Vorlesungsskript

Graphalgorithmen

Sommersemester 2017

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

13. Juli 2017

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* $N_G(v) = \{u \in V \mid \{u, v\} \in E\}$.

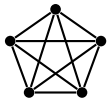
b) Der ***Grad*** *von* v *ist* $\deg_G(v) = |N_G(v)|$.

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

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 $i = 0, \ldots, \ell - 1$, der jede Kante $e \in E$ höchstens einmal durchläuft. 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) Ein Graph $G = (V, E)$ heißt **zusammenhängend**, falls es für alle Paare $\{u, v\} \in \binom{V}{2}$ einen u-v-Weg gibt. G heißt **k-zusammenhängend**, $1 < k < n$, falls G nach Entfernen von beliebigen $l \leq \min\{n - 1, k - 1\}$ Knoten immer noch zusammenhängend ist.*

*h) Ein **Zyklus** ist ein u-v-Weg der Länge $\ell \geq 2$ mit $u = v$.*

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

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

*k) Ein **Kreis** ist ein Zyklus $v_0, v_1 \ldots, v_{\ell-1}, v_0$ der Länge $\ell \geq 3$, für den $v_0, v_1, \ldots, v_{\ell-1}$ paarweise verschieden sind.*

*l) Ein Graph $G = (V, E)$ heißt **kreisfrei**, **azyklisch** oder **Wald**, falls er keinen Kreis enthält.*

*m) Ein **Baum** ist ein zusammenhängender Wald.*

*n) Jeder Knoten $u \in V$ vom Grad $\deg(u) \leq 1$ heißt **Blatt** und die übrigen Knoten (vom Grad ≥ 2) heißen **innere Knoten**.*

Es ist leicht zu sehen, dass die Relation

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

eine Äquivalenzrelation ist. Die durch die Äquivalenzklassen von Z induzierten Teilgraphen heißen die **Zusammenhangskomponenten** (engl. *connected components*) oder einfach Komponenten von G.

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 $i = 0, \ldots, \ell - 1$, der jede Kante $e \in E$ höchstens einmal durchläuft.*

*f) Ein **(gerichteter) Zyklus** ist ein gerichteter u-v-Weg der Länge $\ell \geq 1$ 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 $v_0, v_1 \ldots, v_{\ell-1}, v_0$ der Länge $\ell \geq 1$, für den $v_0, v_1, \ldots, v_{\ell-1}$ 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.

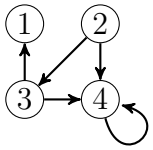
Die ***Adjazenzmatrix*** eines Graphen bzw. Digraphen $G = (V, E)$ mit (geordneter) Knotenmenge $V = \{v_1, \ldots, v_n\}$ ist die $(n \times n)$-Matrix $A = (a_{ij})$ mit den Einträgen

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

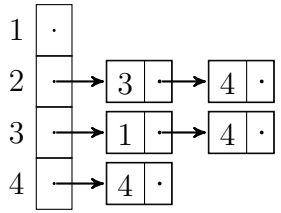
Für ungerichtete Graphen ist die Adjazenzmatrix symmetrisch mit $a_{ii} = 0$ für $i = 1, \ldots, n$.

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, betrachte 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}
```

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 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 deutlich

schnelleren $\mathcal{O}(n^2)$ Algorithmus liefert, aber auch nicht ohne 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.*

◁

Um eine Antwort auf die Frage zu finden, ob auch K_5 und $K_{3,3}$ planar sind, betrachten wir die Gebiete von in die Ebene eingebetteten Graphen.

Durch die Kanten eines eingebetteten Graphen wird die Ebene in so genannte ***Gebiete*** unterteilt. 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 Anzahl der an ein Gebiet g grenzenden Kanten bezeichnen wir mit $d(g)$, wobei Kanten $\{u, v\}$, die nur an g und kein anderes Gebiet grenzen, doppelt gezählt werden.

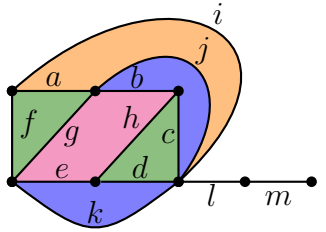
Der ***Rand*** *rand*(g) eines Gebiets g ist die (zirkuläre) Folge aller Kanten, die an g grenzen, wobei jede Kante so durchlaufen wird, dass g „in Fahrtrichtung links“ liegt bzw. bei Erreichen eines Knotens über eine Kante e, u über die im Uhrzeigersinn nächste Kante e' wieder verlassen wird. Auf diese 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

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

Führen zwei Einbettungen von G in die Ebene auf dieselbe Randmenge R, so werden sie als äquivalent angesehen. 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***. 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.

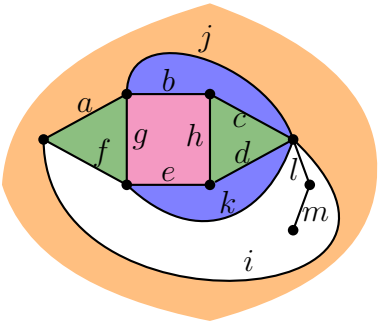
Beispiel 2.9. *Die beiden nebenstehenden Einbettungen eines Graphen $G = (V, E)$ in die Ebene haben jeweils 7 Gebiete und führen beide auf den ebenen Graphen $G' = (V, E, R)$ mit den 7 Rändern*



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

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 Kantenfolgen kann man R und π 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. *K_5 ist nicht planar.*

Beweis. Wegen $n = 5$, also $3n - 6 = 9$, und wegen $m = \binom{5}{2} = 10$ gilt $m \not\leq 3n - 6$. ■

Korollar 2.13. *$K_{3,3}$ ist nicht planar.*

Beweis. Wegen $n = 6$, also $2n - 4 = 8$, und wegen $m = 3 \cdot 3 = 9$ gilt $m \not\leq 2n - 4$. ■

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

Lemma 2.14. *Jeder planare Graph hat einen Minimalgrad $\delta(G) \leq 5$.*

Beweis. Für $n \leq 6$ ist die Behauptung klar. Für $n > 6$ impliziert die Annahme $\delta(G) \geq 6$ 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 2.15. *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*

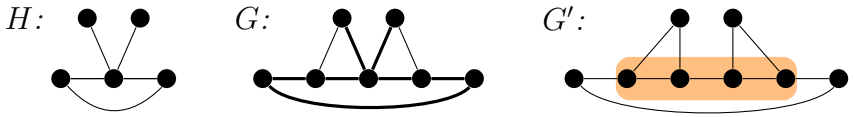
$$E' = \{e \in E \mid v \notin e\} \cup \{\{u, v'\} \mid \{v, v'\} \in E - \{u, v\}\}.$$

  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, so sagen wir auch, G_{uv} entsteht aus G durch* ***Überbrückung*** *des Knotens v.*

- *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 H aus einer isomorphen Kopie von G durch wiederholte Überbrückungen 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 2.16. *Betrachte folgende Graphen:*



Offensichtlich ist G keine Unterteilung von H. Entfernen wir jedoch die beiden dünnen Kanten aus G, so ist der resultierende Teilgraph eine Unterteilung von H, d.h. H ist ein topologischer Minor von G. Dagegen ist kein Teilgraph von G' isomorph zu einer Unterteilung von H und somit ist H kein topologischer Minor von G'. Wenn wir aber die drei umrandeten Kanten von G' kontrahieren, entsteht ein zu H isomorpher Graph, d.h. H ist ein Minor von G'. ◁

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 Knoten,
- Kontraktion einer Kante (bzw. Überbrückung eines Knoten).

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 2.17 (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.

Der resultierende Graph G' ist 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 2.18. *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 Zusammenhangskomponente 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 jeder Kreis in einem planaren Graphen

höchstens zwei paarweise inkompatible Brücken haben kann.

Satz 2.19 (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 Tatsache, dass die Klasse $\mathcal{K}$ der planaren Graphen unter (topologischer) Minorenbildung abgeschlossen ist (d.h. wenn $G \in \mathcal{K}$ und H ein Minor von G ist, dann folgt $H \in \mathcal{K}$).

Die Implikation von *ii*) nach *i*) zeigen wir durch Kontraposition. Sei also $G = (V, E)$ nicht planar. Dann hat G einen 3-zusammenhängenden nicht planaren topologischen Minor $G' = (V', E')$, so dass $G' - e'$ für jede Kante $e' \in E'$ planar ist (siehe Übungen). Wir entfernen eine beliebige Kante $e_0 = \{a_0, b_0\}$ aus G'. Da G' mindestens 5 Knoten hat, ist $G' - e_0$ 2-zusammenhängend. Daher gibt es in $G' - e_0$ einen Kreis K durch die beiden Knoten a_0 und b_0. Wir wählen K zusammen mit einer ebenen Realisierung H' von $G' - e_0$ so, dass K möglichst viele Gebiete in H' einschließt.

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

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.

Behauptung 2.20. *Jede äußere Brücke B besteht aus einer Kante $\{u, v\}$, die zwei Knoten $u \in K(a_0, b_0)$ und $v \in K(b_0, a_0)$ verbindet.*

Zum Beweis der Behauptung nehmen wir an, dass B mindestens einen Kontaktpunkt in $\{a_0, b_0\}$ oder mehr als 2 Kontaktpunkte hat. Dann liegen mindestens zwei dieser Punkte auf $K[a_0, b_0]$ oder auf $K[b_0, a_0]$.

Folglich kann 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.

Im Graphen G' hat K außer den Brücken in $G' - e_0$ noch zusätzlich die Kante e_0 als Brücke. Nun wählen wir 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 B 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 G' 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

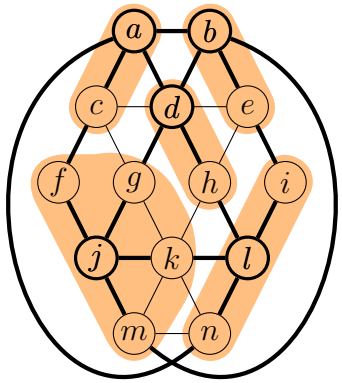
vier Punkte zu B gehören. Sei P_0 ein a_0-b_0-Pfad in B und sei P_1 ein a_1-b_1-Pfad in B. Sei u der erste Knoten auf P_0, der auch auf P_1 liegt und sei v der letzte solche Knoten.

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 G' 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 G' 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.*

Definition 2.23. *Sei $\leq$ eine binäre Relation auf einer Menge A.*

a) $(A, \lesssim)$ heißt ***Quasiordnung****, wenn $\lesssim$ reflexiv und transitiv auf A ist.*

b) $(A, \lesssim)$ heißt ***Wohlquasiordnung****, wenn es zudem zu jeder unendlichen Folge $a_1, a_2, \ldots$ von Elementen aus A Indizes $i < j$ mit $a_i \lesssim a_j$ gibt.*

Beispiele für Quasiordnungen sind $a \lesssim b :\Leftrightarrow |a| \leq |b|$ auf den ganzen oder komplexen Zahlen. Im ersten Fall handelt es sich um eine Wohlquasiordnung, im zweiten nicht, da zum Beispiel die Folge $a_i = (i+1)/i$ eine unendliche ***absteigende Kette*** bildet (d.h. $a_{i+1} \lesssim a_i$ und $a_i \not\lesssim a_{i+1}$ für alle $i \geq 1$). $(\mathbb{N}, \leq)$ ist eine Wohlquasiordnung (sogar eine lineare Wohlordnung, da auch antisymmetrisch und konnex). Die Teilbarkeitsrelation auf den natürlichen Zahlen ist dagegen keine Wohlquasiordnung, da mit der Folge der Primzahlen eine unendliche ***Antikette*** existiert (d.h. die Glieder der Folge sind paarweise unvergleichbar: es gilt $a_i \not\lesssim a_j$ und $a_j \not\lesssim a_i$ für alle $i > j \geq 1$).

Es ist leicht zu sehen, dass die Minorenrelation auf der Menge aller endlichen ungerichteten Graphen keine unendlichen absteigenden Ketten hat. Gemäß folgender Proposition ist sie daher genau dann eine Wohlquasiordnung, wenn es auch keine unendlichen Antiketten gibt.

Proposition 2.24. *Eine Quasiordnung $(A, \lesssim)$ ist genau dann eine Wohlquasiordnung, wenn es in $(A, \lesssim)$ weder unendliche absteigende Ketten noch unendliche Antiketten gibt.*

Beweis. Siehe Übungen. ■

Satz 2.25 (Satz von Robertson und Seymour, 1983-2004)**.** *Die Minorenrelation bildet auf der Menge aller endlichen ungerichteten Graphen eine Wohlquasiordnung.*

Korollar 2.26. *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 $\mathcal{K}$.*

Für den Beweis des Korollars betrachten wir die komplementäre Klasse $\overline{\mathcal{K}}$ aller endlichen Graphen, die nicht zu $\mathcal{K}$ gehören, und zeigen, dass $\overline{\mathcal{K}}$ bis auf Isomorphie nur endlich viele minimale Elemente hat. Sei $\mathcal{M}$ die Menge aller minimalen Elemente von $\overline{\mathcal{K}}$ und entstehe $\mathcal{H}$ aus $\mathcal{M}$, indem wir aus jeder Isomorphieklasse einen Graphen auswählen. Dann hat jeder Graph $G \in \overline{\mathcal{K}}$ einen Minor in $\mathcal{H}$ und umgekehrt gehört jeder Graph G, der einen Minor in $\mathcal{H}$ hat, zu $\overline{\mathcal{K}}$, d.h.

$$\overline{\mathcal{K}} = \{G \mid \exists H \in \mathcal{H} : H \text{ ist ein Minor von } G\}.$$

Da zudem $\mathcal{H}$ eine Antikette bildet, muss $\mathcal{H}$ nach Satz 2.25 endlich sein, womit Korollar 2.26 bewiesen 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 2.27 (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 2.28. *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 2.25 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 2.29. *Sei $G = (V, E)$ ein Graph.*

a) G heißt ***chordal*** *oder* ***trianguliert****, wenn jeder Kreis $K = u_1, \ldots, u_l, u_1$ der Länge $l \geq 4$ in G mindestens eine Sehne hat.*

b) Eine Menge $S \subseteq V$ heißt ***Separator*** *von G, wenn $G - S$ mehr Komponenten als G hat. S heißt* ***x-y-Separator****, wenn die beiden Knoten x und y in verschiedenen Komponenten von $G - S$ liegen.*

Ein Graph 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 2.30. *Für einen Graphen G sind folgende Aussagen äquivalent.*

(i) G ist chordal.

(ii) Jeder inklusionsminimale Separator von 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 Separator S hat, der zwei nicht adjazente Knoten x und y enthält. Seien $G[V_1]$ und $G[V_2]$ zwei Komponenten in $G - S$, die durch S getrennt werden. Da S minimal

ist, sind die beiden Knoten x und y sowohl mit $G[V_1]$ als auch mit $G[V_2]$ verbunden. Betrachte die beiden Teilgraphen $G_i = G[V_i \cup \{x, y\}]$ und wähle jeweils einen kürzesten x-y-Pfad P_i in G_i. Da diese eine Länge ≥ 2 haben, 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 2.31. *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 2.32. *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 Baum mindestens 2 nicht adjazente Blätter hat (abgesehen von K_1 und K_2).

Lemma 2.33. *Jeder nicht vollständige chordale Graph besitzt mindestens 2 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. Falls x_1 und x_2 in verschiedenen Komponenten von G liegen, sei $S = \emptyset$, andernfalls sei S ein minimaler x_1-x_2-Separator. Im zweiten Fall ist S nach Lemma 2.30 eine Clique in G. Seien $G[V_1]$ und $G[V_2]$ die beiden Komponenten von $G - S$ mit $x_i \in V_i$.

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 x_i simplizial in G_i, und da x_i keine Nachbarn außerhalb von $V_i \cup S$ hat, ist x_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 y_i keine Nachbarn außerhalb von $V_i \cup S$ und somit ist y_i auch simplizial in G. ■

Satz 2.34. *Ein Graph ist genau dann chordal, wenn er eine PEO hat.*

Beweis. Falls G chordal ist, lässt sich eine PEO gemäß Lemma 2.33 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 2.35. *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 2.34 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 2.33 nicht chordal. Daher gibt es nach Lemma 2.30 in G_i zwei nicht adjazente Knoten x und y, so dass kein x-y-Separator eine Clique ist. Wie im Beweis von Lemma 2.30 beschrieben, lässt sich mithilfe eines minimalen Separators S, der keine Clique ist, ein induzierter 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 2.36. *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` effizient zu implementieren, benötigen wir einen möglichst effizienten Algorithmus zur Bestimmung einer PEO. Rose, Tarjan und Lueker haben hierfür 1976 einen Linearzeitalgorithmus angegeben, der auf *lexikographischer Breitensuche* (kurz LexBFS oder LBFS) basiert. Bevor wir auf diese Variante der Breitensuche näher eingehen, rekapitulieren wir an dieser Stelle nochmals kurz verschiedene Ansätze zum Durchsuchen von Graphen.

Der folgende Algorithmus `GraphSearch`(V, E) startet eine Suche in einem beliebigen Knoten 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.

Algorithmus `GraphSearch`(V, E)

