(M. Dietzfelbinger, aktualisiert 3. Juni 2020)

1 Maximierung von Flüssen in Netzwerken

Situation 1: Wir betrachten ein System von Leitungen für Flüssigkeiten (z. B. Wasser). Jede Leitung ist eine Einbahnstraße; verschiedene Leitungen treffen sich in Knoten. In einen der Knoten kann man Wasser einleiten („Quelle“, engl.: *source*), an einem anderen kann man es entnehmen („Senke“, engl.: *sink* (heißt auch Ausguss)). Die anderen Knoten sind dicht, d. h. es kann nichts hinzugefügt und nichts entnommen werden. Jede Leitung hat ein maximales Durchleitungsvermögen, Volumen pro Zeiteinheit, gemessen z. B. in m^3/s. Wir wollen einen kontinuierlichen Strom organisieren, in dem pro Zeiteinheit möglichst viel Wasser eingespeist und entnommen wird. Wieviel Wasser (pro Zeiteinheit) sollte durch jede der Leitungen fließen? (Illustration: Abb. 1, wobei man „Bahnhof“ als „Quelle“ und „Stadion“ als „Senke“ liest.)

Situation 2: („Fanströme“) In Dortkirchen gibt es ein Straßensystem und 60 000 Fußballfans, die am Bahnhof ankommen und zu Fuß zum Stadion wollen. Für jede Straße ist eine maximale Gesamtanzahl an Personen festgelegt, die sie passieren können. Wie soll die Polizei die Menschenmassen aufteilen und lenken, so dass alle ankommen? (Illustration: Abb. 1, Kapazitäten in Zehntausend.)

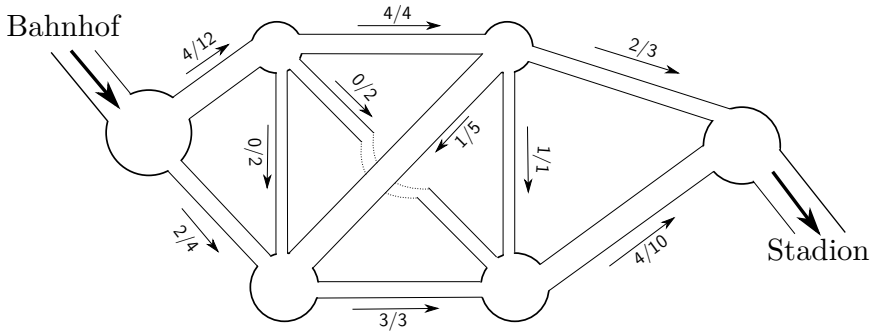


Abbildung 1: Beispiel Fanströme. Aufteilung der 60 000 Fussballfans gemäß der Personenbeschränkung der Straßen.

In diesem Abschnitt stellen wir diese Problemstellung abstrakt dar und betrachten effiziente Algorithmen, die sie lösen.

1.1 Netzwerke und Flüsse

Definition 1.1.1. *Ein* ***Flussnetzwerk*** **(*FNW*)** *ist ein gerichteter Graph (Digraph)* $G = (V, E, q, s, c)$ *mit Zusatzinformationen, wobei* (V, E) *Digraph ist,* $q, s \in V$ (***Quelle*** *und* ***Senke***) *ausgezeichnete Knoten sind und* $c\colon E \to \mathbb{R}_0^+$ *eine Funktion ist (die für jede Kante* e *eine* ***Kapazität*** $c(e) \geq 0$ *festlegt).*

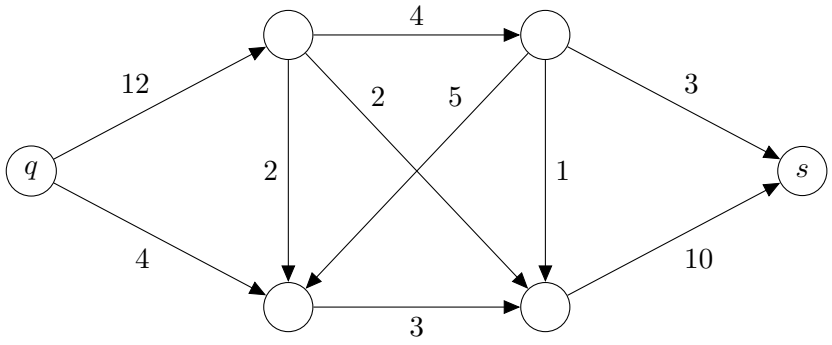


Abbildung 2: Das Flussnetzwerk zum Fanströmeproblem in Abbildung 1.

O. B. d. A. dürfen wir annehmen: Jeder Knoten $v \in V$ ist von q aus erreichbar, und von jedem Knoten $v \in V$ aus ist s erreichbar. (Eventuell vorhandene andere Knoten spielen in unseren Algorithmen keine Rolle. Daher kann man sie gleich weglassen. Die erreichbaren Knoten bestimmt man durch Tiefen- oder Breitensuche von q aus bzw. durch Suche von s aus im Umkehrgraphen G^{R}.) Oft nimmt man zudem an, dass q keine eingehenden Kanten und s keine ausgehenden Kanten hat. (Keiner der Algorithmen, die wir betrachten, benutzt solche Kanten. Daher kann man sie gleich weglassen.)

O. B. d. A. nehmen wir weiter an: (V, E) enthält keine Paare entgegengesetzter Kanten: $(u, v) \in E \Rightarrow (v, u) \notin E$.
(Wenn in einem FNW Kanten (u, v), (v, u) mit $c(u, v), c(v, u) > 0$ vorhanden sind, ersetzt man (v, u) durch zwei Kanten $(v, w), (w, u)$ mit $c(v, w) = c(w, u) = c(v, u)$, wobei w ein neuer Knoten ist. Wenn die Berechnung des maximalen Flusses (s. u.) beendet ist, kann man den neuen Knoten wieder eliminieren.)

Notation: $n = |V|$, $m = |E|$ stets. Mit den gemachten Annahmen gilt dann: $n - 1 \leq m \leq n(n-1)/2$.

Definition 1.1.2. *Ein **(zulässiger) Fluss** in einem Flussnetzwerk $G = (V, E, q, s, c)$ ist eine Funktion $f\colon E \to \mathbb{R}_0^+$, die jeder Kante e einen **Flusswert** $f(e)$ zuordnet, wobei folgende Bedingungen eingehalten werden:*

(i) **(Flusserhaltung in jedem Knoten, Kirchhoffsches Gesetz)** *Für jedes $v \in V$ betrachten wir, wieviel Fluss in v hineingeht und wieviel aus v heraus:*

$$f_{\mathrm{In}}(v) := \sum_{(u,v)\in E} f(u,v), \qquad f_{\mathrm{Out}}(v) := \sum_{(v,u)\in E} f(v,u).$$

Wir verlangen: $\forall v \in V - \{q, s\}\colon f_{\mathrm{In}}(v) = f_{\mathrm{Out}}(v)$.

(ii) **(Einhalten der Kapazitäten)** $\forall e \in E\colon 0 \le f(e) \le c(e)$.

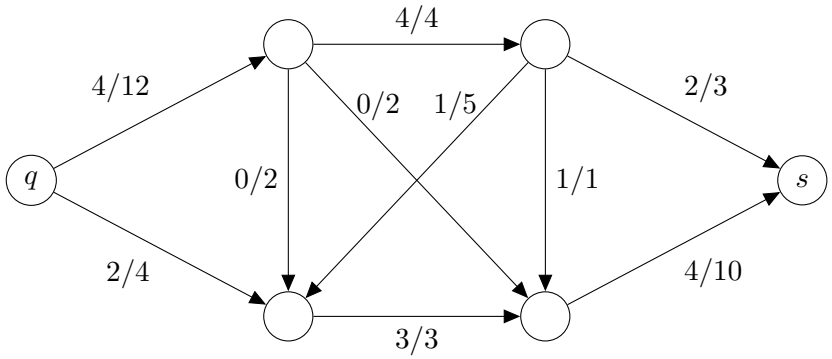
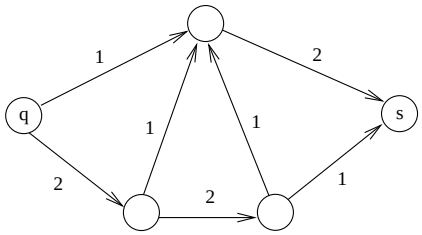


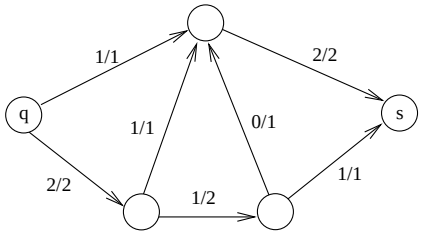
Abbildung 3: Ein zulässiger Fluss im Flussnetzwerk von Abbildung 2 mit Wert 6. Wir nutzen hierbei die Notation $f(e)/c(e)$ für die Darstellung des Flusses $f(e)$ über eine Kante e mit Kapazität $c(e)$.

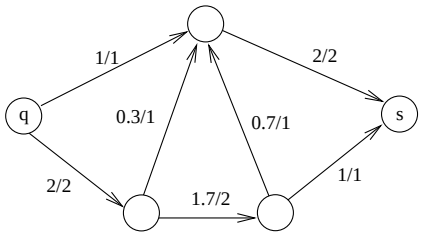
Definition 1.1.3. *Der **Wert** eines Flusses f ist $w(f) = f_{\mathrm{Out}}(q)$.*
(Sollte q eingehende Kanten haben, haben diese stets Flusswert 0.)

Problem: Zu gegebenem FNW G finde einen Fluss f mit möglichst großem $w(f)$.

Es wird sich herausstellen, dass es stets einen Fluss gibt, der den eindeutig bestimmten größtmöglichen Wert realisiert. Dies erscheint einfach, wenn die Kapazitäten ganzzahlig sind. Dann gibt es nur endlich viele legale Flüsse mit ganzzahligen $f(u, v)$, darunter also auch einen mit maximalem Wert. Das folgende Bild zeigt aber, dass es auch maximale Flüsse geben kann, die nicht-ganzzahlige Flusswerte benutzen.







Bei anderen Problemstellungen (Lineare Programmierung, Rucksack usw.) kann es durchaus sein, dass es eine solche „fraktionale“ Lösung gibt, die besser als alle ganzzahligen Lösungen ist. Man muss also noch überlegen, weshalb kein Fluss mit nichtganzzahligen Werten $f(u, v)$ besser sein kann als der beste ganzzahlige Fluss. Wenn die Kapazitäten rational sind, können wir sie mit dem Hauptnenner durchmultiplizieren, ohne das Problem wesentlich zu verändern, und erreichen so die Situation ganzzahliger Kapazitäten. Ganz unklar ist die Situation vorerst bei beliebigen reellen Kapazitäten. Die Existenz eines optimalen Flusses in diesem Fall beweisen wir später (in Abschnitt 1.3).

Bemerkung. (Für Studierende mit Kenntnissen in höherdimensionaler Analysis.) Die Menge der Tupel $f = (f(e))_{e \in E} \in \mathbb{R}^{|E|}$, die (ii) einhalten, ist abgeschlossen und beschränkt, also kompakt. Die Bedingungen (i), also die Flusserhaltungs-Gleichungen, definieren abgeschlossene Mengen, also ist die Menge aller Flüsse ebenfalls kompakt. Die Bewertungsfunktion $f \mapsto w(f)$ ist stetig. Also nimmt sie irgendwo im zulässigen Bereich ihr Maximum an.

1.2 Die Ford-Fulkerson-Methode und das MaxFlow-MinCut-Theorem

(L. R. Ford Jr. und D. R. Fulkerson, 1956.)

Grundidee: Iterative Verbesserung des Flusses, bis man den maximalen Fluss erreicht.

Grundvoraussetzung für das Gelingen: Die Kapazitäten sind ganzzahlig. (Oder rational: mit Hauptnennertrick auf ganzzahlige Kapazitäten umrechnen!)

Gegeben: Fluss f zu einem Flussnetzwerk G. Kann man diesen Fluss verbessern? Kann man noch mehr Fluss von q nach s schicken?

Definition 1.2.1. *Gegeben sei ein FNW G mit Fluss f.*

(a) *Wir definieren **Restkapazitäten** für Paare $(u, v) \in V \times V$, $u \neq v$.*

$$\mathrm{rest}_f(u,v) := \begin{cases} c(u,v) - f(u,v) & \text{, falls } (u,v) \in E \\ f(v,u) & \text{, falls } (v,u) \in E \\ 0 & \text{, sonst.} \end{cases}$$

(b) *Das **Restnetzwerk (RNW)** G_f hat Knotenmenge V, Kantenmenge $E_f = \{(u,v) \mid \mathrm{rest}_f(u,v) > 0\}$ und Kapazitäten $\mathrm{rest}_f(u,v)$ auf Kante $(u,v) \in E_f$.*

Wenn $(u,v) \in E_f$, gilt $(u,v) \in E$ oder $(v,u) \in E$, also ist $|E_f| \leq 2|E|$.

Man beachte, dass G_f im strengen Sinn kein Flussnetzwerk ist, da es entgegengesetzte Kanten haben kann und der Eingangsgrad von q und der Ausgangsgrad von s größer als 0 sein kann. Dennoch kann man definieren, was ein Fluss in G_f ist: Eine Funktion $f' \colon E_f \to \mathbb{R}_0^+$, die die Kapazitätsschranken $0 \leq f'(u,v) \leq \mathrm{rest}_f(u,v)$ und die Kirchhoffregel in $v \in V - \{q,s\}$ einhält *und* für die $f'(u,v) = 0$ oder $f'(v,u) = 0$ gilt, falls beide Kanten in E_f sind.

