TECHNISCHE UNIVERSITÄT WIEN
Vienna | Austria

Maximal flows in networks

Bachelorarbeit aus Diskreter Mathematik

Florian Schager

TU Wien, Vienna, Austria

14. Juni 2021

Übersicht

Wiederholung - Problemstellung

Problem (Maximum flow)

Finde maximalen Fluss von s nach t in Netzwerk $N = (G, c, s, t)$, der folgenden Bedingungen genügt:

1. $0 \leq f(e) \leq c(e)$ für jede Kante e; (Kapazitätsbeschränkung)
2. $\sum_{e^+=v} f(e) = \sum_{e^-=v} f(e)$ für jeden Knoten $v \neq s, t$. (Flusserhaltung)

Wert eines Flusses:

$$w(f) := \sum_{e^-=s} f(e) - \sum_{e^+=s} f(e) = \sum_{e^+=t} f(e) - \sum_{e^-=t} f(e).$$

Wiederholung - Algorithmus

Algorithmus

- Starte mit dem trivialen Fluss: $f(e) = 0, e \in E$.
- Finde einen erweiternden Pfad P und berechne

$$d := \min[\{c(e) - f(e) : e \text{ Vorwärts-Kante } \in P\} \cup \{f(e) : e \text{ Rückwärts-Kante } \in P\}].$$

- Konstruiere erweiterten Fluss f' mit $w(f') = w(f) + d$:

$$f'(e) = \begin{cases} f(e) + d, & e \text{ ist Vorwärts-Kante } \in P \\ f(e) - d, & e \text{ ist Rückwärts-Kante } \in P \\ f(e), & \text{sonst} \end{cases}$$

- Wiederhole solange, bis kein erweiternder Pfad mehr gefunden werden kann.

Hilfsnetzwerke

Sei $N = ((V, E), c, s, t)$ ein Flussnetzwerk.

Definition (Hilfsnetzwerk $N' = ((V, E'), c', s, t)$)

- $e = uv \in E$ mit $f(e) < c(e) \quad \rightarrow \quad e' = uv \in E'$ mit $c(e') = c(e) - f(e)$.
- $e = uv, f(e) > 0 \quad \rightarrow \quad e'' = vu \in E'$ mit $c'(e'') = f(e)$.

- Effizientere Implementierung: Weglassen überflüssiger Information
- Hilfsnetzwerk $N' \rightarrow$ Schichtnetzwerk N''
- Schichtlevel eines Knoten v = Distanz zur Quelle $s : d(s, v)$
- Entferne alle Knoten $v \neq t : d(s, v) \geq d(s, t)$.
- Entferne alle Kanten, welche von einer höheren in eine niedrigere Schicht führen.

Algorithmus - Skizze

Definition

Ein *Flow* f heißt *Blocking Flow*, wenn es bezüglich f keine erweiternden Pfade alleine aus Vorwärtskanten gibt.

Algorithmus (Blocking Flows)

- Starte mit dem trivialen Fluss: $f(e) = 0, e \in E$.
- Wiederhole solange f nicht maximal:
  - Erstelle Schichtnetzwerk N'' bezüglich f.
  - Finde *Blocking Flow* g in N''
  - Erweitere f um g

MKM-Algorithmus (1/3)

Definition (Flusspotential)

Für einen Knoten v definieren wir das Flusspotential als

$$p(v) = \min \left\{ \sum_{e^- = v} c(e), \sum_{e^+ = v} c(e) \right\}.$$

MKM-Algorithmus (2/3)

Algorithmus (BlockMKM (Malhotra, Kumar und Mahashwari))

- Starte mit dem trivialen Fluss: $g(e) = 0, e \in E$.
- Berechne Flusspotentiale für alle $v \in V$.
- Wiederhole solange $s \in V$ und $t \in V$:
  - Finde Knoten w mit minimalem Flusspotential $p(w)$.
  - `Push`$(w, p(w))$.
  - `Pull`$(w, p(w))$.
  - Entferne redundante Ecken und Kanten.

