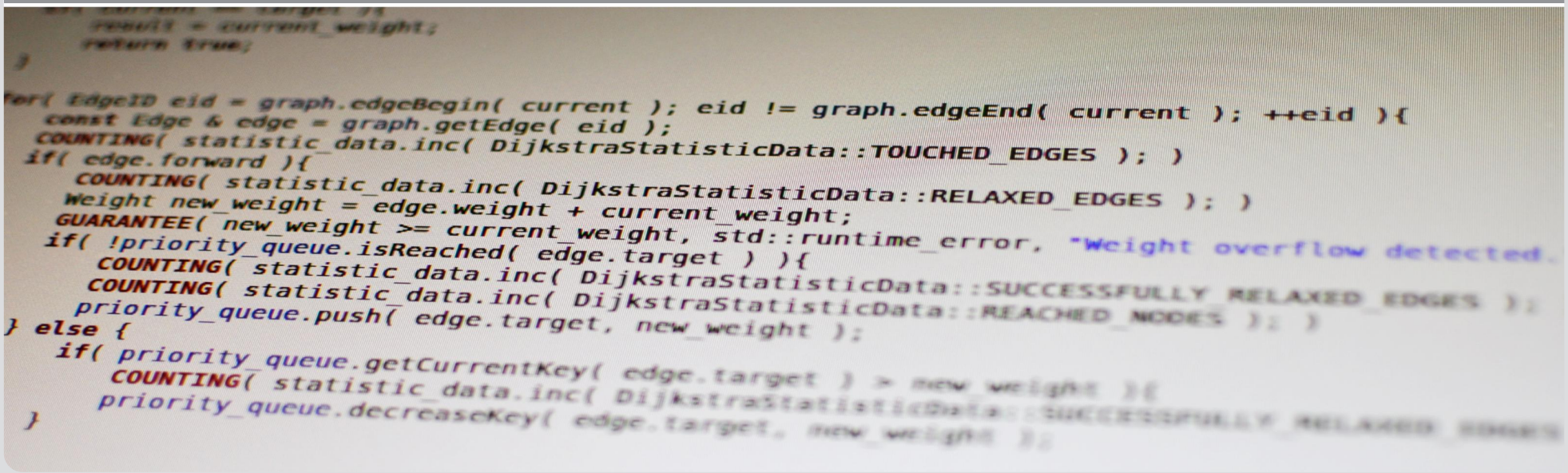
Karlsruher Institut für Technologie

Übung 4 – Algorithmen II

Moritz Laupichler, Nikolai Maas – *{moritz.laupichler, nikolai.maas}@kit.edu*

`https://algo2.iti.kit.edu/AlgorithmenII_WS23.php`

Institut für Theoretische Informatik - Algorithm Engineering

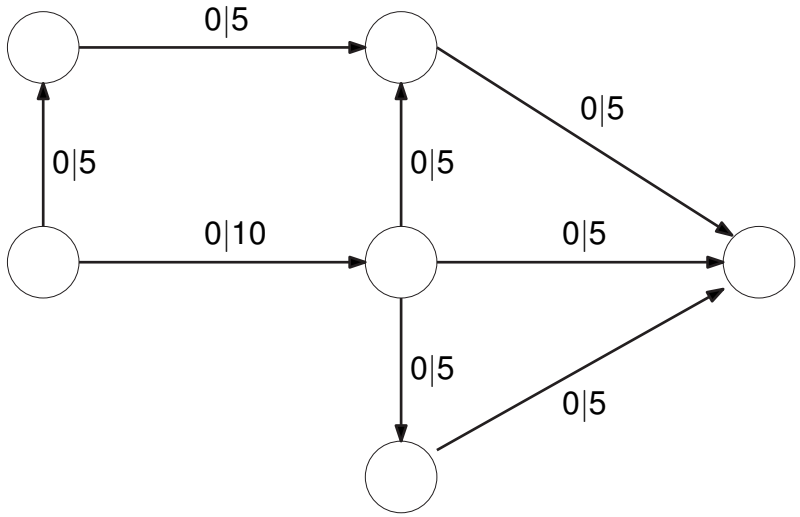


Themenübersicht

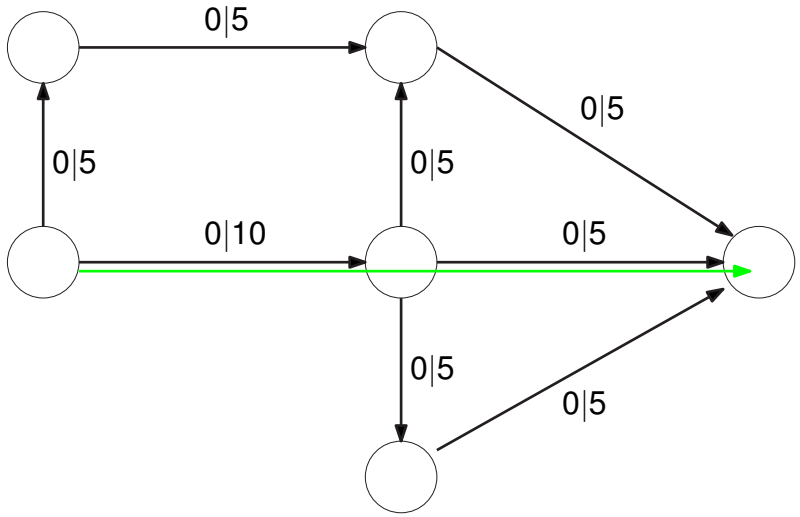
- Maximale Flüsse
  - Ford-Fulkerson
  - Dinitz
  - Preflow-Push