Kanten $(u,v) \in E \cap E_f$ heißen **Vorwärtskanten**. Die Interpretation der Restkapazität ist hier einfach: Man könnte noch **zusätzlich** Fluss im Wert von bis zu $c(u,v) - f(u,v)$ über diese Kante schicken.

Kanten $(u,v) \in E_f$ mit $(v,u) \in E$ heißen **Rückwärtskanten**. Hier kann man gewissermaßen den Fluss von u nach v erhöhen, indem man den Fluss von v nach u *verringert*, um einen Wert von bis zu $f(v,u)$.

Definition 1.2.2. *Gegeben sei ein FNW G mit Fluss f sowie das zugehörige Restnetzwerk G_f. Ein **flussvergrößernder Weg** (kurz: **fv Weg** oder **fvW**, engl.: **augmenting path**) ist ein einfacher Weg p von q nach s in G_f.*

*Die **Kapazität** eines solchen fvW p ist $c(p) := \min\{\mathrm{rest}_f(e) \mid e \text{ liegt auf } p\}$.*

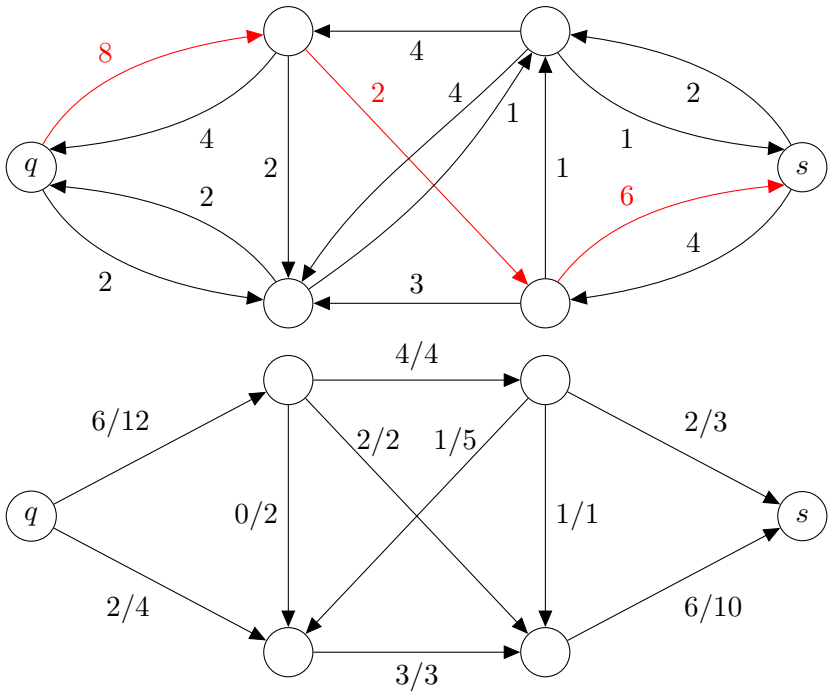


Abbildung 4: Oben: Das Restnetzwerk zum Fluss in Abbildung 3. Wir vergrößern den Fluss aus Abb. 3 entlang des rot markierten Weges p um $c(p) = 2$. Unten: Das Resultat der Flussvergrößerung. Der Fluss hat nun einen Wert von 8.

Bemerkung: Weil in G_f nur Kanten mit positiver Restkapazität vorkommen, gilt $c(p) > 0$.

Wenn p ein flussvergrößernder Weg ist, kann man auf G_f einen Fluss f_p mit Wert $w(f_p) = c(p)$ definieren: Für $(u, v) \in E_f$ setze:

$$f_p(u,v) := \begin{cases} c(p) & \text{, falls } (u,v) \text{ auf } p \text{ liegt} \\ 0 & \text{, sonst.} \end{cases}$$

Weil p ein einfacher Weg ist, können nie sowohl (u, v) als auch (v, u) auf p liegen.

Wir behaupten, dass die Kirchhoff-Bedingung in $v \in V - \{q, s\}$ erfüllt ist und dass $w(f_p) = c(p)$ gilt. Die erste Behauptung verifiziert man dadurch, dass man kontrolliert, dass in jeden Knoten im Inneren von p genau eine Kante von p hineinführt und genau eine aus ihm herausführt, beide haben Flusswert $c(p)$. Aus der Quelle führt eine Kante mit Flusswert $c(p)$ heraus, also ist $w(f_p) = c(p)$.

Immer wenn wir einen Fluss f in G und einen Fluss f' in G_f mit $w(f') > 0$ haben, der

keine Paare entgegengesetzter Kanten benutzt, können wir den Fluss f vergrößern: Für $(u,v) \in E$ setze

$$(f+f')(u,v) := \begin{cases} f(u,v) + f'(u,v) & \text{, falls } f'(u,v) > 0 \text{ ,} \\ f(u,v) - f'(v,u) & \text{, falls } f'(v,u) > 0 \text{ ,} \\ f(u,v) & \text{, sonst .} \end{cases} \tag{1}$$

Ist diese Definition sinnvoll („wohldefiniert“) und liefert sie einen Fluss für G? Wenn $f'(u,v) > 0$ gilt, muss $0 < f'(u,v) \leq \text{rest}_f(u,v) = c(u,v) - f(u,v)$ sein, also ist $0 < f(u,v) + f'(u,v) \leq c(u,v)$. Wenn $f'(v,u) > 0$ gilt, muss $0 < f'(v,u) \leq \text{rest}_f(v,u) = f(u,v)$ sein, also ist $0 \leq f(u,v) - f'(v,u) < f(u,v) \leq c(u,v)$. Die beiden Fälle können nicht gleichzeitig auftreten, weil f' keine Paare entgegengesetzter Kanten benutzt.

Die neuen Werte halten also die Kapazitätsgrenzen ein. Die Kirchhoffregeln sind erfüllt, weil sowohl f als auch f' sie erfüllen. Die Gleichheit $w(f+f') = w(f) + w(f')$ ist offensichtlich.

Die **Ford-Fulkerson-Methode** besteht nun einfach in Folgendem:

(1) Definiere $f := f_0$ für den „Null-Fluss“ f_0, mit $f_0(u,v) = 0$ für alle $(u,v) \in E$.

(2) **repeat**

  (i) Berechne RNW G_f.

  (ii) Wenn es in G_f keinen fvW mehr gibt, **return** f; sonst:

  (iii) Berechne fvW p und Fluss f_p, setze $f := f + f_p$.

Über diese Methode können wir folgendes feststellen:

Lemma 1.2.3. *Sei G FNW mit ganzzahligen Kapazitäten. Dann gilt:*

(a) *Alle Werte $f(u,v)$ und alle $c(p)$, die jemals auftreten, sind ganzzahlig.*

(b) *Sei $C_G := \sum_{(q,v)\in E} c(q,v)$ (die Kapazität aller aus q hinaus führenden Kanten). Dann führt die FF-Methode auf G maximal C_G Schleifendurchläufe aus und gibt einen Fluss f aus, für den es in G_f keinen fvW gibt. Es gilt $w(f) \leq C_G$.*

Beweis: (a) wird per Induktion über die Schleifendurchläufe bewiesen. Die wesentliche Feststellung ist, dass auf Zahlen nur die Operationen Addition, Subtraktion und Minimumsbildung angewendet wird, die die Ganzzahligkeit erhalten. (b) Wir beobachten den Wert $w(f)$ des Flusses im Verlauf des Algorithmus. Dieser ist anfangs 0 und erhöht sich in jedem Schleifendurchlauf um einen Wert $c(p) > 0$, der

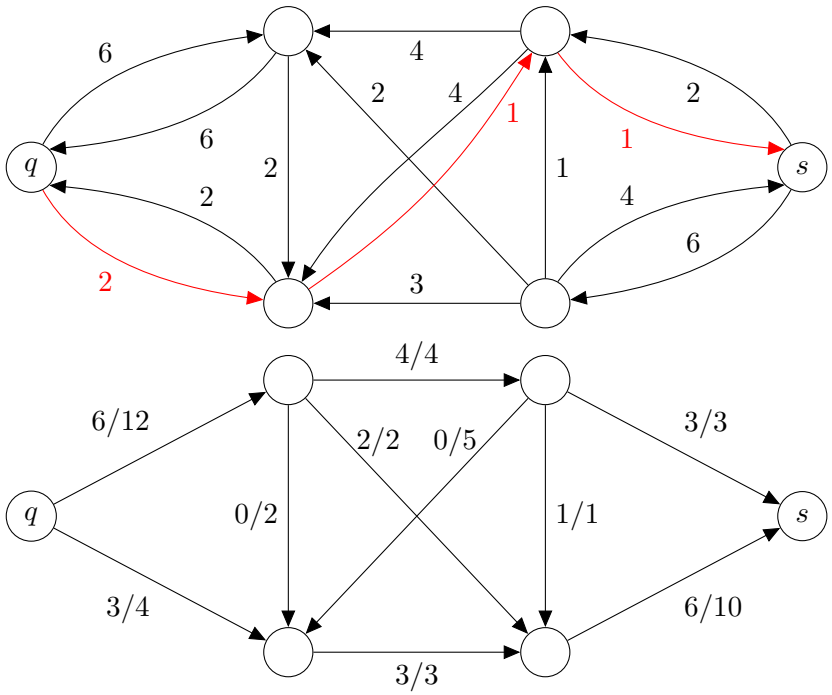


Abbildung 5: Das Restnetzwerk zum Fluss in Abbildung 4. Wir vergrößern den Fluss aus Abb. 4 entlang des rot markierten Pfades p um $c(p) = 1$. Das Resultat ist im unteren Bild zu sehen. Der Fluss hat nun einen Wert von 9.

das Minimum einer Menge von (positiven) Differenzen von ganzen Zahlen ist. Also gilt $c(p) \geq 1$. Größer als C_G kann $w(f)$ nicht werden, also endet die Schleife spätestens nach C_G Durchläufen. (Genauer: Es gibt höchstens $w(f)$ Durchläufe für den ganzzahligen Fluss f, mit dem die Schleife stoppt.) □

Die Schleife endet in einer Situation, wo kein Weg von q nach s in G_f gefunden wird. Das heißt, dass kein solcher Weg existiert. Aber was bedeutet es, wenn es in G_f keinen Weg von q nach s gibt? Wir vermuten, dass dann f maximal ist. Dies ist der wesentliche Inhalt des folgenden Satzes.

Achtung: Die folgenden Überlegungen benutzen die Ganzzahligkeit der Kapazitäten und Flüsse nicht. Die Ergebnisse gelten also allgemein.

Technisch benötigen wir noch einen zentralen Begriff. Ein ***Schnitt*** (engl. *cut*) zerlegt ein FNW in zwei Teilmengen, wobei q und s in verschiedenen Teilen liegen.

Definition 1.2.4. *Ein* ***Schnitt*** (Q, S) *in einem Flussnetzwerk* $G = (V, E, q, s, c)$ *ist eine Aufteilung der Knotenmenge* V *in zwei disjunkte Mengen* Q *und* S *(d. h.* Q *und*

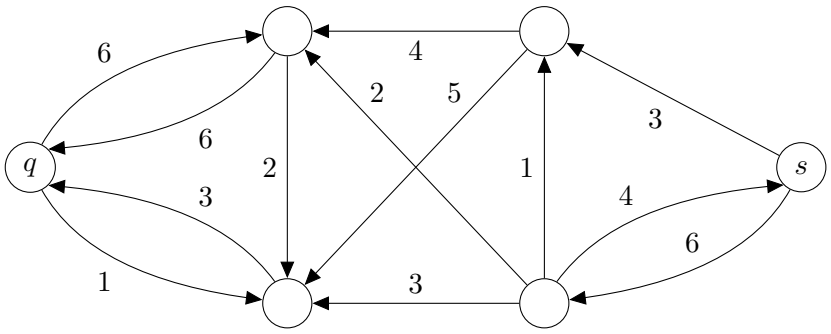


Abbildung 6: Das Restnetzwerk zum Fluss in Abbildung 5. Es existiert kein Weg von der Quelle zur Senke.

*S sind disjunkt und $Q \cup S = V$) mit $q \in Q$ und $s \in S$. Die **Kapazität** eines solchen Schnittes (Q, S) ist definiert als*

$$c(Q, S) := \sum_{\substack{(u,v)\in E,\\ u\in Q, v\in S}} c(u, v)$$

Zwischen Flusswerten und Schnittkapazitäten gibt es eine wesentliche Beziehung: Kein Fluss kann größer sein als die Kapazität eines Schnittes. Bei einem besonderen Schnitt ist dies klar: $Q = \{q\}$ und $S = V - \{q\}$. Die Kapazität ist $\sum_{(q,v)\in E} c(q, v)$, und $w(f) = \sum_{(q,v)\in E} f(q, v)$ kann wegen der Kapazitätsregel nicht größer als dieser Wert sein. Allgemein stellt man folgendes fest.

Lemma 1.2.5. *Sei G ein FNW, f ein beliebiger Fluss, (Q, S) ein beliebiger Schnitt. Wenn wir den **Fluss über den Schnitt** (Q, S) als*

$$f(Q, S) = \sum_{\substack{(u,v)\in E,\\ u\in Q, v\in S}} f(u, v) \quad - \sum_{\substack{(v,u)\in E,\\ v\in S, u\in Q}} f(v, u)$$

definieren[1]*, dann gilt*

$$w(f) = f(Q, S) \leq c(Q, S).$$

Beweis: Übung. (Die Gleichheit kann man gut durch Induktion über die Größe von Q beweisen. Die Ungleichung ist sehr leicht einzusehen.) □

Nun folgt der Hauptsatz über Flussalgorithmen.

