n) = \begin{cases} 2, & n \text{ gerade,} \\ 3, & \text{sonst,} \end{cases}$$

$$\chi'(K_n) = 2\lceil n/2 \rceil - 1 = \begin{cases} n-1, & n \text{ gerade,} \\ n, & \text{sonst.} \end{cases}$$

Das Kantenfärbungsproblem für einen Graphen G lässt sich leicht auf das Knotenfärbungsproblem für einen Graphen G' reduzieren.

Definition 2.48. *Sei $G' = (V, E')$ ein Graph mit $m \geq 1$ Kanten. Dann heißt der Graph $G = L(G') = (E', E)$ mit*

$$E = \left\{ \{e, e'\} \in \binom{E'}{2} \;\middle|\; e \cap e' \neq \emptyset \right\}$$

*der **Kantengraph** oder **Line-Graph** von G'.*

Ist G' ein Multigraph, so verwenden wir als Knotenmenge von $L(G')$ eine Menge $V_{E'}$ mit der Mächtigkeit $|V_{E'}| = |E'|$, die $v_{E'}(e)$ verschiedene Kopien $e^1, \ldots, e^{v_E(e)}$ jeder Kante $e \in E'$ enthält. Die folgenden Beziehungen zwischen einem (Multi-)Graphen G' und dem zugehörigen Line-Graphen lassen sich leicht verifizieren.

Proposition 2.49. *Für den Line-Graphen $G = L(G')$ eines Multigraphen G' gilt:*

(i) $n(G) = m(G')$,
(ii) $\chi(G) = \chi'(G')$,
(iii) $\alpha(G) = \mu(G')$,
(iv) $\omega(G) \geq \Delta(G')$,
(v) $\Delta(G) = \max_{e \in E'} \deg_{G'}(u) + \deg_{G'}(v) - v_{E'}(e) - 1 \leq 2\Delta(G') - 2$, *wobei u und v die Endpunkte der Kante $e = \{u, v\}$ sind.*

Damit erhalten wir aus den Abschätzungen $\omega(G) \leq \chi(G) \leq \Delta(G) + 1$ und $n/\alpha(G) \leq \chi(G) \leq n - \alpha(G) + 1$ die folgenden Abschätzungen für $\chi'(G')$.

Lemma 2.50. *Für jeden Multigraphen mit $m \geq 1$ Kanten gilt $\Delta \leq \chi' \leq 2\Delta - 1$ und $m/\mu \leq \chi' \leq m - \mu + 1$.*

Korollar 2.51. *Für jeden regulären Multigraphen mit einer ungeraden Knotenzahl und $m \geq 1$ Kanten gilt $\chi' \geq \Delta + 1 \geq 3$.*

Beweis. Wegen $\mu \leq (n-1)/2$ und $m = n\Delta/2$ folgt $\chi' \geq m/\mu \geq n\Delta/(n-1) > \Delta$. Da n ungerade und $m \geq 1$ ist, folgt $\Delta \geq 2$. ■

Als nächstes geben wir einen effizienten Algorithmus an, der für jeden Graphen eine $(\Delta + 1)$-Kantenfärbung berechnet. Hierfür benötigen wir folgende Begriffe.

Definition 2.52. *Sei $G = (V, E)$ ein Graph und sei $c\colon E \to \{1, \ldots, k\}$ eine k-Kantenfärbung von G. Weiter sei $F \subseteq \{1, \ldots, k\}$ und es gelte $1 \leq i \neq j \leq k$.*

*a) Ein Nachbar v von u heißt **F-Nachbar** von u, wenn $c(u, v) \in F$ ist. Im Fall $F = \{i\}$ nennen wir v auch den **i-Nachbarn** von u.*

*b) Die Farbe i ist **frei** an einem Knoten u (kurz $i \in \mathit{free}(u)$), falls u keinen i-Nachbarn hat.*

*c) Der Graph $G_{ij} = (V, E_{ij})$ mit $E_{ij} = \left\{e \in E \,\middle|\, c(e) \in \{i, j\}\right\}$ heißt **(i, j)-Subgraph** von G.*

*d) Jede Komponente G' von G_{ij} heißt **(i, j)-Komponente** von G. Je nachdem ob G' ein Pfad oder ein Kreis ist, nennen wir G' auch einen **(i, j)-Pfad** bzw. **(i, j)-Kreis** in G (bzgl. c).*

Man sieht leicht, dass jede (i, j)-Komponente G' von G entweder ein Pfad der Länge $l \geq 0$ oder ein Kreis gerader Länge ist. Zudem können wir aus c eine weitere k-Kantenfärbung c' von G gewinnen, indem wir die beiden Farben i und j entlang der Kanten von G' vertauschen.

Satz 2.53 (Vizing 1964)**.** *Für jeden Graphen G gilt $\chi'(G) \leq \min_{e \in E} \Delta(G - e) + 1 \leq \Delta(G) + 1$.*

Beweis. Wir führen Induktion über m. Der IA $m = 0$ ist klar. Für den IS sei $G' = (V, E')$ ein Graph mit $m + 1$ Kanten und sei $k = \min_{e \in E'} \Delta(G' - e) + 1$. Wir wählen eine beliebige Kante $e_1 = \{y_0, y_1\} \in E'$, so dass der Graph $G = G' - e_1$ den Maximalgrad $\Delta(G) = k - 1$ hat. Dann hat G nach IV eine k-Kantenfärbung $c\colon E \to \{1, \ldots, k\}$. Da zudem unter c an jedem Knoten u mindestens $k - \deg_G(u) \geq 1$ Farben frei sind, folgt $\mathit{free}(u) \neq \emptyset$ für alle $u \in V$. Betrachte nun folgende Prozeduren.

Prozedur `expand`(G, c, y_0, y_1)

```
1  ℓ := 1
2  wähle α_1 ∈ free(y_1)
3  while α_ℓ ∉ free(y_0) ∪ {α_1, ..., α_{ℓ-1}} do
4     sei y_{ℓ+1} der α_ℓ-Nachbar von y_0
5     wähle α_{ℓ+1} ∈ free(y_{ℓ+1})
6     ℓ := ℓ + 1
7  wähle 0 ≤ i < ℓ minimal mit α_ℓ ∈ free(y_0) ∪ {α_1, ..., α_i}
8  if i = 0 then // α_ℓ ∈ free(y_0)
9     recolor(ℓ, α_ℓ)
10 else // α_ℓ = α_i
```

```
11    wähle eine Farbe α_0 ∈ free(y_0)
12    berechne den (α_0, α_i)-Pfad P mit Startknoten y_ℓ und
13    vertausche dabei die Farben α_0 und α_i entlang P
14    sei z der Endknoten von P // z = y_ℓ ist möglich
15    case
16      z = y_0 : recolor(i, α_i)
17      z = y_i : recolor(i, α_0)
18    else recolor(ℓ, α_0)
19  return c
```

Prozedur `recolor`(i, α)

```
1  for j := 1 to i − 1 do c(y_0, y_j) := α_j
2  c(y_0, y_i) := α
```

Wir verifizieren, dass die Abbildung c eine Kantenfärbung von G' ist.

Fall 1 $\alpha_\ell \in \mathit{free}(y_0)$: Da die Farbe α_ℓ an y_0 und für $j = 1, \dots, \ell$ die Farbe α_j an y_j frei ist, können wir $\{y_0, y_j\}$ mit α_j färben.

Fall 2 $z = y_0$: In diesem Fall erreicht P den Knoten $z = y_0$ über die Kante $\{y_0, y_{i+1}\}$. Nach dem Vertauschen von α_0 und α_i entlang P hat diese Kante dann die Farbe α_0, weshalb wir die Kanten $\{y_0, y_j\}$ für $j = 1, \dots, i$ mit α_j färben können.

Fall 3 $z = y_i$: Da $\alpha_i \in \mathit{free}(y_i) \cap \mathit{free}(y_\ell)$ ist, müssen die Endkanten von P mit α_0 gefärbt sein. Nach Vertauschen von α_0 und α_i entlang P ist daher die Farbe α_0 an y_0 und y_i frei, weshalb wir die Kante $\{y_0, y_i\}$ mit α_0 und die Kanten $\{y_0, y_j\}$ für $j = 1, \dots, i-1$ mit α_j färben können.

