

# Effiziente Algorithmen (SS2015)

## Kapitel 1 Grundlagen zu Flüssen

Walter Unger

Lehrstuhl für Informatik 1

7:04 Uhr, den 6. November 2018

# Inhalt I

- 1 Einleitung
  - Anwendungen und Motivation
  - Einfache Beispiele
  - Einfache Beispiele (zweiter Versuch)
  - Minimaler Schnitt
  - Beispiele
- 2 Ford-Fulkerson
  - Einleitung
  - Min-Cut Max-Flow

- Laufzeit
- 3 Algorithmus von Dinitz
  - Einleitung
  - Algorithmus und Beispiel
  - Laufzeit
- 4 Dinitz mit Propagation
  - Einleitung
  - Algorithmus und Beispiel
  - Weiteres Beispiel
  - Laufzeit

# Motivation

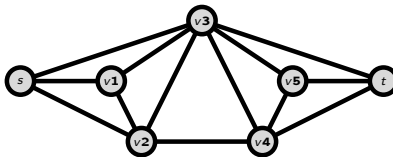
Sehr oft muss die Transportkapazität von Daten/Objekten in einem Netzwerk (gerichteter Graph) bestimmt werden. Dieses wird, wie im Folgenden zu sehen ist, durch ein Flussproblem modelliert. D.h. der Fluss fließt über die Kanten. Er wird von einem speziellen Knoten – der Quelle – gespeist und wird von einem weiteren speziellen Knoten – der Senke – aufgefangen.

Weiterhin werden die Kapazitäten auf den Kanten angegeben. Der Fluss auf einer Kante darf nun nicht größer als diese gegebene Kapazität sein. Sodann müssen die internen Knoten den gesamten eingehenden Fluss weiterleiten.

Bemerkung: Es gibt ohne Einschränkung der Allgemeinheit genau eine Quelle des Flusses und genau eine Senke. Mehrere Quellen (oder Senken) können durch eine Metaquelle (oder Metasenke) simuliert werden. Der Fluss aus der Quelle (bzw. in die Senke) ist nun zu maximieren. Wie dieser maximale Fluss bestimmt wird, wird im Folgenden entwickelt.

# Anwendungen

- Kapazität der Straßen von Aachen nach Köln.
- Kapazität der Datenleitungen von Amerika nach China.
- Kapazität des Deltas des Amazonas.
- Kapazität der Abwasserkanäle von Rom.
- Alle durch Graphen modellierbaren Durchsatzprobleme.
- Hilfreich bei anderen Anwendungen.
- Altes Problem.



# Modellierung

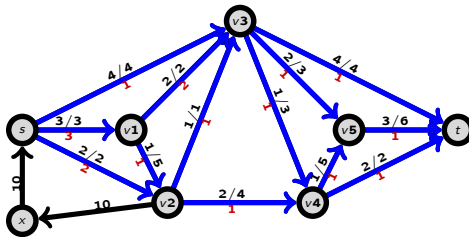
- Das Netzwerk wird als Graph  $G = (V, E)$  modelliert.
- Die Kanten haben eine maximale Kapazität:

$$c : E \mapsto \mathbb{R}^+ \quad \text{oder}$$

$$c : E \mapsto \mathbb{Q}^+ \quad \text{oder}$$

$$c : E \mapsto \mathbb{N}^+$$

- Der Fluss startet an einem Knoten  $s \in V$ .
- Der Fluss endet an einem Knoten  $t \in V$ .



# Das Flussproblem

## Definition (Flussproblem)

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

- $(V, E)$  ist ein gerichteter Graph ( $n = |V|, m = |E|$ )
- $s, t \in V$  und  $s \neq t$
- $c : E \mapsto \mathbb{N}^+$

Ausgabe:  $f : E \mapsto \mathbb{R}_0^+$  mit:

- $\forall e : 0 \leq f(e) \leq c(e)$
- $\forall v \in V \setminus \{s, t\} : \sum_{(a,v) \in E} f((a,v)) = \sum_{(v,a) \in E} f((v,a))$

Ziel: Maximiere  $w(f) = \sum_{(s,v) \in E} f((s,v))$ .

- $s$  heißt Quelle und  $t$  heißt Senke.
- $c$  ist die Kapazitätsfunktion.
- $f$  nennt man die Flussfunktion.

# Bemerkungen zum Flussproblem

$$n = |V|, m = |E|$$

- O.B.d.A.: Quelle  $s$  hat Eingangsgrad 0.
- O.B.d.A.: Senke  $t$  hat Ausgangsgrad 0.
- $f(v, w) = f(e)$  falls  $e \in E$  und  $e = (v, w)$ .
- Kapazitätsbeschränkung:

$$\forall e : f(e) \leq c(e)$$

- Schreibweisen:

$$\begin{aligned} f_{in}(v) &= \sum_{(a,v) \in E} f((a,v)) \\ f_{out}(v) &= \sum_{(v,a) \in E} f((v,a)) \end{aligned}$$

- Flusserhaltung:

$$\forall v \in V \setminus \{s, t\} : f_{in}(v) = f_{out}(v).$$

- Wert des Flusses  $f$  ist:  $w(f) = f_{out}(s) = f_{in}(t)$ .
- Eine Kapazitätsfunktion  $c : E \mapsto \mathbb{Q}^+$  kann durch  $c : E \mapsto \mathbb{N}^+$  modelliert werden.
- Eine Kapazitätsfunktion  $c : E \mapsto \mathbb{R}^+$  führt zu "Problemen".

# Überblick

Mit den folgenden Beispielen werden die Ideen der Algorithmen schrittweise entwickelt:

- Der erste Ansatz ist, den Fluss von der Quelle aus durch den Graphen zu drücken. (siehe "Versuch am einfachen Beispiel")
- Danach sieht man: Falls es mehrere Möglichkeiten gibt, dann können diese dazu führen, dass die Entscheidungen zu komplex werden können. (siehe "Versuch am aufwendigeren Beispiel")
- Das oben erwähnte Beispiel zeigt, dass man strukturiert nach dem Fluss suchen muss. Damit ergibt sich die Idee, einfache Flüsse zu suchen und dann schrittweise diese zu einem Gesamtfluss zu vergrößern.
- In den verbleibenden Beispielen wird daher als vergrößernder Fluss ein Weg gesucht.

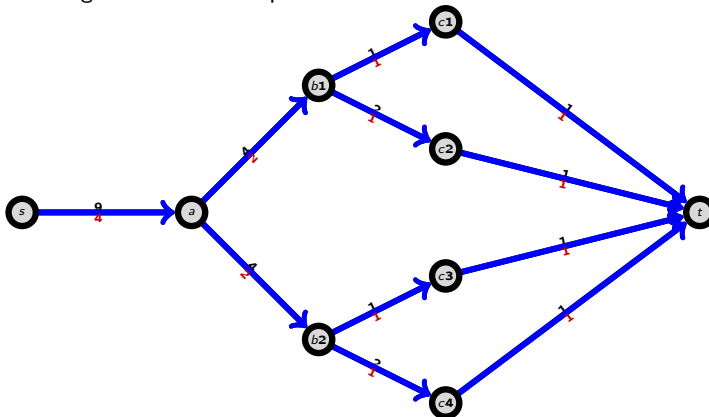


# Versuch am einfachen Beispiel

$$n = |V|, m = |E|$$

Idee: verteile möglichst großen Fluss von der Quelle aus.

Geht relativ gut für dieses Beispiel

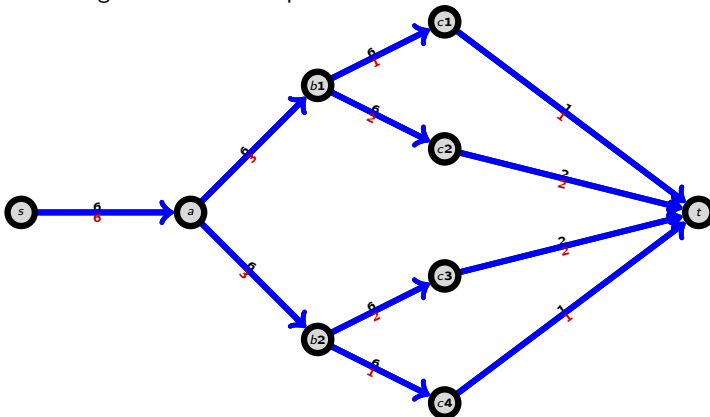


# Versuch am aufwendigeren Beispiel

$$n = |V|, m = |E|$$

Idee: verteile möglichst großen Fluss von der Quelle aus.