[1]Positiv: was fließt von Q nach S? Negativ: was fließt von S nach Q?

Satz 1.2.6 (MaxFlow-MinCut-Theorem)**.** *Sei G ein FNW und f ein Fluss für G. Dann sind äquivalent:*

(i) *f ist maximal.*

(ii) *Das RNW G_f enthält keinen fvW.*

(iii) *Es gibt einen Schnitt (Q, S) mit $c(Q, S) = w(f)$.*

Beweis: Wir verwenden einen Ringschluss „(i) $\Rightarrow$ (ii) $\Rightarrow$ (iii) $\Rightarrow$ (i)“.

„(i) $\Rightarrow$ (ii)“: Wir beweisen die Kontrapositon. (Wenn (ii) falsch ist, ist auch (i) falsch.) Nehmen wir also an, dass das RNW G_f einen fvW p mit Kapazität $c(p) > 0$ hat. Dann ist, wie oben gesehen, $f + f_p$ ein Fluss mit Wert $w(f + f_p) > w(f)$, also ist f nicht maximal.

„(ii) $\Rightarrow$ (iii)“: (Dies ist das entscheidende Argument!) Nehmen wir an, dass es im RNW G_f keinen Weg von q nach s gibt. Wir **definieren** einen Schnitt:

$$\begin{aligned} Q &:= \{v \in V \mid v \text{ ist von } q \text{ aus in } G_f \text{ erreichbar}\}; \\ S &:= V - Q. \end{aligned}$$

Trivialerweise ist $q \in Q$; wegen der Annahme gilt $s \notin Q$, also $s \in S$.

Betrachte nun eine Kante $(u, v) \in E$ mit $u \in Q$ und $v \in S$. Nach der Definition von Q ist u in G_f von q aus erreichbar, aber v nicht. Daher ist $(u, v) \notin E_f$. Nach der Definition von E_f bedeutet dies, dass $f(u, v) = c(u, v)$ ist (die Kante (u, v) ist „gesättigt“). – Nun betrachte eine Kante $(v, u) \in E$ mit $v \in S$ und $u \in Q$. Wieder ist $(u, v) \notin E_f$, also muss $f(v, u) = 0$ sein, wieder nach der Definition von E_f.

Also gilt:

$$f(Q, S) = \sum_{\substack{(u,v)\in E,\\ u\in Q, v\in S}} f(u, v) \quad - \sum_{\substack{(v,u)\in E,\\ v\in S, u\in Q}} f(v, u) = \sum_{\substack{(u,v)\in E,\\ u\in Q, v\in S}} c(u, v) = c(Q, S).$$

Damit ist (iii) gezeigt.

„(iii) $\Rightarrow$ (i)“: Sei (Q, S) ein Schnitt mit $c(Q, S) = w(f)$, und sei f' ein beliebiger anderer Fluss. Dann gilt nach Lemma 1.2.5:

$$w(f') = f'(Q, S) \leq c(Q, S) = w(f).$$

Das heißt, dass kein Fluss einen größeren Wert als $w(f)$ haben kann, also dass f maximal ist. □

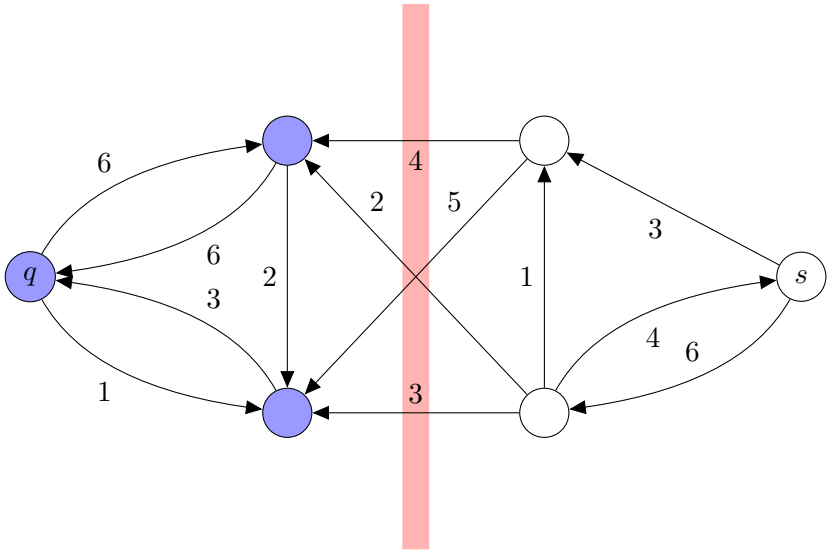


Abbildung 7: Das Restnetzwerk zum Fluss in Abbildung 5. Es existiert kein Weg von der Quelle zur Senke. Wie im Beweis von Satz 1.2.6 bestimmen wir den (Q, S)-Schnitt. Blau gefärbte Knoten sind in der Menge Q, alle anderen in der Menge S. Die Kapazität des Schnittes beträgt 9 (vgl. Abbildung 5).

Aus dem MaxFlow-MinCut-Theorem folgt sofort, dass in dem Moment, in dem die Ford-Fulkerson-Methode anhält, der Fluss f maximal ist, da Bedingung (ii) erfüllt ist. Diese Methode findet also einen maximalen Fluss, wenn die Kapazitäten ganzzahlig (oder rational) sind. Die Anzahl der Runden ist maximal C_G (oder genauer: $w(f)$ für den maximalen Fluss f). Jede Konstruktion eines Restnetzwerks benötigt Zeit $O(m)$, ebenso die Suche nach einem fvW (zum Beispiel mit Tiefensuche) und gegebenenfalls die Flussvergrößerung. Die gesamte Rechenzeit ist also $O(m \cdot C_G)$. Da die Größe der Kapazität (und nicht die Bitlänge der Zahldarstellung) in die Laufzeit eingeht, sprechen wir von einem *pseudopolynomiellen* Algorithmus.[2]

Beobachtung 1.2.7. *Wenn alle Kapazitäten ganzzahlig sind, dann gibt es einen maximalen Fluss f, in dem alle Flusswerte $f(u, v)$ ganzzahlig sind.* (*Grund: Die FF-Methode konstruiert einen solchen Fluss, vgl. Lemma 1.2.3.*)

Bemerkung zum Namen „MaxFlow-MinCut-Theorem“: Bisher haben wir nur nach einem „maximalen Fluss“ (also einem Fluss mit maximalem $w(f)$) gefragt. Die FF-

[2]Man kann eine Realisierung der FF-Methode finden, bei der die Rundenzahl durch $O(m \log C_G)$, also die Laufzeit durch $O(m^2 \log C_G)$ beschränkt ist. Wir werden allerdings sehr schnell noch bessere Algorithmen identifizieren.

Methode findet einen solchen Fluss (falls die Kapazitäten ganzzahlig sind). Genauso könnte man auch nach einem Schnitt (Q, S) fragen, der minimale Kapazität hat (also $c(Q, S) \leq c(Q', S')$ für alle Schnitte (Q', S') erfüllt). Überraschenderweise haben wir dieses Problem schon gelöst! Wenn die FF-Methode anhält, gibt es keinen fvW in G_f mehr. Wir betrachten den Schnitt (Q, S) im Beweisteil „(ii) $\Rightarrow$ (iii)“ und behaupten, dass er minimale Kapazität hat. Wieso? Betrachte einen beliebigen Schnitt (Q', S'). Nach Lemma 1.2.5 gilt $w(f) \leq c(Q', S')$, aber wir haben auch $c(Q, S) = w(f)$. Also gilt $c(Q, S) \leq c(Q', S')$. Daher hat (Q, S) minimale Kapazität. – Anders gesagt: Jeder Algorithmus, der einen maximalen Fluss berechnet, berechnet praktisch gleich einen minimalen Schnitt mit. Und: Der größtmögliche Wert eines Flusses ist gleich der kleinstmöglichen Kapazität eines Schnittes, oder: $\max\{w(f) \mid f \text{ Fluss}\} = \min\{c(Q, S) \mid (Q, S) \text{ Schnitt}\}$, oder: „MaxFlow = MinCut“. Diese Erscheinung ist charakteristisch für Probleme, die sich als „lineares Programm“ formulieren lassen. Das MaxFlow-Problem gehört in diese Klasse; das MinCut-Problem ist zum Flussproblem „dual“. (Details zur Dualität erfährt man in Veranstaltungen, die sich mit linearer Programmierung befassen.)

1.3 Der Algorithmus von Edmonds-Karp

Die Ford-Fulkerson-Methode kann auch für kleine Graphen sehr große Laufzeit haben, die mit dem Wert des maximalen Flusses wächst. Dies wird durch das folgende einfache *Beispiel* deutlich: $G = (V, E)$ hat 4 Knoten q, s, a und b, und Kanten $(q, a), (q, b), (a, s), (b, s)$ mit Kapazität M sowie eine Kante (a, b) mit Kapazität 1. Wenn man als fvW nie (q, a, s) oder (q, b, s) benutzt, sondern immer nur (q, a, b, s) bzw. (q, b, a, s), je nachdem, welcher gerade zur Verfügung steht, dann benötigt die Ermittlung des maximalen Flusses vom Wert $2M$ genau $2M$ Runden. Die ersten Schritte eines solchen Ablaufs sind in Abb. 8 dargestellt.

Es gibt mehrere Möglichkeiten, diese „Falle“ zu vermeiden, die durch das Zulassen völlig beliebiger flussvergrößernder Wege zustandekommt. Eine der einfachsten führt zum Algorithmus von Edmonds und Karp, dargestellt als Algorithmus 1.3.1. Dieser verlangt einfach, einen *möglichst kurzen* fvW zu wählen, also einen mit möglichst wenigen Kanten. Dies erreicht man am effizientesten dadurch, dass man den fvW im RNW mittels *Breitensuche* (Algorithmus BFS) vom Knoten q aus sucht und beim ersten Erreichen der Senke s aufhört.

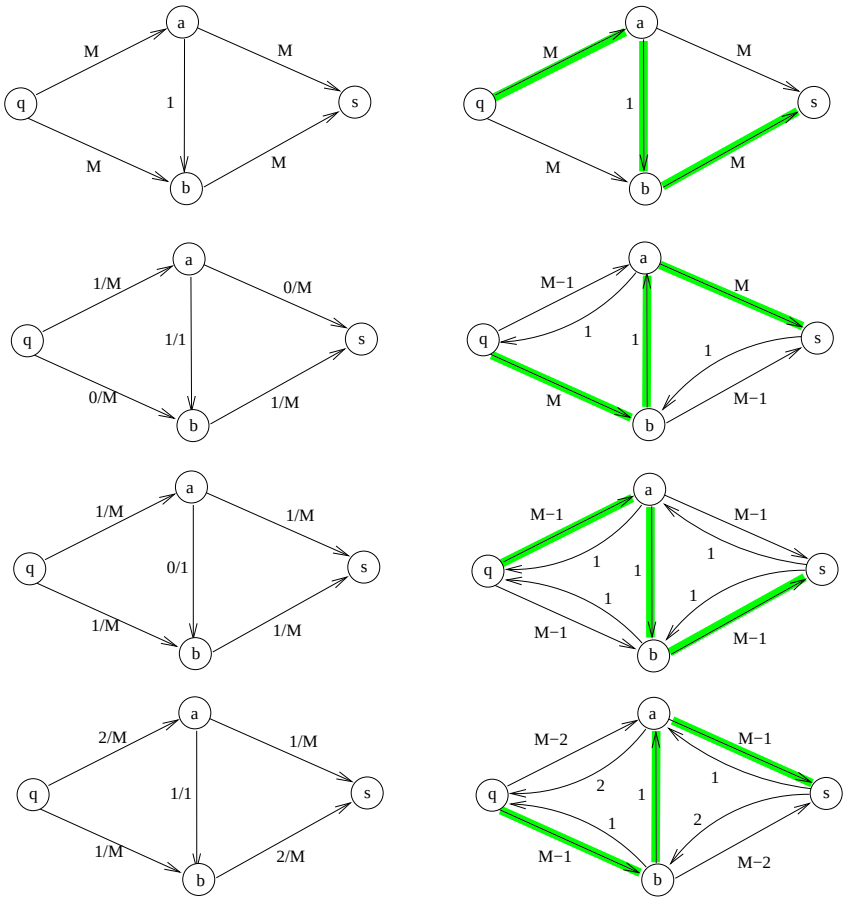


Abbildung 8: Langsame Flussvergrößerung mit der FF-Methode. Erste Zeile: Flussnetzwerk und Restnetzwerk mit FvW. Zweite und dritte Zeile: jeweils ein Fluss und das zugehörige RNW mit FvW. Letzte Zeile: Fluss mit Wert 3 nach drei Runden. Wenn man in derselben Weise fortfährt, erhält man den maximalen Fluss mit Wert $2M$ erst nach $2M$ Flussvergrößerungen. **(Bild korrigiert 19.10.2019)**

Algorithmus 1.3.1 (**Edmonds-Karp**)**.**

INPUT: Flussnetzwerk $G = (V, E, q, s, c)$

METHODE:

```
1   f := der Nullfluss;
2   repeat
3     Berechne das Restnetzwerk G_f;
4     Führe in G_f BFS mit Startknoten q aus; stoppe, sobald s erreicht;
5     Falls s nicht erreicht: return f;
6     Sonst: Der von der BFS gefundene Weg p von q nach s ist fvW;
7     f := f + f_p;   //    vergrößere Fluss entlang p
```

In Abb. 9 ist ein Beispieldurchlauf angegeben. Dabei sind nur die Flüsse im Netzwerk G gezeichnet – die Restnetzwerke muss man jeweils erschließen.

