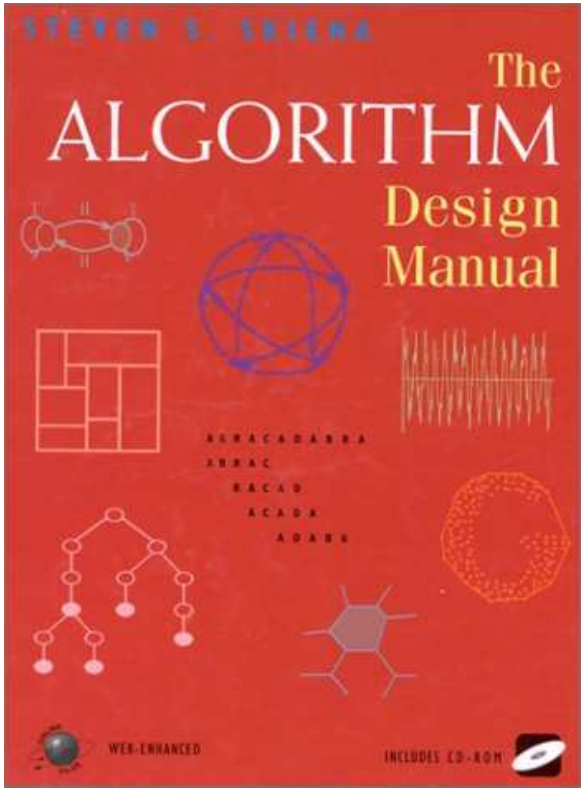
Literatur

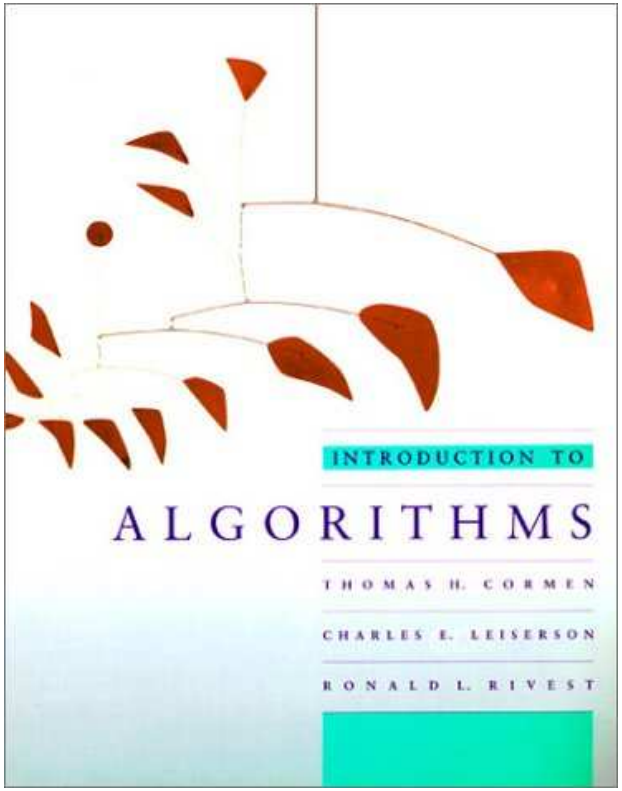


Sehr empfehlenswertes Buch. Es enthält einen Überblick über die wichtigsten Entwurfstechniken für Algorithmen und einen sehr vollständigen Überblick über bekannte Algorithmen mit Verweisen.

Steven S. Skiena: *The Algorithm Design Manual.* Springer Verlag.

Preis: US $69.95, Hardcover mit CD-ROM.

Literatur



Sehr umfangreiches, modernes Buch.
Sehr gut geschrieben.
Enthält alle wichtigen Algorithmen.
Nur ein Buch $\Rightarrow$ dieses!

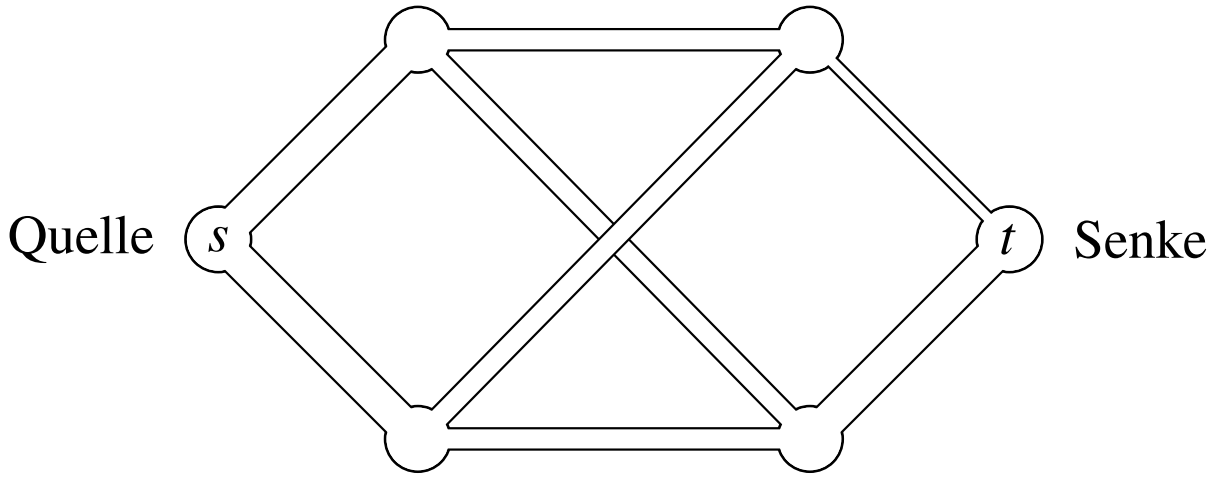
Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest:
Introduction To Algorithms. MIT Press.

Preis: US $69.95, Hardcover, 1028 Seiten!

Netzwerkfluß

(Wiederholung der Grundlagen aus DSAL)

Gegeben ist ein System von Wasserrohren:

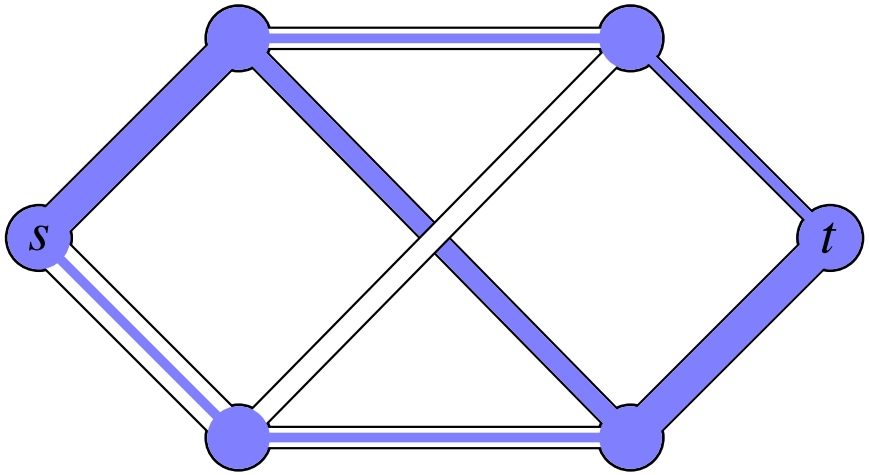


Die Kapazität jedes Rohres ist 3, 5 oder $8\,l/s$.

Frage: Wieviel Wasser kann von der Quelle zur Senke fließen?

Netzwerkfluß

Antwort: Maximal $11\,l/s$ sind möglich.



s-t-Netzwerke

Ein *s-t-Netzwerk* (flow network) ist ein gerichteter Graph $G = (V, E)$, wobei

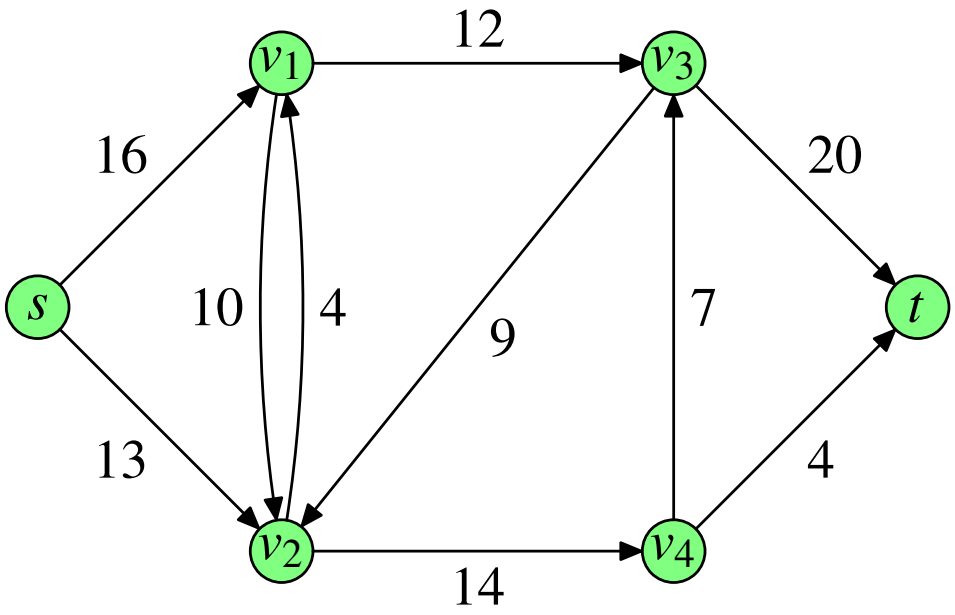
1. jede Kante $(u, v) \in E$ eine *Kapazität* $c(u, v) \geq 0$ hat,
2. es eine *Quelle* $s \in V$ und eine *Senke* $t \in V$ gibt.

Es ist bequem, anzunehmen daß jeder Knoten auf einem Pfad von s nach t liegt.

Falls $(u, v) \notin E$ setzen wir $c(u, v) = 0$.

Es kann Kanten (u, v) und (v, u) mit verschiedener Kapazität geben.

Beispiel eines s-t-Netzwerks



Die Kanten sind mit den Kapazitäten $c(u, v)$ beschriftet.

Flüsse

Ein *Fluß* ist eine Funktion $f : V \times V \to \mathbf{R}$, die Paare von Knoten auf reelle Zahlen abbildet und diese Bedingungen erfüllt:

- *Zulässigkeit:* Für $u, v \in V$ gilt $f(u, v) \leq c(u, v)$.
- *Symmetrie:* Für $u, v \in V$ gilt $f(u, v) = -f(v, u)$.
- *Flußerhaltung:* Für $u \in V - \{s, t\}$ gilt $\sum_{v \in V} f(u, v) = 0$.

Der *Wert* $|f|$ eines Flusses ist definiert als $|f| = \sum_{u \in V} f(s, u)$.

Dies ist gerade der Gesamtfluß aus der Quelle heraus.

Maximale Flüsse

Das Problem des *maximalen Flusses*:

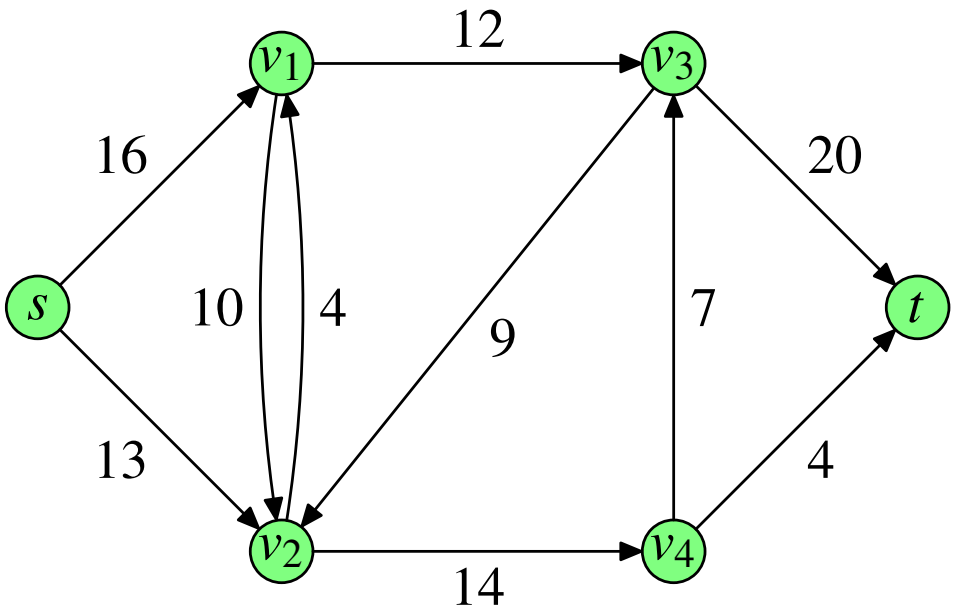
Gegeben:

Ein s-t-Netzwerk.

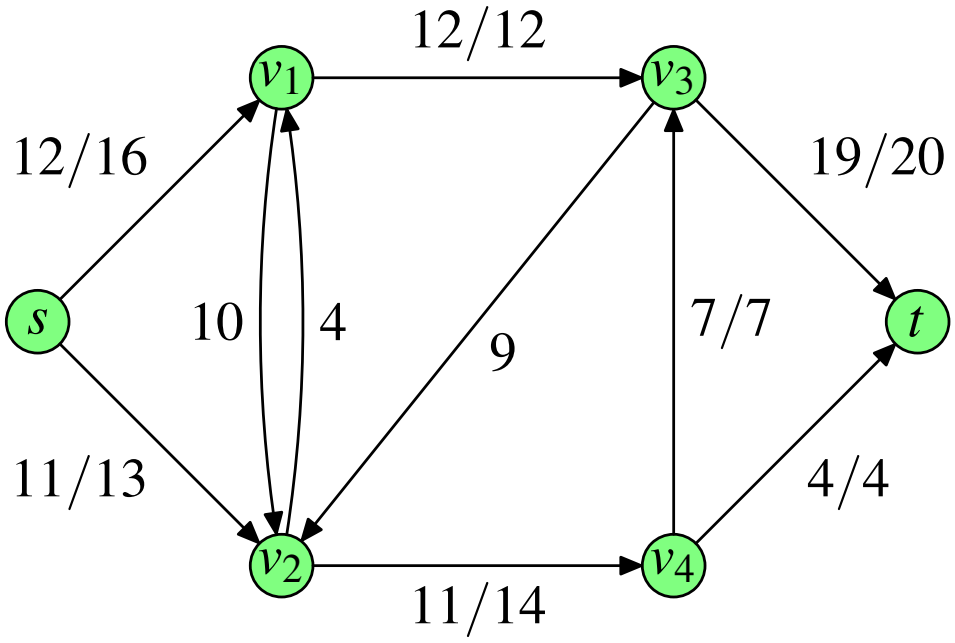
Gesucht:

Ein Fluß mit maximalem Wert.

Was ist der maximale Fluß?

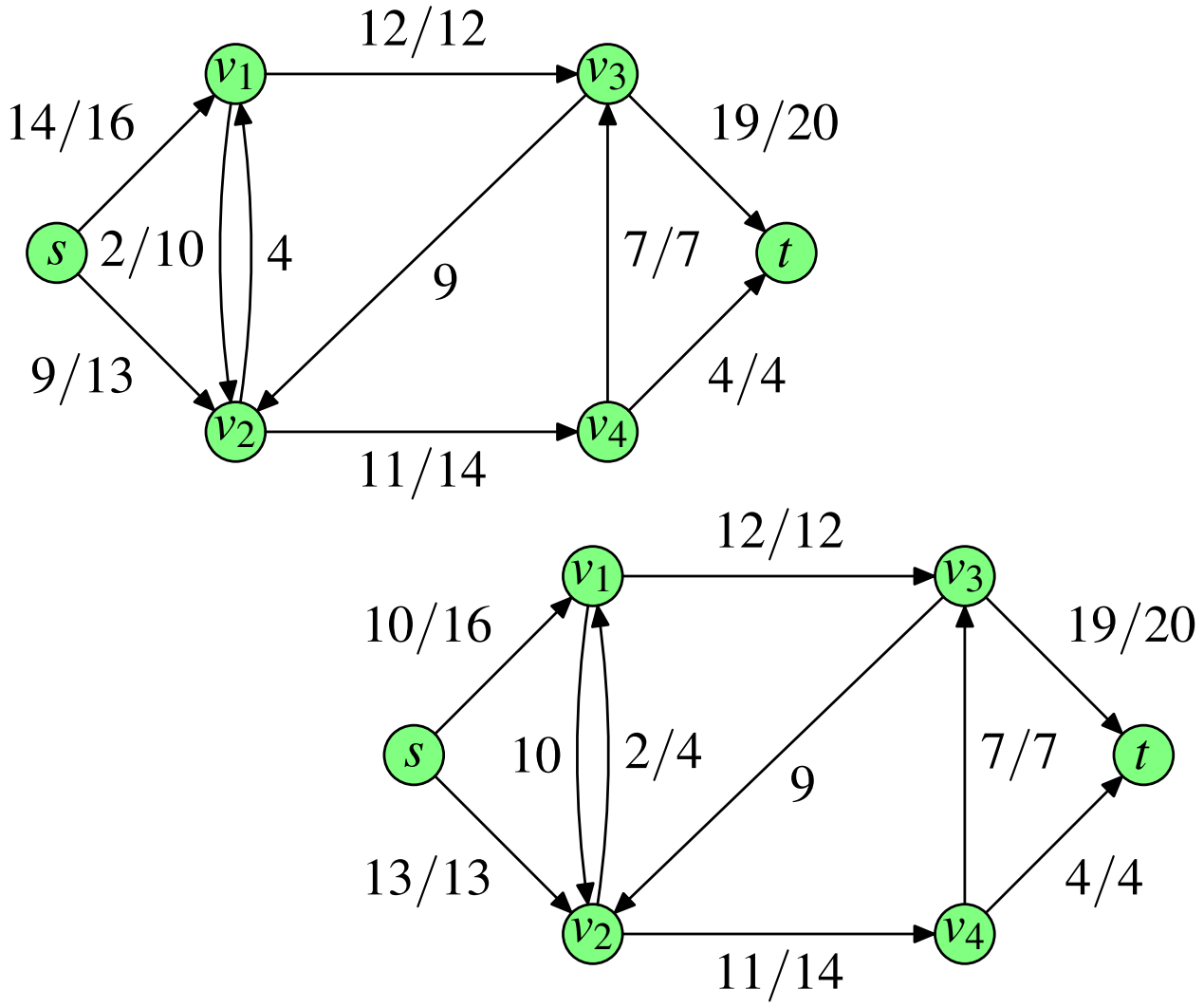


Der maximale Fluß ist 23

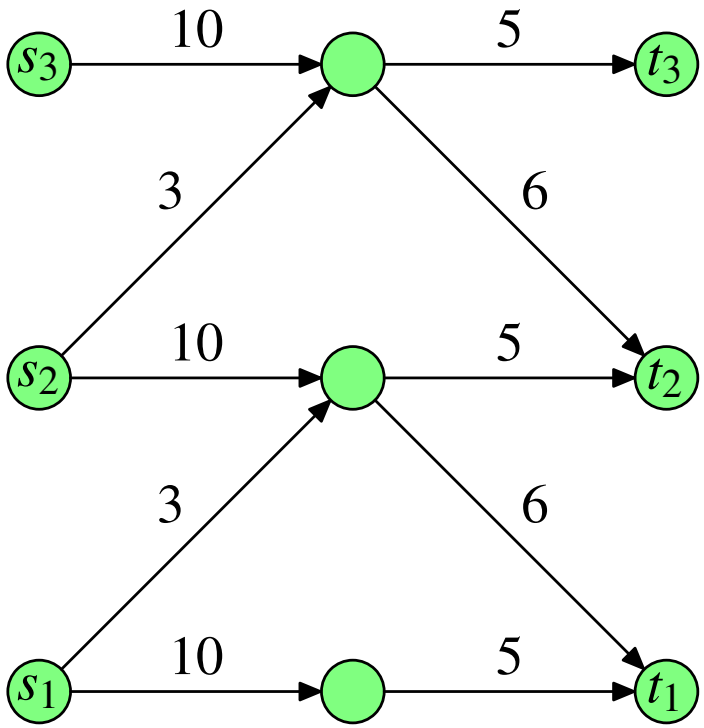


Die Kanten sind mit $c(u, v)$ beschriftet oder mit $f(u, v)/c(u, v)$, falls $f(u, v) > 0$.

Andere optimale Lösungen

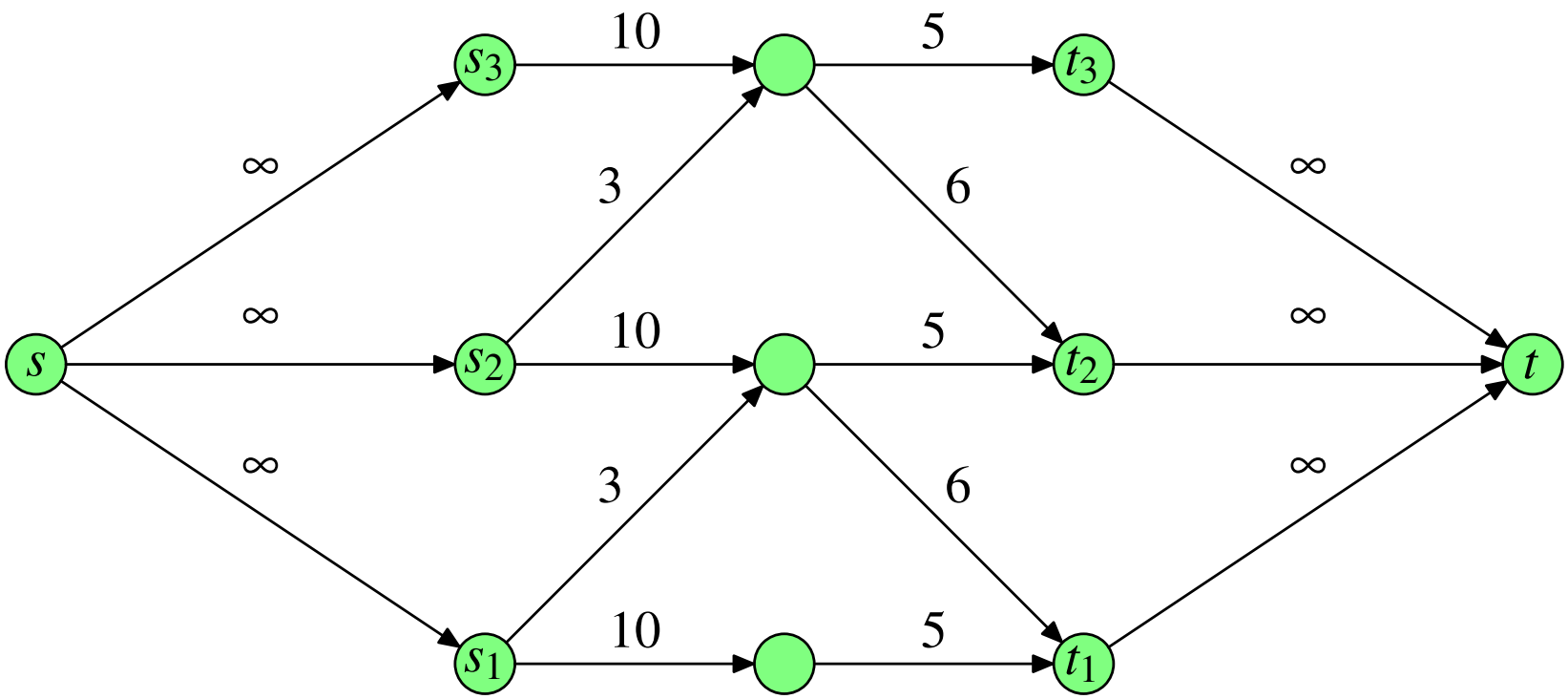


Mehrfache Quellen oder Senken



Mehrfache Quellen oder Senken sind eine **Verallgemeinerung** des Problem des maximalen Flusses.

Mehrfache Quellen oder Senken



→ Neue „Superquelle“ und „Supersenke“ hinzufügen

Existenz des maximalen Flusses

Existiert stets der maximale Fluß

$$\max\{\, |f| \mid f \text{ ist ein } s\text{-}t\text{-Fluß in } G \,\}?$$

Ja, denn die Menge aller Flüsse ist abgeschlossen im $\mathbf{R}^m$ und sie ist nicht leer.

Die stetige Funktion, die einen Fluß auf ihren Wert abbildet, hat daher ein Maximum:

$$|f| = \sum_{u \in V} f(s,u) \text{ ist stetig!}$$

Einige Notationen

Es ist bequem einige Abkürzungen zu verwenden:

- $f(X,Y) = \sum_{x\in X}\sum_{y\in Y} f(x,y)$ für $X, Y \subseteq V$
- $f(x,Y) = \sum_{y\in Y} f(x,y)$ für $Y \subseteq V$
- $f(X,y) = \sum_{x\in X} f(x,y)$ für $X \subseteq V$
- $X - y$ statt $X - \{y\}$

Lemma A

Falls f ein s-t-Fluß für $G = (V, E)$ ist, dann gilt:

1. $f(X, X) = 0$ für $X \subseteq V$
2. $f(X, Y) = -f(Y, X)$ für $X, Y \subseteq V$
3. $f(X \cup Y, Z) = f(X, Z) + f(Y, Z)$ für $X, Y, Z \subseteq V$ mit $X \cap Y = \emptyset$
4. $f(Z, X \cup Y) = f(Z, X) + f(Z, Y)$ für $X, Y, Z \subseteq V$ mit $X \cap Y = \emptyset$

Dieses Lemma ist sehr nützlich, um wichtige Eigenschaften über Flüsse abzuleiten.

Lemma A

Um Lemma A zu beweisen, dürfen wir nur die Eigenschaften eines s-t-Flusses verwenden, also Zulässigkeit, Symmetrie und Flußerhaltung.

Beweis für $f(X,X) = 0$:

$$\begin{aligned} f(X,X) &= \frac{1}{2}\Big(\sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1,x_2) + \sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1,x_2) \Big) \\ &= \frac{1}{2}\Big(\sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1,x_2) + \sum_{x_1 \in X} \sum_{x_2 \in X} f(x_2,x_1) \Big) \\ &= \frac{1}{2} \sum_{x_1 \in X} \sum_{x_2 \in X} \Big(f(x_1,x_2) + f(x_2,x_1) \Big) = 0 \end{aligned}$$

Hier genügt die Symmetrie allein! (Rest als Übungsaufgabe.)

Anwendung des Lemmas

Der Fluß in die Senke sollte intuitiv dem Fluß aus der Quelle entsprechen:

$$f(s, V) = f(V, t)$$

Beweis mit Lemma A:

$$\begin{aligned} f(s,V) &= f(V,V) - f(V-s,V) \\ &= -f(V-s,V) \\ &= f(V,V-s) \\ &= f(V,t) + f(V,V-s-t) \\ &= f(V,t) \text{ wegen Flußerhaltung} \end{aligned}$$

Residualnetzwerke

„Netzwerk minus Fluß = Residualnetzwerk“

Definition:

Gegeben ist ein Netzwerk $G = (V, E)$ und ein Fluß f. Das *Residualnetzwerk* $G_f = (V, E_f)$ zu G und f ist definiert vermöge

$$E_f = \{\, (u, v) \in V \times V \mid c_f(u, v) > 0 \,\},$$

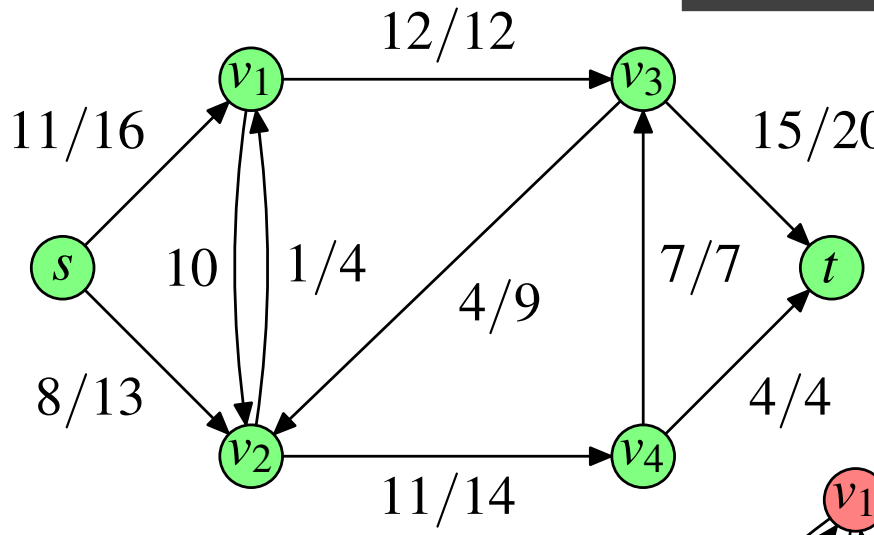
wobei

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

c_f ist die *Restkapazität*.

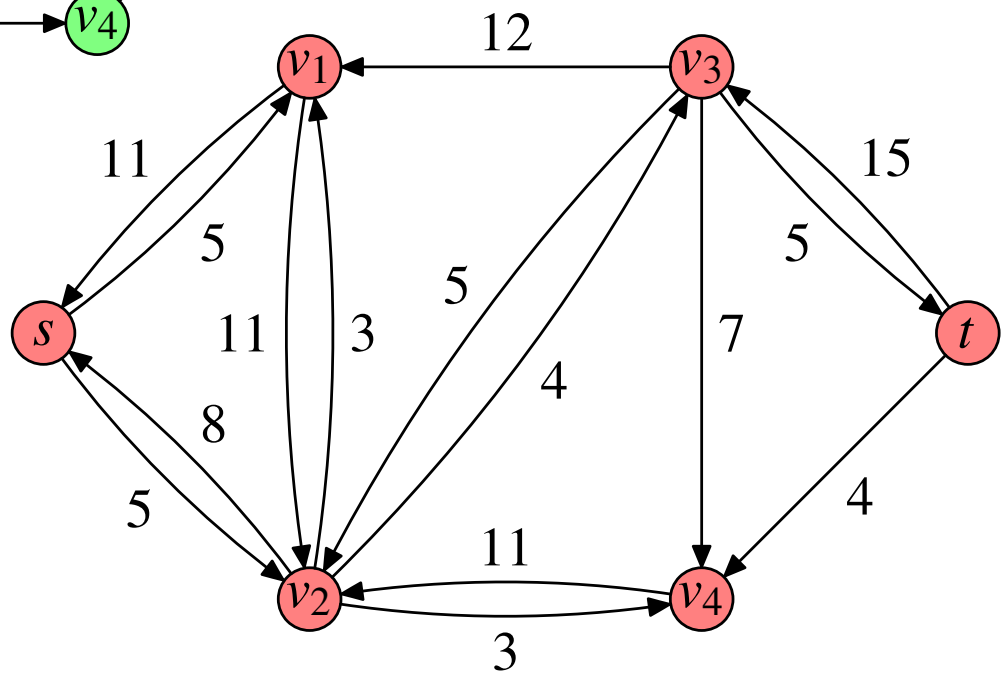
Das s-t-Netzwerk G_f hat die Kapazitäten c_f.

Beispiel



s-t-Netzwerk mit Fluß f

Residualnetzwerk G_f

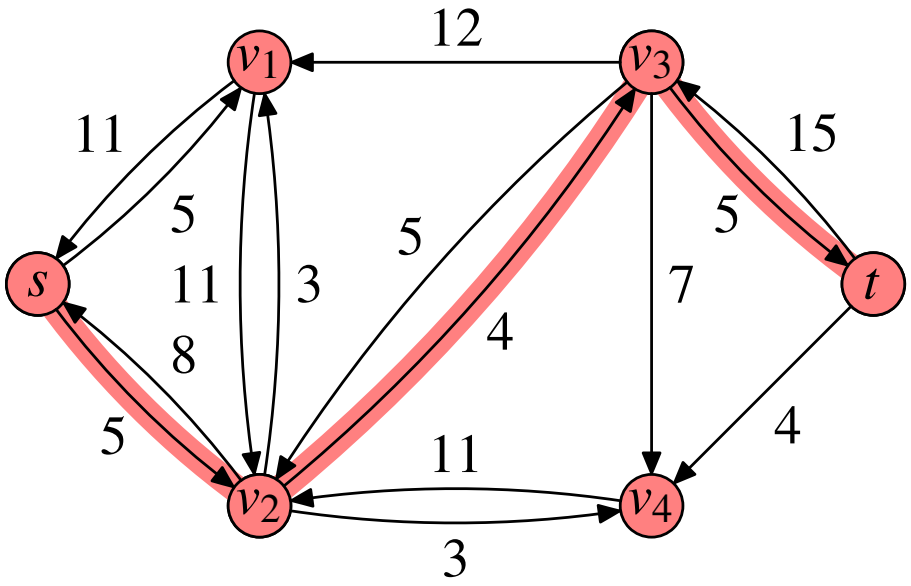


Augmentierende Pfade

Ein s-t-Pfad p in G_f heißt *augmentierender Pfad*.

$c_f(p) = \min\{\, c_f(u,v) \mid (u,v) \text{ ist auf } p \,\}$ heißt *Restkapazität* von p.

Beispiel:



Die Restkapazität dieses Pfades ist 4.

Die Ford–Fulkerson–Methode

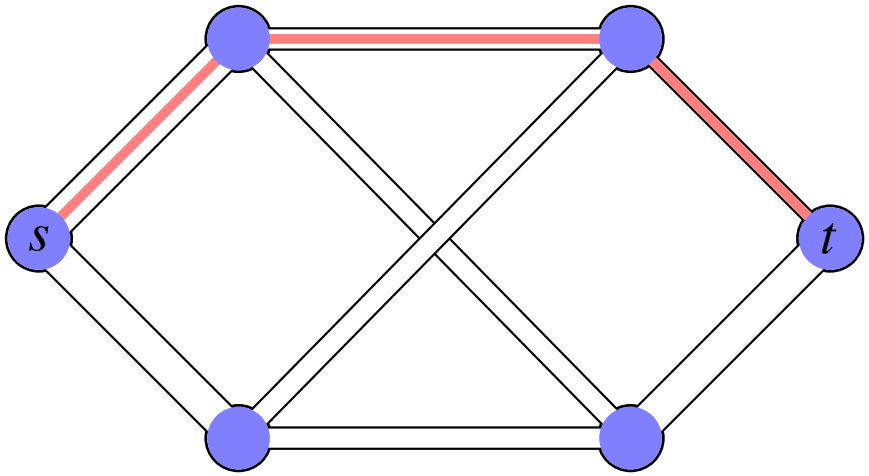
Initialisiere Fluß f zu 0
while es gibt einen augmentierenden Pfad p
  do augmentiere f entlang p
return f

$$f_p(u,v) = \begin{cases} c_f(p) & \text{falls } (u,v) \text{ auf } p \\ -c_f(p) & \text{falls } (v,u) \text{ auf } p \\ 0 & \text{sonst} \end{cases}$$

Augmentiere f entlang p: $f := f + f_p$

f_p ist ein Fluß in G_f

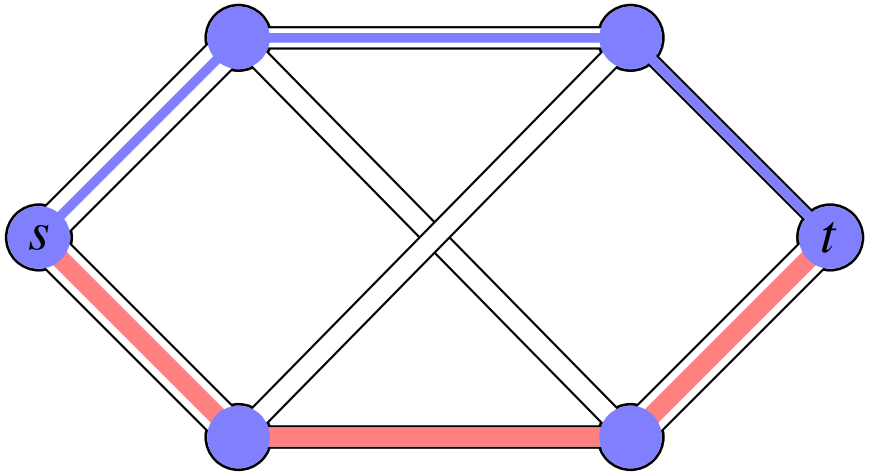
Die Ford–Fulkerson–Methode



Anfangs ist der Fluß 0.

