

Algorithms and Probability

Week 12

09/05/2025 — Georg Hasebe

Lange Pfade

Gegeben: (G, B) , G ein Graph und $B \in \mathbb{N}_0$.

Problem: gibt es einen Pfad der Länge B in G ?

Zur Erinnerung:

Ein **Pfad der Länge** ℓ in einem Graph $G = (V, E)$ ist eine Folge von paarweise verschiedenen Knoten

$$\langle v_0, v_1, \dots, v_\ell \rangle, \text{ mit } \{v_{i-1}, v_i\} \in E \text{ für } i = 1, \dots, \ell.$$

Es liegen $\ell + 1$ Knoten auf einem Pfad der Länge ℓ .

Lange Pfade

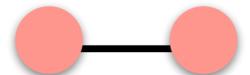
Für beliebige $B \in \mathbb{N}_0$ vermutlich sehr schwer (vgl. $B = n - 1 \Rightarrow$ Hamiltonpfad).

Was passiert wenn B klein ist?

Konkret:

$$B = O(\log n).$$

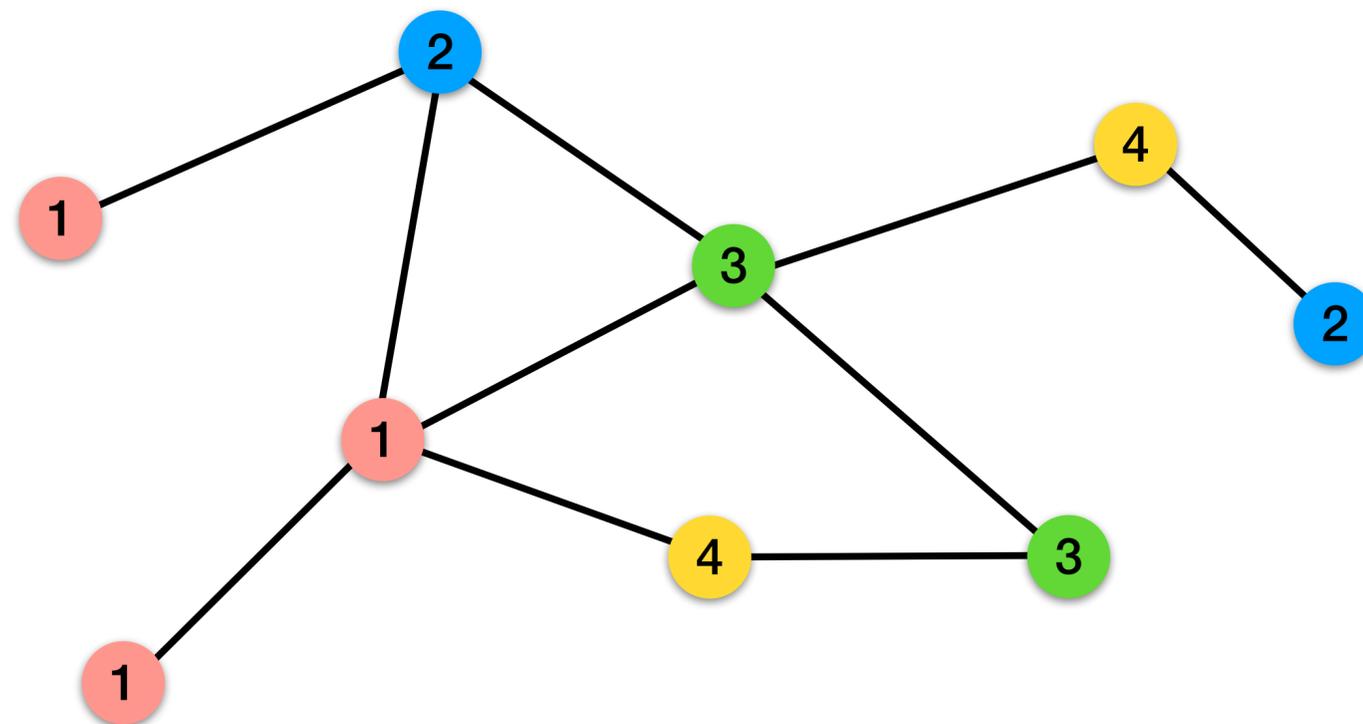
Colorful-Path Problem

Gegeben: (G, γ) , $G = (V, E)$ ein Graph und $\gamma : V \rightarrow [k]$ eine Färbung (muss nicht gültig sein; d.h.  ist erlaubt)

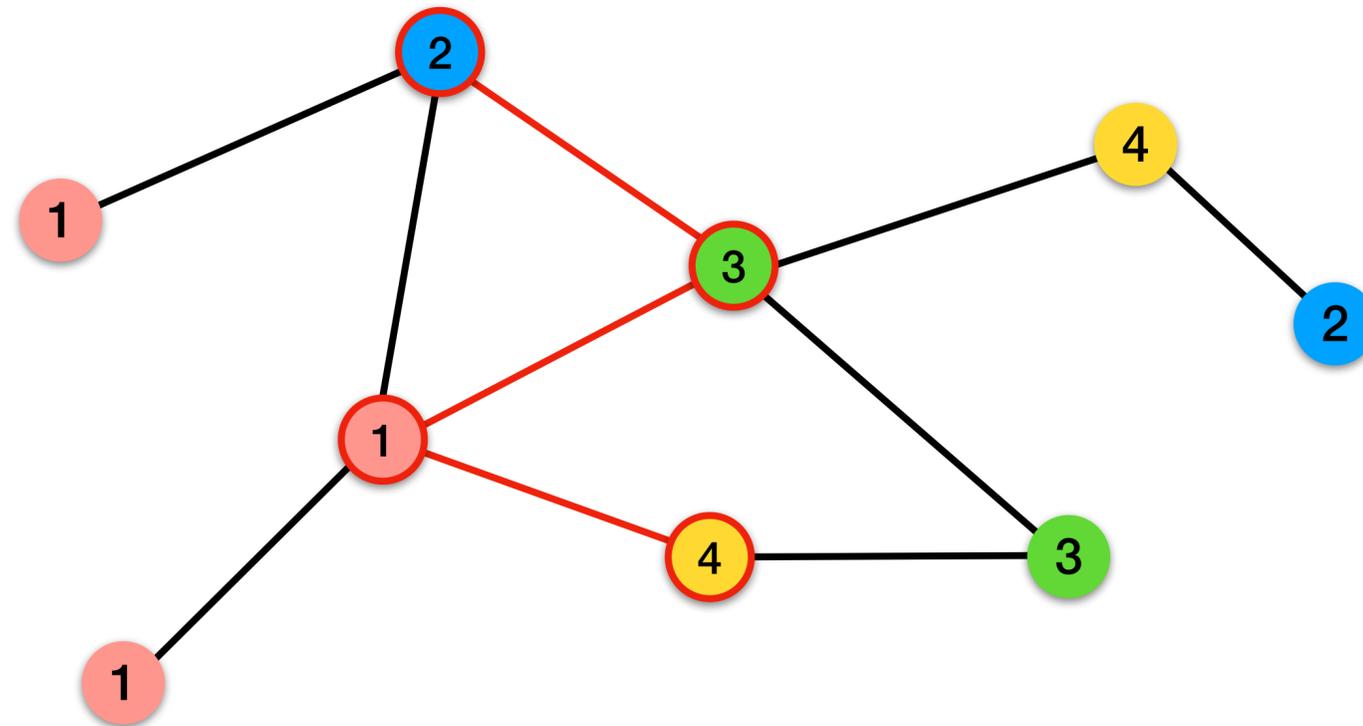
d.h. mit k Knoten

Problem: gibt es einen bunten Pfad der Länge $k - 1$ in G ?

Definition: Ein Pfad heisst **bunt**, falls alle seine Knoten verschiedene Farben haben.



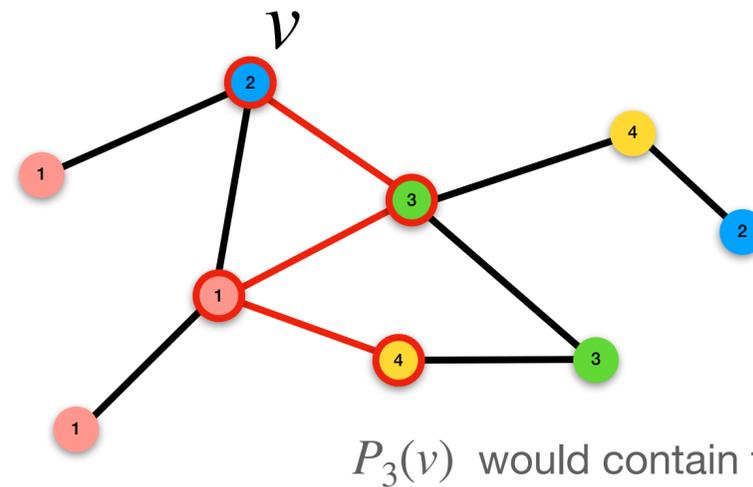
Graph $G = (V, E)$ mit Färbung $\gamma : V \rightarrow [4]$.



Ein bunter Pfad der Länge 3.

Definition: Ein Pfad heisst **bunt**, falls alle seine Knoten verschiedene Farben haben.

DP Algorithmus



Idee:

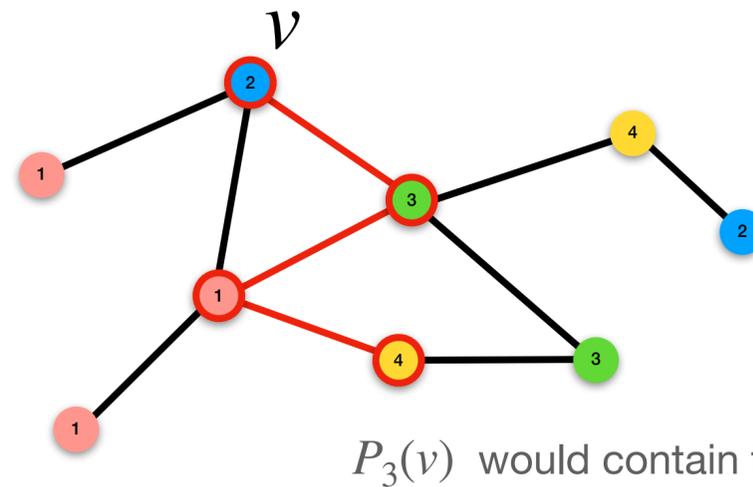
$P_i(v)$ = "Menge aller in v endender bunten Pfade der Länge i ".

Wir brauchen:

Rekursion um von $P_i(v)$ zu $P_{i+1}(v)$ zu kommen.

Die **Lösung** (bunter Pfad der Länge $k - 1$, falls existent) ist in $\bigcup_{v \in V} P_{k-1}(v)$.

DP Algorithmus



Idee:

$P_i(v)$ = "Menge aller in v endender bunten Pfade der Länge i ".

Wir definieren:

$$P_i(v) := \left\{ S \in \underbrace{\binom{[k]}{i+1}} \mid \exists \text{ in } v \text{ endender mit genau } S \text{ gefärbter bunter Pfad} \right\}.$$

Ein bunter Pfad kann eine verschiedene Farben-Abfolge haben. Bei k Farben gibt es genau $\binom{k}{i+1}$

Möglichkeiten um aus k Farben $i+1$ auszuwählen. Dieser Gedanke motiviert auch die Notation.

$$P_i(v) := \left\{ S \in \binom{[k]}{i+1} \mid \exists \text{ in } v \text{ endender mit genau } S \text{ gefärbter bunter Pfad} \right\}.$$

DP Algorithmus

Wie kommen wir nun zu unserer Rekursion?

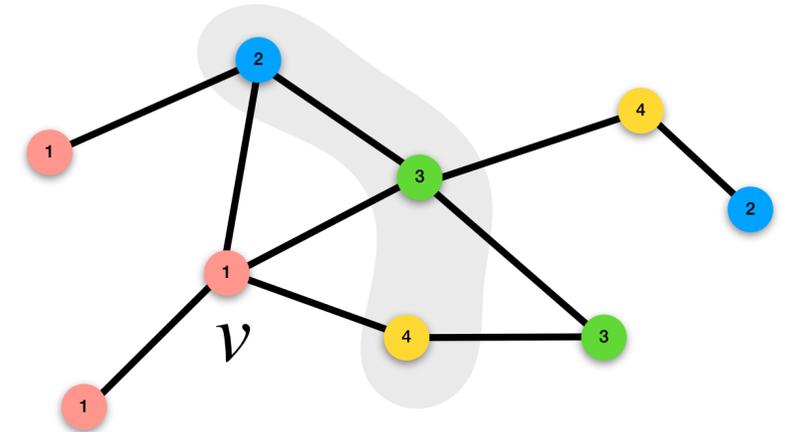
Was ist $P_0(v)$?

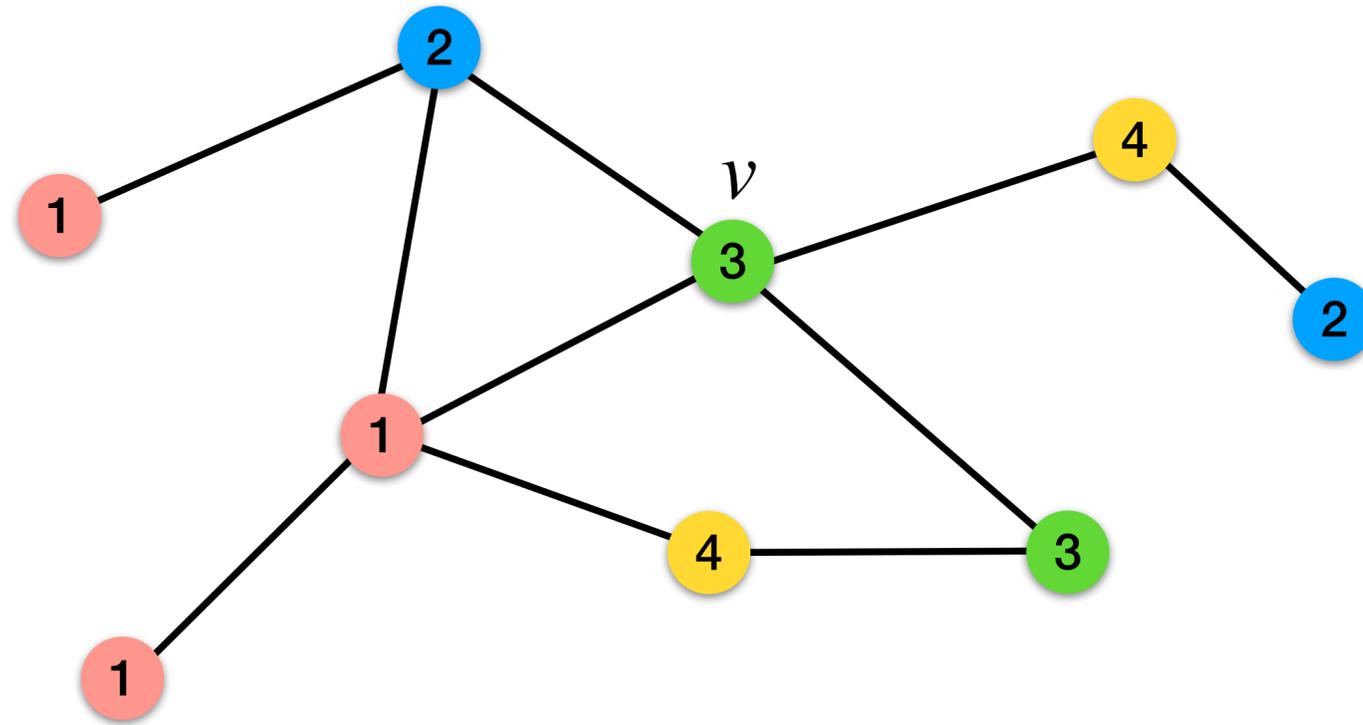
Ein in v endender bunter Pfad (Länge 0) $\Rightarrow P_0(v) = \{ \{ \gamma(v) \} \}$.

Was ist $P_1(v)$?

Ein bunter Pfad der Länge 1 zu v besteht aus einem $\gamma(v)$ -freien bunten Pfad der Länge 0 zu einem Nachbarn x von v plus dem Schritt zu v .

$$P_1(v) = \{ \{ \gamma(x), \gamma(v) \} \mid x \in N(v), \gamma(x) \neq \gamma(v) \}.$$





$$P_0(v) = \{ \text{3} \}$$

$$P_1(v) = \{ \{ \gamma(x), \text{3} \} \mid x \in N(v), \gamma(x) \neq \gamma(v) \} = \{ \{ \text{3}, \text{4} \}, \{ \text{3}, \text{2} \}, \{ \text{3}, \text{1} \} \}$$

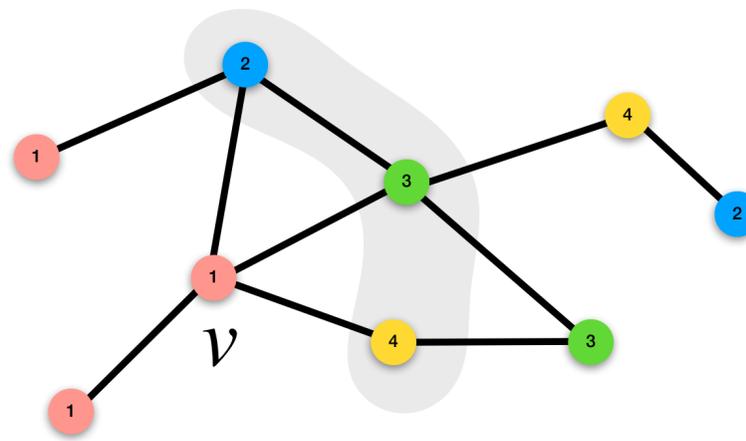
$$P_i(v) := \{S \in \binom{[k]}{i+1} \mid \exists \text{ in } v \text{ endender mit genau } S \text{ gefärbter bunter Pfad}\}.$$

DP Algorithmus

Die vorherigen Überlegungen motivieren folgende Rekursion:

$$P_i(v) = \bigcup_{x \in N(v)} \{R \cup \{\gamma(v)\} \mid R \in P_{i-1}(x) \text{ und } \gamma(v) \notin R\}.$$

Ein bunter Pfad der Länge i zu v besteht aus einem $\gamma(v)$ -freien bunten Pfad der Länge $i - 1$ zu einem Nachbarn x von v plus dem Schritt zu v .



Rekursion ähnelt dem Algorithmus für Hamiltonkreise mit dynamischer Programmierung.

Betrachten Sie einen Graph $G = (V, E)$ mit $|V| = n$ Knoten und eine k -(Knoten)-Färbung $c : V \rightarrow [k]$, wobei $k = \lceil \log n \rceil$.

Welche der folgenden Definitionen kann benutzt werden, um mittels DP in polynomieller Zeit (in n) zu entscheiden, ob G einen bunten Pfad mit k Knoten enthält?

