Netzwerkflüsse

Literatur:

- L. R. Ford Jr. and D. R. Fulkerson, Flows in Networks, Princeton University Press, 1962.
- E. L. Lawler, Combinatorial Optimization: Networks and Matroids, Hoet, Rinehart and Winston, 1976.
- T. C. Hu, Combinatorial Algorithms, Addison-Wesley, 1982.
- R. K. Ahuja, T. L. Magnanti, J. B. Orlin, Network Flows: Theory, Algorithm and Applications, Prentice Hall, 1993.
- T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, 2nd. Ed., MIT Press, 2001.
- J. Kleinberg, E. Tardos, Algorithm Design, Addison Wesley, 2006.

0. Motivation und graphentheoretische Grundlagen

Ein gerichteter Graph $G = (V, E)$ besteht aus einer endlichen, nichtleeren Menge $V := \{v_1, v_2, \dots, v_n\}$ von Knoten und einer Menge $E \subseteq V \times V$ von Kanten. Sei $e := (v, w) \in E$. Dann heißt v Anfangsknoten und w Endknoten von e. e heißt eingehende Kante von w und ausgehende Kante von v. Der Eingangsgrad $in(v)$ (Ausgangsgrad $out(v)$) eines Knotens v ist die Anzahl der eingehenden (ausgehenden) Kanten von v.

Bezeichnungen:

$$IN(v) := \{ e \in E \mid e = (w, v),\ w \in V \}$$
$$OUT(v) := \{ e \in E \mid e = (v, w),\ w \in V \}$$
$$\Gamma^-(v) := \{ w \in V \mid (w, v) \in IN(v) \}$$
$$\Gamma^+(v) := \{ w \in V \mid (v, w) \in OUT(v) \}.$$

Seien $v, w \in V$. Ein Pfad (oder auch Weg) P von v nach w ist eine Folge

$$P := v_0, (v_0, v_1), v_1, (v_1, v_2), v_2, \dots, v_{k-1}, (v_{k-1}, v_k), v_k$$

von Knoten und Kanten, so dass

i) $v_0 = v$, $v_k = w$ und

ii) $(v_i, v_{i+1}) \in E$, $0 \le i < k$.

Wir repräsentieren einen Pfad häufig nur durch seine Knoten; d.h., wir schreiben

$P = v_0, v_1, v_2, \ldots, v_{k-1}, v_k$.

Falls ein Pfad vom Knoten v nach w existiert, dann heißt v Vorgänger von w und w Nachfolger von v. Falls $(v,w) \in E$, dann heißt v direkter Vorgänger von w und w direkter Nachfolger von v.

Ein Graph $G = (V, E)$ heißt zyklisch, falls Knoten $v, w \in V$, $v \neq w$ existieren, so dass G einen Pfad von v nach w und einen Pfad von w nach v enthält. Falls keine derartige Knoten existieren, dann heißt G azyklisch.

Ein gerichteter Graph $G = (V, E, c, s, t)$ mit Quelle $s \in V$, Senke $t \in V \setminus \{s\}$ und Kapazitätsfunktion $c: E \to \mathbb{R}^+$, die jeder Kante $e = (v,w)$ eine positive reelwertige Kapazität $c(v,w)$ zuordnet, heißt Flussnetzwerk oder Transportnetzwerk. Wir vereinbaren $c(v,w) = 0$ für alle $(v,w) \in (V \times V) \setminus E$.

Ein Fluss $f: V \times V \to \mathbb{R}$ ist eine Funktion, die folgende Bedingungen erfüllt:

1. $f(v,w) \leq c(v,w) \quad \forall (v,w) \in V \times V$

(Kapazitätsbedingung)

2. $\sum_{e \in IN(v)} f(e) = \sum_{e \in OUT(v)} f(e) \quad \forall v \in V \setminus \{s,t\}$

(Kirchhoffsches Gesetz)

Die Größe $|f|$ eines Flusses ist derjenige Fluss in die Senke t, der diese nicht wieder verlässt. D.h.,

$$|f| := \sum_{v \in V} f(v,t) - \sum_{w \in V} f(t,w).$$

Beispiel 1:

Seien $A_1, A_2, \dots, A_p$ und $B_1, B_2, \dots, B_q$ Seehäfen. In den Häfen $A_1, A_2, \dots, A_p$ liegen Bananen zur Verschiffung bereit. Zielhäfen sind $B_1, B_2, \dots, B_q$. Bezeichne für $1 \le i \le p$, $1 \le j \le q$

r_i die im Hafen A_i liegende Quantität von Bananen

d_j die von Hafen B_j angeforderte Quantität an Bananen.

Die Kapazität der Schiffahrtslinie von Hafen A_i zum Hafen B_j ist begrenzt; d.h., maximal $c(A_i, B_j)$ Bananen können von A_i nach B_j verschifft werden.

Fragen:

1. Ist es möglich, alle Anforderungen zu befriedigen?
2. Falls nein, welche Quantität von Bananen kann maximal zu den Zielhäfen gebracht werden?

3. Wie sollen die Bananen verschifft werden.

Antwort:

- Reduktion auf ein Flussproblem.
- Lösen des Flussproblems.

Definition des korrespondierenden Flussnetzwerkes:

$G := (V, E, c, s, t)$, wobei

$V = \{A_1, A_2, \dots, A_p, B_1, B_2, \dots, B_q, s, t\}$

$E = \{(A_i, B_j) \mid 1 \le i \le p,\ 1 \le j \le q\}$
$\cup \{(s, A_i) \mid 1 \le i \le p\} \cup \{(B_j, t) \mid 1 \le j \le q\}.$

$c: E \longrightarrow \mathbb{R}^+$, wobei

$c(A_i, B_j)$ Quantität, die von A_i nach B_j verschifft werden kann

$c(s, A_i) = r_i$

$c(B_j, t) = d_j$

Die Lösung der folgenden Frage beantwortet obige Fragen:

Was ist der maximale Fluss von s nach t in G und wie sieht solcher aus?

1. Der Satz von Menger und das Max-flow Min-Cut Theorem

Wir werden zunächst einige allgemeine graphentheoretische Sätze beweisen. Diese ermöglichen uns dann die Entwicklung von Algorithmen zur Lösung von Flussproblemen.

Beispiel 2:

Ein Verkehrsnetz sei durch einen Graphen $G = (V, E)$ dargestellt. Dabei entsprechen Verkehrsknotenpunkte den Knoten in V und Straßen den Kanten in E.
Seien a und b zwei verschiedene Knotenpunkte.

Ziel:
Einrichtung von möglichst wenigen Verkehrskontrollen, so dass jeder Weg von a nach b in G durch mindestens eine Verkehrskontrolle verläuft.

Der Satz von Menger gibt uns die Antwort: Die Mindestanzahl von Kontrollen ist gleich der Maximalanzahl von kreuzungsfreien Wegen von a nach b in G.

◆

Für die Formulierung und den Beweis des Satzes von Menger benötigen wir noch einige Bezeichnungen.

Jeder Pfad in G mit Endknoten a und b heißt kurz a,b-Pfad. Eine Menge von a,b-Pfaden heißt kreuzungsfrei, wenn diese Pfade bis auf die Endknoten keine Knoten (und somit auch keine Kanten) gemeinsam haben.

Ein System von n kreuzungsfreien a,b-Pfaden heißt maximal, falls es keine n+1 kreuzungsfreie a,b-Pfade in G gibt.