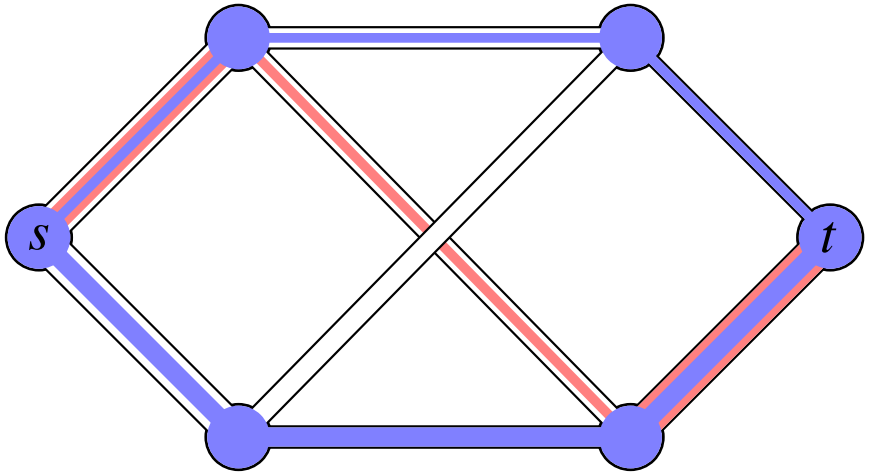
Der augmentierende Pfad ist rot eingezeichnet.

Die Ford–Fulkerson–Methode



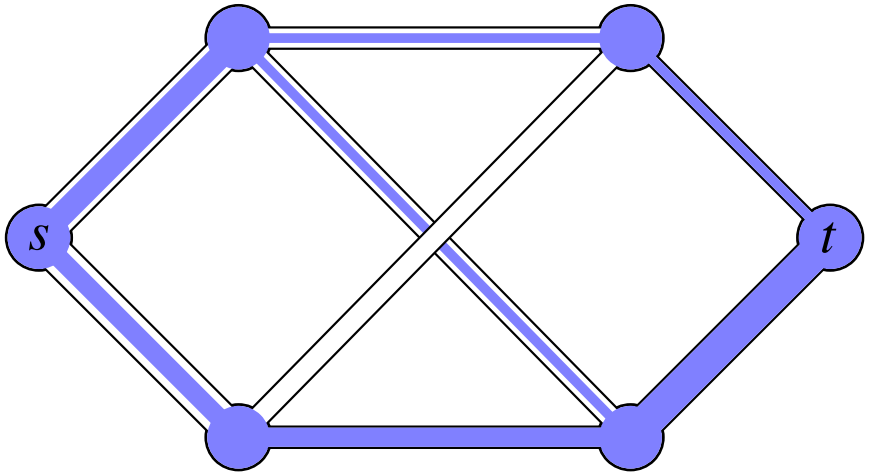
Der augmentierende Pfad hat Kapazität 5.

Die Ford–Fulkerson–Methode



Der augmentierende Pfad hat Kapazität 3.

Die Ford–Fulkerson–Methode



Jetzt gibt es keinen augmentierenden Pfad mehr.

Der Fluß ist maximal.

Korrektheit

Lemma B

Set $G = (V, E)$ ein s-t-Netzwerk und f ein Fluß in G.

Sei f' ein Fluß in G_f.

Dann ist $f + f'$ ein Fluß in G.

Konsequenz:

Die Ford–Fulkerson–Methode berechnet einen Fluß.

Beweis:

Wir müssen zeigen, daß $f + f'$ zulässig, symmetrisch und flußerhaltend ist.

Beweis (Symmetrie)

$$\begin{aligned}(f + f')(u, v) &= f(u, v) + f'(u, v) \\ &= -f(v, u) - f'(v, u) \\ &= -(f(v, u) + f'(v, u)) \\ &= -(f + f')(v, u)\end{aligned}$$

Beweis (Flußerhaltung)

Sei $u \in V - \{s, t\}$.

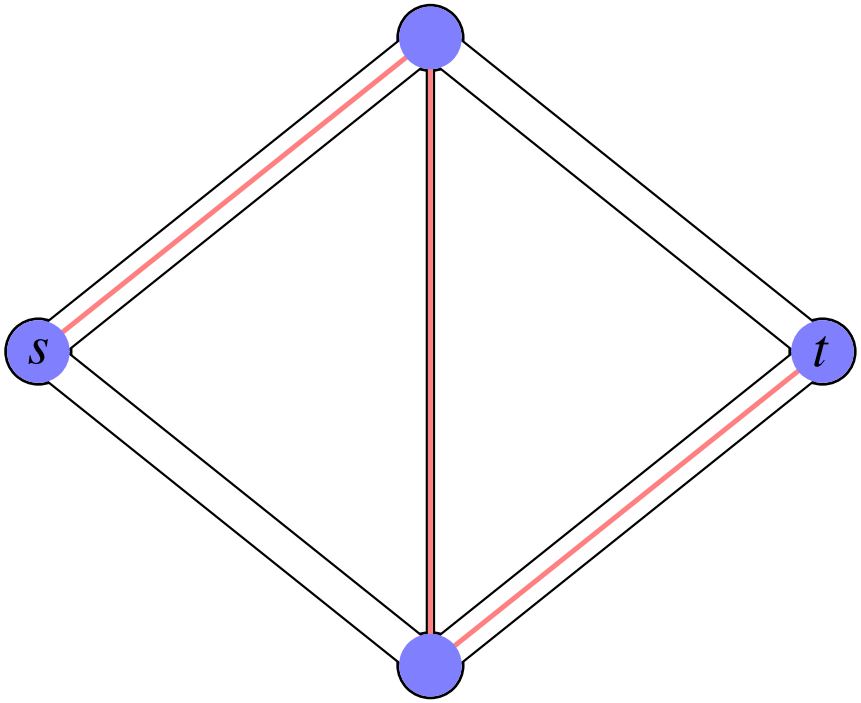
$$\begin{aligned}(f + f')(u, V) &= f(u, V) + f'(u, V) \\ &= 0 + 0 \\ &= 0\end{aligned}$$

Beweis (Zulässigkeit)

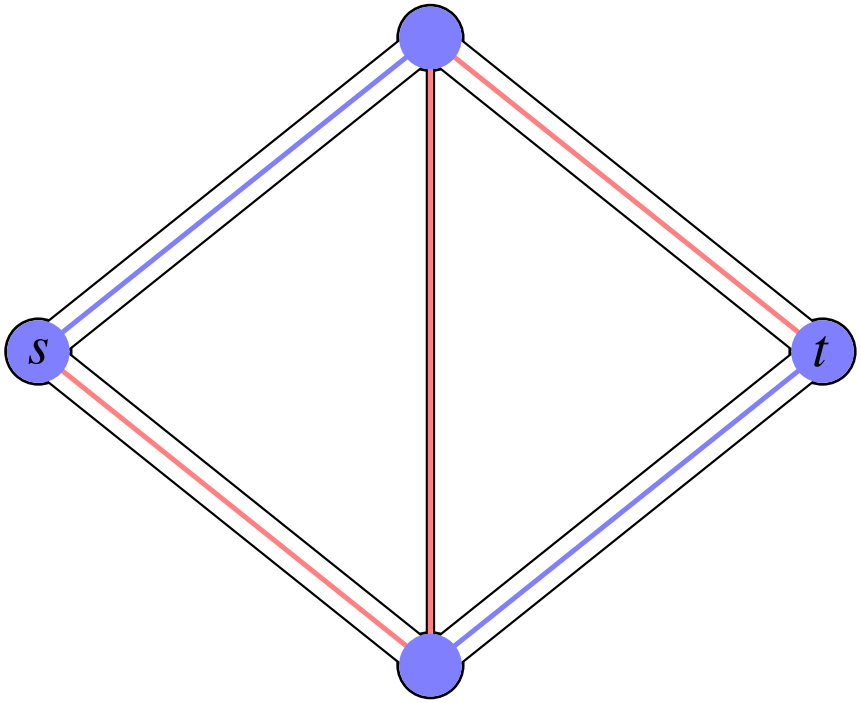
$$\begin{aligned}(f + f')(u, v) &= f(u, v) + f'(u, v) \\ &\leq f(u, v) + c_f(u, v) \\ &= f(u, v) + (c(u, v) - f(u, v)) \\ &= c(u, v)\end{aligned}$$

Der Beweis verwendet, daß f' ein Fluß in G_f ist, aber nicht, daß f ein Fluß in G ist.

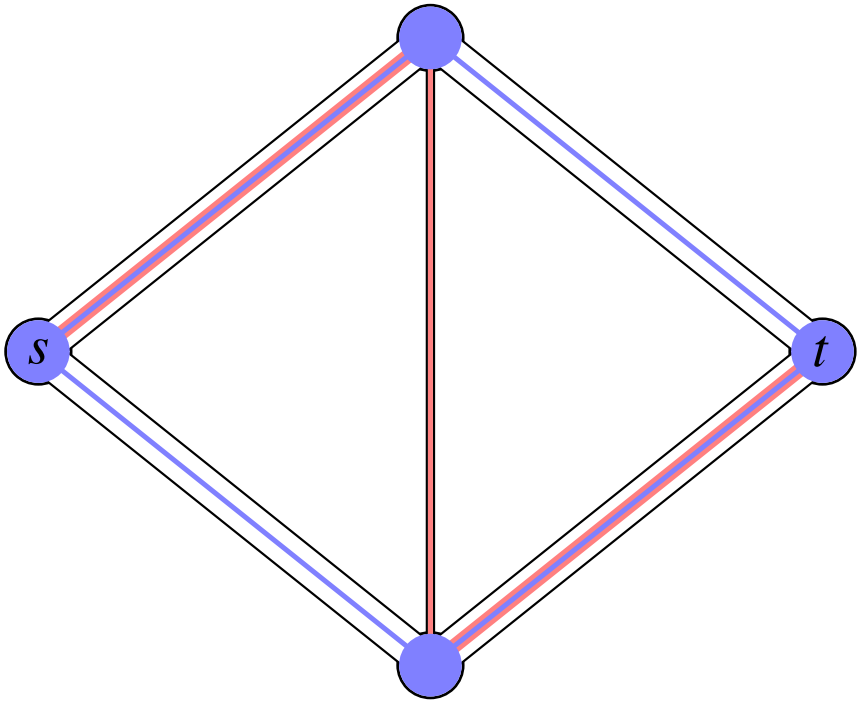
Laufzeit der Ford–Fulkerson–Methode



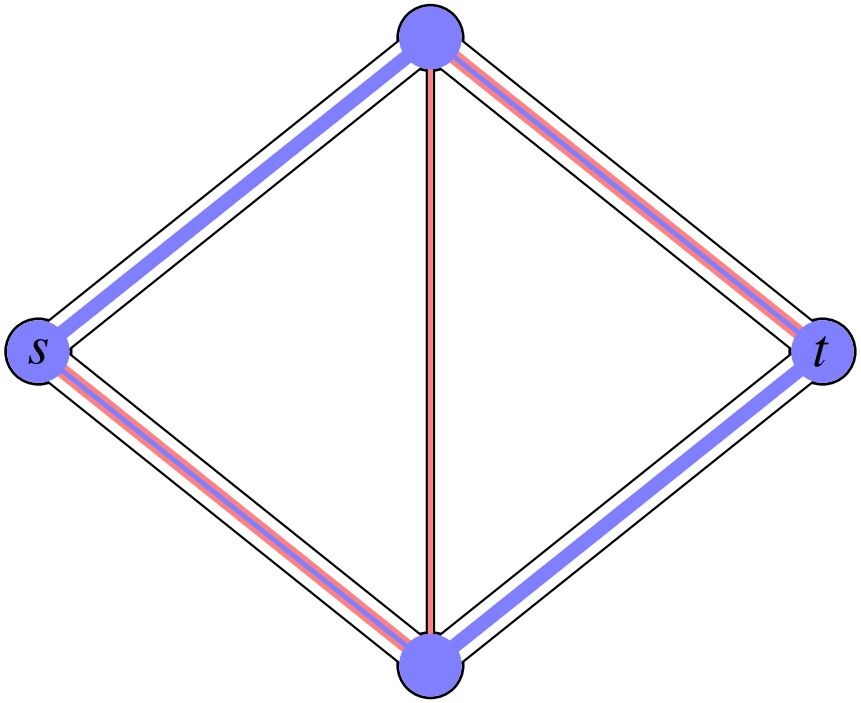
Laufzeit der Ford–Fulkerson–Methode



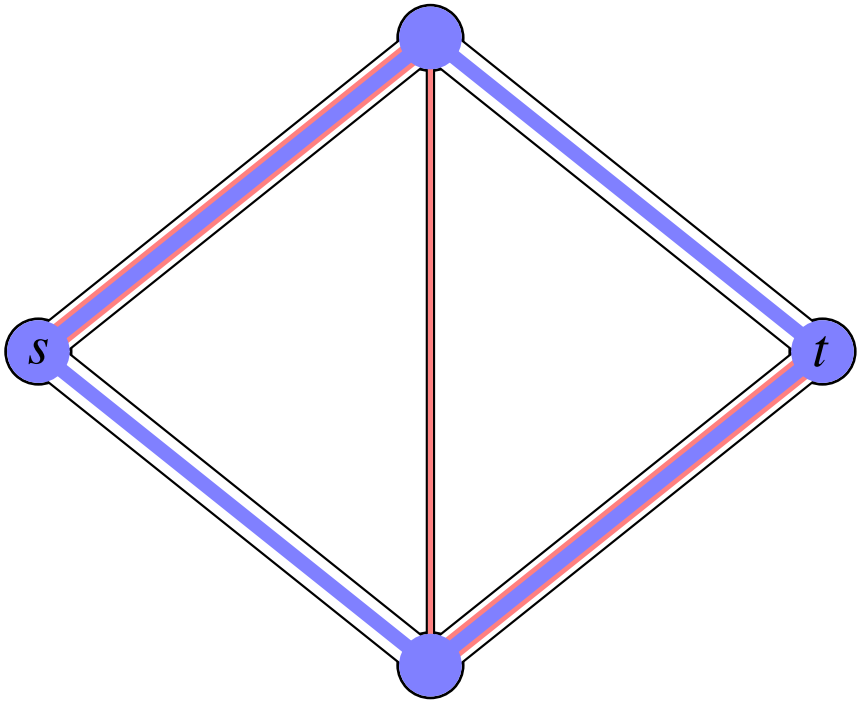
Laufzeit der Ford–Fulkerson–Methode



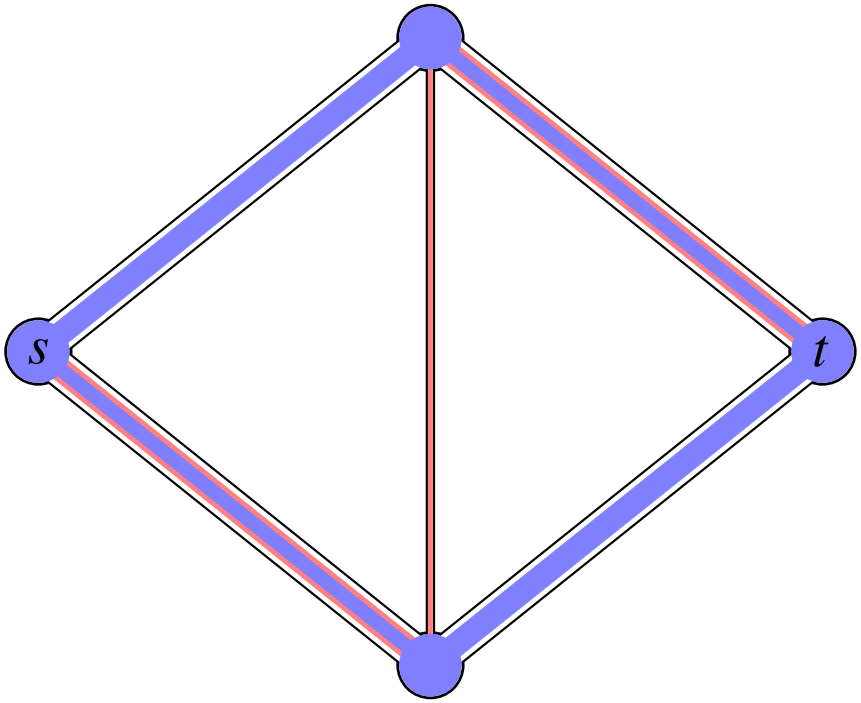
Laufzeit der Ford–Fulkerson–Methode



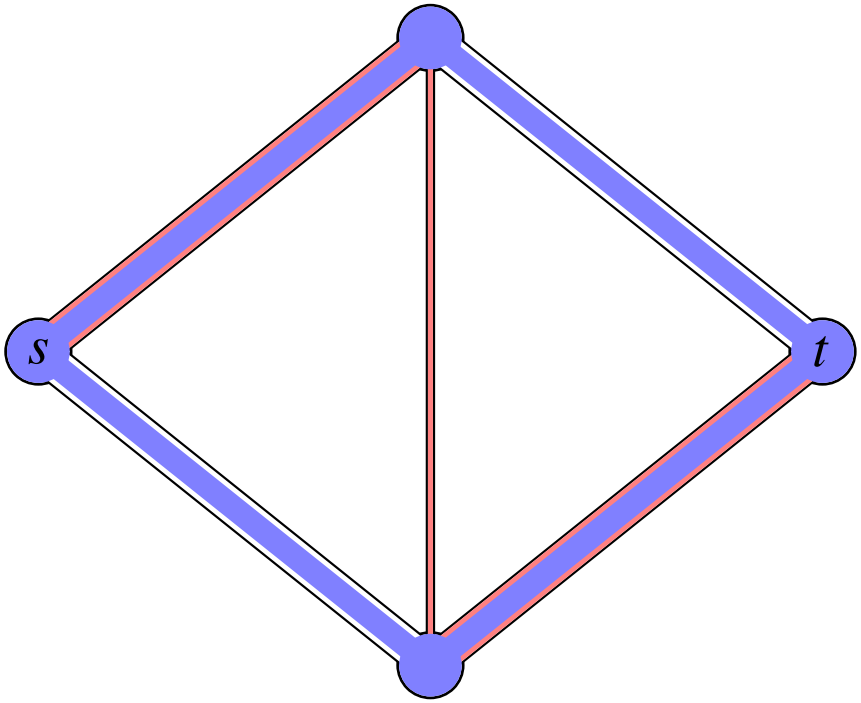
Laufzeit der Ford–Fulkerson–Methode



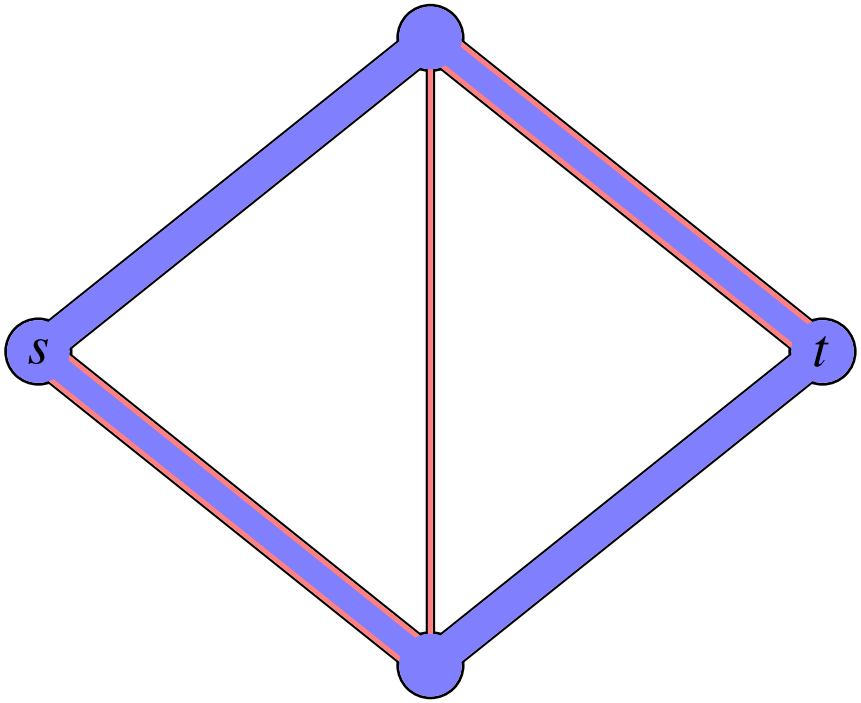
Laufzeit der Ford–Fulkerson–Methode



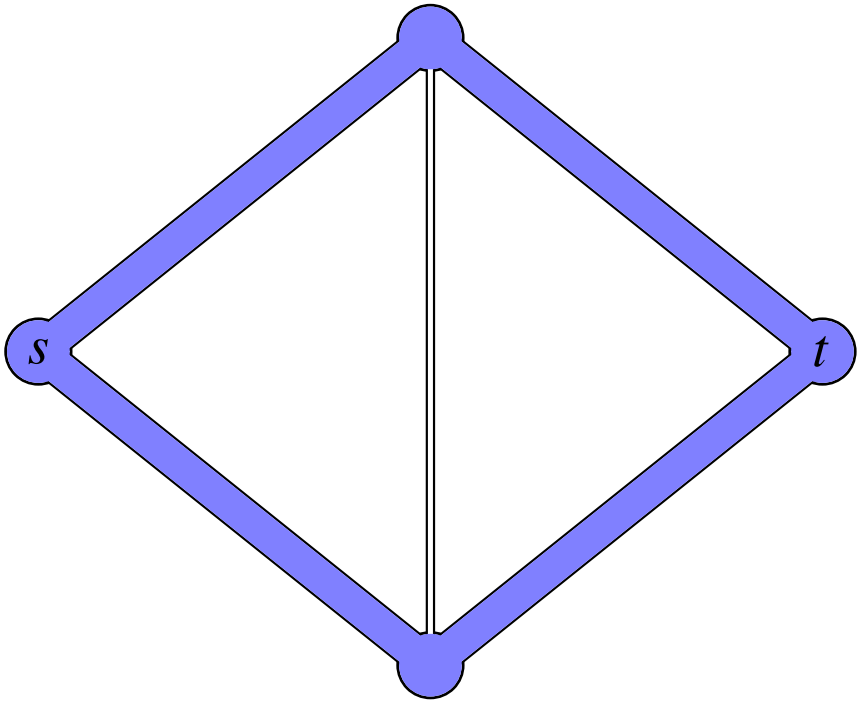
Laufzeit der Ford–Fulkerson–Methode



Laufzeit der Ford–Fulkerson–Methode



Laufzeit der Ford–Fulkerson–Methode



Laufzeit der Ford–Fulkerson–Methode

Ein Flußproblem ist *integral*, wenn alle Kapazitäten ganzzahlig sind.

Theorem.

Die Ford–Fulkerson–Methode benötigt nur $O(f^*)$ Iterationen, um ein integrales Flußproblem zu lösen, falls der Wert eines maximalen Flusses f^* ist.

Beweis.

In jeder Iteration wird der Wert des Flusses um $c_f(p) \geq 1$ erhöht. Er ist anfangs 0 und am Ende f^*.

Korollar.

Bei rationalen Kapazitäten terminiert die Ford–Fulkerson–Methode.

Schnitte in Netzwerken

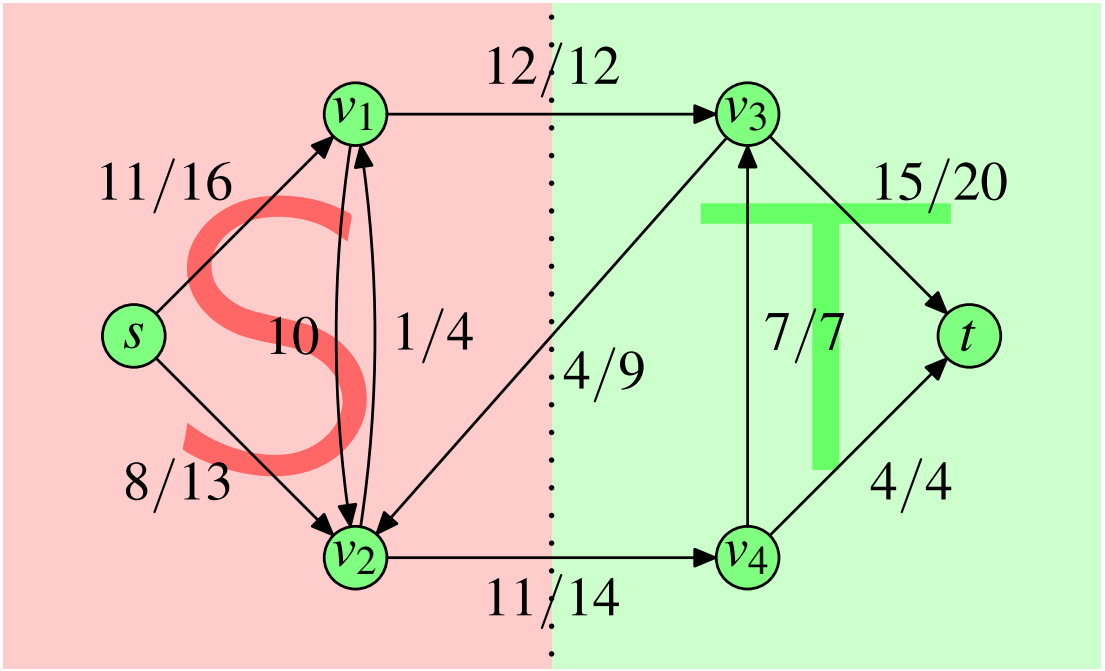
Ein *Schnitt* (S, T) in einem s-t-Netzwerk $G = (V, E)$ ist eine Partition $S \cup T = V$, $S \cap T = \emptyset$ mit $s \in S$ und $t \in T$.

Wenn f ein Fluß in G ist, dann ist $f(S, T)$ der *Fluß über* (S, T).

Die *Kapazität von* (S, T) ist $c(S, T)$.

Ein *minimaler Schnitt* ist ein Schnitt mit minimaler Kapazität.

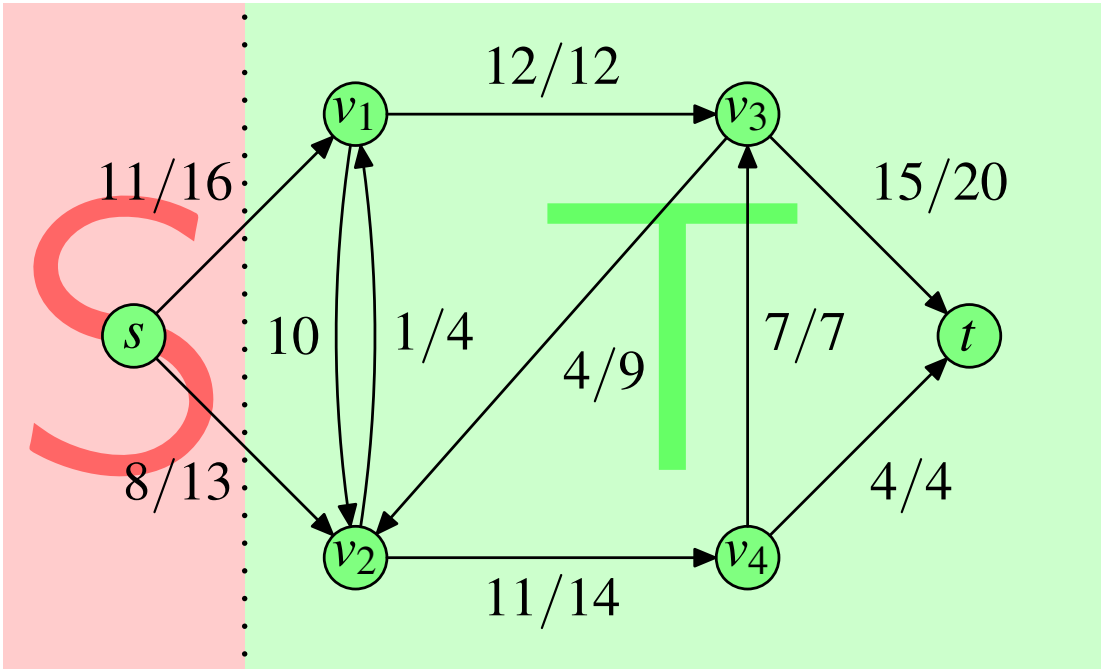
Schnitte in Netzwerken



Der Fluß über (S, T) ist 19.

Die Kapazität von (S, T) ist 26.

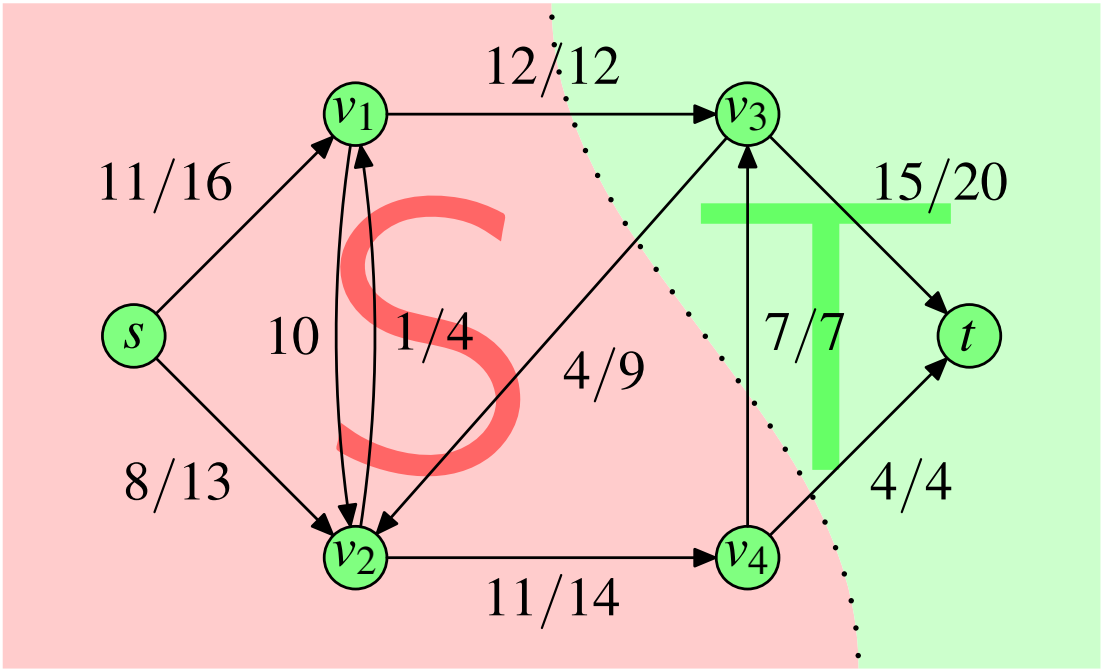
Schnitte in Netzwerken



Der Fluß über (S, T) ist 19.

Die Kapazität von (S, T) ist 29.

Schnitte in Netzwerken



Der Fluß über (S, T) ist 19.

Die Kapazität von (S, T) ist 23.

Fluß über einen Schnitt

Lemma C

Der Fluß über einen Schnitt und der Wert des Flusses sind identisch, d.h. $f(S,T) = |f|$.

Beweis

$$\begin{aligned} f(S,T) &= f(S,V) - f(S,V-T) = f(S,V) - f(S,S) \\ &= f(S,V) = f(s,V) + f(S-s,V) \\ &= f(s,V) = |f| \end{aligned}$$

Spezialfälle

$|f| = f(s,V-s) = f(V-t,t)$

Max-flow Min-cut Theorem

Theorem

Sei f ein Fluß im s-t-Netzwerk $G = (V, E)$.

Dann sind äquivalent:

1. f ist ein maximaler Fluß
2. In G_f gibt es keinen augmentierenden Pfad
3. $|f| = c(S, T)$ für einen Schnitt (S, T)

Folgerungen

Falls die Ford–Fulkerson–Methode terminiert, berechnet sie einen maximalen Fluß.

Die Kapazität eines kleinsten Schnittes gleicht dem Wert eines größten Flusses.

Beweis 1. $\rightarrow$ 2.

Sei f ein maximaler Fluß.

Nehmen wir an, es gebe einen augmentierenden Pfad p.

Dann ist $f + f_p$ ein Fluß in G mit $|f + f_p| > |f|$.

Das ist ein Widerspruch zur Maximalität von f.

Beweis 2. $\rightarrow$ 3.

G_f hat keinen s-t-Pfad.

$S := \{ v \in V \mid$ es gibt einen s-v-Pfad in $G_f \}$

$T := V - S$

Dann ist (S, T) ein Schnitt und es gilt $f(u, v) = c(u, v)$ für alle $u \in S$, $v \in T$.

Nach Lemma C gilt dann $f(S, T) = c(S, T) = |f|$.

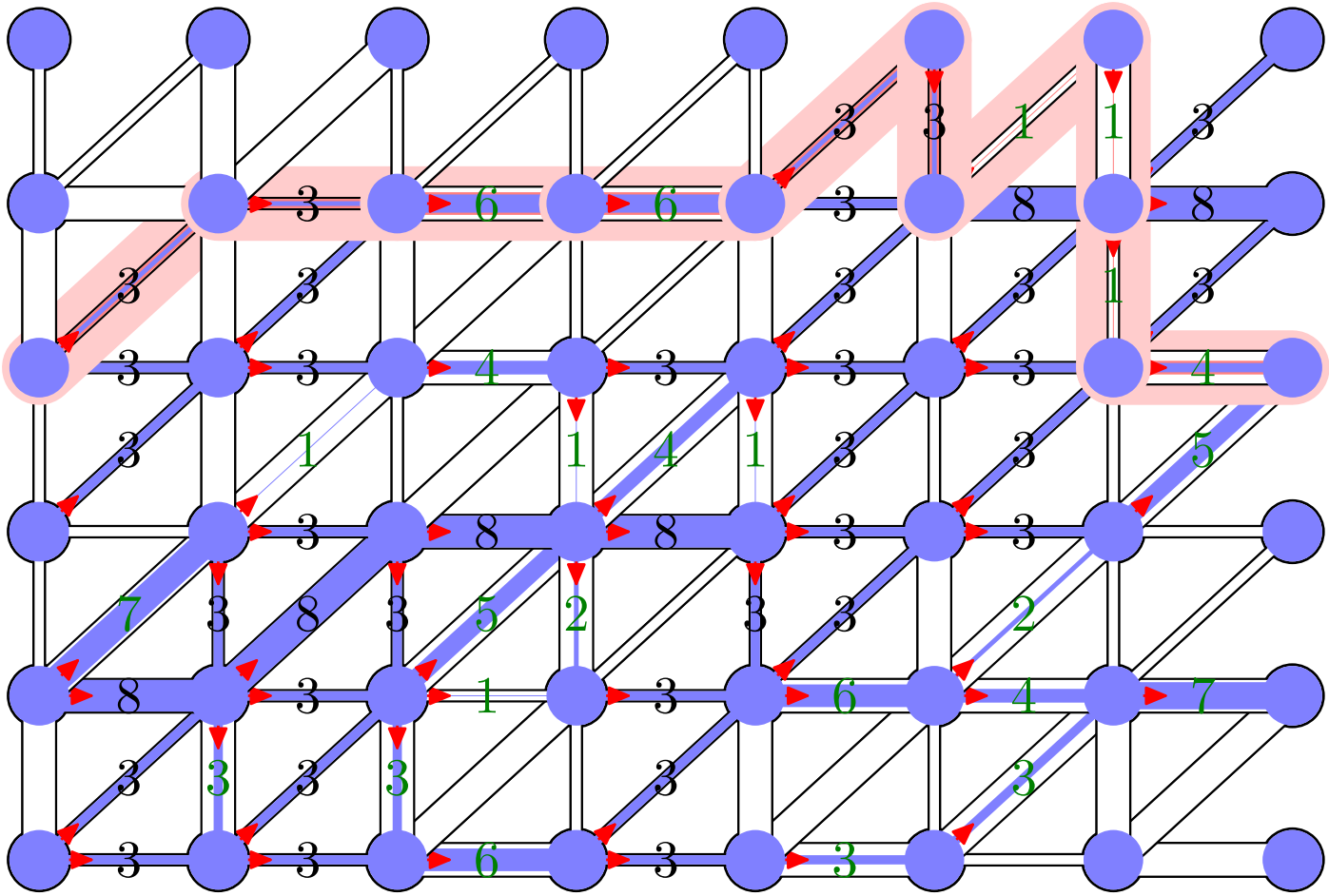
Beweis 3. $\rightarrow$ 1.