$C_i(v) := \{ \text{Farbfolgen } c \in [k]^i \text{ so dass } \exists \text{ ein Pfad mit } i \text{ Knoten beginnend bei } v \text{ mit Farben } c_1, c_2, \dots \}$



$A_i(v) := \{ S \in \binom{[k]}{i} : \exists \text{ ein Pfad mit } i \text{ Knoten und Endknoten } v, \text{ der alle Farben von } S \text{ genau einmal verwendet} \}$



$B_i(v) := \{ \text{alle bunten Pfade mit } i \text{ Knoten und } v \text{ als Endknoten} \}$



DP Algorithmus

$$P_i(v) := \{S \in \binom{[k]}{i+1} \mid \exists \text{ in } v \text{ endender mit genau } S \text{ gefärbter bunter Pfad}\}.$$

$$P_i(v) = \bigcup_{x \in N(v)} \{R \cup \{\gamma(v)\} \mid R \in P_{i-1}(x) \text{ und } \gamma(v) \notin R\}.$$

Bunt(G, i)

G ein γ -gefärbter Graph

```
1: for all  $v \in V$  do
2:    $P_i(v) \leftarrow \emptyset$ 
3:   for all  $x \in N(v)$  do
4:     for all  $R \in P_{i-1}(x)$  mit  $\gamma(v) \notin R$  do
5:        $P_i(v) \leftarrow P_i(v) \cup \{R \cup \{\gamma(v)\}\}$ 
```

Ein bunter Pfad der Länge i zu v besteht aus einem $\gamma(v)$ -freien bunten Pfad der Länge $i - 1$ zu einem Nachbarn x von v plus dem Schritt zu v .

$$P_i(v) := \{S \in \binom{[k]}{i+1} \mid \exists \text{ in } v \text{ endender mit genau } S \text{ gefärbter bunter Pfad}\}.$$

$$P_i(v) = \bigcup_{x \in N(v)} \{R \cup \{\gamma(v)\} \mid R \in P_{i-1}(x) \text{ und } \gamma(v) \notin R\}.$$

DP Algorithmus

Regenbogen(G, γ)

G Graph, γ k -Färbung

- 1: **for all** $v \in V$ **do** $P_0(v) \leftarrow \{\{\gamma(v)\}\}$
 - 2: **for** $i = 1..k - 1$ **do** Bunt(G, i)
 - 3: **return** $\bigcup_{v \in V} P_{k-1}(v) \neq \emptyset$
-

Bunt(G, i)

G ein γ -gefärbter Graph

- 1: **for all** $v \in V$ **do**
 - 2: $P_i(v) \leftarrow \emptyset$
 - 3: **for all** $x \in N(v)$ **do**
 - 4: **for all** $R \in P_{i-1}(x)$ mit $\gamma(v) \notin R$ **do**
 - 5: $P_i(v) \leftarrow P_i(v) \cup \{R \cup \{\gamma(v)\}\}$
-

Laufzeit

Regenbogen(G, γ)	G Graph, γ k -Färbung
1: for all $v \in V$ do $P_0(v) \leftarrow \{\{\gamma(v)\}\}$	$O(n)$
2: for $i = 1..k - 1$ do Bunt(G, i)	???
3: return $\bigcup_{v \in V} P_{k-1}(v) \neq \emptyset$	$O(n)$

Bunt(G, i)	G ein γ -gefärbter Graph
1: for all $v \in V$ do	
2: $P_i(v) \leftarrow \emptyset$	
3: for all $x \in N(v)$ do	
4: for all $R \in P_{i-1}(x)$ mit $\gamma(v) \notin R$ do	
5: $P_i(v) \leftarrow P_i(v) \cup \{R \cup \{\gamma(v)\}\}$	

Laufzeit

Regenbogen(G, γ)	G Graph, γ k -Färbung
1: for all $v \in V$ do $P_0(v) \leftarrow \{\{\gamma(v)\}\}$	$O(n)$
2: for $i = 1..k - 1$ do Bunt(G, i)	???
3: return $\bigcup_{v \in V} P_{k-1}(v) \neq \emptyset$	$O(n)$

Bunt(G, i)	G ein γ -gefärbter Graph
1: for all $v \in V$ do	
2: $P_i(v) \leftarrow \emptyset$	
3: for all $x \in N(v)$ do	
4: for all $R \in P_{i-1}(x)$ mit $\gamma(v) \notin R$ do	$ P_{i-1}(x) $
5: $P_i(v) \leftarrow P_i(v) \cup \{R \cup \{\gamma(v)\}\}$	i

$$\sum_{v \in V} \deg(v)$$

Bunt(G, i)

 G ein γ -gefärbter Graph

Laufzeit1: **for all $v \in V$ do**2: $P_i(v) \leftarrow \emptyset$ 3: **for all $x \in N(v)$ do**4: **for all $R \in P_{i-1}(x)$ mit $\gamma(v) \notin R$ do**5: $P_i(v) \leftarrow P_i(v) \cup \{R \cup \{\gamma(v)\}\}$ $|P_{i-1}(x)|$
 i $\sum_{v \in V} \deg(v)$

Wir haben also $O\left(\sum_{v \in V} \deg(v) \cdot |P_{i-1}(v)| \cdot i\right)$ pro Bunt(G, i) Runde.

Nach dem Handshake Lemma und weil $P_{i-1}(v) \subseteq \binom{[k]}{i}$ und daher $|P_{i-1}(v)| \leq \binom{k}{i}$ bekommen wir pro Bunt(G, i) Runde:

$$O\left(\sum_{v \in V} \deg(v) \cdot |P_{i-1}(v)| \cdot i\right) = O\left(m \cdot \binom{k}{i} \cdot i\right).$$

Laufzeit

Regenbogen(G, γ)	G Graph, γ k -Färbung
---------------------------	----------------------------------

1: for all $v \in V$ do $P_0(v) \leftarrow \{\{\gamma(v)\}\}$	$O(n)$
2: for $i = 1..k - 1$ do Bunt(G, i)	$O\left(m \cdot \binom{k}{i} \cdot i\right)$
3: return $\bigcup_{v \in V} P_{k-1}(v) \neq \emptyset$	$O(n)$

Bunt(G, i)	G ein γ -gefärbter Graph
----------------	-----------------------------------

1: for all $v \in V$ do
2: $P_i(v) \leftarrow \emptyset$
3: for all $x \in N(v)$ do
4: for all $R \in P_{i-1}(x)$ mit $\gamma(v) \notin R$ do
5: $P_i(v) \leftarrow P_i(v) \cup \{R \cup \{\gamma(v)\}\}$

Regenbogen(G, γ)

 G Graph, γ k -Färbung**Laufzeit****1: for all $v \in V$ do $P_0(v) \leftarrow \{\{\gamma(v)\}\}$** $O(n)$ **2: for $i = 1..k - 1$ do Bunt(G, i)** $O\left(m \cdot \binom{k}{i} \cdot i\right)$ **3: return $\bigcup_{v \in V} P_{k-1}(v) \neq \emptyset$** $O(n)$

$$\sum_{i=1}^{k-1} m \cdot \binom{k}{i} \cdot i \leq \sum_{i=1}^k m \cdot \binom{k}{i} \cdot i \leq O(2^k \cdot k \cdot m).$$

Where we used that:

$$\sum_{i=0}^k \binom{k}{i} = 2^k.$$

Regenbogen(G, γ)	G Graph, γ k -Färbung	Laufzeit
1: for all $v \in V$ do $P_0(v) \leftarrow \{\{\gamma(v)\}\}$		$O(n)$
2: for $i = 1..k - 1$ do Bunt(G, i)		$O\left(m \cdot \binom{k}{i} \cdot i\right)$
3: return $\bigcup_{v \in V} P_{k-1}(v) \neq \emptyset$		$O(n)$

Since we have $O(2^k \cdot k \cdot m)$, for $k \leq O(\log n)$ we have a polynomial algorithm.

Colorful-Path und Lange Pfade

Lange Pfade: (G, B) , G ein Graph und $B \in \mathbb{N}_0$.

Für $k = B + 1$, färbe G **zufällig** mit k Farben, und suche einen bunten Pfad mit k Knoten.

Bei zufälliger Färbung, was ist die **Erfolgswahrscheinlichkeit**?

Möglichkeiten einen Pfad der Länge k mit k Farben zu färben?

$$k^k.$$

Möglichkeiten einen Pfad der Länge k mit k Farben zu färben, so dass dieser bunt ist?

$$k!.$$

Colorful-Path und Lange Pfade

Bei zufälliger Färbung, was ist die **Erfolgswahrscheinlichkeit**?

Taylor series for e^k .

$$p_{\text{Erfolg}} := \Pr[\exists \text{ bunter Pfad der Länge } k - 1] \geq \Pr[P \text{ ist bunt}] = \frac{k!}{k^k} \geq e^{-k}.$$

Ein Versuch:

- ▶ Laufzeit $O(2^k km)$. $p_{\text{Erfolg}} \geq e^{-k}$.

$\lceil \lambda e^k \rceil$ Versuche:

- ▶ Laufzeit $O(\lambda(2e)^k km)$.
- ▶ W'keit, dass der Algorithmus den Pfad nicht findet ist

$$\leq (1 - e^{-k})^{\lceil \lambda e^k \rceil} \leq (e^{-e^{-k}})^{\lceil \lambda e^k \rceil} \leq e^{-\lambda}.$$

Möglichkeiten einen Pfad der Länge k mit k Farben zu färben?

$$k^k.$$

Möglichkeiten einen Pfad der Länge k mit k Farben zu färben, so dass dieser bunt ist?

$$k!.$$

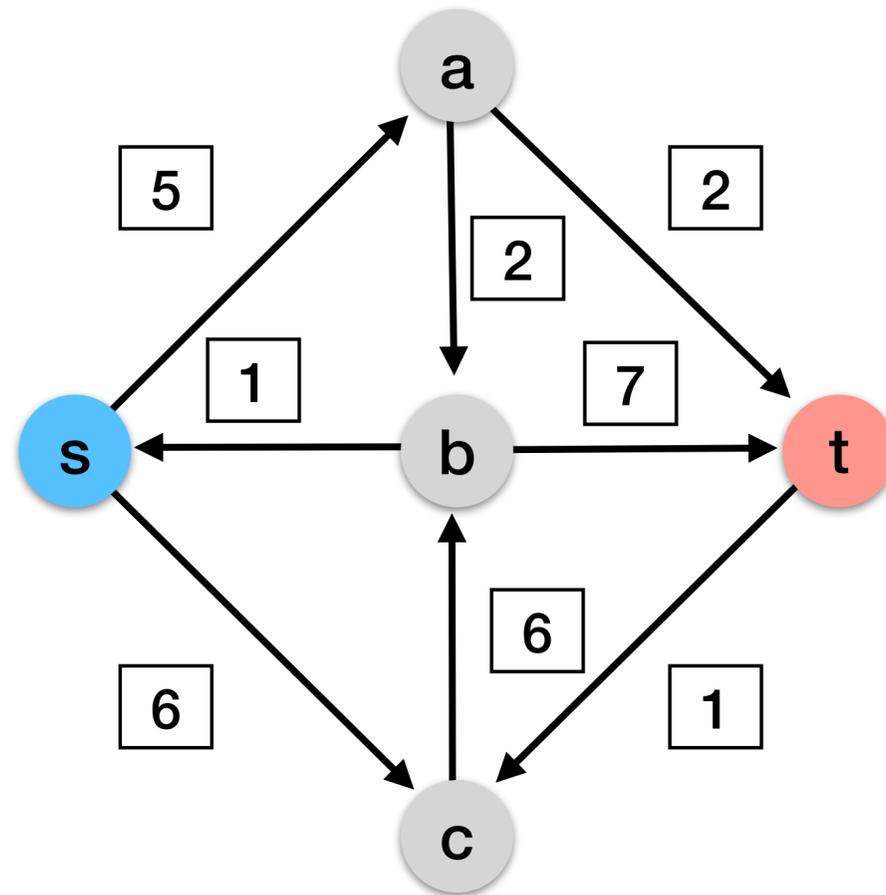
Netzwerk

Ein **Netzwerk** ist ein Tupel $N = (V, A, c, s, t)$, wobei gilt:

- ▶ (V, A) ist ein gerichteter Graph (ohne Schleifen),
- ▶ $s \in V$, die **Quelle**,
- ▶ $t \in V \setminus \{s\}$, die **Senke**, und
- ▶ $c : A \rightarrow \mathbb{R}_0^+$, die **Kapazitätsfunktion**.

Netzwerk

- Quelle (source)
- Senke (sink)
- Kapazität



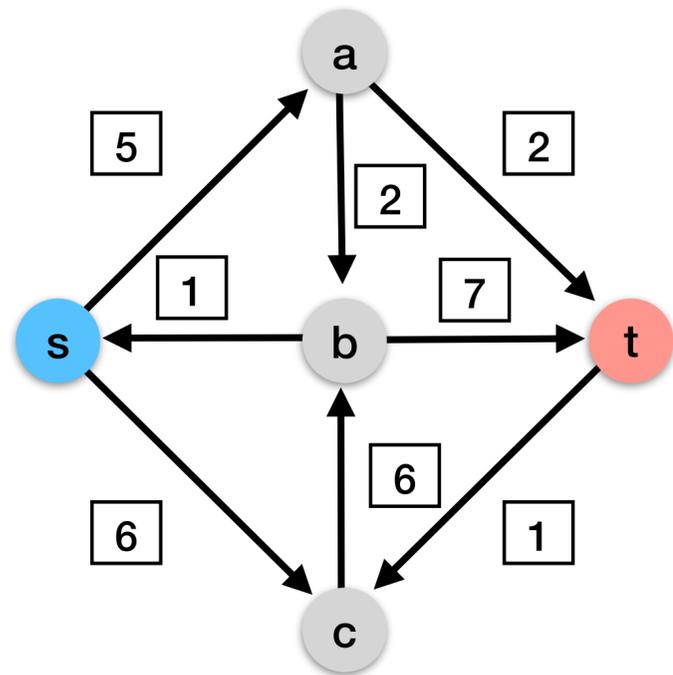
Sei $N = (V, A, c, s, t)$ ein Netzwerk. Ein **Fluss** in N ist eine Funktion $f : A \rightarrow \mathbb{R}$ mit den Bedingungen

- ▶ **Zulässigkeit:** $0 \leq f(e) \leq c(e)$ für alle $e \in A$.
- ▶ **Flusserhaltung:** Für alle $v \in V \setminus \{s, t\}$ gilt

$$\sum_{u \in V: (u,v) \in A} f(u,v) = \sum_{u \in V: (v,u) \in A} f(v,u) .$$

Der **Wert** eines Flusses f ist definiert als

$$\text{val}(f) := \text{netoutflow}(s) := \sum_{u \in V: (s,u) \in A} f(s,u) - \sum_{u \in V: (u,s) \in A} f(u,s) .$$



- Quelle (source)
- Senke (sink)
- Kapazität

Netzwerk $N = (V, A, c, s, t)$.

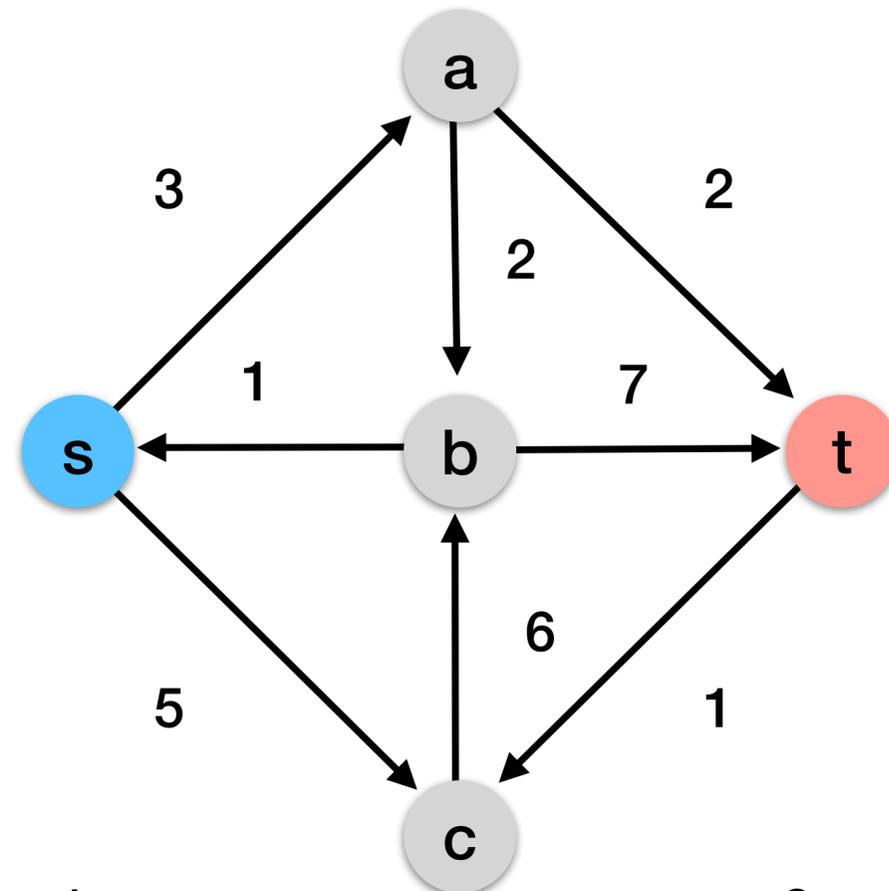
Regeln:

1. Zulässigkeit: $0 \leq f(e) \leq c(e)$ für alle $e \in A$.
2. Flusserhaltung: in v fließt gleich viel raus wie rein für alle $v \in V \setminus \{s, t\}$