Ford-Fulkerson

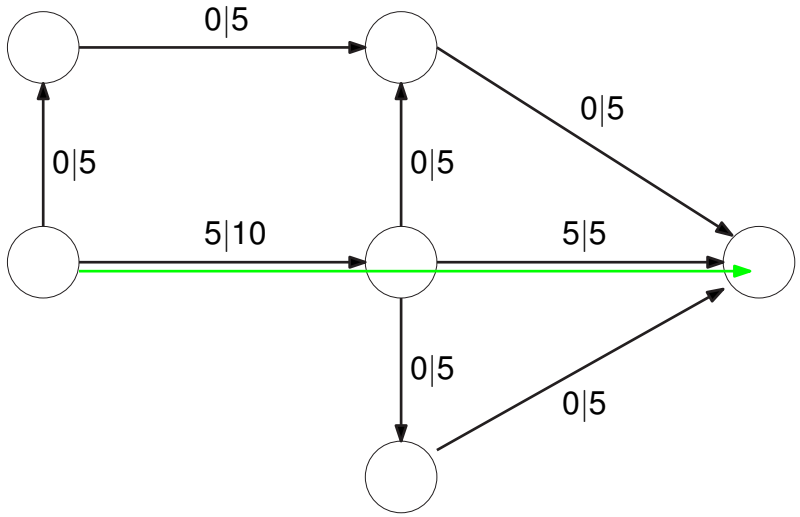
Flüsse und Ford-Fulkerson



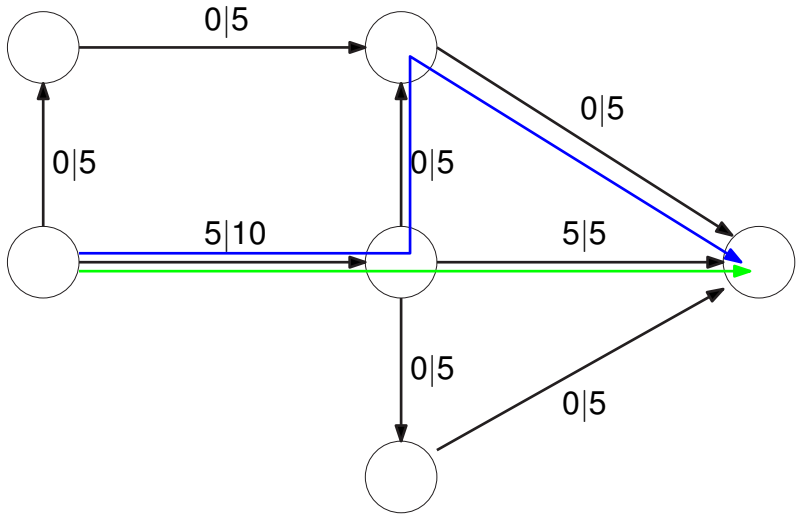
Flüsse und Ford-Fulkerson



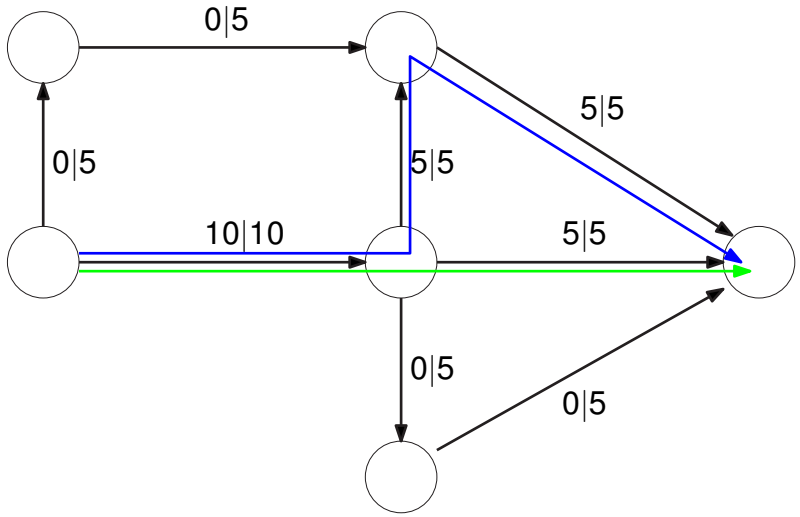
Flüsse und Ford-Fulkerson



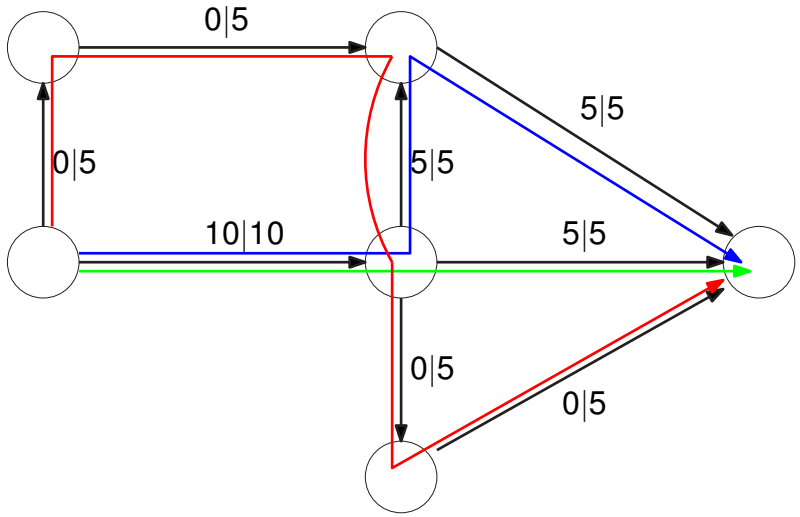
Flüsse und Ford-Fulkerson



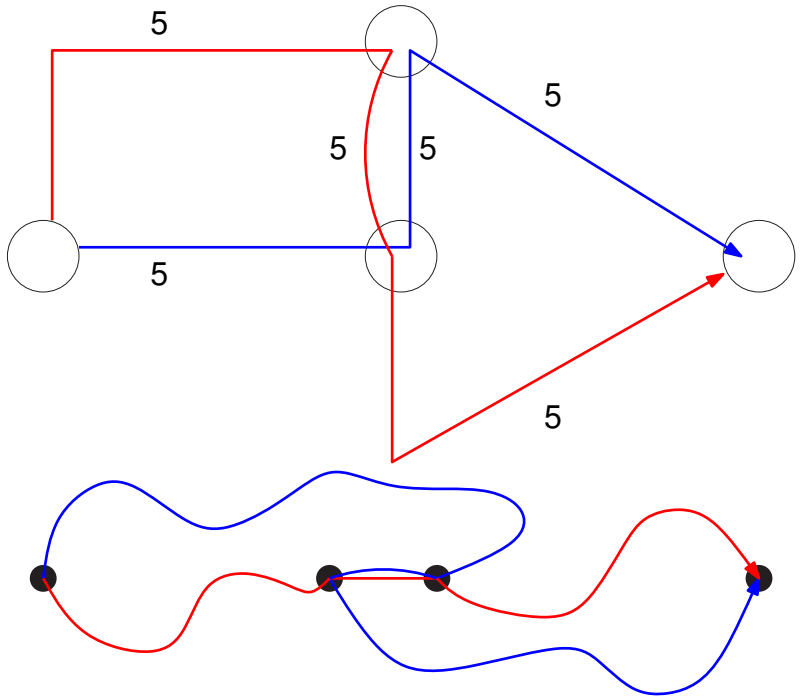
Flüsse und Ford-Fulkerson



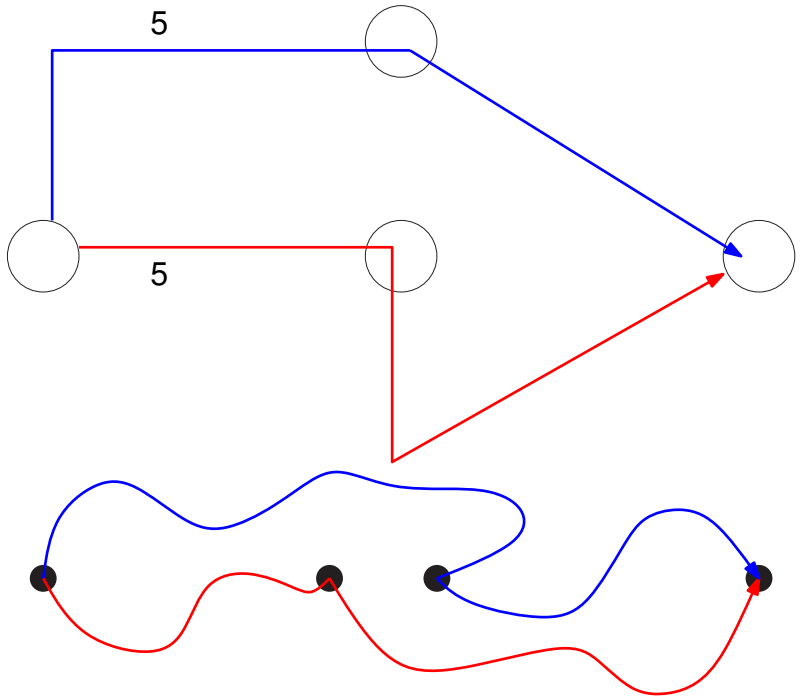
Flüsse und Ford-Fulkerson



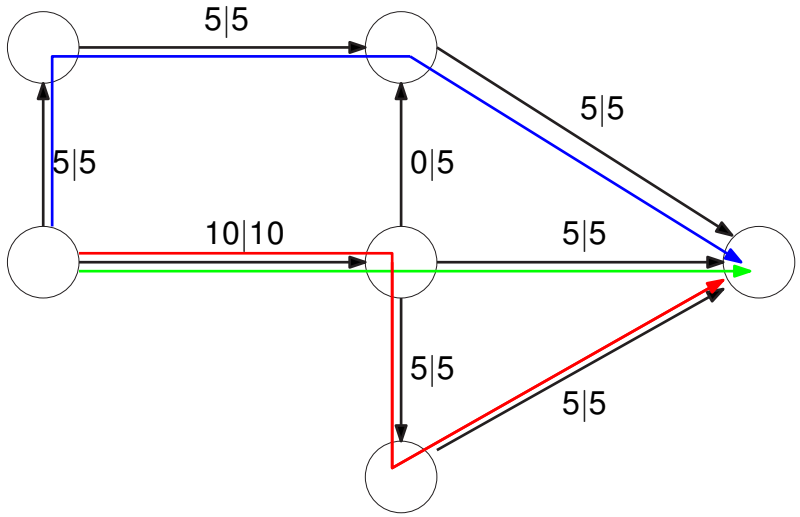
Flüsse und Ford-Fulkerson



Flüsse und Ford-Fulkerson



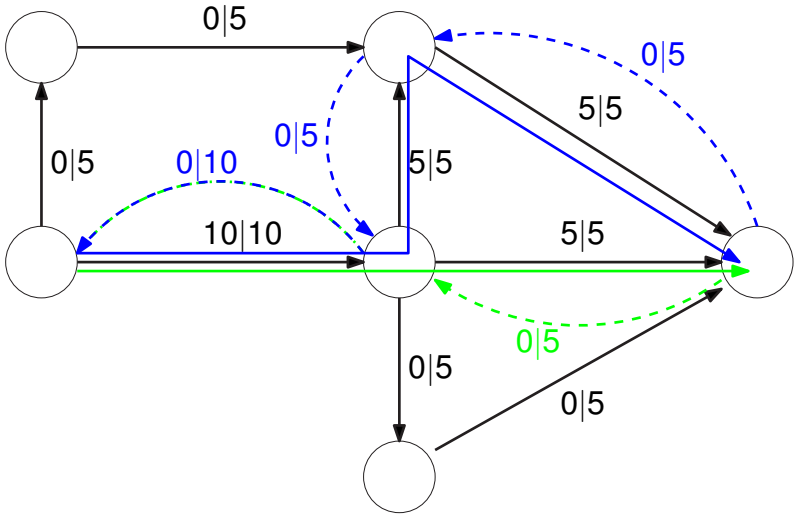
Flüsse und Ford-Fulkerson



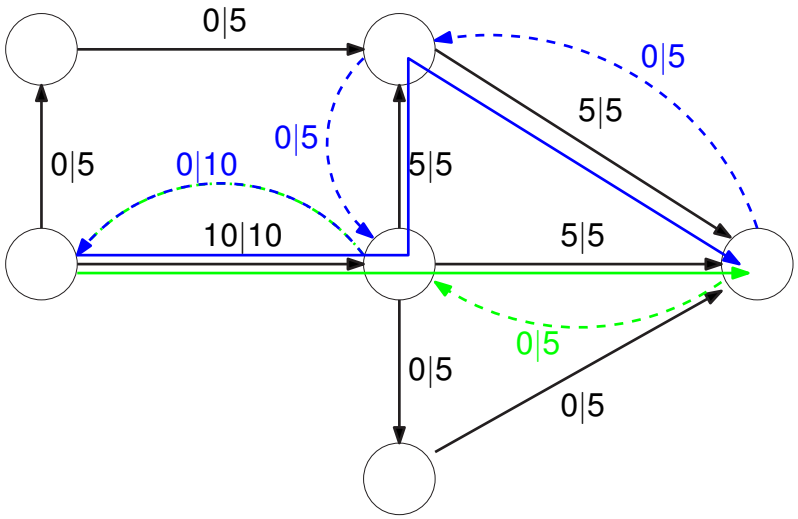
Residualgraph