Wir analysieren nun die Laufzeit des Edmonds-Karp-Algorithmus.

Satz 1.3.2. *Sei G ein FNW. Dann hat der Algorithmus von Edmonds-Karp auf G Laufzeit $O(m^2n) = O(n^5)$. Dies gilt sogar, wenn die Kapazitäten beliebige Zahlen in $\mathbb{R}_0^+$ sind.*

Beweis: Jeder Aufbau des Restnetzwerks und jede Breitensuche kann in Zeit $O(m)$ bewerkstelligt werden. Wir müssen also nur zeigen, dass es maximal $O(nm)$ Iterationen der **repeat**-Schleife („*Runden*“) geben kann, bevor sie (über den **return**-Befehl) verlassen wird. Wir definieren dazu:

$$d_t(v) := \text{Abstand von } v \text{ von } q \text{ im RNW } G_f^{(t)} \text{ nach Runde } t, \text{ für } v \in V.$$

Auch hier bedeutet „Abstand“ natürlich die Länge eines Weges von q zu v mit möglichst wenigen Kanten. $d_t(v) = \infty$ heißt, dass es in $G_f^{(t)}$ keinen q-v-Weg gibt.

Der springende Punkt der Analyse ist, dass diese Abstände nie kleiner werden können.

Lemma 1.3.3. *Für jede Runde $t > 0$ und alle Knoten $v \in V$ gilt: $d_t(v) \geq d_{t-1}(v)$.*

Beweis: Wir betrachten $G_f^{(t-1)}$ und teilen V in die Breitensuche-Levels[3] ein:

$$V_\ell := \{v \in V \mid d_{t-1}(v) = \ell\}, \text{ für } \ell = 0, 1, 2, \ldots \text{ und } \ell = \infty.$$

Direkt aus der Definition folgt: Wenn $(u, v) \in E_f^{(t-1)}$, dann gilt $d_{t-1}(v) \leq d_{t-1}(u) + 1$. D. h.: Entlang von Kanten in $E_f^{(t-1)}$ steigt die Levelnummer um nicht mehr als 1 an. (Wenn $d_t(u) = \infty$, ist diese Ungleichung trivial, mit der Vereinbarung $\infty + 1 = \infty$.)

[3]Diese Levels muss der Algorithmus nicht berechnen; wir brauchen sie nur für die Analyse.

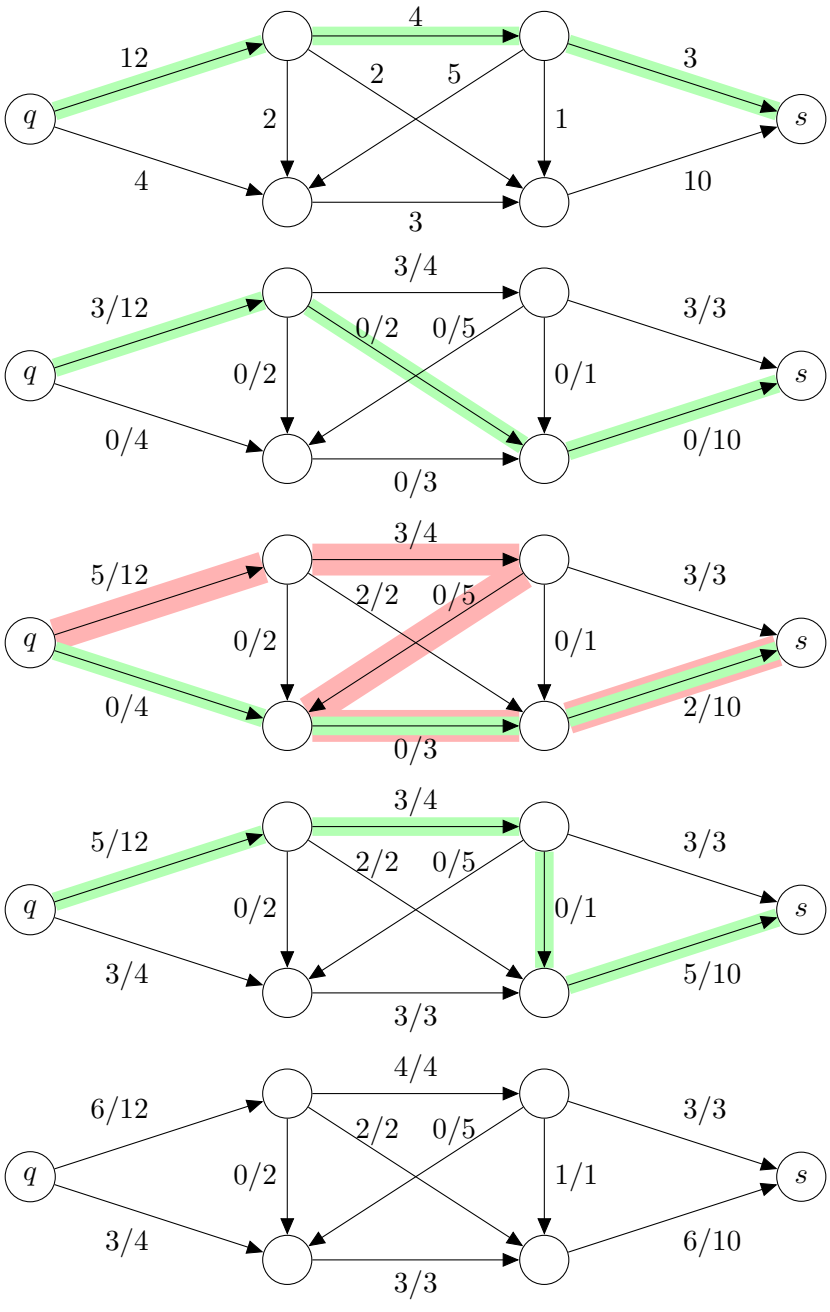


Abbildung 9: Ein Beispieldurchlauf des Edmonds-Karp-Algorithmus.

In Runde t wird der Fluss durch eine Flussvergrößerung $f := f + f_p$ entlang eines kürzesten fvW $p = (u_0, u_1, \ldots, u_L)$ mit $u_0 = q$ und $u_L = s$ vergrößert. Für jeden Knoten u_ℓ auf diesem Weg gilt $u_\ell \in V_\ell$. (Denn: Der Edmonds-Karp-Algorithmus schreibt vor, dass p ein *kürzester* fvW ist. Daher ist auch jedes Anfangsstück von p ein kürzester Weg zu seinem Endknoten u_ℓ.) Durch die Flussvergrößerung ändert sich das Restnetzwerk von $E_f^{(t-1)}$ zu $E_f^{(t)}$. Dabei können Kanten neu erscheinen. (Dass Kanten auch verschwinden, können wir hier ignorieren.) Wenn eine Kante (u, v) in $E_f^{(t)}$ neu erscheint, dann muss die Gegenkante (v, u) auf p liegen, also $v = u_{\ell-1}$ und $u = u_\ell$ gelten, für ein $\ell \in \{1, \ldots, L\}$. Das heißt $d_{t-1}(v) = \ell - 1 = d_{t-1}(u) - 1$: Entlang einer neuen Kante (u, v) sinkt die Levelnummer sogar strikt!

Wir erhalten: Entlang einer beliebigen Kante $(u, v) \in E_f^{(t)}$ (alt oder neu hinzugekommen) steigt die Levelnummer (bezüglich $V_0, V_1, \ldots$) nicht um mehr als 1 an.

Daraus folgt: (1) Für ein beliebiges $v \in V_\ell$, $\ell < \infty$, benötigt jeder Weg in $E_f^{(t)}$ von q ($\in V_0$) nach v mindestens ℓ Kanten, also gilt $d_t(v) \geq \ell = d_{t-1}(v)$. (2) Wenn $v \in V_\infty$, gibt es in $E_f^{(t)}$ überhaupt keinen Weg von q nach v, also gilt auch $d_t(v) = \infty$. □

Beweis von Satz 1.3.2 (Forts.): Wir betrachten ein festes Paar (u, v) mit $(u, v) \in E$ oder $(v, u) \in E$, das heißt, eine mögliche Kante in einem Restnetzwerk $G_f^{(t)}$. Die Kante kann im Verlauf des Algorithmus mehrfach im Restnetzwerk auftauchen und wieder verschwinden. Wir wollen zeigen, dass sie nicht allzu oft verschwinden kann. Dabei können wir $u \neq q$ voraussetzen, also $d_t(u) \geq 1$ stets, da Kanten (q, v) nur einmal verschwinden, aber nie wieder neu erscheinen können. (Kanten (v, q) können nicht auf dem fvW liegen.)

Wenn Kante (u, v) in Runde t aus dem RNW verschwindet, ist sie Flaschenhalskante, liegt also auf dem fvW p für Runde t. Sei $\ell = d_{t-1}(u)$ und $\ell + 1 = d_{t-1}(v)$. In einer späteren Iteration t' kann (u, v) wieder im Restnetzwerk auftauchen. Dazu muss (v, u) auf dem fvW p' für Runde t' liegen, also muss $d_{t'-1}(v) = \ell'$ und $d_{t'-1}(u) = \ell' + 1$ gelten, für ein ℓ'. Wenn nun in einer Runde $t'' > t'$ die Kante (u, v) erneut verschwindet, muss $d_{t''-1}(u) = \ell''$ und $d_{t''-1}(v) = \ell'' + 1$ gelten, für ein ℓ''. Da nach Lemma 1.3.3 der Abstand eines jeden Knotens von q von Runde zu Runde nicht sinken kann, gilt

$$\ell + 1 = d_{t-1}(v) \leq d_{t'-1}(v) = \ell' \text{ und } \ell' + 1 = d_{t'-1}(u) \leq d_{t''-1}(u) = \ell'',$$

also $d_{t''-1}(u) = \ell'' \geq \ell + 2 = d_{t-1}(u) + 2$. Bei jedem Verschwinden von (u, v) ist der Abstand von u von q also um mindestens 2 angestiegen.

Da wir mit $d_0(u) \geq 1$ beginnen, gilt $d_{t-1}(u) \geq 2k - 1$, wenn (u, v) in Runde t zum k-ten Mal verschwindet. Die maximale Distanz ist $n - 1$. Also gilt $d_{t-1}(u) \leq n - 1$, und es folgt $k \leq n/2$. Also kann Kante (u, v) nicht häufiger als $\lfloor \frac{1}{2} n \rfloor$-mal aus dem RNW verschwinden.

Nun beobachten wir, dass es in jeder Runde mindestens eine Flaschenhalskante gibt, dass also eine Kante aus dem RNW verschwindet, nämlich eine der Kanten e auf dem benutzten fvW p, die $c(p) = \text{rest}_f(e)$ erfüllt. Jede der $2m$ Kanten (u, v) mit $(u, v) \in E$ oder $(v, u) \in E$ kann aber nur $\lfloor \frac{1}{2}n \rfloor$-mal Flaschenhalskante sein, also gibt es maximal $2m\lfloor \frac{1}{2}n \rfloor \leq mn$ Runden. □

Bemerkung 1: Die Schleife im Algorithmus kann nur auf eine Weise verlassen werden, nämlich indem festgestellt wird, dass im RNW G_f die Senke nicht mehr erreichbar ist. Das MaxFlow-MinCut-Theorem (Satz 1.2.6) besagt dann, dass der nun vorliegende Fluss f maximal ist.

Bemerkung 2: Durch die kleine Änderung an der Ford-Fulkerson-Methode, die zum Algorithmus von Edmonds-Karp führte, haben wir einen großen Qualitätssprung gemacht, in zweierlei Hinsicht: (i) Die Rechenzeit ist unabhängig von der Größe der Kapazitäten (solange man annimmt, dass man in konstanter Zeit addieren, subtrahieren und das Maximum zweier Zahlen bilden kann). (ii) Der neue Algorithmus funktioniert auch, wenn die Kapazitäten beliebige reelle Zahlen sind. Die Ganzzahligkeitsannahme ist überflüssig geworden.

Bemerkung 3: Weil der Algorithmus von Edmonds-Karp nur eine spezielle Variante der FF-Methode ist, hat er alle Eigenschaften dieser Methode, insbesondere liefert er einen ganzzahligen optimalen Fluss, wenn die Kapazitäten ganzzahlig sind.

Im nächsten Abschnitt wollen wir noch Verfahren zur Laufzeitverbesserung kennenlernen.

1.4 Niveaunetzwerke und Sperrflussmethode

Es kostet $O(m)$ Zeit, ein Restnetzwerk G_f aufzubauen und darin einen kürzesten q-s-Weg zu suchen. Der Algorithmus von Edmonds und Karp nutzt dieses dann eigentlich nur sehr wenig: Er konstruiert einen einzigen flussvergrößernden Weg in G_f, dann wird G_f durch ein neues Restnetzwerk ersetzt und es wird erneut nach einem fvW gesucht. Die nächste Idee zur Laufzeitverbesserung besteht darin, auf der Basis *eines* Restnetzwerkes gleich einen größeren Flussvergrößerungs-Sprung zu machen, indem man nicht nur einen einzelnen Weg betrachtet, sondern in einer Runde den Fluss gewissermaßen entlang mehrerer Wege gleichzeitig erhöht. Allerdings bleibt man dabei, dass man sich nur für kürzeste q-s-Wege im (aktuellen) RNW interessiert. Dazu lässt man aus dem RNW alle Kanten weg, die nicht zu solchen kürzesten Wegen gehören.