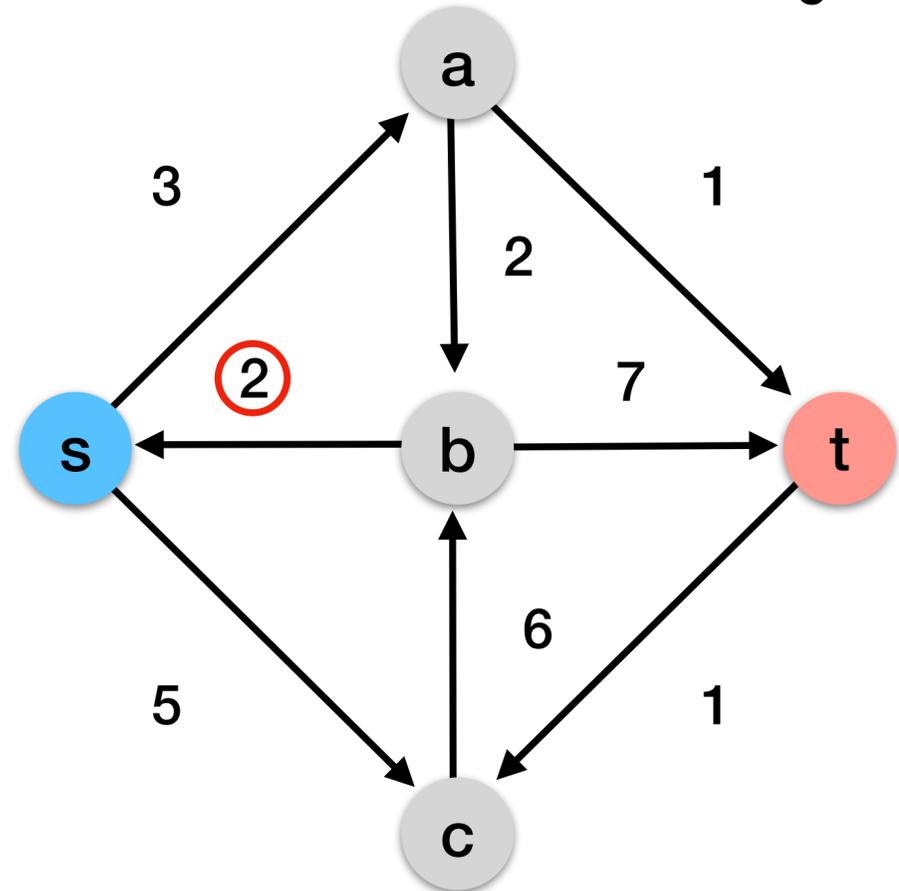
Flusswert/Netoutflow:

$$\text{val}(f) = \sum_{u \in V: (s,u) \in A} f(s,u) - \sum_{u \in V: (u,s) \in A} f(u,s)$$

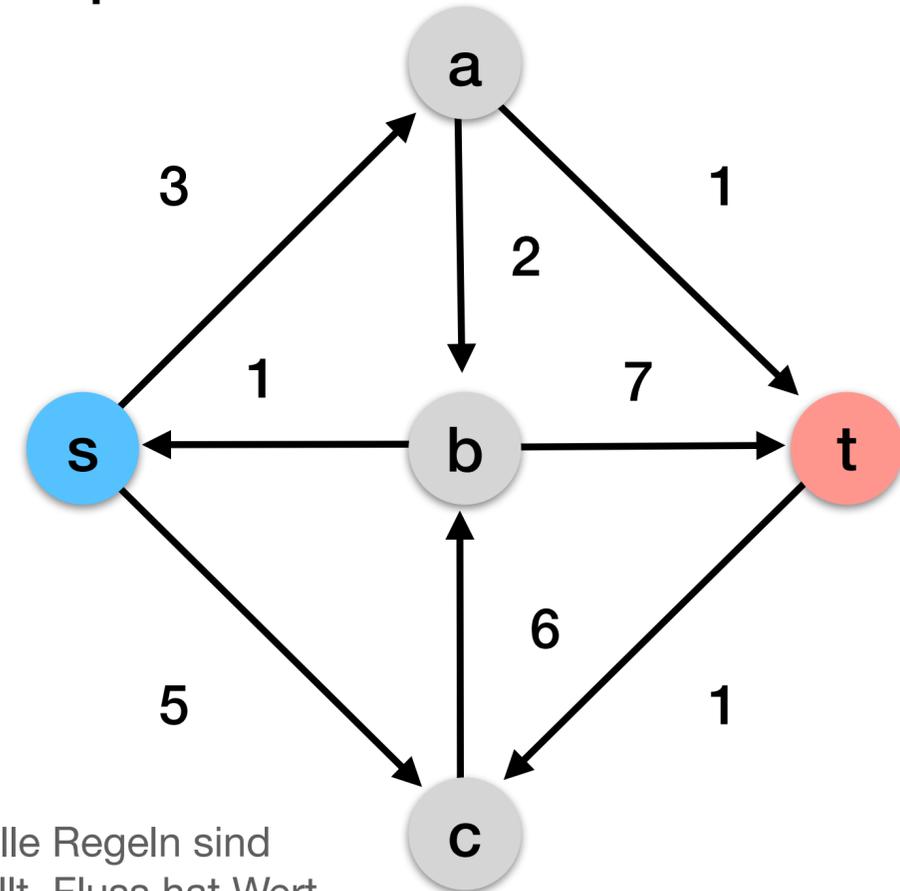
Fluss



⚡ In Knoten a fließt 3 hinein, aber 4 raus.



⚡ Hier gilt $f(e) \not\leq c(e)$.



✓ Alle Regeln sind erfüllt. Fluss hat Wert $3 + 5 - 1 = 7$.

Lemma

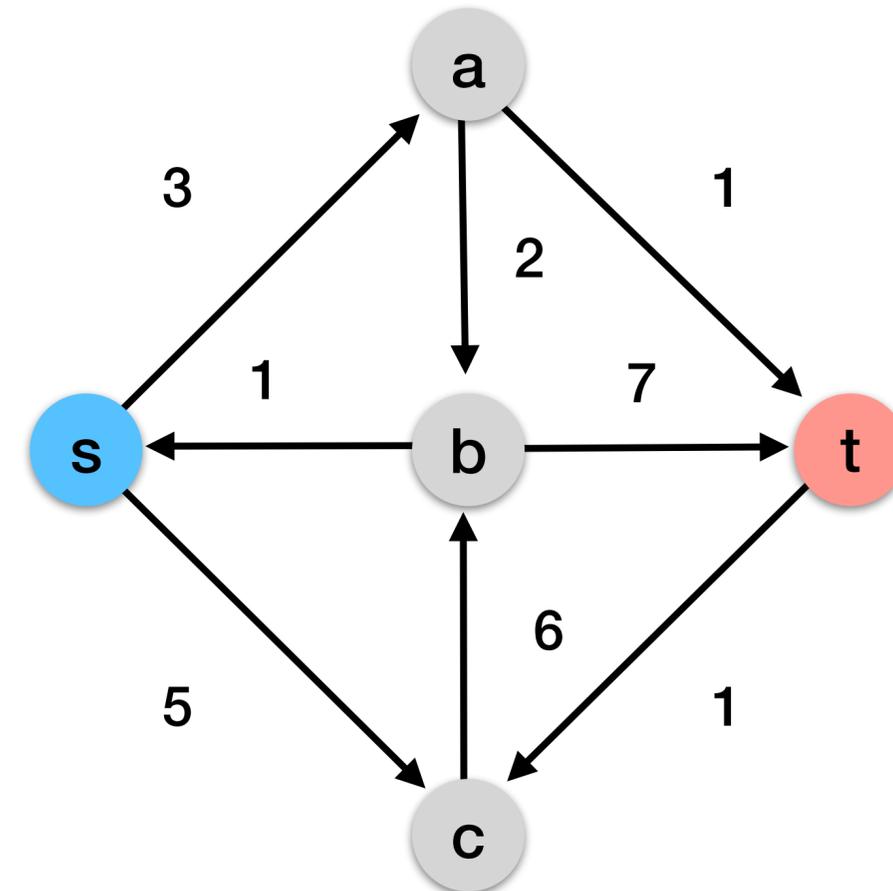
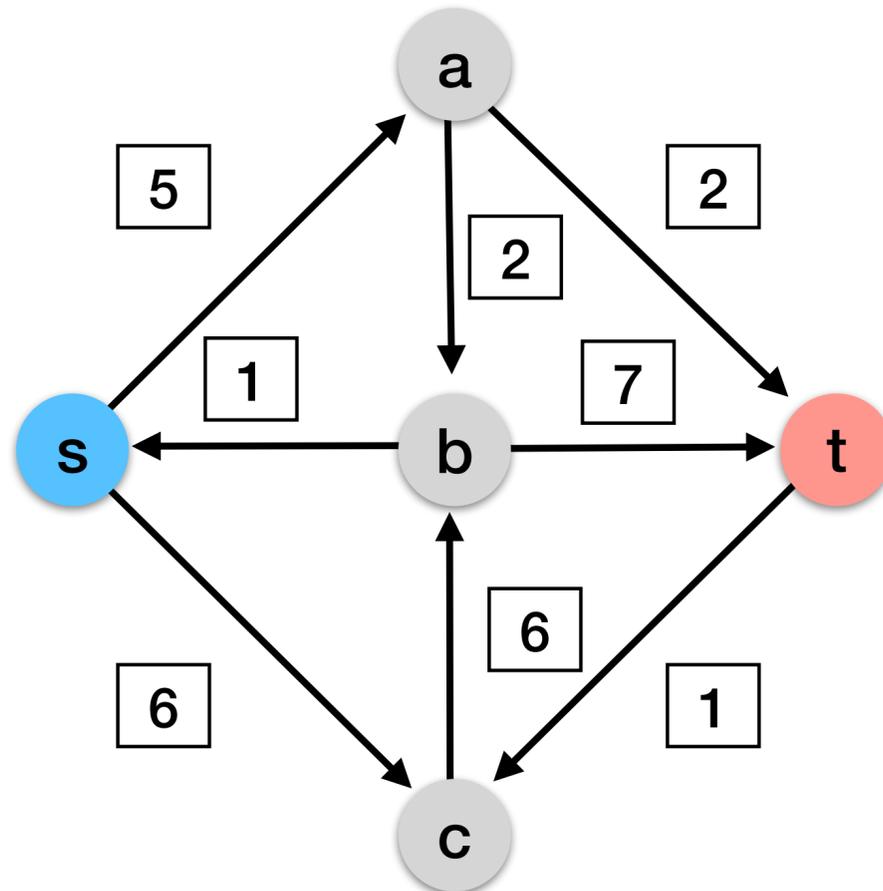
Der **Nettozufluss** der Senke t gleicht dem Wert des Flusses, d.h.

$$\text{netinflow}(t) := \sum_{u \in V: (u,t) \in A} f(u,t) - \sum_{u \in V: (t,u) \in A} f(t,u) = \text{val}(f).$$

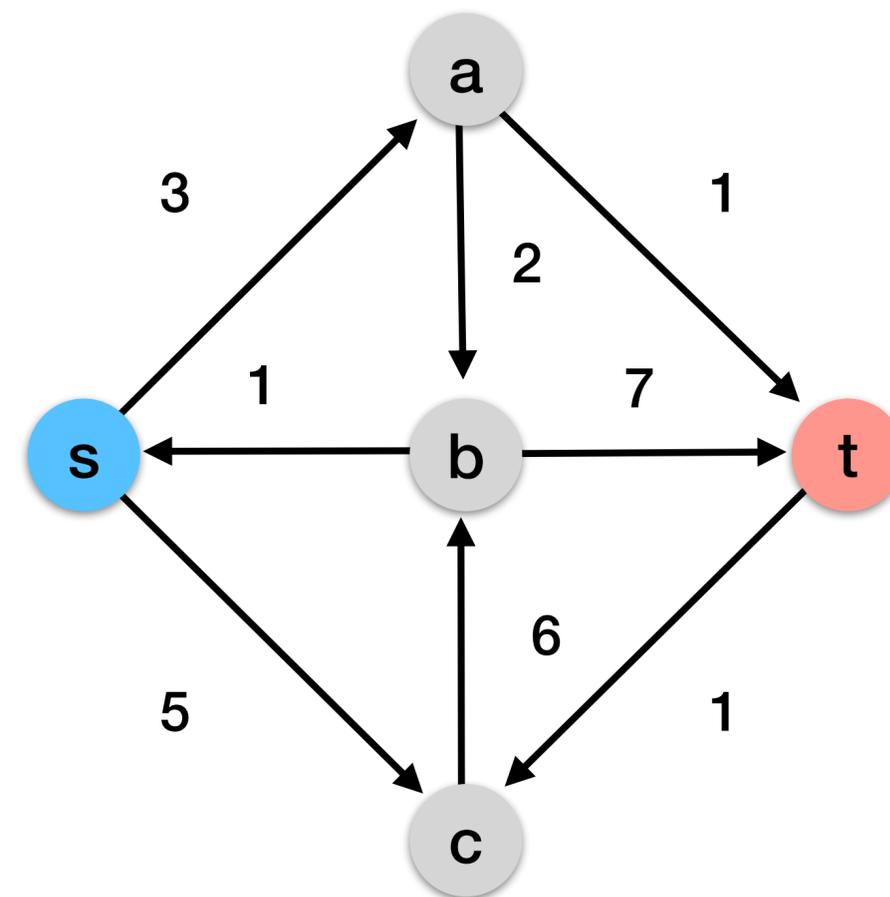
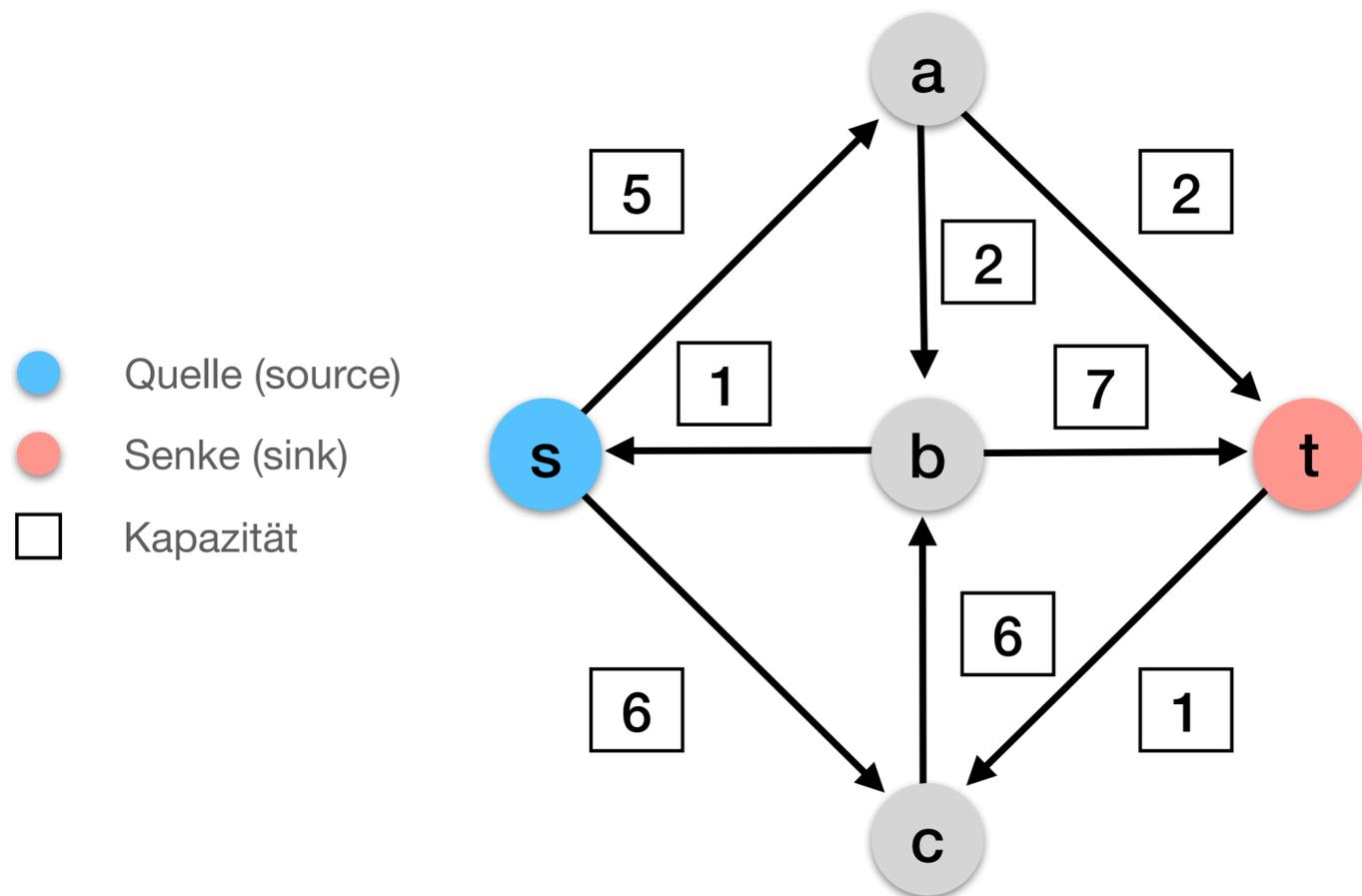
Nettozufluss

- $\text{val}(f) = \text{netoutflow}(s) = 3 + 5 - 1 = 7$.
 - $\text{netinflow}(t) = 1 + 7 - 1 = 7$.
- Wir sehen: $\text{netinflow}(t) = \text{val}(f)$.

- Quelle (source)
- Senke (sink)
- Kapazität



- Wir wissen nun was ein **Netzwerk** und ein **Fluss** eines Netzwerkes ist.
- Wir wollen nun einen **maximalen Fluss** effizient finden. Aber wie?
- **Idee:** betrachte **Schnitte**.



Ist f maximal?

Fluss f mit Wert $3 + 5 - 1 = 7$.

Es gilt also $S \cap T = \emptyset$.

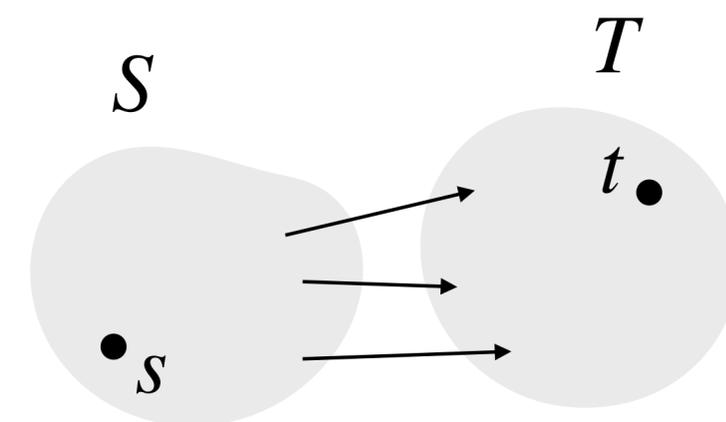
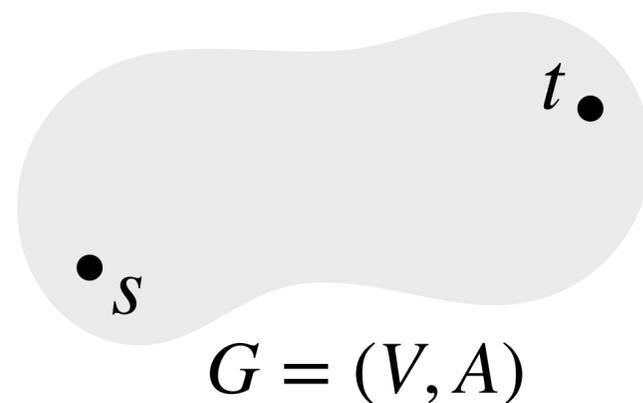
Ein **s - t -Schnitt** für ein Netzwerk (V, A, c, s, t) ist eine Partition (S, T) von V mit $s \in S$ und $t \in T$. Die **Kapazität** eines s - t -Schnitts (S, T) ist durch

$$\text{cap}(S, T) := \sum_{(u,w) \in (S \times T) \cap A} c(u, w)$$

definiert.