MKM-Algorithmus (3/3)

Algorithmus

procedure PUSH$(w, p(w))$
  Sei Q Warteschlange mit einzigem Element w
  $\forall u \in V : b(u) \leftarrow 0; \quad b(y) \leftarrow k$
  repeat
    Entferne v von Q
    while $v \neq t \wedge b(v) \neq 0$ **do**
      Wähle Kante $e = vu; \quad m \leftarrow \min\{c(e), b(v)\}$
      $c(e) \leftarrow c(e) - m; \quad g(e) \leftarrow g(e) + m$
      $p^+(u) \leftarrow p^+(u) - m; \quad b(u) \leftarrow b(u) + m$
      $p^-(v) \leftarrow p^-(v) - m; \quad b(v) \leftarrow b(v) - m$
      Füge u zu Q hinzu
      if $c(e) = 0$ **then** entferne e von E **end**
    end
  until $Q = \emptyset$;
end procedure

Reduktionen auf ein Maximum Flow Problem (1/3)

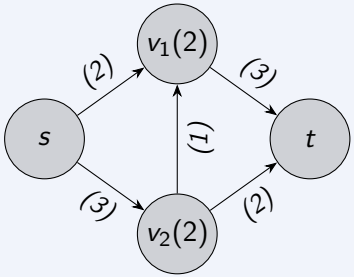
Problem (Knotenkapazitäten)

Zusätzlich zu der Kapazitätsfunktion $c : E \to \mathbb{R}^+$ sei noch eine Kapazitätsfunktion $d : V \to \mathbb{R}^+$ gegeben, welche den maximalen Fluss durch einen Knoten limitiert.

Reduktionen auf ein Maximum Flow Problem (1/3)

Problem (Knotenkapazitäten)

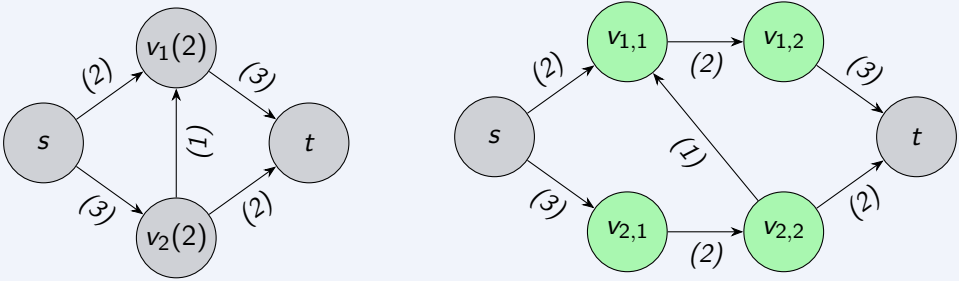
Zusätzlich zu der Kapazitätsfunktion $c : E \to \mathbb{R}^+$ sei noch eine Kapazitätsfunktion $d : V \to \mathbb{R}^+$ gegeben, welche den maximalen Fluss durch einen Knoten limitiert.



Reduktionen auf ein Maximum Flow Problem (1/3)

Problem (Knotenkapazitäten)

Zusätzlich zu der Kapazitätsfunktion $c : E \to \mathbb{R}^+$ sei noch eine Kapazitätsfunktion $d : V \to \mathbb{R}^+$ gegeben, welche den maximalen Fluss durch einen Knoten limitiert.



Reduktionen auf ein Maximum Flow Problem (2/3)

Definition (Bipartiter Graph)

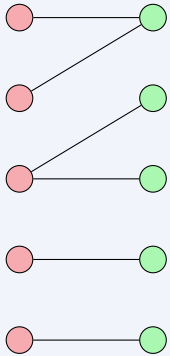
Ein ungerichteter Graph $G = (V, E)$ heißt bipartit, falls sich seine Knoten in zwei disjunkte Teilmengen A und B aufteilen lassen, sodass zwischen den Knoten innerhalb beider Teilmengen keine Kanten verlaufen. Das heißt, für jede Kante $uv \in E$ gilt entweder $u \in A$ und $v \in B$ oder $u \in B$ und $v \in A$.

Problem (Bipartiter Graph - Matching of maximal cardinality)

Sei $G = (V, E)$ ein ungerichteter, bipartiter Graph. Gesucht ist eine Teilmenge $M \subseteq E$ mit maximaler Kardinalität, sodass keine zwei Kanten einen gemeinsamen Endknoten teilen.