1.4.1 Niveaunetzwerke und Sperrflüsse

Definition 1.4.1. *Seien f ein Fluss in einem FNW G und G_f das zugehörige RNW. Sei L der Abstand von q und s in G_f. Das* ***Niveaunetzwerk (NNW)*** *$G'_f = (V'_f, E'_f)$ besteht aus den Knoten und Kanten, die auf Wegen von q nach s der Länge L liegen. Jede Kante in E'_f wird mit ihrer Restkapazität* $\mathrm{rest}_f(u, v)$ *versehen.* Niveau V_ℓ *besteht aus den Knoten in V'_f, die Abstand ℓ von q haben. (Jede Kante (u, v) läuft von einem Niveau $V_{\ell-1}$ zum nächsten, V_ℓ.)*

Wir beschreiben, wie man G'_f aus G_f berechnet. Zunächst führt man in G_f eine Breitensuche von q aus durch. Diese BFS ordnet jedem Knoten v sein Level (sein Niveau) $d(v)$ zu. Sie kann abgebrochen werden, wenn das Level L von s festgestellt wurde und die Nachfolgerlisten aller Knoten auf Level $L - 1 = d(s) - 1$ bearbeitet worden sind. Schon in der Breitensuche kann man auch die Kanten in G_f, die nicht von einem Level $\ell - 1$ zum nächsten Level ℓ gehen, weglassen. Dies liefert ein Zwischenergebnis G''_f. Nun bildet man den Umkehrgraphen von G''_f (alle Kanten herumdrehen) und führt darin Breitensuche von s aus durch. Knoten und Kanten, die dabei nicht erreicht werden, werden ebenfalls weggelassen.

Was übrigbleibt, ist das Niveaunetzwerk G'_f. Zudem kann man aus der Berechnung für jeden Knoten v eine Liste $\mathrm{Out}(v)$ seiner Ausgangskanten (v, u) (zu Knoten u, die um ein Niveau höher als v liegen) und eine Liste $\mathrm{In}(v)$ seiner Eingangskanten (u, v) (von Knoten u, die um ein Niveau niedriger als v liegen) erhalten.

Beispiel: In Abbildung 10 ist dargestellt, wie aus einem Flussnetzwerk G mit Fluss f ein Niveaunetzwerk entsteht.

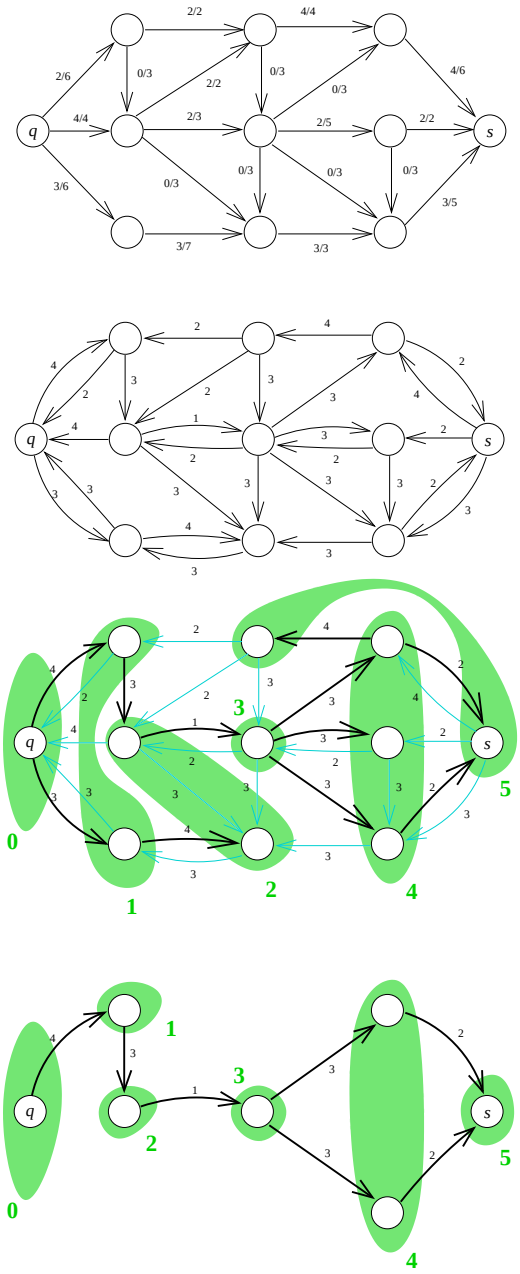


Abbildung 10: Zeile 1: Ein FNW mit Fluss. Zeile 2: Das zugehörige RNW. Zeile 3: das Ergebnis der Breitensuche von der Quelle aus im RNW, mit Levels (fett: Kanten von einem Level zum nächsten); Zeile 4: das resultierende Niveaunetzwerk.

Die Idee ist nun, einen Fluss in G'_f zu berechnen, der entlang von flussvergrößernden Wegen *der Länge L* nicht mehr vergrößert werden kann. Das bedeutet intuitiv, dass alle Wege dieser Länge „verstopft“ sind.

Definition 1.4.2. *Sei f ein Fluss in einem FNW G, und sei $G'_f = (V'_f, E'_f)$ das zugehörige Niveaunetzwerk. Ein* ***Sperrfluss*** *(*engl.: *blocking flow) für G'_f ist ein Fluss $\varphi\colon E'_f \to \mathbb{R}_0^+$ mit folgender Eigenschaft: Jeder q-s-Weg in G'_f hat mindestens eine* ***gesättigte*** *Kante, das ist eine Kante (u, v), die $\varphi(u, v) = \mathrm{rest}_f(u, v)$ erfüllt.*

Beispiel: Die Bilder in Abb. 11 zeigen ein Niveaunetzwerk mit Restkapazitäten und einen Sperrfluss in diesem Niveaunetzwerk.

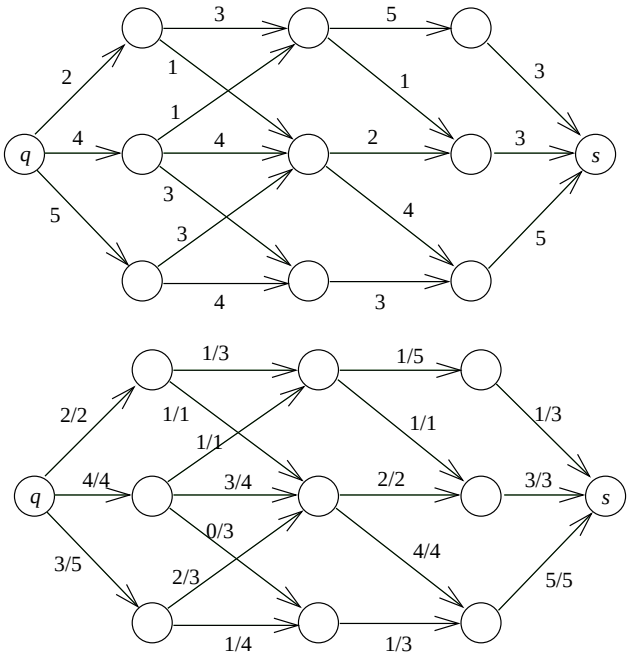


Abbildung 11: Oben: Ein Niveaunetzwerk. Unten: Ein Sperrfluss.

Bemerkung: Die Definition verlangt im Wesentlichen, dass man φ mit der FF-Methode nicht mehr vergrößern kann, wenn man ausschließlich Vorwärtskanten betrachtet. Diese Einschränkung führt zu der Frage, ob es Sperrflüsse gibt, die keine maximalen Flüsse für ihr NNW G'_f sind. Dies ist tatsächlich der Fall (Übung).

1.4.2 Die Sperrflussmethode mit Analyse

Algorithmus 1.4.3 (**Sperrflussmethode**).
INPUT: Flussnetzwerk $G = (V, E, q, s, c)$

METHODE:
1 $f :=$ der Nullfluss;
2 **repeat**
3 Berechne das Niveaunetzwerk G'_f;
4 Falls dabei in der ersten BFS s nicht erreicht: **return** f;
5 Konstruiere einen Sperrfluss φ in G'_f;
6 $f := f + \varphi$; // siehe Gleichung (1)

Algorithmus 1.4.3 gibt die Sperrflussmethode wieder. (Wir sprechen hier von einer „Methode“, weil eine wesentliche Komponente, nämlich die Bestimmung der Sperrflüsse, noch offen bleibt.) Wie vorher sehen wir folgendes: Wenn die Schleife terminiert, dann kann das nur daran liegen, dass im nun vorliegenden RNW s von q aus nicht mehr erreichbar ist, also nach dem MaxFlow-MinCut-Theorem, dass der nun vorliegende und ausgegebene Fluss maximal ist. Es bleibt zu zeigen, dass die Schleife (schnell) terminiert. Schließlich muss man überlegen, wie schnell man Sperrflüsse berechnen kann.

Lemma 1.4.4. *Die Sperrflussmethode terminiert nach maximal* $n-1$ *Sperrflussberechnungen.*

Beweis: Wir werden zeigen, dass die Anzahl der Levels im NNW in jedem Schleifendurchlauf strikt zunimmt. Da dieser Abstand anfangs mindestens 1 ist und nicht größer als $n-1$ sein kann, wenn s von q aus erreichbar ist, kann es nicht mehr als $n-1$ Sperrflussberechnungen und anschließende Flussvergrößerungen geben.

Sei dazu $L := L_{t-1} := d_{t-1}(s)$ der Abstand von q nach s im alten RNW $G_f^{(t-1)}$, für $t \geq 1$. Dann ist L auch die Anzahl der Levels im NNW G'_f in Iteration t. Zu G'_f wird der Sperrfluss φ berechnet, und der Fluss wird mittels $f := f + \varphi$ vergrößert. Wir betrachten nun einen beliebigen q-s-Weg $p = (v_0, v_1, v_2, \ldots, v_r)$ mit $v_0 = q$, $v_r = s$ im RNW $G_f^{(t)}$ *nach* Ausführung dieser Flussvergrößerung. (Falls es keinen solchen Weg gibt, stoppt der Algorithmus in der nächsten Runde ohne Sperrflussberechnung, es ist also nichts zu zeigen.) Wir behaupten, dass r größer als L sein muss.

Es gibt zwei Fälle:

1. Fall: Sämtliche Kanten von p sind im alten RNW $G_f^{(t-1)}$ enthalten. Weil in $G_f^{(t-1)}$ Knoten q und s Abstand L haben, muss $r \geq L$ gelten. *Annahme*: $r = L$. Dann ist p ein kürzester q-s-Weg in $G_f^{(t-1)}$, liegt also komplett in G'_f. Nach der Definition eines Sperrflusses ist in φ mindestens eine Kante e von p gesättigt, d. h. es gilt $\varphi(e) = \text{rest}_f^{(t-1)}(e)$. Nach der Definition von $f + \varphi$ ist dann e nicht im neuen Restnetzwerk $G_f^{(t)}$, ein Widerspruch. Daher gilt in diesem Fall $r > L$.

2. Fall: p enthält mindestens eine Kante (v_{i-1}, v_i), die *nicht* im alten RNW $G_f^{(t-1)}$ enthalten ist. Jede solche Kante kommt durch die Flussvergrößerung ins Restnetzwerk, und das heißt, dass v_i in Niveau $V_{\ell-1}$ und v_{i-1} in Niveau V_ℓ von G'_f liegt, für ein $\ell \in \{1, \ldots, L\}$, in anderen Worten $d_{t-1}(v_i) = d_{t-1}(v_{i-1}) - 1$. Solche Kanten gehen also aus Sicht des alten Niveaunetzwerks zurück in Richtung Quelle! Für Kanten (v_{i-1}, v_i) auf p, die im alten RNW $G_f^{(t-1)}$ enthalten sind, gilt $d_{t-1}(v_i) \leq d_{t-1}(v_{i-1}) + 1$. Weg p mit mindestens einer neuen Kante kann daher mit seinen r Schritten maximal den Abstand $d_{t-1}(v_r) \leq r - 2$ erreichen. Es ist aber $v_r = s$, und $d_{t-1}(s) = L$, also gilt $r \geq L + 2$. □

Der *Algorithmus von Dinitz* ist eine spezielle Version der Sperrflussmethode, in der Sperrflüsse iterativ gefunden werden, wie folgt: Man gibt den Kanten im NNW G'_f ihre Restkapazitäten $\text{rest}_f(u, v)$ und startet mit $\varphi = \varphi_0 \equiv 0$, dem Nullfluss. Nun verfährt man im Wesentlichen wie die Ford-Fulkerson-Methode auf G'_f, allerdings werden nie Rückwärtskanten betrachtet. Dies führt nach einer Reihe von Runden zu einem Sperrfluss. Es ist nicht besonders schwierig, dies so zu organisieren, dass eine Sperrflussberechnung Zeit $O(nm)$ benötigt (Übung). Da es maximal $n-1$ Sperrflussberechnungen gibt, ist die Gesamtlaufzeit des Algorithmus von Dinitz $O(n^2m) = O(n^4)$.