(Partition (S, T) : $S \cup T = V$ und $S \cap T = \emptyset$)

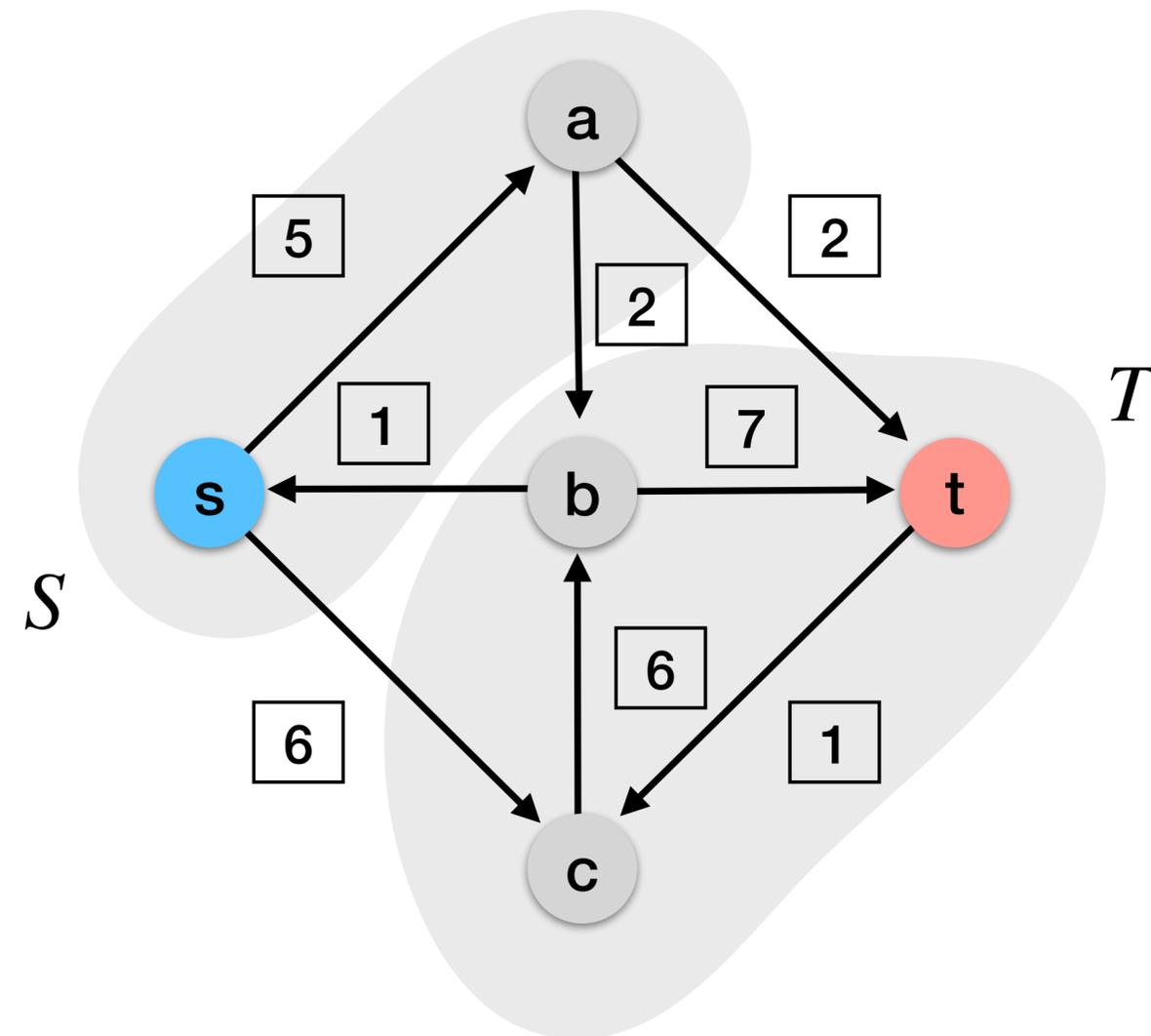
Schnitt



Wir ignorieren die Kanten von T nach S !

Schnitt

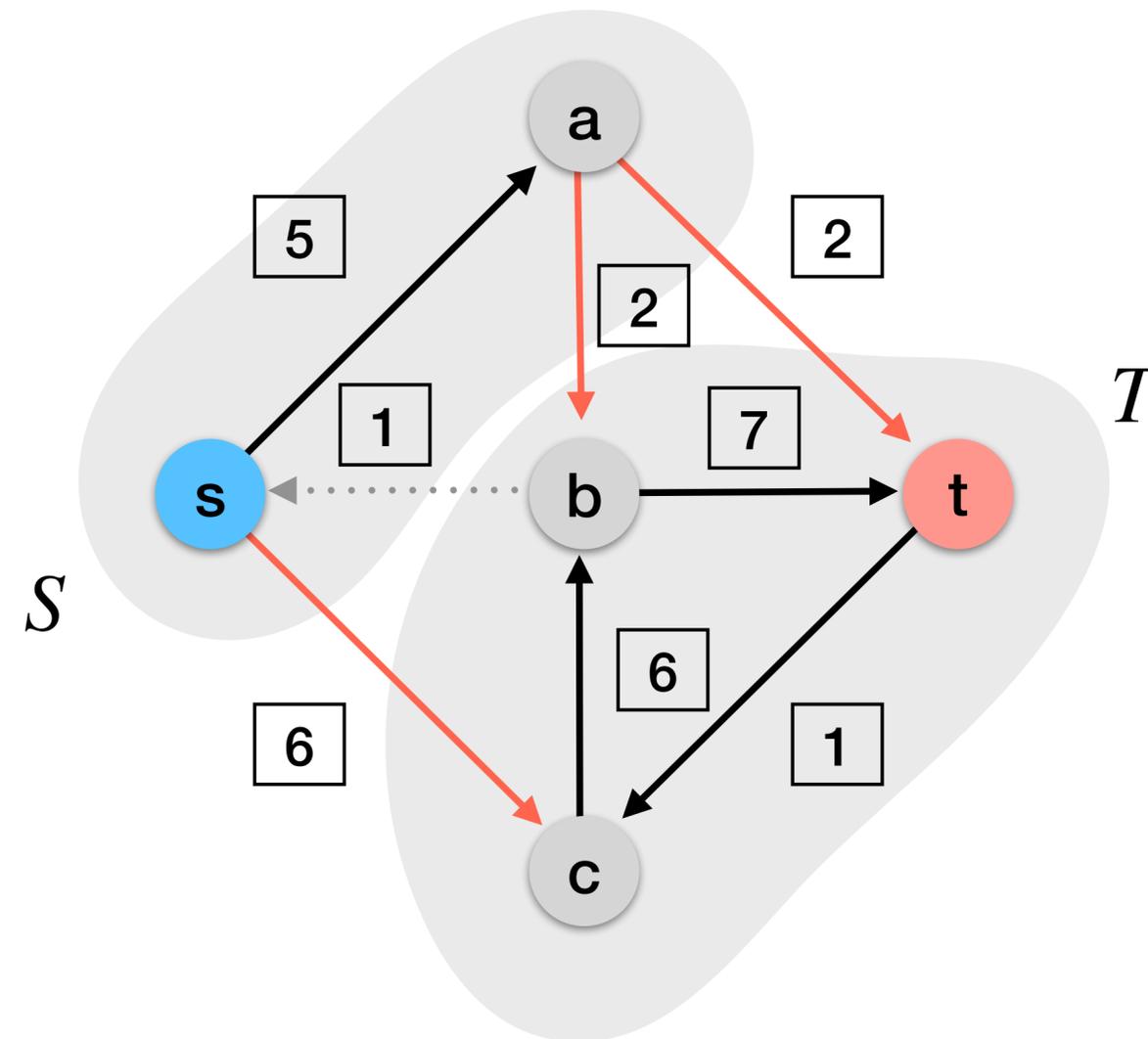
- Quelle (source)
- Senke (sink)
- Kapazität



Was ist also $\text{cap}(S, T) = \sum_{(u,w) \in (S \times T) \cap A} c(u, w)$?

Schnitt

- Quelle (source)
- Senke (sink)
- Kapazität



Die Kante von b nach s geht zwar über den Schnitt, aber von T nach S und wird deswegen **nicht mitgezählt!**

$$\text{cap}(S, T) = \sum_{u, w \in (S \times T) \cap A} c(u, w) = 2 + 2 + 6 = 10.$$

Lemma

Ist f ein Fluss und (S, T) ein s - t -Schnitt in einem Netzwerk (V, A, c, s, t) , so gilt

$$\text{val}(f) \leq \text{cap}(S, T) .$$

Ein Fluss kann nie grösser sein als die Kapazität eines s - t -Schnitts.

Finden wir zu einem Fluss f einen s - t -Schnitt (S, T) mit $\text{cap}(S, T) = \text{val}(f)$, so ist f ein maximaler Fluss.

Der Schnitt (S, T) ist ein einfacher Beweis (ein einfaches Zertifikat) für die Maximalität von f .

Zwischenergebnisse

- Schnitte ermöglichen eine Abschätzung des Flusswertes nach oben, da

$$\text{val}(f) \leq \text{cap}(S, T).$$

- Finden wir also einen Fluss f sodass $\text{val}(f) = \text{cap}(S, T)$, dann ist f maximal.

Verbleibende Fragen:

- Gibt es immer einen maximalen Fluss?
- Gibt es immer einen minimalen Schnitt sodass $\text{val}(f) = \text{cap}(S, T)$?
- Wie bestimmen wir Flüsse/Schnitte effizient?

Maxflow-Mincut Theorem

Satz („Maxflow-Mincut Theorem“)

Jedes Netzwerk $N = (V, A, c, s, t)$ erfüllt

$$\max_{f \text{ Fluss}} \text{val}(f) = \min_{(S, T) \text{ s-t-Schnitt}} \text{cap}(S, T)$$

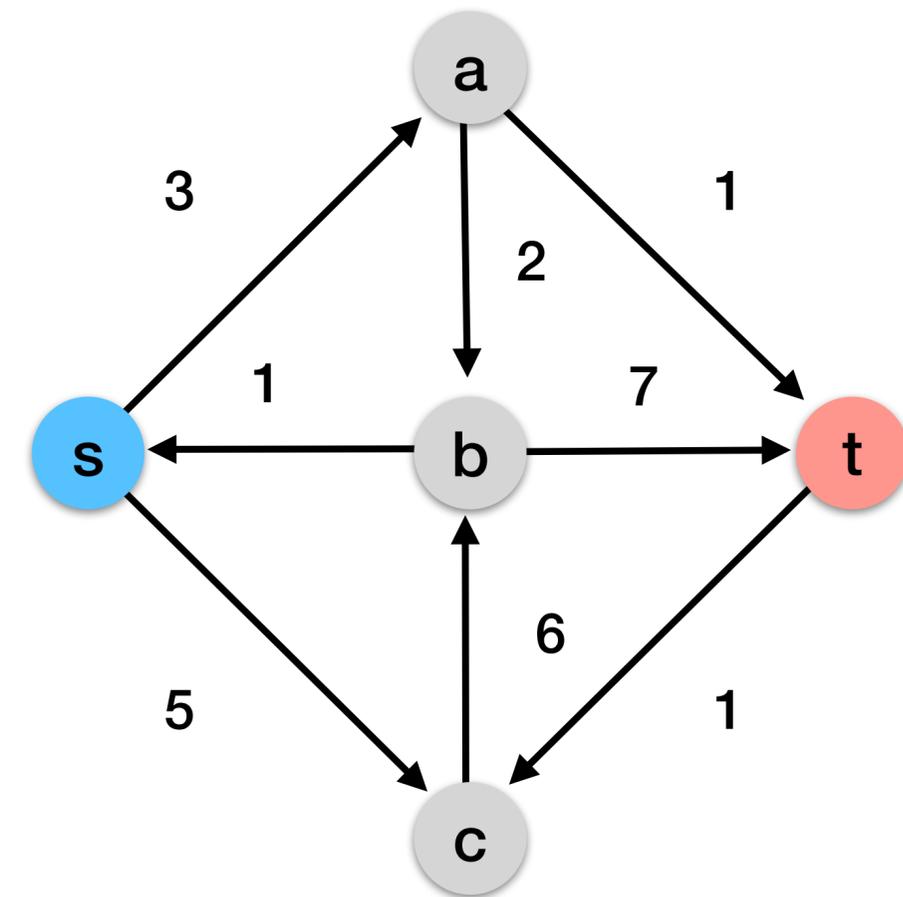
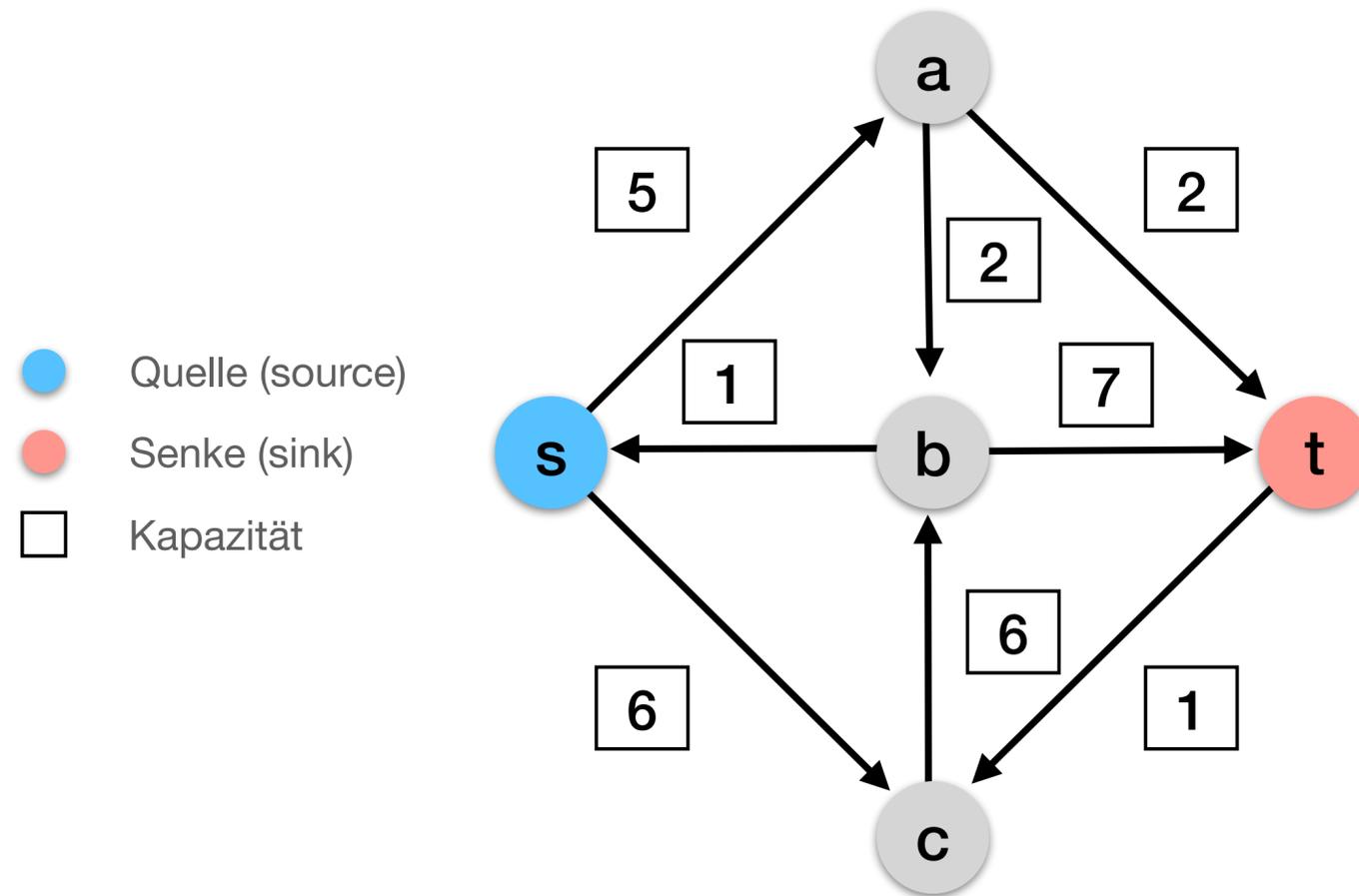
Algorithmus-Idee