Die Verbindungszahl $w_G(a,b)$ von a und b in G ist die Anzahl der Pfade eines maximalen kreuzungsfreien Systems von a,b-Pfaden.

Sei $T \subseteq V \setminus \{a,b\}$. Wir sagen dann, dass a und b durch T getrennt werden, falls jeder a,b-Pfad in G mindestens einen Knoten aus T enthält.

Die Trennungszahl $\tau_G(a,b)$ von a und b in G ist definiert durch:

$$\tau_G(a,b) = \min \{ |T| \subseteq V \setminus \{a,b\} \mid T \text{ trennt } a,b \}$$

Satz 1.1 (Menger 1927)

Sei $G = (V,E)$ ein Graph. Seien $a,b \in V$, $a \neq b$ und $(a,b) \notin E$. Dann gilt:

$$w_G(a,b) = \tau_G(a,b).$$

Beweis:

Falls es keinen a,b-Pfad in G gibt, dann gilt
$\omega_G(a,b) = \tau_G(a,b) = 0$.

Annahme:
Es existiert mindestens ein a,b-Pfad in G.

Da jedes a,b-trennende T auf jedem Weg eines kreuzungsfreien Wegsystems mindestens einen Knoten besitzt, gilt:

$$\tau_G(a,b) \geq \omega_G(a,b).$$

Also ist noch zu zeigen

$$\omega_G(a,b) \geq \tau_G(a,b).$$

Beweis 1 (Dirac)

Annahme: $\omega_G(a,b) < \tau_G(a,b)$

Im folgenden bezeichne k die minimale Anzahl von Knoten, die a und b trennen.

Sei $k \geq 2$ die kleinste Zahl, für die ein Graph mit $\omega_G(a,b) < \tau_G(a,b)$ existiert.

Sei G_0 für das minimale k ein solcher Graph mit Minimalanzahl von Kanten.

Annahme $\Rightarrow$
Es gibt höchstens $(k-1)$ paarweise kreuzungsfreie a,b-Pfade.

1. Beobachtung:
Es gibt keinen Knoten v, der mit a und b direkt verbunden ist.

Andernfalls wäre v in jeder trennenden Knotenmenge und ein a,b-Pfad, der v enthält in jedem maximalen System von paarweise kreuzungsfreien a,b-Pfaden.

=>
$G_0 \setminus \{v\}$ wäre ein Gegenbeispiel zur Behauptung mit trennender Knotenmenge der Größe $(k-1)$. Dies ist ein Widerspruch zur Minimalität von k.

Sei T eine Menge von k Knoten, die a und b trennt.

2. Beobachtung:

Entweder a oder b ist zu allen Knoten in T benachbart.

Bew. der Beobachtung:

Annahme:
Weder a noch b sind zu allen Knoten in T benachbart.

Betrachte den Graphen G_a, den wir aus G_0 erhalten, indem wir diejenige Komponente von

$G_0 \setminus T$, die a enthält, durch einen Knoten a' ersetzen und a' mit jeden Knoten in T verbinden.

Annahme $\Rightarrow$

Die ersetzte Komponente enthält mindestens eine Kante.

Jeder Knoten in T ist mit mindestens einem Knoten in dieser Komponente verbunden

$\Rightarrow$

G_a enthält weniger Kanten als G_0.

Minimalität von G_0 $\Rightarrow$

In G_a existieren k paarweise kreuzungsfreie a,b - Pfade.

Die Segmente dieser Pfade von b nach T sind derart, dass diese paarweise exakt den Knoten b gemeinsam haben. Insbesondere gilt für jedes $w \in T$, dass einer dieser Pfade ein w,b - Pfad ist.

Eine analoge Anwendung der obigen Prozedur auf b anstatt a liefert k kreuzungsfreie a,w - Pfade, $w \in T$.

Diese beide Mengen von Pfaden zusammengesetzt liefern k paarweise kreuzungsfreie a,b - Pfade in G_0.

$\Rightarrow$

$$w_{G_0}(a,b) \geq k = \lambda_{G_0}(a,b)$$

Dies ist ein Widerspruch zu $w_{G_0}(a,b) < \lambda_{G_0}(a,b)$.

□

Betrachte einen kürzesten a,b-Pfad

$$P = a, x_1, x_2, \dots, x_\ell, b.$$

1. Beobachtung $\Rightarrow$ $\ell \geq 2$.

Minimalität von G_0 $\Rightarrow$

$G_0 \setminus \{(x_1,x_2)\}$ enthält eine a,b-trennende Knotenmenge T_0 mit $|T_0| = k-1$

Dann gilt:

$T_1 := T_0 \cup \{x_1\}$ und $T_2 := T_0 \cup \{x_2\}$

trennen a und b in G_0.

Da P ein kürzester a,b-Pfad ist, gilt:

i) b ist nicht zu x_1 benachbart und
ii) a ist nicht zu x_2 benachbart.

i) und Beobachtung 2 $\Rightarrow$

Alle Knoten in T_0 sind zu a benachbart.

ii) und Beobachtung 2 $\Rightarrow$

Alle Knoten in T_0 sind zu b benachbart.

Wegen $|T_{01}| = k-1 \geq 1$ haben a und b mindestens einen gemeinsamen Nachbarn.

Dies ist ein Widerspruch zur 1. Beobachtung. ∎

Sei $S \subseteq V$. $(S, V\setminus S)$ heißt Schnitt von G. Ein Schnitt $(S, V\setminus S)$ separiert s und t, falls $s \in S$ und $t \in V\setminus S$. Sei

$$E(S, V\setminus S) = \{ (x,y) \in E \mid x \in S, y \in V\setminus S \}.$$

Die Kapazität $c(S, V\setminus S)$ eines Schnittes $(S, V\setminus S)$ ist definiert durch

$$c(S, V\setminus S) = \sum_{(x,y) \in E(S, V\setminus S)} c(x,y).$$

Offensichtlich gilt:

$$\max \{ |f| \mid f \text{ Fluss von } s \text{ nach } t \} \leq \min \{ c(S, V\setminus S) \mid (S, V\setminus S) \text{ ist } s,t\text{-separierend} \}$$

Satz 1.2 (Max-Flow Min-Cut Theorem)

Sei $G = (V, E, c, s, t)$ ein Flussnetzwerk. Dann ist der maximale Flusswert in G gleich dem Minimum der Kapazitäten der s und t separierenden Schnitte.

D.h.,

$$\max \{ |f| \mid f \text{ ist Fluss in } G \} = \min \{ c(S, V\setminus S) \mid (S, V\setminus S) \text{ ist } s,t\text{-separierend} \}.$$

Beweis:

Sei $F_{max} = \max \{ |f| \mid f \text{ ist Fluss in } G \}$.

Offensichtlich gilt

$F_{max} \leq \min \{ c(S, V \setminus S) \mid (S, V \setminus S) \text{ ist } s,t\text{-separierend} \}$.

2.2. $\exists$ Schnitt $(S, V \setminus S)$ mit $c(S, V \setminus S) = F_{max}$.

Hierzu geben wir eine einfache Prozedur an, die, gegeben einen Fluss f mit $|f| = F_{max}$, einen Schnitt $(S, V \setminus S)$ mit $c(S, V \setminus S) = F_{max}$ konstruiert. Wir starten mit $S = \{s\}$. In jedem Schritt erweitern wir S um einen Knoten $y \in V \setminus S$, den wir zu S hinzufügen müssen, damit die Behauptung überhaupt erfüllt sein kann. Diese Idee wird durch folgendes Programmstück realisiert:

(1) $S := \{s\}$;

(2) <u>while</u> $\exists\, x \in S, y \in V \setminus S$ mit

$c(x,y) > f(x,y)$ oder $f(y,x) > 0$

<u>do</u>

$S := S \cup \{y\}$

<u>od</u>.