Es gibt noch weitere raffinierte Methoden für die Berechnung von Sperrflüssen. Eine heißt „Backward-Forward-Propagation“. Diese berechnet einen Sperrfluss in Zeit $O(n^2)$, und damit benötigt die gesamte Flussberechnung Zeit $O(n^3)$. Sleator und Tarjan haben 1980/83 eine Datenstruktur („dynamic trees“) angegeben, mit deren Hilfe sich die Konstruktion eines Sperrflusses sogar in Zeit $O(m \log n)$ bewerkstelligen lässt. Dies führt dann zu einem Flussalgorithmus mit Laufzeit $O(nm \log n)$.

Die Methode „Backward-Forward-Propagation“ wird im WS 2019/20 nicht behandelt, ist also nicht prüfungsrelevant.

1.5 Der Preflow-Push-Ansatz

Wir hatten folgende Veranschaulichung für ein Flussproblem betrachtet:

Situation: („Fanströme“) In Dortkirchen gibt es ein Straßensystem und 60 000 Fußballfans, die am Bahnhof ankommen und zu Fuß zum Stadion wollen. Für jede Straße ist eine maximale Gesamtanzahl an Personen festgelegt, die sie (in eine festgelegte Richtung) passieren können. Wie soll die Polizei die Menschenmassen aufteilen und lenken, so dass alle ankommen? Für die Abbildung siehe Abschnitt 1.1.

Wenn wir fragen, wie viele Fans maximal zum Ziel gelangen können, landen wir beim MaxFlow-Problem. Wir betrachten in diesem Abschnitt eine Strategie, die sich vom Ford-Fulkerson-Ansatz („flussvergrößernde Wege“) unterscheidet.

Im Fanströme-Beispiel sieht die Grundidee so aus. Es wird eine Simulation auf dem Papier durchgeführt, bei der man zunächst einmal so viele Fans in das Straßennetz hineinschickt wie auf die Straßen passen, die direkt vom Bahnhof wegführen. Man erlaubt vorläufig, dass Fans auf Plätzen und Straßenkreuzungen „herumstehen“, dass also mehr dort angekommen sind als wieder weggegangen sind („Überschuss“).

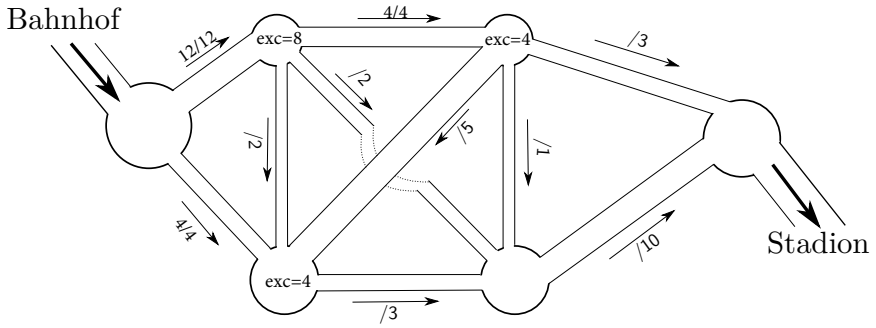


Abbildung 12: Beispiel Fanströme. Straßenkapazitäten und testweise Einleitung von Fanströmen. Erzeugt Überschuss in einigen Knoten.

Nun verschiebt man geschickt Gruppen zwischen Plätzen hin und her und schreibt mit, wie viele Fans „netto“ durch eine Straße gegangen sind (wenn welche zurückgehen, subtrahiert man wieder). Dieser Nettowert muss immer innerhalb der Kapazität der jeweiligen Straße liegen, und man kann von einem Platz nicht mehr Fans abziehen als gerade da sind. Es kann passieren, dass auch Fans zum Bahnhof zurückgeschickt werden, also schließlich gar nicht ins Netzwerk eintreten dürfen. Wenn man es schließlich geschafft hat, alle Fans beim Stadion abzuliefern oder zum Bahnhof zurückzuschicken, ergeben die (Fluss-)Werte an den Straßen einen legalen Plan. Wenn man die Verschiebungen geschickt organisiert, ist das Ergebnis sogar ein maximaler

Fluss.

Wir modellieren das Vorgehen mathematisch. Dazu lockern wir die Kirchhoff-Bedingung und erhalten so den Begriff eines Präflusses („*preflow*“).

Definition 1.5.1. *Ein* ***Präfluss*** *in einem Flussnetzwerk $G = (V, E, q, s, c)$ ist eine Funktion $f\colon E \to \mathbb{R}_0^+$, die jeder Kante e einen Wert $f(e)$ zuordnet, wobei gilt:*

(i) *Mit*

$$f_{\mathrm{In}}(v) := \sum_{(u,v)\in E} f(u,v), \qquad f_{\mathrm{Out}}(v) := \sum_{(v,u)\in E} f(v,u).$$

gilt $f_{\mathrm{In}}(v) \geq f_{\mathrm{Out}}(v)$ für alle $v \in V - \{q\}$.

(ii) *$0 \leq f(e) \leq c(e)$, für alle $e \in E$.*

(Der einzige Unterschied zum Begriff des Flusses ist das „$\geq$“ in Bedingung (i).)

Es ergeben sich „Überschüsse“ (*excess*)

$$\mathrm{ex}(v) := f_{\mathrm{In}}(v) - f_{\mathrm{Out}}(v)$$

in den Knoten $v \in V$. Da q keine Eingangskanten hat, gilt stets $f_{\mathrm{In}}(q) = 0$, also $\mathrm{ex}(q) \leq 0$. Die Quelle ist aber (nach Bedingung (i)) der einzige Knoten, der negativen Überschuss haben kann.

Ein Knoten $v \in V - \{q, s\}$ heißt *aktiv*, wenn $\mathrm{ex}(v) > 0$ ist.

Die Grundidee von Preflow-Push-Algorithmen ist, einen Präfluss zu definieren und durch geeignete Operationen schrittweise so zu verändern, dass schließlich alle Überschüsse in Knoten $v \in V - \{q, s\}$ verschwunden sind. Dann ist ein Fluss entstanden, der sich als maximal erweist, wenn man richtig vorgeht.

Die *Restkapazitäten* $\mathrm{rest}_f(u, v)$ für $(u, v) \in E$ und $(v, u) \in E$ werden für einen Präfluss genauso definiert wie für einen Fluss, ebenso das Restnetzwerk (RNW) G_f, das alle Kanten mit positiven Restkapazitäten enthält.

Um die Operationen zu steuern, benötigt man ein neues zentrales Konzept, so genannte *Höhenfunktionen*. Dabei stellt man sich vor, dass Knoten v auf einem bestimmten Level $h(v)$ platziert ist. Diese Niveaus (Höhen) ändern sich im Verlauf des Algorithmus.

Definition 1.5.2. *Gegeben sei ein* ***Präfluss*** *in einem Flussnetzwerk $G = (V, E, q, s, c)$. Eine* Höhenfunktion *für f ist eine Funktion $h\colon V \to \mathbb{N}$ mit:*

$$(u, v) \in E_f \quad \Rightarrow \quad h(u) \leq h(v) + 1.$$

Wenn man sich die Knoten entsprechend ihrer Höhenwerte auf Niveaus angeordnet denkt, bedeutet dies, dass Restnetzwerkkanten (mit positiver Restkapazität) nicht mehr als ein Niveau nach unten gehen dürfen. Das heißt:

- Wenn $(u, v) \in E$ und $h(u) \geq h(v)+2$, dann muss (u, v) *gesättigt* sein ($f(u, v) = c(u, v)$).
- Wenn $(v, u) \in E$ und $h(u) \geq h(v) + 2$, dann muss (v, u) *leer* sein ($f(v, u) = 0$).

Preflow-Push-Algorithmen manipulieren Paare (f, h), wobei f Präfluss und h zulässige Höhenfunktion ist. Das Ziel ist erreicht, wenn f ein Fluss geworden ist.

Die einfachste Weise, ein Paar (f, h) zu finden, mit dem man beginnen kann, ist folgende: Setze die Quelle auf Höhe n, alle anderen Knoten auf Höhe 0, und schicke über die Ausgangskanten von q den maximal erlaubten Fluss.

Algorithmus 1.5.3 (**Initialisierung**)**.**

Data: *Flussnetzwerk* $G = (V, E, q, s, c)$ *mit* $n = |V|$
Result: *Initialisierung für Preflow-Push*
1 $f(q, v) \leftarrow c(q, v)$ *für Kanten* (q, v) *aus der Quelle;*
2 $f(u, v) \leftarrow 0$ *für* $(u, v) \in E$ *mit* $u \neq q$*;*
3 $h(q) \leftarrow n$*;*
4 $h(v) \leftarrow 0$ *für* $v \in V - \{q\}$*;*

Wir beobachten: f ist ein Präfluss und h ist eine Höhenfunktion für f. (Die einzigen Flussnetzwerkkanten, die steil nach unten gehen, sind gesättigt.) Überschüsse:

- $\mathrm{ex}(v) = c(q, v)$, für v mit $(q, v) \in E$.
- $\mathrm{ex}(v) = 0$, für $v \in V - \{q\}$ mit $(q, v) \notin E$.
- $\mathrm{ex}(q) = -C_G = -\sum_{(q,v)\in E} c(q, v)$.

Aktiv sind also die unmittelbaren Nachfolger von q. Die Summe aller Überschüsse (inklusive dem von q) ist stets 0.

Bemerkung: Bei der Initialisierung verwendet man auch oft eine Variante, die zu einer leichten Beschleunigung führt. Anstatt Knoten $v \in V - \{q\}$ pauschal auf Höhe 0 zu setzen, kann man ihn auch auf Level $d_G(v, s)$ setzen (den Abstand von v zur Senke im Flussnetzwerk G, was ganz am Anfang im Teil $V - \{q\}$ zum Restnetzwerk G_f identisch ist). Die Zahlen $d_G(v, s)$ können leicht durch Breitensuche im Umkehrgraphen (ohne q) von s aus ermittelt werden. Man sieht sofort, dass die Bedingung an die Höhenfunktion erfüllt ist: Wenn $(u, v) \in E$ gilt, dann ist $d_G(u, s) \leq d_G(v, s) + 1$.

Es gibt zwei zentrale Operationen zum Umbau des Paares (f, h). Eine („push“) ändert den Fluss über eine Kante (u, v), die andere („relabel“) ändert den Höhenwert eines Knotens v. (Im Gegensatz zu den FF-Methoden sind diese Manipulationen sehr lokal.) Wegen dieser Bezeichnungen heißen Preflow-Push-Verfahren oft auch *Push-Relabel*-Verfahren.

Die erste Operation verändert den Flusswert auf einer Kante. Zweck ist, Überschuss von einem aktiven Knoten u zu einem Knoten v zu verschieben, entlang einer Kante $(u, v) \in E_f$, also einer Kante mit positiver Restkapazität. Diese Operation darf nur ausgeführt werden, wenn $h(u) = h(v) + 1$ gilt.

Intuition: Überschuss von u nach v verschieben kann man nur, wenn

- u höher sitzt als v,
- u Überschuss *hat*, d. h. $\mathrm{ex}(u) > 0$ gilt, und
- auf der Restnetzwerkkante (u, v) Platz ist, also $\mathrm{rest}_f(u, v) > 0$ gilt.

Die letzte Bedingung bedeutet für $(u, v) \in E$, dass (u, v) nicht gesättigt ist (*Vorwärtskante*, man kann noch mehr Fluss über die Kante schicken) und für $(v, u) \in E$, dass $f(v, u) > 0$ ist (Rückwärtskante, man kann den Fluss über (v, u) verringern). Aus der Definition des Konzepts „Höhenfunktion“ und der ersten und der letzten Bedingung ergibt sich, dass $h(u) = h(v) + 1$ gelten muss, wenn man Überschuss von u nach v verschieben will.

Der Fluss kann sich nicht um mehr als $\mathrm{ex}(u)$ und auch nicht um mehr als $\mathrm{rest}_f(u, v) > 0$ ändern.

Algorithmus 1.5.4 (push).

Data: f, h, Kante $(u, v) \in E_f$ mit: $\mathrm{ex}(u) > 0$ und $h(u) = h(v) + 1$

1 $\delta \leftarrow \min\{\mathrm{ex}(u), \mathrm{rest}_f(u, v)\}$;
2 **if** $(u, v) \in E$ **then** $f(u, v) \leftarrow f(u, v) + \delta$;
3 **if** $(u, v) \notin E$ **then** $f(v, u) \leftarrow f(v, u) - \delta$;
// (implizit:) $\mathrm{ex}(u) \leftarrow \mathrm{ex}(u) - \delta$;
// (implizit:) $\mathrm{ex}(v) \leftarrow \mathrm{ex}(v) + \delta$;
// (implizit:) $\mathrm{rest}_f(u, v) \leftarrow \mathrm{rest}_f(u, v) - \delta$;
// (implizit:) $\mathrm{rest}_f(v, u) \leftarrow \mathrm{rest}_f(v, u) + \delta$;

Es gibt zwei Möglichkeiten:

- Wenn $\mathrm{ex}(u) \geq \mathrm{rest}_f(u, v)$ ist, verschieben wir $\mathrm{rest}_f(u, v)$ Überschuss. Dadurch wird $\mathrm{rest}_f(u, v) = 0$: die Kante (u, v) verschwindet aus dem Restnetzwerk. Wir nennen dies eine *saturierende push-Operation.*

- Wenn $\text{ex}(u) < \text{rest}_f(u,v)$ ist, verschieben wir $\text{ex}(u)$ viel Überschuss. Dadurch wird $\text{ex}(u) = 0$, und u wird inaktiv. Wir nennen dies eine *nichtsaturierende push-Operation*. (Warnung: Später kann u auch wieder aktiv werden, nämlich wenn Überschuss von einem anderen Knoten nach u verschoben wird.)