1. Wir starten mit einem Fluss mit Wert 0.
2. Wir erhöhen den Flusswert nach und nach.

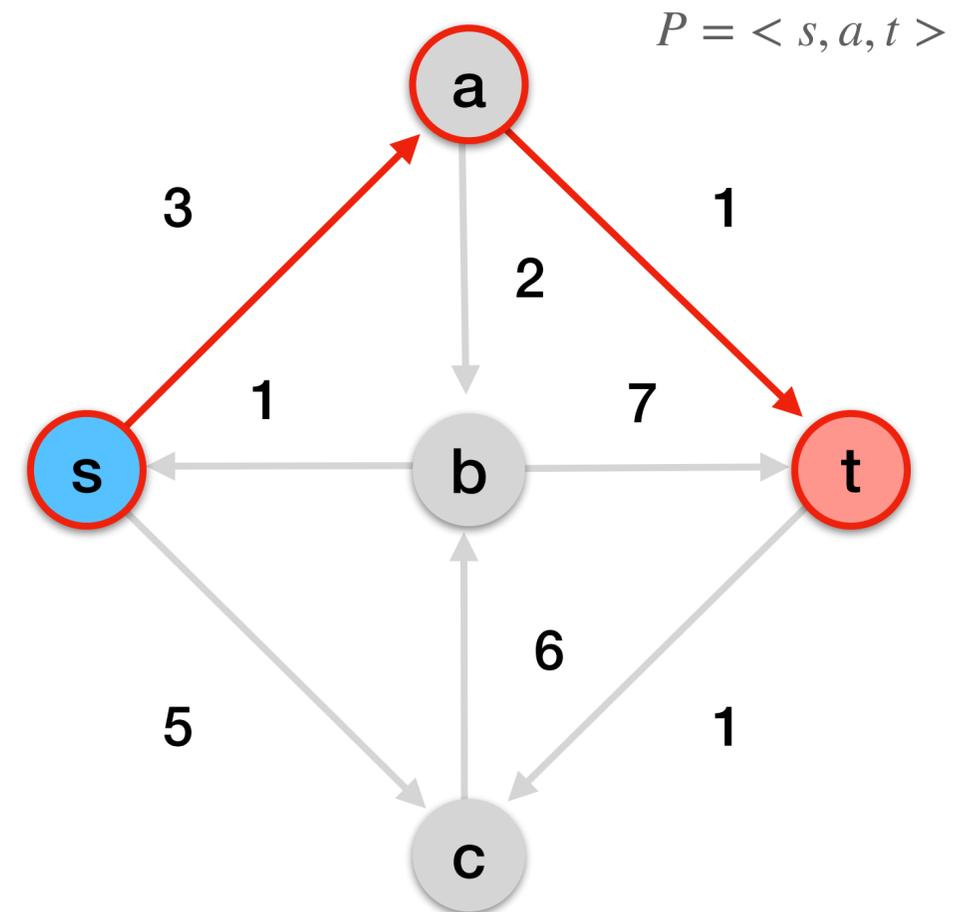
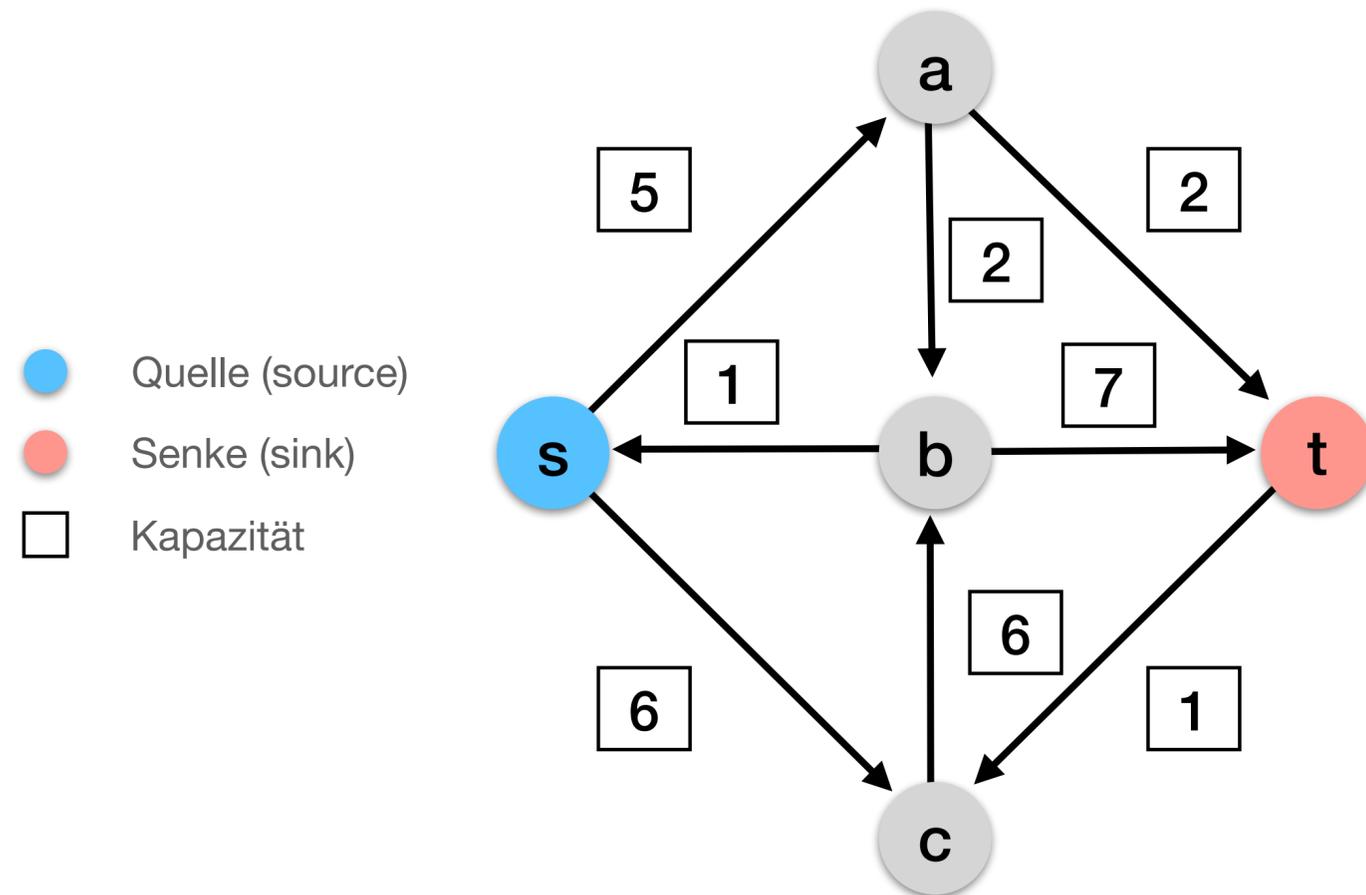
Fragen:

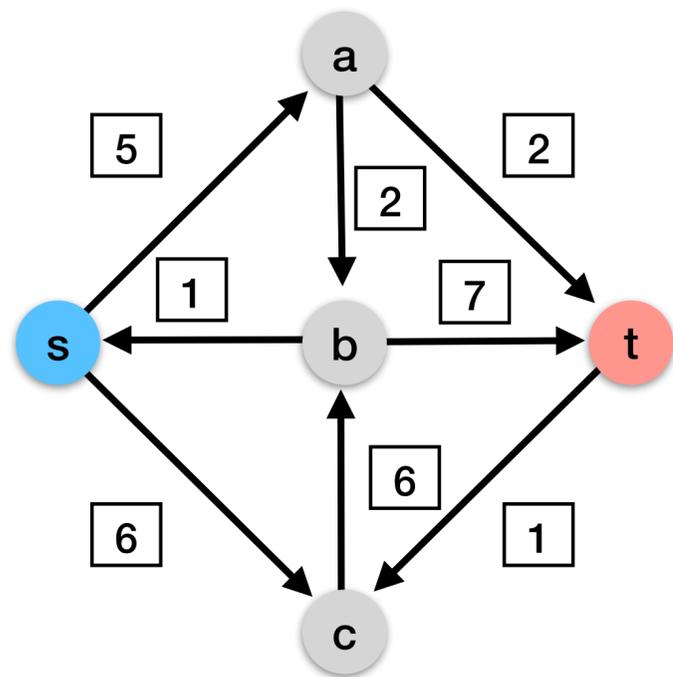
- Wie erhöhen wir den Flusswert?
- Wie lange erhöhen wir?

Flusswerterhöhung



Flusswerterhöhung



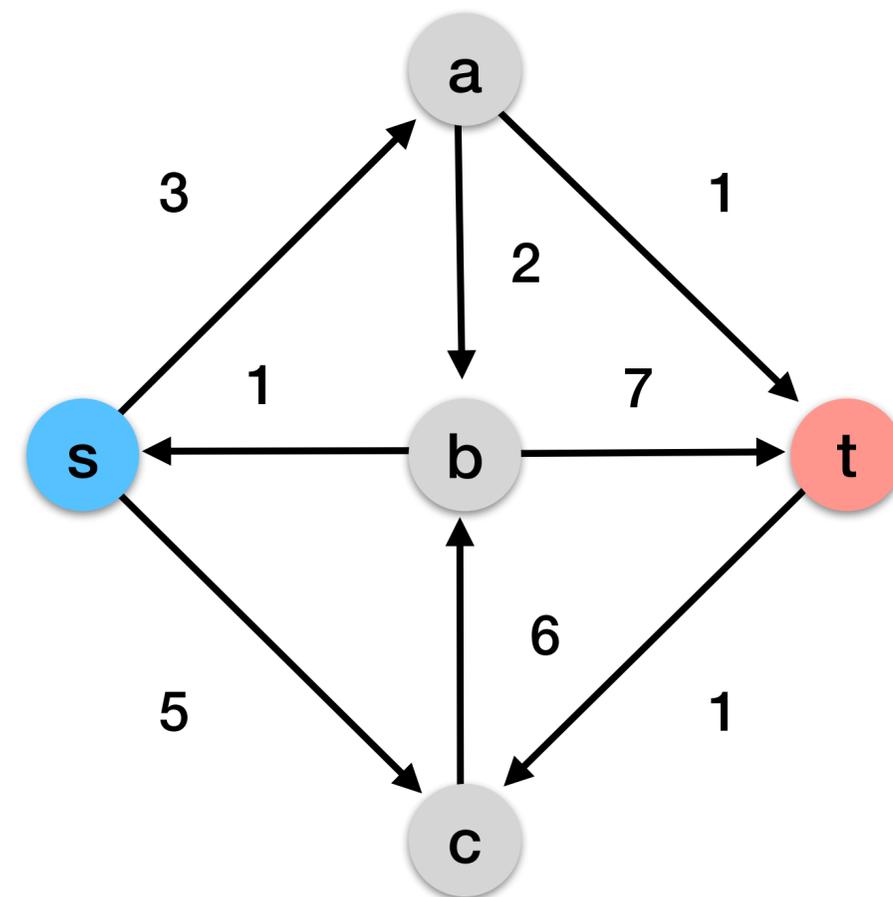


- Quelle (source)
- Senke (sink)
- Kapazität

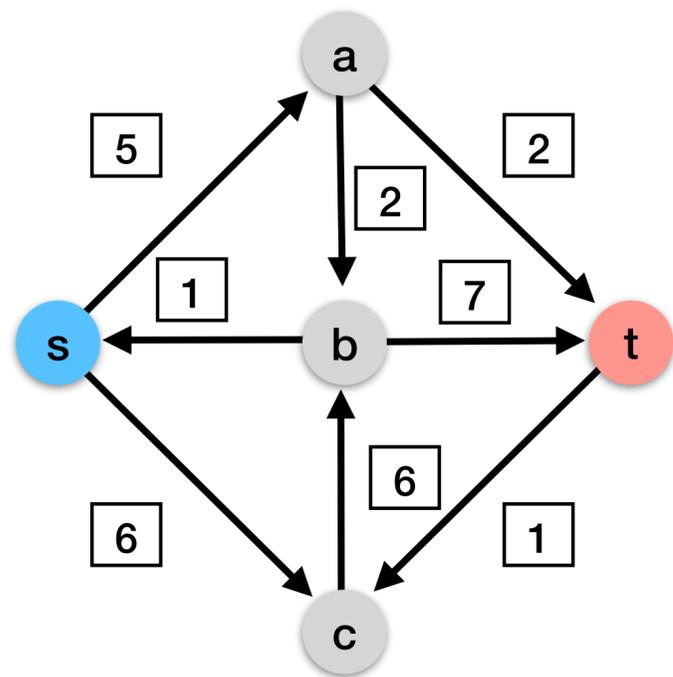
Netzwerk $N = (V, A, c, s, t)$.

Angenommen, wir finden einen **gerichteten Pfad** P von s (Quelle) zu t (Senke), wo der Fluss auf allen Kanten die **Kapazität noch nicht erschöpft** hat, d.h. $f(e) < c(e)$ für alle e auf P .

Flusswerterhöhung



Wir definieren $\delta := \min_{e \in P} c(e) - f(e)$.

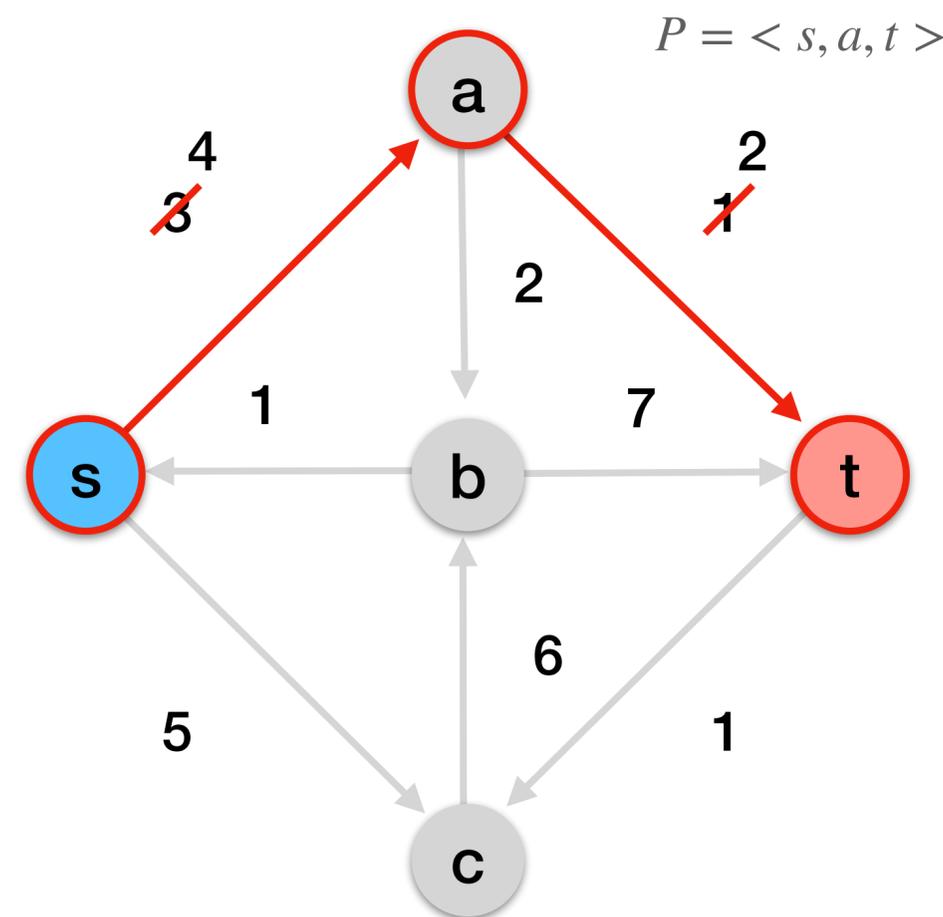


- Quelle (source)
- Senke (sink)
- Kapazität

Netzwerk $N = (V, A, c, s, t)$.

Angenommen, wir finden einen **gerichteten Pfad** P von s (Quelle) zu t (Senke), wo der Fluss auf allen Kanten die **Kapazität noch nicht erschöpft** hat, d.h. $f(e) < c(e)$ für alle e auf P .

Flusswerterhöhung

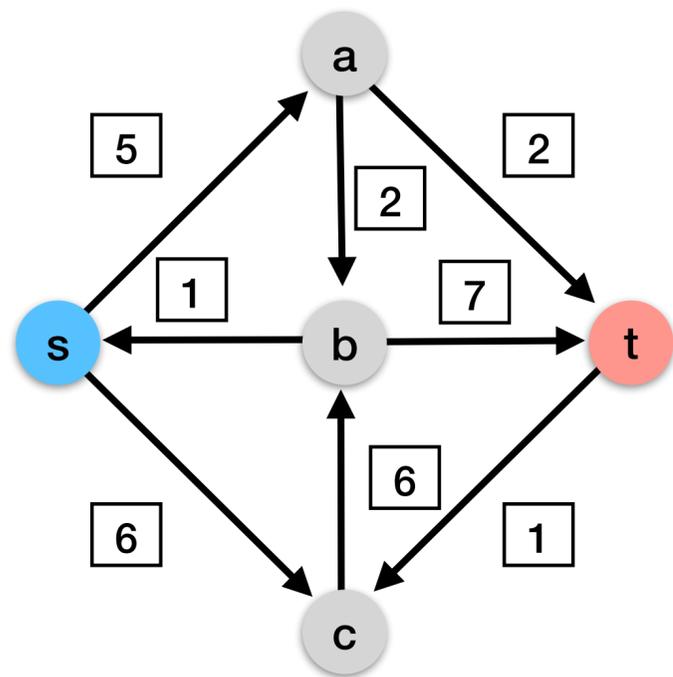


Wir definieren $\delta := \min_{e \in P} c(e) - f(e)$.

Hier: $\delta = 2 - 1 = 1$.

Erhöhe entlang P um $\delta = 1$.

1. Flusseigenschaft wurde nicht verletzt.
 2. Flusswert wurde um δ erhöht.
- Wir haben einen Fluss mit Wert $4 + 5 - 1 = 8$.

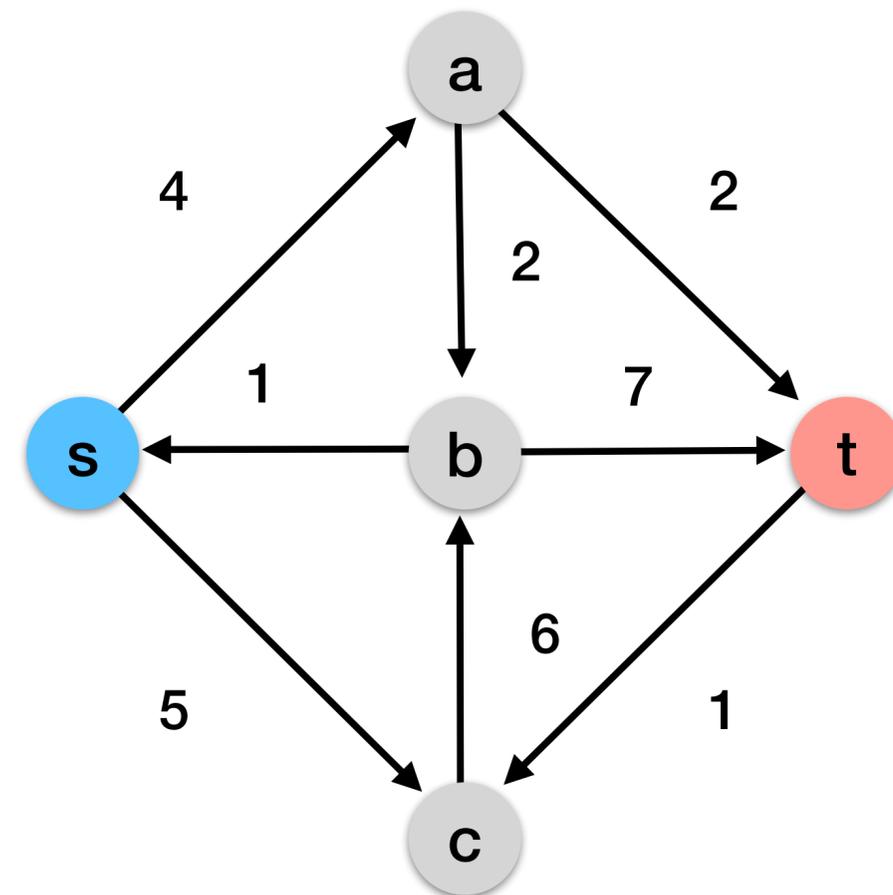


- Quelle (source)
- Senke (sink)
- Kapazität

Netzwerk $N = (V, A, c, s, t)$.

Angenommen, wir finden einen **gerichteten Pfad P** von s (Quelle) zu t (Senke), wo der Fluss auf allen Kanten die **Kapazität noch nicht erschöpft** hat, d.h. $f(e) < c(e)$ für alle e auf P .