<u>Behauptung</u>: $(S, V \setminus S)$ ist s,t-separierend.

<u>Bew. d. Beh.</u>:

<u>Annahme</u>: $t \in S$.

Dann gibt es einen Knoten $x_{e-1} \in S$, der dafür verantwortlich war, dass t zu S hinzugenommen wurde. D.h.,

$$c(x_{e-1}, t) > f(x_{e-1}, t) \quad \text{oder} \quad f(t, x_{e-1}) > 0.$$

Ebenso gibt es einen Knoten $x_{e-2} \in S$, der dafür verantwortlich war, dass x_{e-1} zu S hinzugenommen wurde, oder $x_{e-1} = s$ usw..

Also existiert ein ungerichteter Pfad

$$P = s = x_0, x_1, x_2, \dots, x_e = t$$

mit $x_i \in S$ für $0 \leq i \leq e$.

Für $0 \leq i < e$ sei

$$\varepsilon_i = \max \{ c(x_i, x_{i+1}) - f(x_i, x_{i+1}), f(x_{i+1}, x_i) \}.$$

Konstruktion $\Rightarrow$ $\varepsilon_i > 0$ $\forall i$.

Sei

$$\varepsilon := \min_{0 \leq i < e} \varepsilon_i.$$

<u>Ziel:</u>
Konstruktion eines Flusses f^* mit $|f^*| = F_{max} + \varepsilon$.

<u>Durchführung:</u>
Ändere den Fluss auf den ungerichteten Kanten (x_i, x_{i+1}), $0 \leq i < e$ wie folgt ab:

$$\begin{cases} f^*(x_i, x_{i+1}) := f(x_i, x_{i+1}) + \varepsilon & \text{falls } (x_i, x_{i+1}) \in P \\ f^*(x_{i+1}, x_i) := f(x_{i+1}, x_i) - \varepsilon & \text{falls } (x_{i+1}, x_i) \in P \end{cases}$$

Wahl von ε $\Rightarrow$

f^* verletzt die Kapazitätsbedingung nicht und keine Kante hat negativen Fluss.

Um einzusehen, dass auch das Kirchhoffsche Gesetz erfüllt bleibt, betrachte einen beliebigen Knoten x_i, $0 < i < \ell$. Es gibt folgende Möglichkeiten für die Flussänderungen auf den Kanten mit Endknoten x_i auf P:

In allen vier Fällen folgt aus der Gültigkeit des Kirchhoffschen Gesetzes vor der Flussänderung auch dessen Gültigkeit danach.

$\Rightarrow$

f^* ist ein Fluss.

Ferner gilt:

$$|f^*| = \sum_{v \in V} f^*(v, t)$$

$$= \sum_{v \in V \setminus \{x_{\ell-1}\}} f(v, t) + f^*(x_{\ell-1}, t)$$

$$= \sum_{v \in V \setminus \{x_{e-1}\}} f(v,t) \quad + f(x_{e-1},t) + \varepsilon$$

$$= |f| + \varepsilon$$

$$= F_{max} + \varepsilon.$$

Dies ist ein Widerspruch zur Maximalität von f.

$\Rightarrow$

$$t \notin S.$$

$\Rightarrow$

$(S, V \setminus S)$ ist s,t-Separierend.

Konstruktion $\Rightarrow$

$\forall x \in S, y \in V \setminus S$ gilt:

$$f(x,y) = c(x,y) \quad \text{und} \quad f(y,x) = 0.$$

Also gilt:

$$|f| = \sum_{x \in S, y \in V \setminus S} f(x,y) = \sum_{x \in S, y \in V \setminus S} c(x,y) = c(S, V \setminus S)$$

Als nächstes werden wir eine Knotenversion des Max-Flow Min-Cut Theorems herleiten. Hierzu ordnen wir den Knoten $v \in V \setminus \{s,t\}$ eine Kapazität $c(v)$ zu. Für einen Fluss f muss dann $\forall v \in V \setminus \{s,t\}$ folgendes erfüllt sein:

$$\sum_{w \in \Gamma^-(v)} f(w,v) = \sum_{u \in \Gamma^+(v)} f(v,u) \leq c(v).$$

$S \subseteq V \setminus \{s,t\}$ heißt <u>s,t-separierender Knotenschnitt</u>, falls in $G \setminus S$ kein positiver Fluss von s nach t möglich ist.

<u>Satz 1.3</u>

Sei $G = (V, E, c, s, t)$ ein gerichteter Graph mit einer Kapazitätsfunktion $c: V \setminus \{s,t\} \to \mathbb{R}^+$, einer Quelle s und einer Senke t. Ferner sei $\Gamma^-(s) = \Gamma^+(t) = \emptyset$. Dann gilt:

$$\max\{|f| \mid f \text{ Fluss in } G\} = \min\left\{\sum_{v \in T} c(v) \mid T \subseteq V \setminus \{s,t\} \text{ ist } s,t\text{-separierend}\right\}$$

<u>Beweis:</u>

Wir führen die Knotenversion auf die Kantenversion des Max-Flow Min-Cut Theorems zurück.

Hierzu konstruieren wir aus $G = (V, E, c, s, t)$ ein Flussnetzwerk $G' = (V', E', c', s, t)$ durch:

- Ersetze jeden Knoten $x \in V$ durch $(x_-) — (x_+)$.
- Jede eingehende Kante von x wird eingehende Kante von x_- und jede ausgehende Kante von x wird ausgehende Kante von x_+. Diese Kanten erhalten die Kapazität ∞.
- Die Kante (x_-, x_+) erhält die Kapazität $c'(x_-, x_+) := c(x)$.

Mit Hilfe des Max-Flow Min-Cut Theorems kann nun der Satz leicht bewiesen werden.

Übung:

Arbeiten Sie den Beweis von Satz 1.3 aus.

Der im Beweis des Max-Flow Min-Cut Theorems konstruierte Pfad P heißt augmentierender Pfad. Im Fall, dass der vorliegende Fluss nicht maximal ist, gibt uns der Beweis des Max-Flow Min-Cut Theorems eine Methode zur Konstruktion eines augmentierenden Pfades. Mit Hilfe dieses Pfades kann dann der Fluss vergrößert werden. Dieses Vorgehen heißt Augmentierungsschritt. Somit erhalten wir folgenden Algorithmus zur Konstruktion eines maximalen Flusses in einem Flussnetzwerk:

Algorithmus FORD-FULKERSON

Eingabe: Flussnetzwerk $G = (V, E, c, s, t)$
Ausgabe: maximaler Fluss
Methode:

(1) Starte mit dem Nullfluss f_0. D.h.,
$f_0(x,y) = 0 \quad \forall (x,y) \in E$.

(2) $f := f_0$; gefunden := false;

(3) while ¬ gefunden
do
konstruiere wie im Beweis des Max-Flow Min-Cut Theorems den Schnitt $(S, V \setminus S)$ bezüglich des aktuellen Flusses f;

if $t \in S$
then
Vergrößere, wie im Beweis des Max-Flow Min-Cut Theorems beschrieben, den Fluss f um ε
else
gefunden := true
fi
od.