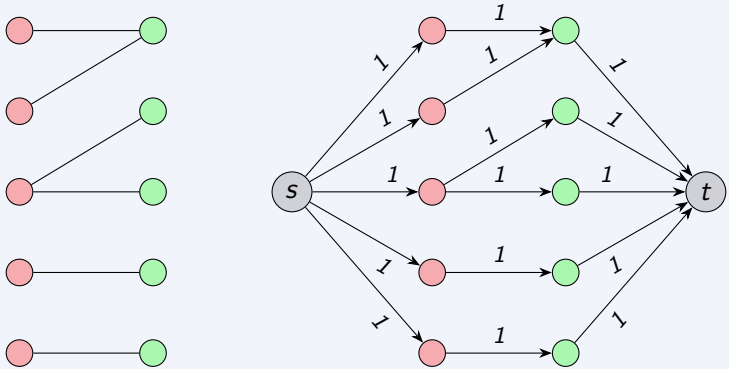
Reduktionen auf ein Maximum Flow Problem (3/3)

Problem (Bipartiter Graph - Matching of maximal cardinality)



Reduktionen auf ein Maximum Flow Problem (3/3)

Problem (Bipartiter Graph - Matching of maximal cardinality)



Preflows

Definition (Fluss)

Ein Fluss ist eine Funktion $f : V \times V \to \mathbb{R}$ die folgenden Bedingungen genügt:

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

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

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

Definition (Preflow)

Ein *Preflow* ist schließlich eine Funktion $f : V \times V \to \mathbb{R}$, welche (1) und (2) erfüllt, sowie eine abgeschwächte dritte Bedingung:

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

Der Wert $e(v) = \sum_{u \in V} f(u, v)$ nennen wir den *flow excess* des Preflows f in v.

Residualgraph

Definition (Residualgraph)

Zu einem gegebenen Preflow f definieren wir vorerst die Residualkapazität $r_f : V \times V \to \mathbb{R}$ durch

$$r_f(v, w) := c(v, w) - f(v, w).$$

Wir definieren zusätzlich einen Residualgraphen $G_f = (V, E_f)$ mit

$$E_f := \{vw \in E : r_f(v, w) > 0\}$$

Labels

Definition

Ein *valid labelling* ist eine Funktion $d : V \to \mathbb{N}_0 \cup \{\infty\}$ mit

(4) $d(s) = |V|, \quad d(t) = 0$

(5) $\forall vw \in E_f : d(v) \leq d(w) + 1.$

Weiters nennen wir einen Knoten $v \neq s$ aktiv, solange $e(v) > 0$ und $d(v) < \infty$.

Algorithmus von Goldberg und Tarjan (1/4)

Initialisierung:

Algorithmus (Goldberg und Tarjan)

1: **procedure** GoldbergTarjan(N, f, v, w)
2: $\quad \forall v \neq s : f(s,v) \leftarrow c(s,v); \quad f(v,s) \leftarrow -c(s,v)$
3: $\quad \forall v, w \neq s : f(v,w) \leftarrow 0$
4: $\quad d(s) = |V|; \quad \forall v \neq s : d(v) \leftarrow 0$
5: $\quad$ **while** $\exists v$ *aktiv* **do**
| Führe eine zulässige Operation aus.
end
6: **end procedure**

Algorithmus von Goldberg und Tarjan (2/4)

Algorithmus (Goldberg und Tarjan)

1: **procedure** $\textsc{Push}(N, f, v, w)$
2: $\quad \delta \leftarrow \min(e(v), r_f(v, w))$
3: $\quad f(v, w) \leftarrow f(v, w) + \delta; \quad f(w, v) \leftarrow f(w, v) - \delta$
4: $\quad r_f(v, w) \leftarrow r_f(v, w) - \delta; \quad r_f(w, v) \leftarrow f_f(w, v) - \delta$
5: $\quad e(v) \leftarrow e(v) - \delta; e(w) \leftarrow e(w) + \delta$
6: **end procedure**

Voraussetzungen:

1. v ist aktiv
2. $r_f(v, w) > 0$
3. $d(v) = d(w) + 1$

Algorithmus von Goldberg und Tarjan (3/4)

Algorithmus (Goldberg und Tarjan)

1: **procedure** RELABEL(N, f, v, d)
2: $\quad d(v) \leftarrow \min\{d(w) + 1 : r_f(v, w) > 0\}$
3: **end procedure**

Voraussetzungen:

1. v ist aktiv
2. $\forall w : r_f(v, w) > 0 \implies d(v) \leq d(w)$

Algorithmus von Goldberg und Tarjan (4/4)

- Maximal $\mathcal{O}(|V|^2|E|)$ Operationen unabhängig von der Reihenfolge.
- Speedup möglich durch bewusste Wahl der Reihenfolge:

Der momentan aktive Knoten wird solange weiterbearbeitet, bis entweder

1. $e(v) = 0$ (kein Exzess mehr vorhanden) oder
2. alle Kanten inzident mit v schon für einen `Push` verwendet wurden und ein `Relabel` stattgefunden hat.