Man beachte, dass durch diese Operation $\text{rest}_f(u,v)$ um δ sinkt und $\text{rest}_f(v,u)$ um δ wächst.

Lemma 1.5.5. (a) *Durch die Ausführung von push(u,v) ändert sich die Eigenschaft, dass f ein Präfluss ist, nicht.*

(b) *Durch die Ausführung von push(u,v) ändert sich die Eigenschaft, dass h eine Höhenfunktion ist, nicht.*

Beweis: (a) ist offensichtlich. (b) Der rest_f-Wert auf der Gegenkante (v,u) steigt; dadurch könnte (v,u) neu ins RNW gelangen. Dies ist für h kein Problem, da $h(v) = h(u) - 1 \leq h(u) + 1$ gilt. Auf den anderen Kanten ändert sich nichts. □

Es kann passieren, dass keine push-Operation ausführbar ist, obwohl es Knoten mit Überschuss gibt. Woran kann das liegen? Knoten sitzen auf bestimmten Höhen. Wenn u Überschuss hat (also $\text{ex}(u) > 0$ gilt), aber im Restnetzwerk bezüglich der Höhenfunktion in einem lokalen Minimum sitzt (also $h(u) \leq h(v)$ für alle $(u,v) \in E_f$ gilt), kann man diesen Überschuss nicht bewegen. In diesem Fall werden aktive Knoten u vorsichtig angehoben, so dass h Höhenfunktion bleibt, aber schließlich doch erreicht wird, dass ein aktiver Knoten u eine Ausgangskante (u,v) hat, auf der eine push-Operation zulässig ist. Dafür gibt es die Operation *relabel*, die nur die Höhenfunktion verändert: Wenn für einen aktiven Knoten u gilt, dass $h(u) \leq h(v)$ für alle v mit $(u,v) \in E_f$, dann können wir die Höhe von u um 1 vergrößern.

Im Bild des Röhrensystems, durch das Wasser geleitet werden soll, lässt sich der Zweck des Anhebens ebenfalls recht anschaulich beschreiben. Wir stellen uns vor, die Knoten hätten Reservoirs, in denen Wasser temporär aufbewahrt werden kann. Knoten können auf einem Niveau zwischen 0 und $2n-1$ liegen. Anfangs wird die Quelle auf Niveau n gesetzt, alle anderen Knoten auf Niveau 0, und die maximale Menge an Wasser wird von q zu seinen Nachfolgern geschickt. In diesen steht nun Überschuss-Wasser. Man kann Wasser von u nach v schicken, aber nur wenn auf der Restnetzwerkkante (u,v) Platz ist *und* wenn u auf einem höheren Niveau sitzt als v (dann fließt nämlich das Wasser von selber nach unten). Der Zweck der nun folgenden relabel-Operation ist es, Knoten vorsichtig anzuheben, so dass dieser Fluss nach unten stattfinden kann, sei es nun in Richtung Senke oder auch zurück zur Quelle. Der Algorithmus besteht dann darin, push- und relabel-Operationen anzuwenden, bis nur noch Knoten s Überschuss hat, also alle anderen temporären Reservoirs leer sind.

Algorithmus 1.5.6 (**relabel**)**.**

Data: f, h, Knoten u mit $\mathrm{ex}(u) > 0$ und
$\forall v \in V : \mathrm{rest}_f(u, v) > 0 \Rightarrow h(u) \leq h(v)$
Result: Hebe aktiven Knoten u ein Niveau höher
1 $h(u) \leftarrow h(u) + 1$;

Durch den Aufruf **relabel**(u) ändert sich nichts daran, dass h Höhenfunktion für f ist: Für Kanten $(u, v) \in E_f$ gilt nachher $h(u) \leq h(v) + 1$; für Kanten $(w, u) \in E_f$ galt vorher $h(w) \leq h(u) + 1$, also nachher $h(w) \leq h(u)$.

Wir können nun die Klasse der Preflow-Push-Algorithmen beschreiben.

Algorithmus 1.5.7 (**Preflow-Push-Methode**)**.**

Data: Flussnetzwerk $G = (V, E, q, s, c)$
Result: Maximaler Fluss f für G
1 Initialisierung(G);
2 **while** *es gibt aktiven Knoten* u **do**
3 wähle einen aktiven Knoten u;
4 **if** $h(u) \leq h(v)$ *für alle Nachfolger* v *von* u *im RNW* G_f **then**
5 relabel(u)
6 **else**
7 wähle v mit $(u, v) \in E_f$ und $h(u) = h(v) + 1$; push(u, v);
8 **end**
9 **end**
10 **return** f;

In der Preflow-Push-Methode sind die Wahl des aktiven Knotens u und gegebenenfalls des Knotens v offengelassen. Es wird sich zeigen, dass diese Entscheidungen für die Terminierung und die Korrektheit des Ergebnisses irrelevant sind. Strategien für diese Entscheidungen bestimmen aber, wie schnell der Algorithmus abläuft.

Beispiel: Tafel, Internet.

`http://www.adrian-haarbach.de/idp-graph-algorithms/implementation/maxflow-push-relabel/index_en.html`

(Vorsicht: Quelle heißt s [„*source*“], Senke heißt t [„*target*“]. Etwas länger, aber instruktiv: Ablauf auf Graph „Jungnickel“.)

Im Beispiel sieht man, dass der Algorithmus in der Lage ist, einerseits Überschuss nach s zu verschieben und andererseits Überschuss, der nicht durch das Netzwerk hindurchpasst, zurück zu q zu schieben. Für den zweiten Teil ist es nötig, dass die

Höhenwerte auch größer als n werden, damit es zu q mit $h(q) = n$ „abwärts“ geht wie in der push-Operation gefordert.

Wir bemerken, dass sich die durch die Initialisierung festgelegten Werte $h(q) = n$ und $h(s) = 0$ nie ändern, da q und s nie aktiv sind.

Lemma 1.5.8. *Wenn h eine Höhenfunktion für Präfluss f ist, dann gibt es im RNW G_f keinen Weg von q nach s.*

Beweis: Da es n Knoten gibt, aber die Menge $\{0, 1, \ldots, n\}$ Größe $n + 1$ hat, muss es ein $k \in \{1, \ldots, n-1\}$ geben, so dass kein Knoten v mit $h(v) = k$ existiert. Ein Weg in G_f von q nach s müsste eine Kante $(u, v) \in E_f$ mit $h(u) > k > h(v)$ enthalten. Eine solche gibt es aber wegen der Definition einer Höhenfunktion nicht. □

Lemma 1.5.9. *Wenn h Höhenfunktion für f ist und kein Knoten aktiv ist, dann ist f ein maximaler Fluss für G.*

Beweis: Wenn kein Knoten aktiv ist, hat jeder Knoten $v \in V - \{q, s\}$ Überschuss 0, und das heißt, dass überall die Kirchhoff-Bedingung erfüllt ist. Daher ist f ein Fluss. Nach Lemma 1.5.8 gibt es im RNW G_f keinen Weg von q nach s. Das Max-Flow-Min-Cut-Theorem sagt dann, dass f maximal ist. □

Was wir noch zeigen müssen, ist, dass die Anwendung der Preflow-Push-Methode nach endlich vielen Schritten dazu führt, dass kein Knoten mehr aktiv ist. Die Strategie hierfür ist folgende: Wir zeigen, dass die Höhen von aktiven Knoten durch $2n-1$ nach oben beschränkt sind; Höhen können also nicht „in den Himmel wachsen“ und die Anzahl der relabel-Operationen ist endlich. Weiter stellen wir fest, dass zwei saturierende push-Operationen auf ein und derselben Kante (u, v) nur vorkommen können, wenn zwischendurch die Höhe $h(u)$ strikt angewachsen ist. Dies führt dazu, dass es nur endlich viele saturierende push-Operationen geben kann. Schließlich zeigen wir (mit einem weiteren Trick), dass auch die Anzahl der nichtsaturierenden push-Operationen endlich ist.

Lemma 1.5.10. *Wenn f Präfluss ist und $\mathrm{ex}(v) > 0$ gilt, dann gibt es im RNW G_f einen Weg von v nach q.*

Beweis: Sei X die Menge aller Knoten v, von denen aus es einen G_f-Weg nach q gibt, und sei $Y = V - X$. Es ist klar, dass $q \in X$ gilt. Es gibt keine Kante $(u, v) \in E_f$ mit $u \in Y$ und $v \in X$. Daraus folgt:

$$f(v, u) = 0 \text{ für } (v, u) \in E, v \in X, u \in Y. \tag{2}$$

Wir zeigen nun, dass die Summe der Überschüsse der Knoten in Y nicht positiv sein kann. Da Überschüsse (außer in $q \in X$) immer nichtnegativ sind, müssen dann alle

Knoten in Y Überschuss 0 haben. Daraus folgt die Behauptung: Jeder Knoten v mit positivem Überschuss muss in der Menge X liegen.

$$\begin{aligned}\sum_{u\in Y} \mathrm{ex}(u) &= \sum_{u\in Y} (f_{\mathrm{In}}(u) - f_{\mathrm{Out}}(u)) \\ &= \sum_{u\in Y} f_{\mathrm{In}}(u) - \sum_{u\in Y} f_{\mathrm{Out}}(u) \\ &= \sum_{\substack{(v,u)\in E \\ v\in V, u\in Y}} f(v,u) \;-\; \sum_{\substack{(u,v)\in E \\ u\in Y, v\in V}} f(u,v) \\ &\overset{(*)}{=} \sum_{\substack{(v,u)\in E \\ v\in X, u\in Y}} f(v,u) \;-\; \sum_{\substack{(u,v)\in E \\ u\in Y, v\in X}} f(u,v).\end{aligned}$$

Die letzte Gleichheit $(*)$ ergibt sich dabei daraus, dass $f(u,v)$ für die Kanten $(u,v) \in E$ mit $u,v \in Y$ in beiden Summen vorkommt, mit entgegengesetzten Vorzeichen. Die erste Summe ist 0 wegen (2), die zweite Summe enthält nur nichtnegative Summanden. Zusammen ergibt sich ein nicht-positiver Wert. □

Intuitiv kann man aus Lemma 1.5.10 auch noch entnehmen, dass es immer im Prinzip möglich ist, einen Überschuss entlang von RNW-Kanten zurück zur Quelle zu schieben.

Lemma 1.5.11. *Die Anzahl der relabel-Operationen ist nicht größer als* $2n(n-1)$.

Beweis: Knoten q und s sind nie an einer relabel-Operation beteiligt.

Aus Lemma 1.5.10 folgt in Kombination mit der definierenden Eigenschaft von Höhenfunktionen, dass Knoten v mit positivem Überschuss niemals größere Höhe als $2n-1$ haben können. Denn: Sei $v = v_0, v_1, \ldots, v_t = q$ ein einfacher Weg von v nach q im RNW G_f. Dann gilt $h(v_{i-1}) - h(v_i) \leq 1$, für $1 \leq i \leq t$. Summation liefert $h(v) - h(q) = h(v_0) - h(v_t) \leq t \leq n-1$, also $h(v) \leq h(q) + n - 1 = 2n - 1$.

Knoten $v \neq q, s$ startet mit $h(v) = 0$. Mit jeder Operation relabel(v) steigt die Höhe an. Andererseits kann die Höhe von v niemals größer als $2n-1$ werden. Daher kann auf v höchstens $(2n-1)$-mal eine relabel-Operation angewendet werden.

Summiert über alle $v \neq q, s$ ergibt sich die Schranke $(2n-1)(n-2) < 2n(n-1)$. □

Lemma 1.5.12. *Die Anzahl der saturierenden push-Operationen ist nicht größer als* $2nm$.

Beweis: Wir betrachten eine beliebige Kante (u,v) mit $(u,v) \in E$ oder $(v,u) \in E$, das heißt, (u,v) ist potenzielle RNW-Kante, und zeigen, dass über die Kante (u,v)

nicht öfter als n-mal eine saturierende push-Operation läuft. Der Beweis ist sehr ähnlich zu dem in der Laufzeitanalyse für den Edmonds-Karp-Algorithmus, und ist eine Übungsaufgabe. □

Nun müssen wir nur noch die Anzahl der nichtsaturierenden push-Operationen abschätzen. Hierfür betrachten wir die Kombination (f, h) aus Präfluss und Höhenfunktion als einen „Zustand" des Systems, und definieren eine Funktion Φ, die jedem solchen Zustand ein „*Potenzial*" $\Phi(f, h) \in \mathbb{N}$ zuordnet. (Man muss dabei den Namen „Potenzial" nicht verstehen – es handelt sich hier nur um eine Sprechweise.) Der Wert $\Phi(f, h)$ ist anfangs 0. Er steigt mit relabel-Operationen (höchstens $2n(n-1)$ viele) und mit saturierenden push-Operationen (höchstens $2nm$ viele) an und sinkt mit jeder nichtsaturierenden push-Operation um mindestens 1. Daraus folgt, dass es nur endlich viele nichtsaturierende push-Operationen geben kann. Um dies genauer zu sehen, definieren wir eine Funktion Φ wie folgt:

$$\Phi(f, h) := \sum_{\substack{v \in V \\ v \text{ ist aktiv}}} h(v).$$

Es ist klar, dass Φ nur Werte in $\mathbb{N}$ annimmt.