Ford und Fulkerson haben anhand eines Beispiels gezeigt, dass bei irrationalen Kapazitäten obiges Verfahren möglicherweise nicht terminiert. Jedoch zeigt folgender Satz, dass, falls alle Kapazitäten ganzzahlig sind, dies nicht der Fall ist.

Satz 1.4 (Integrality Theorem)

Sei $G = (V, E, c, s, t)$ ein Flussnetzwerk, in dem alle Kapazitäten ganzzahlig sind. Dann terminiert Ford und Fulkersons Methode nach maximal $\sum_{(x,y) \in E} c(x,y)$ Augmentierungsschritten mit einem ganzzahligen maximalen Fluss.

Beweis:
Da alle Kapazitäten ganzzahlig sind und wir mit dem Nullfluss starten, ist während der Durchführung des obigen Algorithmus $|f|$ stets ganzzahlig. Des Weiteren erhöht ein Augmentierungs-

schnitt die Größe des Flusses mindestens um eins. Somit folgt die Behauptung aus

$$F_{max} \leq \sum_{(x,y) \in E} c(x,y).$$

∎

Übung:
Zeigen Sie, dass bei rationalen Kapazitäten der Algorithmus FORD-FULKERSON terminiert.

Als nächstes führen wir den Satz von Menger auf die Knotenversion des Max-Flow Min-Cut Theorems zurück.

Beweis 2 (Satz von Menger):

Gib jedem Knoten $v \in V \setminus \{a,b\}$ die Kapazität 1.

Satz 1.3 $\Rightarrow$

$$\max\{|f| \mid f \text{ Fluss in } G\} = \min\left\{\sum_{v \in T} c(v) \mid T \text{ trennt } a,b\right\}.$$

Satz 1.4 $\Rightarrow$
Es gibt einen maximalen Fluss f_{max}, der über jede Kante e entweder den Fluss 0 oder den Fluss 1 schickt.

$\Rightarrow$

$|f_{max}|$ = maximale Anzahl der kreuzungsfreie a,b-Pfade.

Wahl der Knotenkapazitäten ⇒ Behauptung.

Übung:
Entwickeln Sie einen Algorithmus, der, gegeben einen Graphen $G = (V, E)$ und zwei Knoten $a, b \in V$, eine minimale a,b-trennende Knotenmenge berechnet.

Auch bei ganzzahligen Kapazitäten kann Ford und Fulkersons Methode viele Augmentierungsschritte benötigen.

Ziel:
Entwicklung von Strategien zur Auswahl des augmentierenden Pfades, so dass die Methode von Ford und Fulkerson unabhängig von der Größe der Kapazitäten effizient ist.

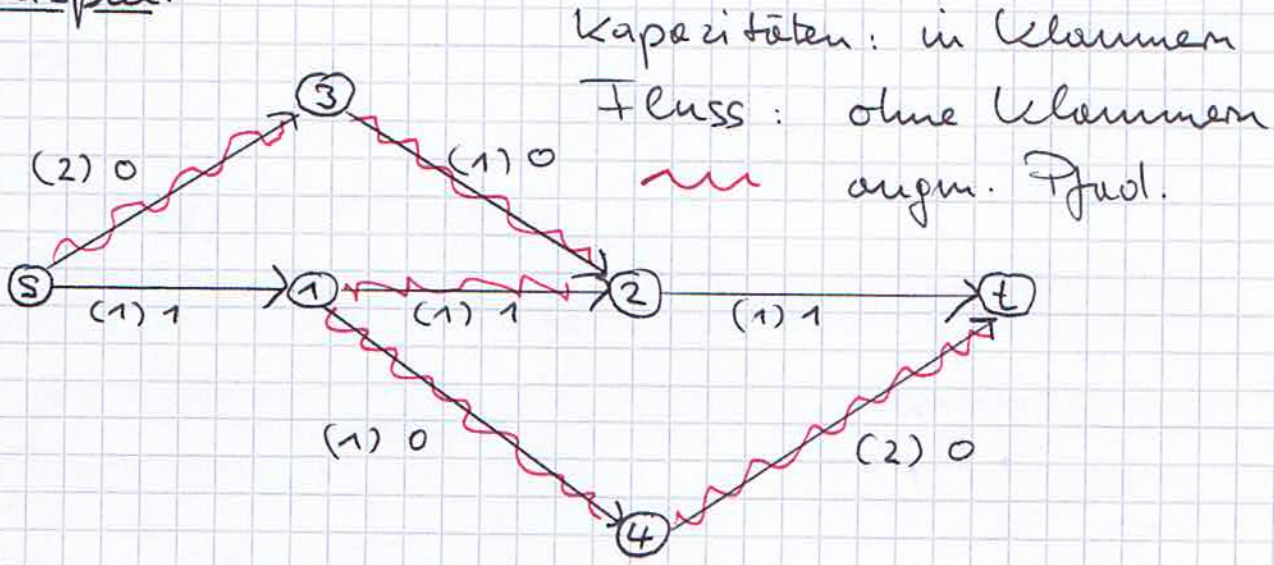
Präzisierung augmentierender Pfad:

Sei $G = (V, E, c, s, t)$ ein Flussnetzwerk und f ein Fluss von s nach t in G. Sei G' derjenige ungerichtete Graph, den wir aus G erhalten, indem wir die Kantenrichtungen ignorieren.
Ein augmentierender Pfad P ist ein Pfad von s nach t in G', so dass

1. $f(x,y) < c(x,y)$ für jede Kante $(x,y) \in E$, die auf P in Vorwärtsrichtung liegt (Vorwärtskante) und

2. $f(y,x) > 0$ für jede Kante $(y,x) \in E$, die auf P in umgekehrter Richtung liegt (Rückwärts-Kante).

Beispiel:



Satz 1.5

Sei $G = (V, E, c, s, t)$ ein Flussnetzwerk. Vom Nullfluss ausgehend gibt es immer eine Möglichkeit, einen maximalen Fluss in höchstens $|E|$ Augmentierungsschritten, die den Fluss jeweils entlang eines einzelnen Pfades im originalen Graphen vergrößern, zu konstruieren.

Beweis:

Sei f^* ein maximaler Fluss. Sei G^* derjenige Teilgraph von G, der durch die Kanten $(v,w) \in E$ mit $f^*(v,w) > 0$ induziert wird. Folgender Algorithmus konstruiert von G^* ausgehend diejenige Pfade, entlang denen augmentiert wird:

(1) $i := 1$

(2) Wiederhole folgenden Schritt solange, bis t in G^* von s nicht mehr erreichbar ist.

Pfadfindungsschritt

(i) Finde einen Pfad P_i von s nach t in G^*.
$\Delta_i := \min_{(v,w) \in P_i} f^*(v,w)$.

(ii) Für alle $(v,w) \in P_i$ tue $f^*(v,w) := f^*(v,w) - \Delta_i$;
Streiche (v,w) aus G^*, sobald $f^*(v,w) = 0$.

(iii) $i := i+1$.

Jeder Pfadfindungsschritt streicht mindestens eine Kante aus G_1^*. Also hat der Algorithmus nach maximal $m := |E|$ Schritten f^* zum Nullfluss reduziert und hält an. Sei k die Anzahl der vom Algorithmus konstruierten Pfade.