Fall 4 In allen anderen Fällen (d.h. $z \notin \{y_0, y_i\}$) ist die Farbe α_0 nach Vertauschen von α_0 und α_i entlang P neben y_0 auch an y_ℓ frei, weshalb wir die Kante $\{y_0, y_\ell\}$ mit α_0 färben können. Zudem bleibt die Farbe α_j für $j = 1, \dots, \ell - 1$ an y_j frei (wegen $\alpha_j \notin \{\alpha_0, \alpha_i\}$ gilt dies auch im Fall $y_j = z$). Daher können wir die Kanten $\{y_0, y_j\}$ für $j = 1, \dots, \ell - 1$ mit α_j färben. ■

Da die Prozedur `expand` mit Hilfe geeigneter Datenstrukturen so implementiert werden kann, dass jeder Aufruf Zeit $\mathcal{O}(n)$ erfordert, und diese Prozedur m-mal aufgerufen wird, um alle m Kanten eines gegebenen Graphen G zu färben, ergibt sich eine Gesamtlaufzeit von $\mathcal{O}(nm)$.

Für einen Graphen G kann $\chi'(G)$ nur einen der beiden Werte $\Delta(G)$ oder $\Delta(G) + 1$ annehmen. Graphen G mit $\chi'(G) = \Delta(G)$ heißen ***Klasse 1*** und Graphen G mit $\chi'(G) = \Delta(G) + 1$ heißen ***Klasse 2***.

Neben allen bipartiten Graphen sind auch die vollständigen Graphen K_n für gerades n Klasse 1. Zudem sind alle planaren Graphen G mit $\Delta(G) \geq 7$ Klasse 1. Für $2 \leq d \leq 5$ existieren planare Graphen G mit $\Delta(G) = d$, die Klasse 2 sind. Für $d = 6$ ist dies offen.

Das Problem, für einen gegebenen Graphen G zu entscheiden, ob er Klasse 1 ist (also $\chi'(G) \leq \Delta(G)$ gilt), ist NP-vollständig.

Der Satz von Vizing lässt sich wie folgt auf Multigraphen verallgemeinern.

Satz 2.54 (Vizing 1964)**.** *Für jeden Multigraphen $G = (V, E)$ gilt $\chi'(G) \leq \max_{u,v \in V}(\deg(u) + v_E(u, v)) \leq \Delta(G) + v(G)$.*

In den Übungen leiten wir noch die folgenden oberen Schranken her.

Korollar 2.55. *Für jeden Multigraphen $G = (V, E)$ gilt:*

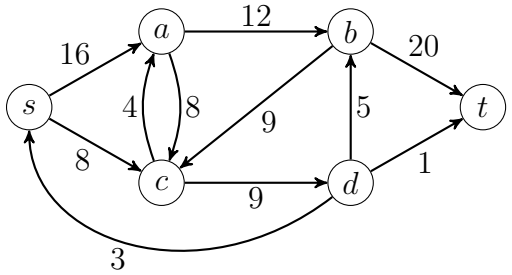
(i) $\chi'(G) \leq 3\Delta(G)/2$,

(ii) falls G bipartit (d.h. $\chi(G) \leq 2$) ist, dann ist $\chi'(G) = \Delta(G)$.

3 Flüsse in Netzwerken

Definition 3.1. *Ein* ***Netzwerk*** $N = (V, E, s, t, c)$ *besteht aus einem gerichteten Graphen* $G = (V, E)$ *mit einer* ***Quelle*** $s \in V$ *und einer* ***Senke*** $t \in V$ *sowie einer* ***Kapazitätsfunktion*** $c : V \times V \to \mathbb{N}$. *Zudem muss jede Kante* $(u, v) \in E$ *positive Kapazität* $c(u, v) > 0$ *und jede Nichtkante* $(u, v) \notin E$ *muss die Kapazität* $c(u, v) = 0$ *haben.*

Die folgende Abbildung zeigt ein Netzwerk N.



Definition 3.2.

a) Ein ***Fluss in N*** *ist eine Funktion* $f : V \times V \to \mathbb{Z}$ *mit*

$f(u, v) \leq c(u, v)$, *(Kapazitätsbedingung)*

$f(u, v) = -f(v, u)$, *(Antisymmetrie)*

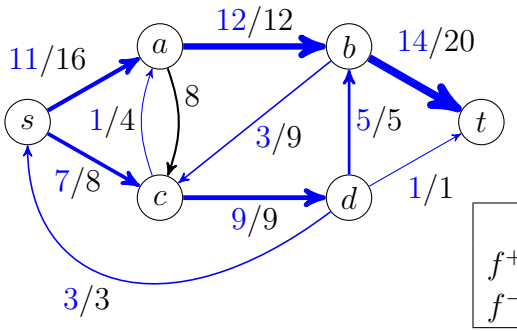
$\sum_{v \in V} f(u, v) = 0$ *für alle* $u \in V \setminus \{s, t\}$ *(Kontinuität)*

b) Der ***Fluss in den Knoten u*** *ist* $f^-(u) = \sum_{v \in V} \max\{0, f(v, u)\}$.

c) Der ***Fluss aus u*** *ist* $f^+(u) = \sum_{v \in V} \max\{0, f(u, v)\}$.

d) Die ***Größe von f*** *ist* $|f| = f^+(s) - f^-(s) = \sum_{v \in V} f(s, v)$.

Die Antisymmetrie impliziert, dass $f(u, u) = 0$ für alle $u \in V$ ist, d.h. wir können annehmen, dass G schlingenfrei ist. Die folgende Abbildung zeigt einen Fluss f in N.



u	s	a	b	c	d	t
$f^+(u)$	18	12	17	10	9	0
$f^-(u)$	3	12	17	10	9	15

3.1 Der Ford-Fulkerson-Algorithmus

Wie kann man für einen Fluss f in einem Netzwerk N entscheiden, ob er vergrößert werden kann? Diese Frage ist leicht zu beantworten, falls f auf $V \times V$ den Wert 0 hat: In diesem Fall genügt es, in $G = (V, E)$ einen Pfad von s nach t zu finden. Andernfalls können wir zu N und f ein Netzwerk N_f konstruieren, so dass f genau dann vergrößert werden kann, wenn sich in N_f der Nullfluss vergrößern lässt.

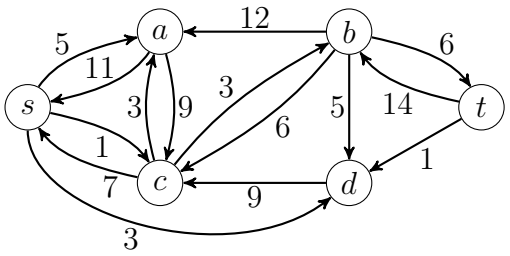
Definition 3.3. *Sei* $N = (V, E, s, t, c)$ *ein Netzwerk und sei* f *ein Fluss in* N. *Das zugeordnete* ***Restnetzwerk*** *ist* $N_f = (V, E_f, s, t, c_f)$ *mit der Kapazität*

$$c_f(u, v) = c(u, v) - f(u, v)$$

und der Kantenmenge

$$E_f = \{(u, v) \in V \times V \mid c_f(u, v) > 0\}.$$

Zum Beispiel führt obiger Fluss auf das folgende Restnetzwerk N_f:



Definition 3.4. *Sei $N_f = (V, E_f, s, t, c_f)$ ein Restnetzwerk. Dann heißt jeder s-t-Pfad P in (V, E_f)* ***Zunahmepfad*** *in N_f. Die* ***Kapazität von P in N_f*** *ist*

$$c_f(P) = \min\{c_f(u,v) \mid (u,v) \text{ liegt auf } P\}$$

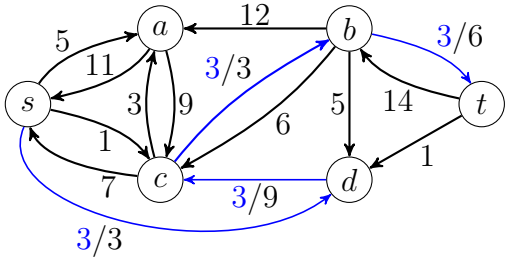
und der ***zu P gehörige Fluss in N_f*** *ist*

$$f_P(u,v) = \begin{cases} c_f(P), & (u,v) \text{ liegt auf } P, \\ -c_f(P), & (v,u) \text{ liegt auf } P, \\ 0, & \text{sonst.} \end{cases}$$

$P = (u_0, \ldots, u_k)$ ist also genau dann ein Zunahmepfad in N_f, falls

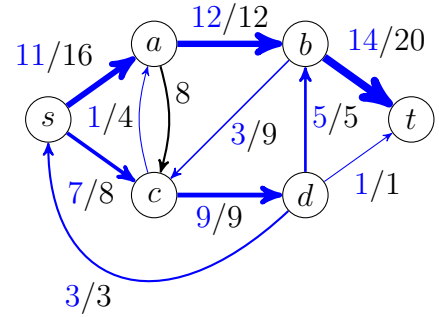
- $u_0 = s$ und $u_k = t$ ist,
- die Knoten $u_0, \ldots, u_k$ paarweise verschieden sind
- und $c_f(u_i, u_{i+1}) > 0$ für $i = 0, \ldots, k-1$ ist.

Die folgende Abbildung zeigt den zum Zunahmepfad $P = s, c, b, t$ gehörigen Fluss f_P in N_f. Die Kapazität von P ist $c_f(P) = 4$.

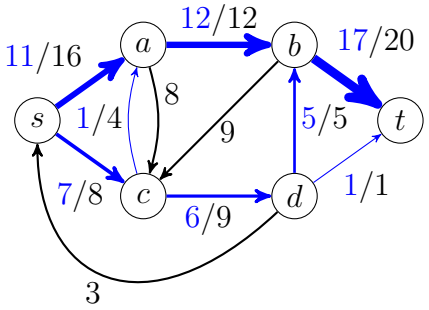


Es ist leicht zu sehen, dass f_P tatsächlich ein Fluss in N_f ist. Durch Addition der beiden Flüsse f und f_P erhalten wir einen Fluss $f' = f + f_P$ in N der Größe $|f'| = |f| + |f_P| = |f| + c_f(P) > |f|$.

Fluss f:



Fluss $f' = f + f_P$:

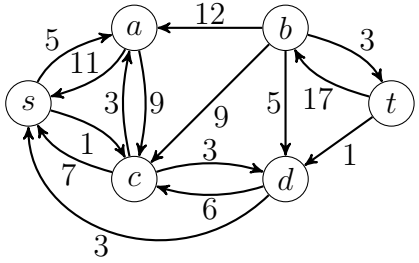


Nun können wir den **Ford-Fulkerson-Algorithmus** angeben.

Algorithmus `Ford-Fulkerson`(V, E, s, t, c)

```
1  for all (u,v) ∈ E ∪ E^R do
2    f(u,v) := 0
3  while es gibt einen Zunahmepfad P in N_f do
4    f := f + f_P
```

Beispiel 3.5. *Für den neuen Fluss erhalten wir nun folgendes Restnetzwerk:*



In diesem existiert kein Zunahmepfad mehr. ◁

Um zu beweisen, dass der Algorithmus von Ford-Fulkerson tatsächlich einen Maximalfluss berechnet, zeigen wir, dass es nur dann im Restnetzwerk N_f keinen Zunahmepfad mehr gibt, wenn der Fluss f maximal ist. Hierzu benötigen wir den Begriff des Schnitts.

Definition 3.6. *Sei $N = (V, E, s, t, c)$ ein Netzwerk und sei $\emptyset \subsetneq S \subsetneq V$. Dann heißt die Menge $E(S) = \{(u, v) \in E \mid u \in S, v \notin S\}$* ***Kantenschnitt*** *(oder einfach* ***Schnitt****; oft wird auch einfach S als Schnitt bezeichnet). Die* ***Kapazität eines Schnittes S*** *ist*

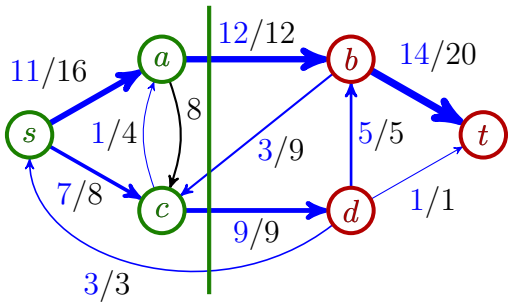
$$c(S) = \sum_{u \in S, v \notin S} c(u, v).$$

Ist f ein Fluss in N, so heißt

$$f(S) = \sum_{u \in S, v \notin S} f(u, v)$$

der ***Nettofluss*** *(oder einfach* ***Fluss****)* ***durch den Schnitt S****. Ist $u \in S$ und $v \notin S$, so heißt S auch* ***u-v-Schnitt****.*

Beispiel 3.7. *Betrachte folgenden Schnitt $S = \{s, a, c\}$ in N:*



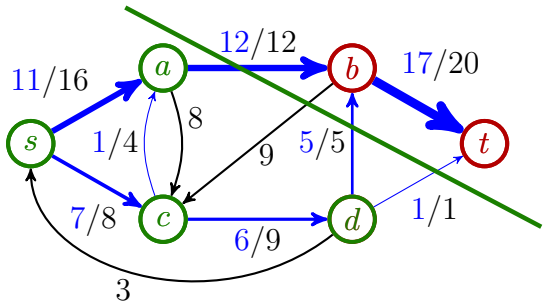
Dieser Schnitt hat die Kapazität

$$c(S) = c(a, b) + c(c, d) = 12 + 9 = 21$$

und der Fluss f durch ihn ist

$$\begin{aligned} f(S) &= f(a, b) + f(c, b) + f(c, d) + f(s, d) \\ &= 12 - 3 + 9 - 3 \\ &= 15. \end{aligned}$$

Dagegen hat der Schnitt $S' = \{s, a, c, d\}$



die Kapazität

$$\begin{aligned} c(S') = c(a, b) \ + \ c(d, b) \ + \ c(d, t) &= 12 + 5 + 1 = 18 \\ = f'(a, b) + f'(d, b) + f'(d, t) &= f'(S'), \end{aligned}$$

die mit dem Fluss f' durch S' übereinstimmt. ◁

Lemma 3.8. *Für jeden s-t-Schnitt S und jeden Fluss f gilt*

$$|f| = f(S) \leq c(S).$$

Beweis. Wir zeigen zuerst die Ungleichung $f(S) \leq c(S)$. Wegen $f(u, v) \leq c(u, v)$ für alle $(u, v) \in V \times V$ folgt

$$f(S) \ = \ \sum_{u \in S, v \notin S} f(u, v) \ \leq \ \sum_{u \in S, v \notin S} c(u, v) \ = \ c(S).$$

Die Gleichheit $|f| = f(S)$ zeigen wir durch Induktion über $k = |S|$.