Sei f ein beliebiger Fluß.

$$\begin{aligned} |f| &= f(S,T) \\ &= \sum_{u \in S} \sum_{v \in T} f(u,v) \\ &\leq \sum_{u \in S} \sum_{v \in T} c(u,v) \\ &= c(S,T) \end{aligned}$$

Der Wert jedes Flusses ist also höchstens $c(S,T)$. Erreicht er sogar $c(S,T)$ ist er folglich maximal.

Wo ist ein minimaler Schnitt?



Wie findet man einen minimalen Schnitt?

1. Einen maximalen Fluß berechnen.
2. Eine Kante (u, v) ist *kritisch*, wenn $c(u, v) = f(u, v)$.
3. S besteht aus Knoten, die von s aus über unkritische Kanten erreicht werden können.
4. T besteht aus allen anderen Knoten.

Es gibt aber bessere, direkte Methoden!

Ganzzahlige Flüsse

Theorem

Wenn alle Kapazitäten ganzzahlig sind, dann findet die Ford–Fulkerson–Methode einen maximalen Fluß f, so daß alle $f(u, v)$ ganzzahlig sind.

Beweis

Induktion zeigt, daß die Kapazität eines augmentierenden Pfads ganzzahlig ist und $f(u, v)$ stets ganzzahlig bleiben.

Bipartites Matching

Gegeben:

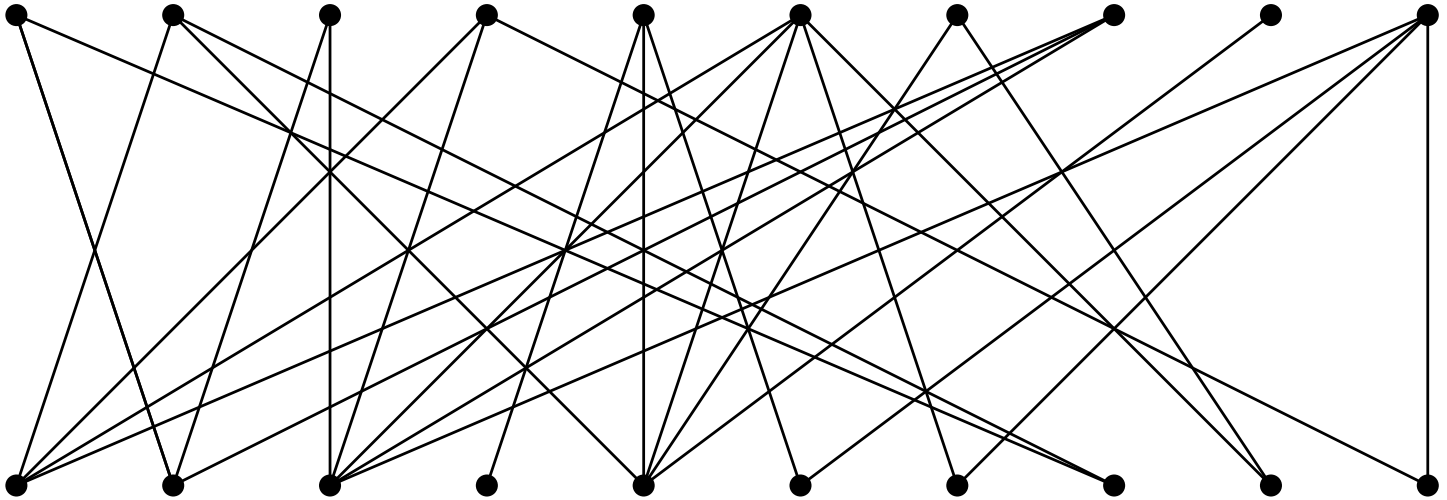
Ein bipartiter, ungerichteter Graph (V_1, V_2, E).

Gesucht:

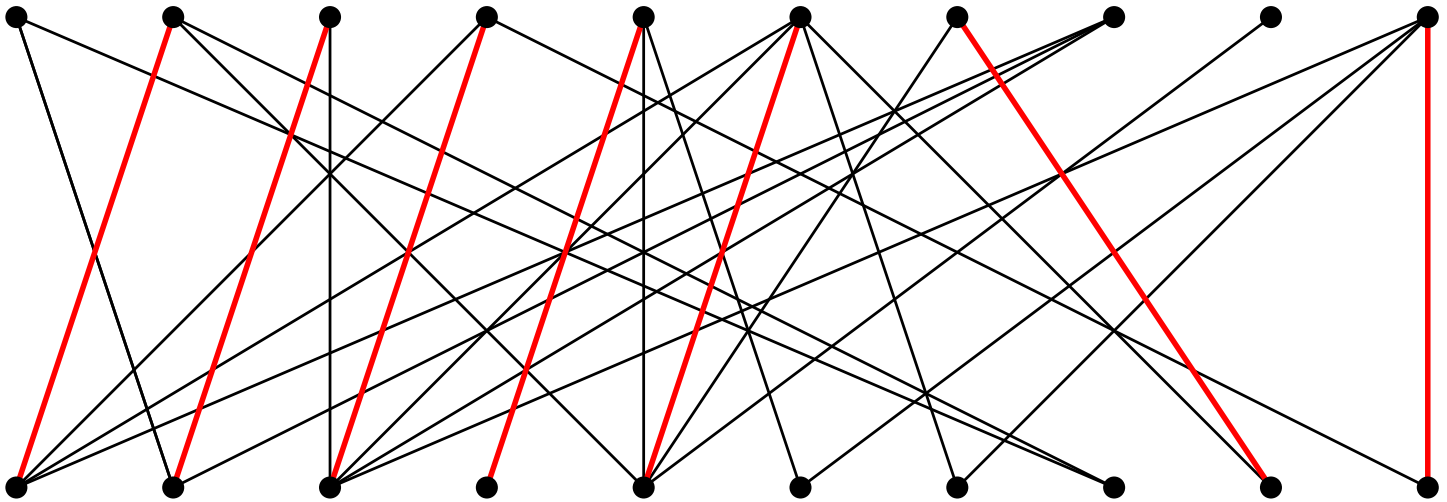
Ein Matching (Paarung) maximaler Kardinalität.

Ein *Matching* ist eine Menge paarweise nicht inzidenter Kanten, also $M \subseteq E$ mit $m_1, m_2 \in M,\ m_1 \neq m_2 \Rightarrow m_1 \cap m_2 = \emptyset$.

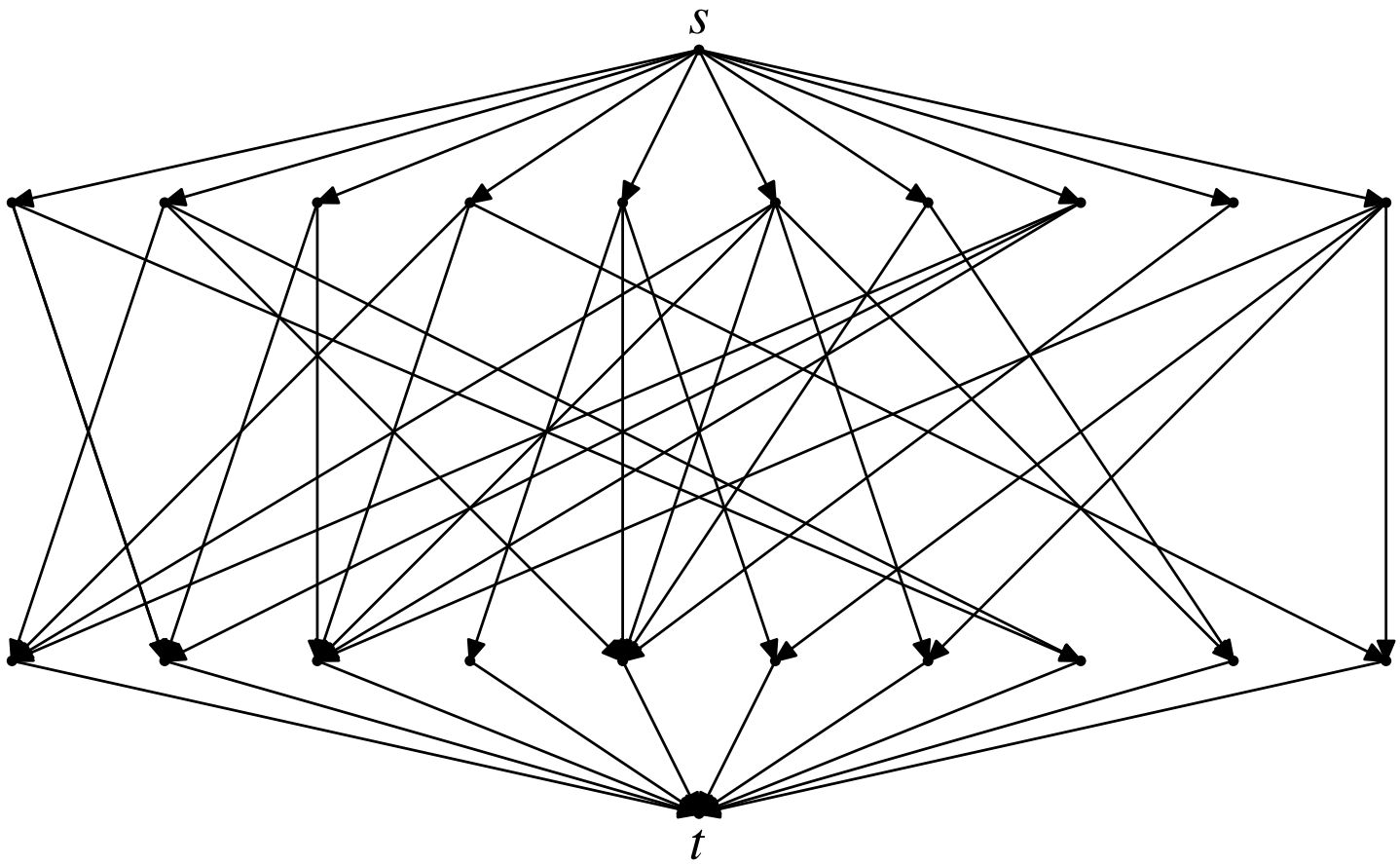
Beispiel



Beispiel

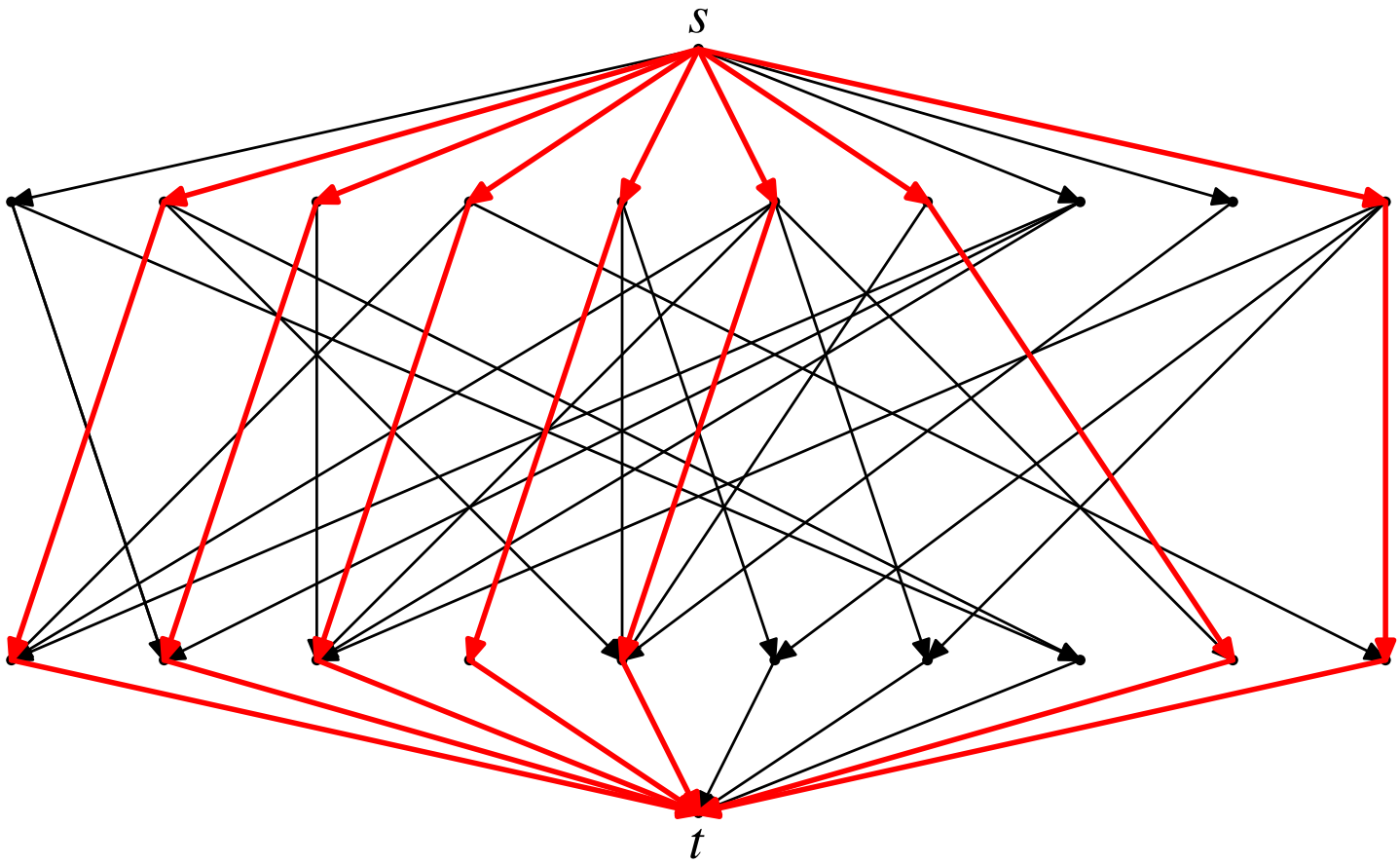


Lösung als Flußproblem

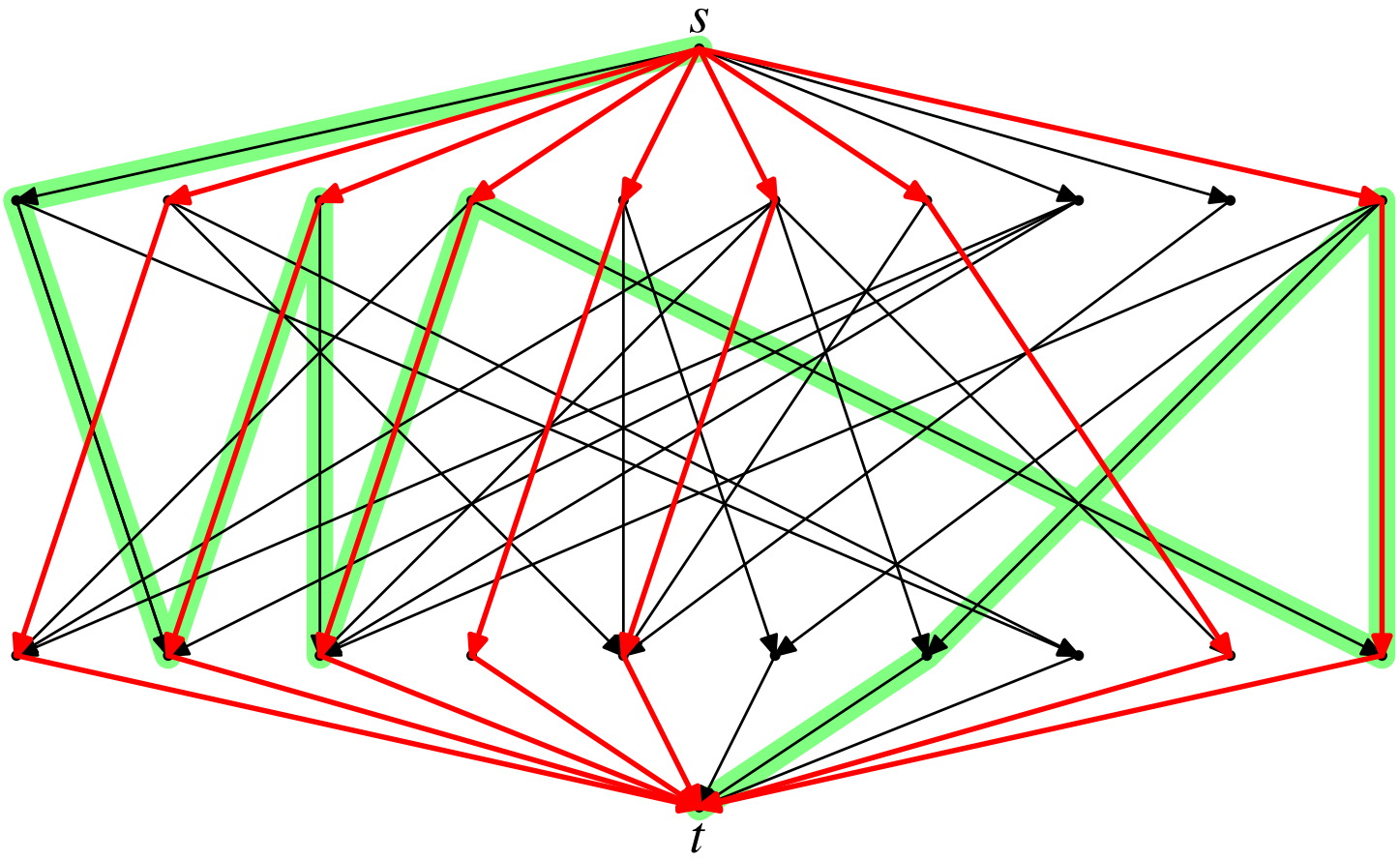


Alle Kapazitäten sind 1. Maximaler ganzzahliger Fluß entspricht einem Matching maximaler Kardinalität.

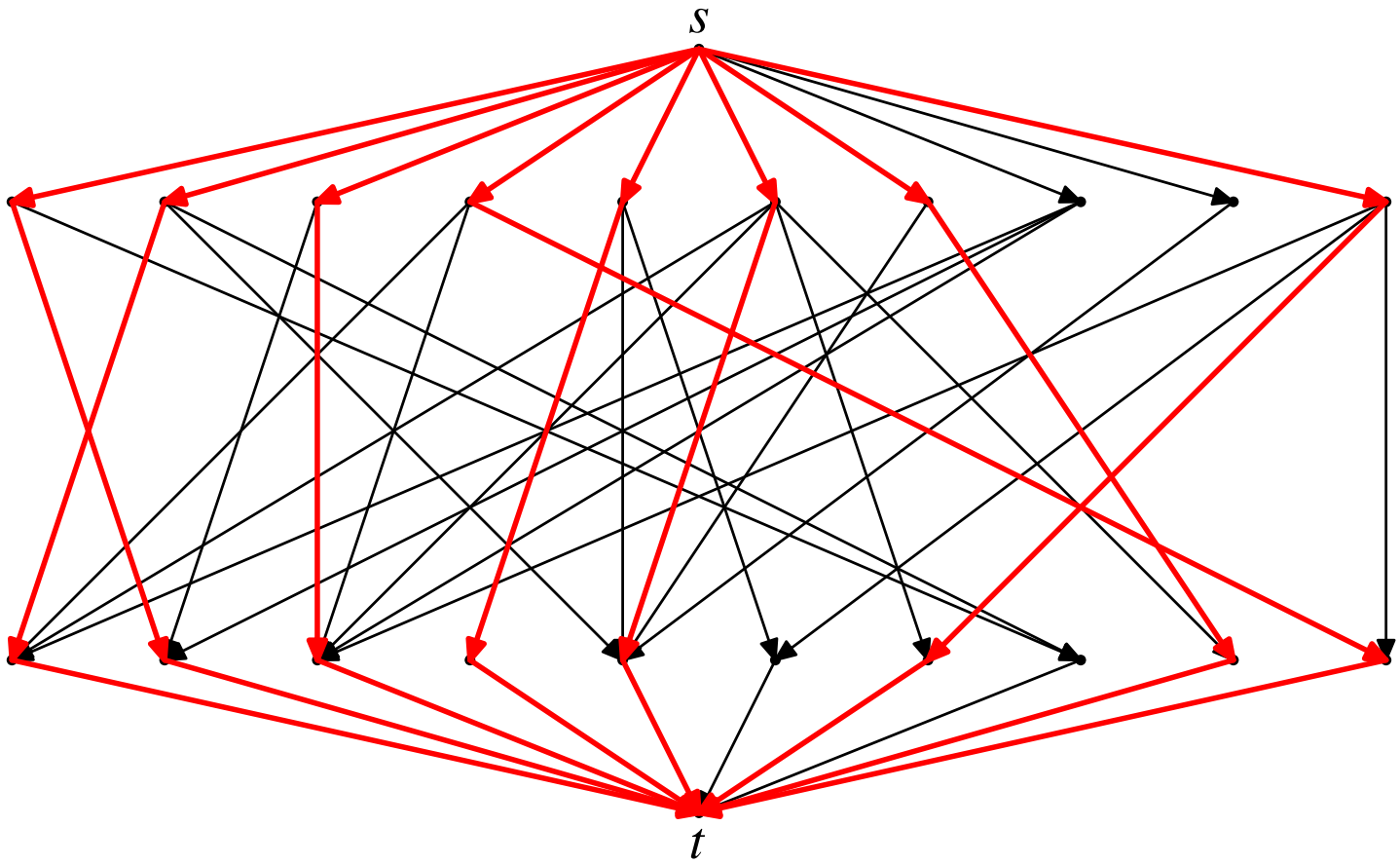
Gibt es einen augmentierenden Pfad?



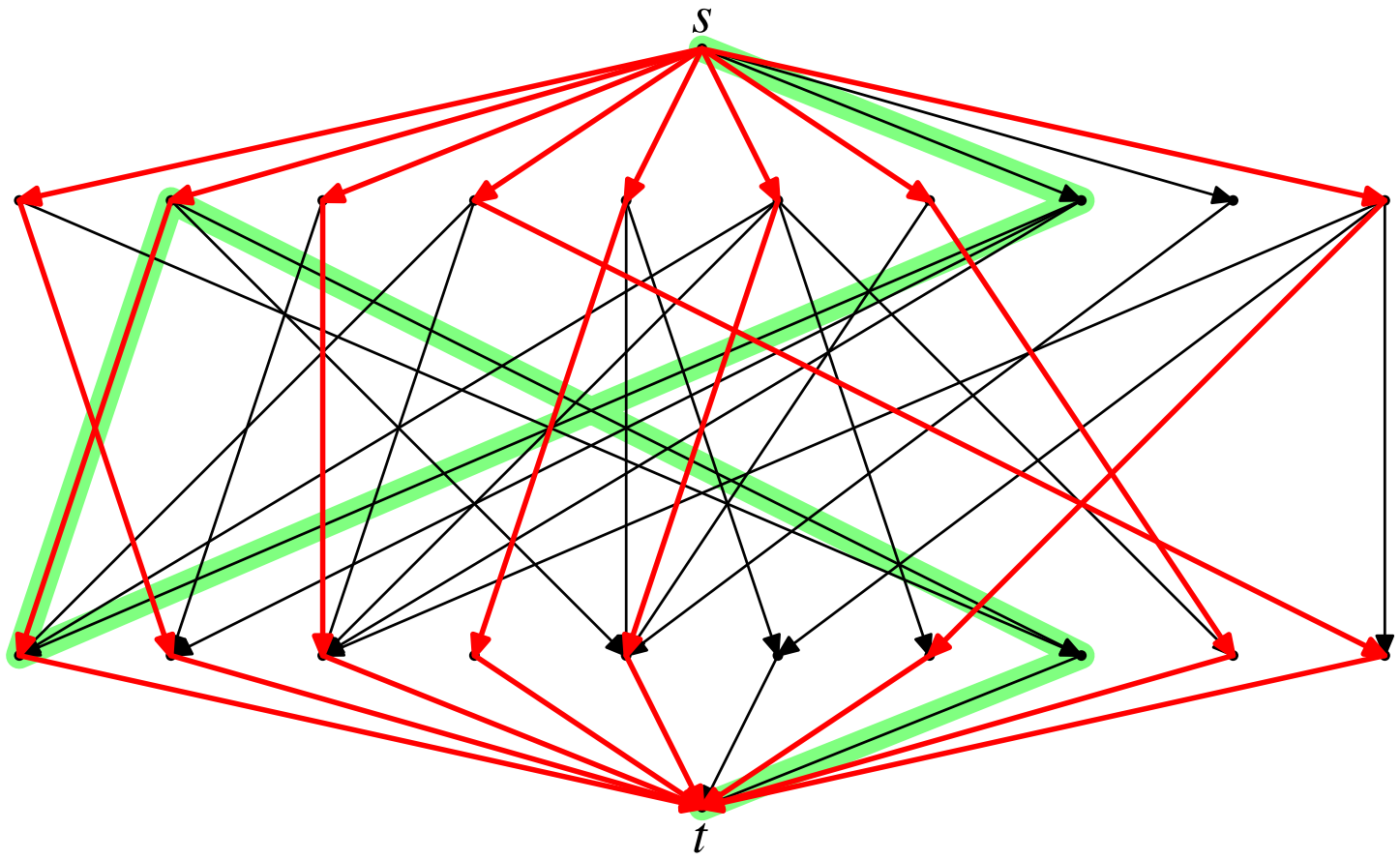
Gibt es einen augmentierenden Pfad? – Ja



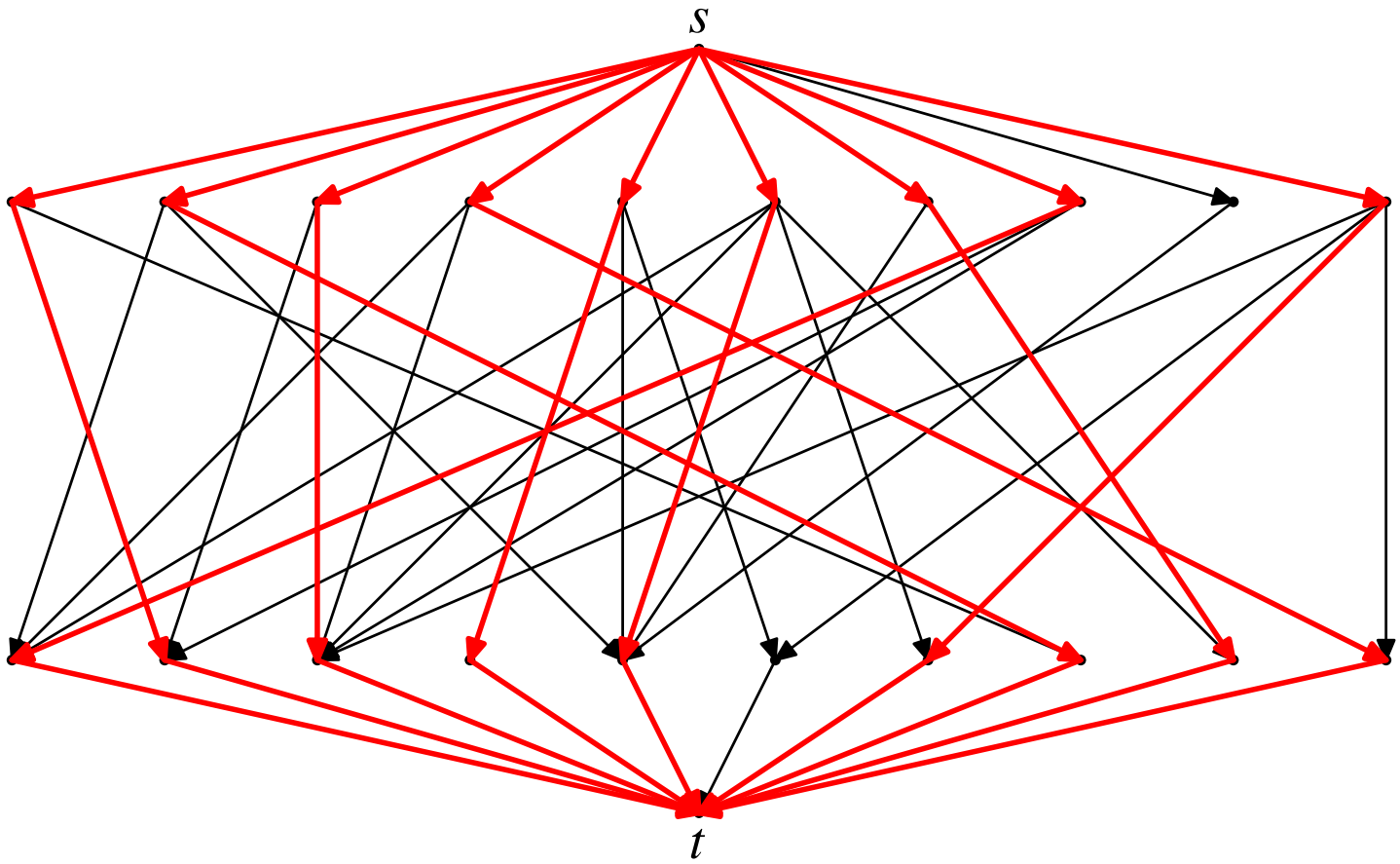
⇒ Neues Matching



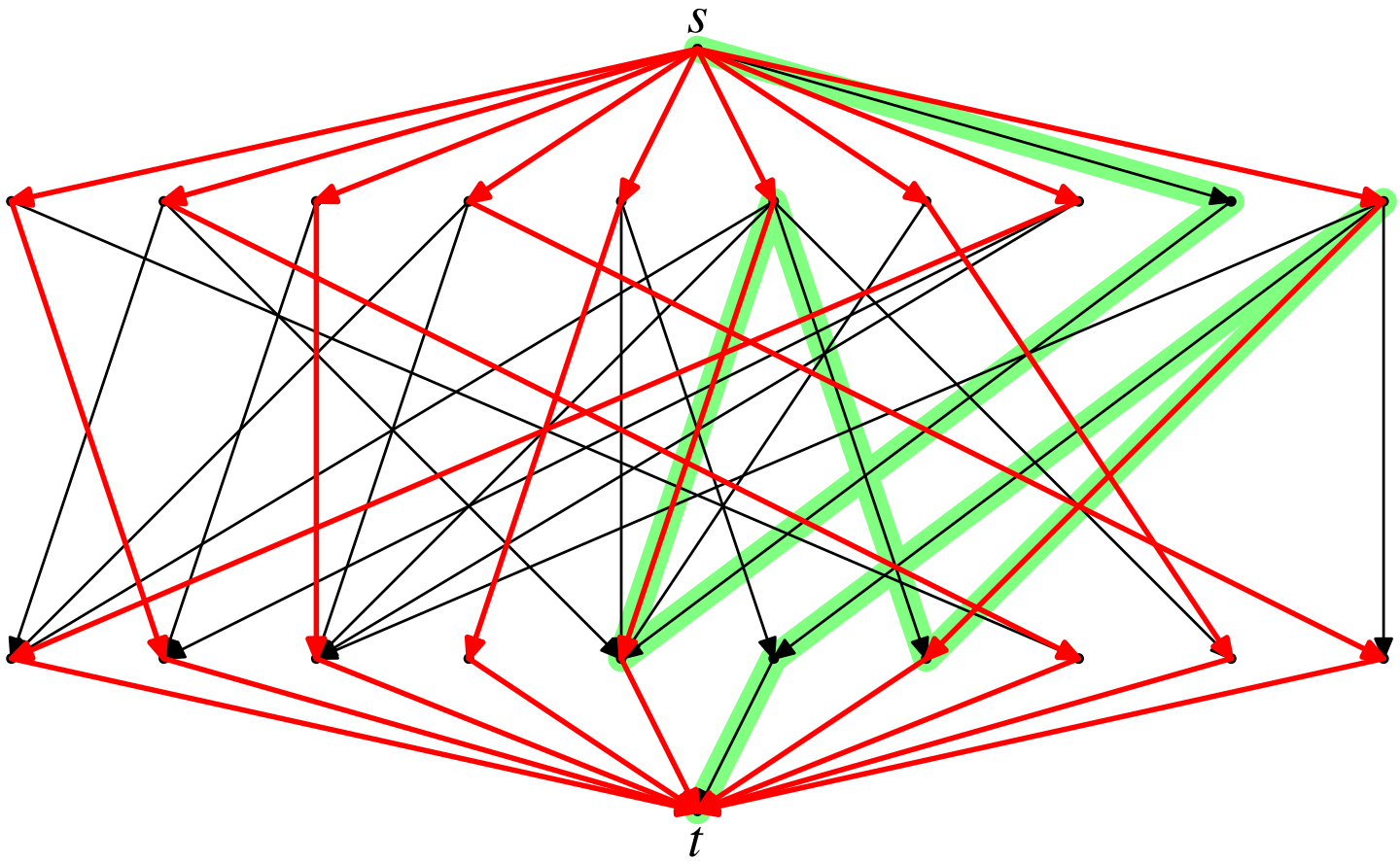
Es gibt wieder einen augmentierenden Pfad



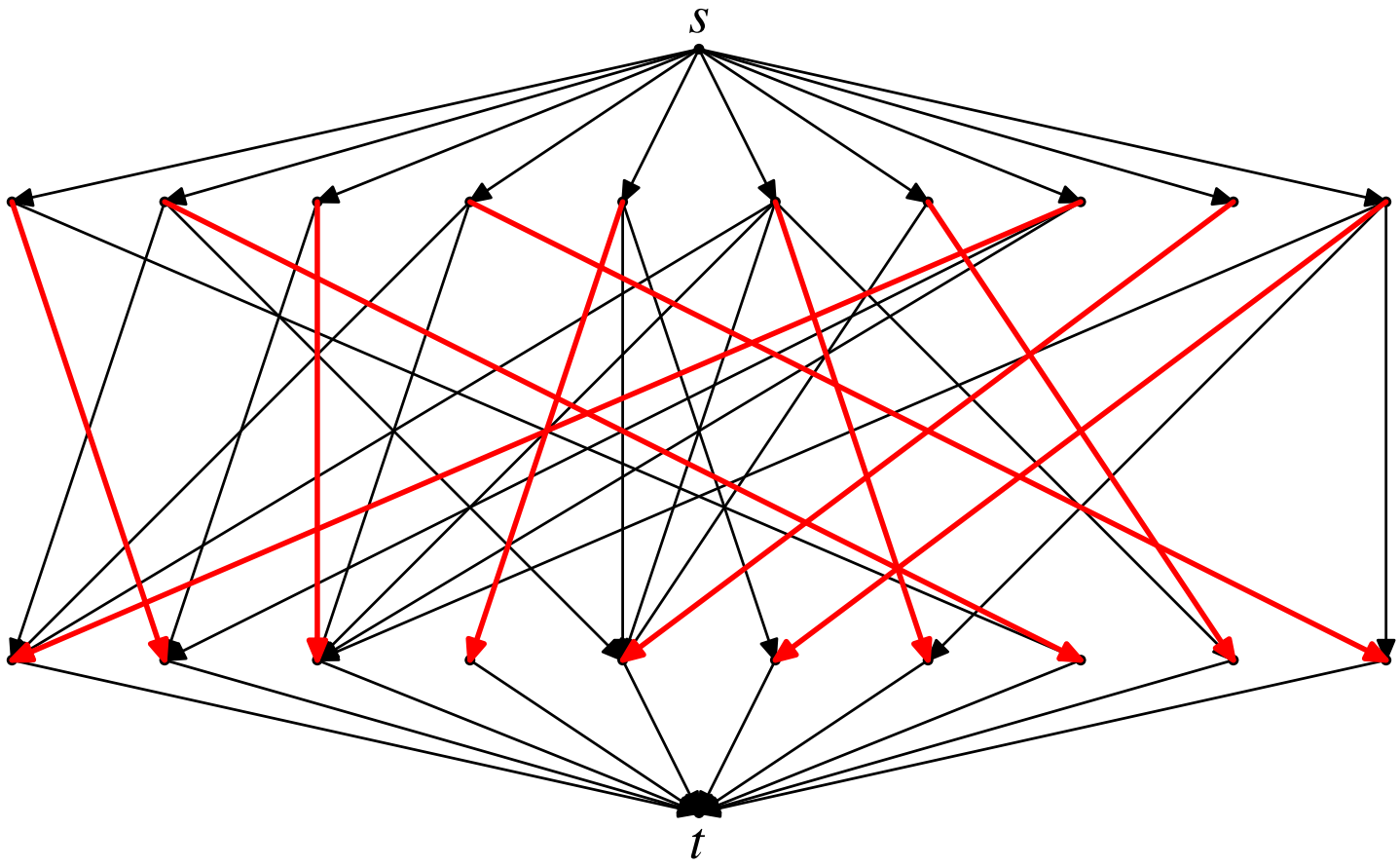
⇒ Neues Matching



Es gibt wieder einen augmentierenden Pfad



Ergebnis: Perfektes Matching



Laufzeit

Finden eines Matchings maximaler Kardinalität dauert nur $O(|E| \cdot \min\{|V_1|, |V_2|\})$ mit der Ford–Fulkerson–Methode.