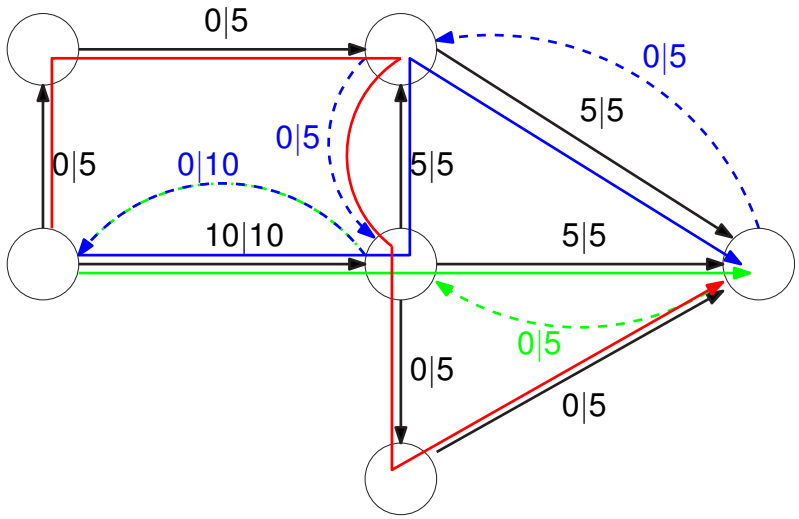
- Verwalten von Restkapazitäten
- Modellierung und Erkennung von “Gegenflüssen”
- $c^f(e) = c(e) - f(e)$: Restkapazität
  Hier: Fluss $f(e)$ und Gesamtkapazität $c(e)$
- $c^f(e^{\text{rev}}) = f(e)$: Restkapazität in Gegenrichtung
- Keine Kanten mit Gewicht 0
- Flüsse über Kanten e und e^{rev} erlaubt
  - Fluss über Kante $\Rightarrow$ Update beider Kanten

Flüsse und Ford-Fulkerson



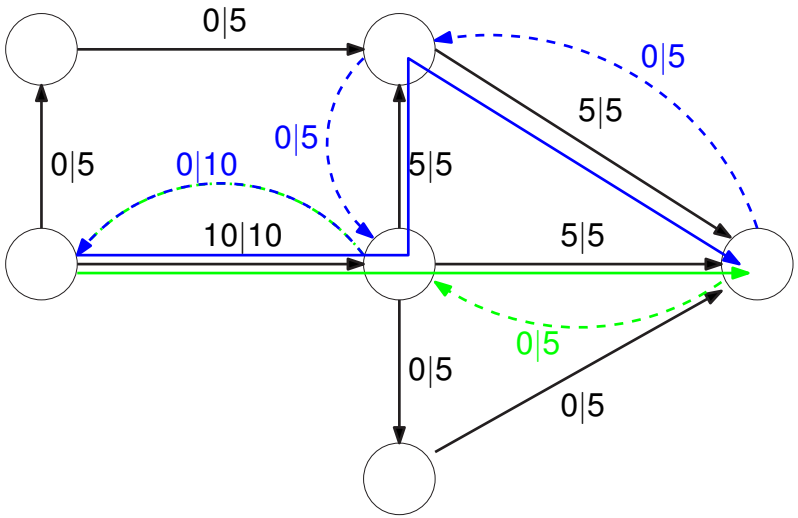
Flüsse und Ford-Fulkerson

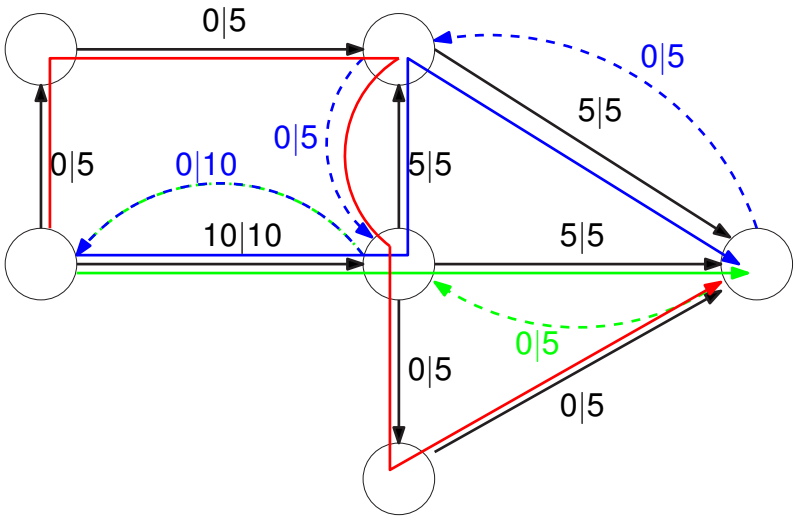


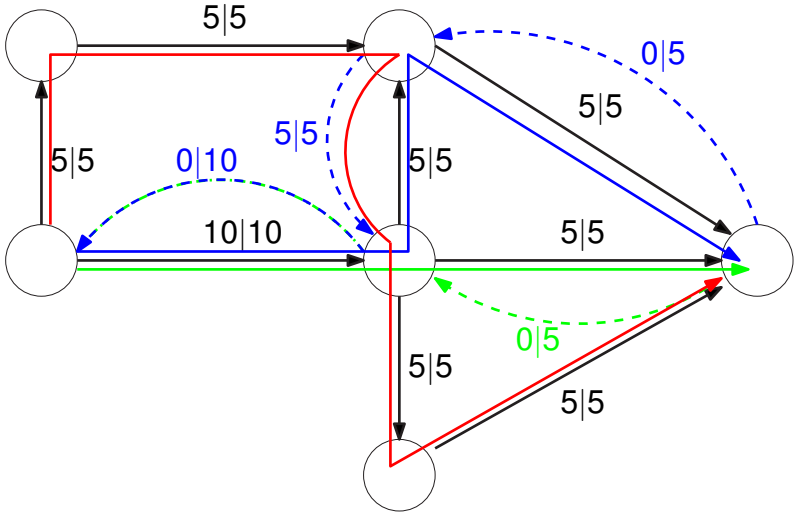


Moritz Laupichler, Nikolai Maas:
Übung 4 – Algorithmen II
Institut für Theoretische Informatik
Algorithm Engineering