Geht nicht so gut für dieses Beispiel



# Situation

$$n = |V|, m = |E|$$

- Ein “direktes” Bestimmen des Flusses von der Quelle aus scheitert.
- Es muss eine einfachere Struktur für einen Fluss gesucht werden.
- Die Lösung ergibt sich dann aus mehreren dieser einfachen Strukturen.
- Idee: Wähle als einfache Struktur den Pfad (Weg).
- Algorithmus:
  - ① Starte mit dem leeren Fluss.
  - ② Bestimme einen Pfad  $P$ , auf dem der Fluss von  $s$  nach  $t$  vergrößerbar ist.
  - ③ Bestimme den maximalen vergrößernden Fluss  $p$  auf  $P$ .
  - ④ Erweitere die Lösung um  $p$  auf dem Pfad  $P$ .
  - ⑤ Wiederhole so oft, wie es einen passenden Pfad  $P$  gibt.

## Neue Situation

Das Verteilen eines möglichst grossen Startflusses führt zu komplexen algorithmischen Problemen, falls der Fluss blockiert wird. Dies war in den Beispielen gut zu sehen. Damit ergibt sich die Idee, dafür zu sorgen, daß eine solche Blockade nicht geschieht. Daher wird im Folgenden nur nach Wegen, die noch Fluss führen können, gesucht.

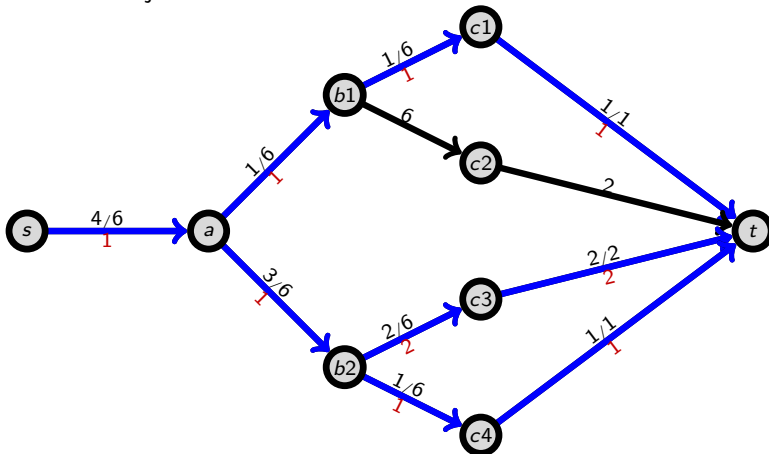
Das Verteilen eines möglichst grossen Flusses wird aber nicht vollständig ignoriert. Am Ende dieses Kapitels wird genau das ausgenutzt, um die Laufzeit des Algorithmus von Dinitz zu verbessern. Dabei wird der schwächste Knoten eines Teilnetzwerks (siehe Niveaunetzwerk) gesucht und der Fluss, der über ihn gehen kann durch das Netzwerke gedrückt. In dem Teilnetzwerk sind die Wege eindeutig und damit kann der Fluss des schwächsten Knoten nicht blockiert werden.

Hier betrachten wir aber erst mal nur Flusserweiterungen durch Pfade. Der aktuelle Fluss wird dann mit dem möglichen Fluss über den Pfad erweitert.

## Zweiter Versuch am einfachen Beispiel

$$n = |V|, m = |E|$$

Idee: finde in jedem Schritt einen erweiternden Pfad von  $s$  nach  $t$ .

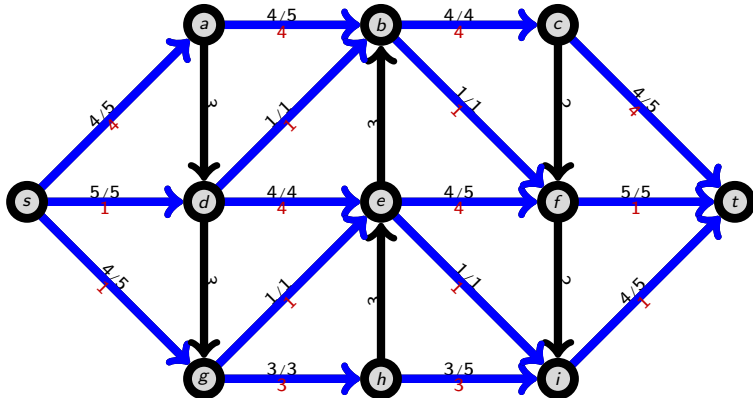


# Kleines Beispiel

$$n = |V|, m = |E|$$

Idee: finde in jedem Schritt einen flussvergrößernden Pfad von  $s$  nach  $t$ .

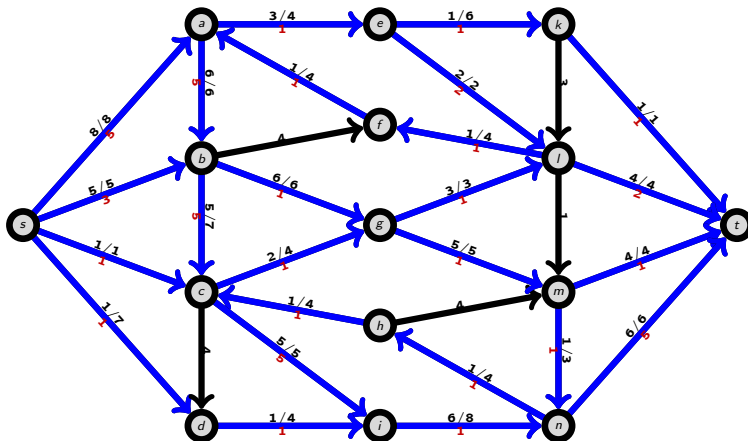
Geht hier auch gut.



## schönes Beispiel

$$n = |V|, m = |E|$$

Idee: finde in jedem Schritt einen erweiternden Pfad von  $s$  nach  $t$ .



# Situation

$$n = |V|, m = |E|$$

- Algorithmus findet in den Beispielen einen maximalen Fluss.
- Algorithmus “findet” eine Barriere, die die Optimalität anzeigt.
- Ziel: Nutze diese Barriere, um Optimalität zu zeigen.
- Definiere diese Barriere (Cut oder Schnitt).
- Und zeige dann: Größe des Schnittes und maximaler Fluss sind gleich.



## Idee zum minimalen Schnitt

Wir haben nun einen Ansatz, um einen maximalen Fluss zu bestimmen. Die Idee ist, suche Flusswege, die einen bisher bestimmten Fluss erweitern. Falls ein solcher Weg nicht mehr existiert, so hoffen wir, daß der nun akkumulierte Fluss eine maximaler ist.

Um dies zu beweisen, wird der Schnitt eines Graphen definiert. Ein Schnitt besteht informell aus den Kanten, die von einer Knotenmenge, die die Quelle aber nicht die Senke enthält, herauszeigt. Die Kapazität dieser Kanten eines jeden Schnitts beschränkt den Fluss. Daher ist der minimale Schnitt eine obere Schranke für den maximalen Fluss.

Wir hoffen nun, daß unser bisheriges Verfahren einen Fluss bestimmt, der vom Wert her, den Wert eines minimalen Schnitts erreicht.

Bemerkung: Bei diesem Versuch werden wir sehen, es ist noch eine Anpassung an den Algorithmus notwendig, damit diese Behauptung gilt. Dazu wird im Folgenden das Restnetzwerk eingeführt. Dieses Restnetzwerk wird dem Algorithmus erlauben, Fehlentscheidungen zurückzunehmen.