$k = 1$: In diesem Fall ist $S = \{s\}$ und somit

$$|f| \ = \ \sum_{v} f(s, v) \ = \ \sum_{v \neq s} f(s, v) \ = \ f(S).$$

$k-1 \rightsquigarrow k$**:** Sei S ein Schnitt mit $|S| = k > 1$ und sei $w \in S - \{s\}$. Betrachte den Schnitt $S' = S - \{w\}$. Dann gilt

$$f(S) = \sum_{u \in S, v \notin S} f(u,v) = \sum_{u \in S', v \notin S} f(u,v) + \sum_{v \notin S} f(w,v)$$

und

$$f(S') = \sum_{u \in S', v \notin S'} f(u,v) = \sum_{u \in S', v \notin S} f(u,v) + \sum_{u \in S'} f(u,w).$$

Daher folgt

$$f(S) - f(S') = \sum_{v \notin S} f(w,v) - \sum_{u \in S'} f(u,w) \quad = \sum_{v \neq w} f(w,v) = 0.$$

Nach Induktionsvoraussetzung folgt somit $f(S) = f(S') = |f|$. ■

Satz 3.9 (Min-Cut-Max-Flow-Theorem)**.** *Sei f ein Fluss in einem Netzwerk $N = (V, E, s, t, c)$. Dann sind folgende Aussagen äquivalent:*

1. *f ist maximal, d.h. für jeden Fluss f' in N gilt $|f'| \leq |f|$.*
2. *In N_f existiert kein Zunahmepfad.*
3. *Es gibt einen s-t-Schnitt S in N mit $c(S) = |f|$.*

Beweis. Die Implikation „1 ⇒ 2“ ist klar, da die Existenz eines Zunahmepfads in N_f zu einer Vergrößerung von f führen würde.

Für die Implikation „2 ⇒ 3“ betrachten wir den Schnitt

$$S = \{u \in V \mid u \text{ ist in } N_f \text{ von } s \text{ aus erreichbar}\}.$$

Da in N_f kein Zunahmepfad existiert, gilt dann

- $s \in S$, $t \notin S$ und
- $c_f(u,v) = 0$ für alle $u \in S$ und $v \notin S$.

Wegen $c_f(u,v) = c(u,v) - f(u,v)$ folgt somit

$$|f| = f(S) = \sum_{u \in S, v \notin S} f(u,v) = \sum_{u \in S, v \notin S} c(u,v) = c(S).$$

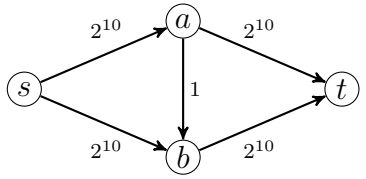
Die Implikation „3 ⇒ 1“ ergibt sich aus der Tatsache, dass im Fall $c(S) = |f|$ für jeden Fluss f' die Abschätzung $|f'| = f'(S) \leq c(S) = |f|$ gilt. ■

Der obige Satz gilt auch für Netzwerke mit Kapazitäten in $\mathbb{R}^+$.

Sei $c_0 = c(S)$ die Kapazität des Schnittes $S = \{s\}$. Dann durchläuft der Ford-Fulkerson-Algorithmus die while-Schleife höchstens c_0-mal. Bei jedem Durchlauf ist zuerst das Restnetzwerk N_f und danach ein Zunahmepfad in N_f zu berechnen.

Die Berechnung des Zunahmepfads P kann durch Breitensuche in Zeit $\mathcal{O}(n+m)$ erfolgen. Da sich das Restnetzwerk nur entlang von P ändert, kann es in Zeit $\mathcal{O}(n)$ aktualisiert werden. Jeder Durchlauf benötigt also Zeit $\mathcal{O}(n+m)$, was auf eine Gesamtlaufzeit von $\mathcal{O}(c_0(n+m))$ führt. Da der Wert von c_0 jedoch exponentiell in der Länge der Eingabe (also der Beschreibung des Netzwerkes N) sein kann, ergibt dies keine polynomielle Zeitschranke. Bei Netzwerken mit Kapazitäten in $\mathbb{R}^+$ kann der Ford-Fulkerson-Algorithmus sogar unendlich lange laufen (siehe Übungen).

Bei nebenstehendem Netzwerk benötigt Ford-Fulkerson zur Bestimmung des Maximalflusses abhängig von der Wahl der Zunahmepfade zwischen 2 und 2^{11} Schleifendurchläufe.



Im günstigsten Fall wird nämlich ausgehend vom Nullfluss f_1 zuerst der Zunahmepfad $P_1 = (s,a,t)$ mit der Kapazität 2^{10} und dann im Restnetzwerk N_{f_1} der Pfad $P_2 = (s,b,t)$ mit der Kapazität 2^{10} gewählt.

Im ungünstigsten Fall werden abwechselnd die beiden Zunahmepfade $P_1 = (s,a,b,t)$ und $P_2 = (s,b,a,t)$ (also $P_i = P_1$ für ungerades i und

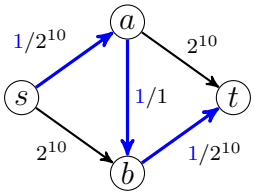
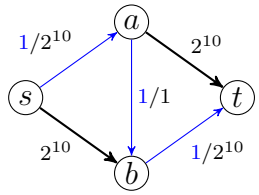
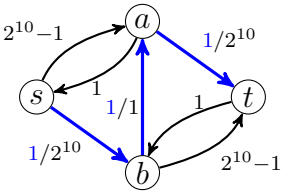
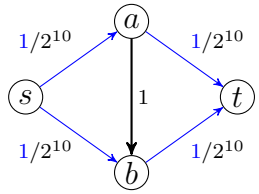
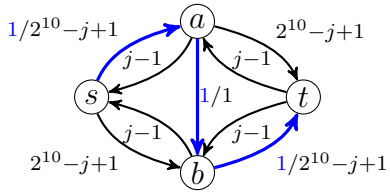
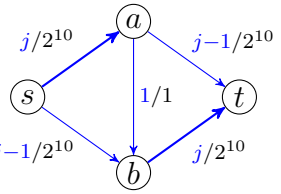
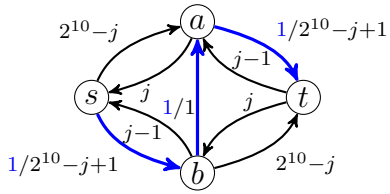
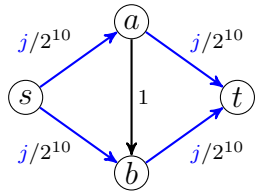
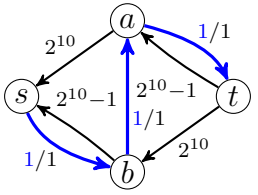
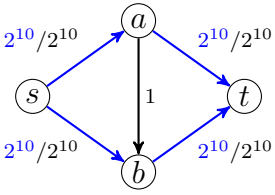
$P_i = P_2$ für gerades i) mit der Kapazität 1 gewählt. Dies führt auf insgesamt 2^{11} Schleifendurchläufe (siehe nebenstehende Tabelle).

Nicht nur in diesem Beispiel lässt sich die exponentielle Laufzeit wie folgt vermeiden:

- Man betrachtet nur Zunahmepfade mit einer geeignet gewählten Mindestkapazität. Dies führt auf eine Laufzeit, die polynomiell in n, m und $\log c_0$ ist (siehe Übungen).
- Man bestimmt in jeder Iteration einen kürzesten Zunahmepfad im Restnetzwerk mittels Breitensuche in Zeit $\mathcal{O}(n+m)$. Diese Vorgehensweise führt auf den *Edmonds-Karp-Algorithmus*, der eine Laufzeit von $\mathcal{O}(nm^2)$ hat (unabhängig von der Kapazitätsfunktion).
- Man bestimmt in jeder Iteration einen Fluss g im Restnetzwerk N_f, der nur Kanten benutzt, die auf einem kürzesten s-t-Pfad in N_f liegen. Zudem hat g die Eigenschaft, dass g auf jedem kürzesten s-t-Pfad P mindestens eine Kante $e \in P$ *sättigt* (d.h. der Fluss $g(e)$ durch e schöpft die Restkapazität $c_f(e)$ von e vollkommen aus), weshalb diese Kante in der nächsten Iteration fehlt. Dies führt auf den *Algorithmus von Dinitz*. Da die Länge der kürzesten s-t-Pfade im Restnetzwerk in jeder Iteration um mindestens 1 zunimmt, liegt nach spätestens $n-1$ Iterationen ein maximaler Fluss vor. Dinitz hat gezeigt, dass der Fluss g in Zeit $\mathcal{O}(nm)$ bestimmt werden kann. Folglich hat der Algorithmus von Dinitz eine Laufzeit von $\mathcal{O}(n^2m)$. Malhotra, Kumar und Maheswari fanden später einen $\mathcal{O}(n^2)$-Algorithmus zur Bestimmung von g. Damit lässt sich die Gesamtlaufzeit auf $\mathcal{O}(n^3)$ verbessern.