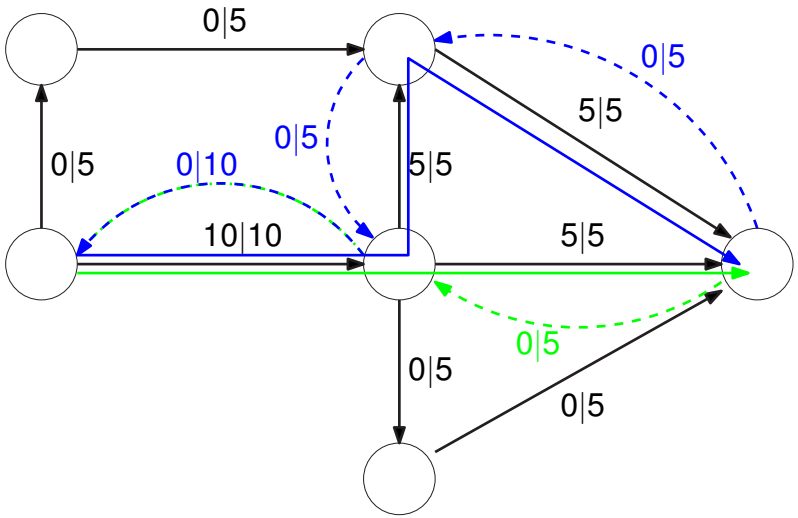
Flüsse und Ford-Fulkerson

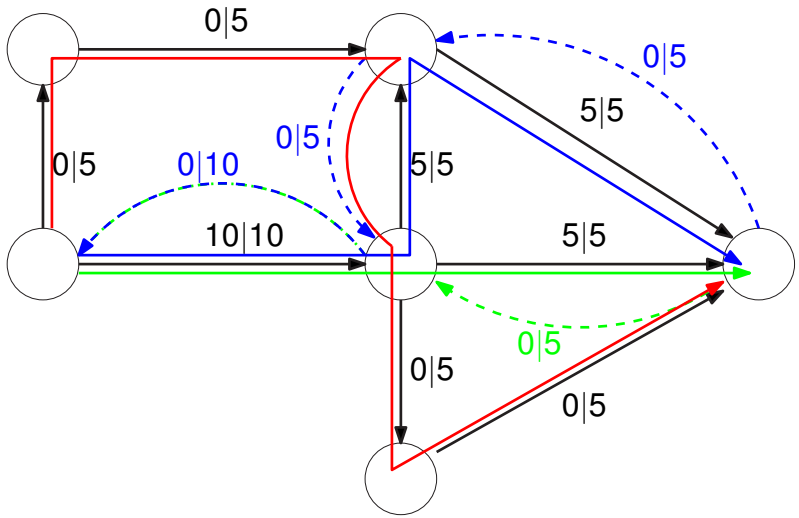


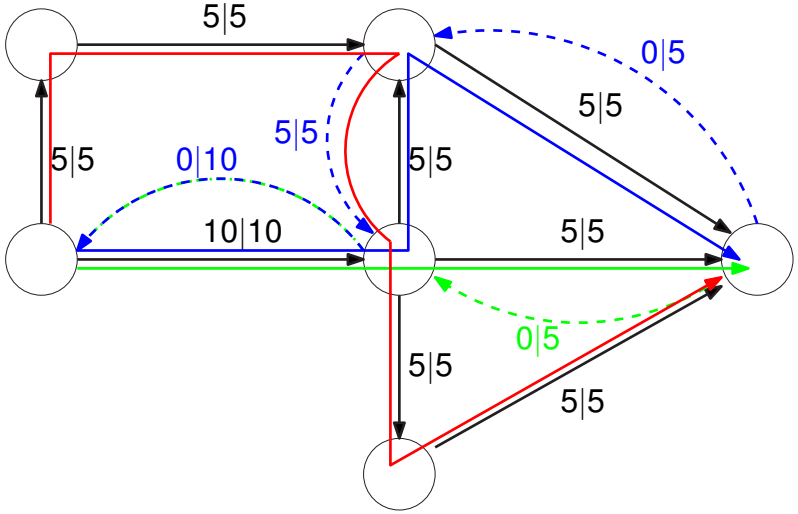


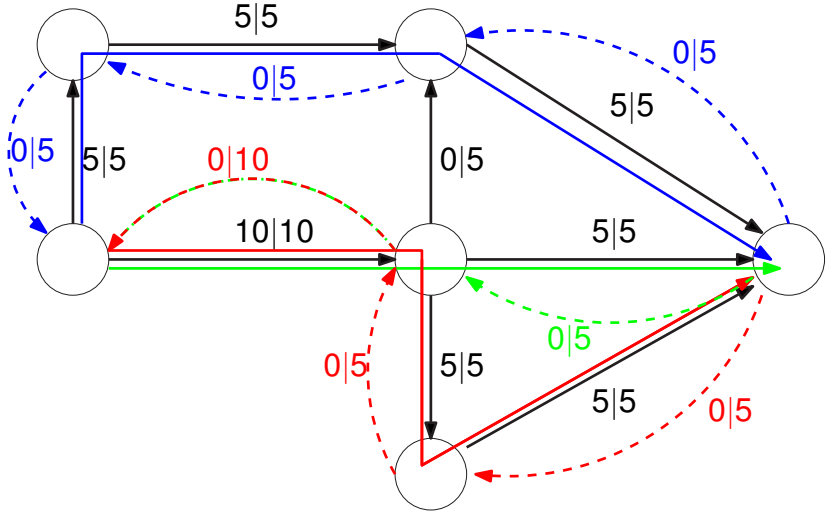


Flüsse und Ford-Fulkerson



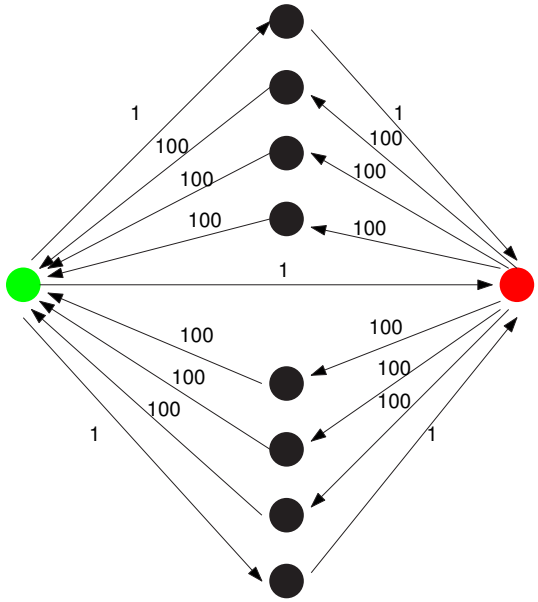






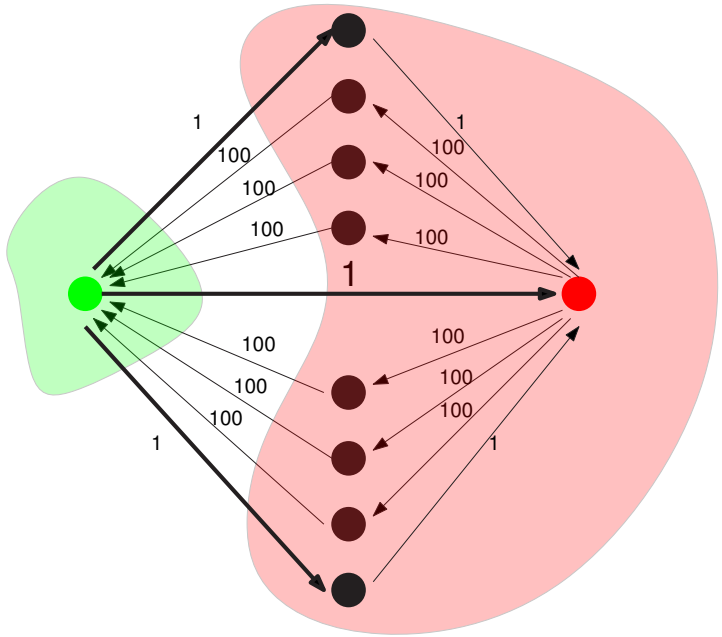
Max Flow - Min Cut

Max Flow - Min Cut



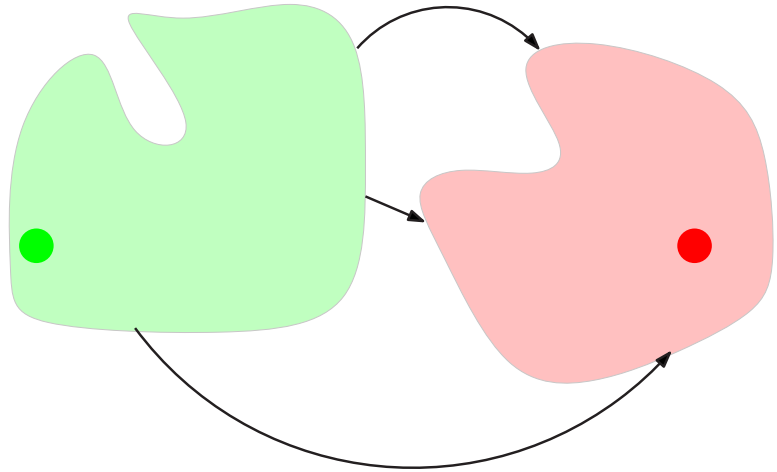
Max Flow - Min Cut

- S-T-Schnitte betrachten nur Kanten $S \rightarrow T$
- Kanten $T \rightarrow S$ werden nicht berücksichtigt
- Flow muss durch alle möglichen S-T-Schnitte
  $\Rightarrow$ Max Flow = Min S-T-Cut



Max Flow - Min Cut

- S-T-Schnitte betrachten nur Kanten $S \to T$
- Kanten $T \to S$ werden nicht berücksichtigt
- Flow muss durch alle möglichen S-T-Schnitte
  $\Rightarrow$ Max Flow = Min S-T-Cut



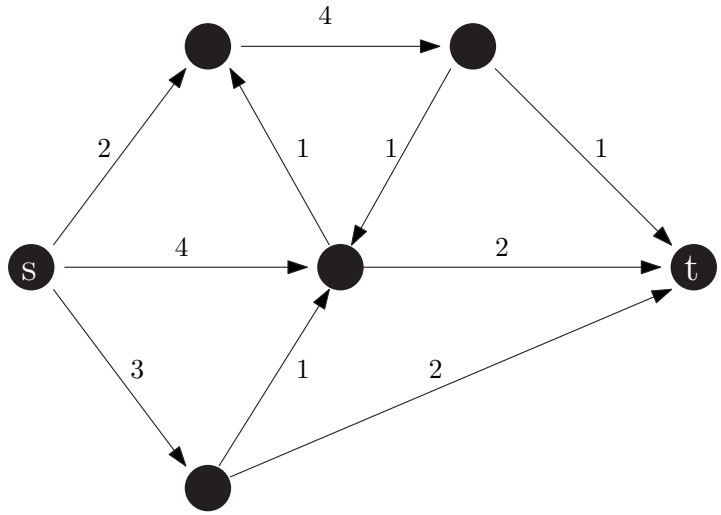
Karlsruher Institut für Technologie

Dinitz

Dinitz

Distanzlabel

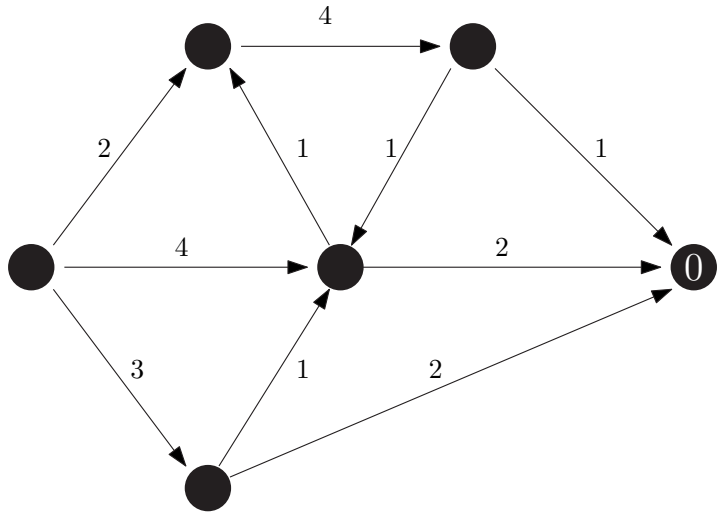
- Geben Distanz im Residualgraphen zur Senke *t* an (hop-based)
- Rückwärtsgerichtete Breitensuche
  - Start bei Knoten *t*
  - Layer: Knoten mit gleicher Distanz zu *s* im BFS-Baum
- *Layer Graph* (Schichtgraph)