```
1  R ← ∅ // Menge der erreichten Knoten
2  L ← () // Ausgabeliste
3  repeat
4    wähle u ∈ V \ R
5    R ← R ∪ {u}
6    A ← {u} // Menge der noch abzuarbeitenden Knoten
7    while A ≠ ∅ do
8      entferne u aus A
9      append(L, u)
10     A ← A ∪ (N(u) \ R)
11     R ← R ∪ N(u)
12 until R = V
13 return(L)
```

Der Algorithmus `GraphSearch`(V, E) findet in jedem Durchlauf der repeat-Schleife eine neue Zusammenhangskomponente des Eingabegraphen $G = (V, E)$. Dies bedeutet, dass alle Knoten, die zu einer Zusammenhangskomponente gehören, konsekutiv ausgegeben werden. Zudem ist jeder Knoten, der nicht als erster in seiner Zusammenhangskomponente ausgegeben wird, mit einem zuvor ausgegebenen

Knoten verbunden. Die folgende Definition fasst diese Eigenschaften der Ausgabeliste zusammen.

Definition 2.37. *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:*

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

Satz 2.38. *Für jeden Graphen $G = (V, E)$ gibt der Algorithmus* `GraphSearch`(V, E) *eine SO von G aus.*

Beweis. Siehe Übungen. ■

Realisieren wir die Menge der abzuarbeitenden 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 S zur Speicherung der noch abzuarbeitenden Knoten bewirkt, dass die Suche nach unerreichten Knoten mit einem Nachbarn eines Nachbars v des aktuellen Knotens u fortgesetzt wird, bevor die noch nicht erreichten übrigen Nachbarn von u besucht werden.

Algorithmus `DFS`(V, E)

```
1  R ← ∅ // Menge der erreichten Knoten
2  L ← () // Ausgabeliste
3  repeat
4    wähle u ∈ V \ R
5    R ← R ∪ {u}
6    append(L, u)
7    S ← (u) // Keller der abzuarbeitenden Knoten
8    while S ≠ () do
9      u ← top(S)
10     if ∃v ∈ N(u) \ R then
11       push(S, v)
12       append(L, v)
13       R ← R ∪ {v}
14     else
15       pop(S)
16 until R = V
17 return(L)
```

Definition 2.39. *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 2.40. *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) sogar 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
5    Q ← (u) // Warteschlange der abzuarb. Knoten
6    while Q ≠ () do
7      u ← dequeue(Q)
```

```
    append(L, u)
    for all v ∈ N(u)\R do enqueue(Q, v)
    R ← R ∪ N(u)
until R = V
return(L)
```

Definition 2.41. *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 2.42. *Für jeden Graphen $G = (V, E)$ gibt der Algorithmus* `BFS(V, E)` *eine BO von G aus.*

Beweis. Siehe Übungen. ■

Der Unterschied von LexBFS zur normalen Breitensuche besteht darin, dass die zulässigen Ausgabefolgen gegenüber der BFS weiter eingeschränkt werden. Hierzu wird die Menge der noch nicht abgearbeiteten Knoten in eine Folge von Teilmengen zerlegt, welche vom Algorithmus wiederholt verfeinert wird. Der Name von LexBFS rührt daher, dass die Knoten in einer Reihenfolge ausgegeben werden, die auch bei einer gewöhnlichen Breitensuche auftreten kann, bei dieser aber nicht garantiert ist. Bei einer Breitensuche werden die noch nicht besuchten Nachbarn des aktuellen Knotens in beliebiger Reihenfolge zur Warteschlange hinzugefügt und später auch wieder in dieser Reihenfolge entfernt. Dagegen werden bei einer LexBFS die Knoten in der Warteschlange nachträglich umsortiert, falls dies notwendig ist, um eine lexikalische Sortierung der Knoten zu erhalten (siehe Definition 2.43).

Algorithmus `LexBFS(V, E, u)`

```
L ← () // Ausgabeliste
Q ← (V) // Warteschlange von Knotenmengen
while Q ≠ () do
  u ← Dequeue(Q)
  append(L, u)
  Splitqueue(Q, N(u))
return(L)
```