Lemma 1.5.13. (a) *Eine relabel-Operation erhöht $\Phi(f, h)$ um höchstens 1.*

(b) *Eine saturierende push-Operation erhöht $\Phi(f, h)$ um höchstens $2n - 2$.*

(c) *Eine nichtsaturierende push-Operation verringert $\Phi(f, h)$ um mindestens 1.*

(d) *Es gibt höchstens $4n^2m - 2nm$ nicht-saturierende push-Operationen.*

Beweis: (a) relabel(u) vergrößert die Höhe des aktiven Knotens u um 1.

(b) Durch die saturierende push-Operation push(u, v) kann Knoten v vom Zustand „inaktiv" in Zustand „aktiv" wechseln. Seine Höhe ist maximal $2n - 2$ (denn $h(u) \leq 2n - 1$ und $h(u) = h(v) + 1$), und um diesen Wert kann $\Phi(f, h)$ ansteigen.

(c) Wenn push(u, v) nichtsaturierend ist, ist nachher u inaktiv geworden und es könnte sein, dass v vorher inaktiv war und nun aktiv ist. $\Phi(f, h)$ sinkt also um mindestens $h(u) - h(v) = 1$.

(d) Der Anfangswert von $\Phi(f, h)$ nach der Initialisierung ist 0. Nach Lemma 1.5.11 kann $\Phi(f, h)$ durch alle relabel-Operationen zusammen um höchstens $2n(n-1)$ ansteigen. Nach Lemma 1.5.12 in Kombination mit (b) kann $\Phi(f, h)$ durch alle saturierenden push-Operationen zusammen um nicht mehr als $2nm(2n-2) = 4nm(n-1)$ ansteigen. Die Summe aller Anstiege ist also kleiner als $4nm(n-1) + 2n(n-1) \leq 4nm(n-1) + 2nm \leq 4n^2m - 2nm$. (Wir können annehmen, dass G zumindest schwach zusammenhängend ist, also mindesten $n - 1$ Kanten hat.) □

Es folgt, dass es nicht mehr als $4n^2m$ push-Operationen geben kann. Was bleibt noch zu tun? Zunächst einmal muss man überlegen, wieviel Rechenzeit für die $O(n^2m)$ push- und die $O(n^2)$ relabel-Operationen benötigt wird. Man kann die Dinge recht leicht so arrangieren, dass man in konstanter Zeit eine Kante (u, v) für ein push(u, v) findet (man hat eine Wartemenge mit solchen Kanten (u, v)) bzw. einen Knoten u für ein relabel(u) findet (ebenfalls eine Wartemenge), und dass push(u, v) $O(1)$ Zeit benötigt, relabel(u) Zeit $O(n)$ (durchlaufe die Ausgangskanten von u und prüfe, ob sie für push(u, v) in Frage kommen). Damit ist die Rechenzeit $O(n^2m) + O(n^3) = O(n^2m)$.

Satz 1.5.14. *Die Preflow-Push-Methode kann (auf einfache Weise) so implementiert werden, dass die Rechenzeit durch $O(n^2m)$ beschränkt ist.*

Durch geschickte Wahl der Reihenfolge der push- und relabel-Operationen kann die Rechenzeit noch sehr verbessert werden. Zunächst nimmt man lokale Strukturierungen vor, die es einem ersparen, in jeder Runde den ganzen Graphen anzusehen.

Eine ganz einfache Variante ist die, dass man einen aktiven Knoten u, für den keine push-Operation möglich ist, gleich auf ein Niveau hebt, bei dem dies möglich ist. Dies kürzt das Anheben in Einzelschritten ab.

Algorithmus 1.5.15 (**relabel-1**)**.**

Data: f, h, Knoten u mit $\mathrm{ex}(u) > 0$ und
$\forall v \in V : \mathrm{rest}_f(u, v) > 0 \Rightarrow h(u) \leq h(v)$
Result: Hebe aktiven Knoten u ein Niveau höher als niedrigsten Nachfolger in G_f

1 $h(u) \leftarrow \min\{h(v) \mid (u, v) \in E_f\} + 1$;

Eine weitere Vorgehensweise zur Strukturierung der Operationen ist die, einen einmal ausgesuchten Knoten u mit positivem Überschuss *komplett zu leeren*. Das heißt: Man schiebt Überschuss mit saturierenden push-Operationen zu Nachbarn, so lange dies möglich ist. Wenn u dann immer noch Überschuss hat, wird relabel-1(u) ausgeführt; danach kann wieder mit push-Operationen begonnen werden. Damit angestrebte Laufzeiteffekte realisiert werden können, muss man die Untersuchung der von u ausgehenden Kanten passend organisieren. Dazu benötigt man die *Adjazenzliste* von u, in der in der Form einer linearen Liste alle Kanten aufgeführt sind, die möglicherweise in E_f aus u herausführen, also alle Kanten $(u, v) \in E$ und alle Kanten (u, v) mit $(v, u) \in E$. Sobald relabel(u) ausgeführt wurde, fängt man in der Adjazenzliste von u wieder von vorne an. Wenn Knoten u komplett geleert werden konnte, merkt man sich die aktuelle Stelle in der Adjazenzliste und fährt später (wenn u wieder aktiv geworden ist und danach erneut discharge(u) aufgerufen wird) an derselben Stelle in der Adjazenzliste fort.

Algorithmus 1.5.16 (**Discharge**).

```
   Data: f, h, Knoten u mit ex(u) > 0, Zeiger u.current in die Adjazenzliste von u
   Result: bewirkt push- und relabel-Operationen an u; schließlich ex(u) = 0
 1 while u ist aktiv do
 2 |   v ← u.current;
 3 |   if v = NULL then
 4 |   |   relabel-1(u);
 5 |   |   setze u.current auf den Anfang der Adjazenzliste von u;
 6 |   else
 7 |   |   if rest_f(u, v) > 0 und h(v) + 1 = h(u) then
 8 |   |   |   push(u, v)
 9 |   |   else
10 |   |   |   schiebe u.current um 1 Position weiter
11 |   |   end
12 |   end
13 end
```

Man muss sich vergewissern, dass die Ausführung der relabel-1-Operation in Zeile 4 erlaubt ist, das heißt, dass $h(u) \leq h(v)$ für alle v mit $(u, v) \in E_f$ gilt. Wenn die Situation u.current = NULL auftritt, heißt das, dass vorher die Adjazenzliste von u (eventuell in mehreren Abschnitten) komplett durchlaufen worden ist, ohne dass sich $h(u)$ geändert hat, und dass alle betrachteten ausgehenden Kanten (u, w)

- $h(w) \geq h(u)$ erfüllten (daran ändert sich nichts, da kein relabel(u) stattfand), oder
- $h(w) = h(u) - 1$ erfüllten und entweder von vorneherein nicht in E_f waren, oder
- durch die aktuelle oder eine frühere Discharge-Operation gesättigt worden sind.

Daher kann es keine Kanten (u, w) geben, an denen push ausführbar wäre, und relabel ist erlaubt.

Eine geläufige Vorgehensweise für eine Konkretisierung der Preflow-Push-Methode ist die „Highest-Label-Heuristik“, bei der immer Knoten mit möglichst großer Höhe ausgewählt werden, auf die dann Discharge angewendet wird.

Algorithmus 1.5.17 (**Highest-Label**).

```
Data: Flussnetzwerk G = (V, E, q, s, c)
Result: Maximaler Fluss f für G
Initialisierung(G);
foreach Knoten u ∈ V − {q, s} do
    setze u.current auf den Anfang der Adjazenzliste von u
end
while es gibt aktives u do
    wähle aktives u mit h(u) maximal;
    Discharge(u);
end
```

Eine erste, nicht sehr schwierige, Analyse ergibt eine Rechenzeitschranke von $O(n^3)$.

Wir zeigen hier nur kurz, dass die Anzahl der nichtsaturierenden push-Operationen durch $O(n^3)$ beschränkt werden kann. Dazu überlegen wir Folgendes: Eine nichtsaturierende push-Operation an Knoten u macht diesen auf jeden Fall inaktiv. Wenn (irgendwelche saturierenden push-Operationen und) $n-2$ nichtsaturierende push-Operationen in Folge ausgeführt wurden, ohne jegliche relabel-Operation, dann sind alle Knoten in $V - \{q, s\}$ inaktiv geworden, und der Algorithmus hält an. Das heißt: Damit der Algorithmus weiterläuft, muss nach spätestens $n-3$ nichtsaturierenden push-Operationen eine relabel-Operation folgen. Da die Anzahl der relabel-Operationen nur $O(n^2)$ ist, ist die Anzahl der nichtsaturierenden push-Operationen durch $n \cdot O(n^2) = O(n^3)$ beschränkt.

Eine gewisse technische Schwierigkeit liegt noch im Nachweis, dass es in derselben Zeitschranke möglich ist, immer einen aktiven Knoten mit größtem h-Wert zu finden. Hierfür wird auf die Literatur verwiesen.

Satz 1.5.18. *Die Preflow-Push-Methode mit der Highest-Label-Regel kann so implementiert werden, dass die Rechenzeit durch $O(n^3)$ beschränkt ist.*

Eine zweite, etwas aufwendigere Analyse (siehe die Arbeit von Cheryian und Mehlhorn) führt zu einer Rechenzeitschranke von $O(n^2\sqrt{m})$ für die Highest-Label-Heuristik. Dies ist für Kantenanzahlen $m = o(n^2)$ noch besser als $O(n^3)$.

Literatur:
Jon Kleinberg, Éva Tardos, Algorithm Design, Addison-Wesley 2006, Section 7.4.
J. Cheriyan, K. Mehlhorn, An analysis of the highest-level selection rule in the preflow-push max-flow algorithm, Information Processing Letters 69 (1999): 239–242.

Die folgende Tabelle dient nur zur Orientierung für Interessierte, soll also nicht auswendig gelernt werden ...

Max-Flow Algorithmen nach Veröffentlichungsjahr (Auswahl)
(Quelle: Wikipedia, `http://de.wikipedia.org/wiki/Flüsse_und_Schnitte_in_Netzwerken` und `https://en.wikipedia.org/wiki/Maximum_flow_problem`)

Jahr	Autor(en)	Name, Methode	Laufzeiten
1956	**Ford, Fulkerson**	Ford-Fulkerson, flussvergr. Wege	$\mathcal{O}(m \cdot n \cdot u_{\max})$ *
1969	**Edmonds, Karp**	Edmonds-Karp, kürzeste flussv. Wege	$\mathcal{O}\left(m^2 \cdot n\right)$
1970	**Dinitz**	Algorithmus von Dinitz, Sperrfluss	$\mathcal{O}\left(m \cdot n^2\right)$
1972	**Edmonds/Karp, Dinitz**		$\mathcal{O}\left(m^2 \cdot \log\left(u_{\max}\right)\right)$ *
1973	**Dinitz, Gabow**		$\mathcal{O}\left(m \cdot n \cdot \log\left(u_{\max}\right)\right)$ *
1974	Karzanov	Sperrfluss	$\mathcal{O}\left(n^3\right)$
1978	Malhotra, Pramodh-Kumar, Maheshwari	Sperrfluss, Backward-Forward-Propagation	$\mathcal{O}\left(n^3\right)$
1977	Cherkassky		$\mathcal{O}\left(n^2 \cdot \sqrt{m}\right)$
1980	Galil, Naamad		$\mathcal{O}\left(m \cdot n \cdot (\log n)^2\right)$
1983	Sleator, Tarjan	Dinitz-Algo mit *Dynamic Trees*	$\mathcal{O}\left(m \cdot n \cdot \log n\right)$
1986	**Goldberg, Tarjan**	Goldberg-Tarjan-Algo, **Preflow-Push** generisch	$\mathcal{O}(n^2 \cdot m)$
1986		Preflow-Push, FIFO-Regel oder Highest-Label	$\mathcal{O}(n^3)$
1986	**Goldberg, Tarjan**	Preflow-Push,	$\mathcal{O}\left(m \cdot n \cdot \log(n^2/m)\right)$
1987	Ahuja, Orlin		$\mathcal{O}\left(m \cdot n + n^2 \cdot \log\left(u_{\max}\right)\right)$ *
1990	Cheriyan, Hagerup, Mehlhorn	Preflow-Push (randomisiert)	$\mathcal{O}(n^3/\log n)$
1990	Alon	Preflow-Push	$\mathcal{O}\left(m \cdot n + n^{8/3} \cdot (\log n)\right)$
1992	**King, Rao, Tarjan**	Preflow-Push	$\mathcal{O}\left(m \cdot n + n^{2+\varepsilon}\right)$
1994	**King, Rao, Tarjan**	Preflow-Push	$\mathcal{O}\left(m \cdot n \cdot \log_{\frac{m}{n \cdot \log(n)}}(n)\right)$
1997	**Goldberg, Rao**	Sperrfluss	$\mathcal{O}\left(m^{3/2} \log(n^2/m) \log(u_{\max})\right)$
⋮	⋮	⋮	⋮
2012	**Orlin**	Neuer Algo für $m \leq n^{1+\varepsilon}$, KSR für $m \geq n^{1+\varepsilon}$	$\mathcal{O}\left(mn\right)$

$n = |V|$ = „Anzahl der Knoten“, $m = |E|$ = „Anzahl der Kanten“, $u_{\max}$ = „Maximum der Kapazitätsfunktion u“,
nur brauchbar, wenn alle Kapazitäten ganzzahlig sind. Beachte: $u_{\max} \leq C \leq n u_{\max}$.