Dinitz

Distanzlabel

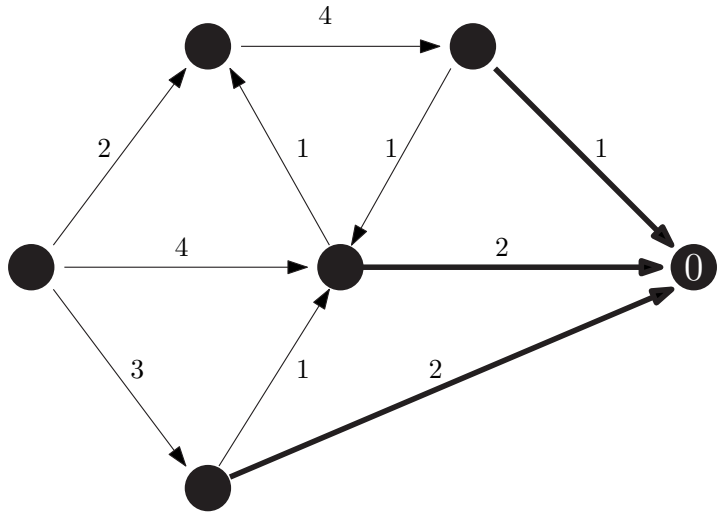
- Geben Distanz im Residualgraphen zur Senke *t* an (hop-based)
- Rückwärtsgerichtete Breitensuche
  - Start bei Knoten *t*
  - Layer: Knoten mit gleicher Distanz zu *s* im BFS-Baum
- *Layer Graph* (Schichtgraph)



Dinitz

Distanzlabel

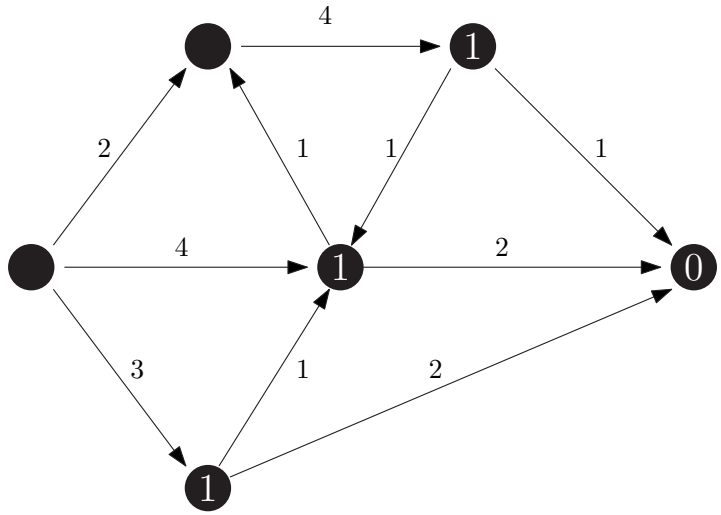
- Geben Distanz im Residualgraphen zur Senke *t* an (hop-based)
- Rückwärtsgerichtete Breitensuche
  - Start bei Knoten *t*
  - Layer: Knoten mit gleicher Distanz zu *s* im BFS-Baum
- *Layer Graph* (Schichtgraph)



Dinitz

Distanzlabel

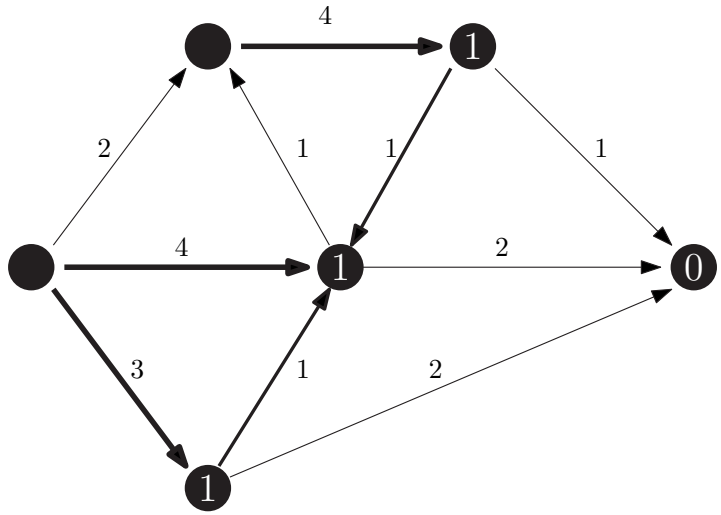
- Geben Distanz im Residualgraphen zur Senke *t* an (hop-based)
- Rückwärtsgerichtete Breitensuche
  - Start bei Knoten *t*
  - Layer: Knoten mit gleicher Distanz zu *s* im BFS-Baum
- *Layer Graph* (Schichtgraph)



Dinitz

Distanzlabel

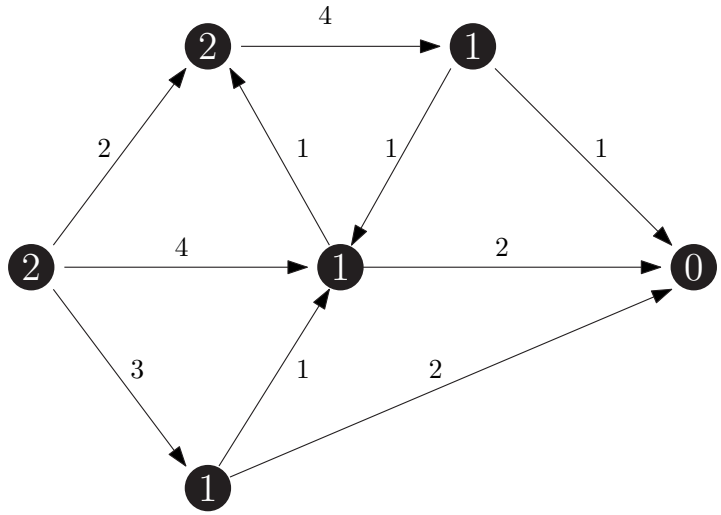
- Geben Distanz im Residualgraphen zur Senke *t* an (hop-based)
- Rückwärtsgerichtete Breitensuche
  - Start bei Knoten *t*
  - Layer: Knoten mit gleicher Distanz zu *s* im BFS-Baum
- *Layer Graph* (Schichtgraph)



Dinitz

Distanzlabel

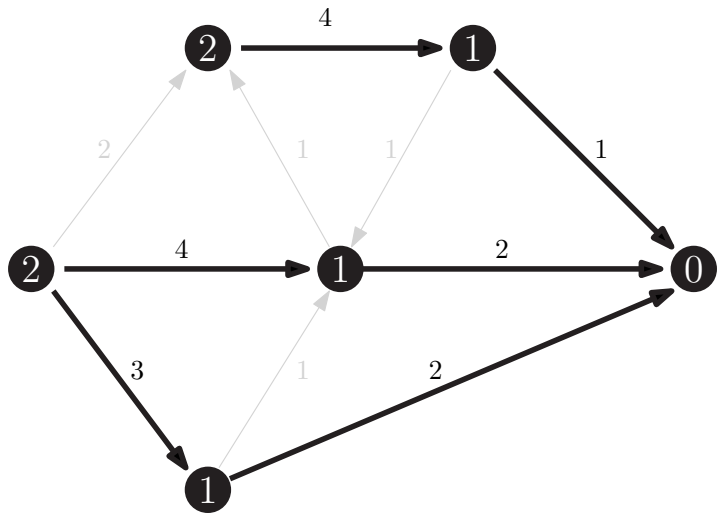
- Geben Distanz im Residualgraphen zur Senke *t* an (hop-based)
- Rückwärtsgerichtete Breitensuche
  - Start bei Knoten *t*
  - Layer: Knoten mit gleicher Distanz zu *s* im BFS-Baum
- *Layer Graph* (Schichtgraph)



Dinitz

Schichtgraph für Graph $G = (V, E)$

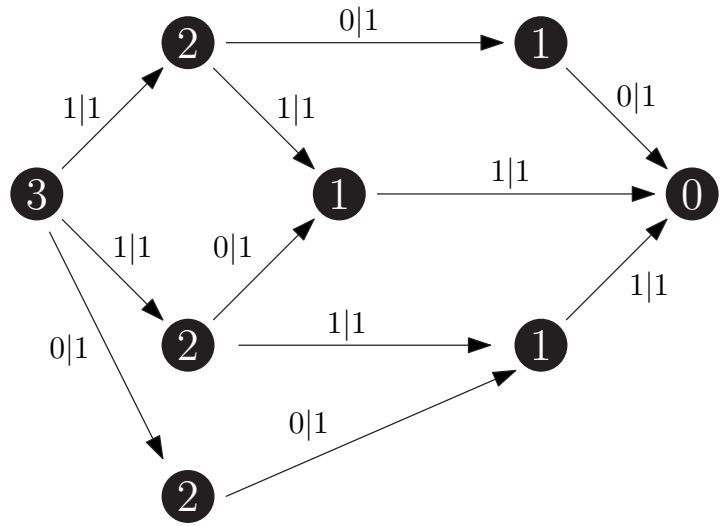
- Knotenmenge $V_S = V$
- $E' = \{e = (u, v) \in E | f(e) < c(e), d(u) = d(v) + 1\}$
- $E'^{rev} = \{e^{rev} = (v, u)^{rev} | f(v, u) > 0, d(u) = d(v) + 1\}$
- Kantenmenge $E_S = E' \cup E'^{rev}$
  $\Rightarrow$ azyklisch