Prozedur `Dequeue(Q)`

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

Prozedur `Splitqueue(Q, S)`

```
for T in Q with T ∩ S ∉ {∅, T} do
  ersetze (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 2.43. *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 also 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 *lexikalisch* (also wie im Lexikon) 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. In den Übungen wird gezeigt, dass man sogar eine *lexikographische* Ordnung auf den kompletten Zeilen von A erhält, falls man die Diagonaleinträge von A auf 1 setzt und die Knoten in jeder Menge der Warteschlange Q nach absteigendem Knotengrad in G sortiert.

Satz 2.44. *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 $a_{kl} = 0$ geben. Ansonsten wäre der Knoten u_l spätestens beim Besuch von u_j in eine Menge vor dem Knoten u_k sortiert worden und könnte daher nicht nach dem Knoten u_k ausgegeben werden. ■

Lemma 2.45. *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 $A[i_2, i_3] = 0$, wobei wir i_3 möglichst klein wählen.

Falls nun $A[i_2, i_3] = 1$ ist, haben wir einen induzierten Kreis $G[u_{i_0}, u_{i_1}, u_{i_2}, u_{i_3}]$ 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 zeitbeschränkt 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 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. Neben

Graphen treten hierbei auch ***Multigraphen*** $G = (V, E)$ auf, d.h. die Kantenmenge E ist eine Multimenge. In diesem Fall können 2 Kanten nicht nur einen, sondern sogar beide Endpunkte gemeinsam haben. Wie bei Graphen gehen wir aber davon aus, dass jede Kante 2 verschiedene Endpunkte hat, d.h. G ist schlingenfrei.

Eine Multimenge A lässt sich durch eine Funktion $v_A\colon A \to \mathbb{N}$ beschreiben, wobei $v_A(a)$ die Anzahl der Vorkommen von a in A angibt. Die Mächtigkeit von A ist $|A| = \sum_{a\in A} v_A(a)$. A ist Teilmenge einer Multimenge B, wenn $v_A(a) \leq v_B(a)$ für alle $a \in A$ gilt. Wie bei Mengen bezeichnen wir die Menge aller k-elementigen Teilmengen von B mit $\binom{B}{k}$.

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 $c\colon E \to \{1, \dots, k\}$ 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 je 2 Kanten, die einen gemeinsamen Endpunkt 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 $E_1, \dots, E_k$. Umgekehrt liefert jede Zerlegung von E in k disjunkte Matchings eine k-Kantenfärbung von G.

Ist G ein Multigraph, so können wir eine k-Kantenfärbung von G auch durch eine Funktion c beschreiben, die jeder Kante $e \in E$ eine Menge $c(e) \subseteq \{1, \dots, k\}$ von $|c(e)| = v_E(e)$ verschiedenen Farben zuordnet, so dass $c(e) \cap c(e') = \emptyset$ für alle $e \neq e' \in E$ 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 $L(G) = (E, E')$ mit*

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

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

Ist G ein Multigraph, so ersetzen wir die Multimenge E in $L(G)$ durch eine Menge derselben Mächtigkeit, die für jede Kante $e \in E$ $v_E(e)$ verschiedene Kopien von e enthält. Die folgenden Beziehungen zwischen einem Graphen G und $L(G)$ lassen sich leicht verifizieren.

Proposition 2.49. *Sei $G' = L(G)$ der Line-Graph eines Graphen G. Dann gilt*

(i) $n(G') = m(G)$,
(ii) $\chi(G') = \chi'(G)$,
(iii) $\alpha(G') = \mu(G)$,
(iv) $\omega(G') \geq \Delta(G)$,
(v) $\Delta(G') = \max_{\{u,v\}\in E} \deg_G(u) + \deg_G(v) - 2 \leq 2\Delta(G) - 2$.

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 Graphen G 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 k-regulären Graphen mit einer ungeraden Knotenzahl und $m \geq 1$ Kanten gilt $\chi'(G) \geq k+1 \geq 3$.*

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

Als nächstes geben wir einen Algorithmus an, der für jeden Graphen G eine k-Kantenfärbung mit $k \leq \Delta(G)+1$ berechnet. Für den Beweis 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 $1 \leq i \neq j \leq k$.*

*a) Ein Nachbar v von u heißt **F-Nachbar** von u, wenn $c(u, v) \in F$ ist (wobei $c(u, v)$ für $c(\{u, v\})$ steht). Im Fall $F = \{i\}$ nennen wir v auch einen **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 **(i,j)-Subgraph** von G ist der Subgraph $G_{ij} = (V, E_{ij})$ mit $E_{ij} = \Big\{e \in E \;\Big|\; c(e) \in \{i, j\}\Big\}$.*

*d) Jede Zusammenhangskomponente G' von G_{ij} heißt **(i,j)-Komponente** von G. Ist G' ein Pfad oder ein Kreis, so nennen wir G' auch **(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. Wir bezeichnen diese k-Kantenfärbung c' mit $\mathit{switch}(c, i, j, G')$.

Satz 2.53 (Vizing 1964)**.** *Für jeden Graphen G gilt $\chi'(G) \leq \Delta(G)+1$.*

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

Prozedur `expand`$(G, c, e_1 = \{y_0, y_1\})$