3.2 Der Edmonds-Karp-Algorithmus

Der Edmonds-Karp-Algorithmus ist eine spezielle Form von Ford-Fulkerson, die nur Zunahmepfade mit möglichst wenigen Kanten benutzt, welche mittels Breitensuche bestimmt werden.

i	Fluss f_{P_i} in N_{f_i}	neuer Fluss f_{i+1} in N
1		
2		
⋮		
$2j-1$, $1 < j \leq 2^{10}$		
$2j$, $1 < j < 2^{10}$		
⋮		
2^{11}		

Algorithmus `Edmonds-Karp`(V, E, s, t, c)

```
1  for all (u,v) ∈ E ∪ E^R do
2    f(u,v) := 0
3  repeat
4    P := zunahmepfad(f)
5    if P ≠ ⊥ then addierepfad(f, P)
6  until P = ⊥
```

Prozedur `zunahmepfad`(f)

```
1  for all v ∈ V \ {s} do
2    parent(v) := ⊥
3  parent(s) := s
4  Q := (s)
5  while Q ≠ () ∧ parent(t) = ⊥ do
6    u := dequeue(Q)
7    for all e = (u,v) ∈ E ∪ E^R do
8      if c(e) − f(e) > 0 ∧ parent(v) = ⊥ then
9        c'(e) := c(e) − f(e)
10       parent(v) := u
11       enqueue(Q, v)
12 if parent(t) = ⊥ then
13   P := ⊥
14 else
15   P := parent-Pfad von s nach t
16   c_f(P) := min{c'(e) | e ∈ P}
17 return P
```

Prozedur `addierepfad`(f, P)

```
1  for all e ∈ P do
2    f(e) := f(e) + c_f(P)
3    f(e^R) := f(e^R) − c_f(P)
```


Die Prozedur `zunahmepfad(f)` berechnet im Restnetzwerk N_f einen (gerichteten) s-t-Pfad P, sofern ein solcher existiert. Dies ist genau dann der Fall, wenn die while-Schleife mit $\texttt{parent}(t) \neq \bot$ abbricht. Der Pfad P lässt sich dann mittels `parent` wie folgt zurückverfolgen. Sei

$$u_i = \begin{cases} t, & i = 0, \\ \texttt{parent}(u_{i-1}), & i > 0 \text{ und } u_{i-1} \neq s \end{cases}$$

und sei $\ell = \min\{i \geq 0 \mid u_i = s\}$. Dann ist $u_\ell = s$ und $P = (u_\ell, \ldots, u_0)$ ein s-t-Pfad, den wir als den ***parent-Pfad*** von s nach t bezeichnen.

Satz 3.10. *Der Edmonds-Karp-Algorithmus durchläuft die repeat-Schleife höchstens $(nm/2)$-mal und hat somit eine Laufzeit von $O(nm^2)$.*

Beweis. Sei k die Anzahl der Schleifendurchläufe und seien $P_1, \ldots, P_k$ die Zunahmepfade, die der Algorithmus bei Eingabe N berechnet, d.h. $f_{i+1} = f_i + f_{P_i}$, wobei f_1 der triviale Nullfluss ist. Eine Kante e auf P_i heißt ***kritisch*** für P_i, falls der Fluss f_{P_i} im Restnetzwerk N_{f_i} die Kante e ***sättigt***, d.h. $f_{P_i}(e) = c_{f_i}(e)$. Man beachte, dass eine kritische Kante e für P_i wegen $c_{f_{i+1}}(e) = c_{f_i}(e) - f_{P_i}(e) = 0$ nicht in $N_{f_{i+1}}$ enthalten ist, wohl aber die Kante e^R, da $c_{f_{i+1}}(e^R) = c(e^R) - f_{i+1}(e^R) = c(e^R) + f_{i+1}(e) = c(e^R) + c(e) > 0$ ist.

Sei $d_i(u, v)$ die minimale Länge eines Pfades von u nach v im Restnetzwerk N_{f_i} und sei $l_i = d_i(s, t)$ die Länge von P_i. Wir zeigen zuerst, dass die Abstände jedes Knotens $u \in V$ von s und von t beim Übergang von N_{f_i} zu $N_{f_{i+1}}$ höchstens zu- aber nicht abnehmen. Hierzu beweisen wir für jeden kürzesten Pfad $P = (u_0, \ldots, u_l)$ von $u_0 = s$ nach $u_l = u$ in $N_{f_{i+1}}$ (d.h. $d_{i+1}(s, u_h) = h$ für $h = 0, \ldots, l$) die Ungleichungen

$$d_i(s, u_h) \leq d_i(s, u_{h-1}) + 1 \text{ für } h = 1, \ldots, l.$$

Falls die Kante $e = (u_{h-1}, u_h)$ auch in N_{f_i} enthalten ist, ist nichts zu zeigen. Andernfalls muss $f_{i+1}(e) \neq f_i(e)$ sein, d.h. e oder e^R müssen auf P_i liegen. Da e nicht in N_{f_i} ist, muss $e^R = (u_h, u_{h-1})$ auf P_i liegen. Da P_i ein kürzester Pfad von s nach t in N_{f_i} ist, folgt $d_i(s, u_{h-1}) =$

$d_i(s,u_h)+1$, was $d_i(s,u_h) = d_i(s,u_{h-1})-1 \leq d_i(s,u_{h-1})+1$ impliziert. Nun folgt

$$d_i(s,u) \leq d_i(s,u_{l-1}) + 1 \leq \cdots \leq d_i(s,s) + l = l = d_{i+1}(s,u).$$

Vollkommen analog lässt sich $d_i(u,t) \leq d_{i+1}(u,t)$ zeigen, womit wir folgende Behauptung bewiesen haben.

Behauptung 3.11. *Für jeden Knoten $u \in V$ gilt $d_i(s,u) \leq d_{i+1}(s,u)$ und $d_i(u,t) \leq d_{i+1}(u,t)$.*

Daraus ergibt sich nun folgende Behauptung.

Behauptung 3.12. *Für $1 \leq i < j \leq k$ gilt: Falls $e = (u,v)$ in P_i und $e^R = (v,u)$ in P_j enthalten ist, so ist $l_j \geq l_i + 2$.*

Da P_i und P_j kürzeste Zunahmepfade sind, folgt

$$l_j = d_j(s,t) = \underbrace{d_j(s,v)}_{\geq d_i(s,v)} + \underbrace{d_j(u,t)}_{\geq d_i(u,t)} + 1 \geq \underbrace{d_i(s,v)}_{d_i(s,u)+1} + \underbrace{d_i(u,t)}_{d_i(v,t)+1} + 1 = l_i + 2.$$

Da jeder Zunahmepfad P_i mindestens eine kritische Kante enthält und $E \cup E^R$ höchstens m Kantenpaare der Form $\{e, e^R\}$ enthält, impliziert schließlich folgende Behauptung, dass $k \leq mn/2$ ist.

Behauptung 3.13. *Zwei Kanten e und e^R sind zusammen höchstens $n/2$-mal kritisch.*

Seien $P_{i_1}, \ldots, P_{i_h}$, $i_1 < \cdots < i_h$, die Pfade, für die eine der Kanten in $\{e, e^R\}$ kritisch ist. Falls $e' \in \{e, e^R\}$ kritisch für P_{i_j} mit $1 \leq j < h$ ist, dann verschwindet e' aus $N_{f_{i_j+1}}$. Daher muss unabhängig davon, ob e' oder e'^R kritisch für $P_{i_{j+1}}$ ist, ein Pfad $P_{j'}$ mit $i_j < j' \leq i_{j+1}$ existieren, der e'^R enthält. Wegen Behauptung 3.11 und Behauptung 3.12 ist $\ell_{i_{j+1}} \geq \ell_{j'} \geq \ell_{i_j} + 2$. Daher ist

$$n - 1 \geq \ell_{i_h} \geq \ell_{i_1} + 2(h-1) \geq 1 + 2(h-1) = 2h - 1,$$

was $h \leq n/2$ impliziert. ■

Man beachte, dass der Beweis auch bei Netzwerken mit reellen Kapazitäten seine Gültigkeit behält.