- Der Fluß ist höchstens $f^* = \min\{|V_1|, |V_2|\}$.
- Finden eines Pfads dauert $O(|E|)$.

Variante der Ford–Fulkerson–Methode

Dieser Algorithmus funktioniert bei ganzzahligen Kapazitäten:

$K := 2^{\lfloor \log_2(\max\{\, c(u,v) \mid (u,v) \in E \,\}) \rfloor}$
$f := 0$
while $K \geq 1$ **do**
  while es gibt einen augmentierenden Pfad p mit $c_f(p) \geq K$ **do**
    augmentiere f entlang p
  $K := K/2$
return f

Die Laufzeit ist $O(|E|^2 \log K)$.

$\Rightarrow$ vergleiche mit $O(|E| f^*)$.

Variante der Ford–Fulkerson–Methode

Theorem

Die Laufzeit dieser Variante beträgt $O(|E|^2 \log C)$, wobei $C = \max\{\, c(u,v) \mid (u,v) \in E \,\}$.

Beweis

- Die Restkapazität eines minimalen Schnitts ist stets höchstens $2K|E|$.
- Für jedes K gibt es nur $|E|$ Augmentierungen
- Es gibt $O(\log C)$ verschiedene K

Der Edmonds–Karp–Algorithmus

Die Ford–Fulkerson–Methode kann sehr langsam sein, auch wenn das Netzwerk klein ist.

Der Edmonds–Karp–Algorithmus ist polynomiell in der Größe des Netzwerks.

Initialisiere Fluß f zu 0
while es gibt einen augmentierenden Pfad **do**
  finde einen kürzesten augmentierenden Pfad p
  augmentiere f entlang p
return f

Unterschied: Es wird ein kürzester Pfad gewählt

Der Edmonds–Karp–Algorithmus

for each edge $(u, v) \in E$ **do**
  $f(u, v) := 0$
  $f(v, u) := 0$
while there exists a path from s to t in G_f **do**
  $p :=$ a shortest path from s to t in G_f
  $c_f(p) := \min\{\, c_f(u, v) \mid (u, v) \text{ is in } p \,\}$
  for each edge (u, v) in p **do**
    $f(u, v) := f(u, v) + c_f(p)$
    $f(v, u) := -f(u, v)$
return f

Analyse des Edmonds–Karp–Algorithmus

Lemma D

Gegeben ist ein s-t-Netzwerk $G = (V, E)$. Sei f ein Fluß, der während der Ausführung des Edmonds–Karp–Algorithmus vorkommt und f' der Fluß nach der anschließenden Augmentierung.

Dann gilt: $\delta'(s, v) \geq \delta(s, v)$ für alle $v \in V$.

Hierbei ist $\delta(u, v)$ (bzw. $\delta'(u, v)$) die Länge des kürzesten Pfads von u nach v in G_f (bzw. in $G_{f'}$).

Analyse des Edmonds–Karp–Algorithmus

Beweis

Durch Widerspruch. Nehmen wir an, es gibt ein v mit

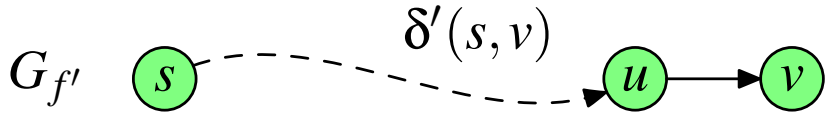
$$\delta(s,v) > \delta'(s,v). \tag{W}$$

Ohne Einschränkung können wir annehmen, daß

$$\delta(s,u) > \delta'(s,u) \Longrightarrow \delta'(s,u) \geq \delta'(s,v). \tag{M}$$

Wir müssen nur unter den möglichen v ein solches mit minimalem $\delta'(s,v)$ wählen.

Analyse des Edmonds–Karp–Algorithmus



u ist vorletzter Knoten auf kürzestem s-v-Pfad.

1. Möglichkeit: $(u,v) \in E[G_f]$

Dann gilt:

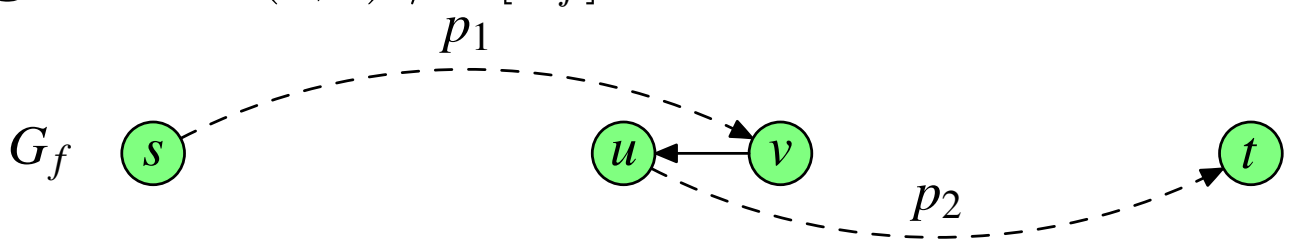
$$\delta'(s,v) = \delta'(s,u) + 1 \geq \delta(s,u) + 1 \text{ wegen (M)}$$

und

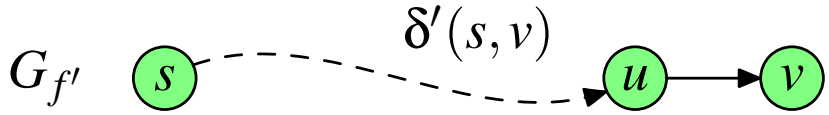
$$\delta(s,v) \leq \delta(s,u) + 1 \text{ wegen } (u,v) \in E[G_f]$$

Daraus folgt $\delta'(s,v) \geq \delta(s,v)$. Widerspruch zu (W).

2. Möglichkeit: $(u, v) \notin E[G_f]$



$p_1, (v, u), p_2$ wird als augmentierender Pfad benutzt.



Es gilt:

$$\delta(s, u) = \delta(s, v) + 1 \text{ und } \delta'(s, u) = \delta'(s, v) - 1$$

Wegen (M) gilt ausserdem:

$$\delta'(s, u) \geq \delta(s, u)$$

Zusammen folgt $\delta'(s, v) \geq \delta(s, v) + 2$, ein Widerspruch zu (W). $\square$

Analyse des Edmonds–Karp–Algorithmus

Theorem

Der Edmonds–Karp–Algorithmus findet einen maximalen Fluß in $O(|E|^2|V|)$ Schritten.

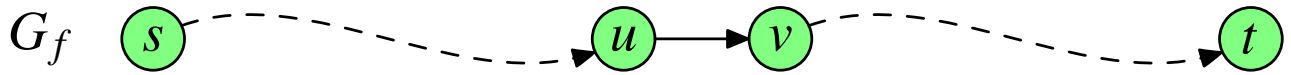
Beweis

- Als Spezialfall der Ford–Fulkerson–Methode ist der Algorithmus korrekt.
- Jede Augmentierung benötigt $O(|E|)$ Schritte. Ein kürzester augmentierender Pfad kann mit BFS gefunden werden.
- Wir werden zeigen, daß nur $O(|E| \cdot |V|)$ Augmentierungen vorgenommen werden.

Sei f der Fluß vor und f' der Fluß nach einer Augmentierung.

Eine Kante (u, v) heißt *kritisch*, wenn $f(u, v) < c(u, v)$ und $f'(u, v) = c(u, v)$.

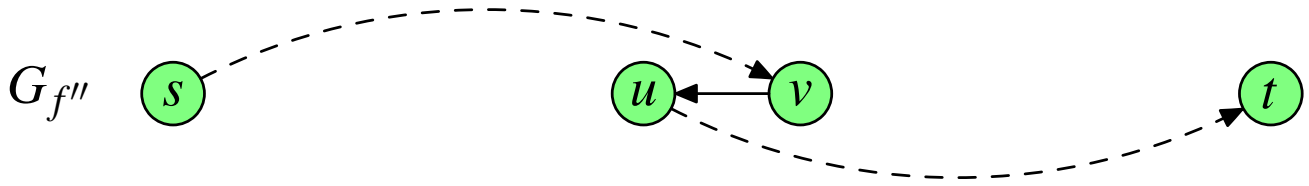
Jeder augmentierende Pfad enthält eine kritische Kante:



Sei (u, v) kritisch. Kann (u, v) später nocheinmal kritisch werden?

(u, v) verschwindet aus dem Residualnetzwerk.

Bevor (u, v) wieder kritisch wird, muß (u, v) wieder im Residualnetzwerk erscheinen:



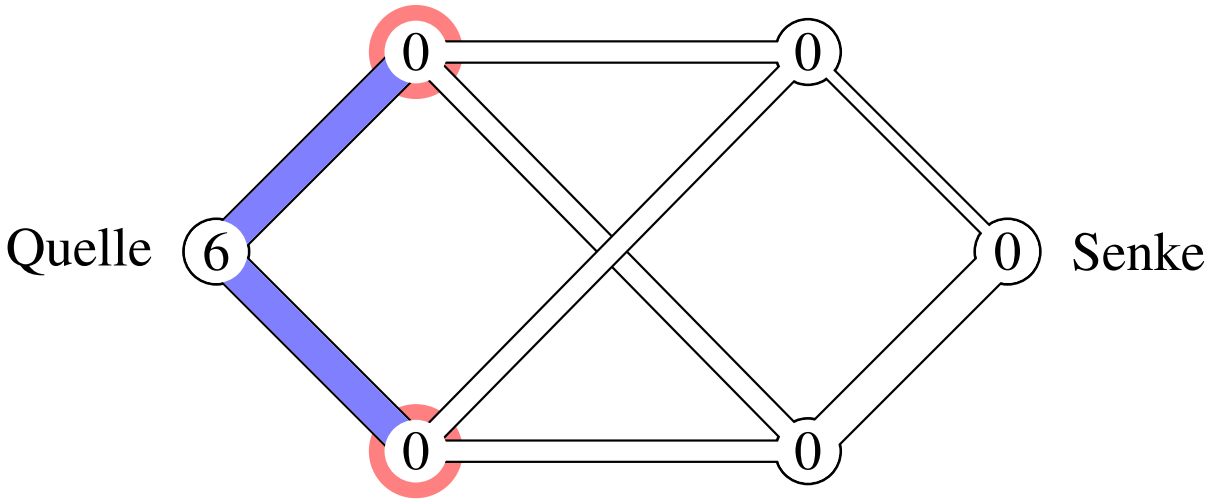
$\Rightarrow$ Der augmentierende Pfad geht durch (v, u).

- $\delta''(s, u) = \delta''(s, v) + 1$
- $\delta(s, v) = \delta(s, u) + 1$
- $\delta''(s, v) \geq \delta(s, v)$ wegen Lemma D
- $\Rightarrow \delta''(s, u) > \delta(s, u)$.

Bevor (u, v) wieder kritisch werden kann, hat sich der Abstand von s zu u erhöht.

Also kann eine Kante nur $|V|$ mal kritisch werden und es gibt höchstens $|E| \cdot |V|$ Augmentierungen. $\square$

Preflow-Push-Algorithmen

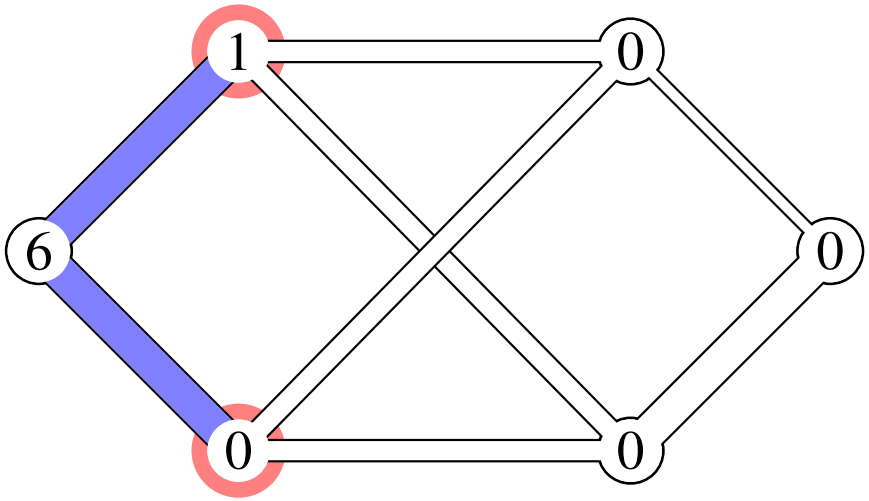


Überfließende Knoten sind rot markiert.

Die Zahlen geben die *Höhe* an.

(Die Kapazitäten sind wieder 3, 5 und 8.)

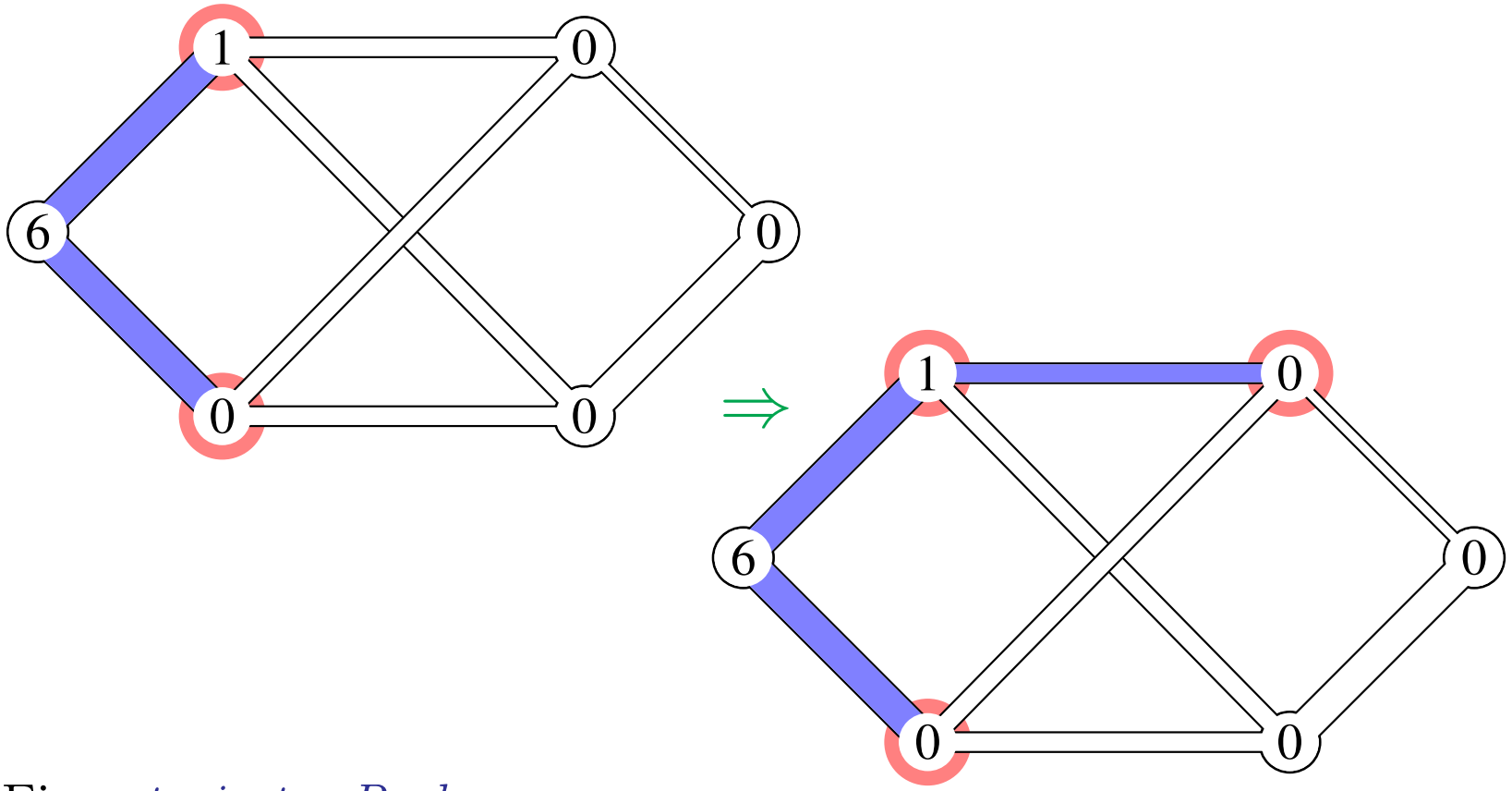
Preflow-Push-Algorithmen



Ein *Lift-Operation*.

Falls alle Nachbarn eines überfließender Knoten im Residualnetzwerk nicht kleinere Höhe haben, darf der Knoten „geliftet“ werden. Seine neue Höhe ist um eins höher als die minimale Höhe seiner Nachbarn im Residualnetzwerk.

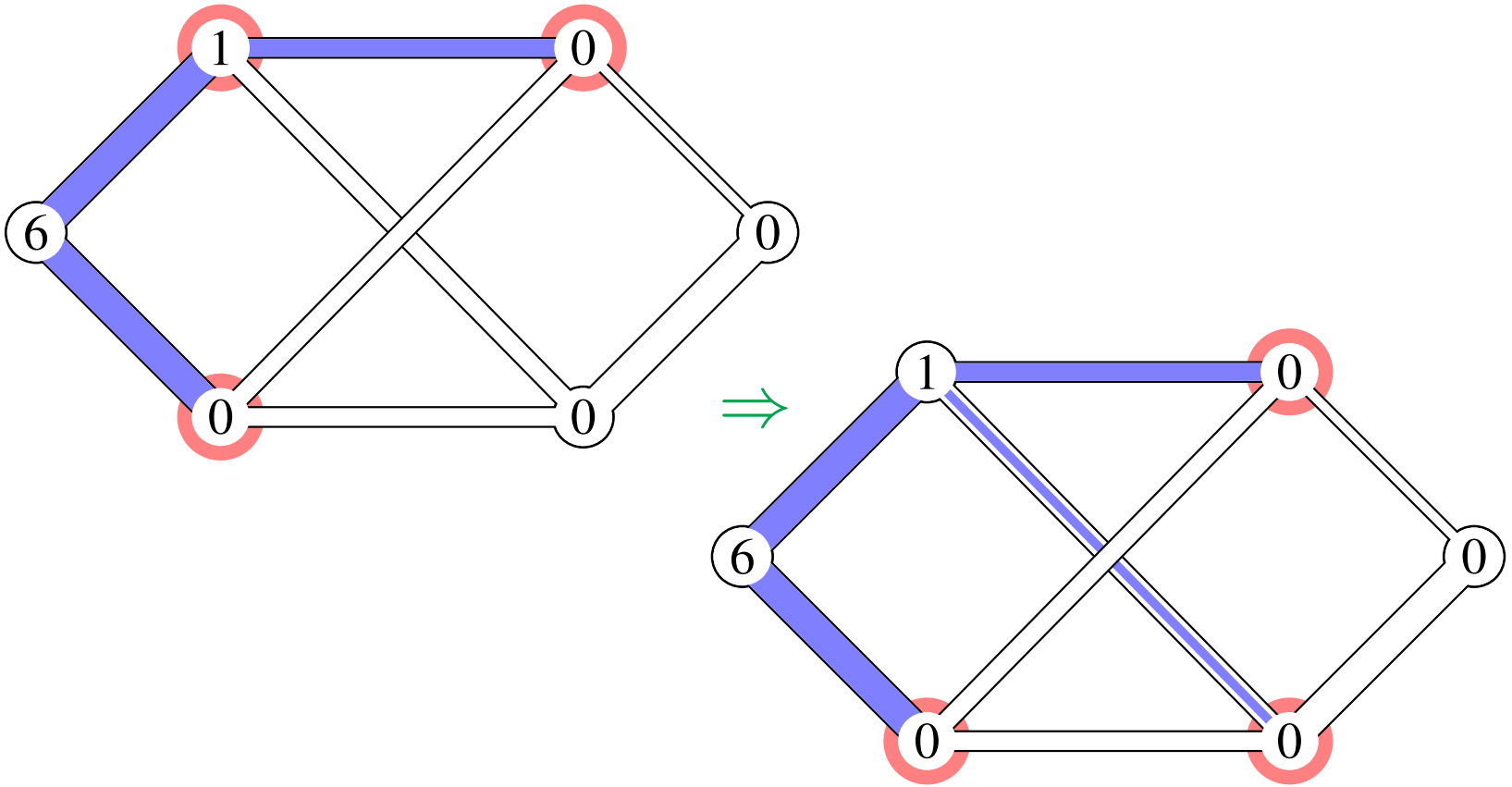
Preflow-Push-Algorithmen



Ein *saturierter Push*.

Zu einem Nachbarn im Residualnetzwerk kann Fluß „gepusht“ werden. Die Höhe muß dorthin um 1 abnehmen.

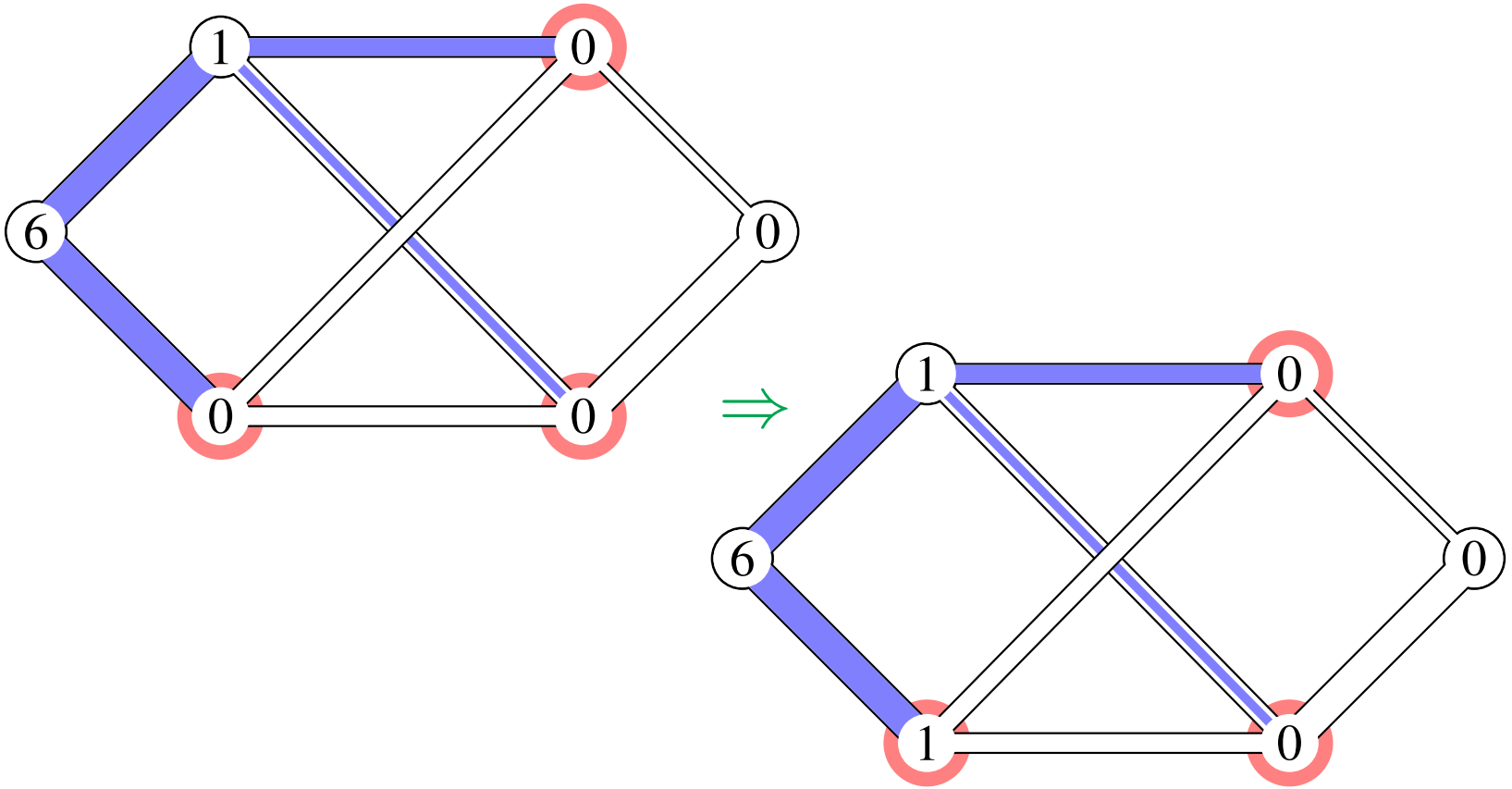
Preflow-Push-Algorithmen



Ein *nicht-saturierter Push*.

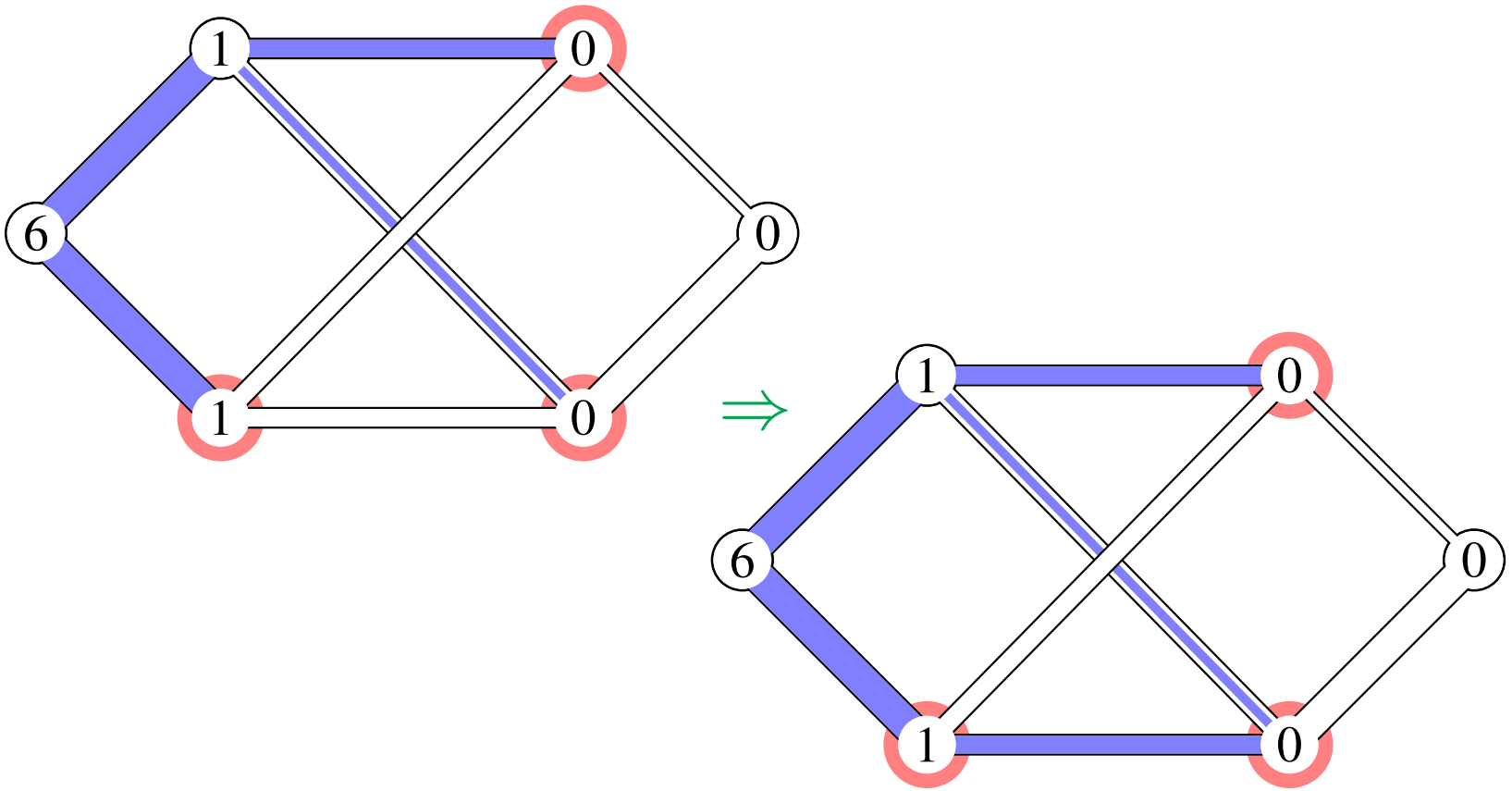
Ein „Push“ ist *saturiert*, wenn die Restkapazität der Kante erschöpft wird.

Preflow-Push-Algorithmen



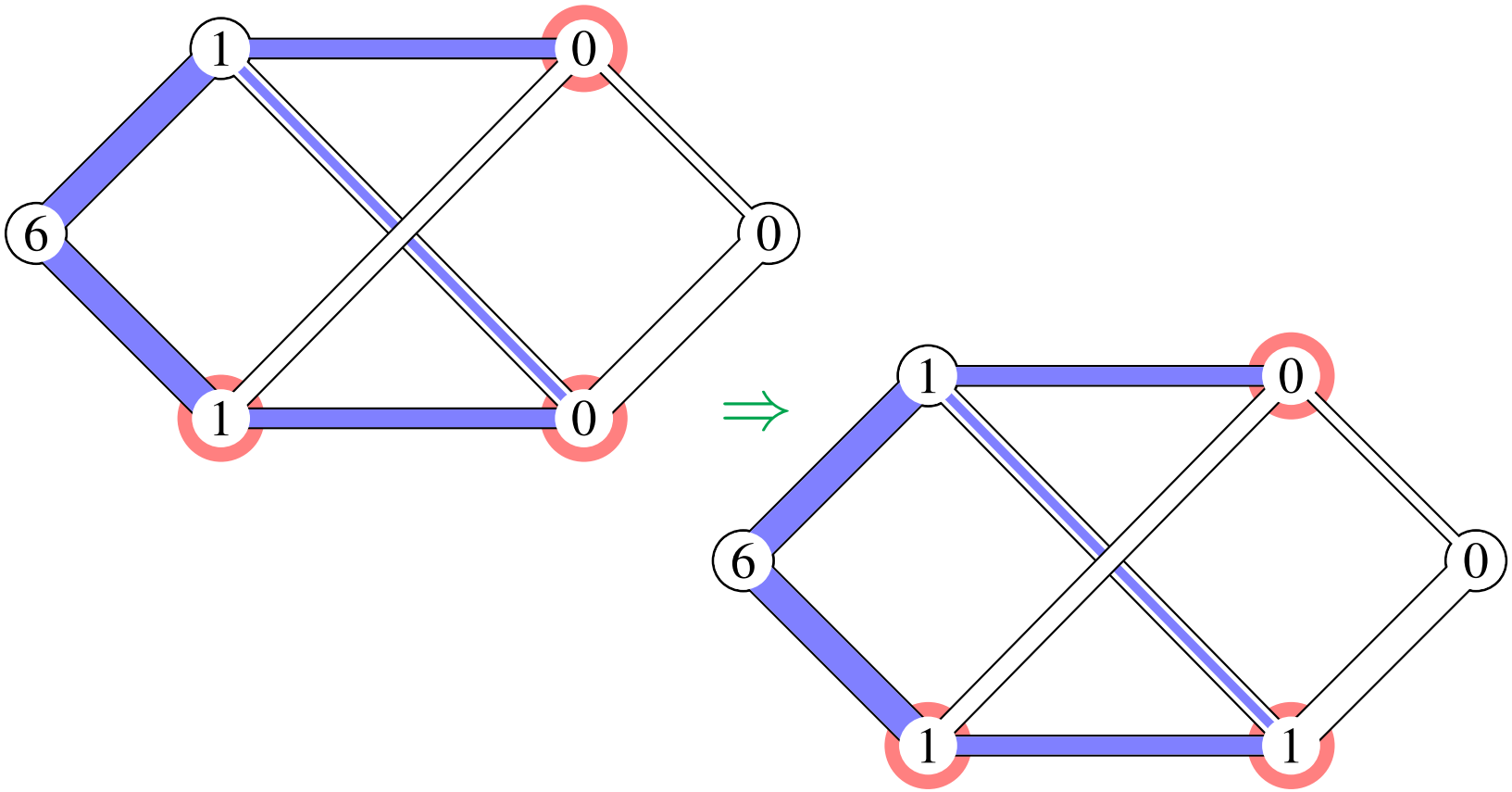
Welche Operationen sind hier auf die überfließenden Knoten anwendbar?

Preflow-Push-Algorithmen



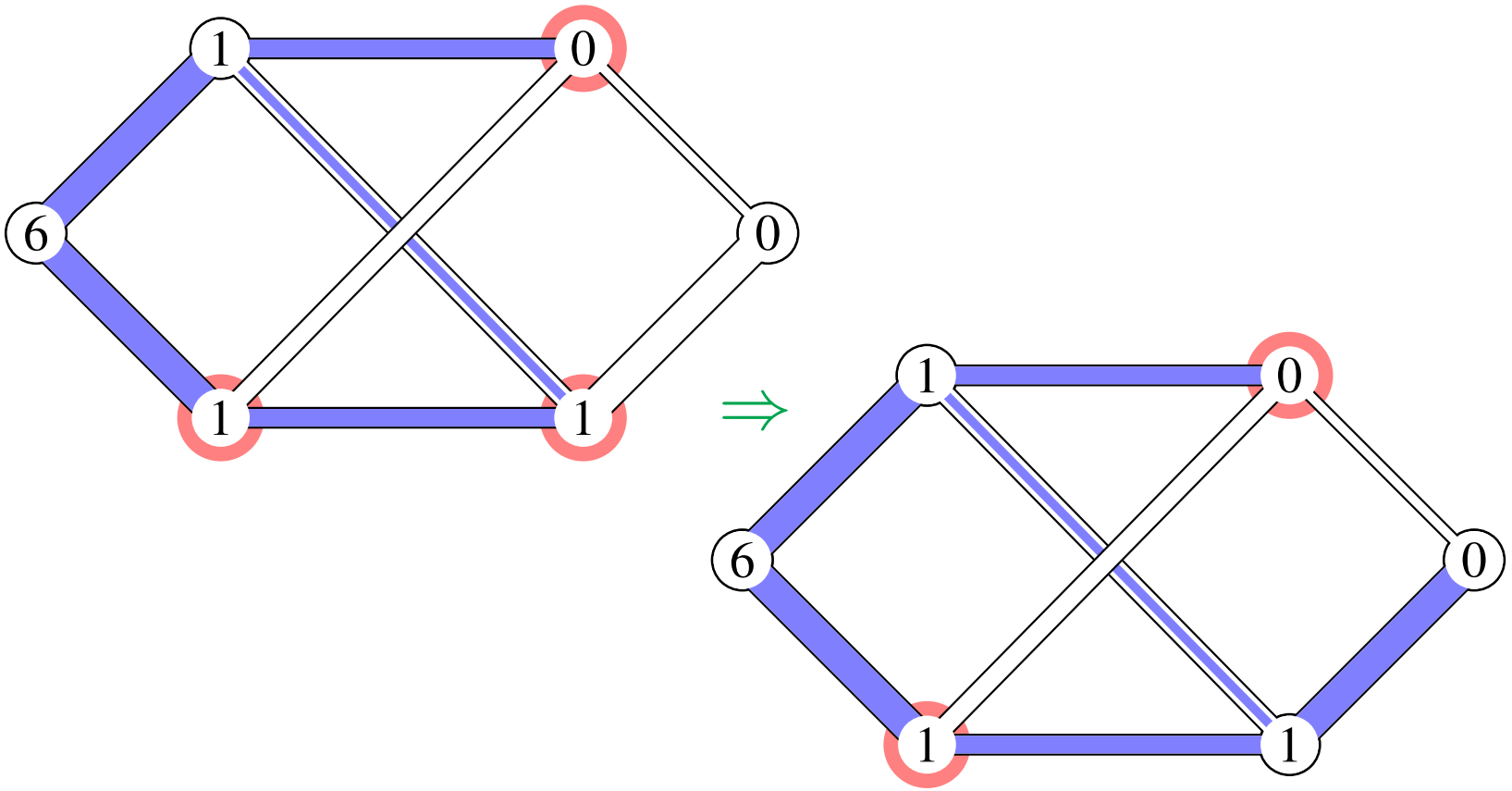
Es wurde ein saturierter Push durchgeführt.