```
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 Endknoten y_ℓ
13   c' ← switch(c, α_0, α_i, P)
14   sei z der Knoten am anderen Ende 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  c'(y_0, y_i) ← α
2  for j ← 1 to i − 1 do c'(y_0, y_j) ← α_j
```

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 $j = 1, \ldots, \ell$ 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 $j = 1, \ldots, i$ 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 $j = 1, \ldots, i-1$ mit α_j färben können.

Fall 4 In allen anderen Fällen 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. Da zudem die Farbe α_j für $j = 1, \ldots, \ell - 1$ an y_j frei bleibt (auch wenn $z \in \{y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_\ell\}$ ist), können wir die Kanten $\{y_0, y_j\}$ für $j = 1, \ldots, \ell - 1$ 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 $\mathcal{O}(nm)$. Zudem erhalten wir aus dem Beweis des Satzes von Vizing folgende Konsequenzen.

Korollar 2.54. *Für jeden Multigraphen $G = (V, E)$ gilt*

(i) $\chi'(G) \leq \Delta(G) + \max_{e \in E} v_E(e)$.

(ii) $\chi'(G) \leq 3\Delta(G)/2$.

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

Beweis. Siehe Übungen. ■

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.

Zum Schluss dieses Kapitels zeigen wir, dass die entsprechende Frage für Knotenfärbungen sehr leicht entscheidbar ist.

2.4 Der Satz von Brooks

Satz 2.55 (Brooks 1941)**.** *Für einen zusammenhängenden Graphen G gilt $\chi(G) = \Delta(G) + 1$ genau dann, wenn $G = C_{2n+1}$ oder $G = K_n$ für ein $n \geq 1$ ist.*

Beweis. Es ist klar, dass die Graphen $G = C_{2n+1}$ und $G = K_n$, $n \geq 1$, die chromatische Zahl $\Delta(G) + 1$ haben. Für $\Delta(G) \leq 2$ sind dies auch die einzigen zusammenhängenden Graphen mit dieser Eigenschaft.

Sei nun $G \neq K_{d+1}$ ein zusammenhängender Graph mit Maximalgrad $\Delta(G) = d \geq 3$. Wir zeigen induktiv über n, dass $\chi(G) \leq d$ ist. Im Fall $n \leq 4$ (IA) ist dies klar, da wir den K_4 ausgeschlossen haben. Für den IS können wir also $n \geq 5$ annehmen.

Falls $\kappa(G) \leq 1$ ist, hat G $k \geq 2$ Blöcke $B_1, \ldots B_k$. Dann ist jeder Block B_i nach IV (bzw. wegen $\Delta(B_i) < \Delta(G) = d$) d-färbbar und somit auch $\chi(G) \leq d$. Es bleibt also der Fall, dass $\kappa(G) \geq 2$ ist.

Behauptung 2.56. *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 $G \neq K_{d+1}$ 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. Wir wenden auf den Graphen $G - \{a, b\}$ eine Suche mit dem Startknoten u_1 an. Sei $(u_1, \ldots, u_{n-2})$ die resultierende Suchordnung. Nun starten wir `greedy-color` mit der Reihenfolge $(a, b, u_{n-2}, \ldots, u_1)$.

Dann berechnet `greedy-color` eine d-Färbung c, da die Knoten a und b die Farbe $c(a) = c(b) = 1$ erhalten. Zudem ist jeder Knoten u_i, $i > 1$, mit einem Knoten u_j mit $j < i$ verbunden, weshalb $c(u_i) \leq \deg(u_i) \leq d$ ist. Zuletzt erhält auch u_1 eine Farbe $c(u_1) \leq d$, da u_1 bereits zwei Nachbarn a und b mit derselben Farbe hat. ■

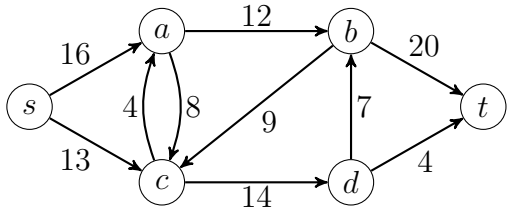
In den Übungen wird folgende Folgerung aus dem Beweis des Satzes von Brooks gezeigt.

Korollar 2.57. *Es gibt einen Linearzeitalgorithmus, der alle Graphen G mit $\Delta(G) \leq 3$ mit $\chi(G)$ Farben färbt.*

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*

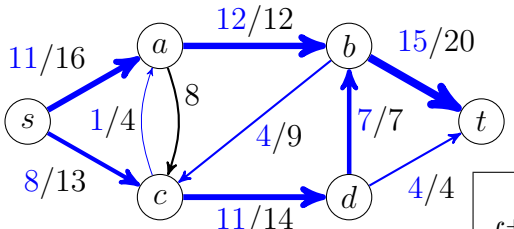
$$f(u, v) \leq c(u, v), \qquad \text{(Kapazitätsbedingung)}$$
$$f(u, v) = -f(v, u), \qquad \text{(Antisymmetrie)}$$
$$\textstyle\sum_{v \in V} f(u, v) = 0 \text{ für alle } u \in V \setminus \{s, t\} \qquad \text{(Kontinuität)}$$

*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
$f^+(u)$	19	12	19	12	11	0
$f^-(u)$	0	12	19	12	11	19

3.1 Der Ford-Fulkerson-Algorithmus

Wie lässt sich für einen Fluss f in einem Netzwerk N entscheiden, ob er vergrößert werden kann? Diese Frage lässt sich leicht beantworten, falls f der konstante Nullfluss $f = 0$ ist: 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.

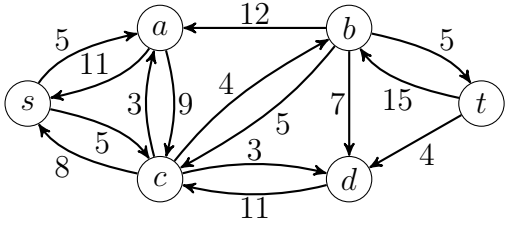
Definition 3.3. *Sei $N = (V, E, s, t, c)$ ein Netzwerk und sei f ein Fluss in N. Das zugeordnete* ***Restnetzwerk*** *ist $N_f = (V, E_f, s, t, c_f)$ mit der Kapazität*

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

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*

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

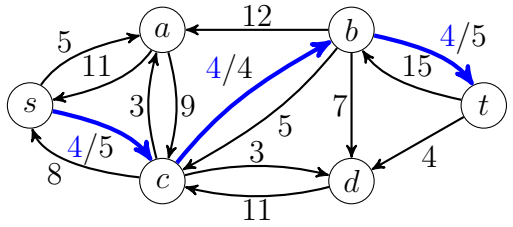
und der ***zu P gehörige Fluss in N_f*** *ist*

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

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

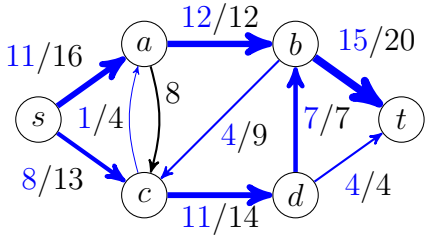
- $u_0 = s$ und $u_k = t$ ist,
- die Knoten $u_0, \ldots, u_k$ paarweise verschieden sind
- und $c_f(u_i, u_{i+1}) > 0$ für $i = 0, \ldots, k - 1$ ist.

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

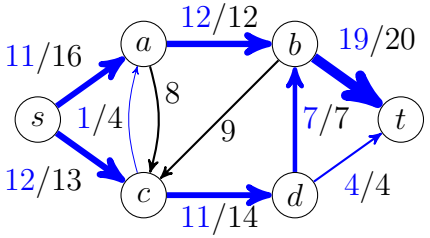


Es ist leicht zu sehen, dass f_P tatsächlich ein Fluss in N_f ist. Durch Addition der beiden Flüsse f und f_P erhalten wir einen Fluss $f' = f + f_P$ in N der Größe $|f'| = |f| + |f_P| > |f|$.

Fluss f:



Fluss $f' = f + f_P$:

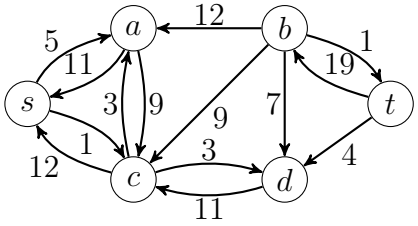


Nun können wir den **Ford-Fulkerson-Algorithmus** angeben.

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*

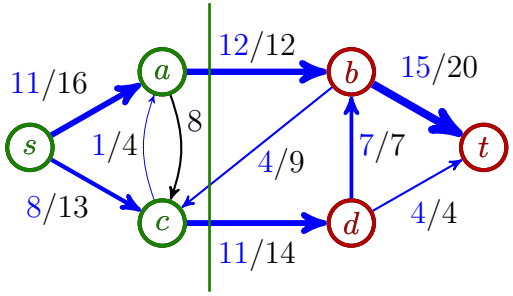
$$c(S) = \sum_{u \in S, v \notin S} c(u, v).$$

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:*



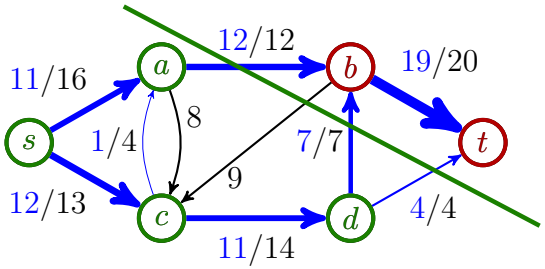
Dieser Schnitt hat die Kapazität

$$c(S) = c(a, b) + c(c, d) = 12 + 14 = 26$$

und der Fluss f durch ihn ist

$$f(S) = f(a, b) + f(c, b) + f(c, d) = 12 - 4 + 11 = 19.$$

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



die Kapazität

$$\begin{aligned} c(S') &= c(a,b)+c(d,b)+c(d,t) = 12+7+4 = 23 \\ &= f'(a,b)+f'(d,b)+f'(d,t) = f'(S'), \end{aligned}$$

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

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

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

$k-1 \rightsquigarrow k$**:** 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

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

Nach Induktionsvoraussetzung folgt somit $f(S) = f(S') = |f|$. Schließlich folgt wegen $f(u,v) \leq c(u,v)$ die Ungleichung

$$f(S) \;=\; \sum_{(u,v) \in E(S)} f(u,v) \;\leq\; \sum_{(u,v) \in E(S)} c(u,v) \;=\; c(S). \qquad \blacksquare$$

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

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

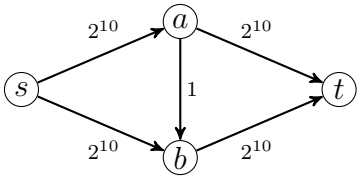
Die Implikation „*3* ⇒ *1*“ ergibt sich aus der Tatsache, dass im Fall $c(S) = |f|$ für jeden Fluss f' die Abschätzung $|f'| = f'(S) \leq c(S) = |f|$ gilt. ■

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.

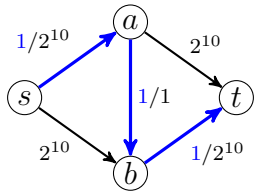
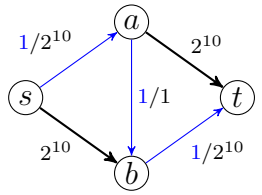
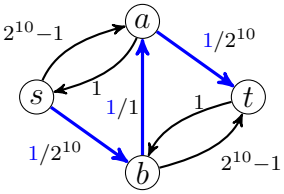
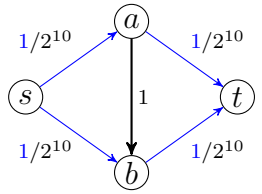
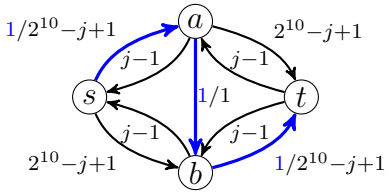
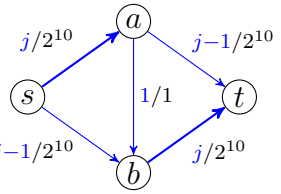
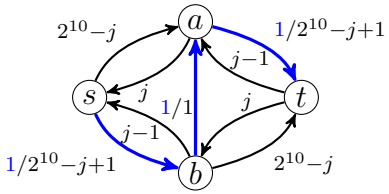
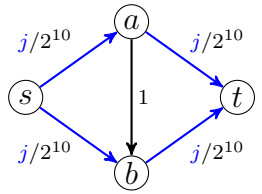
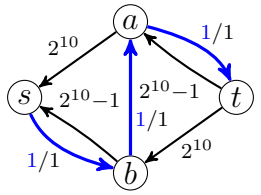
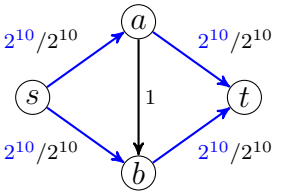


Im günstigsten Fall wird nämlich ausgehend vom Nullfluss f_1 zuerst der Zunahmepfad $P_1 = (s, a, t)$ mit der Kapazität 2^{10} und dann im Restnetzwerk N_{f_1} der Pfad $P_2 = (s, b, t)$ mit der Kapazität 2^{10} gewählt.

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).

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}		

- 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.

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 `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 f_1 der triviale Nullfluss und seien $P_1, \dots, P_k$ die Zunahmepfade, die der Edmonds-Karp-Algorithmus der Reihe nach berechnet, d.h. $f_{i+1} = f_i + f_{P_i}$. Eine Kante e in P_i heißt ***kritisch*** für P_i, falls der Fluss f_{P_i} im Restnetzwerk N_{f_i} die Kante e ***sättigt***, d.h. $c_{f_i}(e) = f_{P_i}(e) = c_{f_i}(P_i)$. Man beachte, dass eine kritische Kante e in 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.

Wir überlegen uns zunächst, dass die Längen ℓ_i von P_i (schwach) monoton wachsen. Hierzu zeigen wir, dass die Abstände jedes Knotens $u \in V$ von s und von t beim Übergang von N_{f_i} zu $N_{f_{i+1}}$ nicht kleiner werden können. Sei $d_i(u, v)$ die minimale Länge eines Pfades von u nach v im Restnetzwerk N_{f_i}.

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

Hierzu zeigen wir folgende Behauptung.

Behauptung 3.12. *Für jeden kürzesten Pfad $P = (u_0, \dots, u_h)$ von $u_0 = s$ nach $u_h = u$ in $N_{f_{i+1}}$ gilt $d_i(s, u_j) \leq d_i(s, u_{j-1}) + 1$, falls die Kante $e = (u_{j-1}, u_j)$ auch in N_{f_i} enthalten ist, und $d_i(s, u_j) = d_i(s, u_{j-1}) - 1$, falls e nicht in N_{f_i} enthalten ist.*

Falls die Kante $e = (u_{j-1}, u_j)$ auch in N_{f_i} enthalten ist, ist die Behauptung klar. Andernfalls muss $f_{i+1}(e) \neq f_i(e)$ sein, d.h. e oder e^R müssen in P_i vorkommen. Da e nicht in N_{f_i} ist, muss $e^R = (u_j, u_{j-1})$ auf P_i liegen. Da P_i ein kürzester Pfad von s nach t in N_{f_i} ist, folgt $d_i(s, u_{j-1}) = d_i(s, u_j) + 1$, was $d_i(s, u_j) = d_i(s, u_{j-1}) - 1$ impliziert.
Damit ist Behauptung 3.12 bewiesen und es folgt

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

Die folgende Behauptung lässt sich analog zu Behauptung 3.12 zeigen.

Behauptung 3.13. *Für jeden kürzesten Pfad $P = (u_0, \dots, u_h)$ von $u_0 = u$ nach $u_h = t$ in $N_{f_{i+1}}$ gilt $d_i(u_{j-1}, t) \leq d_i(u_j, t) + 1$, falls die Kante $e = (u_{j-1}, u_j)$ auch in N_{f_i} enthalten ist, und $d_i(u_{j-1}, t) \leq d_i(u_j, t) - 1$, falls e nicht in N_{f_i} enthalten ist.*

Damit folgt

$$d_i(u, t) \leq d_i(u_1, t) + 1 \leq \cdots \leq d_i(t, t) + h = h = d_{i+1}(u, t),$$

womit Behauptung 3.11 bewiesen ist. Als nächstes zeigen wir folgende Behauptung.

Behauptung 3.14. *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 $l_j \geq l_i + 2$.*

Dies folgt direkt aus Behauptung 3.11 und der Tatsache, dass P_i und P_j kürzeste Zunahmepfade sind:

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

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

Seien $P_{i_1}, \dots, P_{i_h}$, $i_1 < \cdots < i_h$, die Pfade, in denen e oder e^R kritisch ist. Falls $e' \in \{e, e^R\}$ kritisch in P_{i_j} ist, dann verschwindet e' aus $N_{f_{i_j+1}}$. Damit also e oder e^R kritisch in $P_{i_{j+1}}$ sein können, muss ein Pfad $P_{j'}$ mit $i_j < j' \leq i_{j+1}$ existieren, der e'^R enthält. Wegen Behauptung 3.11 und Behauptung 3.14 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.

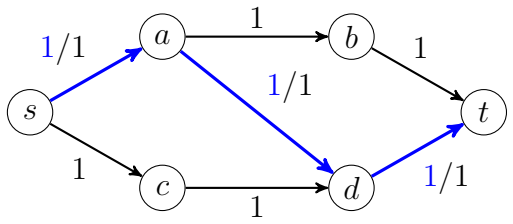
Die Analyse der Laufzeit des Edmonds-Karp-Algorithmus beruht auf der Tatsache, dass der Fluss f_{P_i} durch den Zunahmepfad P_i, der in jedem Schleifendurchlauf auf den aktuellen Fluss f_i addiert wird, auf *mindestens einem* kürzesten Pfad im Restnetzwerk N_{f_i} eine Kante sättigt. Dies hat zur Folge, 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 3.16. *Sei $N = (V, E, s, t, c)$ ein Netzwerk und sei g ein Fluss in N. Der Fluss g **sättigt** eine Kante $e \in E$, falls $g(e) = c(e)$ ist. g heißt **blockierend**, falls g mindestens eine Kante e 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 3.17. *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*

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

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`(V, E, s, t, c)

```
1  for all (u,v) ∈ V × V do
2    f(u,v) ← 0
3  while es gibt einen blockierenden Fluss g in N'_f mit |g| > 0 do
4    f ← f + g
```

Das zum Restnetzwerk $N_f = (V, E, s, t, c_f)$ gehörige Schichtnetzwerk $N'_f = (V', E', s, t, c'_f)$ wird von der Prozedur `schichtnetzwerk`(f) in Zeit $O(n+m)$ berechnet. Für die Berechnung eines blockierenden Flusses g im Schichtnetzwerk N'_f werden wir 2 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 wurden. Falls t erreicht wurde, werden alle Kanten aus E' entfernt, die nicht zwischen zwei Knoten aus V' verlaufen, wobei V' außer der Senke t alle Knoten u mit einem Abstand $d(s,u) < d(s,t)$ in N_f enthält. Andernfalls existiert in N_f (und damit in N'_f) kein (blockierender) Fluss g mit $|g| > 0$.

Prozedur `schichtnetzwerk`(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 V' ← {v ∈ V | niv(v) < niv(t)} ∪ {t}
14 if head(Q) = t then E' ← E' ∩ (V' × V')
```

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 konstantem 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 S rekonstruieren, weshalb er ***S-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 N'_f).