# Minimaler Cut

$$n = |V|, m = |E|$$

## Definition (Cut und Fluss über Cut)

Sei  $G = (V, E, s, t, c)$  gegeben. Ein Cut (Schnitt) ist ein Paar  $(S, T)$  mit:

- $V = S \dot{\cup} T$  und  $s \in S$  und  $t \in T$ .
- Die Kapazität des Schnittes ist:

$$c(S, T) = \sum_{(v,w) \in E, v \in S, w \in T} c(v, w).$$

- Für eine Flussfunktion  $f$  ist der Fluss über den Schnitt  $(S, T)$ :

$$f(S, T) = \sum_{(v,w) \in E, v \in S, w \in T} f(v, w) - \sum_{(w,v) \in E, v \in S, w \in T} f(w, v).$$

- Zeige nun: Maximaler Fluss  $\leq$  Minimaler Cut.
- Zeige weiter: Maximaler Fluss  $\geq$  Minimaler Cut.

# Fluss und Schnitt

$$n = |V|, m = |E|$$

Lemma (Max-Flow  $\leq$  Min-Cut)

Für jeden Fluss  $f$  und jeden Schnitt  $(S, T)$  gilt:  $w(f) = f(S, T) \leq c(S, T)$ .

Beweis: Zeige:  $w(f) = f(S, T)$ :

- $f(S, T) = \sum_{(v,w) \in E, v \in S, w \in T} f(v, w) - \sum_{(w,v) \in E, v \in S, w \in T} f(w, v)$
- $\sum_{(v,w) \in E, v \in S, w \in S} f(v, w) - \sum_{(v,w) \in E, v \in S, w \in S} f(v, w) = 0$
- $\sum_{(w,v) \in E, v \in S, w \in S} f(w, v) - \sum_{(w,v) \in E, v \in S, w \in S} f(w, v) = 0$
- Zusammengefasst ergibt sich:

$$\begin{aligned} f(S, T) &= \sum_{(v,w) \in E, v \in S, w \in T} f(v, w) - \sum_{(w,v) \in E, v \in S, w \in T} f(w, v) \\ &\quad + \sum_{(v,w) \in E, v \in S, w \in S} f(v, w) - \sum_{(w,v) \in E, v \in S, w \in S} f(w, v) \\ &\quad + \sum_{(w,v) \in E, v \in S, w \in S} f(w, v) - \sum_{(v,w) \in E, v \in S, w \in S} f(v, w) \end{aligned}$$

- Weiter ergibt sich:

$$\begin{aligned} f(S, T) &= \sum_{(v,w) \in E, v \in S, w \in V} f(v, w) - \sum_{(w,v) \in E, v \in S, w \in V} f(w, v) \\ &= w(f) \text{ wegen Flusserhaltung} \end{aligned}$$

# Situation

$$n = |V|, m = |E|$$

- Es gilt:  $w(f) = f(S, T) \leq c(S, T)$

$$\begin{aligned} \sum_{(v,w) \in E, v \in S, w \in T} f(v, w) - \sum_{(w,v) \in E, v \in S, w \in T} f(w, v) \\ \leq \sum_{(v,w) \in E, v \in S, w \in T} c(v, w) \end{aligned}$$

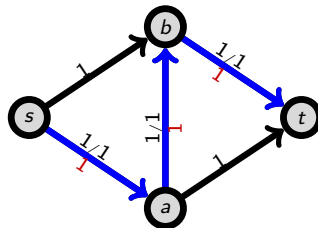
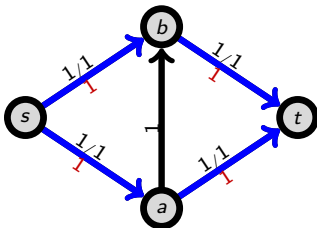
- Wir hoffen auf Gleichheit, d.h. hoffen auf:

$$\sum_{(w,v) \in E, v \in S, w \in T} f(w, v) = 0$$

- D.h. die "Rückwärtskanten" sollen keinen Fluss führen.
- War das in den bisherigen Beispielen schon mal der Fall?
- Betrachte dazu das folgende Beispiel.

## Beispiele

$$n = |V|, m = |E|$$

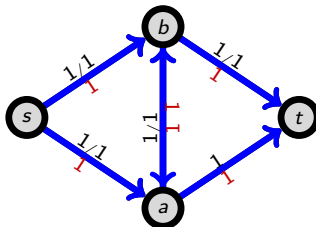


Da ist nicht immer ein Cut zu sehen!

# Neue Situation

$$n = |V|, m = |E|$$

- Beim bisherigen Verfahren können Rückwärtskanten mit einem nichtleeren Fluss auftreten.
- Der bisherige Algorithmus liefert nicht immer den maximalen Fluss.
- Fehlentscheidungen müssen behoben werden.
- Der Fluss über die Rückwärtskante muss zurückgenommen werden!



# Restnetzwerk

$$n = |V|, m = |E|$$

Wir nehmen o.B.d.A bei einem Netzwerk  $G = (V, E, s, t, c)$  an:

- $\forall e = (v, w) \in E : f(v, w) \cdot f(w, v) = 0.$
- Alternativ o.E.d.A:  $(v, w) \in E \implies (w, v) \notin E.$

## Definition (Restnetzwerk)

Das Restnetzwerk  $G_f$  zu einem Netzwerk  $G = (V, E, s, t, c)$  ist:

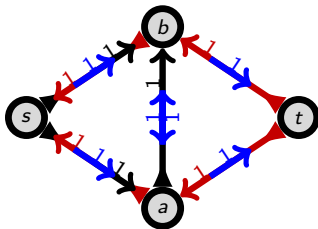
- Für alle Paare  $(v, w) \in V^2$ :

$$\text{rest}_f(v, w) = \begin{cases} c(v, w) - f(v, w) & (v, w) \in E \\ f(w, v) & (w, v) \in E \\ 0 & \text{sonst} \end{cases}$$

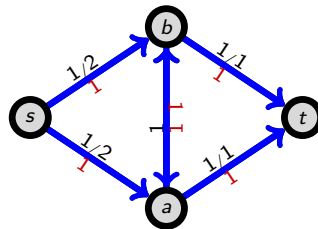
- $G_f = (V, E_f)$  mit  $E_f = \{(v, w) \in V^2 \mid \text{rest}_f(v, w) > 0\}$

# Einfaches Beispiel für Restnetzwerk

$$n = |V|, m = |E|$$



Das Restnetzwerk



Das Netzwerk



# Ford-Fulkerson-Methode (1957)

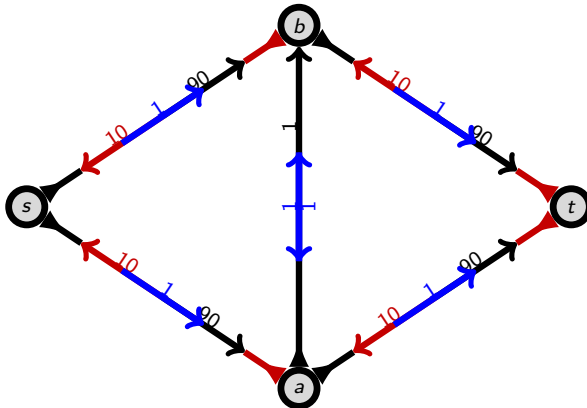
$$n = |V|, m = |E|$$

- ① Gegeben ist Netzwerk:  $G = (V, E, s, t, c)$
- ②  $\forall e \in E : f(e) = 0.$
- ③ Wiederhole:
  - ① Bestimme Restnetzwerk  $G_f$
  - ② Bestimme flussvergrößernden Weg  $P$  in  $G_f$ .
  - ③ Falls es  $P$  nicht gibt, gebe  $f$  aus und breche ab.
  - ④ Bestimme maximale Flussvergrößerung  $p$  auf  $P$ .
  - ⑤ Vergrößere  $f$  auf dem Pfad  $P$  um  $p$ .