Preflow-Push-Algorithmen



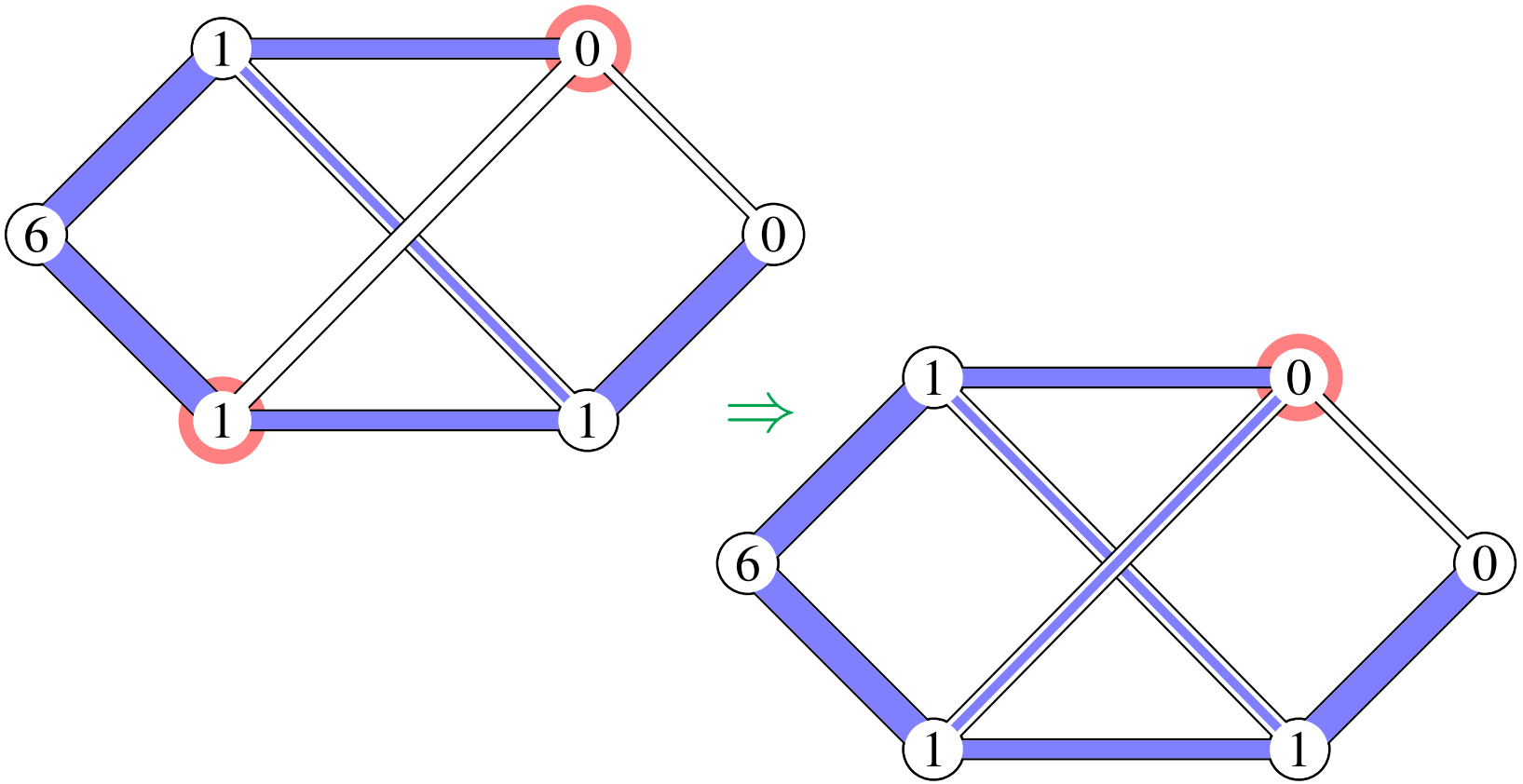
Es wurde eine Lift-Operation durchgeführt.

Preflow-Push-Algorithmen



Es wurde ein saturierter Push durchgeführt.

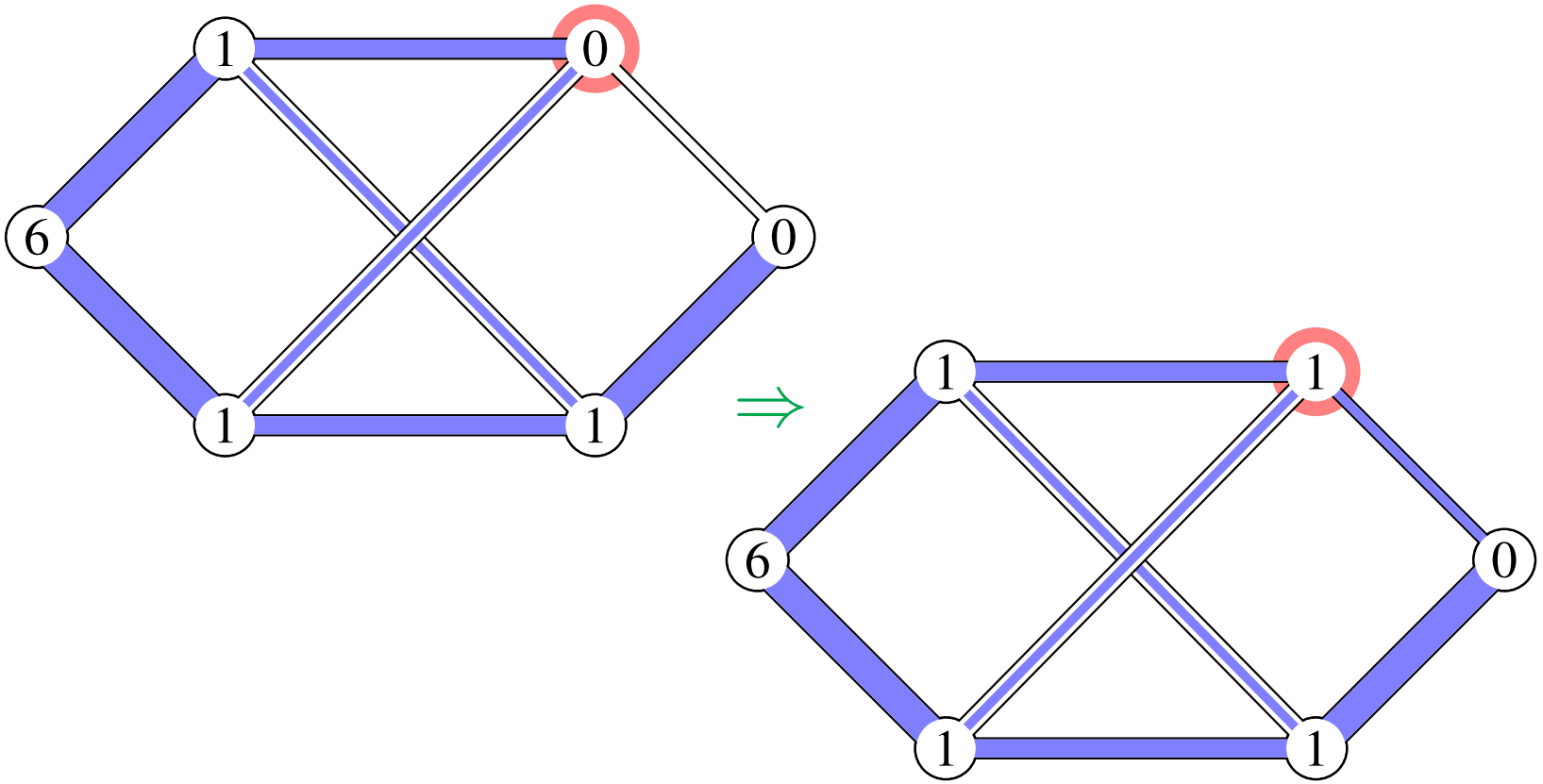
Preflow-Push-Algorithmen



Es wurde ein nicht-saturierter Push durchgeführt.

Jetzt gibt es nur eine Wahlmöglichkeit.

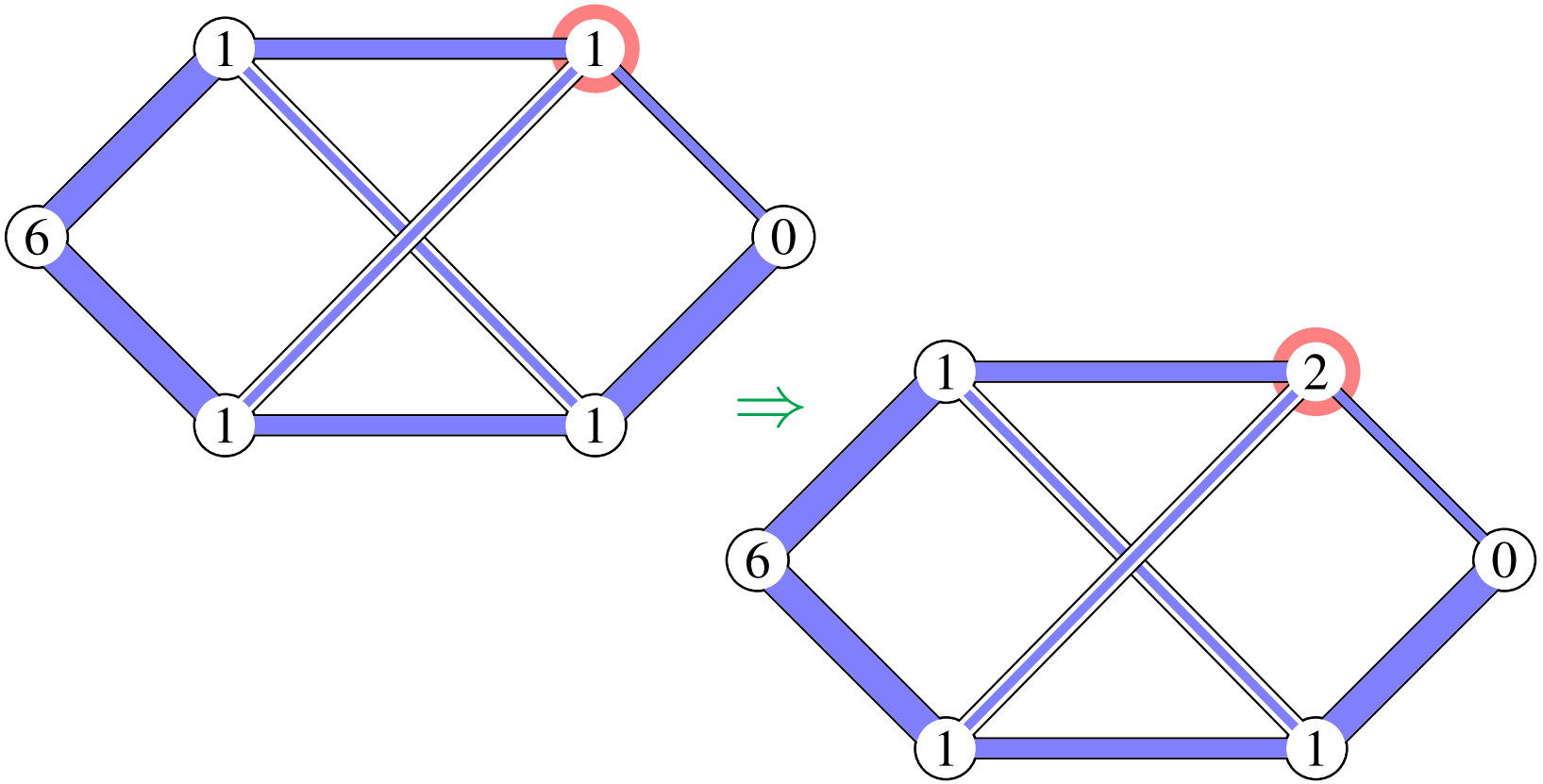
Preflow-Push-Algorithmen



Der Knoten wurde auf Höhe 1 geliftet.

Anschließend wurde ein Push durchgeführt.

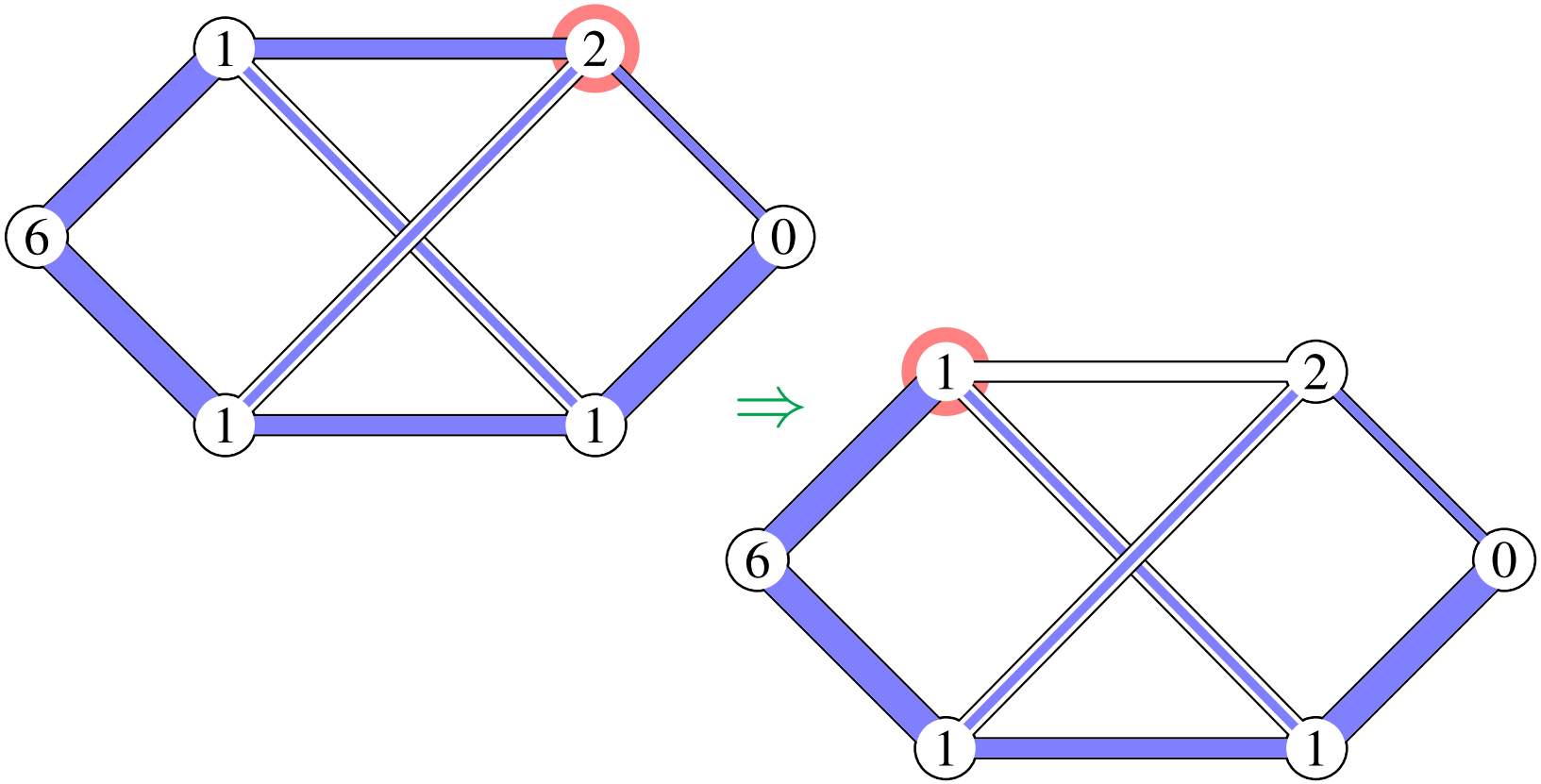
Preflow-Push-Algorithmen



Wir mußten den Knoten auf Höhe 2 liften.

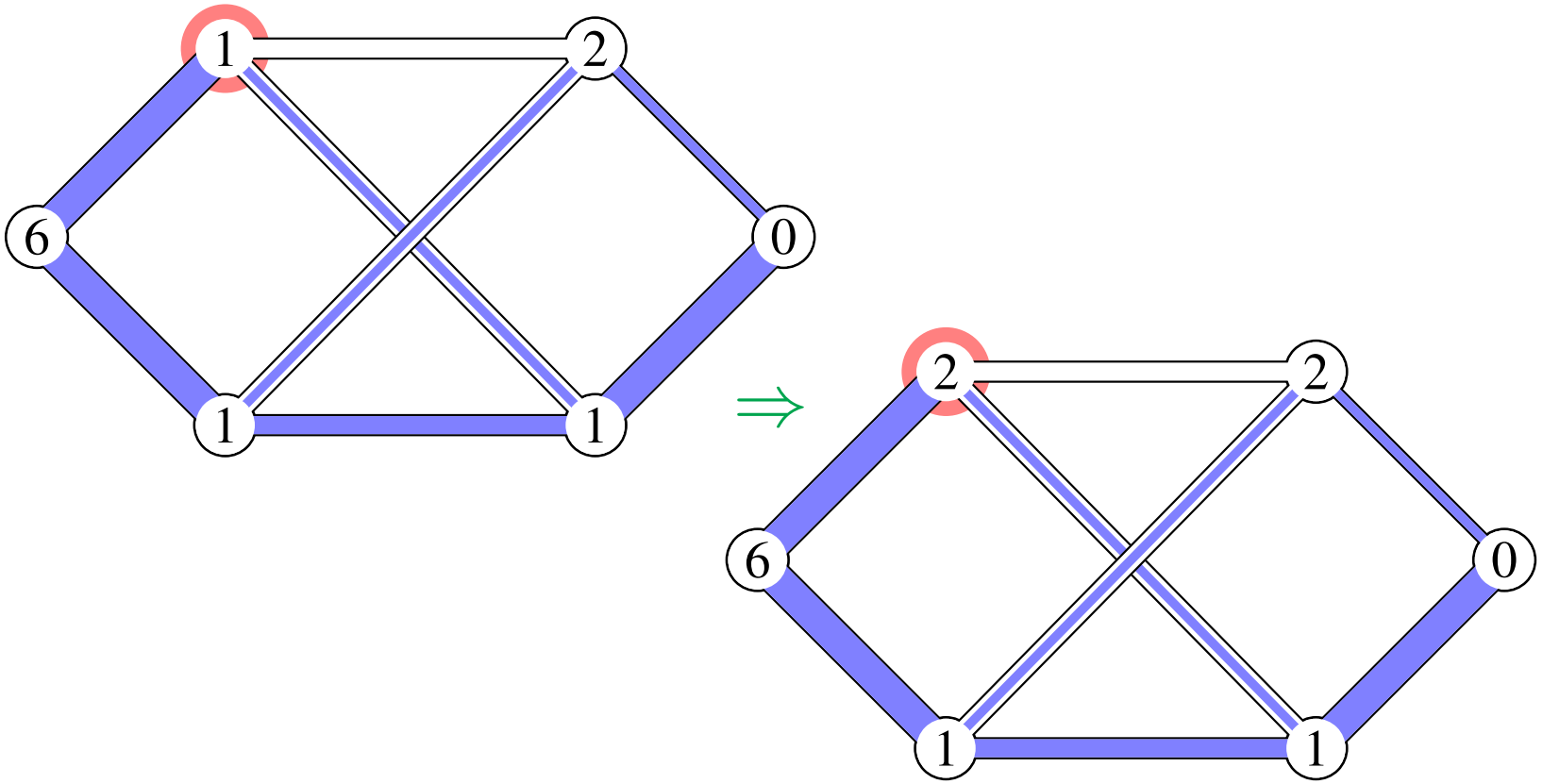
Es gibt jetzt zwei anwendbare Push-Operationen.

Preflow-Push-Algorithmen



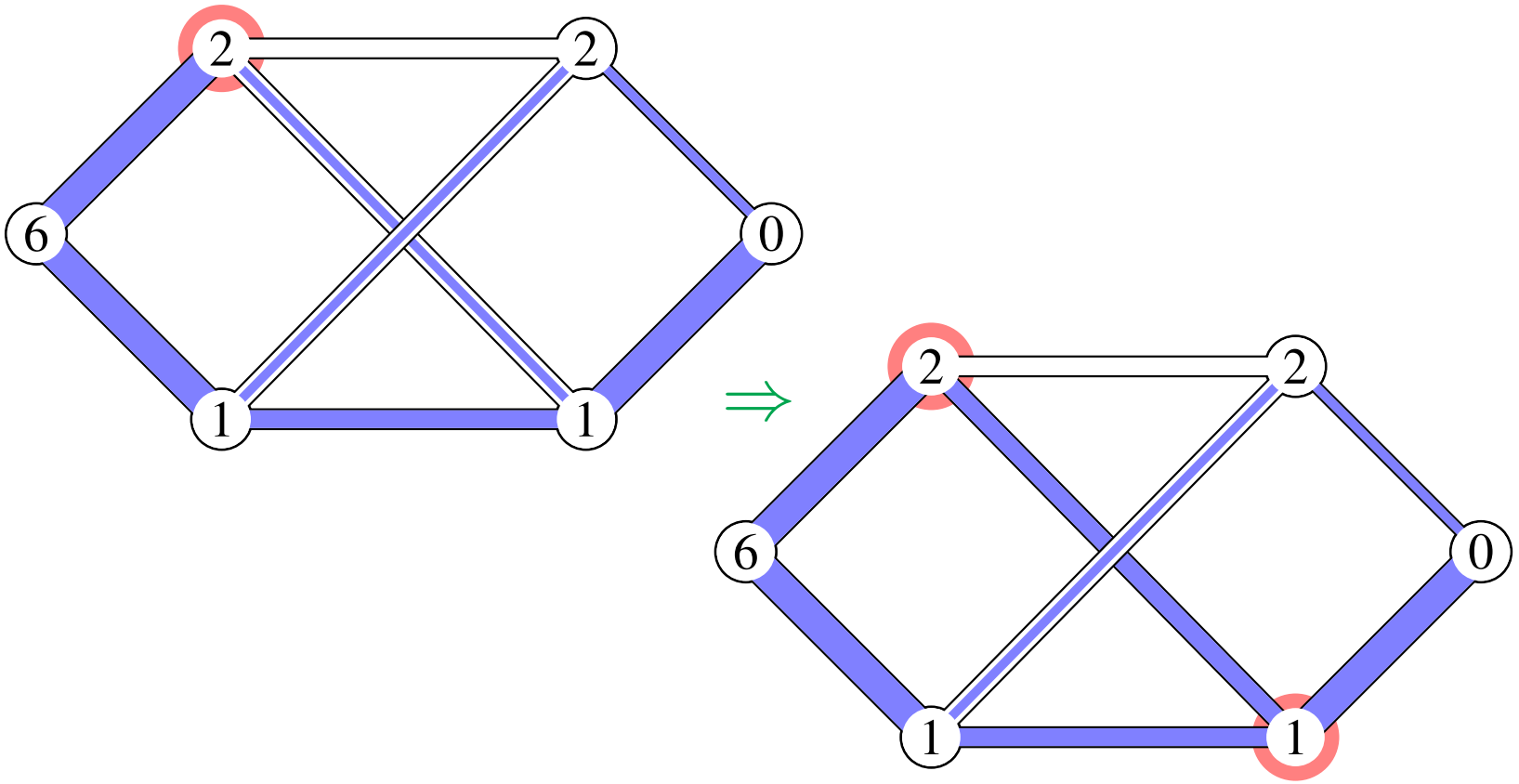
Durch einen saturierenden Push, wurde Fluß ausgelöscht.

Preflow-Push-Algorithmen



Eine Lift-Operation.

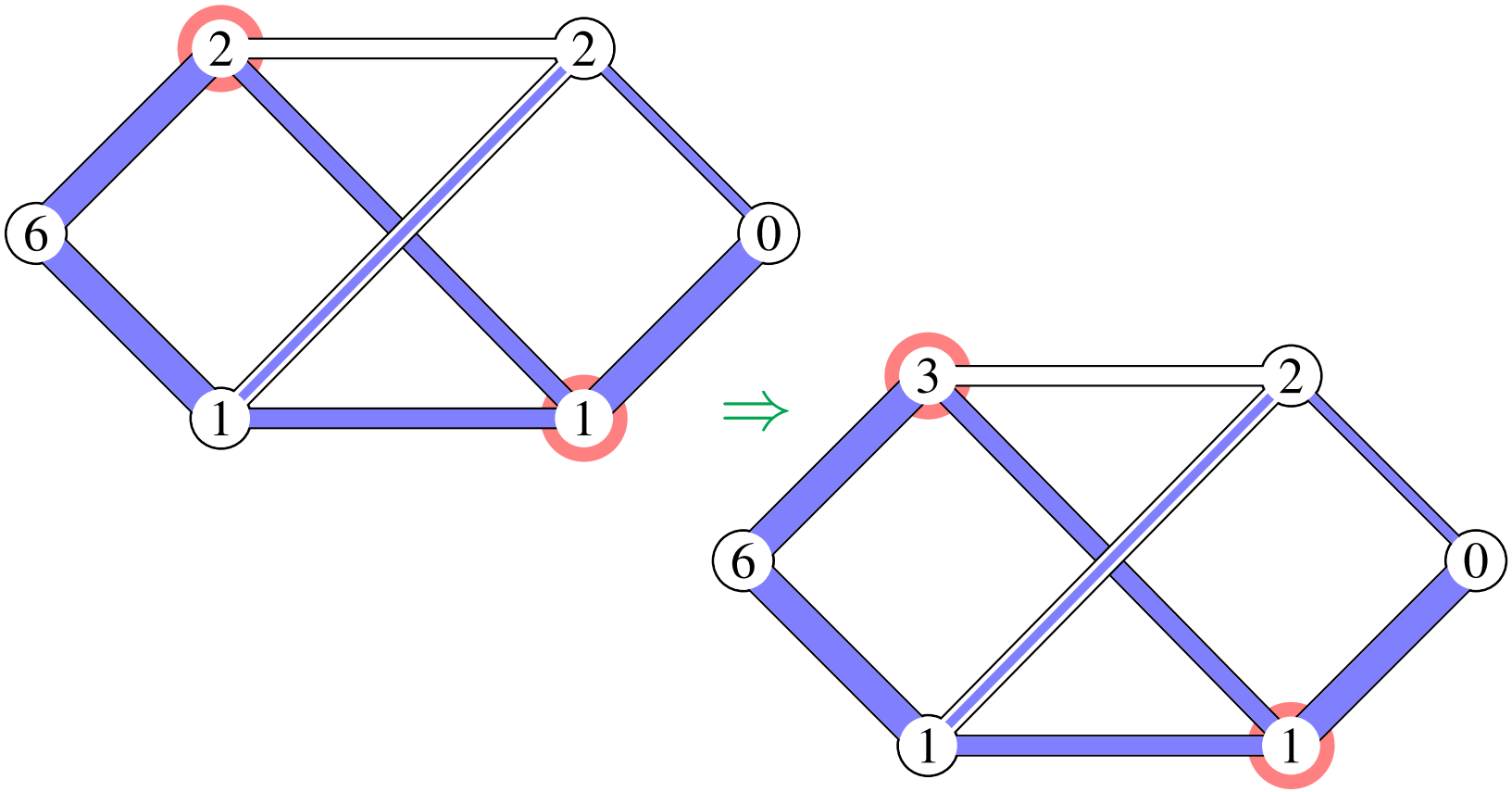
Preflow-Push-Algorithmen



Es wurde ein saturierter Push durchgeführt.

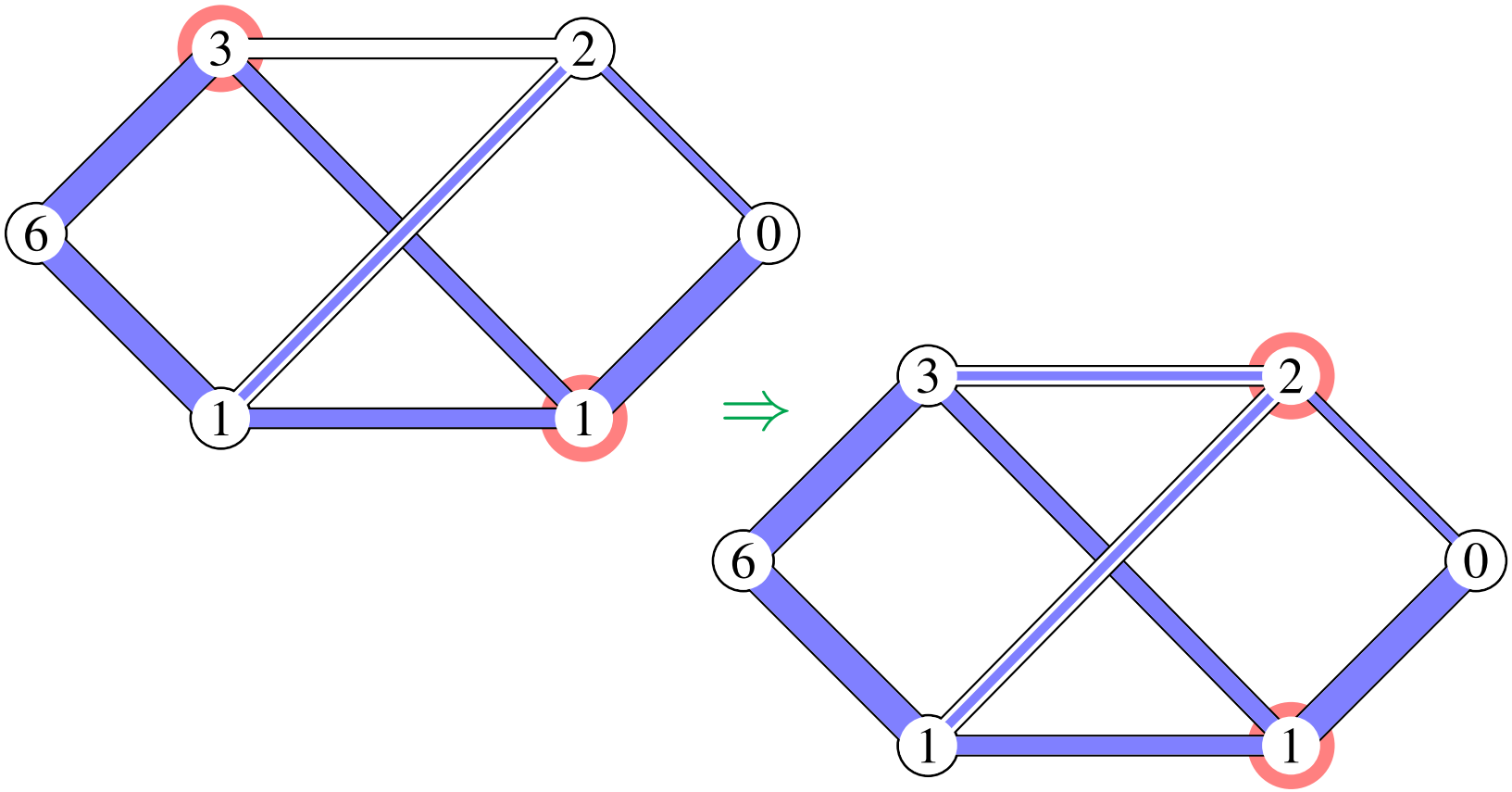
Dadurch gibt es jetzt zwei überfließende Knoten.

Preflow-Push-Algorithmen



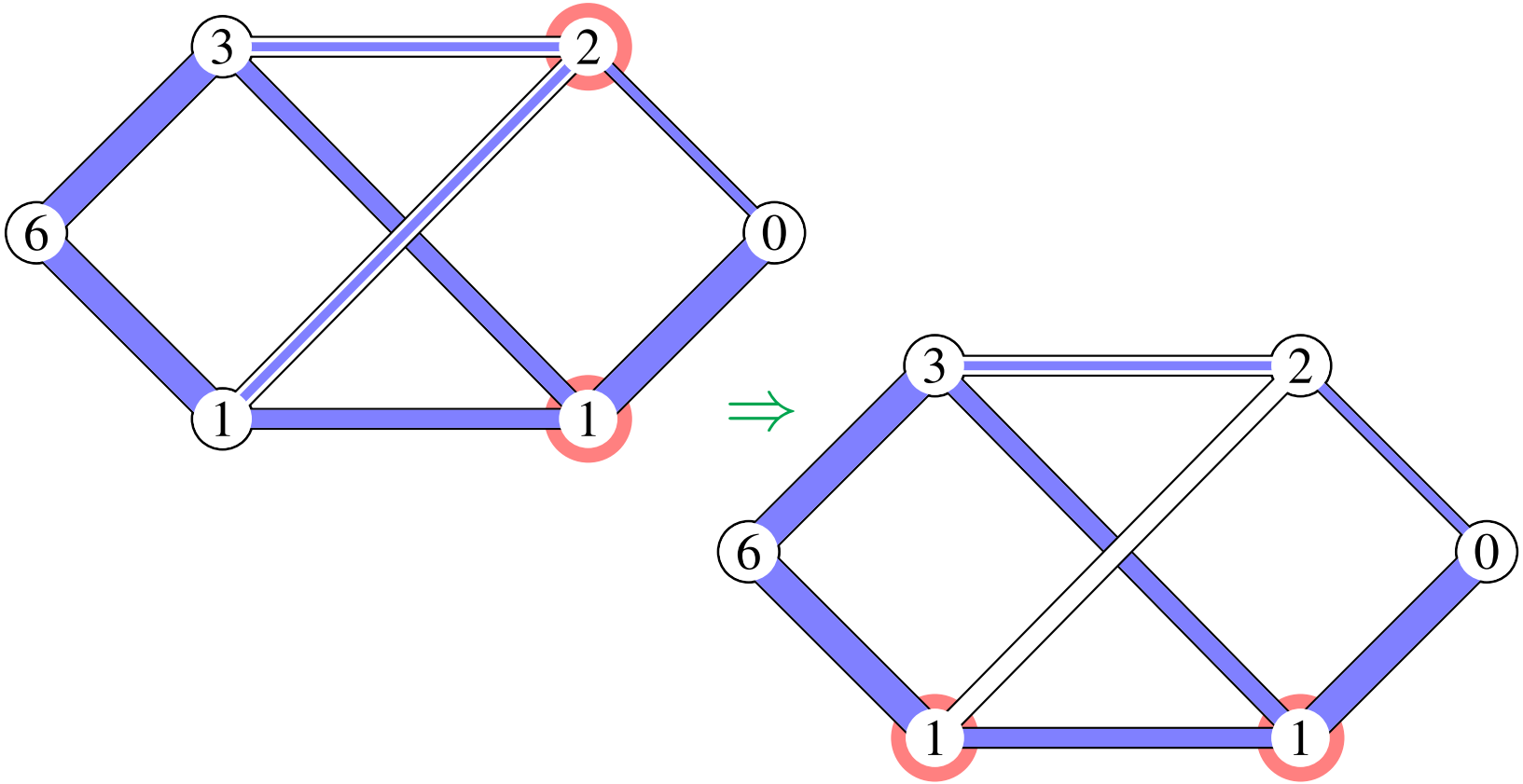
Ein Knoten wurde auf Höhe 3 geliftet.