Flusserhöhung



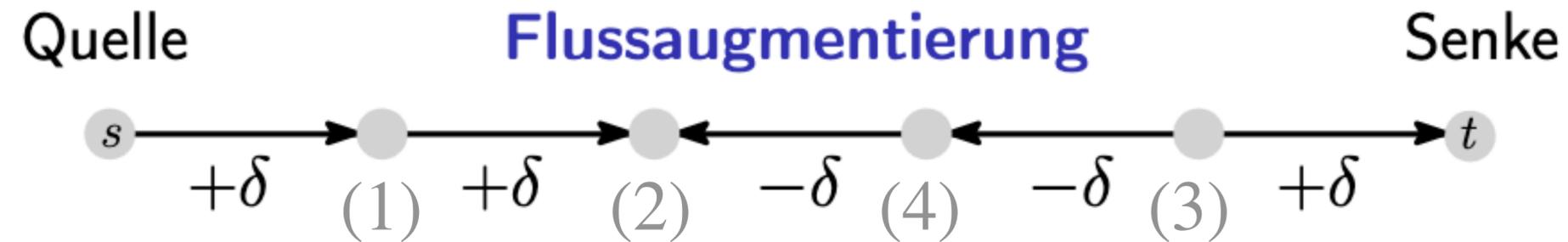
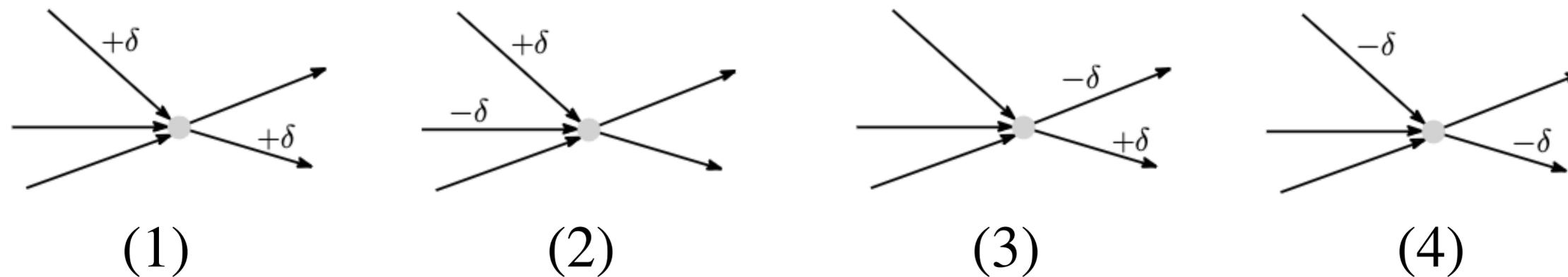
Fluss mit Wert $4 + 5 - 1 = 8$.

Man sieht schnell: es gibt keinen solchen Pfad P mehr.

Aber ist f maximal?

Nein! Warum?

Lokale Veränderungen des Flusses, die die Flusserhaltung erhalten:



augmentierender Pfad (ungerichteter Pfad!)

Erinnerung: augmentieren bedeutet so viel wie steigern, verstärken, erweitern.

Hier wird unser Flusswert gesteigert, denn wir haben 3-mal $+\delta$ und nur 2-mal $-\delta$, also insgesamt $+\delta$.

Restnetzwerk

Für $e = (u, v)$, sei $e^{\text{opp}} := (v, u)$ (entgegen gerichtete Kante).

Sei $N = (V, A, c, s, t)$ ein Netzwerk **ohne entgegen gerichtete Kanten**¹ und sei f ein Fluss in N . Das **Restnetzwerk**

$N_f := (V, A_f, r_f, s, t)$ ist wie folgt definiert:

1. Ist $e \in A$ mit $f(e) < c(e)$, dann ist e eine Kante in A_f , mit

$$r_f(e) := c(e) - f(e).$$

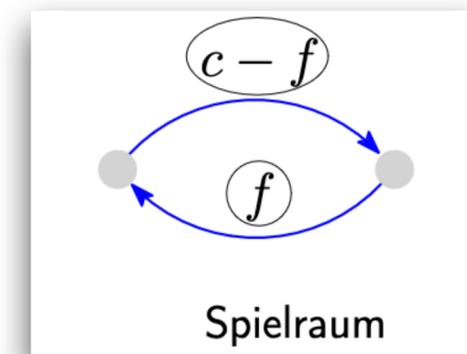
2. Ist $e \in A$ mit $f(e) > 0$, dann ist e^{opp} in A_f , mit

$$r_f(e^{\text{opp}}) = f(e).$$

3. A_f enthält nur Kanten wie in (1) und (2).

$r_f(e)$, $e \in A_f$, nennen wir die **Restkapazität** der Kante e .

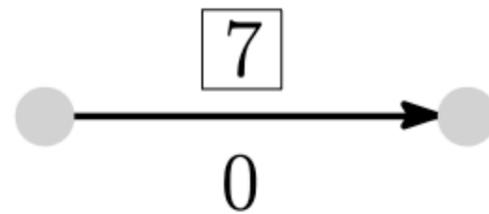
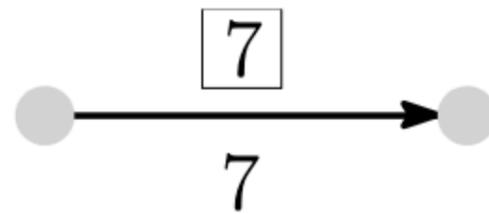
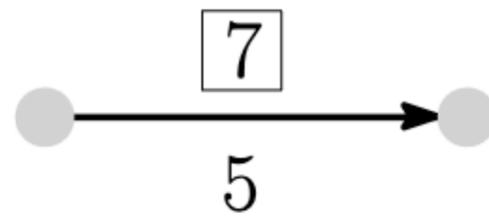
Restkapazität = „Spielraum“



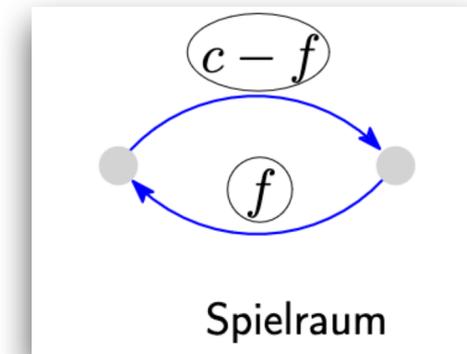
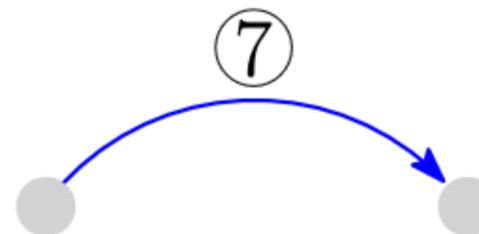
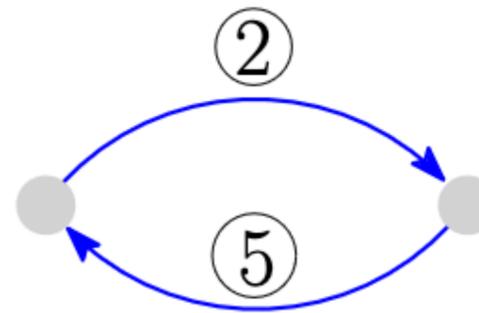
[1]: d.h. zwischen zwei Knoten gibt es nie zugleich die beiden Kanten beider Richtungen. Vereinfachende Annahme, ist aber nicht essentiell

Restnetzwerk

Netzwerk



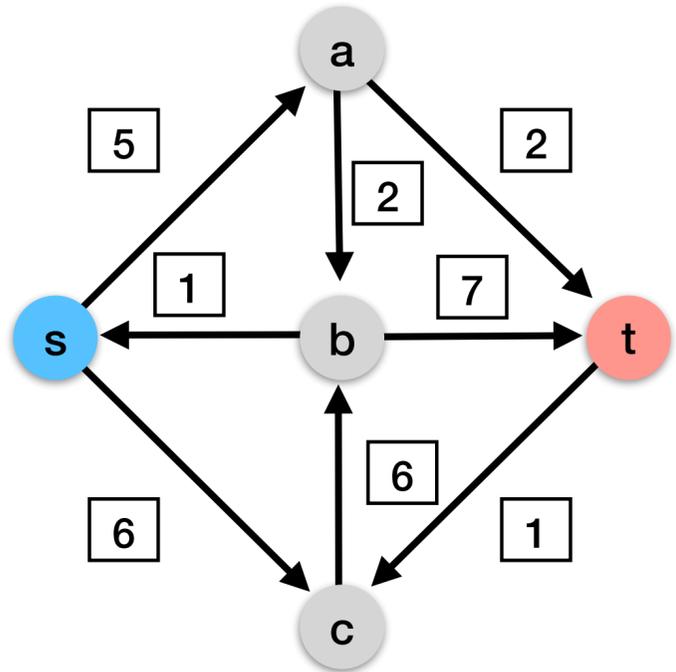
Restnetzwerk
Restkapazität



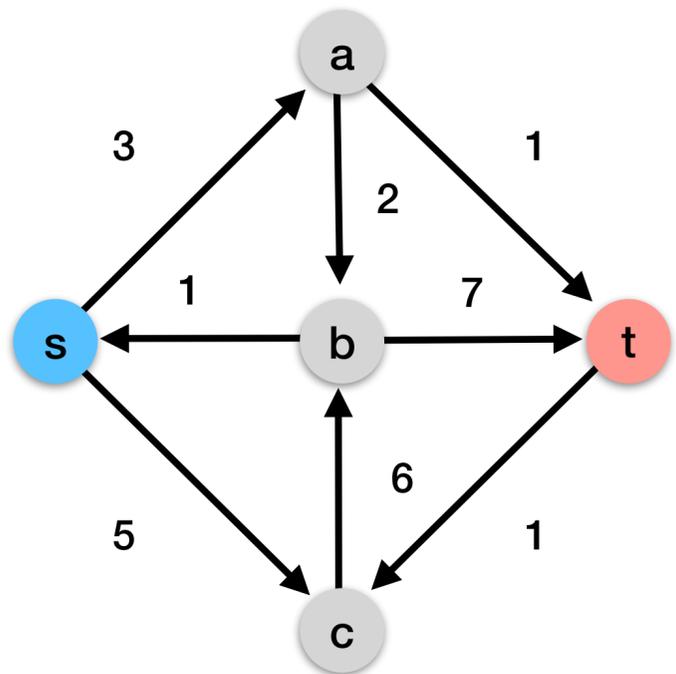
1. Ist $e \in A$ mit $f(e) < c(e)$, dann ist e eine Kante in A_f , mit $r_f(e) := c(e) - f(e)$.
2. Ist $e \in A$ mit $f(e) > 0$, dann ist e^{opp} in A_f , mit $r_f(e^{\text{opp}}) = f(e)$.

Wegen strikt kleiner/grösser, gibt es in den beiden unteren Fällen nur jeweils eine Kante!

- Quelle (source)
- Senke (sink)
- Kapazität

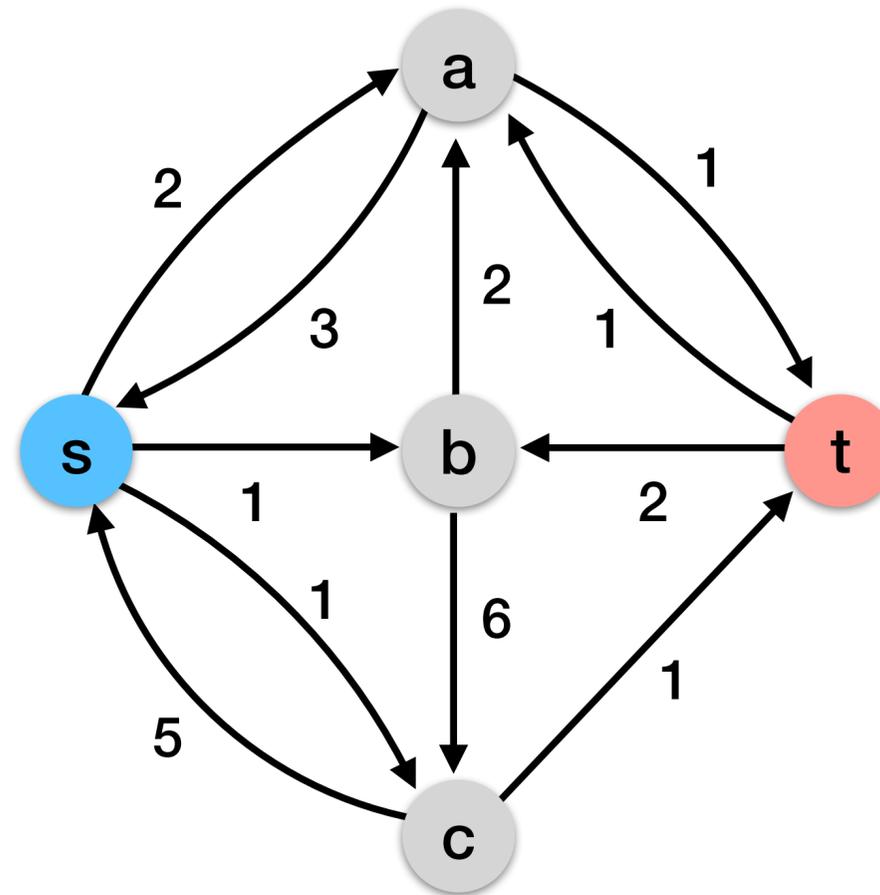
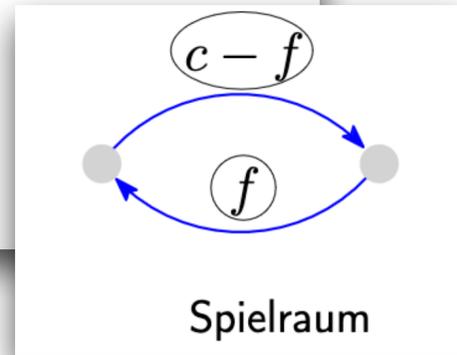


Netzwerk $N = (V, A, c, s, t)$.



Fluss f mit Wert $3 + 5 - 1 = 7$.

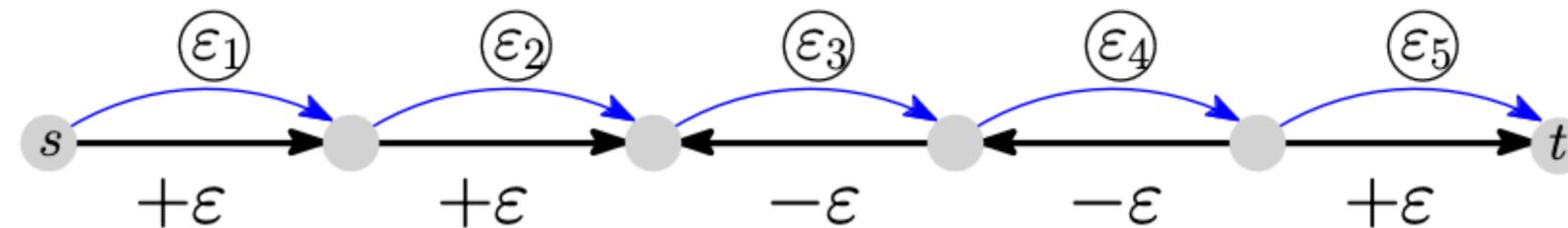
1. Ist $e \in A$ mit $f(e) < c(e)$, dann ist e eine Kante in A_f , mit $r_f(e) := c(e) - f(e)$.
2. Ist $e \in A$ mit $f(e) > 0$, dann ist e^{opp} in A_f , mit $r_f(e^{opp}) = f(e)$.



Restnetzwerk $N_f = (V, A_f, r_f, s, t)$.

Augmentierende Pfade

Wir betrachten einen gerichteten s - t -Pfad in N_f :

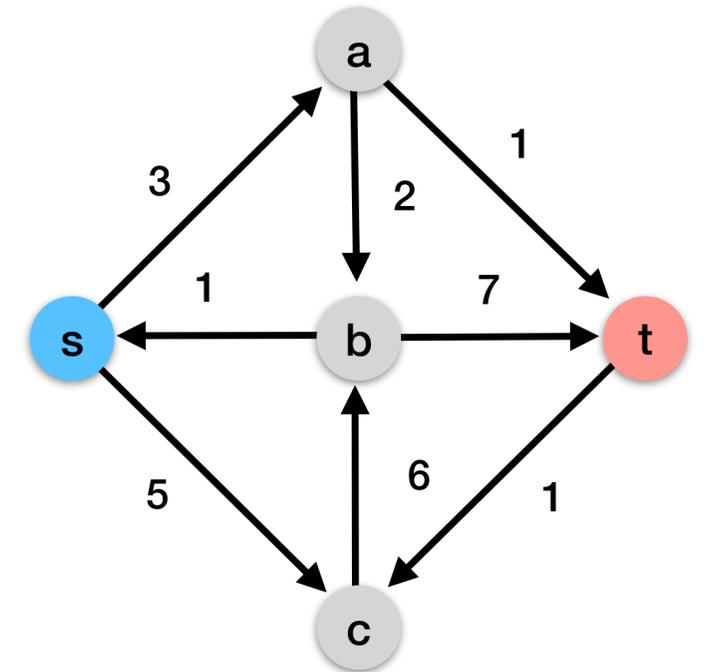
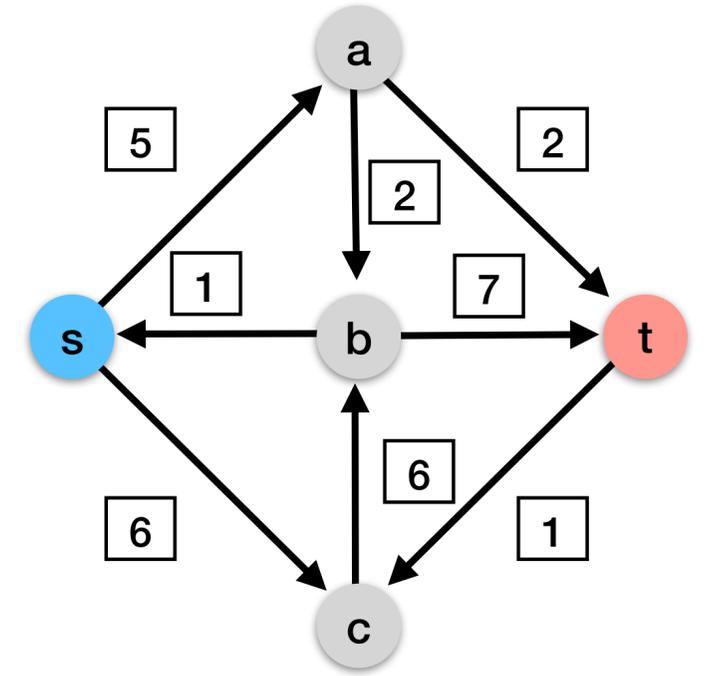
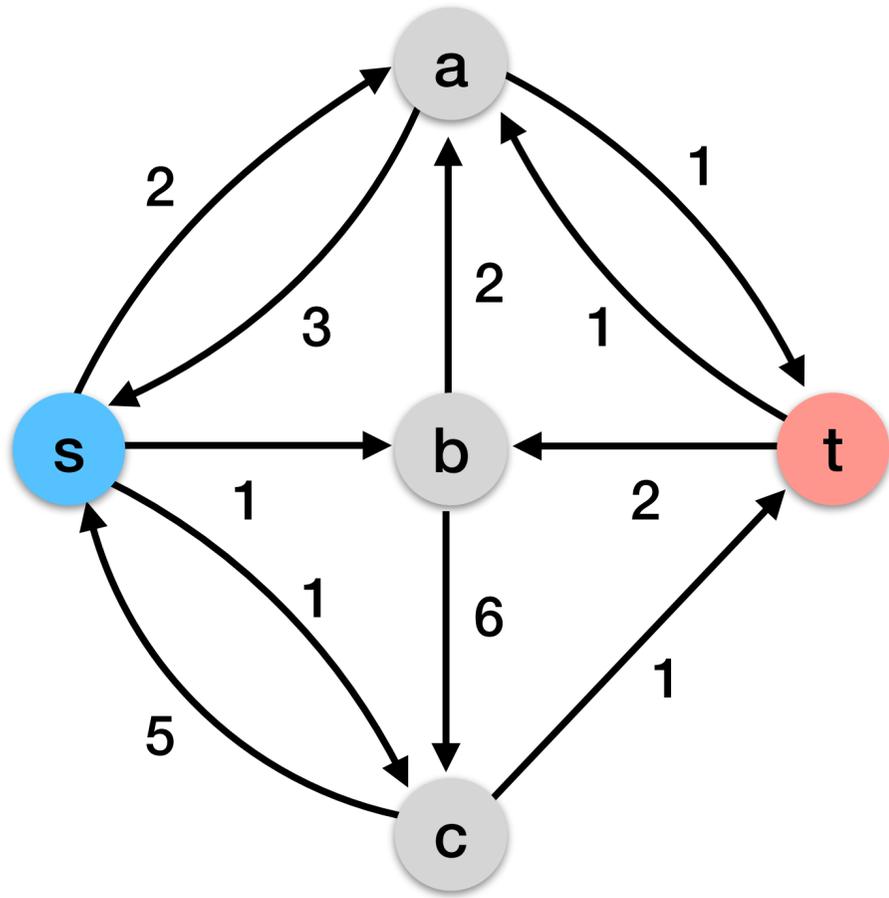