Dinitz

Blockierender Fluss

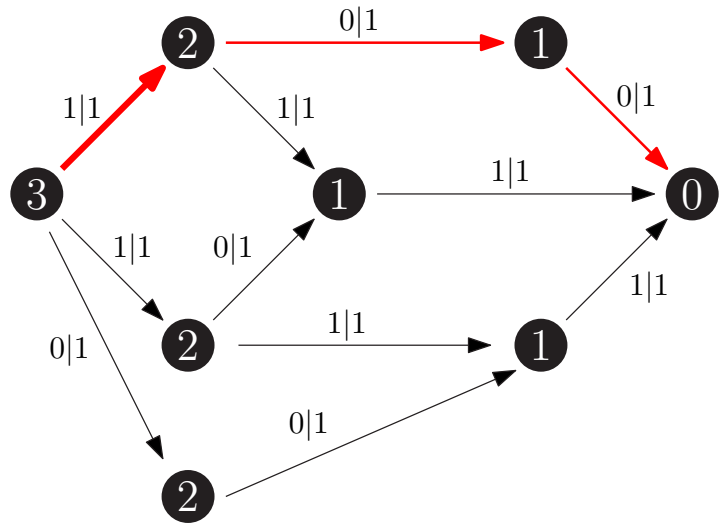
- Kein weiterer Fluss möglich
  "Auf jedem Pfad ist mindestens eine Kante ausgelastet"
- *Blocking flow als atomare Operation*
  - Berechnung auf Schichtgraph
  - Kein Residualgraph
  - Kein Rückfluss möglich
- i.A. nicht maximal auf Schichtgraph und Redisualgraph



Dinitz

Blockierender Fluss

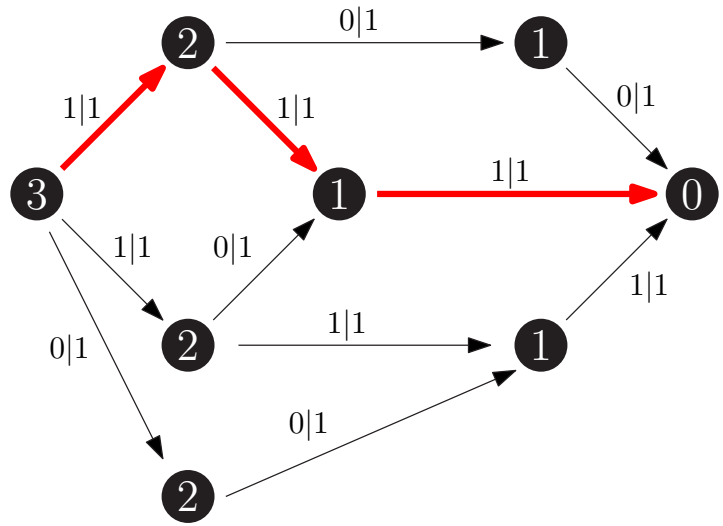
- Kein weiterer Fluss möglich
  "Auf jedem Pfad ist mindestens eine Kante ausgelastet"
- *Blocking flow als atomare Operation*
  - Berechnung auf Schichtgraph
  - Kein Residualgraph
  - Kein Rückfluss möglich
- i.A. nicht maximal auf Schichtgraph und Redisualgraph



Dinitz

Blockierender Fluss

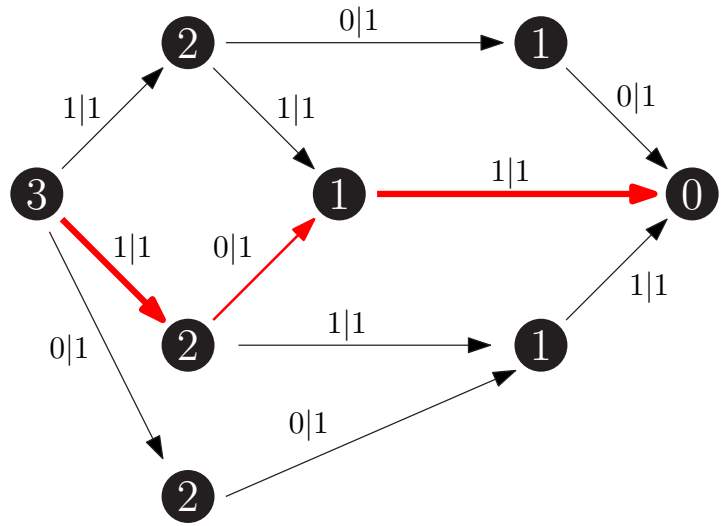
- Kein weiterer Fluss möglich
  "Auf jedem Pfad ist mindestens eine Kante ausgelastet"
- *Blocking flow als atomare Operation*
  - Berechnung auf Schichtgraph
  - Kein Residualgraph
  - Kein Rückfluss möglich
- i.A. nicht maximal auf Schichtgraph und Redisualgraph



Dinitz

Blockierender Fluss

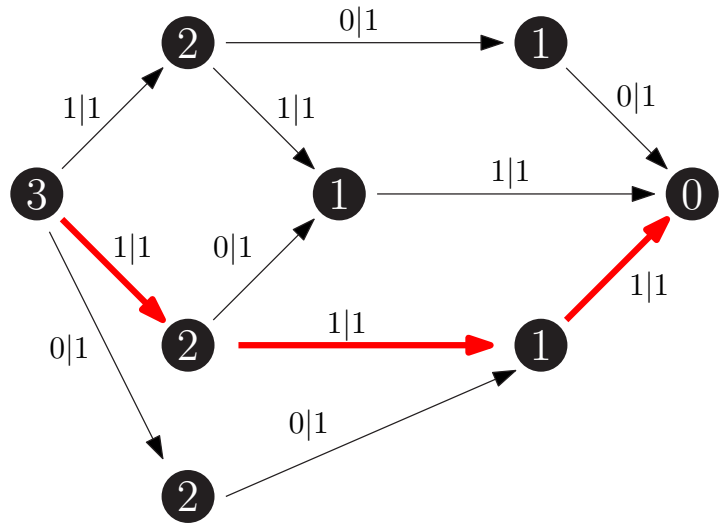
- Kein weiterer Fluss möglich
  "Auf jedem Pfad ist mindestens eine Kante ausgelastet"
- *Blocking flow als atomare Operation*
  - Berechnung auf Schichtgraph
  - Kein Residualgraph
  - Kein Rückfluss möglich
- i.A. nicht maximal auf Schichtgraph und Redisualgraph



Dinitz

Blockierender Fluss

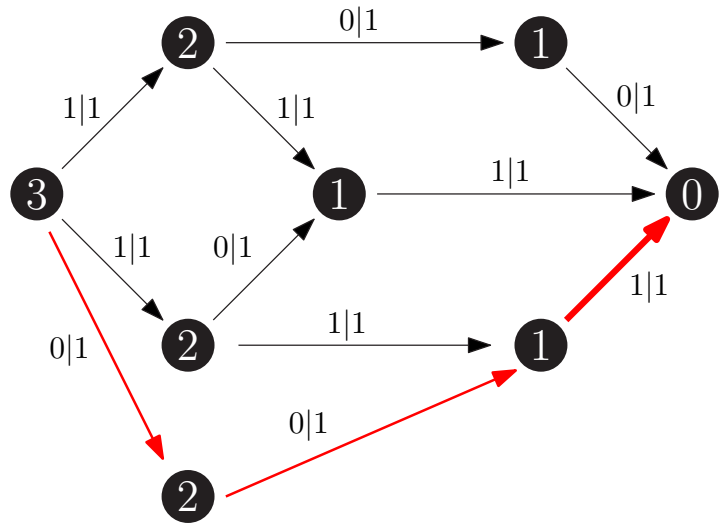
- Kein weiterer Fluss möglich
  "Auf jedem Pfad ist mindestens eine Kante ausgelastet"
- *Blocking flow als atomare Operation*
  - Berechnung auf Schichtgraph
  - Kein Residualgraph
  - Kein Rückfluss möglich
- i.A. nicht maximal auf Schichtgraph und Redisualgraph



Dinitz

Blockierender Fluss

- Kein weiterer Fluss möglich
  "Auf jedem Pfad ist mindestens eine Kante ausgelastet"
- *Blocking flow als atomare Operation*
  - Berechnung auf Schichtgraph
  - Kein Residualgraph
  - Kein Rückfluss möglich
- i.A. nicht maximal auf Schichtgraph und Redisualgraph



Dinitz

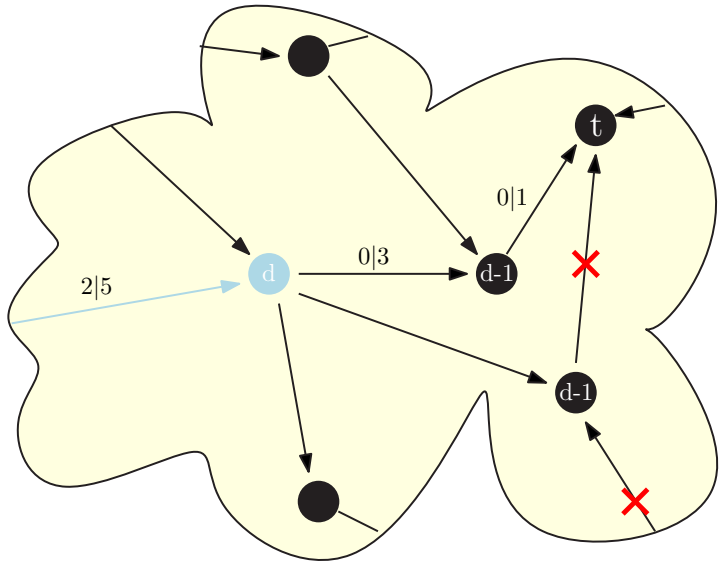
Blockierender Fluss: Operationen

- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
  - extend - gehe einen Knoten näher ans Ziel (Schichtgraph)
  - retreat - Sackgasse gefunden, gehe zurück, lösche Kante
  - breakthrough - Tiefensuche hat Senke erreicht, lösche saturierte Kanten