Preflow-Push-Algorithmen



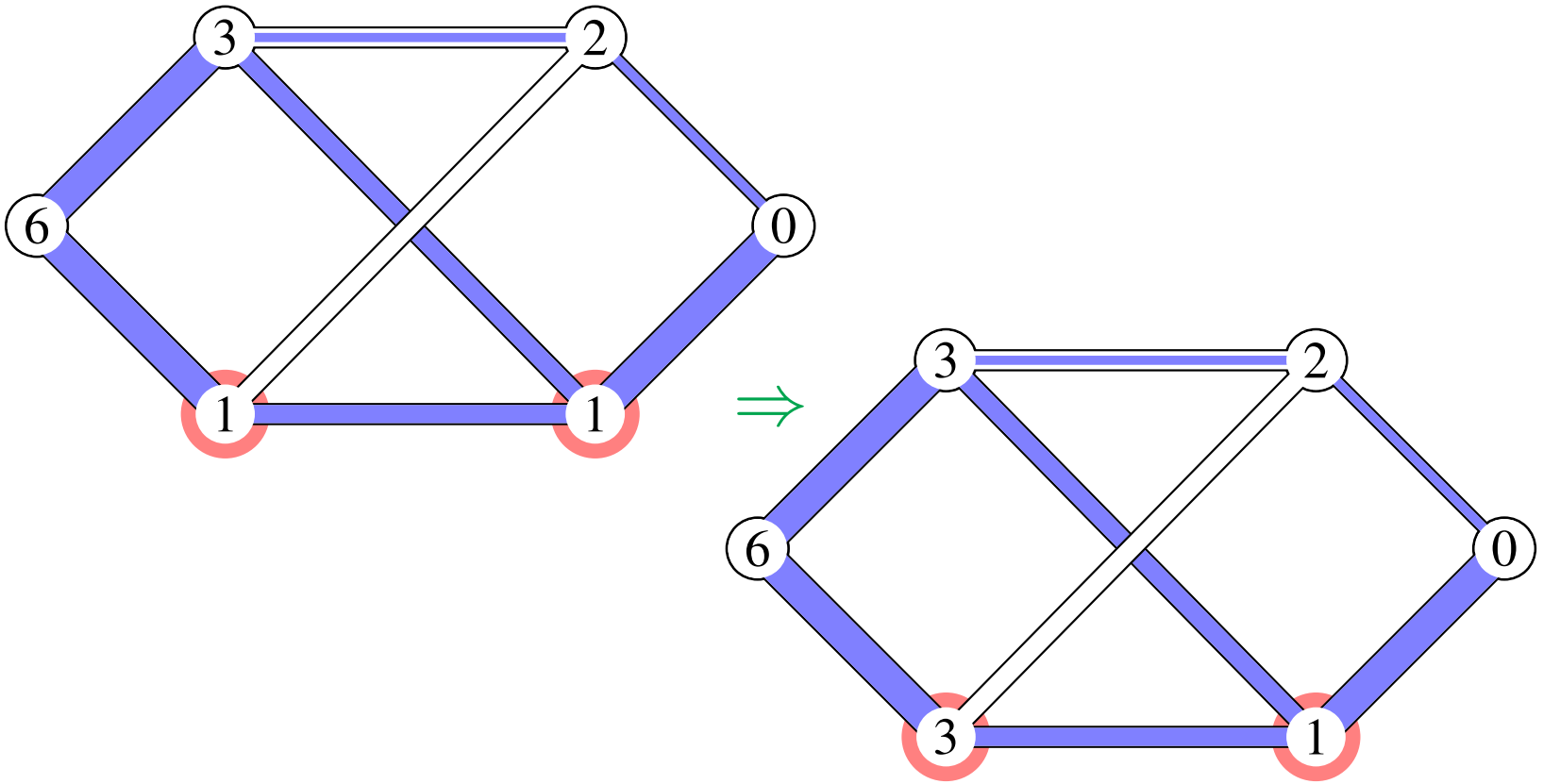
Es wurde ein nicht-saturierter Push durchgeführt.

Preflow-Push-Algorithmen

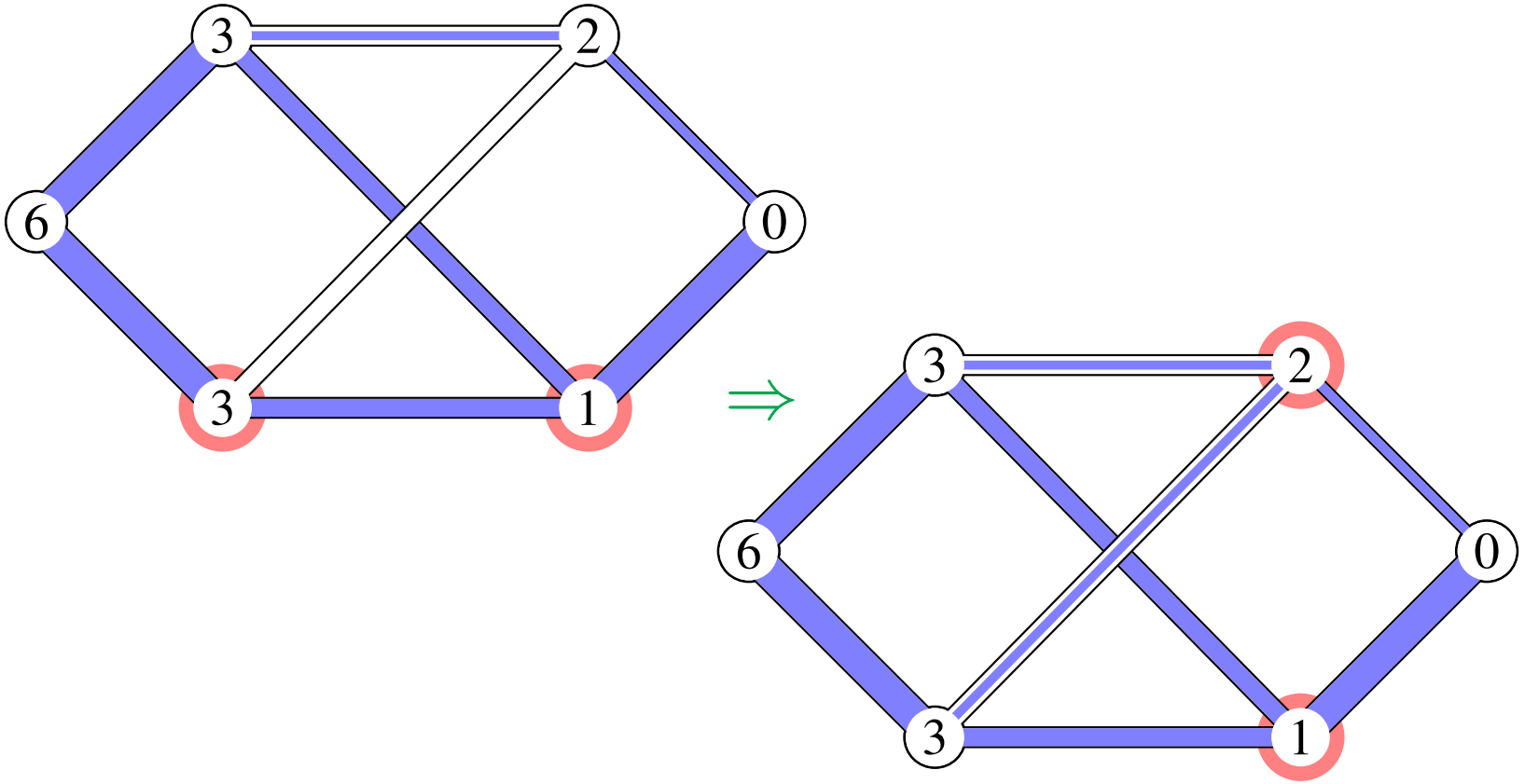


Es wurde ein saturierter Push durchgeführt.