Laufzeit:  $O(C \cdot m)$  mit  $C = \sum_{e \in E} c(e)$ .

# Beispiel

$$n = |V|, m = |E|$$

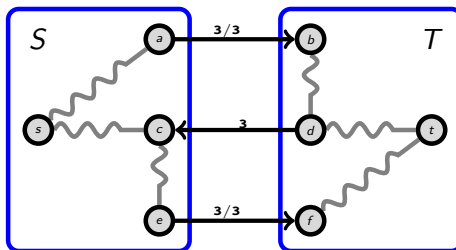


Ein Algorithmus würde nicht die Gedult verlieren.

# Rückblick

$$n = |V|, m = |E|$$

- Sei  $(S, T)$  der minimale Schnitt.
- Der Algorithmus kann Kanten von  $T$  nach  $S$  verwenden.
- Damit ist der aktuelle Fluss über den Schnitt kleiner als die Kapazität des minimalen Schnittes.
- Durch Rückwärtskanten werden diese "Fehler" behoben.
- Damit findet der Algorithmus den minimalen Schnitt.
- Und damit kann das Min-Cut Max-Flow Theorem bewiesen werden.



# Idee zum Minimum Cut Maximum Flow Beweis

Beim Beweis werden drei Aussagen per Ringschluss als äquivalent gezeigt.

- Die erste Aussage ist: der Fluss  $f$  ist ein maximaler Fluss. Damit ergibt sich sofort als Folgerung die folgende Aussage:
- Die zweite Aussage ist: Es gibt keinen vergrößernden Weg im Restnetzwerk. Damit gibt es auch eine Menge  $S$  von Knoten, die im Restnetzwerk noch von  $s$  aus erreichbar sind. Diese Menge  $S$  ist dann ein Schnitt, da  $t$  nicht in  $S$  sein kann. Da alle Kanten von  $S$  nach  $V \setminus S$  nicht nutzbar sind, sind die Vorwärtskanten voll gefüllt und die Rückwärtskanten leer. Damit ist der Wert des Schnitts gleich dem Wert von dem Fluss. Und es folgt damit die dritte Aussage:
- Die dritte Aussage ist: Es gibt einen Schnitt und einen Fluss mit gleichem Wert. Aus dieser Aussage folgt auch direkt die erste Aussage.

## Min-Cut Max-Flow

$$n = |V|, m = |E|$$

## Theorem (Minimum Cut Maximum Flow)

Sei  $f$  ein Fluss auf einem Netzwerk  $G = (V, E, s, t, c)$ .

Dann sind die folgenden Aussagen äquivalent:

- $f$  ist ein maximaler Fluss.
- Das Restnetzwerk  $G_f$  enthält keinen vergrößernden Weg von  $s$  nach  $t$ .
- Es gibt einen Schnitt  $(S, T)$  mit:  $w(f) = c(S, T)$ .

Beweis:

- Zeige  $c) \implies a)$ .
- Zeige  $a) \implies b)$ .
- Zeige  $b) \implies c)$ .

Zeige c)  $\implies$  a)

$$n = |V|, m = |E|$$

c) Es gibt einen Schnitt  $(S, T)$  mit:  $w(f) = c(S, T)$ .

a)  $f$  ist ein maximaler Fluss.

- Wegen Lemma (Max-Flow  $\leq$  Min-Cut) gilt:
- $w(f) = c(S, T) \geq \text{Min-Cut} \geq \text{Max-Flow}$
- Damit muss  $f$  ein maximaler Fluss sein.

Zeige a)  $\implies$  b)

$$n = |V|, m = |E|$$

a)  $f$  ist ein maximaler Fluss.

b) Das Restnetzwerk  $G_f$  enthält keinen vergrößernden Weg von  $s$  nach  $t$ .

- Beweis durch Widerspruch.
- Sei  $f$  maximaler Fluss.
- Sei  $P$  ein vergrößernder Weg von  $s$  nach  $t$  in  $G_f$  mit Fluss  $p$ .
- Dann kann  $f$  um  $p$  vergrößert werden.
- $f$  ist daher nicht maximal.
- Widerspruch.

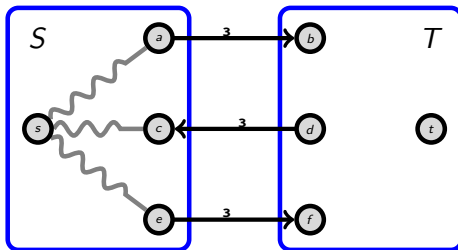
Zeige b)  $\implies$  c)

$$n = |V|, m = |E|$$

b) Das Restnetzwerk  $G_f$  enthält keinen vergrößernden Weg von  $s$  nach  $t$ .

c) Es gibt einen Schnitt  $(S, T)$  mit:  $w(f) = c(S, T)$ .

- Setze  $S = \{v \in V \mid \exists \text{ Weg von } s \text{ nach } v \text{ in } G_f\}$
- Setze  $T = V \setminus S$ .
- Da es keinen vergrößernden Weg in  $G_f$  gibt, gilt:  $t \in T$ .





Zeige b)  $\implies$  c)

$$n = |V|, m = |E|$$

b) Das Restnetzwerk  $G_f$  enthält keinen vergrößernden Weg von  $s$  nach  $t$ .

c) Es gibt einen Schnitt  $(S, T)$  mit:  $w(f) = c(S, T)$ .

- Setze  $S = \{v \in V \mid \exists \text{ Weg von } s \text{ nach } v \text{ in } G_f\}$
- Setze  $T = V \setminus S$ .
- Da es keinen vergrößernden Weg in  $G_f$  gibt, gilt:  $t \in T$ .
- $\forall (v, w) \in E \cap (S, T) : f(v, w) = c(v, w)$ , denn  $\text{rest}_f(v, w) = 0$ .
- $\forall (v, w) \in E \cap (T, S) : f(v, w) = 0$ , denn  $\text{rest}_f(v, w) = 0$ .
- Nach Definition von  $c(S, T)$  und  $f(S, T)$  gilt:

$$\begin{aligned} c(S, T) &= \sum_{(v,w) \in E, v \in S, w \in T} c(v, w) \\ &= \sum_{(v,w) \in E, v \in S, w \in T} f(v, w) - \sum_{(w,v) \in E, v \in S, w \in T} f(w, v) \\ &= f(S, T) \\ &= w(f) \end{aligned}$$

# Idee zur Laufzeit

Nehmen wir erst mal an, es würden nie Rückwärtskanten genutzt werden. Dann ist die Abschätzung der Laufzeit einfach. Bei jedem Aufruf von BFS wird mindestens eine Kante ausgefüllt. Da es maximal  $m = |E|$  viele Kanten gibt und die Laufzeit von BFS  $O(m)$  ist, ergibt sich eine Laufzeit von  $O(m^2)$ . Es muss nur noch ein Faktor von  $O(n)$  erklärt werden, der sich aus dem Nutzen von Rückwärtskanten ergibt.

Solange eine Kante ihre Position im BFS-Baum nicht verändert, ist die Rückwärtskante nicht nutzbar. Wenn nun die Rückwärtskante genutzt wird, so hat sich der Abstand von der Quelle vergrößert. Um dies zu zeigen, werden im folgenden Beweis induktiv die neuen Kanten einzeln in den Graphen eingefügt. Durch Beweis der Invariante – kein Knoten verringert den Abstand zur Quelle – im induktiven Beweis ergibt sich die Behauptung.

Damit kann eine Kante höchstens Tiefe des BFS-Baums oft ihre Nutzungsrolle ändern. Damit ergibt sich die Behauptung, denn die Tiefe des BFS-Baums ist durch  $n = |V|$  beschränkt.