3.3 Der Algorithmus von Dinitz

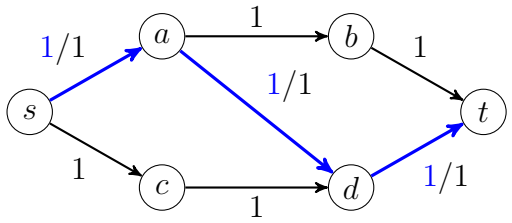
Man kann zeigen, dass sich in jedem Netzwerk ein maximaler Fluss durch Addition von höchstens m Zunahmepfaden konstruieren lässt (siehe Übungen). Es ist nicht bekannt, ob sich solche Pfade in Zeit $O(n+m)$ bestimmen lassen. Wenn ja, würde dies auf eine Gesamtlaufzeit von $O(n+m^2)$ führen. Für dichte Netzwerke (d.h. $m = \Theta(n^2)$) hat der Algorithmus von Dinitz die gleiche Laufzeit $O(n^2 m) = O(n^4)$ und die verbesserte Version ist mit $O(n^3)$ in diesem Fall sogar noch schneller.

Die Analyse der Laufzeit des Edmonds-Karp-Algorithmus beruht auf der Tatsache, dass der Fluss f_{P_i} durch den Zunahmepfad P_i, der in jedem Schleifendurchlauf auf den aktuellen Fluss f_i addiert wird, auf *mindestens einem* kürzesten Pfad im Restnetzwerk N_{f_i} eine Kante sättigt. Dies hat zur Folge, dass nicht mehr als $nm/2$ Zunahmepfade P_i benötigt werden, um einen maximalen Fluss zu erhalten.

Dagegen addiert der Algorithmus von Dinitz in jedem Schleifendurchlauf auf den aktuellen Fluss f_i einen Fluss g_i, der auf *jedem* kürzesten Pfad im Restnetzwerk N_{f_i} mindestens eine Kante sättigt. Wir werden sehen, dass maximal $n-1$ solche Flüsse g_i benötigt werden.

Definition 3.14. *Ein Fluss g in einem Netzwerk $N = (V,E,s,t,c)$* ***sättigt*** *eine Kante $e \in E$, falls $g(e) = c(e)$ ist. g heißt* ***blockierend****, falls g mindestens eine Kante auf jedem Pfad P von s nach t sättigt.*

Nach dem Min-Cut-Max-Flow-Theorem gibt es zu jedem maximalen Fluss f einen s-t-Schnitt S, so dass alle Kanten in $E(S)$ gesättigt sind. Da jeder Pfad von s nach t mindestens eine Kante in $E(S)$ enthalten muss, ist jeder maximale Fluss auch blockierend. Für die Umkehrung gibt es jedoch einfache Gegenbeispiele, wie etwa



Ein blockierender Fluss muss also nicht unbedingt maximal sein. Tatsächlich ist g genau dann ein blockierender Fluss in N, wenn es im Restnetzwerk N_g keinen Zunahmepfad gibt, der nur aus *Vorwärtskanten* $e \in E$ mit $g(e) < c(e)$ besteht.

Der Algorithmus von Dinitz berechnet anstelle eines kürzesten Zunahmepfades P im aktuellen Restnetzwerk N_f einen blockierenden Fluss g im Schichtnetzwerk N'_f. Dieses enthält nur diejenigen Kanten von N_f, die auf einem kürzesten Pfad mit Startknoten s liegen. Zudem werden aus N'_f alle Knoten $u \neq t$ entfernt, die einen Abstand $d(s,u) \geq d(s,t)$ in N_f haben. Der Name rührt daher, dass jeder Knoten in N'_f einer Schicht S_j zugeordnet wird.

Definition 3.15. *Sei $N = (V, E, s, t, c)$ ein Netzwerk. Das zugeordnete* ***Schichtnetzwerk*** *ist $N' = (V', E', s, t, c')$ mit der Knotenmenge $V' = S_0 \cup \cdots \cup S_\ell$ und der Kantenmenge*

$$E' = \bigcup_{j=1}^{\ell} \{(u,v) \in E \mid u \in S_{j-1} \wedge v \in S_j\}$$

sowie der Kapazitätsfunktion

$$c'(e) = \begin{cases} c(e), & e \in E', \\ 0, & \textit{sonst}, \end{cases}$$

wobei $\ell = 1 + \max\{d(s,u) < d(s,t) \mid u \in V\}$ und

$$S_j = \begin{cases} \{u \in V \mid d(s,u) = j\}, & 0 \leq j \leq \ell - 1, \\ \{t\}, & j = \ell \end{cases}$$

ist und $d(x,y)$ die Länge eines kürzesten Pfades von x nach y in N bezeichnet.

Der Algorithmus von Dinitz arbeitet wie folgt.

Algorithmus `Dinitz`(N), $N = (V, E, s, t, c)$

```
1  for all (u,v) ∈ E ∪ E^R do
2    f(u,v) := 0
3  repeat
4    S := schichtnetzwerk(N, f)
5    if S ≠ ⊥ then f := f + blockfluss(S)
6  until S = ⊥
```

Das zum Restnetzwerk $N_f = (V, E_f, s, t, c_f)$ gehörige Schichtnetzwerk $S = N'_f = (V', E'_f, s, t, c'_f)$ wird von der Prozedur `schichtnetzwerk`(N, f) in Zeit $O(n+m)$ berechnet. Für die Berechnung eines blockierenden Flusses g im Schichtnetzwerk N'_f werden wir zwei Algorithmen angeben: Eine Prozedur `blockfluss1`, deren Laufzeit durch $O(nm)$ und eine Prozedur `blockfluss2`, deren Laufzeit durch $O(n^2)$ beschränkt ist.

Wir beschreiben zuerst die Prozedur `schichtnetzwerk`. Diese Prozedur führt in N_f eine modifizierte Breitensuche mit Startknoten s durch und speichert dabei in der Menge E' nicht nur alle Baumkanten, sondern zusätzlich alle Querkanten (u,v), die auf einem kürzesten Weg von s zu v liegen. Die Suche bricht ab, sobald t am Kopf der Schlange erscheint oder alle von s aus erreichbaren Knoten abgearbeitet sind.

Falls t erreicht wurde, werden außer der Senke t alle Knoten u, die in N_f einen Abstand $d(s,u) < d(s,t)$ von der Quelle s haben, in der Menge V' zusammengefasst. Zudem werden alle Kanten aus E' wieder entfernt, die nicht zwischen zwei Knoten aus V' verlaufen.

Wurde dagegen t nicht erreicht, so existiert in N_f (und damit in N'_f) kein (blockierender) Fluss g mit $|g| > 0$ und somit auch kein Zunahmepfad in N_f, d.h. f ist maximal.

Prozedur `schichtnetzwerk`(N, f)

```
1  E' := ∅
2  for all v ∈ V do niv(v) := n
3  niv(s) := 0; Q := (s)
4  while Q ≠ () ∧ head(Q) ≠ t do
5    u := dequeue(Q)
6    for all e = (u, v) ∈ E ∪ E^R do
7      if c(e) − f(e) > 0 ∧ niv(v) > niv(u) then
8        E' := E' ∪ {e}
9        c'(e) := c(e) − f(e)
10       if niv(v) > niv(u) + 1 then
11         niv(v) := niv(u) + 1
12         enqueue(Q, v)
13 if head(Q) = t then
14   V' := {v ∈ V | niv(v) < niv(t)} ∪ {t}
15   E' := E' ∩ (V' × V')
16   return (V', E', s, t, c')
17 else return ⊥
```

Die Laufzeitschranke $O(n+m)$ folgt aus der Tatsache, dass jede Kante in $E \cup E^R$ höchstens einmal besucht wird und jeder Besuch mit einem konstanten Zeitaufwand verbunden ist.