Bestimme die kleinste Restkapazität $\varepsilon := \min_i \varepsilon_i$

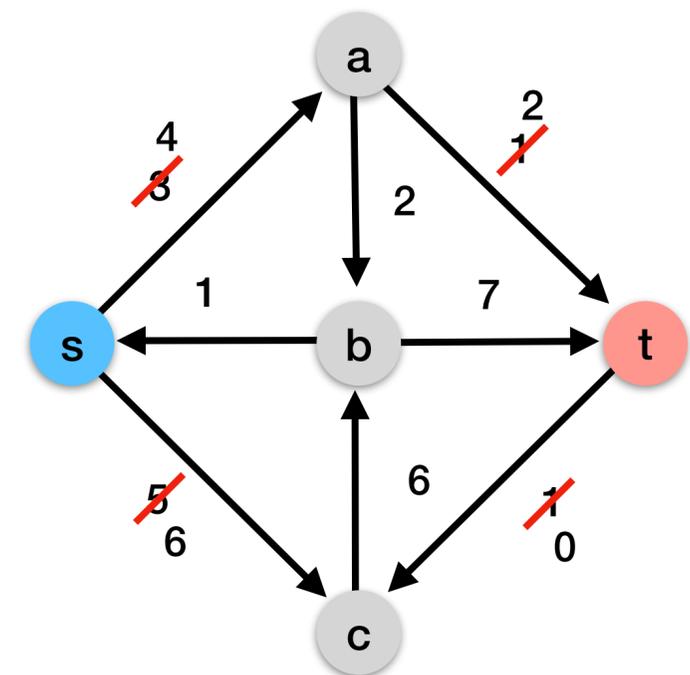
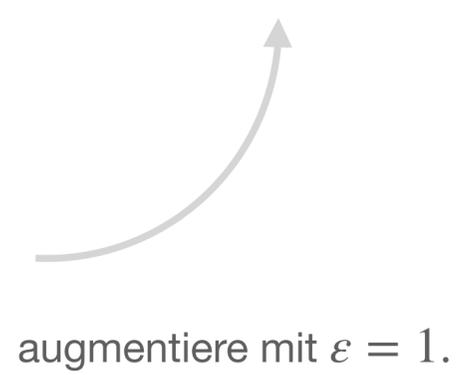
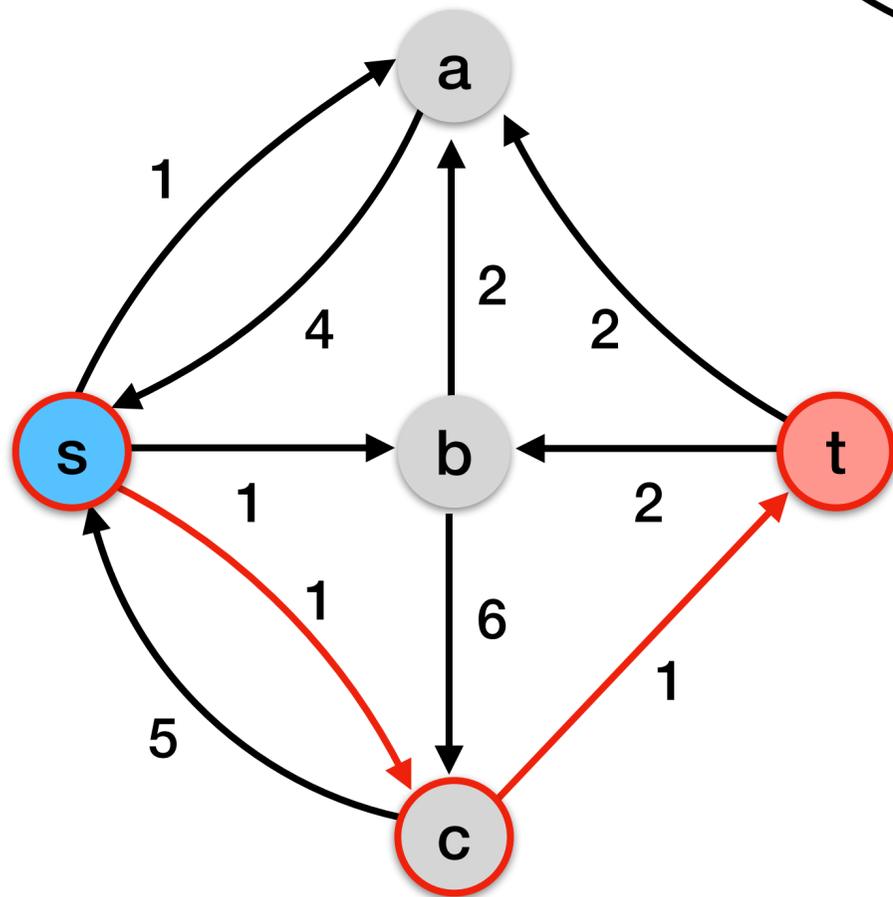
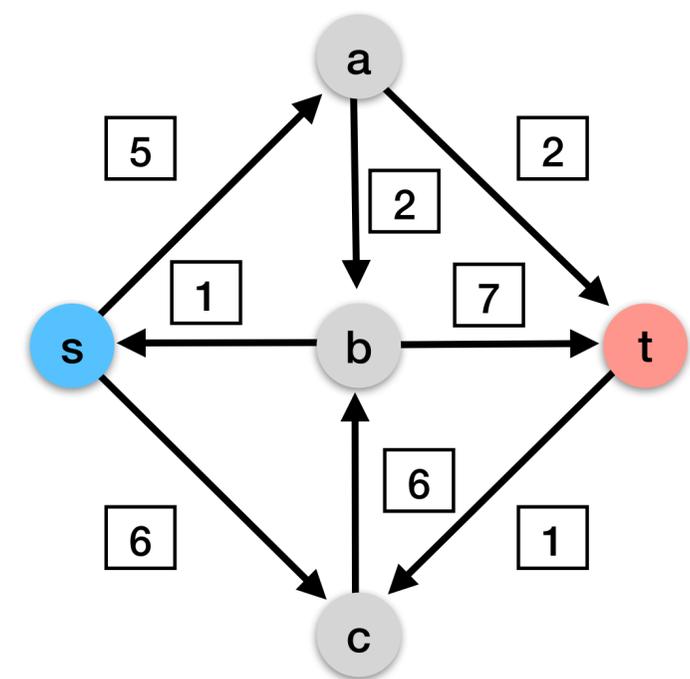
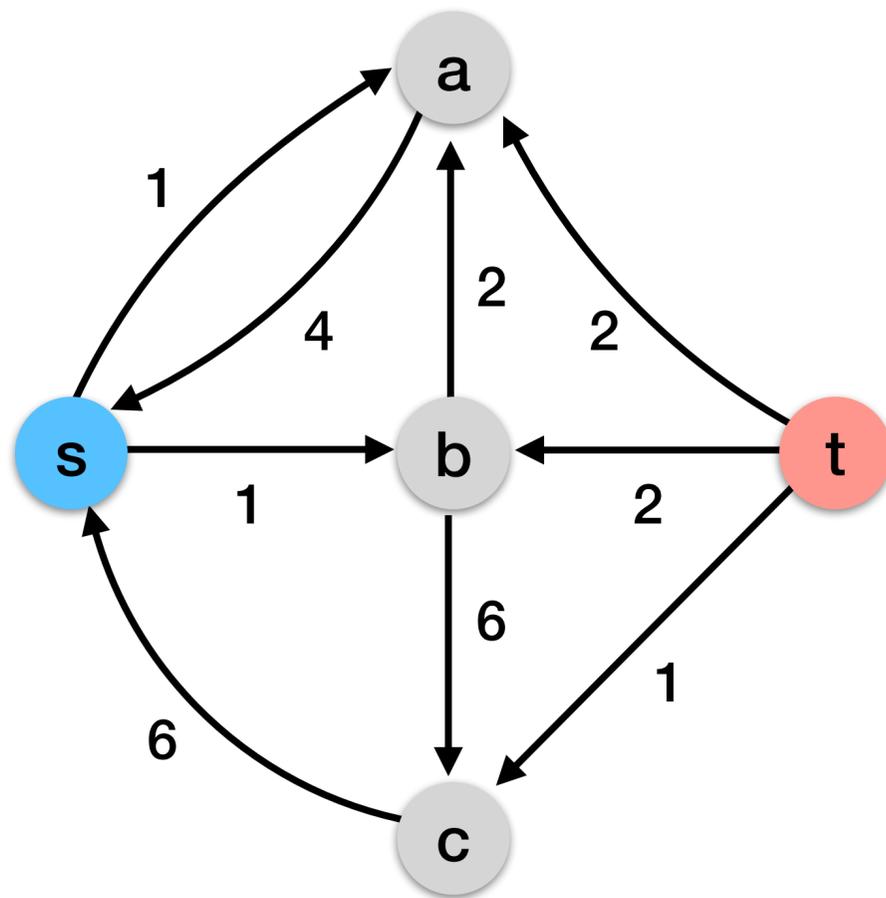
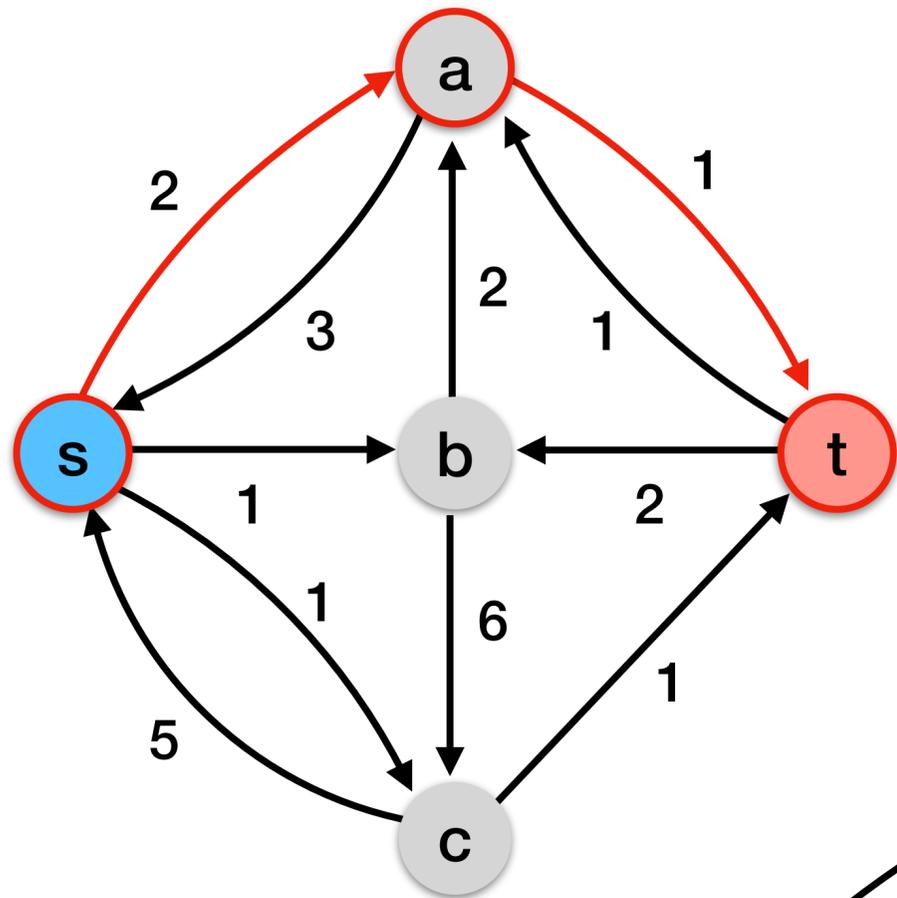
Augmentiere f entlang des Pfades um ε .

Der blaue Pfad ist ein gerichteter Pfad im Restnetzwerk N_f . Die drunterliegenden Kanten, sind die Kanten des Netzwerkes.

1. Zeigt die blaue Kante in die selbe Richtung zeigt, dann gehen wir entlang der Restkapazität, können also noch erhöhen.
2. Zeigt die blaue Kante in die entgegengesetzte Richtung, dann gehen wir entlang des Flusses und reduzieren.



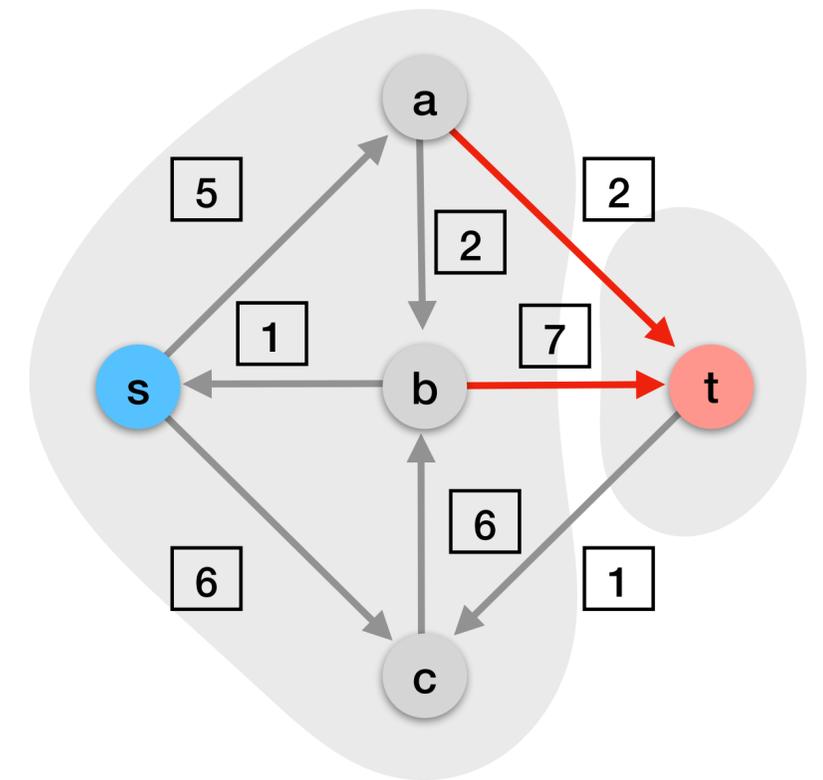
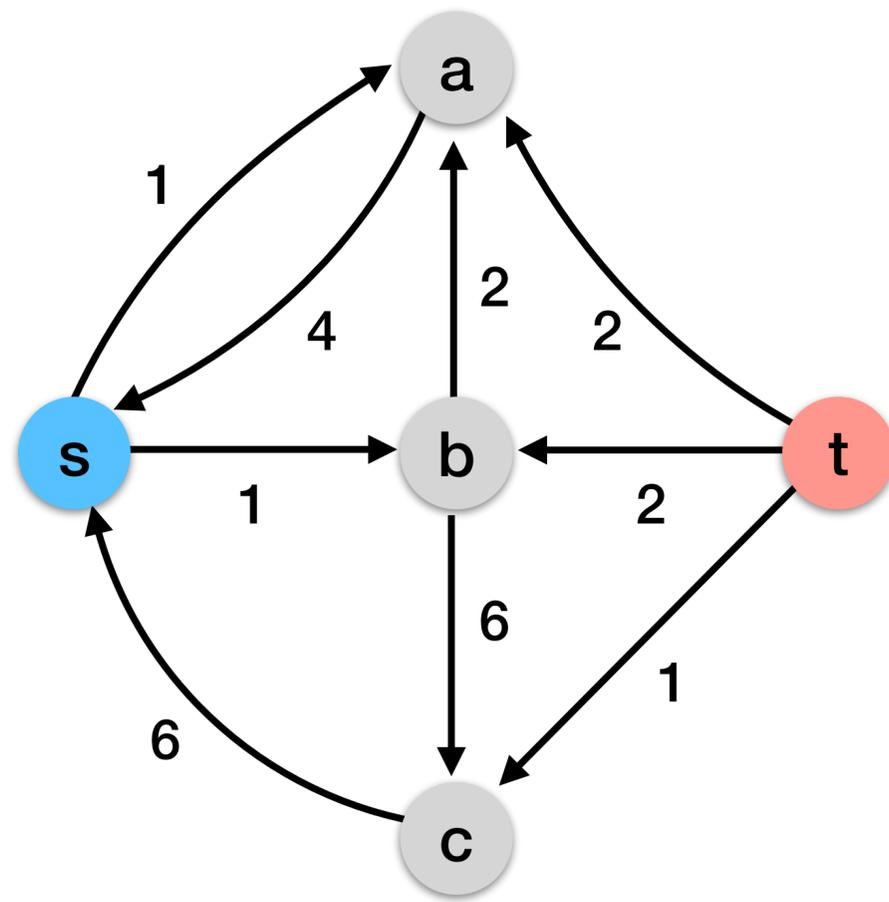
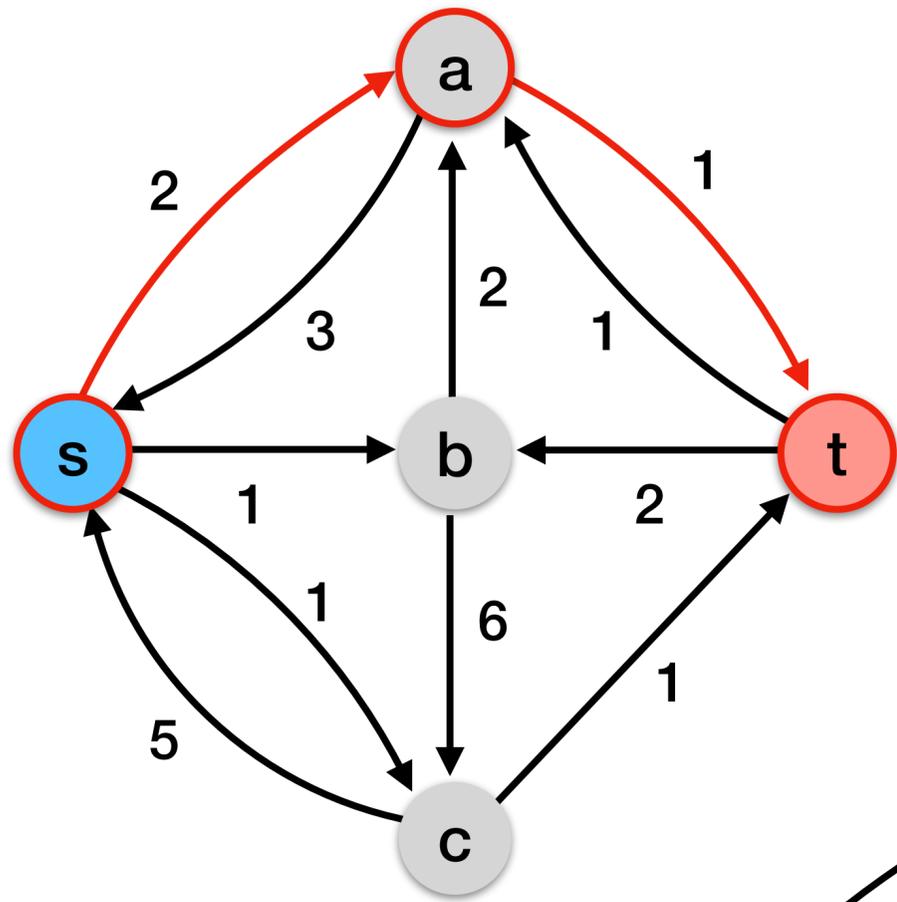
Fluss f mit Wert 7.



augmentiere mit $\varepsilon = 1$.