- `Fifoflow` (first in, first out): Laufzeit $\mathcal{O}(|V|^3)$.
- `Hlflow` (highest label): Laufzeit $\mathcal{O}(|V|^2|E|^{1/2})$.

Algorithmus von Goldberg und Tarjan (`Hlflow`) - Beispiel

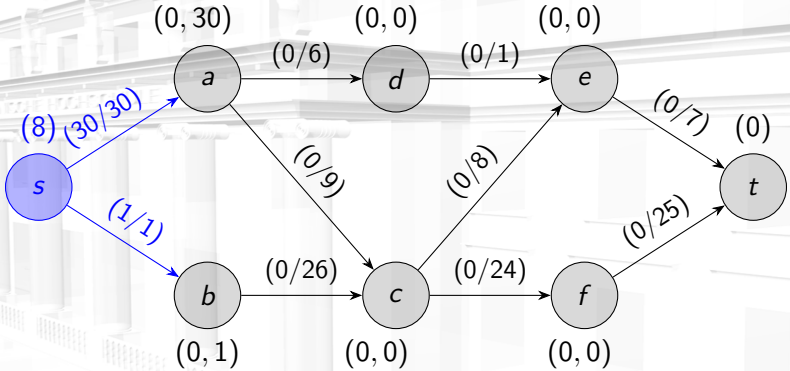


Abbildung: Initialisierung, $Q = (a, b)$

Algorithmus von Goldberg und Tarjan (`Hlflow`) - Beispiel

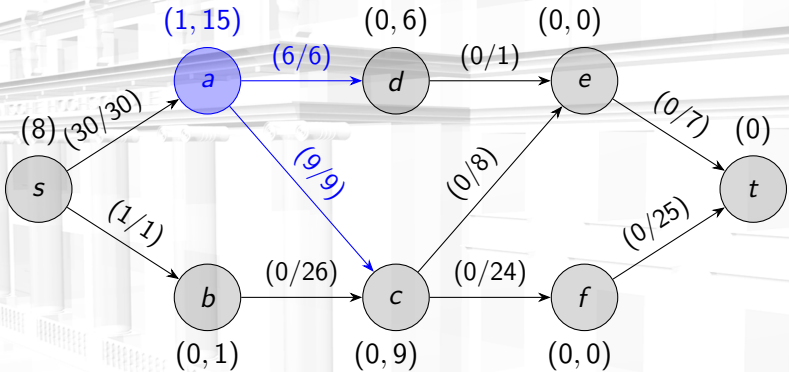


Abbildung: `Relabel`(a), `Push`(a, c), `Push`(a, d), $Q = (a, b, c, d)$

Algorithmus von Goldberg und Tarjan (`Hlflow`) - Beispiel

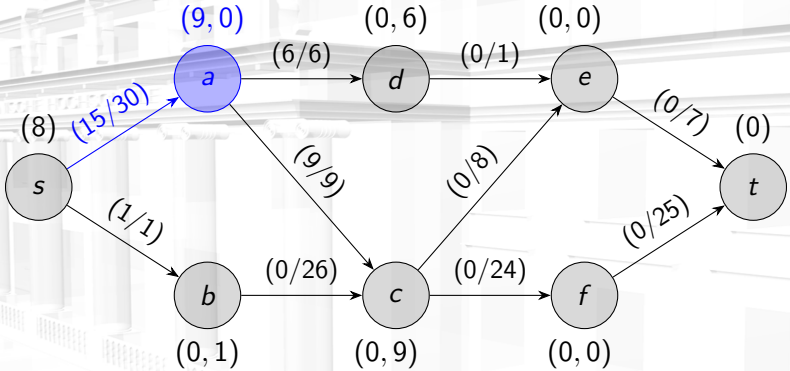


Abbildung: `Relabel`(a), `Push`(a, s), $Q = (b, c, d)$