Nun kommen wir zur Beschreibung der Prozedur `blockfluss1`. Beginnend mit dem Nullfluss g bestimmt diese in der repeat-Schleife mittels Tiefensuche einen s-t-Pfad P im Schichtnetzwerk N'_f, addiert den Fluss $(f+g)_P$ zum aktuellen Fluss g hinzu, aktualisiert die Restkapazitäten aller Kanten e auf dem Pfad P und entfernt aus E' die von g gesättigten Kanten. Der Pfad P lässt sich hierbei direkt aus dem Inhalt des Kellers K rekonstruieren, weshalb er ***K-Pfad*** genannt wird. Man beachte, dass die Kapazitäten der Kanten e auf dem Pfad P nur in Vorwärtsrichtung verkleinert, aber anders als bei Ford-Fulkerson und Edmonds-Karp nicht auch sofort in Rückwärtsrichtung angepasst werden. Dies geschieht erst, nachdem g zu einem blockierenden Fluss angewachsen ist.

Falls die Tiefensuche in einem Knoten $u \neq s$ in einer Sackgasse endet (weil E' keine von u aus weiterführenden Kanten enthält), wird die zuletzt besuchte Kante (u', u) ebenfalls aus E' entfernt und die Tiefensuche vom Startpunkt u' dieser Kante fortgesetzt (back tracking). Die Prozedur `blockfluss1` bricht ab, sobald alle Kanten mit Startknoten s aus E' entfernt wurden und somit in (V', E') keine Pfade mehr von s nach t existieren (d.h. g ist ein blockierender Fluss in S).

Prozedur `blockfluss1`(S), $S = (V', E', s, t, c')$

```
1  for all e ∈ E' ∪ E'^R do g(e) := 0
2  u := s; K := (s)
3  done := false
4  repeat
5    if ∃ e = (u, v) ∈ E' then
6      push(K, v)
7      c''(e) := c'(e) − g(e)
8      u := v
9    elsif u = t then
10     P := K-Pfad von s nach t
11     c'_g(P) := min{c''(e) | e ∈ P}
12     for all e ∈ P do
13       if c''(e) = c'_g(P) then E' := E' \ {e}
14       g(e) := g(e) + c'_g(P); g(e^R) := −g(e)
15     u := s; K := (s)
16   elsif u ≠ s then
17     pop(K)
18     u' := top(K)
19     E' := E' \ {(u', u)}
20     u := u'
21   else done := true
22 until done
23 return g
```

Die Laufzeitschranke $O(nm)$ folgt aus der Tatsache, dass sich die Anzahl der aus E' entfernten Kanten nach spätestens n Schleifendurchläufen um 1 erhöht.

Satz 3.16. *Der Algorithmus von Dinitz durchläuft die while-Schleife höchstens $(n-1)$-mal.*

Beweis. Sei f_1 der Nullfluss in N und seien $g_1, \ldots, g_k$ die blockierenden Flüsse, die der Dinitz-Algorithmus der Reihe nach berechnet, d.h. $f_{i+1} = f_i + g_i$. Zudem sei $d_i(u,v)$ die minimale Länge eines Pfades von u nach v im Restnetzwerk N_{f_i} und sei $l_i = d_i(s,t)$. Wir zeigen, dass $l_i < l_{i+1}$ ist. Da $l_1 \geq 1$ und $l_k \leq n-1$ ist, folgt $k \leq n-1$.

Hierzu beweisen wir zunächst, dass für jeden kürzesten Pfad $P = (u_0, \ldots, u_l)$ von $u_0 = s$ nach $u_l = u$ in $N_{f_{i+1}}$ (d.h. $d_{i+1}(s,u_h) = h$) für $h = 1, \ldots, l$ folgende (Un)gleichungen gelten:

$$d_i(s,u_h) \leq d_i(s,u_{h-1}) + 1, \text{ falls } (u_{h-1},u_h) \in E_{f_i} \tag{3.1}$$
$$d_i(s,u_h) = d_i(s,u_{h-1}) - 1, \text{ falls } (u_{h-1},u_h) \notin E_{f_i} \tag{3.2}$$

Es ist klar, dass (3.1) gilt, falls die Kante $e = (u_{h-1}, u_h)$ auch in N_{f_i} enthalten ist. Andernfalls ist $f_{i+1}(e) \neq f_i(e)$, d.h. $g_i(e) \neq 0$. Da e nicht in N_{f_i} und somit auch nicht in N'_{f_i} enthalten ist, muss e^R in N'_{f_i} sein. Da N'_{f_i} nur Kanten auf kürzesten Pfaden mit Startknoten s enthält, folgt $d_i(s,u_{h-1}) = d_i(s,u_h) + 1$, was (3.2) impliziert. Aus (3.1 + 3.2) folgt

$$d_i(s,u_l) \leq d_i(s,u_{l-1}) + 1 \leq \cdots \leq d_i(s,s) + l = l = d_{i+1}(s,u_l)$$

und wir haben folgende Behauptung bewiesen.

Behauptung 3.17. *Für jeden Knoten $u \in V$ gilt $d_i(s,u) \leq d_{i+1}(s,u)$.*

Um nun zu zeigen, dass $l_i < l_{i+1}$ für $i = 1, \ldots, k-1$ gilt, sei $P = (u_0, u_1, \ldots, u_{l_{i+1}})$ ein kürzester Pfad von $s = u_0$ nach $t = u_{l_{i+1}}$ in $N_{f_{i+1}}$ (und somit auch in $N'_{f_{i+1}}$). Mit Behauptung 3.17 folgt, dass $d_i(s,u_h) \leq d_{i+1}(s,u_h) = h$ für $h = 0, \ldots, l_{i+1}$ ist. Wir unterscheiden zwei Fälle.

- Wenn alle Knoten u_h in N'_{f_i} enthalten sind, muss ein h mit $d_i(s,u_h) \leq d_i(s,u_{h-1})$ existieren. Würde nämlich $d_i(s,u_h) > d_i(s,u_{h-1})$ gelten, so wären die Kanten (u_{h-1},u_h) für $h = 1, \ldots, h$ wegen (3.2) in N_{f_i} enthalten und somit $d_i(s,u_h) = d_i(s,u_{h-1}) + 1$. Dies hätte zur Folge, dass P ein kürzester Pfad von s nach t in N_{f_i} und somit ein s-t-Pfad in N'_{f_i} wäre, der von g_i nicht blockiert wird. Da aber g_i blockierend ist, muss also ein h mit $d_i(s,u_h) \leq d_i(s,u_{h-1})$ existieren und es folgt unter Verwendung von (3.1) + 3.2):
$$l_i = d_i(s,t) \leq d_i(s,u_h) + l_{i+1} - h \leq \underbrace{d_i(s,u_{h-1})}_{\leq d_{i+1}(s,u_{h-1}) = h-1} + l_{i+1} - h < l_{i+1}$$
- Falls mindestens ein Knoten u_h nicht in N'_{f_i} enthalten ist, sei u_h der erste solche Knoten auf P. Da $u_h \neq t$ ist, folgt $d_{i+1}(s,u_h) = h < l_{i+1}$. Zudem liegt die Kante $e = (u_{h-1},u_h)$ nicht nur in $N_{f_{i+1}}$, sondern wegen $f_{i+1}(e) = f_i(e)$ (da weder e noch e^R zu N'_{f_i} gehören) auch in N_{f_i}. Da somit u_{h-1} in N'_{f_i} und e in N_{f_i} ist, kann u_h nur aus dem Grund nicht zu N'_{f_i} gehören, dass $d_i(s,u_h) = d_i(s,t)$ ist. Daher folgt unter Verwendung von (3.1 + 3.2) sowie Behauptung 3.17 auch in diesem Fall die Ungleichung $l_i < l_{i+1}$:
$$l_i = d_i(s,t) = d_i(s,u_h) \leq \underbrace{d_i(s,u_{h-1})}_{\leq d_{i+1}(s,u_{h-1})} + 1 = d_{i+1}(s,u_h) < l_{i+1} \quad \blacksquare$$

Korollar 3.18. *Der Algorithmus von Dinitz berechnet bei Verwendung der Prozedur* `blockfluss1` *einen maximalen Fluss in Zeit $O(n^2m)$.*

Die Prozedur `blockfluss2` benötigt nur Zeit $O(n^2)$, um einen blockierenden Fluss g im Schichtnetzwerk N'_f zu berechnen, was auf eine Gesamtlaufzeit des Algorithmus von Dinitz von $O(n^3)$ führt. Zu ihrer Beschreibung benötigen wir folgende Notation.