Beginnend mit dem Nullfluss erhöhen wir nun sukzessive den Fluss um

Δ_1 entlang den Kanten auf P_1
Δ_2 " " " " P_2
⋮
Δ_k " " " " P_k

Wir erhalten auf diese Art und Weise f^* mittels $k \leq m$ Augmentierungsschritten.

2. Effiziente Algorithmen zur Berechnung eines maximalen Flusses

2.1 Dinic's Methode (1970)

Ziel:
Entwicklung von Regeln, die die Auswahl derart steuern, so dass Ford und Fulkersons Methode nach dem Augmentieren von "wenigen" augmentierenden Pfaden terminiert

1. Regel (Edmonds und Karp 1969)

Augmentiere stets einen (beliebigen), bezüglich der Anzahl der Kanten, kürzesten augmentierenden Pfad.

Resultat: $G_R = (V, E, c, s, t)$, $|V| = n$, $|E| = m$

$\leq \frac{m \cdot n}{2}$ Augmentierungen genügen

$\rightsquigarrow O(m^2 \cdot n)$ - Algorithmus.

2. Regel (Dinic 1970)

Betrachte gleichzeitig alle kürzesten augmentierenden Pfade und augmentiere solange solche, bis nur noch längere augmentierende Pfade existieren.

Für die Entwicklung des Verfahrens von Dinic benötigen wir zunächst einige Bezeichnungen:

Ein Fluss f in einem Flussnetwerk $G=(V,E,c,s,t)$ heißt <u>blockierend</u>, falls auf jedem Pfad von s nach t eine Kante (x,y) mit $f(x,y)=c(x,y)$ liegt. Eine derartige Kante heißt <u>saturiert</u>. Eine Kante $e \in E$ heißt <u>nützlich von x nach y</u>, falls $e=(x,y)$ und $f(e) < c(x,y)$ oder $e=(y,x)$ und $f(e) > 0$.

Dinics Algorithmus arbeitet in Phasen. In jeder Phase konstruiert dieser zunächst aus dem gegebenen Flussnetzwerk G bei vorliegendem Fluss $f: E \to \mathbb{R}$ ein <u>geschichtetes</u> Flussnetzwerk G_f, das genau die kürzesten augmentierenden Pfade enthält. Danach konstruiert der Algorithmus im geschichteten Netzwerk einen blockierenden Fluss.

<u>Konstruktion von G_f:</u>

(1) $V_0 := \{s\}$; $i := 0$;

(2) $H := \{v \in V \setminus (\bigcup_{j \le i} V_j) \mid \exists u \in V_i$ mit (u,v) (oder (v,u)) ist nützlich von u nach $v\}$.

(3) <u>if</u> $H = \emptyset$
<u>then</u>
HALT
<u>fi</u>;

(4) <u>if</u> $t \in H$
<u>then</u>
$\ell := i+1$; $V_\ell := \{t\}$; HALT
<u>else</u>
$V_{i+1} := H$; $i := i+1$; goto (2)
fi.

Lemma 2.1

Falls obiger Algorithmus in Schritt 3 terminiert, dann ist der aktuelle Fluss f maximal.

Beweis:

Sei $S = \bigcup_{j=0}^{i} V_j$.

Konstruktion $\Rightarrow s \in S$ und $t \in V \setminus S$.

$\Rightarrow$

$(S, V \setminus S)$ ist ein s,t-separierender Schnitt.

Alle Kanten von S nach $V \setminus S$ sind saturiert und alle Kanten von $V \setminus S$ nach S haben Fluss 0.

$\Rightarrow$

$$|f| = c(S, V \setminus S).$$

Also folgt die Behauptung aus dem Max-Flow Min-Cut Theorem. ∎

Für $1 \leq i \leq \ell$ sei

$$E_i := \{ e \in E \mid \exists x \in V_{i-1}, y \in V_i : e \text{ ist möglich von } x \text{ nach } y \}.$$

Für $e = (v,w) \in E_i$ definieren wir die <u>reduzierte Kapazität</u> $\tilde{c}(e)$ von e:

$$\tilde{c}(e) := \begin{cases} c(e) - f(e) & \text{falls } v \in V_{i-1} \\ f(e) & \text{falls } v \in V_i. \end{cases}$$

Das geschichtete Netzwerk $G_f = (V, E', \tilde{c}, s, t)$ ist

definiert durch

$$E' := \{ (x,y) \mid \exists\, j \in \{1,2,\dots,e\} : x \in V_{j-1},\ y \in V_j,\ (x,y) \in E;\ \text{oder}\ (y,x) \in E; \}$$

Ziel:
Vergrößerung des Flusses f zu einem Fluss f' durch

1. Berechnung eines blockierenden Flusses $\bar{f}$ in G_f und
2. Definition des Flusses f' für $e = (x,y) \in E$ durch

$$f'(e) = \begin{cases} f(e) + \bar{f}(e) & \text{falls } x \in V_{j-1},\ y \in V_j,\ (x,y) \in E' \\ & \text{für ein } j \\ f(e) - \bar{f}(e) & \text{falls } x \in V_j,\ y \in V_{j-1},\ (y,x) \in E' \\ & \text{für ein } j \\ f(e) & \text{sonst} \end{cases}$$

Lemma 2.2
f' ist ein Fluss in $G = (V, E, c, s, t)$.

Beweis

Übung

■

Eine Phase des Algorithmus von Dinic besteht aus

1. der Berechnung des geschichteten Netzwerkes $G_f = (V, E', \tilde{c}, s, t)$,

2. der Berechnung eines blockierenden Flusses $\bar{f}$ in G_f und
3. der Vergrößerung des Flusses f in G zu einem Fluss f'.

Ziel:
Beweis, dass die Anzahl der Phasen durch $n := |V|$ beschränkt ist.

Bezeichne l_k den Index der letzten Schicht in der k-ten Phase.

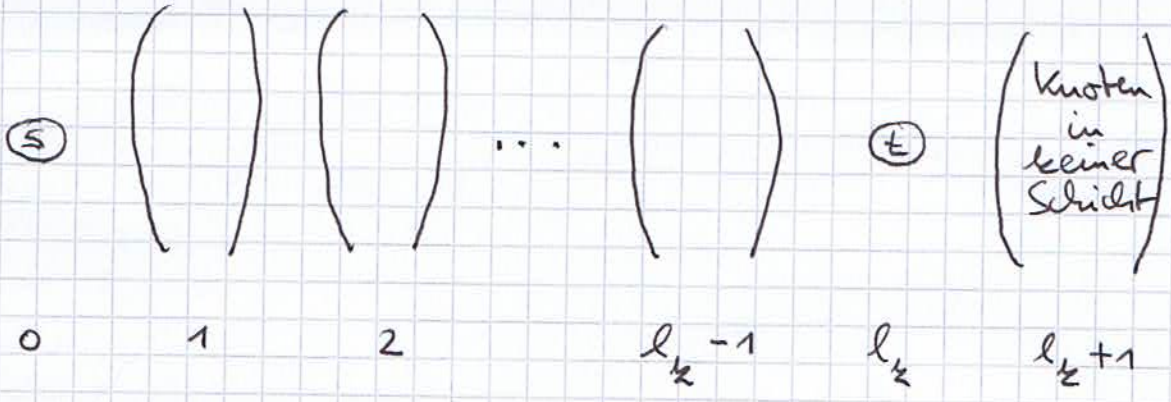
Lemma 2.3
Falls die k-te Phase nicht die letzte Phase ist, dann gilt $l_{k+1} > l_k$.