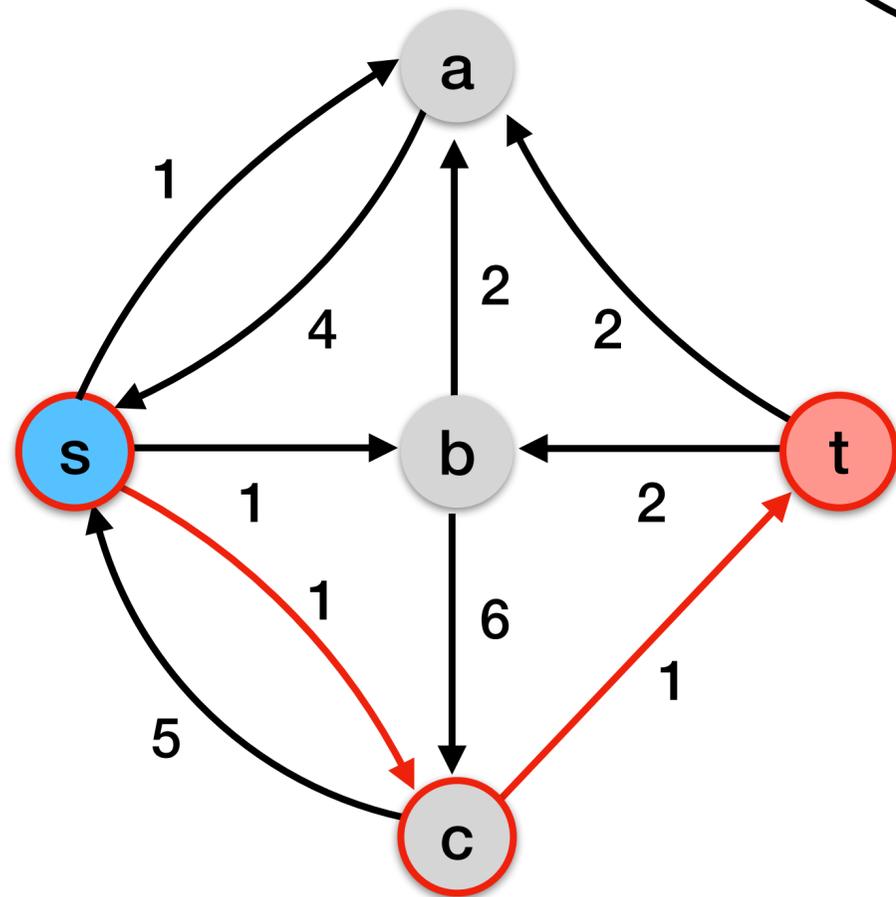
augmentiere mit $\varepsilon = 1$.

Fluss f mit Wert ~~9~~
~~8~~
~~7~~.



Schnitt mit $\text{cap}(S, T) = 2 + 7 = 9$.

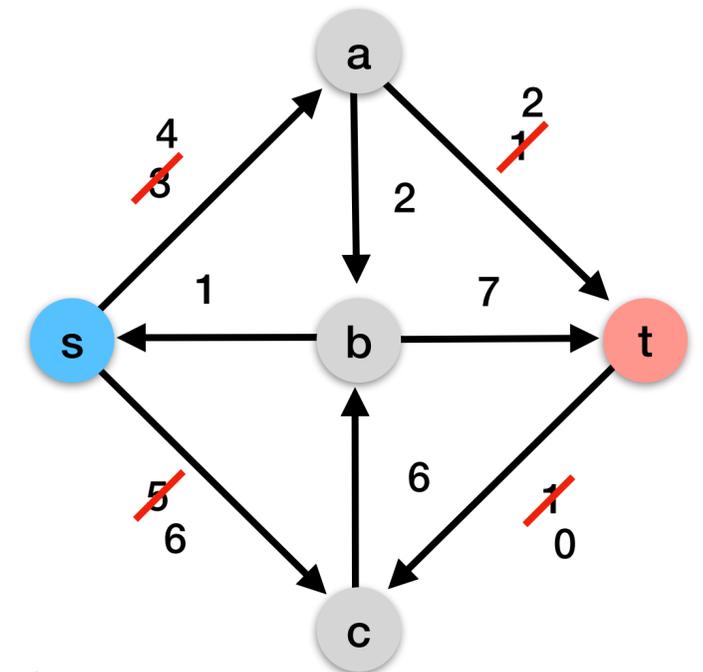
augmentiere mit $\varepsilon = 1$.



augmentiere mit $\varepsilon = 1$.

Fluss f mit Wert ~~8~~ ⁹.

Nach dem Maxflow-Mincut Theorem ist f maximal!



Satz

Sei N ein Netzwerk (ohne entgegen gerichtete Kanten).

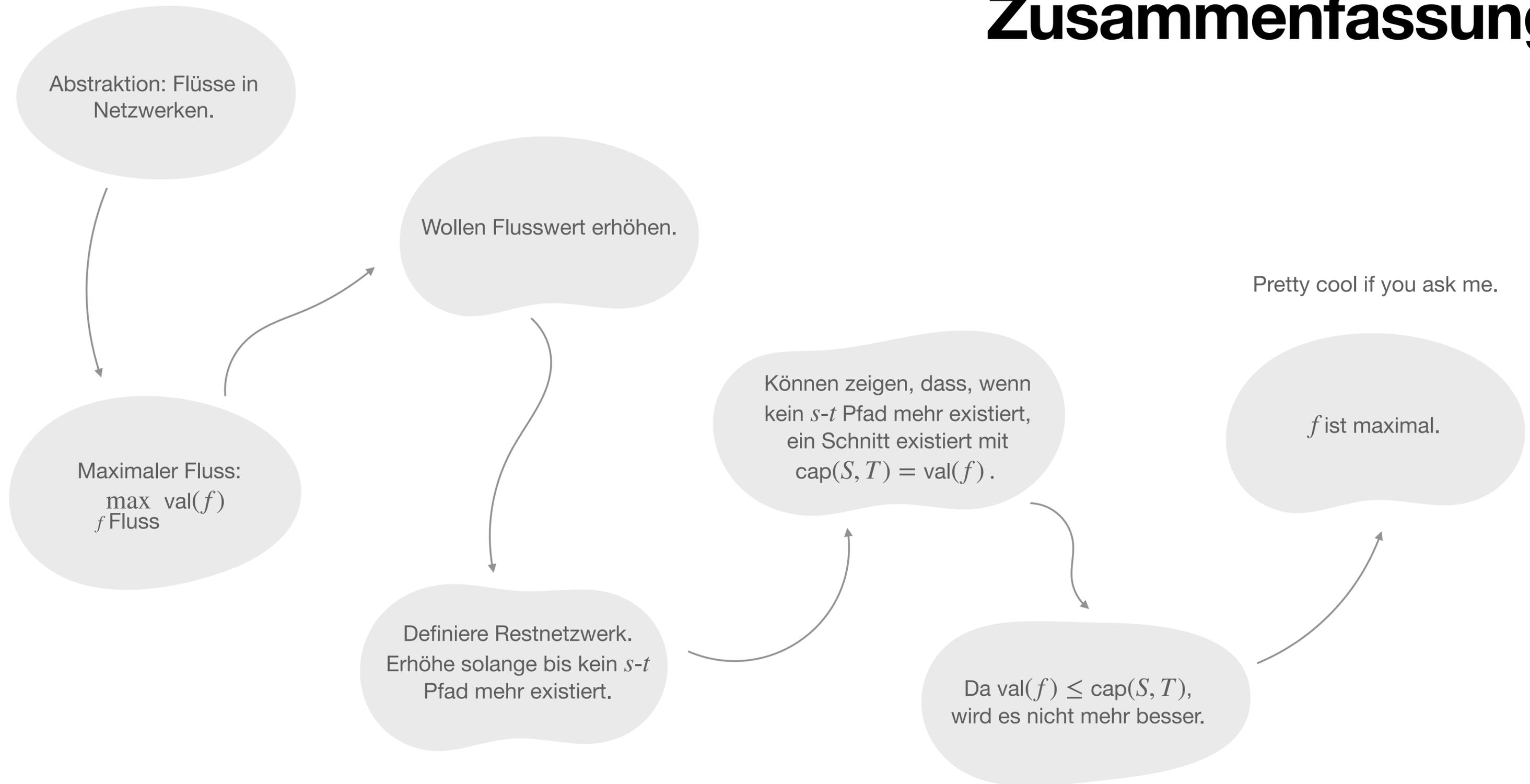
Ein Fluss f ist maximaler Fluss

*\Leftrightarrow
es im Restnetzwerk N_f keinen gerichteten s - t -Pfad gibt.*

Für jeden maximalen Fluss f

gibt es einen s - t -Schnitt (S, T) mit $\text{val}(f) = \text{cap}(S, T)$.

Zusammenfassung



Ford-Fulkerson

Gegeben: Ein Netzwerk $N = (V, A, c, s, t)$.

Gesucht: Ein maximaler Fluss f .

Ford-Fulkerson(V, A, c, s, t)

1: $f \leftarrow \mathbf{0}$

▷ Fluss konstant 0

2: **while** \exists s - t -Pfad P in N_f **do**

▷ augmentierender Pfad

3: Augmentiere den Fluss entlang P

4: **return** f

▷ maximaler Fluss

Sei $n := |V|$ und $m := |A|$ für Netzwerk $N = (V, A, c, s, t)$.

▶ Angenommen $c: A \rightarrow \mathbb{N}_0$ und $U := \max_{e \in A} c(e)$. Dann gilt
$$\text{val}(f) \leq \text{cap}(\{s\}, V \setminus \{s\}) \leq (n - 1)U$$

und es gibt **höchstens $(n - 1)U$ Augmentierungsschritte**.

▶ **Ein Augmentierungsschritt**

Suche s - t -Pfad in N_f , Augmentieren, Aktualisierung von N_f

benötigt $O(m)$ Zeit.

Satz (Ford-Fulkerson mit ganzzahligen Kapazitäten)

Sei $N = (V, A, c, s, t)$ ein Netzwerk mit $c: A \rightarrow \mathbb{N}_0^{\leq U}$, $U \in \mathbb{N}$, ohne entgegen gerichtete Kanten.² Dann gibt es einen ganzzahligen maximalen Fluss. Er kann in Zeit $O(mnU)$ berechnet werden.

1. Das Restnetzwerk wird nicht in jedem Schritt neu konstruiert, sondern schrittweise entlang des gewählten augmentierenden Pfades verändert.
2. In jedem Schritt erhöhen wir in diesem Fall den Wert des Flusses um einen ganzzahligen Wert (≥ 1). Und das gerade weil die Kapazitäten ganzzahlig sind! Bei irrationalen Kapazitäten terminiert der Algorithmus nicht immer.

Zusammenfassung

Der **Ford-Fulkerson Algorithmus** zeigt, dass es unter den gegebenen Umständen (Ganzzahligkeit, keine entgegen gerichtete Kanten) einen maximalen Fluss gibt.

Satz („Maxflow-Mincut Theorem“, ganzzahlig)

Jedes Netzwerk ohne entgegen gerichtete Kanten mit ganzzahligen Kapazitäten erfüllt

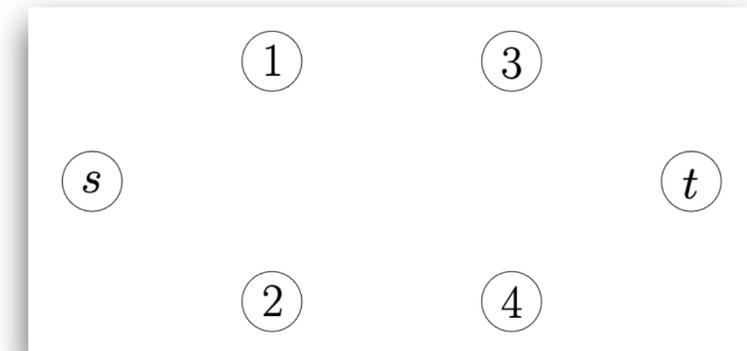
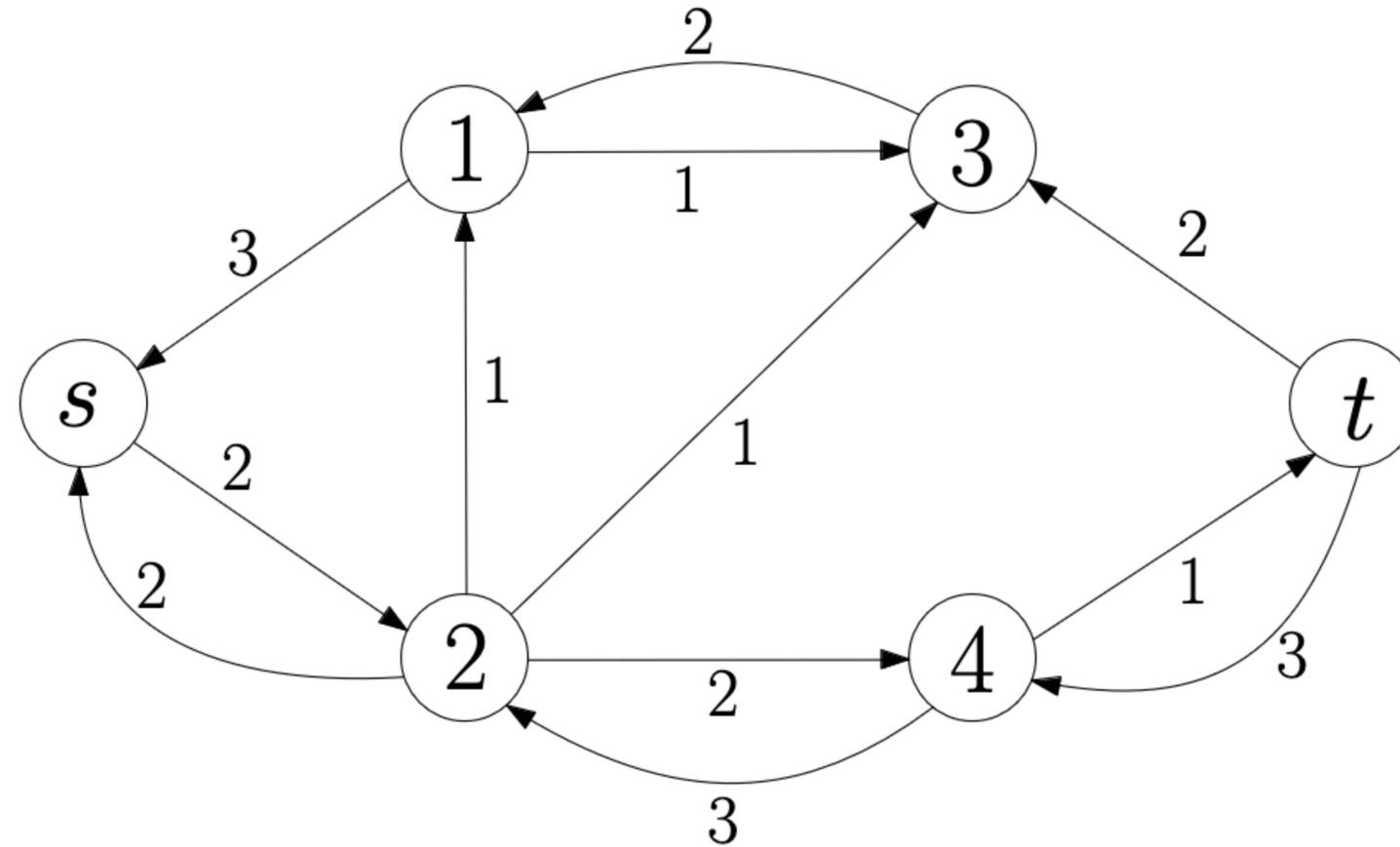
$$\max_{f \text{ Fluss}} \text{val}(f) = \min_{(S, T) \text{ s-t-Schnitt}} \text{cap}(S, T) .$$

Empfehlungen

- Beweis Lemma 3.6.
- Beweis Lemma 3.8.
- Beweis Satz 3.11.

Aufgabe 4 – Restnetzwerk

Sei N ein Netzwerk ohne entgegengesetzte Kanten und sei f ein Fluss in G . Unten abgebildet sehen Sie das Restnetzwerk R_f .



This template was given.

(a) Ist f maximal? Geben Sie eine kurze Begründung für Ihre Antwort.

(1 Punkte)

(b) Rekonstruieren sie N und f .

(4 Punkte)

AlgoWahr FS21.

cont'd on iPad, please see notes.