Definition 3.19. *Sei $N = (V,E,s,t,c)$ ein Netzwerk.*

a) Der ***Durchsatz eines Knotens $u \in V$*** *ist*

$$D(u) = \begin{cases} c^+(u), & u = s, \\ c^-(u), & u = t, \\ \min\{c^+(u), c^-(u)\}, & \textit{sonst,} \end{cases}$$

wobei $c^+(u) = \sum_{v \in V} c(u,v)$ die ***Ausgangskapazität*** *und $c^-(u) = \sum_{v \in V} c(v,u)$ die* ***Eingangskapazität von u*** *ist.*

b) Ein Fluss g in N ***sättigt einen Knoten $u \in V$****, falls*

- *$u = s$ ist und g alle Kanten $(s,v) \in E$ mit Startknoten s sättigt, oder*
- *$u = t$ ist und g alle Kanten $(v,t) \in E$ mit Zielknoten t sättigt, oder*
- *$u \in V - \{s,t\}$ ist und g alle Kanten $(u,v) \in E$ mit Startknoten u oder alle Kanten $(v,u) \in E$ mit Zielknoten u sättigt.*

Die Korrektheit der Prozedur `blockfluss2` basiert auf folgender Proposition.

Proposition 3.20. *Wenn ein Fluss g in einem Netzwerk N auf jedem s-t-Pfad P mindestens einen Knoten u sättigt, dann ist g blockierend.*

Beweis. Falls g mindestens einen Knoten u auf dem s-t-Pfad P sättigt, dann sättigt g auch mindestens eine Kante auf dem Pfad P. ■

Beginnend mit dem trivialen Fluss $g = 0$ berechnet die Prozedur `blockfluss2` für jeden Knoten u den Durchsatz $D(u)$ im Schichtnetzwerk N'_f und wählt in jedem Durchlauf der repeat-Schleife einen Knoten u mit minimalem Durchsatz $D(u)$. Dann benutzt sie die Prozeduren `propagierevor` und `propagiererück`, um den aktuellen Fluss g um den Wert $D(u)$ zu erhöhen und die Restkapazitäten der betroffenen Kanten sowie die Durchsatzwerte $D(v)$ der betroffenen Knoten entsprechend zu aktualisieren.

Anschließend werden alle gesättigten Knoten aus V' und alle gesättigten Kanten aus E' entfernt. Hierzu werden in der Menge B alle Knoten gespeichert, deren Durchsatz durch die Erhöhungen des Flusses g auf 0 gesunken ist.

Prozedur `blockfluss2`(S), $S = (V', E', s, t, c')$

```
1  for all e ∈ E' ∪ E'^R do g(e) := 0
2  for all u ∈ V' do
3    D⁺(u) := Σ_{(u,v)∈E'} c'(u,v)
4    D⁻(u) := Σ_{(v,u)∈E'} c'(v,u)
5  repeat
6    for all u ∈ V' \ {s,t} do D(u) := min{D⁻(u), D⁺(u)}
7    D(s) := D⁺(s)
8    D(t) := D⁻(t)
9    wähle u ∈ V' mit D(u) minimal
10   B := {u}
11   propagierevor(u)
12   propagiererück(u)
13   while ∃v ∈ B \ {s,t} do
14     B := B \ {v}; V' := V' \ {v}
15     for all e = (v,w) ∈ E' do
16       D⁻(w) := D⁻(w) − c'(v,w)
17       if D⁻(w) = 0 then B := B ∪ {w}
18       E' := E' \ {e}
19     for all e = (w,v) ∈ E' do
20       D⁺(w) := D⁺(w) − c'(w,v)
21       if D⁺(w) = 0 then B := B ∪ {w}
22       E' := E' \ {e}
23 until u ∈ {s,t}
24 return g
```

Da in jedem Durchlauf der repeat-Schleife mindestens ein Knoten u gesättigt und aus V' entfernt wird, wird nach höchstens $n-1$ Iterationen einer der beiden Knoten s oder t als Knoten u mit minimalem

Durchsatz $D(u)$ gewählt und die repeat-Schleife verlassen. Da nach Beendigung des letzten Durchlaufs der Durchsatz von s oder von t gleich 0 ist, wird einer dieser beiden Knoten zu diesem Zeitpunkt von g gesättigt. Nach Proposition 3.20 ist somit g ein blockierender Fluss.

Die Prozeduren `propagierevor` und `propagiererück` propagieren den Fluss durch u in Vorwärtsrichtung hin zu t bzw. in Rückwärtsrichtung hin zu s. Dies geschieht in Form einer Breitensuche mit Startknoten u unter Benutzung der Kanten in E' bzw. E'^R. Da der Durchsatz $D(u)$ von u unter allen Knoten minimal ist, ist sichergestellt, dass der Durchsatz $D(v)$ jedes Knotens v ausreicht, um den für ihn ermittelten Zusatzfluss in Höhe von $z(v)$ weiterzuleiten.

Prozedur `propagierevor`(u)

```
1  for all v ∈ V' do z(v) := 0
2  z(u) := D(u)
3  Q := (u); R := {u}
4  while Q ≠ () do
5    v := dequeue(Q)
6    while z(v) ≠ 0 ∧ ∃e = (v, w) ∈ E' do
7      if w ∉ R then enqueue(Q, w)
8      R := R ∪ {w}
9      m := min{z(v), c'(e)}; z(v) := z(v) − m; z(w) := z(w) + m
10     aktualisierekante(e, m)
```

Prozedur `aktualisierekante`(e, m), $e = (v, w)$

```
1  g(e) := g(e) + m
2  c'(e) := c'(e) − m
3  if c'(e) = 0 then E' := E' \ {e}
4  D+(v) := D+(v) − m
5  if D+(v) = 0 then B := B ∪ {v}
6  D−(w) := D−(w) − m
7  if D−(w) = 0 then B := B ∪ {w}
```

Die Prozedur `propagiererück` unterscheidet sich von der Prozedur `propagierevor` nur dadurch, dass in Zeile 6 die Bedingung $\exists e = (v, w) \in E'$ durch die Bedingung $\exists e = (w, v) \in E'$ ersetzt wird.

Da die repeat-Schleife von `blockfluss2` maximal $(n-1)$-mal durchlaufen wird, werden die Prozeduren `propagierevor` und `propagiererück` höchstens $(n-1)$-mal aufgerufen. Sei a die Gesamtzahl der Durchläufe der inneren while-Schleife von `propagierevor`, summiert über alle Aufrufe. Da in jedem Durchlauf eine Kante aus E' entfernt wird (falls $m = c'(v, u)$ ist) oder der zu propagierende Fluss $z(v)$ durch einen Knoten v auf 0 sinkt (falls $m = z(v)$ ist), was pro Knoten und pro Aufruf höchstens einmal vorkommt, ist $a \leq n^2 + m$. Der gesamte Zeitaufwand ist daher $O(n^2 + m)$ innerhalb der beiden while-Schleifen und $O(n^2)$ außerhalb. Die gleichen Schranken gelten für `propagiererück`.

Eine ähnliche Überlegung zeigt, dass die while-Schleife von `blockfluss2` einen Gesamtaufwand von $O(n + m)$ hat. Folglich ist die Laufzeit von `blockfluss2` $O(n^2)$.

Korollar 3.21. *Der Algorithmus von Dinitz berechnet bei Verwendung der Prozedur* `blockfluss2` *einen maximalen Fluss in Zeit* $O(n^3)$.