Beweis:

Annahme:
Nach der k-ten Phase existiert ein augmentierender Pfad.

Ziel:
Beweis, dass nach der k-ten Phase sich die Länge eines kürzesten augmentierenden Pfades strikt vergrößert hat.

Wir betrachten hierzu das geschichtete Netzwerk, das in der k-ten Phase konstruiert wird.



Wir teilen die in G bezüglich des Flusses f nützlichen Kanten in drei Klassen ein:

A: nützliche Kanten von Schicht i nach Schicht $i+1$ für ein $i < \ell_k$,

B: nützliche Kanten von Schicht i, $1 \leq i \leq \ell_k + 1$ nach Schicht j für ein $j \leq i$ und

C: nützliche Kanten von Schicht $\ell_k - 1$ nach Schicht $\ell_k + 1$.

Konstruktion $\Rightarrow$

1. $E' = A$, d.h., G_f enthält genau die Kanten in A. Dies impliziert, dass nach der k-ten Phase die Kanten in B und C auf dieselbe Art und Weise nützlich sind wie vorher. Eine Kante in A behält ihre Nützlichkeit oder wird nützlich von Schicht $i+1$ nach Schicht i, anstatt von Schicht i nach Schicht $i+1$.

2. Ein nach der k-ten Phase kürzester augmentierender Pfad muss jede Schicht passieren, d.h., $|P| \geq \ell_k$.

3. Da in der k-ten Phase ein blockierender Fluss konstruiert und dementsprechend augmentiert wurde, muss auf P eine Kante e liegen, für die gilt:

   i) $e \in B \cup C$ oder

   ii) $e \in A$ und nach der k-ten Phase ist e möglich von Schicht $i+1$ nach Schicht i und nicht mehr von Schicht i nach Schicht $i+1$, für ein $i < l_k$.

In beiden Fällen verlängert e den Pfad um mindestens eine Kante. Also gilt $|P| \geq l_k + 1$.

■

Korollar 2.1

Die Anzahl der benötigten Phasen ist $\leq n-1$.

Insgesamt haben wir folgenden Satz bewiesen:

Satz 2.1

Sei $G = (V, E, c, s, t)$ ein Flussnetzwerk. Dann kann ein maximaler Fluss f_{max} in Zeit $O(n \cdot m + n \cdot b)$ konstruiert werden, wobei $n = |V|$, $m = |E|$ und b diejenige Zeit ist, die für die Berechnung eines blockierenden Flusses in einem geschichteten Netzwerk mit $\leq n$ Knoten und $\leq m$ Kanten benötigt wird.

Durch Idee zur Konstruktion eines blockierenden Flusses war die folgende:

1. Berechne mittels Tiefensuche einen Pfad P von s nach t in G_f.
2. Augmentiere diesen. Falls für eine Kante e auf P danach $\bar{f}(e) = \bar{c}(e)$, dann entferne diese Kante aus G_f.
3. Wiederhole das Ganze solange, bis kein Pfad von s nach t in G_f mehr existiert.

Falls während der Tiefensuche ein Backtrack erfolgt, dann können wir betreffenden Knoten nebst allen inzidenten Kanten aus G_f entfernen.

Dinics Methode terminiert erst dann, wenn in G_f kein Pfad von s nach t mehr existiert.

$\Rightarrow$

Es wird ein blockierender Fluss berechnet.

Da das Augmentieren eines Pfades P in G_f mindestens eine Kante auf P blockiert, werden höchstens m Pfade augmentiert.

Da im Fall einer Pop-Operation betreffender Knoten nebst allen inzidenten Kanten aus G_f entfernt werden, ist die für die Pop-Operationen benötigte Gesamtzeit durch $O(n+m)$ begrenzt.

Eine Push-Operation, die nicht zu einem augmentierenden Pfad führt, zieht die korrespondierende Pop-Operation nach sich. Also ist die hierfür benötigte Gesamtzeit auch durch $O(n+m)$ begrenzt.

Ein augmentierender Pfad hat die Länge $< n$. Also ist die übrige benötigte Zeit durch $O(m \cdot n)$ begrenzt. Insgesamt ergibt sich folgender Satz:

Satz 2.2

Dinic' Algorithmus berechnet einen maximalen Fluss in Zeit $O(n^2 \cdot m)$.

Übung:
Arbeiten Sie den Algorithmus von Dinic aus.

2.2 Karzanovs Methode (1974)

Ziel:
Entwicklung von effizienteren Verfahren zur Berechnung eines blockierenden Flusses.

Karzanovs Idee:
Saturiere in jedem Schritt einen Knoten anstelt eine Kante.

Die Kapazität $c(v)$ eines Knotens ist die maximale Flussgröße, die durch den Knoten v durchgeschickt werden kann. D.h.,

$$c(v) := \min \left\{ \sum_{e \in IN(v)} c(e), \sum_{e \in OUT(v)} c(e) \right\}.$$

Ein Fluss f saturiert einen Knoten v, falls

$$\sum_{e \in IN(v)} f(e) = \sum_{e \in OUT(v)} f(e) = c(v).$$

Ein Knoten v heißt <u>blockiert</u>, falls auf jedem Pfad von v zur Senke t eine saturierte Kante existiert.

<u>Idee</u>:

- Bringe von s ausgehend möglichst viel Fluss in das Netzwerk. Dabei kann es passieren, dass mehr Fluss in einen Knoten gelangt, als abgeführt werden kann.
- Lass von solchen Knoten den überflüssigen Fluss zurückfließen und versuche diesen Fluss auf anderen Weg nach t zu bringen.
- Sorge dafür, dass einmal blockierte Knoten immer blockiert bleiben.

<u>Bezeichnungen</u>:

Sei $G = (V, E, c, s, t)$ ein Flussnetzwerk.
$p: E \to \mathbb{R}$ heißt <u>Preflow</u>, falls

1) $0 \leq p(e) \leq c(e) \quad \forall e \in E$ und

2) $\underbrace{\sum_{e \in IN(v)} p(e)}_{p_{in}(v)} \geq \underbrace{\sum_{e \in OUT(v)} p(e)}_{p_{out}(v)} \quad \forall v \in V \setminus \{s, t\}.$

Falls für ein $v \in V$ $p_{in}(v) > p_{out}(v)$, dann heißt v <u>unbalanciert</u>. Falls alle $v \in V \setminus \{s, t\}$ balanciert sind, dann ist der Preflow ein Fluss.

Sei $G_f = (V, E', \tilde{c}, s, t)$ das bezüglich des Flusses f konstruierte geschichtete Netzwerk.

Karzanovs Methode besteht aus zwei Schritten, die solange iteriert werden, bis alle Knoten balanciert sind. Anschließend enthält das Netzwerk einen blockierenden Fluss.

Schritt 1: Vorwärtsbringen von Preflow in G_f.

Schritt 2: Balancierung des Preflows in G_f.

Ein Knoten ist unbalanciert, da mehr Fluss ankommt als abgeführt werden kann. Somit sind alle ausgehende Kanten einer unbalancierten Knotens saturiert und dieser demzufolge blockiert.

Der Preflow p an einem unbalancierten Knoten v wird durch Verringerung von eingehenden Fluss derart balanciert, dass anschließend

$$p_{in}(v) = p_{out}(v) = \tilde{c}(v).$$

Also bleibt v blockiert.

Nach dem Balancieren verbieten wir, dass sich Preflow auf eingehenden Kanten von v ändert.