Prozedur `blockfluss1`(f)

```
1  for all e ∈ E' ∪ E'^R do g(e) ← 0
2  u ← s; S ← (s)
3  done ← false
4  repeat
5    if ∃ e = (u, v) ∈ E' then
6      push(S, v)
7      c''(e) ← c'(e) − g(e)
8      u ← v
9    elsif u = t then
10     P ← S-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; S ← (s)
16   elsif u ≠ s then
17     pop(S)
18     u' ← top(S)
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 3.18. *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}. Wir zeigen, dass $d_i(s,t) < d_{i+1}(s,t)$ ist. Da $d_1(s,t) \geq 1$ und $d_k(s,t) \leq n-1$ ist, folgt $k \leq n-1$.

Behauptung 3.19. *Für jeden Knoten $u \in V$ gilt $d_i(s,u) \leq d_{i+1}(s,u)$.*

Hierzu zeigen wir folgende Behauptung.

Behauptung 3.20. *Für jeden kürzesten Pfad $P = (u_0, \ldots, u_h)$ von $u_0 = s$ nach $u_h = u$ in $N_{f_{i+1}}$ gilt $d_i(s,u_j) \leq d_i(s,u_{j-1}) + 1$, falls die Kante $e = (u_{j-1}, u_j)$ auch in N_{f_i} enthalten ist, und $d_i(s,u_j) = d_i(s,u_{j-1}) - 1$, falls e nicht in N_{f_i} enthalten ist.*

Falls die Kante $e = (u_{j-1}, u_j)$ auch in N_{f_i} enthalten ist, ist die Behauptung klar. Andernfalls muss $f_{i+1}(e) \neq f_i(e)$ sein, d.h. $g_i(e)$ muss ungleich 0 sein. Da e nicht in N_{f_i} und somit auch nicht in N'_{f_i} enthalten ist, muss $e^R = (u_j, u_{j-1})$ in N'_{f_i} sein. Da N'_{f_i} nur Kanten auf kürzesten Pfaden mit Startknoten s enthält, folgt $d_i(s,u_{j-1}) = d_i(s,u_j) + 1$, was $d_i(s,u_j) = d_i(s,u_{j-1}) - 1$ impliziert.
Damit ist Behauptung 3.20 bewiesen und es folgt Behauptung 3.19:

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

Nun zeigen wir folgende Behauptung.

Behauptung 3.21. *Für $i = 1, \ldots, k-1$ gilt $d_i(s,t) < d_{i+1}(s,t)$.*

Sei $P = (u_0, u_1, \ldots, u_h)$ ein kürzester Pfad von $s = u_0$ nach $t = u_h$ in $N_{f_{i+1}}$ (und somit auch in $N'_{f_{i+1}}$). Mit Behauptung 3.19 folgt, dass

$d_i(s,u_j) \le d_{i+1}(s,u_j) = j$ für $j = 0,\dots,h$ ist, also insbesondere $d_i(s,t) \le d_{i+1}(s,t) = h$ ist.

Wenn alle Knoten u_j in N'_{f_i} enthalten sind, muss ein j mit $d_i(s,u_j) \le d_i(s,u_{j-1})$ existieren, woraus wegen Behauptung 3.20

$$d_i(s,t) \le d_i(s,u_j) + h - j \le \underbrace{d_i(s,u_{j-1})}_{\le j-1} + h - j < h = d_{i+1}(s,t)$$

folgt. Würde nämlich $d_i(s,u_j) > d_i(s,u_{j-1})$ für $j = 1,\dots,h$ gelten, so wären nach Behauptung 3.20 alle Kanten (u_{j-1},u_j) auch in N'_{f_i} enthalten. Dann wäre aber P ein kürzester Pfad von s nach t in N_{f_i} und somit ein s-t-Pfad in N'_{f_i}, der von g_i nicht blockiert wird.

Falls mindestens ein Knoten u_j nicht in N'_{f_i} enthalten ist, sei u_j der erste solche Knoten auf P. Da $u_j \neq t$ ist, folgt $d_{i+1}(s,u_j) = j < h = d_{i+1}(s,t)$. Zudem liegt die Kante $e = (u_{j-1},u_j)$ 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_{j-1} in N'_{f_i} und e in N_{f_i} ist, kann u_j nur aus dem Grund nicht zu N'_{f_i} gehören, dass $d_i(s,u_j) = d_i(s,t)$ ist. Daher folgt wegen $d_i(s,u_j) \le d_i(s,u_{j-1}) + 1$ (Behauptung 3.20) und $d_i(s,u_{j-1}) \le d_{i+1}(s,u_{j-1})$ (Behauptung 3.19)

$$d_i(s,t) = d_i(s,u_j) \le \underbrace{d_i(s,u_{j-1})}_{\le d_{i+1}(s,u_{j-1})} + 1 = d_{i+1}(s,u_j) < d_{i+1}(s,t). \quad \blacksquare$$

Korollar 3.22. *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 3.23. *Sei $N = (V,E,s,t,c)$ ein Netzwerk und sei g ein Fluss in N sowie $u \in V$.*

a) Der Fluss g ***sättigt den Knoten u****, 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.*

b) Der ***Durchsatz von u*** *ist*

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

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.*

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

Proposition 3.24. *Sei $N = (V,E,s,t,c)$ ein Netzwerk und sei g ein Fluss in N. Wenn g auf jedem s-t-Pfad in N mindestens einen Knoten u sättigt, dann ist g blockierend.*

Beweis. Dies folgt aus der Tatsache, dass ein Fluss g in N, der auf jedem s-t-Pfad P mindestens einen Knoten u sättigt, auch mindestens eine Kante auf dem Pfad P sättigt. $\blacksquare$

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`(f)

```
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
7      D(u) ← min{D⁻(u), D⁺(u)}
8    D(s) ← D⁺(s)
9    D(t) ← D⁻(t)
10   wähle u ∈ V' mit D(u) minimal
11   B := {u}
12   propagierevor(u)
13   propagiererück(u)
14   while ∃u ∈ B \ {s,t} do
15     B ← B \ {u}; V' ← V' \ {u}
16     for all e = (u,v) ∈ E' do
17       D⁻(v) ← D⁻(v) − c'(u,v)
18       if D⁻(v) = 0 then B := B ∪ {v}
19       E' ← E' \ {e}
20     for all e = (v,u) ∈ E' do
21       D⁺(v) ← D⁺(v) − c'(v,u)
22       if D⁺(v) = 0 then B := B ∪ {v}
23       E' ← E' \ {e}
24 until u ∈ {s,t}
25 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 3.24 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)
4  while Q ≠ () do
5    v ← dequeue(Q)
6    while z(v) ≠ 0 ∧ ∃e = (v,w) ∈ E' do
7      m ← min{z(v), c'(e)}; z(v) ← z(v) − m; z(w) ← z(w) + m
8      aktualisierekante(e,m)
9      enqueue(Q,w)
```

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

```
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 `propagierevorück` unterscheidet sich von der Prozedur `propagierevor` nur dadurch, dass in Zeile 5 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 `propagierevorück` 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 `propagierevorück`.

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

Auf Netzwerken, deren Flüsse durch jede Kante oder durch jeden Knoten durch eine relativ kleine Zahl C beschränkt sind, lassen sich noch bessere Laufzeitschranken für den Dinitz-Algorithmus nachweisen. Hierzu benötigen wir folgende Beziehungen zwischen einem Netzwerk und den zugehörigen Restnetzwerken.

Lemma 3.26. *Sei $N = (V, E, s, t, c)$ ein Netzwerk, f ein Fluss in N und N_f das zugehörige Restnetzwerk. Zudem sei $h : V \times V \to \mathbb{Z}$.*

(i) Die Funktion h ist genau dann ein Fluss (bzw. maximaler Fluss) in N_f, wenn $f + h$ ein Fluss (bzw. maximaler Fluss) in N ist.

(ii) Für jede Kante $e \in E \cup E^R$ gilt $c_f(e) + c_f(e^R) = c(e) + c(e^R)$.

(iii) Für jeden Knoten $u \in V \setminus \{s, t\}$ gilt $c_f^+(u) = c^+(u)$ und $c_f^-(u) = c^-(u)$ und somit $D_f(u) = D(u)$.

Beweis.

(i) Da f die Antisymmetrie und die Kontinuität erfüllt, übertragen sich diese Eigenschaften von h auf $f + h$ und umgekehrt. Weiter gilt

$$h(u, v) \leq \underbrace{c_f(u, v)}_{c(u,v)-f(u,v)} \Leftrightarrow f(u, v) + h(u, v) \leq c(u, v),$$

d.h. h erfüllt genau dann die Kapazitätsbedingung in N_f, wenn $f + h$ sie in N erfüllt. Zudem ist $f + h$ genau dann ein maximaler Fluss in N, wenn h ein maximaler Fluss in N_f ist, da jeder Fluss h' in N_f mit $|h'| > |h|$ einen Fluss $f + h'$ der Größe $|f + h'| > |f + h|$ in N und jeder Fluss g in N mit $|g| > |f + h|$ einen Fluss $g - f$ der Größe $|g - f| = |g| - |f| > |f + h| - |f| = |h|$ in N_f liefern würde.

(ii) Es gilt $\underbrace{c_f(e)}_{c(e)-f(e)} + \underbrace{c_f(e^R)}_{c(e^R)-f(e^R)} = c(e) + c(e^R) - (\underbrace{f(e) + f(e^R)}_{=0})$.

(iii) Für jeden Knoten $u \in V \setminus \{s, t\}$ gilt

$$c_f^+(u) = \sum_{v \in V} \underbrace{c_f(u, v)}_{c(u,v)-f(u,v)} = \underbrace{\sum_{v \in V} c(u, v)}_{c^+(u)} - \underbrace{\sum_{v \in V} f(u, v)}_{=0}.$$

Die Gleichheit $c_f^-(u) = c^-(u)$ folgt analog. ■

Lemma 3.27. *Sei $N = (V, E, s, t, c)$ ein Netzwerk mit einem maximalen Fluss f der Größe $F > 0$ und sei ℓ die Länge des zugehörigen Schichtnetzwerks N'.*

(i) Falls jeder Knoten $u \in V \setminus \{s, t\}$ einen Durchsatz $D(u) \leq C$ hat, gilt $\ell \leq 1 + (n - 2)C/F$.

(ii) Falls jede Kante $e \in E$ eine Kapazität $c(e) \leq C$ hat, gilt $\ell \leq \min\{mC/F, 2n\sqrt{C/F}\}$.

Beweis.

(i) Seien $S_0, \dots, S_\ell$ alle Schichten von N'. Da der Fluss f in N durch die Knoten jeder einzelnen Schicht S_j für $j = 1, \dots, \ell - 1$ fließt, und jeder dieser Knoten einen Durchsatz $\leq C$ hat, muss

$$F \leq C|S_j| \text{ bzw. } F/C \leq |S_j|$$

sein, woraus $(\ell - 1)F/C \leq |S_1| + \dots + |S_{\ell-1}| \leq n - 2$ bzw. $\ell \leq 1 + (n-2)C/F$ folgt.

(ii) Seien $S_0, \dots, S_{\ell-1}$ die ersten $\ell - 1$ Schichten von N' und sei $S_\ell = V - \bigcup_{j=0}^{\ell-1} S_j$. Sei E_j die Menge der Kanten von S_j nach S_{j+1} und sei k_j deren Anzahl. Da der Fluss f in N jeweils durch die k_j Kanten von S_j nach S_{j+1} fließt, die alle eine Kapazität $\leq C$ haben, muss

$$F \leq Ck_j \leq C|S_j||S_{j+1}| \text{ bzw. } F/C \leq k_j \leq |S_j||S_{j+1}|$$

sein, woraus sofort $m \geq \sum_{j=0}^{\ell-1} k_j \geq \ell F/C$ bzw. $\ell \leq mC/F$ folgt. Da wegen $F/C \leq |S_j||S_{j+1}|$ zudem mindestens eine von zwei benachbarten Schichten S_j und S_{j+1} mindestens $\sqrt{F/C}$ Knoten enthält, folgt auch

$$(\ell/2)\sqrt{F/C} \leq |S_0| + \dots + |S_\ell| = n \text{ bzw. } \ell \leq 2n\sqrt{C/F}. \quad \blacksquare$$

Satz 3.28. *Sei $a(N)$ die Anzahl der Schleifendurchläufe des Algorithmus von Dinitz bei Eingabe eines Netzwerks $N = (V, E, s, t, c)$.*

(i) Falls jeder Knoten $u \in V \setminus \{s, t\}$ einen Durchsatz $D(u) \leq C$ hat, so gilt $a(N) \leq (1 + 2(Cn)^{1/2})$.

(ii) Falls jede Kante $e \in E$ eine Kapazität $c(e) \leq C$ hat, so gilt $a(N) \leq \min\{(2^3 mC)^{1/2}, (2^6 Cn^2)^{1/3}\}$.