Dinitz

Blockierender Fluss: Operationen

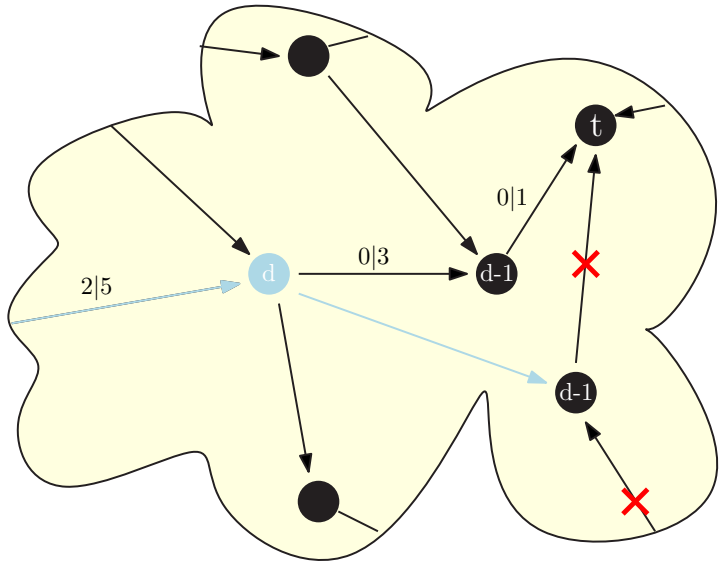
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
  - **extend** - gehe einen Knoten näher ans Ziel (Schichtgraph)
  - retreat - Sackgasse gefunden, gehe zurück, lösche Kante
  - breakthrough - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



Dinitz

Blockierender Fluss: Operationen

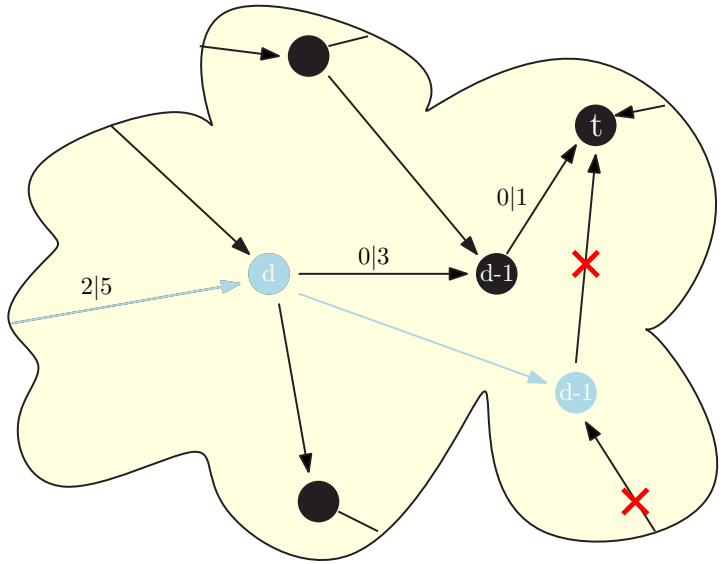
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
  - **extend** - gehe einen Knoten näher ans Ziel (Schichtgraph)
  - retreat - Sackgasse gefunden, gehe zurück, lösche Kante
  - breakthrough - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



Dinitz

Blockierender Fluss: Operationen

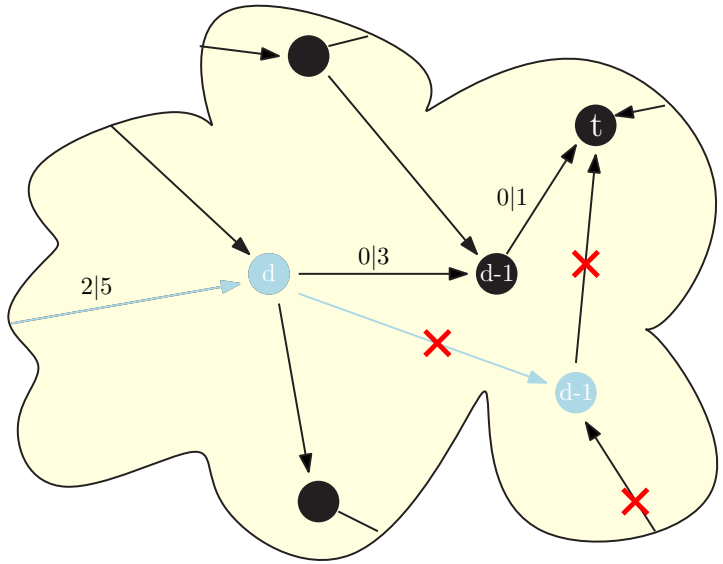
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
  - **extend** - gehe einen Knoten näher ans Ziel (Schichtgraph)
  - retreat - Sackgasse gefunden, gehe zurück, lösche Kante
  - breakthrough - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



Dinitz

Blockierender Fluss: Operationen

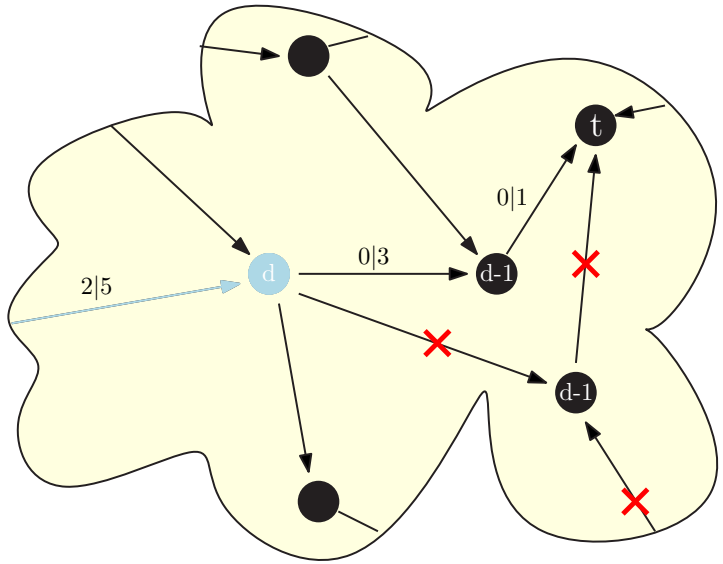
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
  - extend - gehe einen Knoten näher ans Ziel (Schichtgraph)
  - **retreat** - Sackgasse gefunden, gehe zurück, lösche Kante
  - breakthrough - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



Dinitz

Blockierender Fluss: Operationen

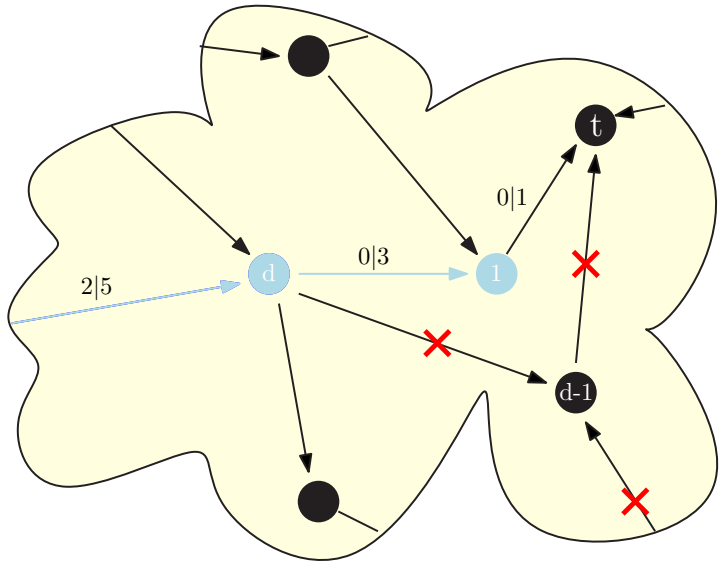
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
  - extend - gehe einen Knoten näher ans Ziel (Schichtgraph)
  - **retreat** - Sackgasse gefunden, gehe zurück, lösche Kante
  - breakthrough - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



Dinitz

Blockierender Fluss: Operationen

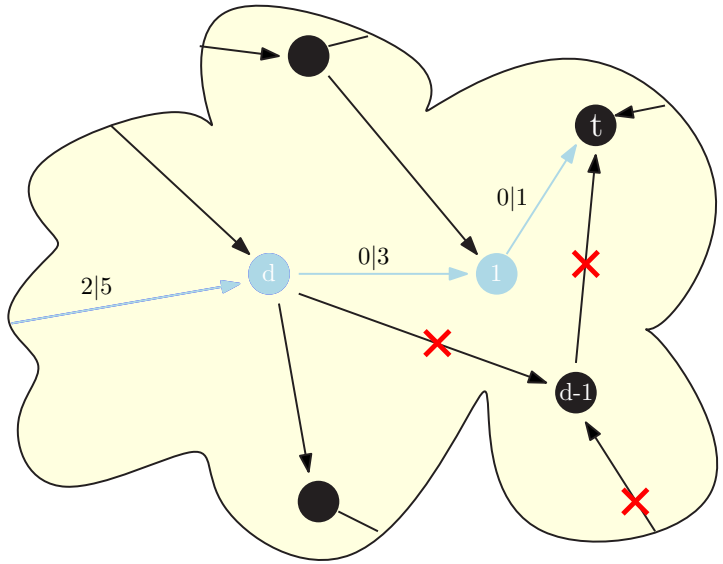
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
  - **extend** - gehe einen Knoten näher ans Ziel (Schichtgraph)
  - retreat - Sackgasse gefunden, gehe zurück, lösche Kante
  - breakthrough - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