Ein Kante e heißt offen falls der Preflow auf e geändert werden darf. Andernfalls heißt e geschlossen. Zu Beginn sind alle Kanten offen.

Schritt 1:

Für alle Knoten $v \in V$ seien die Kanten in $Out(v)$ in einer beliebigen Ordnung durchnummeriert.

Idee:

Vom Knoten v wird Preflow in die nächste Schicht gebracht, indem

- möglichst viel Preflow über die erste offene nicht saturierte Kante
- dann möglichst viel Preflow über die zweite offene nicht saturierte Kante

⋮

u. s. w.

geschickt wird.

Dabei gilt bzgl. den offenen Kanten in $Out(v)$ stets folgende Invariante:

- Die ersten Kanten sind alle saturiert. Diese werden von einer nicht saturierten Kante e mit $p(e) \geq 0$ gefolgt. Wir bezeichnen diese Kante immer mit $k(v)$. Für alle nachfolgenden Kanten e gilt $p(e) = 0$.

Für die Organisation von Schritt 2 verwalten wir die eingehenden Kanten $e \in IN(w)$ eines Knotens w mit $p(e) > 0$ mit Hilfe eines Kellers K_w.

- Sobald $e \in IN(w)$ einen Preflow $p(e) > 0$ erhält, wird e auf K_w abgelegt.
- Falls im Verlauf von Schritt 2 der eingehende Fluss von w reduziert wird, wird zunächst möglichst viel Fluss auf $TOP(K_w)$ reduziert, dann möglichst viel Fluss auf dem zweitobersten Kellerelement u.s.w.

Schritt 1: Vorwärtsbringen von Preflow in G_f.

(1) Starte in s und bringe soviel Preflow wie möglich nach Schicht 1. D.h.,

```
for alle e ∈ OUT(s)
    do
        p(e) := c̃(e)
    od ;
START := 1 ;
```

(2) `for i := START step 1 until ℓ-1`

$V_\ell = \{t\}$

```
    do
        for alle v ∈ V_i
            do
                P := p_in(v)
                while P > 0 ∧ k(v) definiert
                    do
                        e := k(v) ;
                        P' := min { c̃(e) - p(e) , P } ;
                        p(e) := p(e) + P' ;
                        P := P - P'
                od
            od
    od
```

Nach Durchführung von Schritt 1 ist für jeden Knoten $v \in V \setminus \{s,t\}$ genau eine der beiden folgenden Fälle erfüllt:

1. $p_{in}(v) = p_{out}(v)$.
2. $p_{in}(v) > p_{out}(v) = \tilde{c}(v)$ (d.h., v ist unbalanciert)

Schritt 2: Balancieren des Preflows.

Sei V_{max} die Schicht mit höchstem Index, die einen unbalancierten Knoten enthält.

Idee:

- Für alle unbalancierte Knoten $v \in V_{max}$ tue: Unter Verwendung von K_v reduziere gemäß LIFO den eingehenden Fluss von v, bis v balanciert ist. Anschließend gilt somit
$$p_{in}(v) = p_{out}(v) = \tilde{c}(v).$$
Schließe alle eingehenden Kanten von v.
- Nachdem alle unbalancierte Knoten in V_{max} balanciert sind, bringe wieder Preflow in G_f vorwärts.

Beobachtung:

Da nur für Knoten in Schicht V_{max-1} der ausgehende Fluss verringert wurde, kann nun Schritt 1 mit START = max-1 durchgeführt werden.

Schritt 1 und Schritt 2 werden solange abwechselnd durchgeführt, bis kein unbalancierter Knoten in G_f existiert.

Übung:
Arbeiten Sie Karzanovs Algorithmus aus. Beweisen Sie, dass ein Fluss berechnet wird.

Beobachtung:
Nach dem ersten Vorwärtsbringen von Preflow gilt:

i) s ist blockiert (, da jede ausgehende Kante von s saturiert ist.)

ii) Jeder unbalancierte Knoten v ist blockiert (, da jede von v ausgehende Kante saturiert ist.)

Korrektheit:
Es genügt zu zeigen, dass ein blockierender Fluss konstruiert wird. Dies folgt unmittelbar aus folgenden Lemma.

Lemma 2.4
Ein blockierter Knoten bleibt während der gesamten Phase blockiert.

Beweis:

Beh.:
Jeder Knoten, der vor einem Balanciersschritt blockiert ist, bleibt dies auch nach dem Balanciersschritt.

Bew. der Beh.:

- Sei v ein blockierter Knoten in einer Schicht V_j mit $j \geq max$. Da $\forall$ Kanten e auf einem Pfad von v nach t in G_f der Preflow $p(e)$ durch den Balancierungsschritt nicht verändert wird, bleibt v blockiert.

- Sei v ein unbalanciertes und somit auch ein blockierter Knoten in Schicht V_{max}. D.h., der eingehende Preflow von v wird reduziert.

  Annahme:
  Der Preflow auf der Kante $e = (w, v)$ mit $w \in V_{max-1}$ wird reduziert.

  Sei $x \in V_j$, $j < max$ ein beliebiger Knoten, der vor der Reduktion von $p(e)$ blockiert war.

  Da sich der Preflow auf allen Pfaden von x nach t, die e nicht enthielten, nicht geändert hat und alle Pfade von x nach t, die e enthalten, durch den blockierten Knoten v gehen, bleibt x auch nach der Reduktion von $p(e)$ blockiert.

■

Da s zu Beginn blockiert ist, folgt aus Lemma 2.4 direkt, dass obiger Algorithmus einen blockierenden Fluss konstruiert.

Aufwandsanalyse:

Lemma 2.5

Obiger Algorithmus findet in $O(n^2)$ Zeit einen blockierenden Fluss.

Beweis:

- **Balancierungsschritt:**
  Wenn Preflow auf einer Kante e reduziert wird, dann wird e geschlossen.
  $\Rightarrow$
  Anzahl der Preflow-Reduktionen ist durch m beschränkt.

- **Vorwärtsbringen von Preflow:**
  Da jede Kante höchstens einmal saturiert wird, ist die Anzahl von Erhöhungen von Preflow auf einer Kante, wodurch diese saturiert wird, durch m beschränkt.

  Zwischen zwei Balancierungsschritten existiert für jeden nicht blockierten Knoten höchstens eine Kante, die eine nicht saturierende Erhöhung des Preflows erfährt.
  $\Rightarrow$
  Für die Gesamtanzahl A der nicht saturierenden Erhöhungen gilt

$$A \leq \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = O(n^2)$$

■

Bemerkung:
Durch Verwendung von relativ aufwendigen Datenstrukturen konnte die benötigte Zeit bzgl. nicht saturierenden Erhöhungen von $O(n^2)$ auf $O(n \cdot \log \frac{n^2}{m})$ gesenkt werden. Siehe hierzu

A.V. Goldberg, R.E. Tarjan, A new approach to the maximum flow problem, JACM (1988), 921-940.

Eine geschickte Modifikation von Karzanovs Methode wurde 1978 von Malhotra, Pramodh Kumar und Maheshwari durchgeführt.