Beweis. Sei $F = |f|$ die Größe eines maximalen Flusses f in N.

(i) Da die Anzahl a der Schleifendurchläufe durch F beschränkt ist, können wir $F > (Cn)^{1/2}$ annehmen. Betrachte den i-ten Schleifendurchlauf, in dem ein blockierender Fluss im Schichtnetzwerk N'_{f_i} berechnet wird. Da $f - f_i$ nach Lemma 3.26(i) ein maximaler Fluss in N_{f_i} der Größe $R_i = F - |f_i|$ ist und nach Lemma 3.26(iii) jeder Knoten $u \in V \setminus \{s, t\}$ in N_{f_i} den gleichen Durchsatz wie in N hat, folgt nach Lemma 3.27(i), dass N'_{f_i} eine Länge $\ell_i \leq 1 + nC/R_i$ hat. Damit ist die Anzahl a der Schleifendurchläufe durch

$$a \leq i + R_{i+1} \leq \ell_i + R_{i+1} \leq R_{i+1} + 1 + nC/R_i$$

beschränkt. Nun wählen wir i so, dass $R_i > (Cn)^{1/2}$ und $R_{i+1} \leq (Cn)^{1/2}$ ist. Dann folgt

$$a - 1 \leq R_{i+1} + nC/R_i < (Cn)^{1/2} + nC/(Cn)^{1/2} = 2(Cn)^{1/2}.$$

(ii) Betrachte den i-ten Schleifendurchlauf, in dem ein blockierender Fluss im Schichtnetzwerk N'_{f_i} berechnet wird. Da jede Kante $e \in E_{f_i}$ nach Lemma 3.26(ii) eine Kapazität $c_{f_i}(e) \leq 2C$ hat und $f - f_i$ nach Lemma 3.26(i) ein maximaler Fluss in N_{f_i} ist und die Größe $R_i = F - |f_i|$ hat, folgt nach Lemma 3.27(ii), dass N'_{f_i} eine Länge $\ell_i \leq 2mC/F$ hat. Damit ist die Anzahl a der Schleifendurchläufe durch

$$a \leq i + R_{i+1} \leq \ell_i + R_{i+1} \leq R_{i+1} + 2mC/R_i$$

beschränkt. Falls $F \leq (2mC)^{1/2}$ ist, folgt sofort $a \leq (2mC)^{1/2}$. Andernfalls wählen wählen wir i so, dass $R_i > (2mC)^{1/2}$ und $R_{i+1} \leq (2mC)^{1/2}$ ist. Dann folgt

$$a \leq (2mC)^{1/2} + (2mC)^{1/2} = (2^3 mC)^{1/2}.$$

Zudem folgt nach Lemma 3.27(ii), dass N'_{f_i} eine Länge $\ell_i \leq 2n\sqrt{2C/R_i}$ hat. Damit ist die Anzahl a der Schleifendurchläufe auch durch

$$a \leq i + R_{i+1} \leq \ell_i + R_{i+1} \leq R_{i+1} + 2n\sqrt{2C/R_i}$$

beschränkt. Falls $F \leq (2n\sqrt{2C})^{2/3}$ ist, folgt sofort $a \leq (2n\sqrt{2C})^{2/3}$. Andernfalls wählen wir i so, dass $R_i > (2n\sqrt{2C})^{2/3}$ und $R_{i+1} \leq (2n\sqrt{2C})^{2/3}$ ist. Dann folgt

$$a \leq (2n\sqrt{2C})^{2/3} + 2n\sqrt{2C}/(2n\sqrt{2C})^{1/3} = (2^6 Cn^2)^{1/3}. \quad \blacksquare$$

Korollar 3.29. *Sei $T(N)$ die Laufzeit des Algorithmus von Dinitz unter Verwendung der Prozedur* `blockfluss1` *bei Eingabe eines Netzwerks $N = (V, E, s, t, c)$.*

(i) Falls jeder Knoten $u \in V \setminus \{s, t\}$ einen Durchsatz $d(u) \leq C$ hat, so gilt $T(N) = O((nC + m)\sqrt{Cn})$.

(ii) Falls jede Kante $e \in E$ eine Kapazität $c(e) \leq C$ hat, so gilt $T(N) = O(\min\{(mC)^{3/2}, C^{4/3}n^{2/3}m\})$.

Beweis. Zunächst ist leicht zu sehen, dass die Kapazitätschranke auf den Kanten oder Knoten auch für jedes Schichtnetzwerk N'_{f_i} gilt.

(i) Jedesmal wenn `blockfluss1` einen s-t-Pfad P im Schichtnetzwerk findet, verringert sich der Durchsatz $c''(u)$ der auf P liegenden Knoten u um den Wert $c'_g(P) \geq 1$, da der Fluss g durch diese Knoten um diesen Wert steigt. Daher kann jeder Knoten an maximal C Flusserhöhungen beteiligt sein, bevor sein Durchsatz auf 0 sinkt. Da somit pro Knoten ein Zeitaufwand von $O(C)$ für alle erfolgreichen Tiefensuchschritte, die zu einem s-t-Pfad führen, und zusätzlich pro Kante ein Zeitaufwand von $O(1)$ für alle nicht erfolgreichen Tiefensuchschritte anfällt, läuft `blockfluss1` in Zeit $O(nC + m)$.

(ii) Jedesmal wenn `blockfluss1` einen s-t-Pfad P im Schichtnetzwerk findet, verringert sich die Kapazität $c''(e)$ der auf P liegenden Kanten e um den Wert $c'_g(P) \geq 1$. Da somit pro Kante ein Zeitaufwand von $O(C)$ für alle erfolgreichen Tiefensuchschritte und $O(1)$ für alle nicht erfolgreichen Tiefensuchschritte anfällt, läuft `blockfluss1` in Zeit $O(mC + m) = O(mC)$. $\blacksquare$

4 Matchings

Definition 4.1. *Sei $G = (V, E)$ ein Graph.*

- *Zwei Kanten $e, e' \in E$ heißen* ***unabhängig****, falls $e \cap e' = \emptyset$ ist.*
- *Eine Kantenmenge $M \subseteq E$ heißt* ***Matching*** *in G, falls alle Kanten in M paarweise unabhängig sind.*
- *Sei $M \subseteq E$. Ein Knoten $v \in V$ heißt* ***M-gebunden****, falls v Endpunkt einer Kante $e \in M$ (also $v \in \bigcup M$) ist und sonst* ***M-frei****.*
- *Ein Matching M heißt* ***perfekt****, falls alle Knoten in G M-gebunden sind (also $V = \bigcup M$ ist).*
- *Die Matchingzahl von G ist*

$$\mu(G) = \max\{|M| \mid M \text{ ist ein Matching in } G\}$$

- *Ein Matching M heißt* ***maximal****, falls $|M| = \mu(G)$ ist. M heißt* ***gesättigt****, falls es in keinem größeren Matching enthalten ist.*

Offensichtlich ist $M \subseteq E$ genau dann ein Matching, wenn $|\bigcup M| = 2|M|$ ist. Das Ziel besteht nun darin, ein maximales Matching M in G zu finden.

Beispiel 4.2. *Ein gesättigtes Matching muss nicht maximal sein:*

$M = \{\{v, w\}\}$ ist gesättigt, da es sich nicht erweitern lässt. M ist jedoch kein maximales Matching, da $M' = \{\{v, x\}, \{u, w\}\}$ größer ist.

Die Greedy-Methode, ausgehend von $M = \emptyset$ solange Kanten zu M hinzuzufügen, bis sich M nicht mehr zu einem größeren Matching erweitern lässt, funktioniert also nicht.

Durch eine einfache Reduktion des bipartiten Matchingproblems auf ein Flussproblem erhält man aus Korollar 3.29 das folgende Resultat.

Satz 4.3. *In einem bipartiten Graphen lässt sich ein maximales Matching in Zeit $O(m\sqrt{n})$ bestimmen.*

Beweis. Sei $G = (A, B, E)$ der gegebene bipartite Graph. Konstruiere das Netzwerk $N = (V, E', s, t, c)$ mit den Knoten $V = A \cup B \cup \{s, t\}$ und den Kanten

$$E' = (\{s\} \times A) \cup \Big\{(u, v) \in A \times B \;\Big|\; \{u, v\} \in E\Big\} \cup (B \times \{t\}),$$

die alle Kapazität 1 haben. Es ist leicht zu sehen, dass sich aus jedem Matching M in G ein Fluss f in N konstruieren lässt mit $|M| = |f|$ und umgekehrt. Es genügt also, einen maximalen Fluss in N zu finden. Nach Korollar 3.29 ist dies mit dem Algorithmus von Dinitz unter Einsatz von `blockfluss1` in Zeit $O(m\sqrt{n})$ möglich, da der Durchsatz aller Knoten außer s und t durch 1 beschränkt ist. ■

In den Übungen wird gezeigt, dass sich die Laufzeit durch eine verbesserte Analyse sogar durch $O(m\sqrt{\mu})$ begrenzen lässt.

Die Konstruktion aus Satz 4.3 lässt sich nicht ohne Weiteres auf allgemeine, nicht bipartite Graphen verallgemeinern. Wir werden jedoch sehen, dass sich manche bei den Flussalgorithmen verwendete Ideen auch für Matchingalgorithmen einsetzen lassen. So lassen sich Matchings, die nicht maximal sind, ähnlich vergrößern wie dies bei nicht maximalen Flüssen durch einen Zunahmepfad möglich ist.

Definition 4.4. *Sei $G = (V, E)$ ein Graph und sei M ein Matching in G.*

1. *Ein Pfad $P = (u_0, \ldots, u_l)$ in G der Länge $l \geq 1$ heißt **M-alternierend**, falls für $i = 1, \ldots, l-1$ gilt:*
$$e_i = \{u_{i-1}, u_i\} \in M \Leftrightarrow e_{i+1} = \{u_i, u_{i+1}\} \notin M.$$
2. *Ein Kreis $C = (u_1, \ldots, u_l)$ in G heißt **M-alternierend**, falls der Pfad $P = (u_1, \ldots, u_l)$ M-alternierend ist und zudem gilt:*
$$\{u_1, u_2\} \in M \Leftrightarrow \{u_1, u_l\} \notin M.$$
3. *Ein M-alternierender Pfad $P = (u_0, \ldots, u_l)$ heißt **M-vergrößernder Pfad** (oder einfach **M-Pfad**), falls beide Endpunkte von P M-frei sind.*

Satz 4.5. *Ein Matching M in G ist genau dann maximal, wenn es keinen M-Pfad in G gibt.*

Beweis. Ist $P = (u_0, \ldots, u_l)$ ein M-Pfad, so liefert $M' = M \triangle P$ ein Matching der Größe $|M'| = |M| + 1$ in G. Hierbei identifizieren wir P mit der Menge $\{\{u_{i-1}, u_i\} \mid i = 1, \ldots, l\}$ der auf P liegenden Kanten. Ist dagegen M nicht maximal und M' ein größeres Matching, so betrachten wir die Kantenmenge $M \triangle M'$. Da jeder Knoten in dem Graphen $G' = (V, M \triangle M')$ höchstens den Grad 2 hat, lässt sich G' in disjunkte Kreise und Pfade zerlegen. Da diese Kreise und Pfade M-alternierend sind, und M' größer als M ist, muss mindestens einer dieser Pfade ein M-Pfad sein. ■

Damit haben wir das Problem, ein maximales Matching in einem Graphen G zu finden, auf das Problem reduziert, zu einem Matching M in G einen M-Pfad zu finden (sofern ein solcher existiert).