Algorithmus von Goldberg und Tarjan (`Hlflow`) - Beispiel

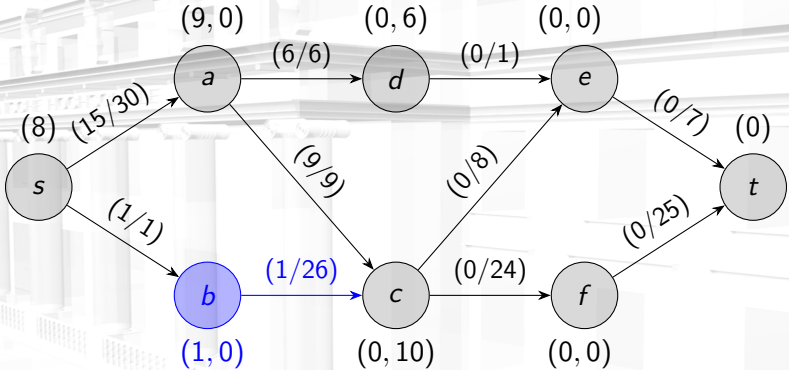


Abbildung: `Relabel`(b), `Push`(b, c), $Q = (c, d)$

Algorithmus von Goldberg und Tarjan (`Hlflow`) - Beispiel

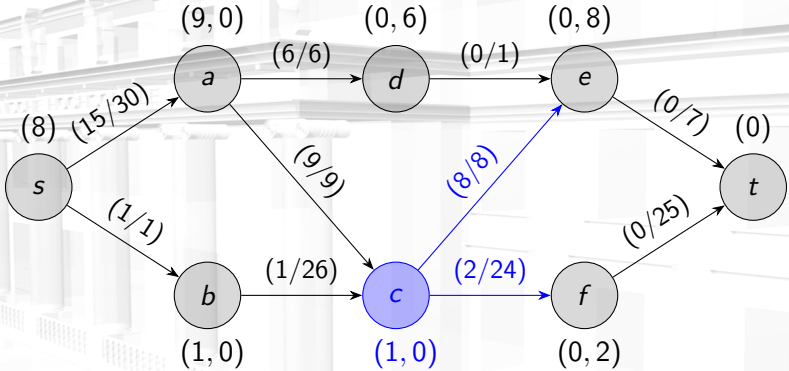


Abbildung: `Relabel`(c), `Push`(c, e), `Push`(c, f), $Q = (d, e, f)$

Algorithmus von Goldberg und Tarjan (`Hlflow`) - Beispiel

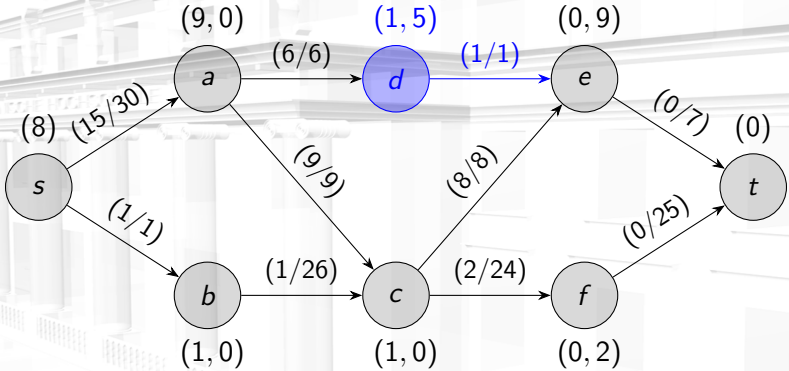


Abbildung: `Relabel`(d), `Push`(d, e), $Q = (d, e, f)$

Algorithmus von Goldberg und Tarjan (`Hlflow`) - Beispiel

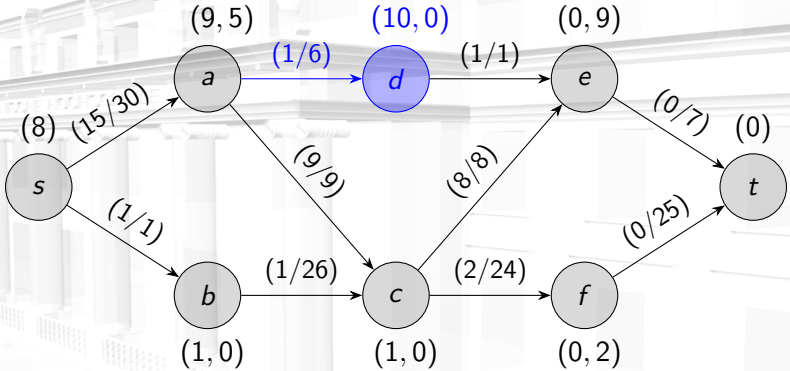


Abbildung: `Relabel`(d), `Push`(d, a), $Q = (a, e, f)$