# Ford-Fulkerson mit BFS

$$n = |V|, m = |E|$$

Theorem (Edmonds und Karp, 1969)

*Die Laufzeit der Ford-Fulkerson Methode mit BFS ist  $O(m^2 \cdot n) = O(n^5)$ .*

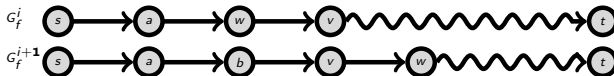
Beweis:

- Zeige: es reichen  $O(m \cdot n)$  Iterationen.
- Durch einen flussvergrößernden Weg werden Kanten gelöscht und ggf. hinzugefügt.
- Sei nun  $G_f^i$  das Restnetzwerk  $G_f$ , welches in Iteration  $i$  verwendet wird.
- Zeige:  $\forall v \in V : \text{dist}_{G_f^i}(s, v) \leq \text{dist}_{G_f^{i+1}}(s, v)$ .
- Dies gilt sicher für Kanten, die in  $G_f^i$  gelöscht wurden.
- Die hinzugefügten Kanten von  $G_f^i$  zu  $G_f^{i+1}$  werden auf der folgenden Folie behandelt.

$\forall v \in V : \text{dist}_{G_f^i}(s, v) \leq \text{dist}_{G_f^{i+1}}(s, v)$  (neue Kanten)

$n = |V|, m = |E|$

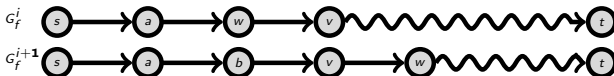
- Wir fügen kantenweise die neuen Kanten zu  $G_f^{i+1}$  hinzu.
- D.h. zeigen induktiv (über die Anzahl der eingefügten Kanten):  
 $\forall x \in V : \text{dist}_{G_f^i}(s, x) \leq \text{dist}_{G_f^{i+1}}(s, x)$ .
- Sei  $(v, w)$  eine Kante mit  $(v, w) \in E(G_f^{i+1})$  und  $(v, w) \notin E(G_f^i)$ .
- Kante  $(v, w)$  wird eingefügt, wenn sie vorher als Kante  $(w, v)$  auf flussvergrößerndem Weg in  $G_f^i$  lag.
- Für den Knoten  $v$  gilt damit:  $\text{dist}_{G_f^i}(s, v) \leq \text{dist}_{G_f^{i+1}}(s, v)$ , denn nur für  $w$  kann die Distanz verbessert werden.
- Damit gilt:  $\text{dist}_{G_f^i}(s, v) + 1 = \text{dist}_{G_f^{i+1}}(s, w)$ .
- Es kann sich also der Abstand von  $w$  nicht verbessern,
- es gilt sogar:  $\text{dist}_{G_f^i}(s, w) + 2 \leq \text{dist}_{G_f^{i+1}}(s, w)$ .



# Beweis der Laufzeit

$$n = |V|, m = |E|$$

- Es gilt:  $\forall v \in V : \text{dist}_{G_f^i}(s, v) \leq \text{dist}_{G_f^{i+1}}(s, v)$ .
- Falls  $(v, w)$  gelöscht wird, so gilt:  $\text{dist}_{G_f^i}(s, v) + 1 \leq \text{dist}_{G_f^{i+1}}(s, w)$ .  
Wir sprechen hier von einer Flaschenhalskante.
- Falls  $(v, w)$  wieder eingefügt wird, so gilt:  $\text{dist}_{G_f^i}(s, w) + 2 \leq \text{dist}_{G_f^{i+1}}(s, w)$ .  
Beachte:  $(v, w)$  lag auf einem kürzesten Weg in  $G_f^i$ .
- Wegen  $\text{dist}_{G_f^i}(s, t) \leq n - 1$  gilt:  
Jede Kante  $(v, w)$  kann höchstens  $\lfloor (n - 1)/2 \rfloor$  mal gelöscht werden.
- In jeder Iteration wird mindestens eine Kante entfernt.
- Im Restnetzwerk sind höchstens  $2 \cdot m$  Kanten
- Anzahl der Iterationen damit höchstens:  $\frac{n}{2} \cdot 2 \cdot m = n \cdot m$ .



# Ideen zur Laufzeitverbesserung

Um die Laufzeit zu verbessern, wird im Folgenden versucht, immer möglichst viel Fluss pro Schleifendurchlauf zu berechnen. Dieses geschieht in zwei Schritten.

- Im ersten wird die Idee von BFS bei Ford-Fulkerson genutzt. Es wird das Schichtennetzwerk (Niveaunetzwerk) mit BFS bestimmt, d.h. alle nicht genutzten Kanten werden ignoriert. Es gibt nur Kanten von einer Schicht (Knoten mit gleichem Abstand zur Quelle) zu einer anderen benachbarten Schicht. Statt nur einen Weg wie bei Ford-Fulkerson zu bestimmen, wird aber nun das Schichtennetzwerk voll ausgefüllt. Man beachte: Das Schichtennetzwerk ergibt sich aus dem Restnetzwerk. Es muss also zum Ausfüllen des Schichtennetzwerks keine weiteren Anpassungen gemacht werden.
- Im zweiten Schritt wird im Schichtennetzwerk der schwächste Knoten des Schichtennetzwerks gesucht und der Fluss, der über ihn gehen kann, durch das Netzwerk gedrückt. Im Schichtennetzwerk sind die Wege eindeutig und damit kann der Fluss des schwächsten Knoten nicht blockiert werden.

# Idee

- Versuche die Laufzeit zu verbessern.
- Erste Verbesserung war: suche kürzesten verbessernden Weg.
- Hier nun: suche viele kürzeste verbessernden Wege.
- D.h. versuche mehr als nur einen Weg pro Iteration zu finden.
- Verwende ein Niveaunetzwerk.
- In einem Niveaunetzwerk gibt es nur Kanten zwischen benachbarten Schichten.

# Niveaunetzwerk

## Definition (Niveaunetzwerk)

Sei  $G_f = (V, E_f)$  ein Restnetzwerk. Setze nun:

- Für  $i \in \mathbb{N}_0$  sei:  $V_i = \{v \in V \mid \text{dist}_{G_f}(s, v) = i\}$ .
- $E'_f = \{(v, w) \in E_f \mid \exists i : v \in V_i \wedge w \in V_{i+1}\}$ .
- $c' : E'_f \mapsto \mathbb{N}$  mit:  $c'(e) = c(e)$
- Das Niveaunetzwerk ist dann:  $G'_f = (V, E'_f, s, t, c')$ .

Bemerkung  $G'_f$  kann mittels Breitensuche in Zeit  $O(m)$  bestimmt werden.



# Sperrfluss

## Definition (Sperrfluss)

Sei  $G'_f = (V, E'_f, s, t, c')$  ein Niveaunetzwerk und  $f'$  ein Fluss auf  $G'$ .

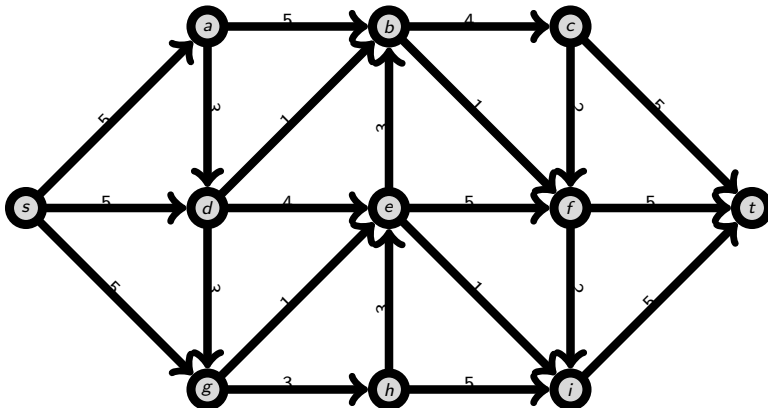
- Eine Kante  $e \in E'_f$  heißt saturiert, falls  $f'(e) = \text{rest}_f(e)$ .
- $f'$  heißt Sperrfluss, falls jeder Weg von  $s$  nach  $t$  in  $G'_f$  eine saturierte Kante hat.
- Ein Knoten  $v$  heißt saturiert, falls  $\text{pot}(v) = 0$ , mit:
  - $\text{pot}(v) = \min\{\sum_{e \in N_{in}(v)} \text{rest}_f(e), \sum_{e \in N_{out}(v)} \text{rest}_f(e)\}$ .