4.1 Der Algorithmus von Edmonds

Sei G ein Graph ohne isolierte Knoten und sei M ein Matching in G. Dann sucht der Algorithmus von Edmonds wie folgt nach einem M-Pfad in G. Jeder Knoten u hat einen von 3 Zuständen: gerade,

ungerade oder unerreicht. Zu Beginn sind alle M-freien Knoten gerade und alle M-gebundenen Knoten unerreicht. Dann wird ausgehend von den M-freien Knoten als Wurzeln ein Suchwald W in G aufgebaut, indem ausgehend von den geraden Knoten u eine Kante zu einem unerreichten oder ebenfalls geraden Knoten v besucht wird.

Ist v unerreicht, so wird der aktuelle Suchwald W nicht nur um die Kante (u, v), sondern auch um die Matching-Kante $(v, M(v))$ erweitert, wobei $M(v)$ der Matchingpartner von v ist. Zudem wechselt der Zustand von v von unerreicht zu ungerade und der von $M(v)$ von unerreicht zu gerade. Somit erhält jeder erreichte Knoten v genau dann den Zustand gerade, wenn der Pfad in W von v zu seiner Wurzel r_v eine gerade Länge hat.

Ist v dagegen gerade, so gibt es zwei Unterfälle. Haben u und v verschiedene Wurzeln $r_u \neq r_v$, so wurde ein M-Pfad $P(u, v)$ gefunden, der von r_u zu u über v zu r_v verläuft, und M kann vergrößert werden.

Andernfalls befindet sich v im gleichen Suchbaum wie u, d.h. u und v haben einen gemeinsamen Vorfahren b, so dass durch Verbinden des Pfades P von b nach u, der Kante $\{u, v\}$ und des Pfades P' von b nach v ein Kreis $C = C(u, v)$ in W entsteht. Da u und v beide gerade sind, hat C eine ungerade Länge. Zudem muss auch b gerade sein, da jeder ungerade Knoten in W genau ein Kind hat. Der Kreis C wird als ***Blüte*** und der Knoten b als ***Basis*** von C bezeichnet.

Zwar führt das Auffinden einer Blüte C nicht direkt zu einem M-Pfad. Sie bedeutet dennoch einen Fortschritt, da sich G wie folgt verkleinern lässt. Wir kontrahieren die Blüte C zu ihrer Basis b, die die Nachbarschaften aller Knoten in C zu Knoten außerhalb von C erbt. Nun suchen wir in dem kontrahierten Graphen G_C nach einem M_C-Pfad, wobei das Matching M_C aus M durch Entfernen aller Kanten auf dem Kreis C entsteht. Folgendes Lemma zeigt, dass sich tatsächlich aus jedem M_C-Pfad in G_C ein M-Pfad in G gewinnen lässt.

Lemma 4.6. *Falls C eine Blüte in G ist, lässt sich jeder M_C-Pfad P in G_C zu einem M-Pfad in G expandieren.*

Beweis. Falls P nicht auch ein M-Pfad in G ist, muss P den Knoten b enthalten. Sei a der Nachbar von b in P, der mit b nicht über eine Matchingkante verbunden ist (d.h. $\{a, b\} \notin M_C$). Da die Kante $\{a, b\}$ in G fehlt (sonst wäre P ein M-Pfad in G), muss a in G einen Nachbarn c in der Blüte C haben. Nun ersetzen wir die Kante $\{a, b\}$ in P durch die Kante $\{a, c\}$ und fügen P denjenigen der beiden Pfade $P_1(c, b)$ und $P_2(c, b)$ von c nach b in C hinzu, der c über eine Matchingkante verlässt (also gerade Länge hat). ■

Die folgende Prozedur `FindePfad` berechnet einen vergrößernden Pfad für G, falls das aktuelle Matching M nicht bereits maximal ist. Da M nicht mehr als $n/2$ Kanten enthalten kann, muss diese Prozedur höchstens $(n/2 + 1)$-mal aufgerufen werden, um ein maximales Matching in G zu berechnen.

Prozedur `FindePfad`$(G = (V, E), M)$

```
1  Q ← ∅
2  for u ∈ V do
3    parent(u) ← ⊥
4    if ∃e ∈ M : u ∈ e then
5      zustand(u)← unerreicht
6    else
7      zustand(u) ← gerade
8      root(u) ← u
9      Q ← Q ∪ {(u, v) | {u, v} ∈ E}
10 while Q ≠ ∅ do
11   entferne eine Kante (u, v) aus Q
12   if zustand(v) = unerreicht then
13     parent(v) ← u
14     zustand(v) ← ungerade
15     parent(M(v)) ← v
16     zustand(M(v)) ← gerade
17     root(M(v)) ← root(v) ← root(u)
18     Q ← Q ∪ {(M(v), w) | {M(v), w} ∈ E \ M}
```

```
19  if zustand(v) = gerade then
20    if root(u) = root(v) then // Blüte gefunden
21      kontrahiere die Blüte C = C(u, v) zu ihrer Basis b,
22      modifiziere dabei parent und Q entsprechend und
23      speichere den Kreis C unter der Basis b ab
24      füge alle Kanten (b, a) zu Q hinzu, so dass a ∉ C ist
25      und ein ungerader Knoten c ∈ C mit {c, a} ∈ E existiert
26    else // M-Pfad gefunden
27      berechne den r_u-r_v-Pfad P in W ∪ W^R ∪ {u, v}, expandiere
28      P zu einem M-Pfad P' in G und gib P' zurück
```

Da sich die Anzahl der Knoten von G bei jeder Kontraktion einer Blüte mindestens um 2 verringert, werden höchstens $n/2$ Blüten kontrahiert bis ein M-Pfad gefunden wird. Bei Verwendung entsprechender Datenstrukturen zur Verwaltung der Blüten lässt sich die Prozedur `FindePfad` in Zeit $O(m)$ implementieren, was auf eine Gesamtlaufzeit von $O(nm)$ für den Algorithmus von Edmonds führt.

Tatsächlich lässt sich die Laufzeit wie im bipartiten Fall auf $O(m\sqrt{\mu})$ verbessern. Dazu berechnet man ähnlich wie beim Algorithmus von Dinitz pro Runde nicht nur einen M-Pfad, sondern in Zeit $O(m)$ eine maximale Menge knotendisjunkter M-Pfade, die alle eine minimale Länge haben. Man kann zeigen, dass $O(\sqrt{\mu})$ solcher Runden ausreichen, um ein maximales Matching zu finden (vgl. Hopcroft-Karp-Algorithmus im bipartiten Fall und Algorithmus von Micali und Vazirani für beliebige Graphen).

Für den Beweis der Korrektheit des Edmonds-Algorithmus (genauer: zum Nachweis der Maximalität des berechneten Matchings M) benötigen wir den Begriff der Odd Set Cover (OSC). Es ist leicht zu sehen, dass für jedes Matching M in einem Graphen $G = (V, E)$ und jede Knotenüberdeckung $C \subseteq V$ in G die Ungleichung $|M| \leq |C|$ gilt. Wir werden sehen, dass es in bipartiten Graphen sogar ein Matching M und eine Knotenüberdeckung C mit $|M| = |C|$ gibt. Die Angabe einer Knotenüberdeckung C mit $|C| = |M|$ bietet also eine einfache Möglichkeit, die Maximalität von M nachzuweisen. Dies geht jedoch nicht in allen Graphen, da z.B. der K_4 nur Matchings der Größe ≤ 2 und Knotenüberdeckungen der Größe ≥ 3 hat.

Definition 4.7. *Sei $G = (V, E)$ ein Graph. Eine Menge $S = \{v_1, \ldots, v_k, V_1, \ldots, V_\ell\}$ von Knoten $v_1, \ldots, v_k \in V$ und Teilmengen $V_1, \ldots, V_\ell \subseteq V$ heißt **OSC** (engl. odd set cover) in G, falls*

- *es für jede Kante $e \in E$ einen Knoten $v_i \in S$ mit $v_i \in e$ oder eine Menge $V_j \in S$ mit $e \subseteq V_j$ gibt und*
- *alle Mengen $V_j \in S$ eine ungerade Größe $n_j = |V_j|$ haben.*

*Das **Gewicht** von S ist $w(S) = k + \sum_{j=1}^{\ell}(n_j - 1)/2$.*

Im Fall $\ell = 0$ ist $S = \{v_1, \ldots, v_k\}$ also eine Knotenüberdeckung in G (oder kurz VC für engl. vertex cover).

Lemma 4.8. *Für jedes Matching M in einem Graphen $G = (V, E)$ und jede OSC $S = \{v_1, \ldots, v_k, V_1, \ldots, V_\ell\}$ in G gilt $|M| \leq w(S)$.*

Beweis. M kann für jeden Knoten $v_i \in S$ höchstens eine Kante e mit $v_i \in e$ und für jede Menge $V_j \in S$ höchstens $(n_j - 1)/2$ Kanten $e \subseteq V_j$ enthalten. ∎

Satz 4.9. *Der Algorithmus von Edmonds berechnet ein maximales Matching M für $G = (V, E)$.*

Beweis. Der Algorithmus gibt M aus, sobald die Prozedur `FindePfad` die Suche nach einem M-Pfad erfolglos abbricht. Wir analysieren die Struktur des Suchwalds $W = (V_W, E_W)$ zu diesem Zeitpunkt und gewinnen aus W eine OSC S für G mit $w(S) = |M|$.

Sei $V_0 \subseteq V_W$ die Menge der geraden und $V_1 = \{u_1, \ldots, u_k\} \subseteq V_W$ die Menge der ungeraden Knoten in W. Weiter seien $b_1, \ldots, b_\ell \in V_0$ die Knoten, zu denen alle Knoten in den gefundenen Blüten kontrahiert wurden, und für $j = 1, \ldots, \ell$ sei $C_j \subseteq V$ die Menge aller Knoten, die zu b_j kontrahiert wurden (d.h. von den Knoten in C_j ist nur noch b_j

in V_W vorhanden). Zudem sei $V_2 = V_0 \cup C_1 \cup \cdots \cup C_\ell \subseteq V$ die Menge aller geraden und zu einem geraden Knoten kontrahierten Knoten.

Zunächst überlegen wir uns, dass es in E keine Kante $\{u, v\}$ zwischen V_2 und V_u geben kann. Dies liegt daran, dass der Algorithmus im Fall $u \in V_2$ eine Kante (u', v) mit $u' \in V_2$ (wobei $u' = u$ ist oder u' zur selben Menge C_j wie u gehört) zu Q hinzufügen und somit v nach dem Entfernen der Kante (u', v) aus Q ungerade werden würde.

Zudem muss jede Kante $e \in E$, die beide Endpunkte in V_2 hat, in einer der Mengen C_i liegen, da sonst E_W eine Kante (u, v) mit $u, v \in V_0$ enthalten würde, die nach dem Entfernen aus Q entweder zu einer weiteren Blüte oder zu einem M-Pfad führen würde.

Folglich muss jede Kante $e \in E$ entweder

- einen ungeraden Endpunkt haben (d.h. $e \cap V_1 \neq \emptyset$) oder
- komplett in einer Menge C_j liegen (d.h. $\exists j : e \in C_j$) oder
- zwei unerreichte Knoten verbinden (d.h. $e \subseteq V_u$).

Daher können wir wie folgt eine OSC S für G konstruieren. Wir fügen alle ungeraden Knoten $u_1, \ldots, u_k$ und alle Mengen $C_1, \ldots, C_\ell$ zu S hinzu. Man beachte, dass die Mengen C_j eine ungerade Größe $n_j = |C_j|$ haben, da C_j durch eine Folge von Kontraktionen auf die Menge $\{b_j\}$ verkleinert wird und bei jeder Kontraktion gerade viele Knoten aus C_j verschwinden.

Zudem fügen wir im Fall, dass $V_u \neq \emptyset$ ist, einen beliebigen Knoten $u_0 \in V_u$ als Einzelknoten und im Fall, dass $|V_u| \geq 4$ ist, auch noch die Menge $C_0 = V_u \setminus \{u_0\}$ zu S hinzu. Man beachte, dass $|V_u|$ gerade und somit $n_0 = |C_0|$ ungerade ist, da alle Knoten in V_u durch eine Matchingkante $e \subseteq V_u$ gebunden sind.