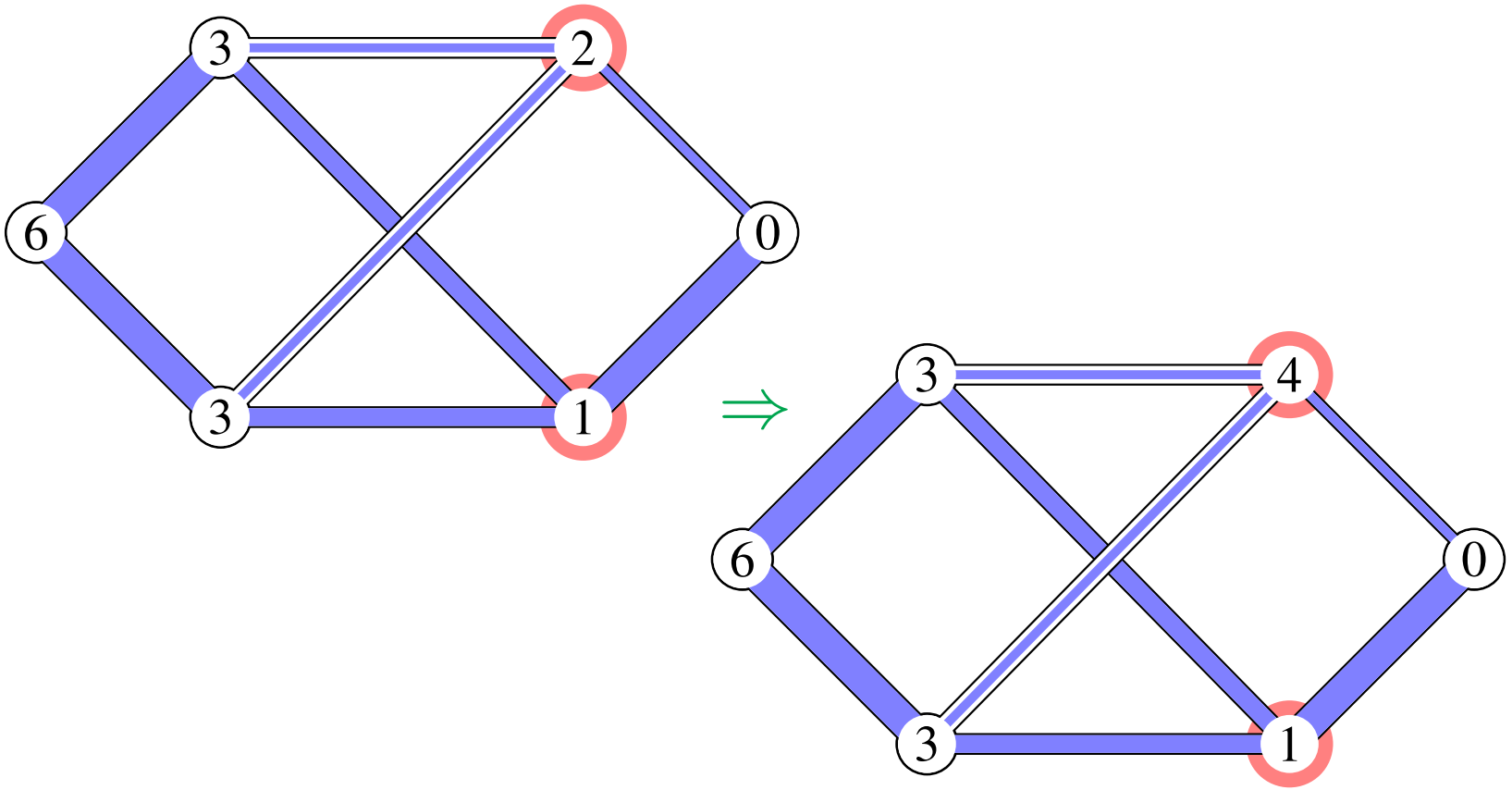
Preflow-Push-Algorithmen



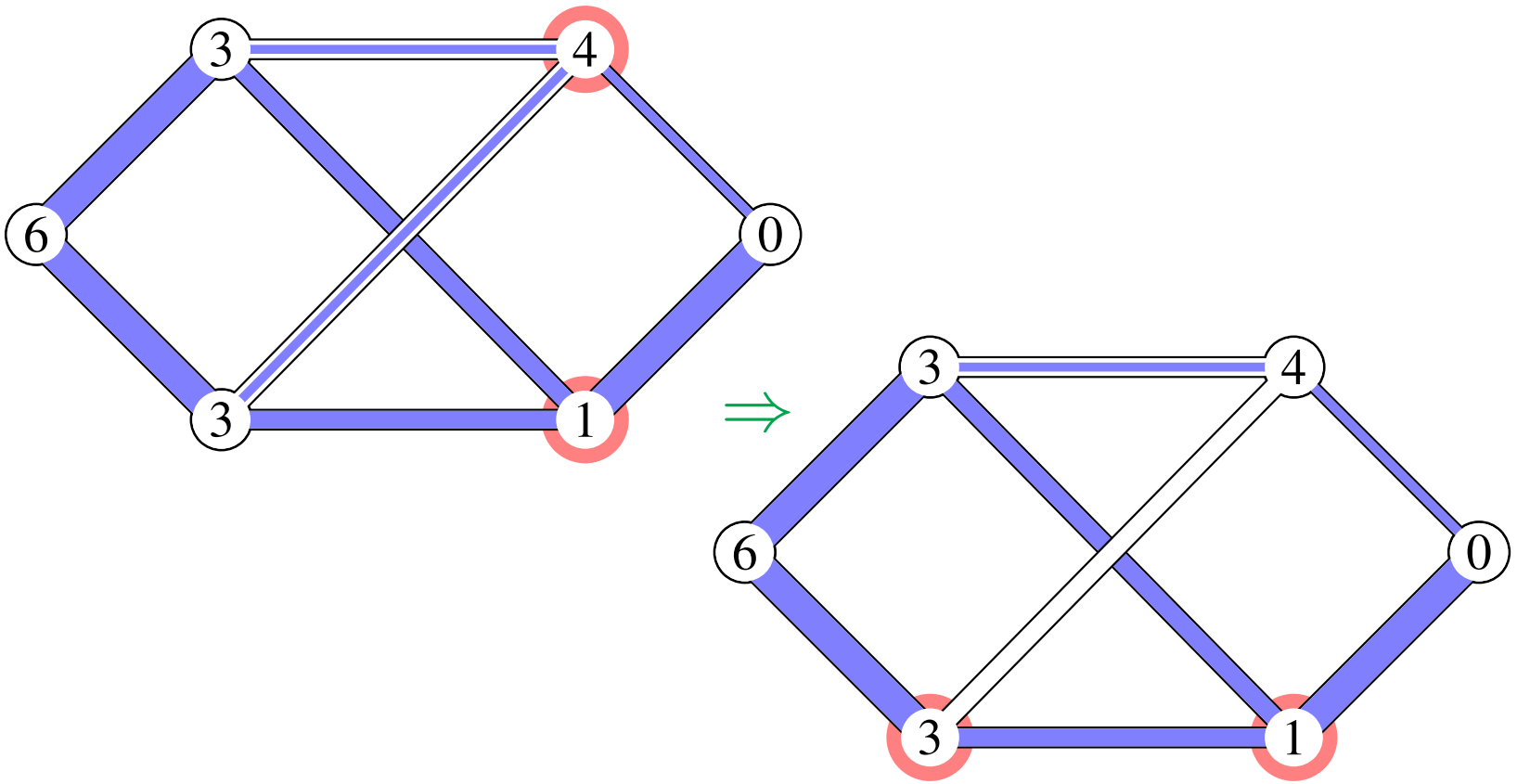
Preflow-Push-Algorithmen



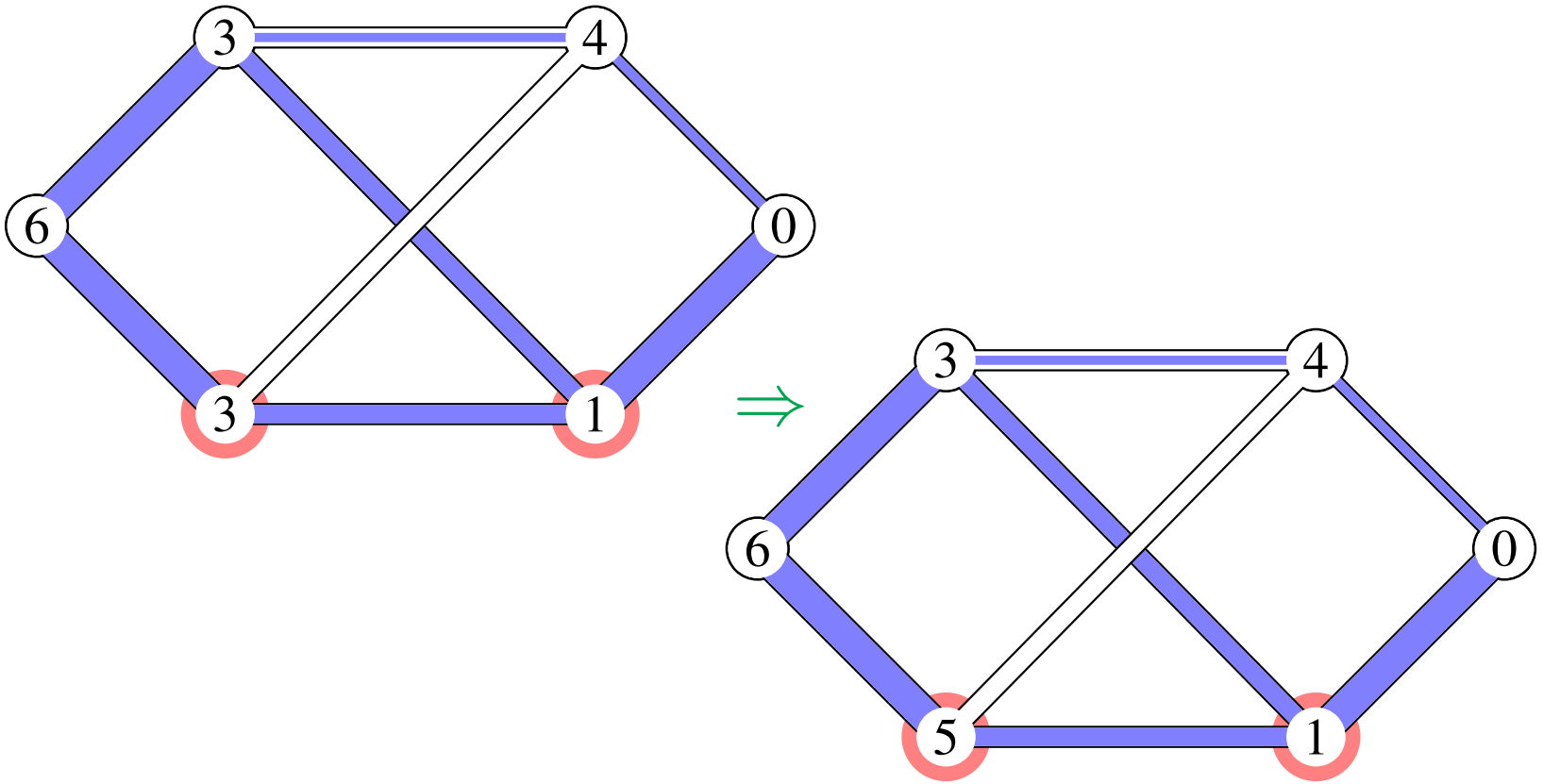
Preflow-Push-Algorithmen



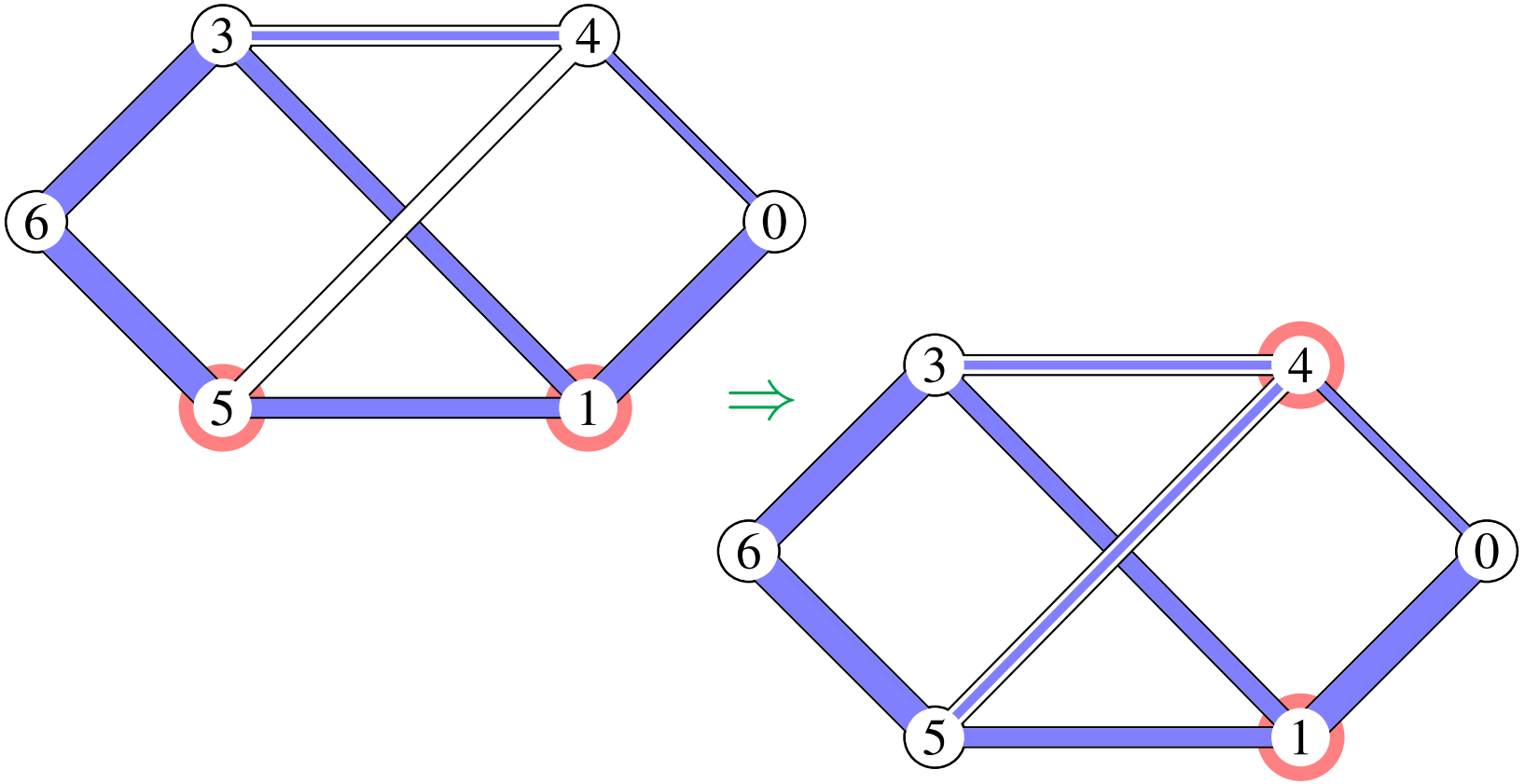
Preflow-Push-Algorithmen



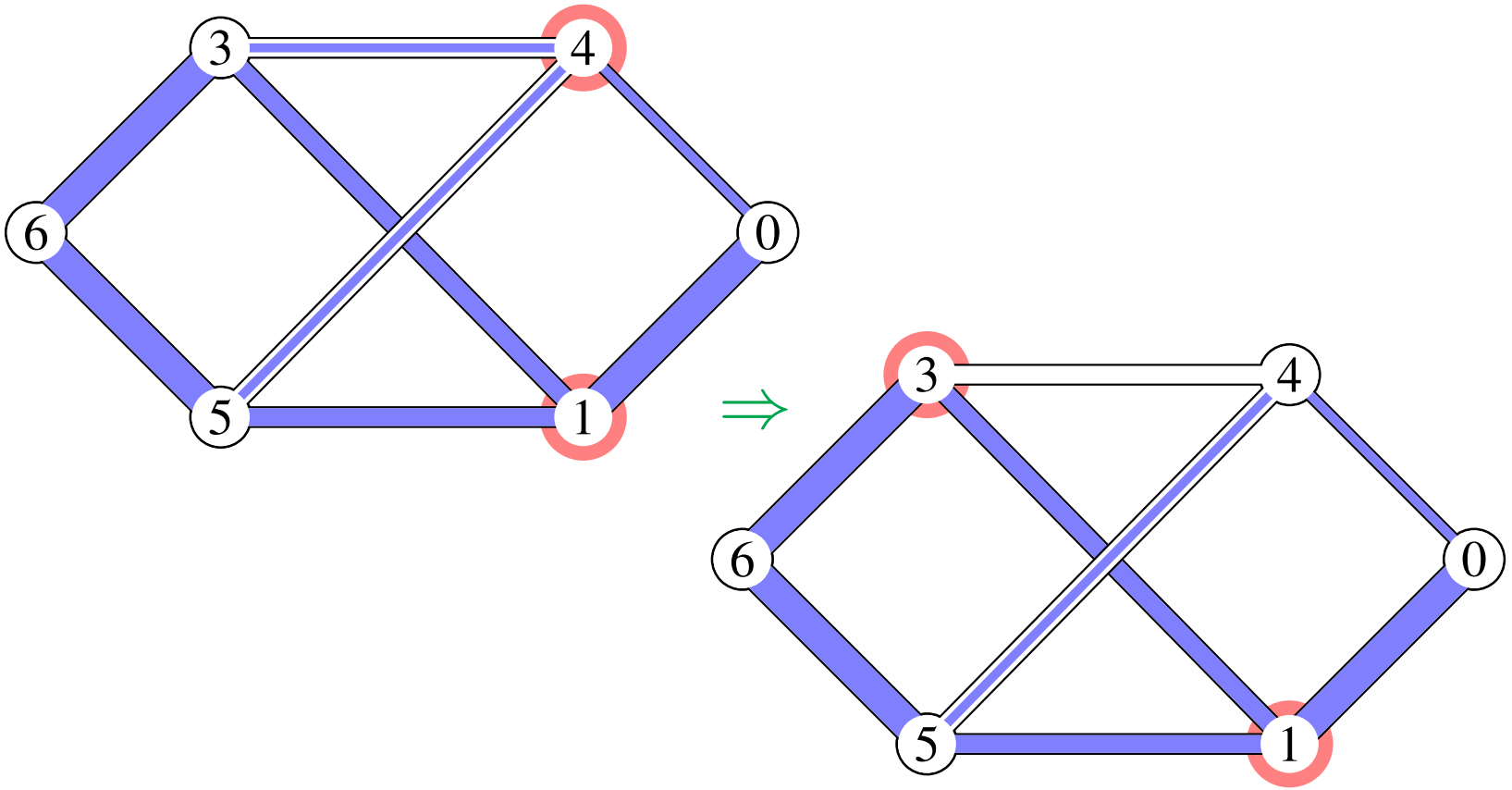
Preflow-Push-Algorithmen



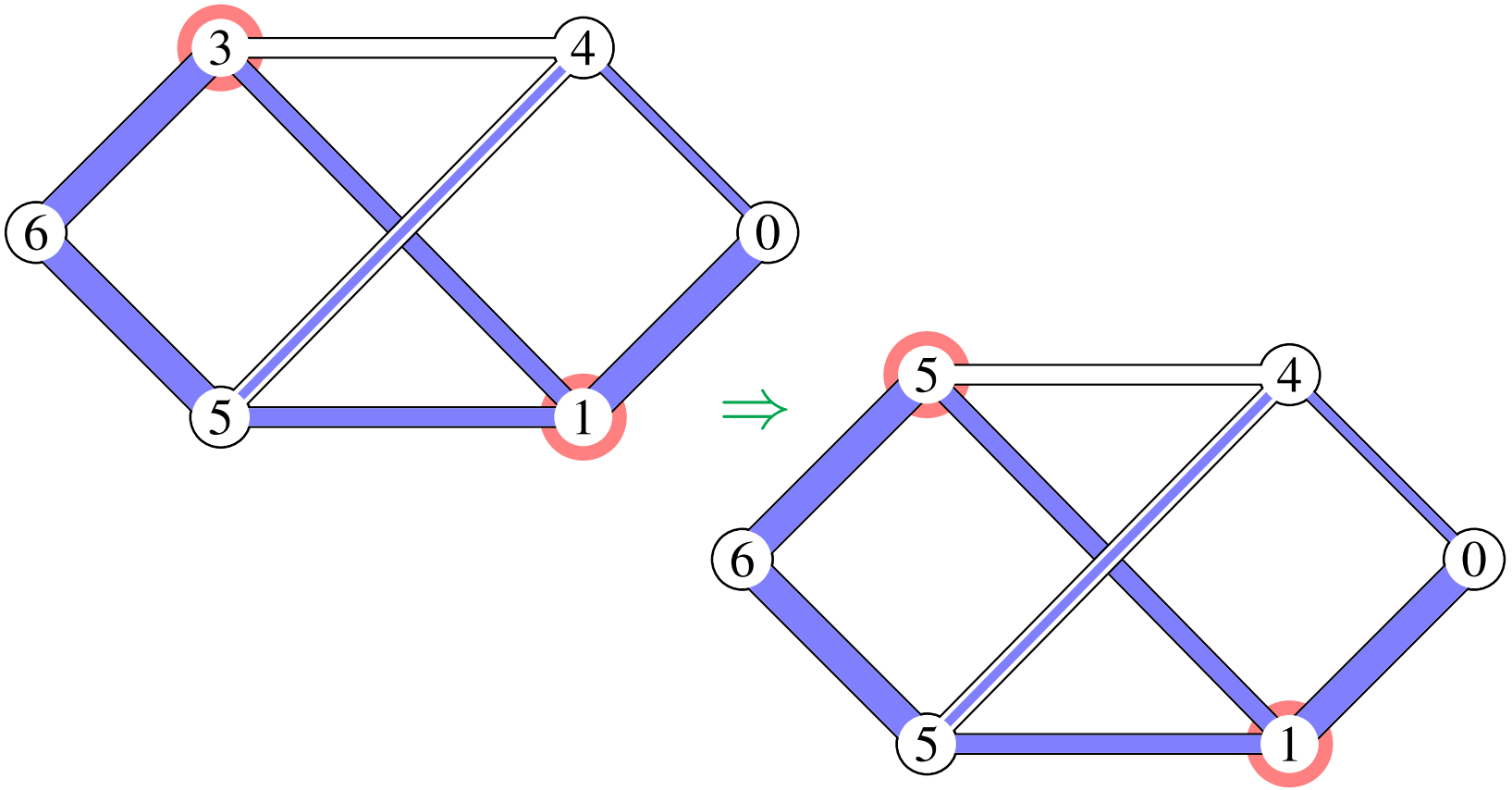
Preflow-Push-Algorithmen



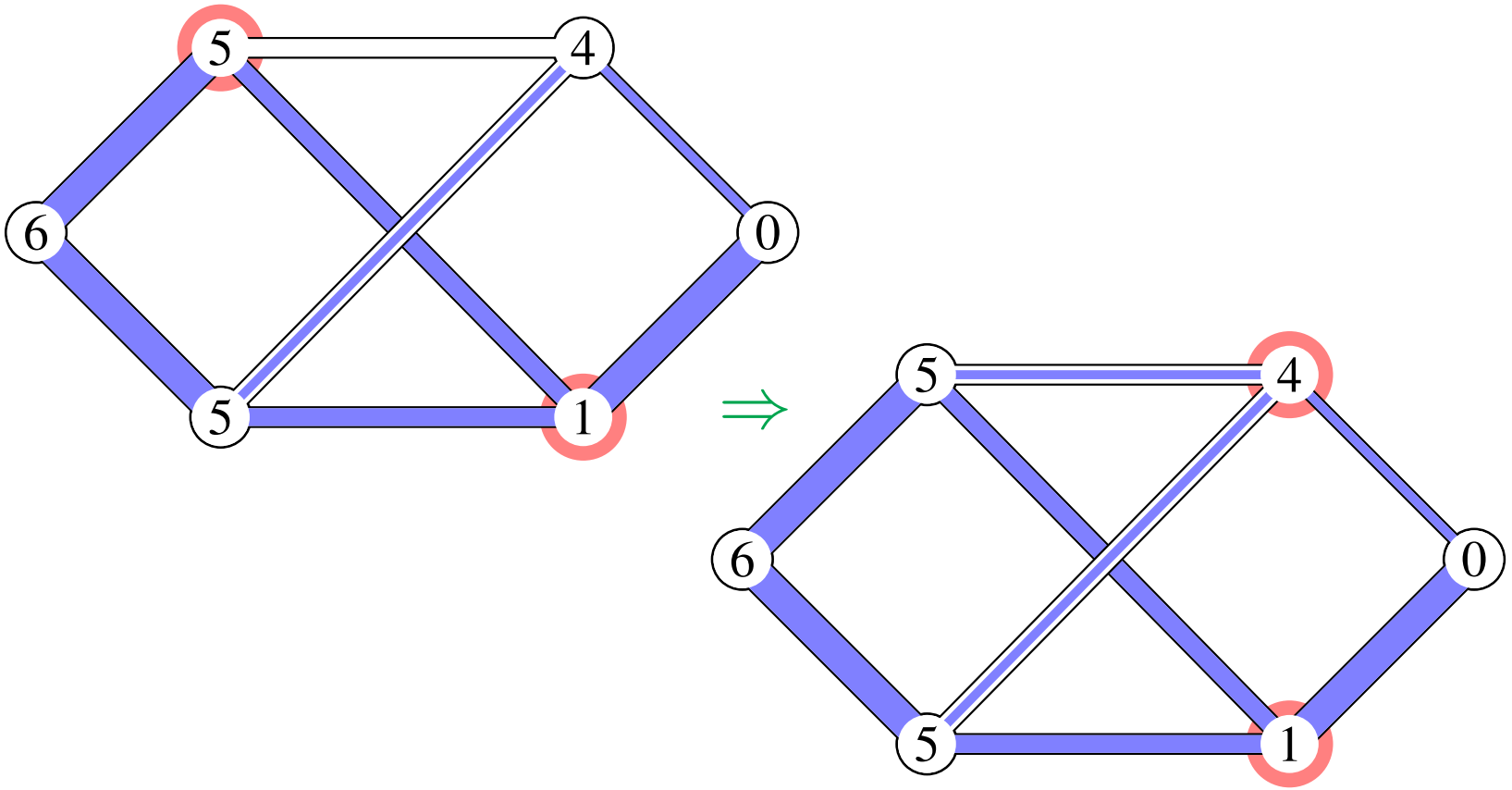
Preflow-Push-Algorithmen



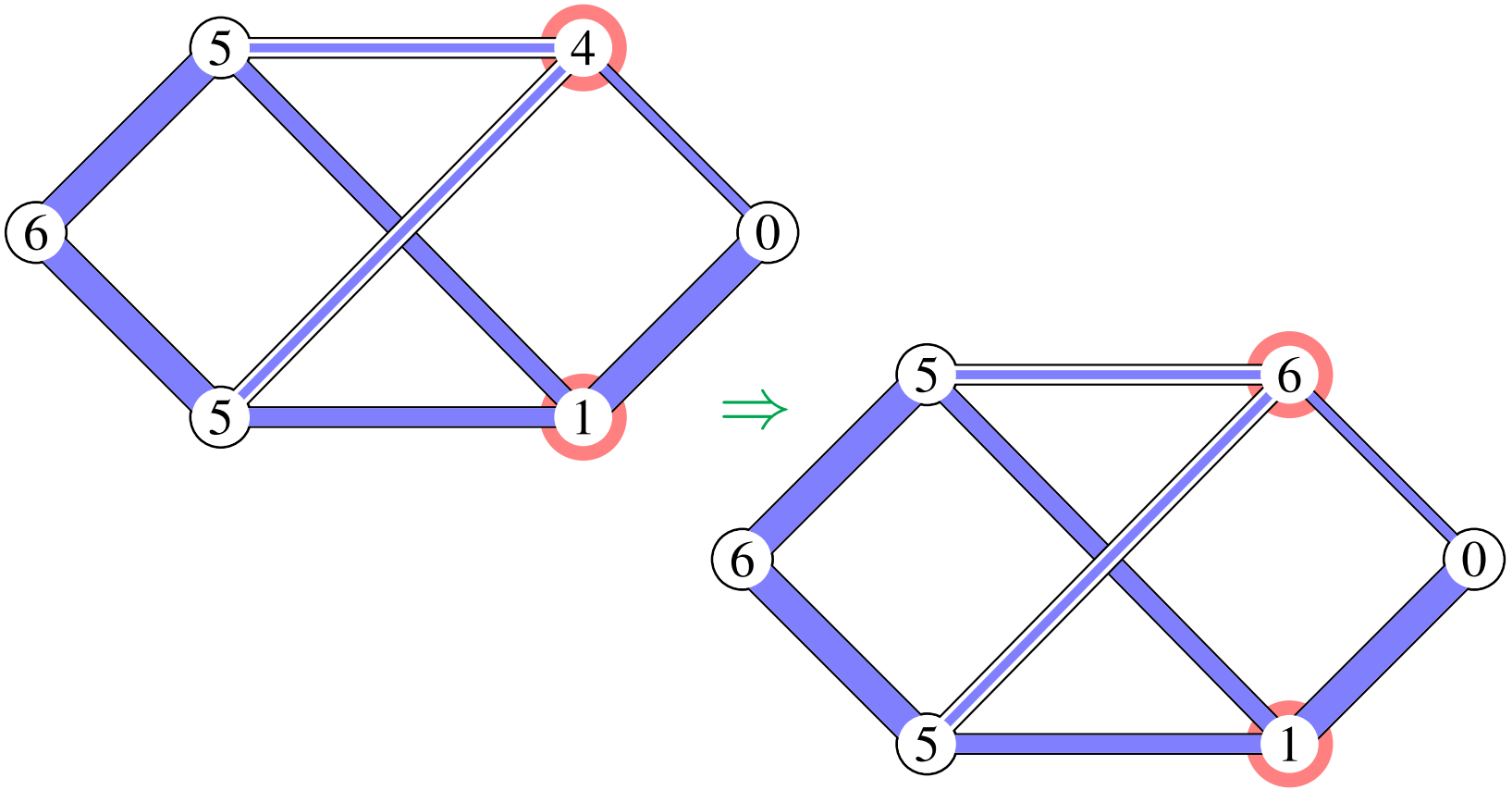
Preflow-Push-Algorithmen



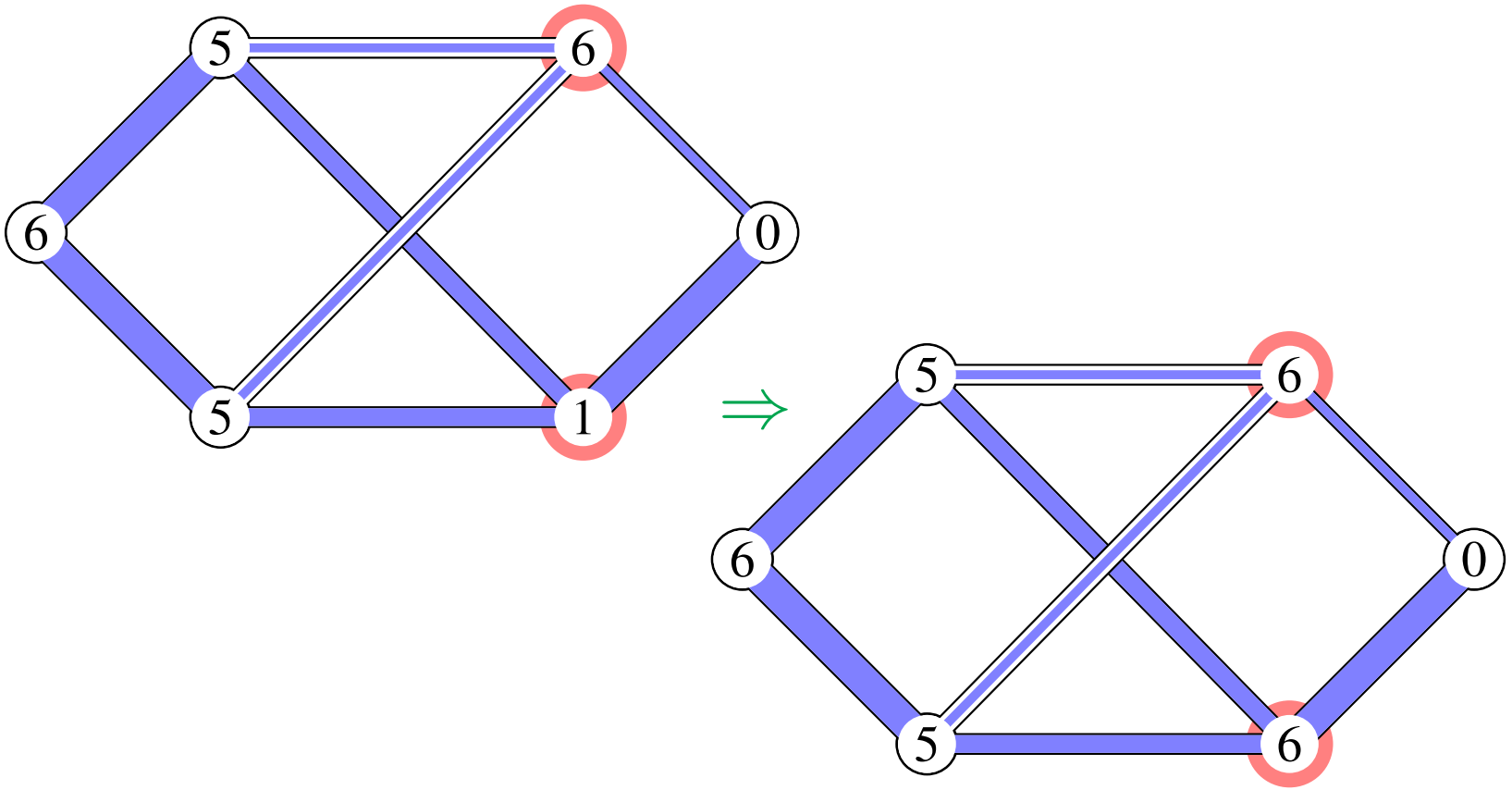
Preflow-Push-Algorithmen



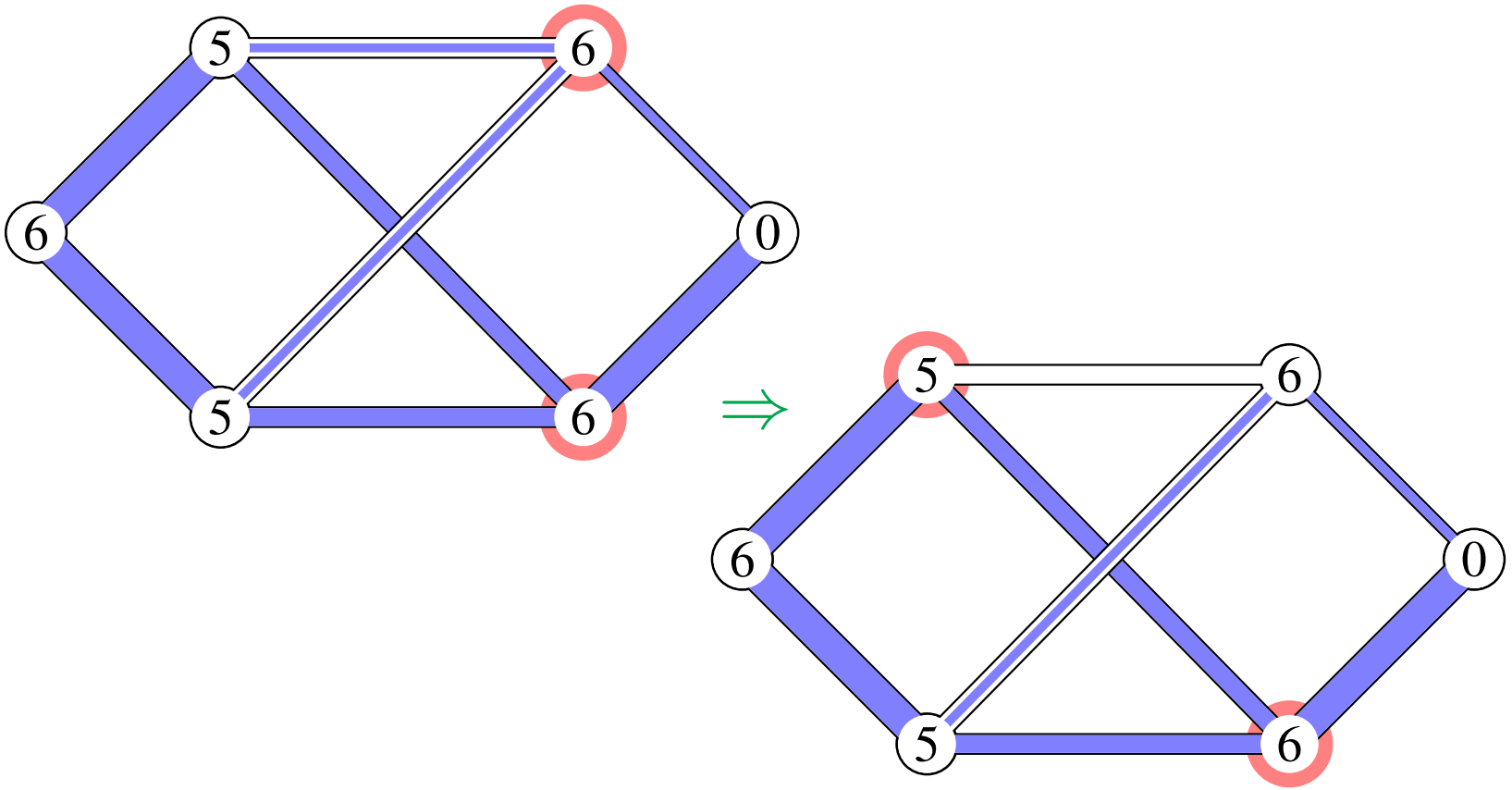
Preflow-Push-Algorithmen



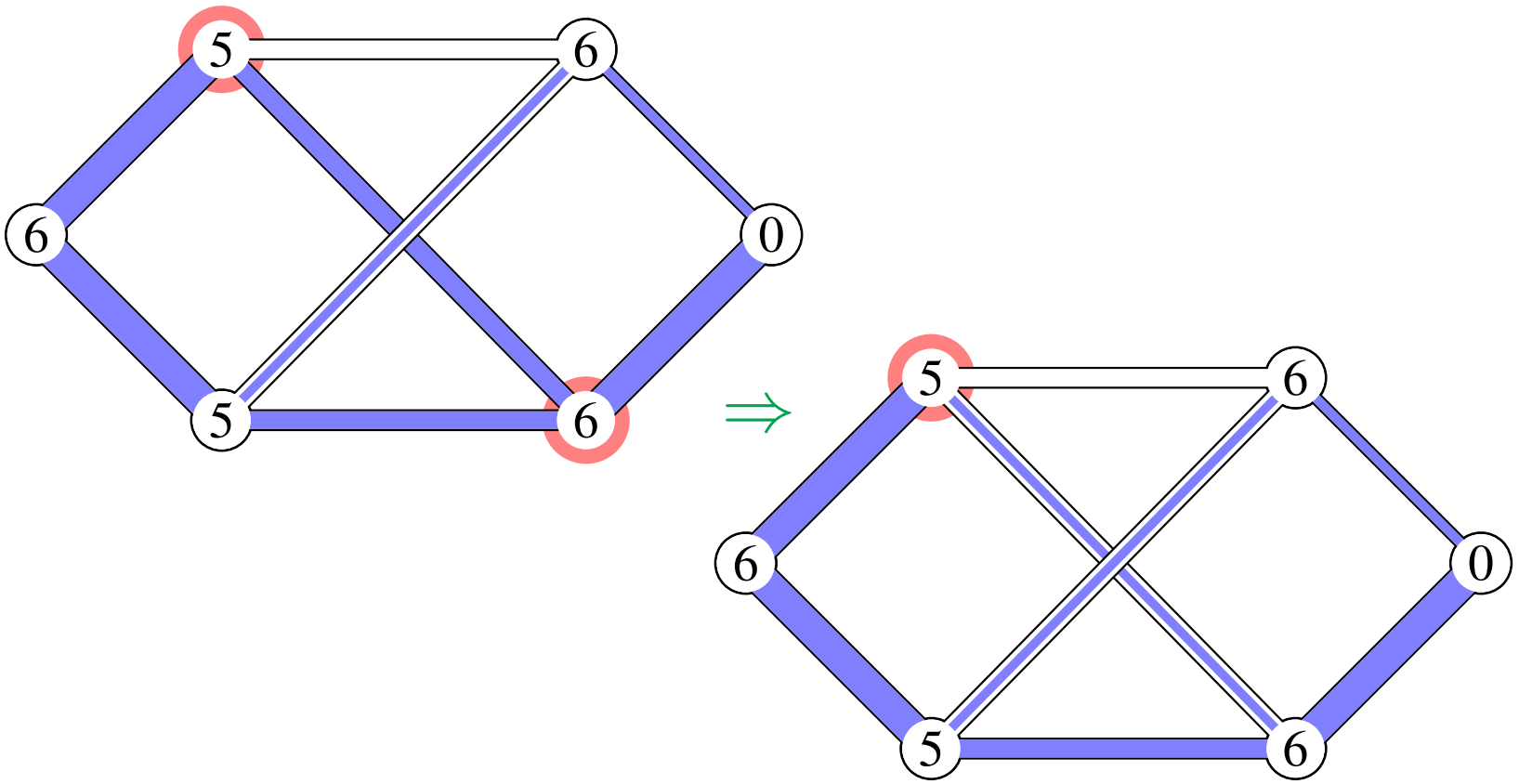
Preflow-Push-Algorithmen



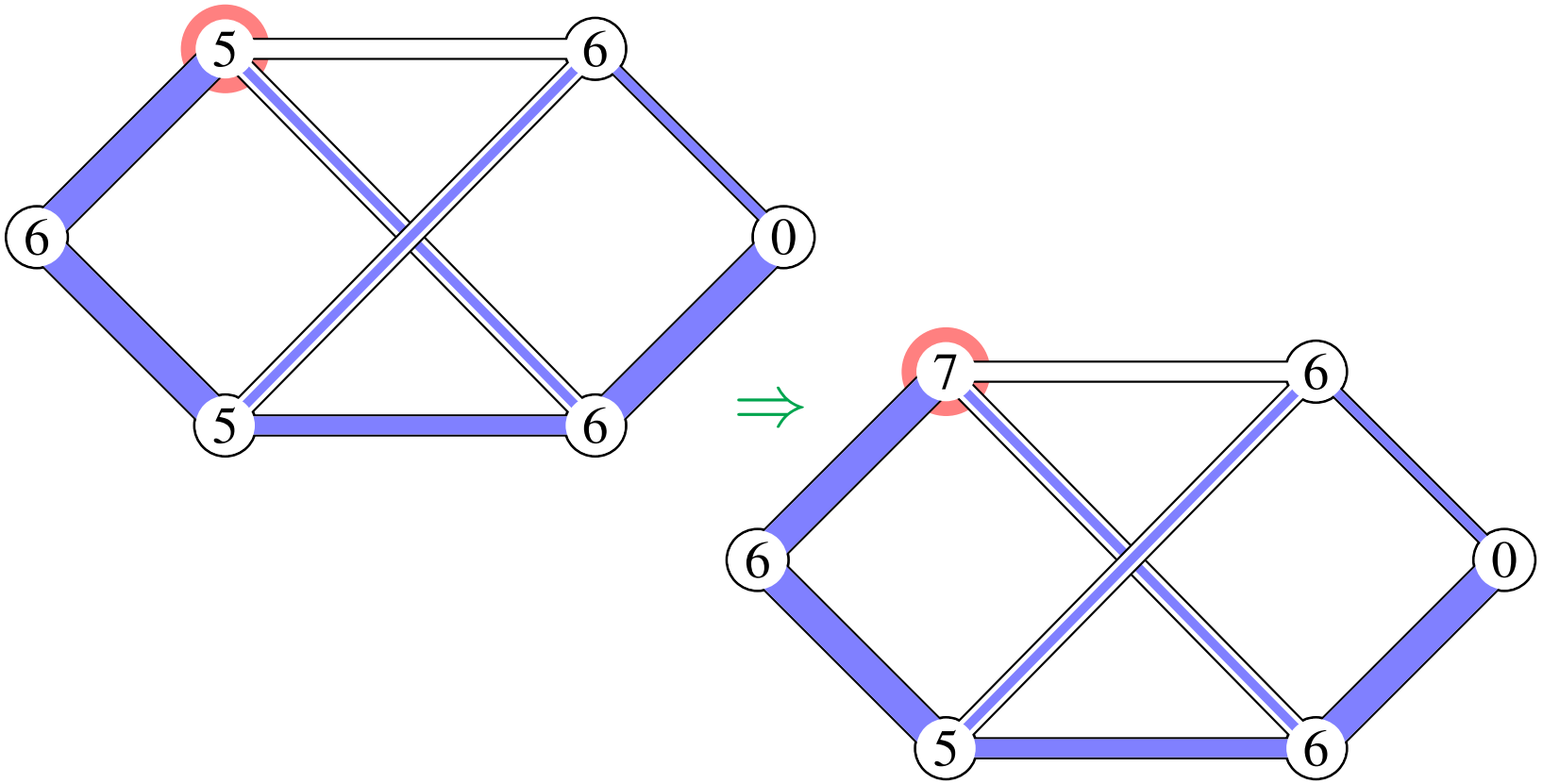
Preflow-Push-Algorithmen



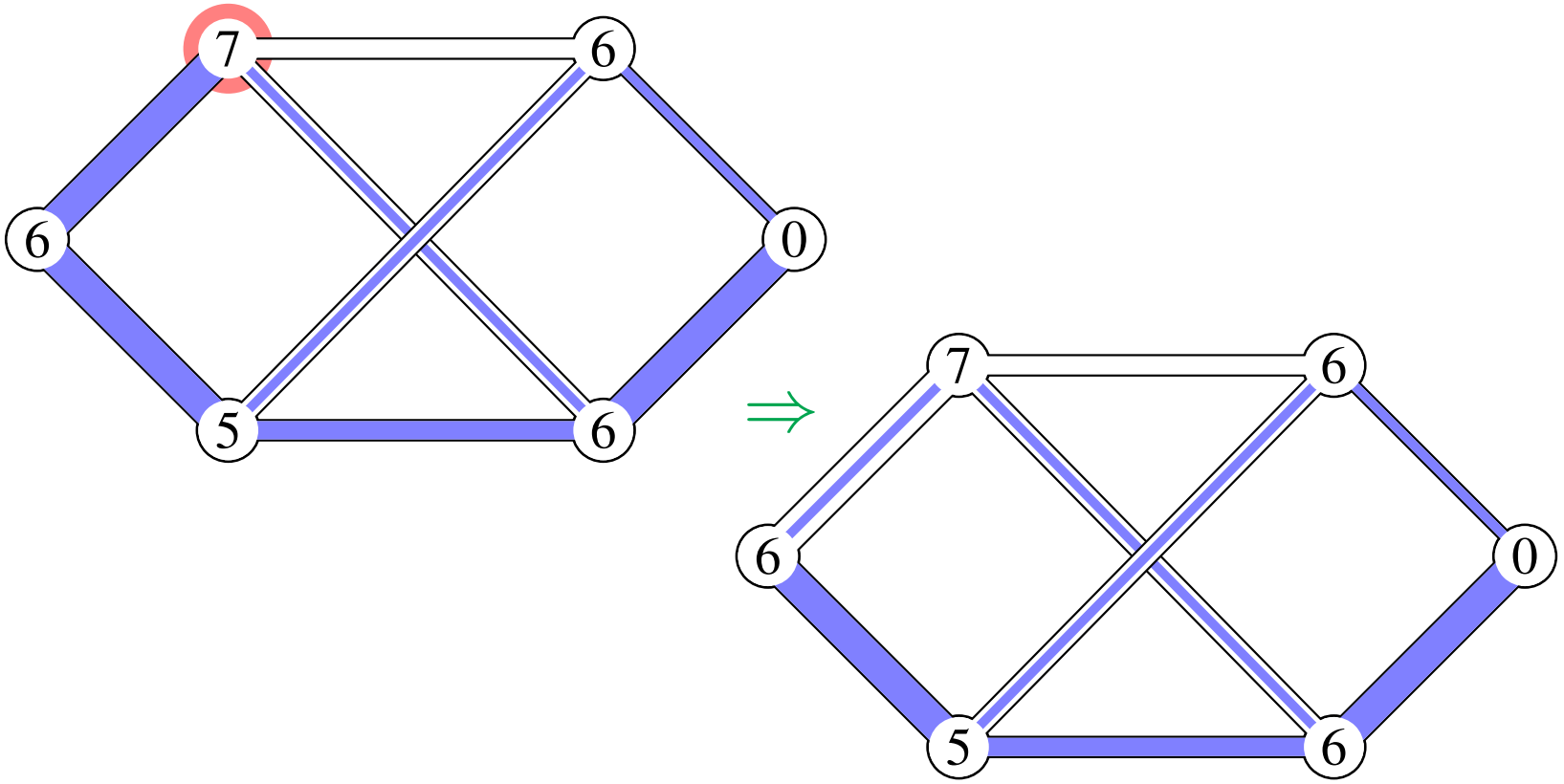
Preflow-Push-Algorithmen



Preflow-Push-Algorithmen



Preflow-Push-Algorithmen



- Es gibt keinen überfließenden Knoten mehr.
- Der „preflow“ ist jetzt ein echter (maximaler) Fluß.

Preflows

Ein *preflow* ist eine Funktion $f : V \times V \to \mathbf{R}$, die Paare von Knoten auf reelle Zahlen abbildet und diese Bedingungen erfüllt:

- *Zulässigkeit:* Für $u, v \in V$ gilt $f(u, v) \leq c(u, v)$.
- *Symmetrie:* Für $u, v \in V$ gilt $f(u, v) = -f(v, u)$.
- *statt Flußerhaltung:* Für $u \in V - \{s, t\}$ gilt $f(V, u) \geq 0$.

Wir definieren $e(u) = f(V, u)$, den *Exzeß* von u.

Falls $e(u) > 0$ dann ist u *überfließend*.

Höhenfunktionen

Definition

Sei $G = (V, E)$ ein s-t-Netzwerk und f ein Preflow auf G.

Eine Funktion $h\colon V \to \mathbf{N}$ ist eine *Höhenfunktion*, wenn

- $h(s) = |V|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1$ für alle $(u, v) \in E_f$

„Eine Kante im Residualnetzwerk führt höchstens um 1 nach unten.“

Die Push-Operation $PUSH(u, v)$

Vorbedingungen:

- $e(u) > 0$
- $c_f(u, v) > 0$
- $h(u) = h(v) + 1$

$$
\begin{aligned}
&d_f(u, v) := \min\{e(u), c_f(u, v)\} \\
&f(u, v) := f(u, v) + d_f(u, v) \\
&f(v, u) := -f(u, v) \\
&e(u) := e(u) - d_f(u, v) \\
&e(v) := e(v) + d_f(u, v)
\end{aligned}
$$

Die Lift-Operation $LIFT(u)$

Vorbedingungen:

- $e(u) > 0$
- $h(u) \leq h(v)$ falls $(u, v) \in E_f$

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

Der einfache Preflow-Push-Algorithmus

Eingabe: Ein s-t-Netzwerk $G = (V, E)$

for each $u \in V$ **do** $h(u) := 0$, $e(u) := 0$
$h(s) := |V|$
for each $(u, v) \in E$ **do** $f(u, v) := 0$, $f(v, u) := 0$
for each $u \in \{ u \mid (s, u) \in E \}$ **do**
    $f(s, u) := c(s, u)$
    $f(u, s) := -c(s, u)$
    $e(u) := c(s, u)$
while es gibt eine anwendbare Lift-
        oder Push-Operation **do**
    wähle eine Operation aus und wende sie an
return f

Korrektheit

Lemma E

Während der Ausführung des Preflow-Push-Algorithmus ist h eine Höhenfunktion.

Beweis

Induktion über Anzahl der Operationen.

Anfangs führen keine Kanten nach unten ausser von s. Diese sind aber keine Residualkanten.

Beweis (Fortsetzung)

1. $LIFT(u)$: Es gibt kein Problem mit einer Kante, wenn ihr Endpunkt gehoben wird.

   Sei also $(u, v) \in E_f$. Vorher galt $h(u) \leq h(v)$. Danach muß $h(u) \leq h(v) + 1$ gelten.

2. $PUSH(u, v)$: Es kann eine neue Kante $(v, u) \in E_f$ entstehen. Vorher galt aber $h(u) = h(v) + 1$, also führt die neue Kante sogar um eins nach oben.

Korrektheit

Lemma F

Sei $G = (V, E)$ ein s-t-Netzwerk, f ein Preflow und h eine Höhenfunktion auf G.

Dann gibt es keinen s-t-Pfad in G_f.

Beweis

Gäbe es einen s-t-Pfad, hätte er höchstens Länge $|V| - 1$. Jede Kante führt um höchstens eins nach unten. Es startet auf Höhe $|V|$.

Also kann er nicht auf Höhe 0 enden. Widerspruch.

Korrektheit

Theorem

Der Preflow-Push-Algorithmus berechnet einen maximalen Fluß, falls er terminiert.

Beweis

Wenn er terminiert, gibt es keine überfließende Knoten. Also ist der Preflow sogar ein Fluß.

Wegen Lemma F gibt es keinen augmentierenden Pfad. Also ist der Fluß gemäß Min-Cut-Max-Flow-Theorem maximal. □

Laufzeitanalyse

Lemma G

Es gibt von jedem überfließenden Knoten u einen Pfad nach s in G_f.

Beweis

Nehmen wir an, es gäbe keinen solchen Pfad.

Sei $U = \{\, v \mid v \text{ von } u \text{ in } G_f \text{ erreichbar} \,\}$.

$$\begin{aligned} 0 &\geq f(V-U,U) \text{ sonst Residualkante aus } U \text{ heraus} \\ &= f(V-U,U) + f(U,U) \\ &= f(V,U) = e(U) \end{aligned}$$

Also ist $e(u) = 0$. Widerspruch. $\square$

Laufzeitanalyse

Lemma

Es werden nur $O(|V|^2)$ Lift-Operationen ausgeführt.

Beweis

- Eine Lift-Operation erhöht $h(u)$ um mindestens 1
- Sie wird nur auf überfließende Knoten angewendet
- $h(u) < 2|V|$, da Pfad in G_f zu s mit $h(s) = |V|$ (Lemma G)

Laufzeitanalyse

Lemma

Es werden nur $O(|V| \cdot |E|)$ saturierte Push-Operationen durchgeführt.

Beweis

- Wird eine Push-Operation auf (u, v) angewandt, dann führt (u, v) um 1 nach unten
- (u, v) verschwindet dann aus G_f (weil saturierter Push)
- Vor der nächsten Push-operation auf (u, v) muß ein Push auf (v, u) durchgeführt werden, dabei führt (u, v) um 1 nach oben
- Zwischen zwei saturierten Pushs auf (u, v) erhöht sich $h(u)$

$\Rightarrow$ Nur $O(|V|)$ saturierte Pushs auf (u, v) weil $h(u) = O(|V|)$.

Laufzeitanalyse

Lemma

Es werden nur $O(|V|^2 \cdot |E|)$ nicht-saturierte Push-Operationen durchgeführt.

Beweis

Sei $\Phi = \sum_{u \in X} h(u)$ mit $X = \{ v \mid e(v) > 0 \}$.

- Eine Lift-Operation erhöht Φ höchstens um $|V|$
- Ein saturierter Push erhöht Φ höchstens um $2|V|$
- Ein nicht-saturierter Push erniedrigt Φ um mindestens 1

Insgesamt wird Φ höchstens um $O(|V|^2|E|)$ erhöht.

$\Rightarrow$ höchstens $O(|V|^2|E|)$ nicht-saturierte Pushs.

Der einfache Preflow-Push-Algorithmus

Theorem

Ein maximaler Fluß kann in $O(|V|^2|E|)$ Schritten berechnet werden.

Beweisskizze

Der einfache Preflow-Push-Algorithmus muß so implementiert werden, daß

- in konstanter Zeit ein überfließender Knoten u gefunden wird, falls noch einer existiert,
- in konstanter Zeit bestimmt wird, ob eine Push- oder Lift-Operation anwendbar ist,
- eine Lift-Operation in $O(|V|)$ Schritten durchgeführt werden kann, und
- eine Push-Operation in konstanter Zeit durchgeführt wird.

Minimum Cost Flow Problem

Eingabe:

Ein s-t-Netzwerk $G = (V, E)$, eine Zahl $\bar{f}$ und eine *Kostenfunktion* $b\colon E \to \mathbf{R}_{\mathbf{0}}^{+}$.

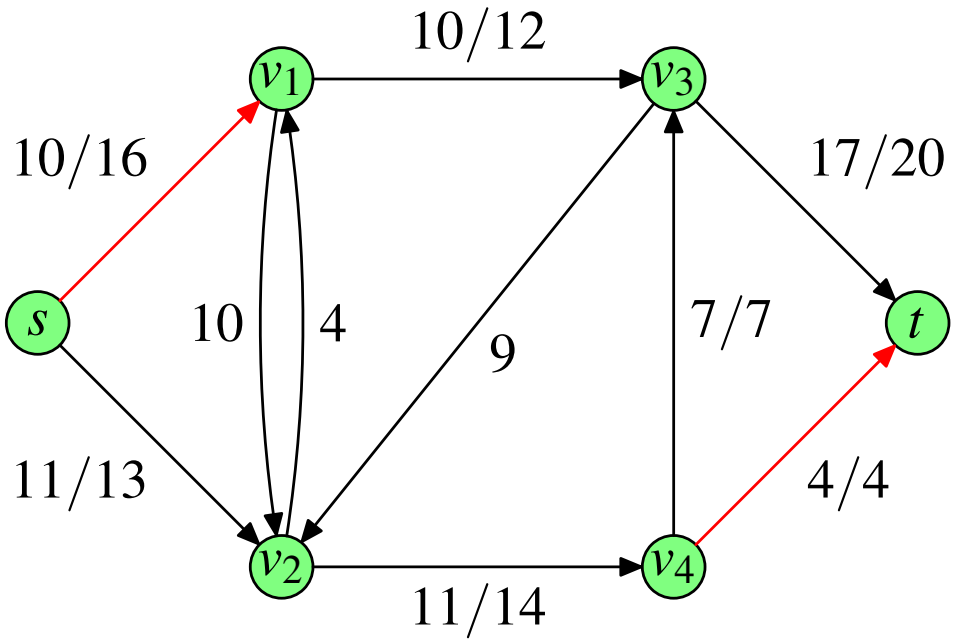
Gesucht:

Ein Fluß f mit $|f| = \bar{f}$ und minimalen Kosten, falls ein solcher existiert.

Die *Kosten* eines Flusses f sind

$$\sum_{e \in E} b(e) f(e).$$

Beispiel



Die schwarzen Kanten haben Kosten 1 und die roten haben Kosten 5.

Die Kosten des Flusses sind 126 und der Wert 21.

Gibt es einen billigeren Fluß mit Wert 21?

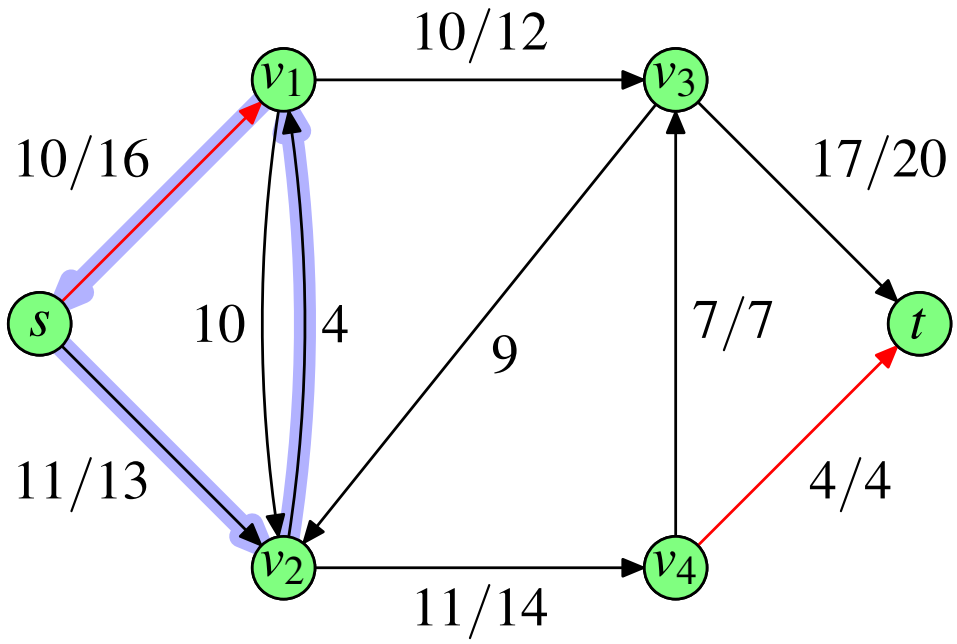
Minimum Cost Flow Problem

Theorem (Minimum Cost Flow Theorem)

1. Ein Fluß f mit Wert $|f|$ hat genau dann minimale Kosten, wenn es in G_f keinen Kreis mit negativen Kosten gibt.
2. Seien f ein Fluß mit minimalen Kosten und p ein augmentierender Pfad mit minimalen Kosten. Dann entsteht wieder ein Fluß mit minimalen Kosten, wenn f entlang p augmentiert wird.

Die Kosten eines Pfads (oder Kreises) im Residualnetzwerk sind die Kosten der Kanten in Vorwärtsrichtung minus die Kosten der Kanten in Rückwärtsrichtung.

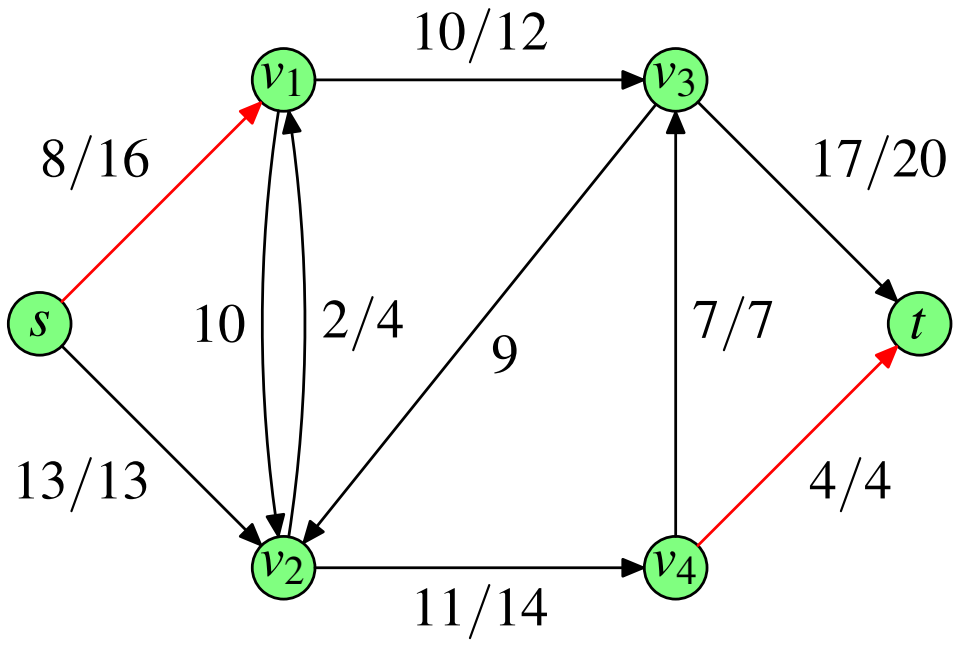
Beispiel



Da der Fluß nicht minimale Kosten hat, muß es einen Kreis in G_f mit negativen Kosten geben.