D.h. ein maximaler Fluss in  $G'$  ist ein Sperrfluss.

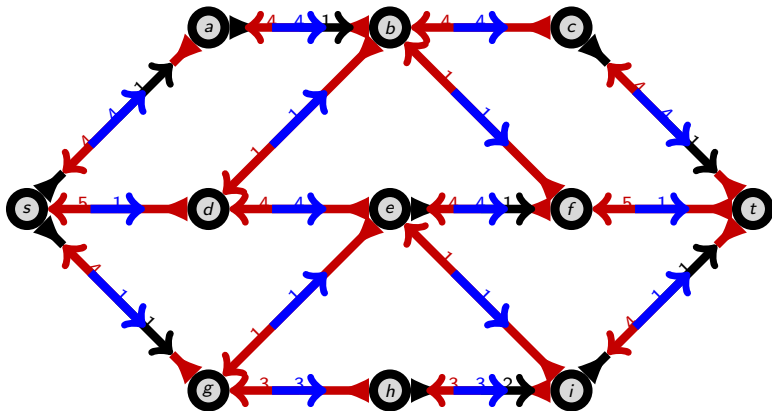
# Dinitz Algorithmus (1970)

- ① Gegeben ist Netzwerk:  $G = (V, E, s, t, c)$
- ②  $\forall e \in E : f(e) = 0.$
- ③ Wiederhole:
  - ① Bestimme Restnetzwerk  $G_f$
  - ② Bestimme Niveaunetzwerk  $G'_f$
  - ③ Bestimme Sperrfluss  $f'$  in  $G'_f$ .
  - ④ Falls  $w(f') = 0$ , gebe  $f$  aus und breche ab.
  - ⑤ Erweitere  $f$  um  $f'$ .
- Beachte, dass beim Erweitern von  $f$  um  $f'$  die Flüsse auf entgegengesetzte Kanten subtrahiert werden.
- Beachte: die Korrektheit ergibt sich analog zur Ford-Fulkerson Methode.

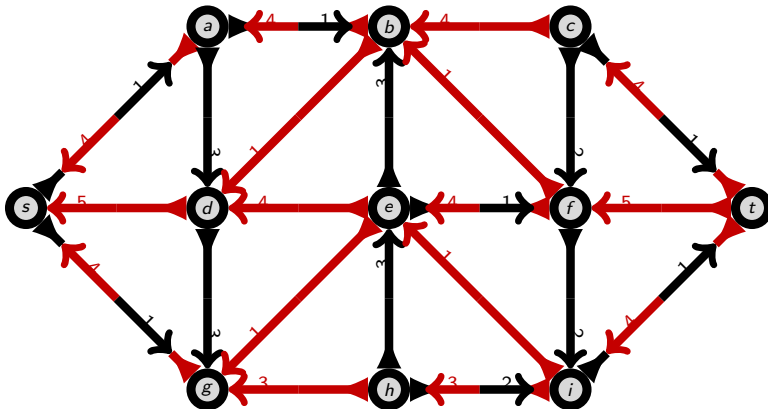
## kleines Beispiel (Dinitz)



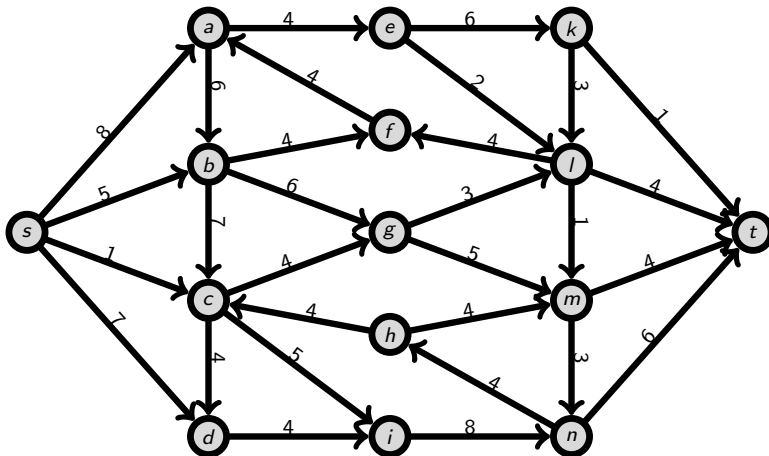
# kleines Beispiel (Niveaunetzwerk)



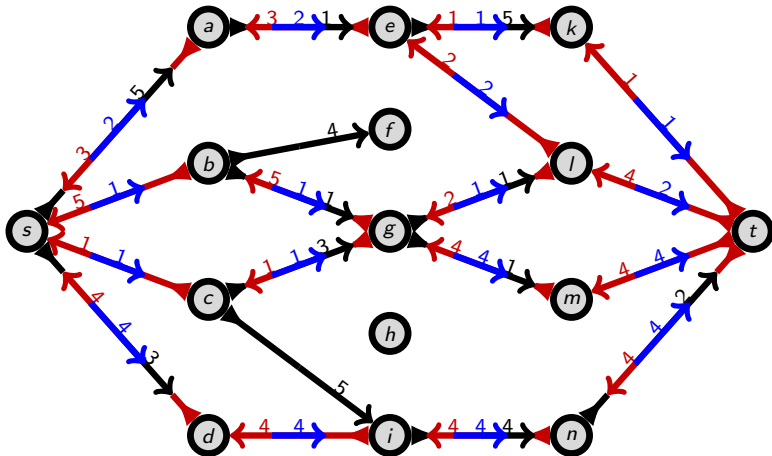
# kleines Beispiel (Schritt 2)



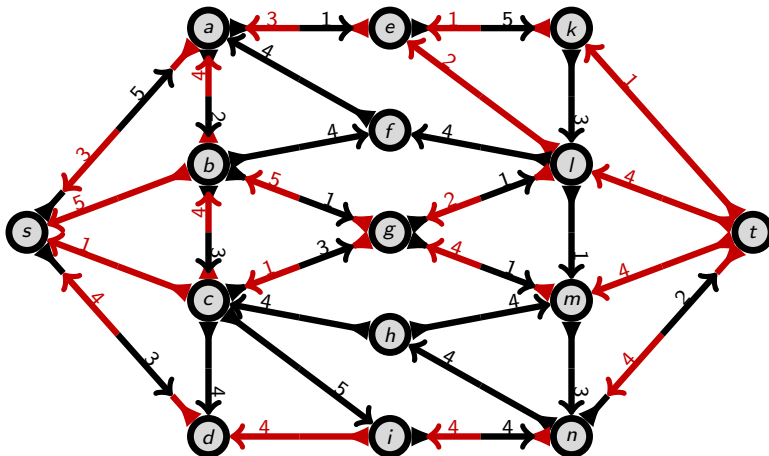
## schönes Beispiel (Dinitz)