Algorithmus von Goldberg und Tarjan (`Hlflow`) - Beispiel

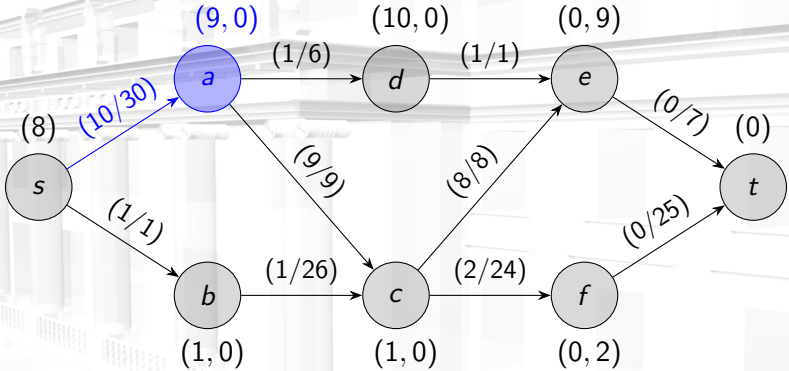


Abbildung: `Push`(a, s), $Q = (e, f)$

Algorithmus von Goldberg und Tarjan (`Hlflow`) - Beispiel

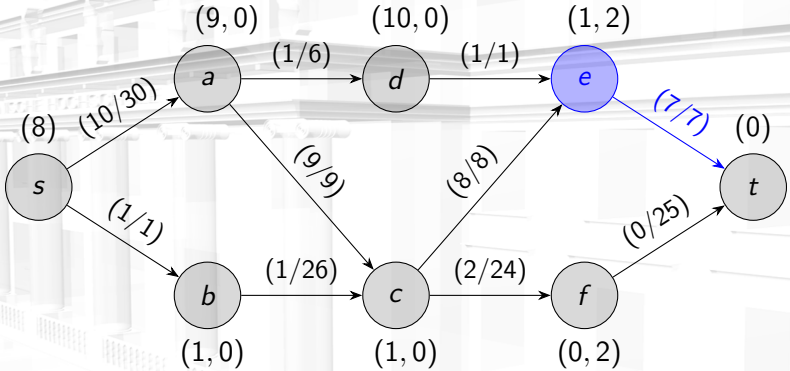


Abbildung: `Relabel`(e), `Push`(e, t), $Q = (e, f)$

Algorithmus von Goldberg und Tarjan (`Hlflow`) - Beispiel

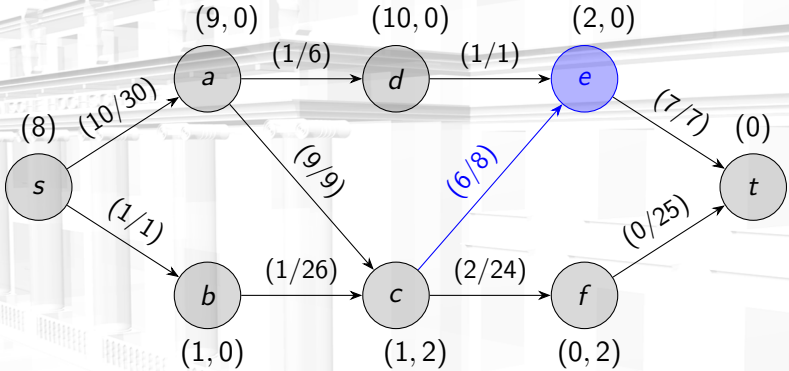


Abbildung: `Relabel`(e), `Push`(e, c), $Q = (c, f)$

Algorithmus von Goldberg und Tarjan (`Hlflow`) - Beispiel

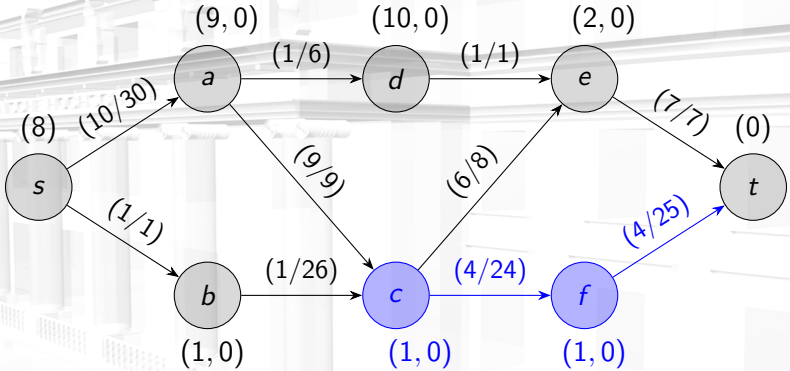


Abbildung: `Push`(c, f), `Relabel`(f), `Push`(f, t), $Q = \emptyset$