Seine Restkapazität ist 2 und seine Kosten sind -3.

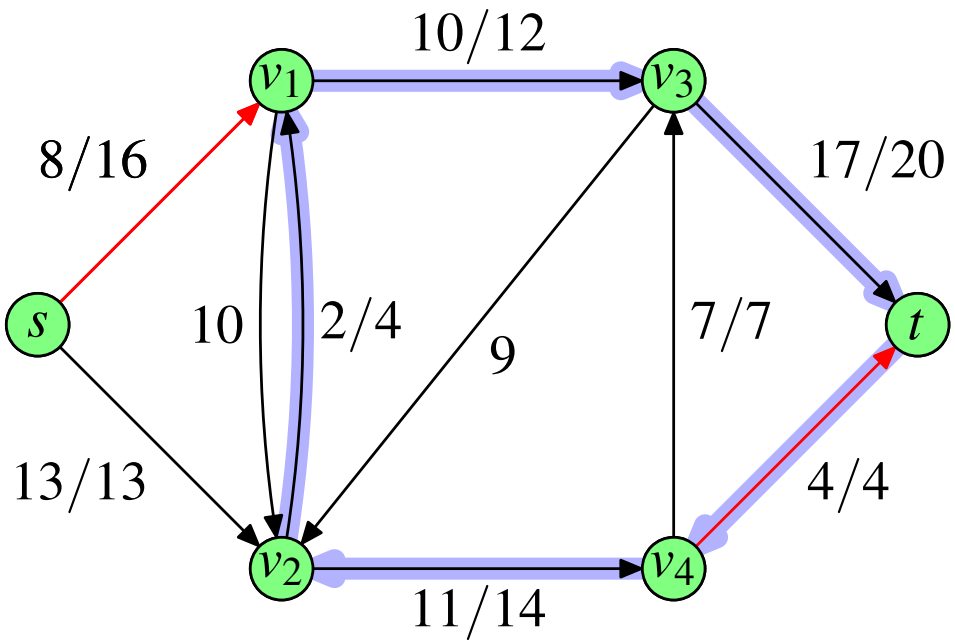
Beispiel



Augmentiert man den Fluß entlang des Kreises, erhalten wir einen neuen, billigeren Fluß.

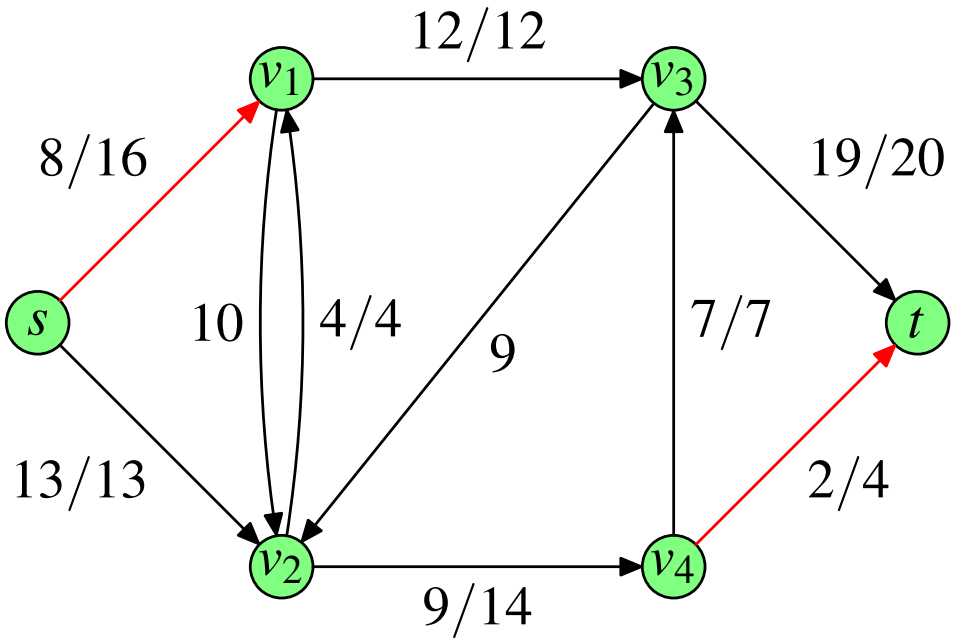
Die Kosten sind jetzt 120. Gibt es einen noch billigeren?

Beispiel



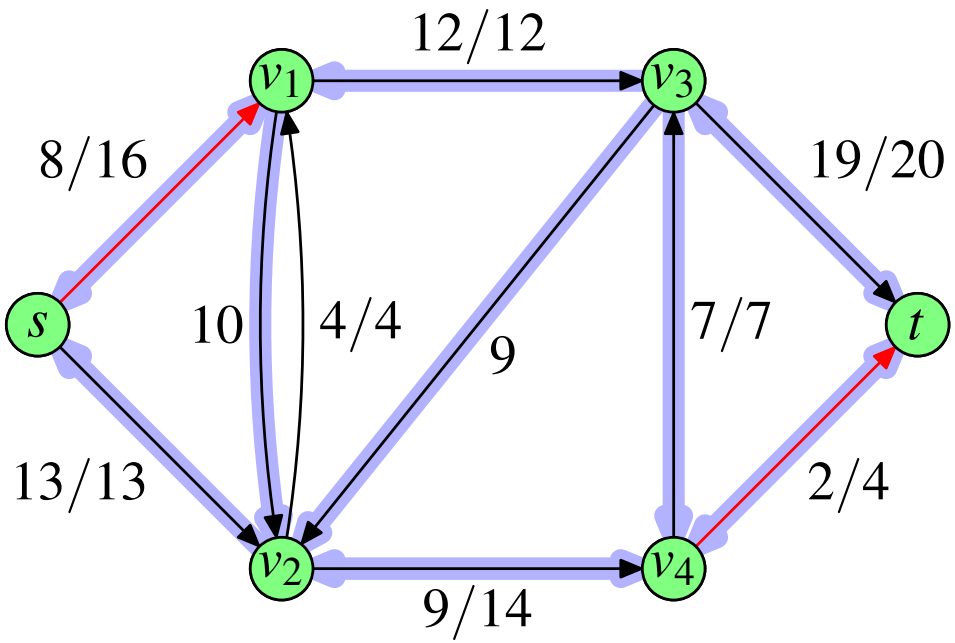
Da es wieder einen augmentierenden Kreis mit negativen Kosten gibt, kann der Fluß noch nicht minimale Kosten haben.

Beispiel



Hat der Fluß jetzt minimale Kosten?

Beispiel



Der Fluß hat minimale Kosten und es gibt keinen Kreis mit negativen Kosten im Residualnetzwerk.

Beweis (1. Teil)

Wenn ein Kreis mit negativen Kosten existiert, dann hat der Fluß nicht minimale Kosten:

Sei K ein Kreis in G_f mit negativen Kosten und Restkapazität $c_f(K)$.

Ändern wir den Fluß indem wir ihn auf jeder Kante von K um $c_f(K)$ erhöhen, dann sinken die Kosten.

$\Rightarrow$ Dies widerspricht der Annahme, daß f minimale Kosten hat.

Andere Richtung:

Angenommen f hat nicht minimale Kosten. Dann existiert ein Fluß f^* mit kleineren Kosten, aber $|f| = |f^*|$.

Sei $\Delta = f^* - f$. Da $|f| = |f^*|$, gilt $\Delta(V, s) = \Delta(V, t) = 0$.

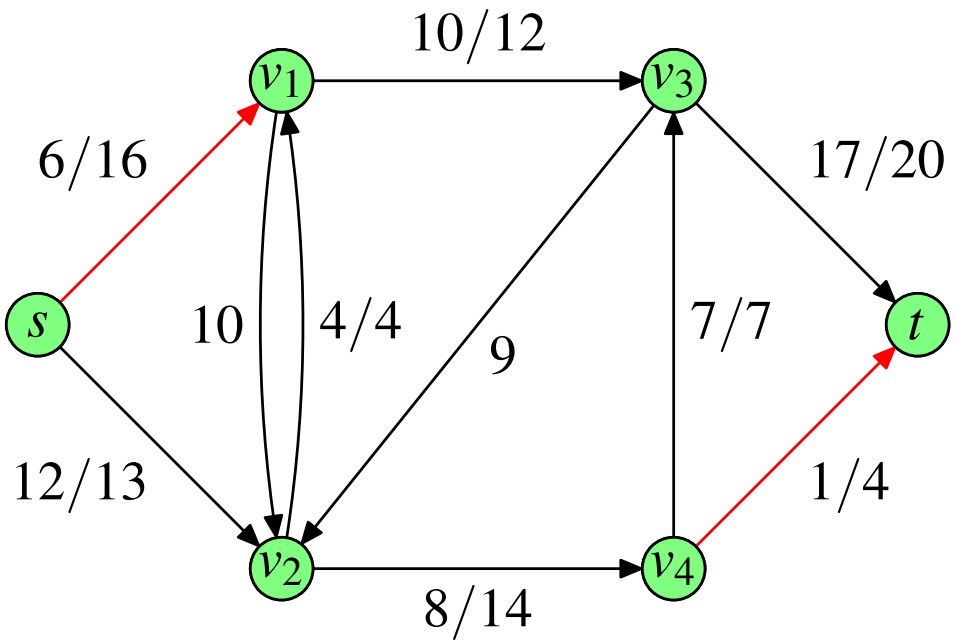
Es gilt also $\Delta(V, u) = 0$ für alle $u \in V$.

$\Rightarrow$ $\Delta(u, v) > 0$ ist eine Vereinigung von Flüssen auf Kreisen.

$\Rightarrow$ Einer der Kreise muß negative Kosten haben

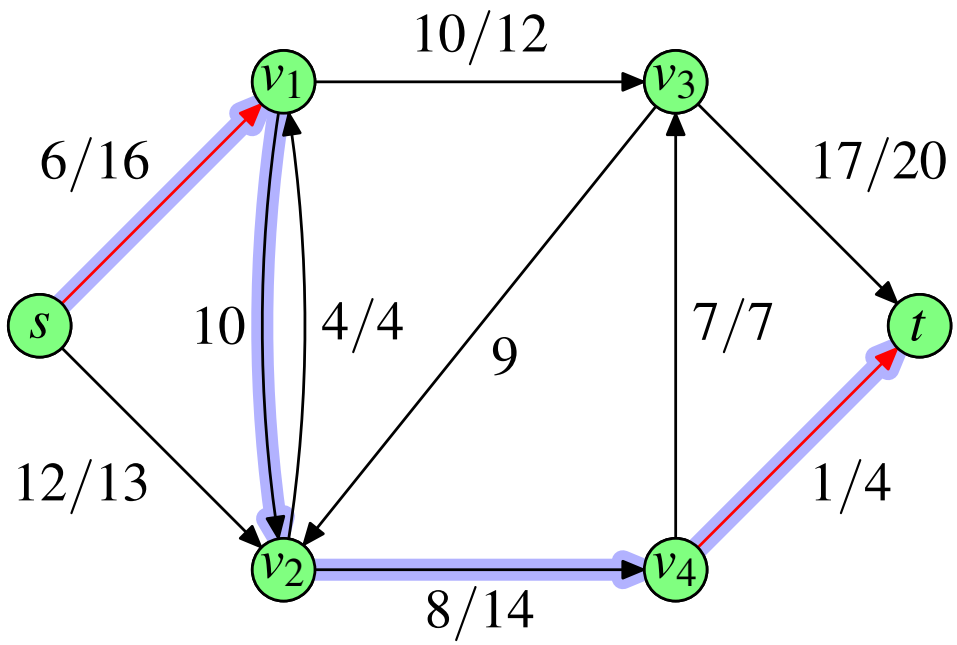
(Damit ist der erste Teil bewiesen.)

Beispiel



Der Fluß hat minimale Kosten. Was passiert, wenn er mit einem augmentierenden Pfad augmentiert wird?

Beispiel



Dieser augmentierende Pfad hat nicht minimale Kosten, denn von s nach v_2 gibt es eine billigere Abkürzung.

Geht man die Abkürzung und dann den übersprungenen Teil rückwärts, erhalten wir einen Kreis mit negativen Kosten.

Beweis (2. Teil)

Damit durch eine Augmentierung ein Kreis im Residualnetzwerk entsteht, muß der augmentierende Pfad durch eine Kante des Kreises rückwärts gehen.

Sei p ein augmentierender Pfad, durch den ein Kreis K mit negativen Kosten entsteht. Die Kante (u, v) liege auf p und die Kante (v, u) liege auf K.

Sei p' folgender Pfad: Von s entlang p bis u, dann entlang K bis v, dann entlang p bis t.

Die Kosten von p' sind dann kleiner als von p. Also kann p kein augmentierender Pfad mit minimalen Kosten sein.

Ein einfacher Algorithmus

Gesucht: Fluß mit Wert B und minimalen Kosten

Initialisiere Fluß f zu 0
while es gibt einen augmentierenden Pfad p **do**
    finde einen augmentierenden Pfad p
        mit minimalen Kosten
    augmentiere f entlang p mit $\min\{B - |f|, c_f(p)\}$
return f

Durch das Minimum Cost Flow Theorem ist garantiert, daß der gefundene Fluß minimale Kosten hat.

Die Edmonds–Karp–Variante

for each edge $(u, v) \in E$ **do**
  $f(u, v) := 0$
  $f(v, u) := 0$
while there exists a path from s to t in G_f **do**
  $p :=$ a shortest, minimum cost path from s to t in G_f
  $c_f(p) := \min\{\, c_f(u, v) \mid (u, v) \text{ is in } p \,\}$
  for each edge (u, v) in p **do**
    $f(u, v) := f(u, v) + \min\{B - |f|, c_f(p)\}$
    $f(v, u) := -f(u, v)$
return f

Lemma D'

Gegeben ist ein s-t-Netzwerk $G = (V, E)$. Sei f ein Fluß, der während der Ausführung der Edmonds–Karp–Variante vorkommt und f' der Fluß nach der anschließenden Augmentierung.

Dann gilt: $\delta'(s, v) \geq \delta(s, v)$ für alle $v \in V$.

Hierbei sind $\delta(u, v)$ (bzw. $\delta'(u, v)$) die Kosten des billigsten Pfads von u nach v in G_f (bzw. in $G_{f'}$).

Beweis:

Wie Lemma D

Theorem

Die Edmonds–Karp–Variante findet einen Fluß mit Wert B und minimalen Kosten in endlich vielen Schritten.

Beweis

- Solange die Kosten der augmentierenden Pfade gleichbleiben, werden unter ihnen kürzeste gewählt und der Algorithmus arbeitet wie der Edmonds–Karp–Algorithmus.

  $\Rightarrow$ Nach $O(|E| \cdot |V|)$ Iterationen, wird ein Pfad mit höheren Kosten gewählt (Lemma D').

- Es gibt nur endlich viele, mögliche augmentierende Pfade und damit nur endlich viele mögliche Kosten.

Bipartites Matching maximalen Gewichts

Gegeben:

Ein bipartiter, ungerichteter Graph (V_1, V_2, E) und eine Gewichtsfunktion $b\colon E \to \mathbf{R}_0^+$.

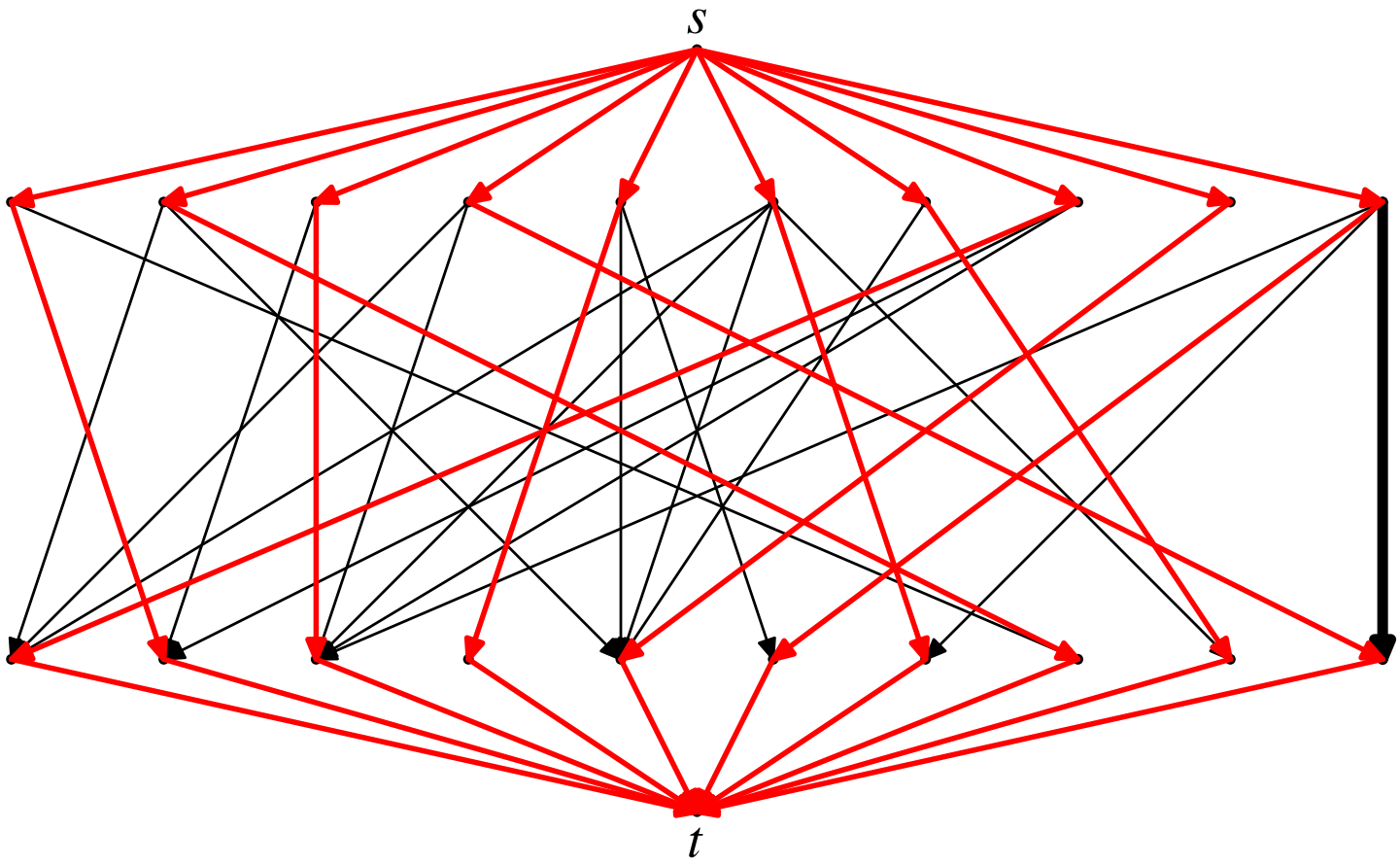
Gesucht:

Ein Matching mit maximalem Gewicht.

Das Gewicht eines Matchings M ist

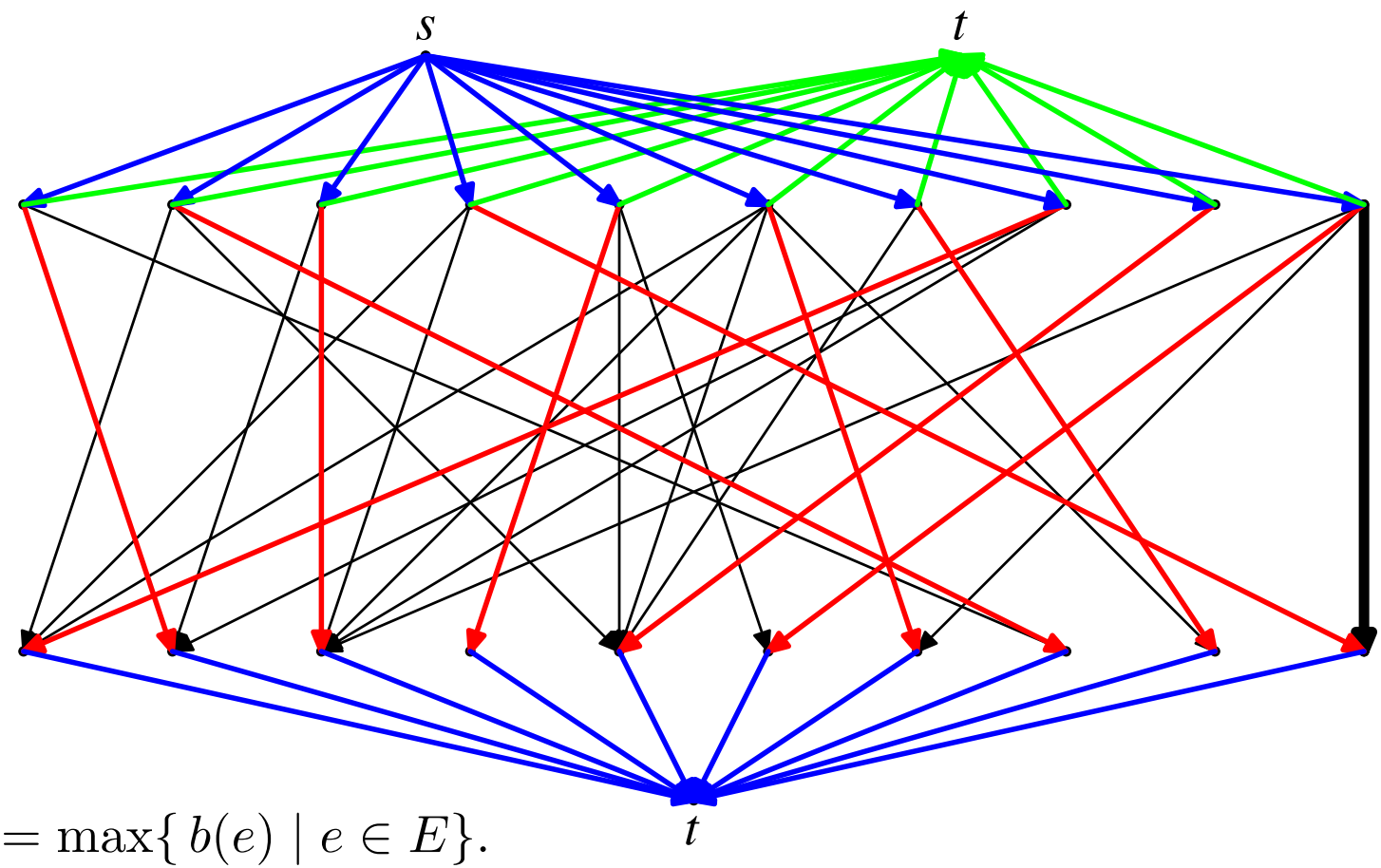
$$b(M) = \sum_{e \in M} b(e).$$

Beispiel



Alle Kanten haben Gewicht 1, nur die dicke Kante hat 3.

Hat das perfekte Matching auch maximales Gewicht?

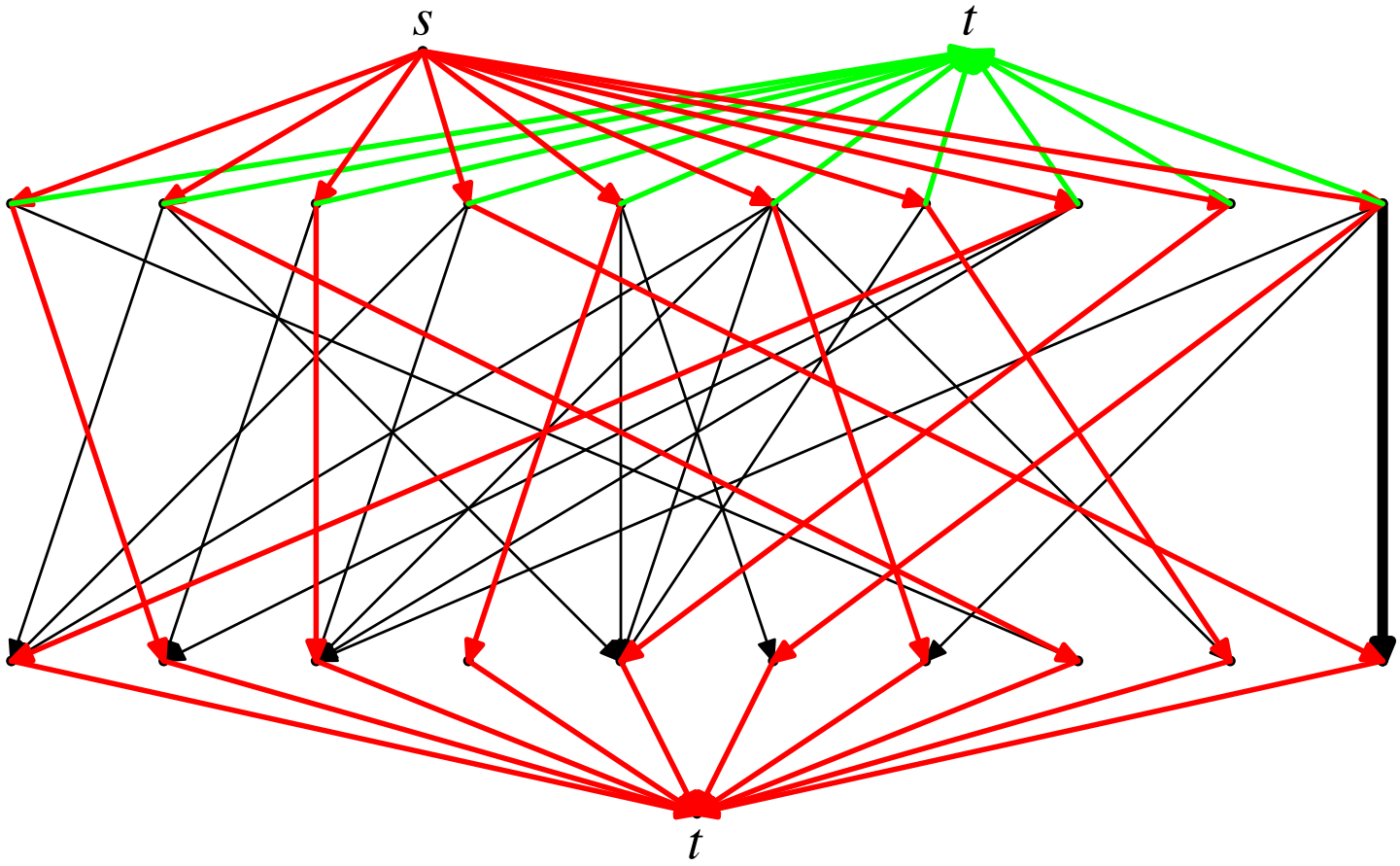


$w^* = \max\{\, b(e) \mid e \in E\}$.

Blaue Kanten haben Kosten 0, grüne Kanten haben Kosten w^*,
eine Matching-Kante e hat Kosten $w^* - b(e)$.

Matching mit Gewicht $b \Longrightarrow$ Fluß mit Kosten $nw^* - b$.

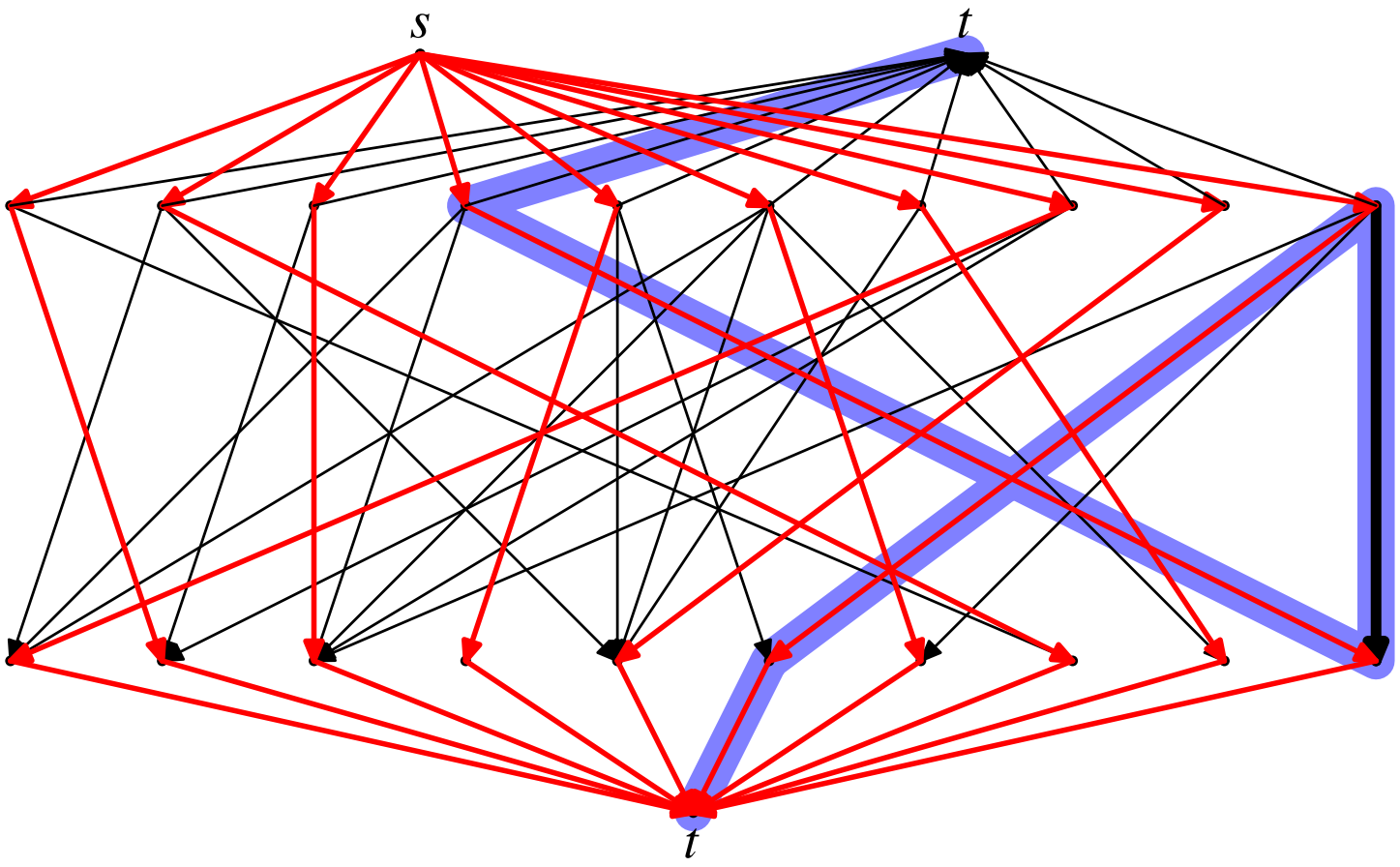
Beispiel



Dem perfekten Matching entspricht der rote Fluß mit Wert 10.

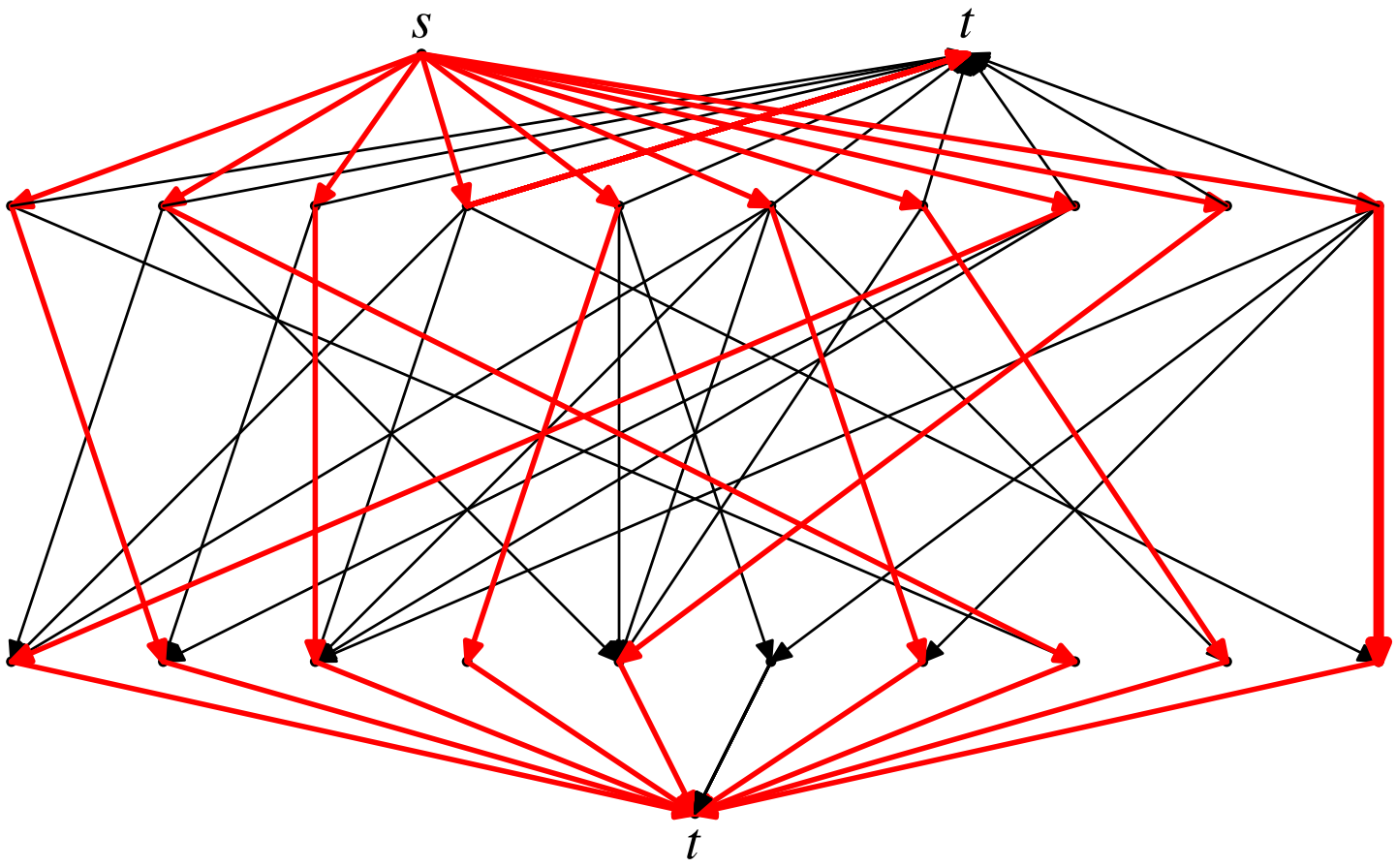
Wie groß sind die Kosten? ($w^* = 3$)

Beispiel



Die Kosten sind nicht minimal, denn es gibt einen Kreis in G_f mit negativen Kosten.

Beispiel



Jetzt hat der Fluß Wert 10 und minimale Kosten. Das entsprechende Matching ist kleiner, hat aber maximales Gewicht.

Bipartites Matching maximalen Gewichts

Bipartites Matching maximalen Gewichts läßt sich mittels Flusses minimaler Kosten lösen:

Die Kosten des Flusses sind

- $(n - |M|)w^*$ und
- $\sum_{e \in M} (w^* - b(e))$

Zusammen sind es $nw^* - b(M)$ (mit $n = |V_1| \geq |V_2|$)

Dieser Wert ist minimal, wenn $b(M)$ maximal ist.$\square$