# schönes Beispiel (Niveaunetzwerk 1)

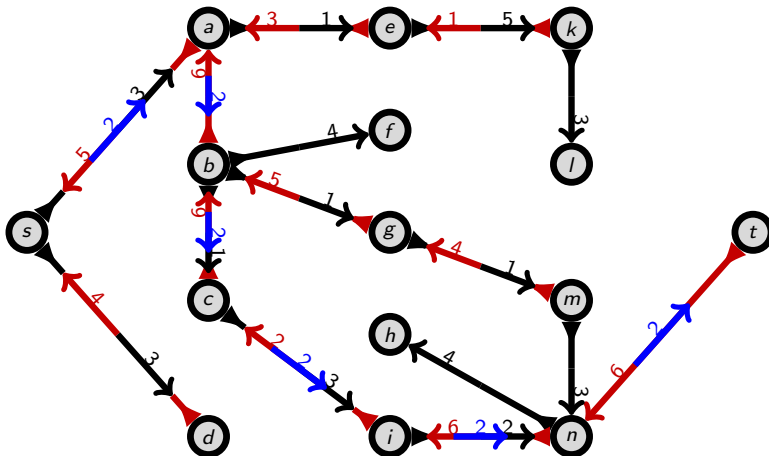


## schönes Beispiel (Schritt 2)

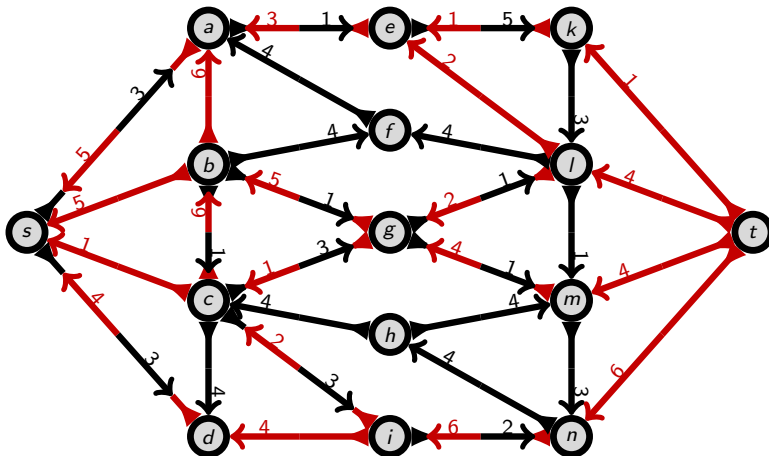




## schönes Beispiel (Niveaunetzwerk 2)



## schönes Beispiel (Schritt 3)



# Anzahl Iterationen (Dinitz)

## Lemma

*Die Anzahl der Iterationen beim Algorithmus von Dinitz ist maximal  $n - 1$ .*

Beweis:

- Sei  $l = \text{dist}_{G'_f}(s, t)$  Abstand von  $s$  und  $t$ .
- Alle Wege der Länge  $l$  werden in  $G'_f$  getrennt.
- Die neu entstehenden Kanten führten aber vorher von Schicht  $i$  zur Schicht  $i - 1$ .
- Die Wege über neue Kanten haben damit eine Länge von mindestens  $l + 2$ .

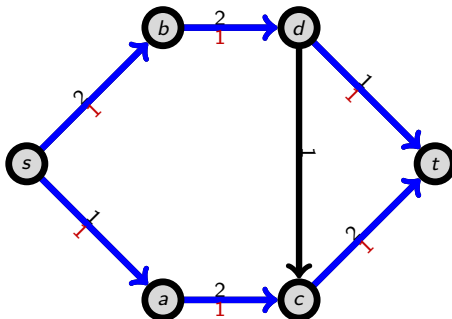
## Lemma

*Die Laufzeit des Algorithmus von Dinitz ist  $O(m \cdot n^2)$ .*

Beweis: Übung

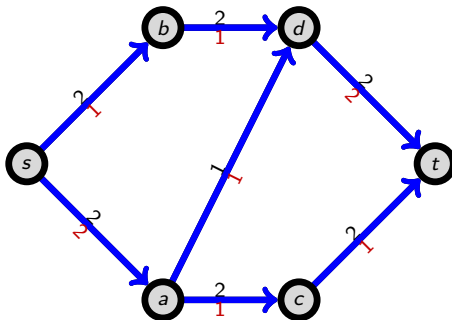
## Beispiel zu den Schichten

- Die Kante  $(d, c)$  ist nicht im Schichtennetzwerk.
- Der Fluss füllt die Kanten  $(s, a)$  und  $(d, t)$  aus.
- Das nächste Schichtennetzwerk hat eine Schicht mehr.



## Beispiel zu neuen Kanten

- Wir betrachten die Kante  $(a, d)$ . Weglänge ist 3.
- Durch den Fluss ergibt sich eine neue Kante  $(d, a)$ .
- Ein Weg über  $(d, a)$  hat dann aber Länge  $3 + 2$ .



# Idee der Propagation

Um eine Laufzeit von  $O(n^3)$  zu erreichen, muss das Niveaunetzwerk in  $O(n^2)$  saturiert werden (Sperrfluss bestimmt werden). Das geschieht, indem man den schwächsten Knoten bestimmt und dann den Fluss, der über diesen schwächsten Knoten gehen kann, propagiert. D.h. der durch ihn mögliche Fluss wird als Überfluss Richtung Senke (und Quelle) propagiert.

Da wir auf einem Niveaunetzwerk arbeiten, wo die Wege zur Senke (oder Quelle) eindeutig sind, d.h. jede Kante wird in einer eindeutigen Richtung genutzt, kann der Fluss von dem schwächsten Knoten nie blockiert werden.

Beim Propagieren wird der Fluss BFS-artig verteilt, d.h. der Überfluss an einem Knoten wird erst über alle Kanten verteilt, bevor der nächste Knoten mit Überfluss betrachtet wird. Es werden erst alle Knoten auf einem Level behandelt, bevor das nächste betrachtet wird.

Die ausgehenden (und eingehenden) Kanten eines Knoten werden, solange es geht, vollständig genutzt. Dadurch gibt es höchstens eine ausgehende (und eingehende) Kante pro Knoten, die noch nicht saturiert ist.

# Situation

- Die Berechnung des Sperrflusses auf einem Niveaunetzwerk
- Ziel: Suche besseren Algorithmus zur Sperrflussberechnung.
- Idee: Fülle einen Knoten mit Fluss aus.
- D.h. suche den Knoten  $v$ , der am wenigsten Fluss  $f_v$  weiterleiten kann.
- Propagiere dann diesen Fluss  $f_v$  von  $v$  nach  $s$  und nach  $t$ .
- D.h. mache: Forward-Backward-Propagation.
- Im folgenden wird nur diese neue Sperrflussberechnung angegeben.

# Definitionen

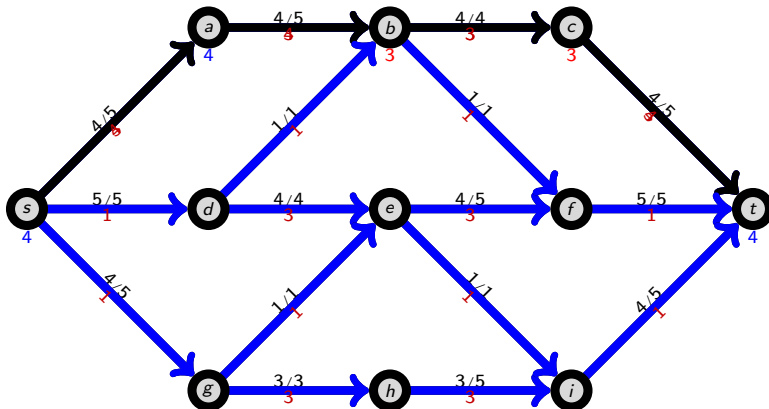
- Sei  $G'_f = (V, E'_f, s, t, c')$  ein Niveaunetzwerk.
- $N_{in}(v) = \{(w, v) \mid (w, v) \in E'_f\}$ .
- $N_{out}(v) = \{(v, w) \mid (v, w) \in E'_f\}$ .
- $\text{pot}(e) = \text{rest}_f(e)$  ist das Potential einer Kante  $e$ .
- $\text{pot}(s) = \sum_{e \in N_{out}(s)} \text{pot}(e)$  ist das Potential von der Quelle  $s$ .
- $\text{pot}(t) = \sum_{e \in N_{in}(t)} \text{pot}(e)$  ist das Potential von der Senke  $t$ .
- $\text{pot}(v) = \min\{\sum_{e \in N_{in}(v)} \text{pot}(e), \sum_{e \in N_{out}(v)} \text{pot}(e)\}$  ist das Potential eines Knoten  $v \in V \setminus \{s, t\}$ .