Dann ist S eine OSC für G, da jede Kante $e \in E$ entweder einen Endpunkt in S hat oder von einer der Mengen $C_j \in S$ überdeckt wird. Zudem ist $w(S) = |M|$, da sich M in $|S|$ Mengen $M_i = \{e \in M \mid u_i \in e\}$ und $M'_j = \{e \in M \mid e \subseteq C_j\}$ zerlegen lässt mit $|M_i| = 1$ und $|M'_j| = (n_j - 1)/2$. ■

Der Algorithmus von Edmonds lässt sich leicht dahingehend modifizieren, dass er zusammen mit dem berechneten Matching M eine OSC S mit $w(S) = |M|$ zum Nachweis der Maximalität von M ausgibt.

Korollar 4.10. *Für jeden Graphen G gilt*

$$\mu(G) = \min\{w(S) \mid S \text{ ist eine OSC in } G\}.$$

Korollar 4.11 (Satz von König)**.** *Für bipartite Graphen G gilt*

$$\mu(G) = \min\{|C| \mid C \text{ ist eine Knotenüberdeckung in } G\}.$$

Zudem lässt sich eine (kleinste) Knotenüberdeckung C der Größe $|C| = \mu(G)$ in Zeit $O(m\sqrt{\mu(G)})$ berechnen.

Beweis. Sei $G = (A, B, E)$ und sei $W = (V_W, E_W)$ der Suchwald zum Zeitpunkt, wenn der Algorithmus von Edmonds die Suche nach einem M-Pfad erfolglos abbricht. Da G bipartit ist, werden keine Blüten gefunden. Daher hat jede Kante $e \in E$ entweder einen ungeraden Endpunkt oder ist in V_u enthalten und somit einen Endpunkt in $V_u \cap A$. Folglich ist $C = V_1 \cup (V_u \cap A)$ eine VC für G. Zudem gilt $|C| = |M|$, da jede Matchingkante genau einen Endpunkt in C hat.

Wir berechnen zuerst in Zeit $O(m\sqrt{\mu})$ mit dem Algorithmus von Dinitz ein maximales Matching M für G, starten danach die Prozedur `FindePfad`(G, M) zum Aufbau des Suchwalds W in Zeit $O(n + m)$ und geben C aus. ■

5 Baum- und Pfadweite

Definition 5.1. *Sei $G = (V, E)$ ein Graph.*

a) Eine ***Baumzerlegung*** *(kurz* ***TD*** *für tree decomposition) von G ist ein Tripel (V_T, E_T, X), wobei $T = (V_T, E_T)$ ein Baum ist und $X : V_T \to \mathcal{P}(V) \setminus \{\emptyset\}$ die folgenden 3 Eigenschaften erfüllt (für (V_T, E_T, X) schreiben wir meist (T, X) und für $X(t)$ meist X_t).*

- *Es gilt $V = \bigcup_{t \in V_T} X_t$ (die Mengen $X_t \subseteq V$ heißen* ***Taschen****).*
- *Für jede Kante $e \in E$ gibt es eine Tasche X_t mit $e \subseteq X_t$.*
- *Für jeden Knoten $u \in V$ ist der induzierte Teilgraph $T[X^{-1}(u)]$ von T zusammenhängend (also ein Teilbaum), wobei $X^{-1}(u) = \{t \in V_T \mid u \in X_t\}$ ist.*

b) Die ***Weite*** *von (T, X) ist $w(T, X) = \max_{t \in V_T} |X_t| - 1$.*

c) Die ***Baumweite*** *$tw(G)$ von G ist die kleinste Weite aller möglichen Baumzerlegungen von G.*

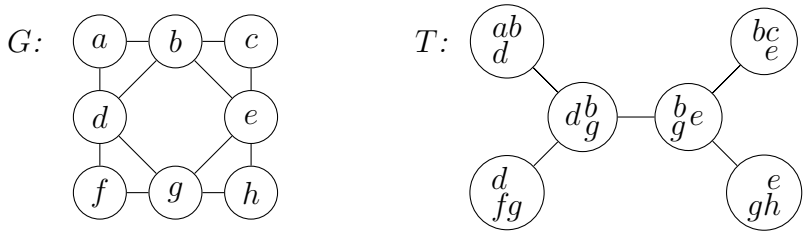
d) Eine Baumzerlegung (T, X) von G heißt ***Pfadzerlegung*** *(kurz* ***PD*** *für path decomposition), wenn T ein Pfad ist. Die* ***Pfadweite*** *$pw(G)$ von G ist die kleinste Weite aller möglichen Pfadzerlegungen von G.*

e) $TW(k) := \{G \mid tw(G) \leq k\}$ und $PW(k) := \{G \mid pw(G) \leq k\}$.

Beispiel 5.2. *(i) Der leere Graph $E_n = (V, \emptyset)$ hat Baum- und Pfadweite $tw(E_n) = pw(E_n) = 0$. Wir generieren für jeden Knoten $u \in V$ eine Tasche $X_u = \{u\}$, die nur diesen Knoten enthält, und verbinden diese Taschen in beliebiger Reihenfolge zu einem Pfad. Umgekehrt muss die Kantenmenge jedes Graphen G mit $tw(G) = 0$ leer sein, da jede Tasche nur einen Knoten enthält, d.h. $TW(0)$ besteht aus allen leeren Graphen.*

(ii) Jeder Baum $G = (V, E)$ hat eine Baumweite $tw(G) \leq 1$. Z.B. hat die TD (T, X) mit $V_T = E \cup \big\{\{u\} \;\big|\; u \in V\big\}$, $X_t = t$ für $t \in V_T$ und $E_T = \{\{s, t\} \mid s \subset t\}$ die Weite $w(T, X) \leq 1$.

(iii) Folgender Graph G hat eine Baumzerlegung (T, X) der Weite 2:

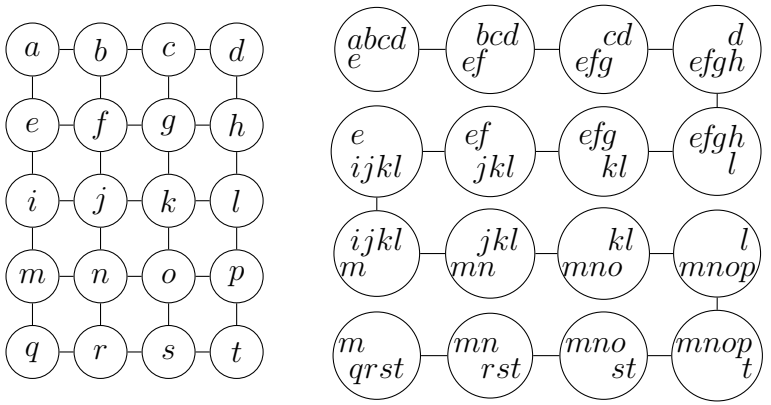


Der Baum $T = (\{1, \ldots, 6\}, E_T)$ verbindet die Taschen $X_1 = \{a, b, d\}$, $X_2 = \{b, d, g\}$, $X_3 = \{b, e, g\}$, $X_4 = \{b, e, c\}$, $X_5 = \{e, g, h\}$, $X_6 = \{d, f, g\}$ durch die 5 Kanten $\{1, 2\}$, $\{2, 3\}$, $\{3, 4\}$, $\{3, 5\}$ und $\{2, 6\}$.

(iv) Für den Gittergraphen $G_{k \times \ell} = P_k \times P_\ell$ mit $k\ell$ Knoten gilt

$$tw(G_{k \times \ell}) \leq pw(G_{k \times \ell}) \leq \min\{k, \ell\}.$$

Der Graph $G_{5 \times 4}$ hat bspw. folgende Pfadzerlegung der Weite 4:



◁

Proposition 5.3. *Sei $G = (V, E)$ ein Graph und sei $H = (V', E')$ ein Teilgraph von G oder $H = G_{uv}$ für eine Kante $\{u, v\} \in E$. Dann gilt $tw(H) \leq tw(G)$ und $pw(G_{uv}) \leq pw(G)$.*

Beweis. Sei (T, X) eine Baum- bzw. Pfadzerlegung von G. Dann ist (T, X') mit $X'_t = X_t \cap V'$ eine Baum- bzw. Pfadzerlegung von (V', E') mit $w(T, X') \leq w(T, X)$. Zudem ist (T, X'') mit

$$X''_t = \begin{cases} X_t, & v \notin X_t \\ (X_t \setminus \{v\}) \cup \{u\}, & v \in X_t \end{cases}$$

eine Baum- bzw. Pfadzerlegung von G_{uv} mit $w(T, X'') \leq w(T, X)$. ■

Korollar 5.4. *Die Graphklassen $TW(k)$ und $PW(k)$ sind unter Minorenbildung abgeschlossen.*

Definition 5.5. *Eine Baumzerlegung (T, X) heißt **kompakt**, wenn alle Taschen paarweise unvergleichbar sind, d.h. für alle $s \neq t \in V_T$ gilt $X_s \not\subseteq X_t$ und $X_t \not\subseteq X_s$.*

Da im Fall $X_s \subseteq X_t$ aufgrund der Definition von Baumzerlegungen auch $X_s \subseteq X_{t'}$ für alle Knoten t' auf dem Pfad von s nach t gilt, ist (T, X) genau dann kompakt wenn für alle $\{s, t\} \in E_T$ die Bedingungen $X_s \not\subseteq X_t$ und $X_t \not\subseteq X_s$ gelten. Es genügt also, die Taschen von benachbarten Baumknoten zu vergleichen.

Proposition 5.6. *Sei $G = (V, E)$ ein Graph.*

(i) Jede Baumzerlegung (T, X) von G lässt sich effizient in eine kompakte Baumzerlegung (T', X') von G transformieren, die nur Taschen aus X enthält.

(ii) Für jede kompakte Baumzerlegung (T, X) von G gilt $n(T) \leq n(G)$.

Beweis.

(i) Der Algorithmus durchsucht T ausgehend von einem beliebigen Blatt. Dabei prüft er für jede Kante $\{s, t\} \in E_T$ (wobei s näher am Startknoten liege als t), ob $X_s \subseteq X_t$ oder $X_s \supseteq X_t$ gilt. Falls ja, kontrahiert er die Kante $\{s, t\}$ zu s und ersetzt X_s durch $X_s \cup X_t \in \{X_s, X_t\}$. Anschließend setzt er die Suche im resultierenden Baum T_{st} mit s als aktuellem Knoten fort.

Dieser Algorithmus benötigt $O\Big(w(T, X) \cdot n(T)\Big)$ Zeit, um (T, X) in eine kompakte Baumzerlegung zu transformieren. Ist T ein Pfad, so liefert der Algorithmus eine kompakte Pfadzerlegung.

(ii) Sei (T, X) eine kompakte Baumzerlegung von $G = (V, E)$. Wir definieren eine injektive Abbildung $f : V_T \to V$, die jedem Baumknoten $t \in V_T$ einen Knoten $v_t \in X_t$ zuordnet. Beginnend mit $D = \emptyset$ wählen wir ein beliebiges Blatt t in dem Baum $T - D$. Da (T, X) kompakt ist, enthält die Tasche X_t einen Knoten v_t, der nicht in der Tasche X_s des Nachbars s von t in $T - D$ vorkommt. Da dann v_t auch in keiner anderen Tasche vorkommen kann (sonst wäre $T[X^{-1}(v_t)]$ nicht zusammenhängend), setzen wir $f(t) := v_t$ und fügen t zu D hinzu. Dies führen wir solange fort, bis $D = V_T$, also f für alle $t \in V_T$ definiert ist. ■