Sei f ein Fluss in Flussnetzwerk $G = (V, E, c, s, t)$, $G_f = (V, E', \tilde{c}, s, t)$ das korrespondierende geschichtete Netzwerk und $v \in V$. Dann ist das Flusspotential $\varphi_f(v)$ von v definiert durch

$$\varphi_f(v) := \tilde{c}(v) - \sum_{(u,v) \in E'} f(u,v).$$

$r \in V$ heißt Referenzknoten, falls

$$\varphi_f(r) = \min \{ \varphi_f(v) \mid \varphi_f(v) > 0, v \in V \}.$$

Lemma 2.6
Sei r ein Referenzknoten im geschichteten Flussnetzwerk $G_f = (V, E', \tilde{c}, s, t)$. Ferner sei $\varphi_f(v) > 0$ $\forall v \in V \setminus \{s,t\}$ mit $\exists$ Pfad von s nach v und ein Pfad von v nach t in G_f. Dann kann f derart um $\varphi_f(r)$ zu einem Fluss f' erhöht werden, dass

ausschließend $\rho_f'(r) = 0$.

Beweis:

$\forall v \in V' \setminus \{s,t\}$ gilt $\rho_f(r) \leq \rho_f(v)$. Also können wir von r ausgehend zusätzlich den Fluss $\rho_f(r)$ Schicht für Schicht nach t schicken.

Ebenso kann von r ausgehend rückwärts ein zusätzlicher Fluss $\rho_f(r)$ von s nach r konstruiert werden. ∎

Algorithmus MPM

Eingabe: geschichtetes Netzwerk $G_f = (V', E', \tilde{c}, s, t)$

Ausgabe: blockierender Fluss $\bar{f}$.

Methode:

(1) Bestimme einen Referenzknoten r.
(2) Bringe von r den Fluss $\rho_f(r)$ nach t.
(3) Bringe von r ausgehend rückwärts den Fluss $\rho_f(r)$ von s nach r.
(4) Entferne alle Kanten und Knoten, über die kein Fluss mehr geschickt werden kann, aus dem Netzwerk.
(5) Falls das Netzwerk nicht leer ist, dann goto (1)

Zeitanalyse:

	ein Durchlauf (1)–(5)	insgesamt
(1)	$\leq n-1$ Vergleiche	
(2) + (3)	(n nicht sättigend)	(m sättigend)
(4) + (5)		$O(n+m)$
Σ		$O(n^2)$

Übung:
Arbeiten Sie den Algorithmus MPM aus. Beweisen Sie seine Korrektheit und vervollständigen Sie die Zeitanalyse.

2.3 Goldbergs Methode (1985)

Falls nicht gezeigt werden kann, dass Dinic' Methode mit viel weniger als $n-1$ Phasen auskommt, dann können wir mit dieser Methode bestenfalls einen $O(n \cdot m)$ - Algorithmus für das maximale Flussproblem erhalten.

⇝

Zum Finden von effizienteren Algorithmen ist es sinnvoll, von den geschichteten Netzwerken wegzukommen.

Idee (Goldberg):

- Anstatt das geschichtete Netzwerk explizit zu konstruieren, verwalte für alle Knoten Distanzmarkierungen, die eine Schätzung der Distanz des jeweiligen Knotens zur Senke beinhalten.
- Bearbeite unter Berücksichtigung der Distanzmarkierungen einen Preflow.

⇝

Hierzu benötigen wir zwei Operationen:

1. Vorwärtsbringen von überschießenden Preflow (pushing)
2. Aufbereitung von Distanzmarkierungen (relabeling)

Bezeichnungen:

Seien $G = (V, E, c, s, t)$ ein Flussnetzwerk und $v, w \in V$. Die Distanz $d_G(v,w)$ von v nach w ist definiert durch

$$d_G(v,w) := \begin{cases} \min\{|P| \mid P \text{ ist Pfad von } v \text{ nach } w \text{ in } G\} & \text{falls solch ein Pfad } P \text{ existiert} \\ \infty & \text{sonst} \end{cases}$$

Wir modifizieren folgendermaßen die Definition eines Flusses:

$f: V \times V \to \mathbb{R}$ ist genau dann ein Fluss in G, falls gilt:

1) $f(v,w) \leq c(v,w)$ $\quad \forall (v,w) \in V \times V$
(Kapazitätsbedingung)

2) $f(v,w) = -f(w,v)$ $\quad \forall (v,w) \in V \times V$
(Antisymmetriebedingung)

3) $\sum_{u \in V} f(u,v) = 0$ $\quad \forall v \in V \setminus \{s,t\}$
(Flusserhaltungsbedingung)

Bemerkung:
Die Antisymmetriebedingung impliziert, dass das Kirchhoff'sche Gesetz durch die Flusserhaltungsbedingung ersetzt werden kann.

Übung:
Beweisen Sie die Äquivalenz der neuen und der alten Definition eines Flusses.

Als nächstes passen wir unsere Definition eines Preflows an die neue Definition eines Flusses an.

$p: V \times V \to \mathbb{R}$ heißt Preflow in G, falls gilt:

$\forall (v,w) \in V \times V$ gelten die Kapazitäts-, die Antisymmetrie- und folgende abgeschwächte Flusserhaltungsbedingung:

3') $\sum_{u \in V} p(u,v) \geq 0$ $\quad \forall v \in \in V \setminus \{s,t\}$.

Für einen Preflow p bezeichnet $\Delta_p(v)$, $v \in V \setminus \{s,t\}$ den <u>überschießenden Preflow</u> in v. D.h.,

$$\Delta_p(v) := \sum_{u \in V} p(u,v) \quad \left(= p_{in}(v) - p_{out}(v) \right).$$

Für $(v,w) \in V \times V$ bezeichnet

$$r_p(v,w) := c(v,w) - p(v,w)$$

die <u>Restkapazität</u> von (v,w).

<u>Beobachtung</u>:

a) Für $v \in V \setminus \{s,t\}$ gilt:

($\Delta_p(v) > 0 \wedge \exists w \in V : r_p(v,w) > 0$

$\Rightarrow$ $\Delta_p(v)$ kann um bis zu

$$\delta := \min \{ r_p(v,w), \Delta_p(v) \}$$

durch Erhöhung von $p(v,w)$ verringert werden.)

b) Für $(v,w) \in V \times V$ impliziert $r_p(v,w) > 0$

i) $(v,w) \in E \wedge p(v,w) < c(v,w)$

oder

ii) $(w,v) \in E \wedge p(w,v) > 0$.

<u>Bezeichnungen</u>:

$(v,w) \in V \times V$ heißt <u>nützlich</u>, falls $r_p(v,w) > 0$.

Der Restgraph $G_p = (V, E_p, c, s, t)$ bzgl. Preflow p ist definiert durch

$$E_p := \{ (v,w) \in V \times V \mid r_p(v,w) > 0 \}.$$

Eine Funktion $d: V \to \mathbb{N}$ heißt gültige Markierung, falls gilt:

i) $d(s) = n \quad , d(t) = 0$

ii) $d(v) \leq d(w) + 1 \quad \forall (v,w) \in E_p$.

Gültige Markierungen halten stets folgende Invariante aufrecht:

Invariante (*):

- $d(v) < n \Rightarrow d_{G_p}(v,t) \geq d(v)$
- $d(v) \geq n \Rightarrow d_{G_p}(v,s) \geq d(v) - n$.

Übung:
Beweisen Sie die Invariante (*).

Ein Knoten $v \in V \setminus \{s,t\}$ heißt aktiv, falls $d(v) < \infty$ und $\Delta p(v) > 0$.

Idee:
In jedem Schritt wird ein aktiver Knoten betrachtet und an ihm "gearbeitet".

Nun können wir den allgemeinen Algorithmus zur Berechnung eines maximalen Flusses in einem Flussnetzwerk formulieren.