# Idee (Forward Propagation)

- ① Sei  $G'_f = (V, E'_f, s, t, c')$  ein Niveaunetzwerk.
- ② Bestimme  $v \in V$  mit:  $\text{pot}(v) > 0$  und  $\forall w \in V : \text{pot}(v) \leq \text{pot}(w)$ .
- ③ Lege auf  $v$  einen Überschuss von  $\text{pot}(v)$ , d.h. Setze  $U(v) = \text{pot}(v)$ .
- ④ Setze  $U(w) = 0$  für alle Knoten  $w \in V \setminus \{v\}$ .
- ⑤ Solange es einen Knoten  $v'$  gibt mit  $U(v') > 0$ , verschiebe den Überfluss auf die Nachfolger aus  $N_{\text{out}}(v')$ .
- ⑥ Verwende, um gute Laufzeit zu erreichen, dazu eine Schlange.

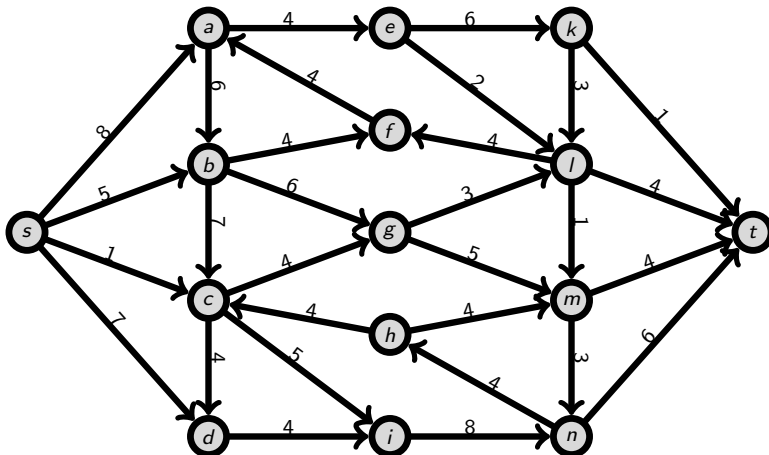
# kleines Beispiel (Niveaunetzwerk 1)



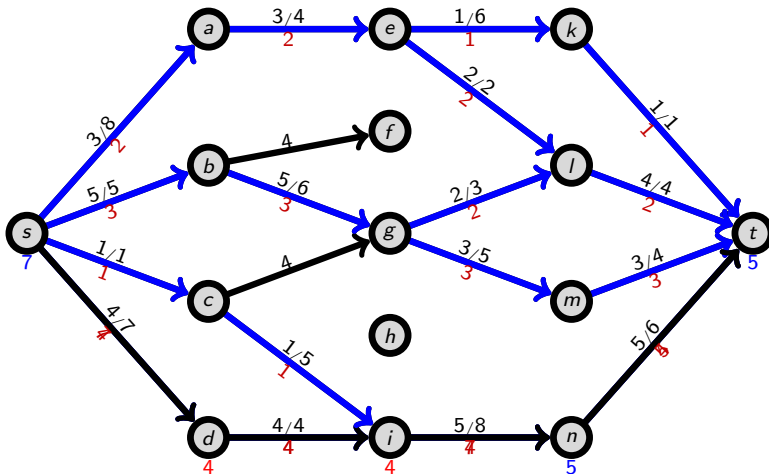
# Algorithmus (Forward Propagation)

- ① Sei  $G'_f = (V, E'_f, s, t, c')$  ein Niveaunetzwerk.
- ② Bestimme  $v \in V$  mit:  $\text{pot}(v) > 0$  und  $\forall w \in V : \text{pot}(v) \leq \text{pot}(w)$ .
- ③ Setze  $U(v) = \text{pot}(v)$  und  $\forall w \in V \setminus \{v\}$  setze  $U(w) = 0$ .
- ④ *Enqueue*( $v, Q$ ).
- ⑤ Solange  $Q$  nicht leer ist, mache:
  - ①  $v = \text{Dequeue}(Q)$ .
  - ② Solange  $U(v) > 0$  mache:
    - ① Für jedes  $e = (v, w) \in V_{\text{out}}(v)$ :
    - ②  $f''(e) = \min\{\text{pot}(e), U(v)\}$
    - ③  $U(v) = U(v) - f''(e)$
    - ④  $U(w) = U(w) + f''(e)$
    - ⑤ Falls  $w \neq t$  und  $U(w) = f''(e)$ , mache *Enqueue*( $w, Q$ ).

## schönes Beispiel (Dinitz)



## schönes Beispiel (Niveaunetzwerk 1)



# Iterierte Propagation

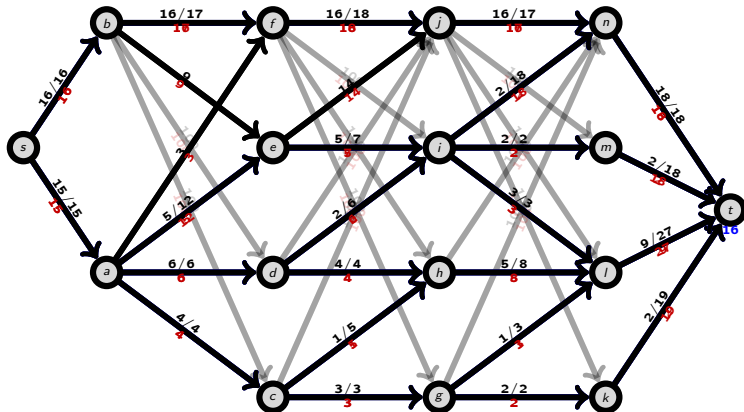
- ① Sei  $G'_f = (V, E'_f, s, t, c')$  ein Niveaunetzwerk.
- ② Solange kein Sperrfluss berechnet ist, wiederhole:
  - ① Führe eine Propagationsphase aus.
  - ② Solange es saturierte Kanten und Knoten gibt, entferne diese.

Lemma (Anzahl der Iterationen)

*Nach spätestens  $n - 1$  Propagationsphasen ist in dem Niveaunetzwerk ein Sperrfluss bestimmt.*

Beweis: In jeder Iteration wird mindestens ein Knoten saturiert.

# Beispiel zur Laufzeit



# Forward-Propagation

## Lemma (Laufzeit einer Propagation)

*Eine Forward-Propagation kann in Zeit  $O(n + I)$  durchgeführt werden. Dabei ist  $I$  die Anzahl der neu saturierten Kanten.*

Beweis:

- Wegen der FIFO-Schlange  $Q$  werden die Niveaus nacheinander bearbeitet.
- Jeder Knoten wird höchstens einmal aus  $Q$  entnommen.
- Bei jedem Knoten werden von den betrachteten Kanten alle bis auf höchstens eine saturiert.
- Pro Knoten gibt es maximal eine Kante, die betrachtet, aber nicht saturiert wird.
- Damit:
  - maximal  $n$  Knoten werden betrachtet.
  - maximal  $I$  Kanten werden saturiert und nicht mehr betrachtet.
  - maximal  $n$  Kanten werden betrachtet, aber nicht saturiert.



# Sperrflussberechnung

## Lemma (Laufzeit der Sperrflussberechnung)

*Ein Sperrfluss kann in Zeit  $O(m + n^2)$  berechnet werden.*

Beweis:

- Nach spätestens  $n - 1$  Propagationsphasen ist in dem Niveaunetzwerk ein Sperrfluss bestimmt.
- Sei  $l_i$  die Anzahl der saturierten Kanten in der  $i$ -ten Propagationsphase ( $1 \leq i \leq n - 1$ ).
- Laufzeit damit:

$$\sum_{i=1}^{n-1} O(n + l_i) = O(n^2 + \sum_{i=1}^{n-1} l_i) = O(n^2 + m)$$

## Theorem (Laufzeit von Dinitz Algorithmus)

*Der Algorithmus von Dinitz hat Laufzeit  $O(n^3)$ .*

## Literatur

- Ahuja, Magnanti, Orlin: Network Flows: Theory, Algorithms, and Applications, Prentice Hall, 1993.
- Cormen, Leiserson, Rives: Introduction to Algorithms, First Edition, MIT Press, 1990.
- Cormen, Leiserson, Rives: Introduction to Algorithms, Second Edition, MIT Press, 2001.
- Ottmann, Widmayer: Algorithmen und Datenstrukturen. BI-Wiss.-Verl. 1990.