Dinitz

Blockierender Fluss: Operationen

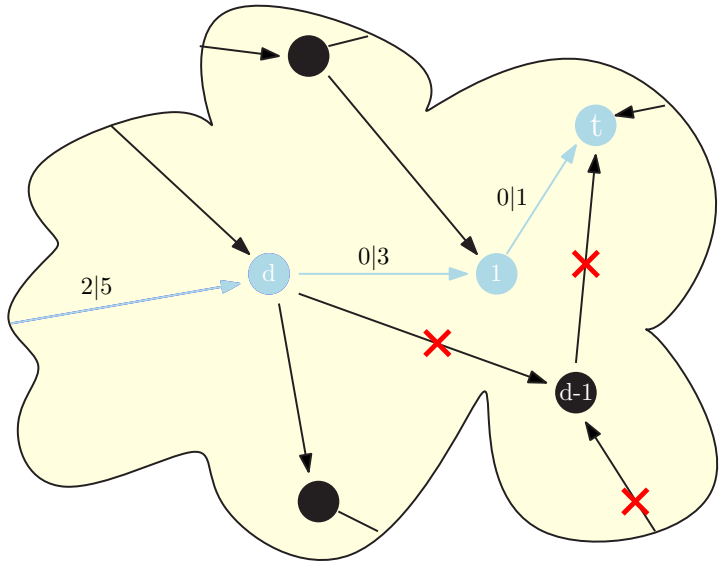
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
  - **extend** - gehe einen Knoten näher ans Ziel (Schichtgraph)
  - retreat - Sackgasse gefunden, gehe zurück, lösche Kante
  - breakthrough - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



Dinitz

Blockierender Fluss: Operationen

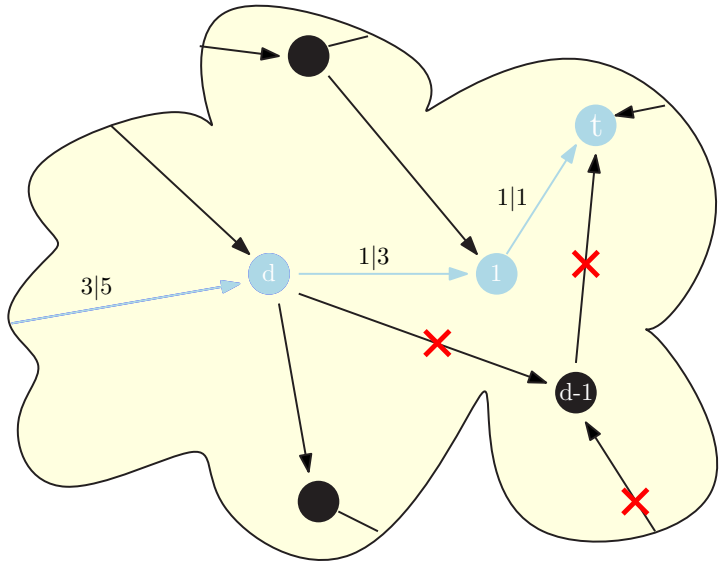
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
  - extend - gehe einen Knoten näher ans Ziel (Schichtgraph)
  - retreat - Sackgasse gefunden, gehe zurück, lösche Kante
  - **breakthrough** - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



Dinitz

Blockierender Fluss: Operationen

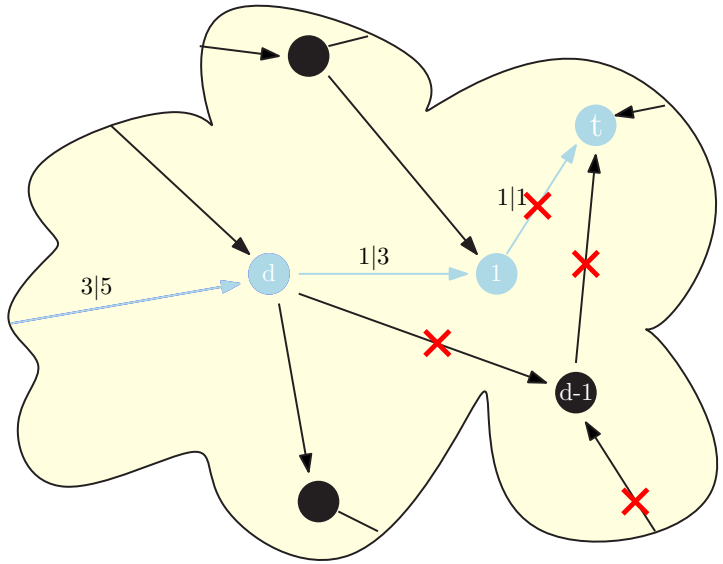
- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
  - extend - gehe einen Knoten näher ans Ziel (Schichtgraph)
  - retreat - Sackgasse gefunden, gehe zurück, lösche Kante
  - **breakthrough** - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



Dinitz

Blockierender Fluss: Operationen

- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
  - extend - gehe einen Knoten näher ans Ziel (Schichtgraph)
  - retreat - Sackgasse gefunden, gehe zurück, lösche Kante
  - **breakthrough** - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



Dinitz

Kosten pro Blockierendem Fluss

- #breakthrough $\leq m$
  - Jedes breakthrough saturiert mindestens eine Kante
    $\Rightarrow$ Kein breakthrough über saturierte Kante mehr möglich
  - **Laufzeit** $O(n)$
- #retreat $\leq m$
  - Jedes retreat löscht eine Kante
  - **Laufzeit** $O(1)$
- #extends $\leq$ #retreats $+ n \cdot$ #breakthrough
  - **Retreat:** Vorher **ein extend**
  - **Breakthrough:** Vorher $\leq n$ **erfolgreiche extends**
    Schichtgraph ist azyklisch
  - **Laufzeit** $O(1)$
- Blockierender Fluss in $O(nm)$

Dinitz

Laufzeit

- Pro Phase
  - Rückwärtsgerichtete Breitensuche: **Laufzeit** $O(n+m)$
  - Blockierender Fluss: **Laufzeit** $O(nm)$
- Phase in $O(nm)$
- #Phasen $\leq n$

  (Lemma 1: Jede Phase erhöht Label von s um mindestens 1)

  $\Rightarrow$ Gesamtlaufzeit $O(n^2 m)$

Dinitz

Laufzeit

- Pro Phase
  - Rückwärtsgerichtete Breitensuche: **Laufzeit** $O(n+m)$
  - Blockierender Fluss: **Laufzeit** $O(nm)$
- Phase in $O(nm)$
- #Phasen $\leq n$ (Beweis in Vorlesung)

  (Lemma 1: Jede Phase erhöht Label von s um mindestens 1)

  $\Rightarrow$ Gesamtlaufzeit $O(n^2 m)$

Dinitz

Kosten pro Phase - *Unit Capacity Network*

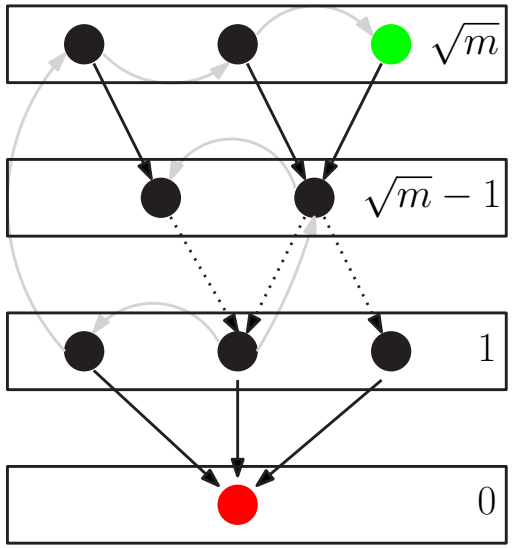
Amortisierte Kosten

- Retreat und breakthrough
  - Löscht alle beteiligten Kanten
    $\Rightarrow$ Jede Kante nur an einem retreat/breakthrough beteiligt
- Extend
  - Pro Kante ein *extend*
  - Bezahlt für breakthrough und retreat
  - #extends $= O(m)$
  - Kosten $O(1)$
- Phase in $O(m+n)$

Dinitz

Anzahl Phasen - *Unit Capacity Network*

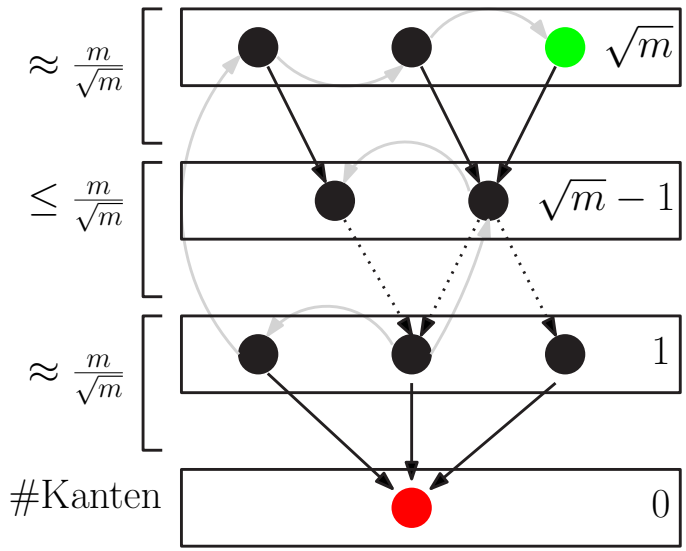
- Nach $\sqrt{m}$ Phasen
  - Graph hat mindestens $\sqrt{m}$ Layer
    (Lemma 1: Jede Phase erhöht Label von s um mindestens 1)
- Durchschnittlich $m/\sqrt{m}$ Kanten pro Layer
  $\Rightarrow$ es gibt Layer i mit höchstens $\sqrt{m}$ Kanten zu Layer $i-1$
- Induziert Schnitt mit Kapazität $\leq \sqrt{m}$ (im Residualgraphen), zwischen $S = \{v | d(v) \geq i\}$ und $T = V \setminus S$
- Jede Phase erhöht Fluss um ≥ 1
  $\Rightarrow$ Zusätzlich $\leq \sqrt{m}$ Phasen



Dinitz

Anzahl Phasen - *Unit Capacity Network*

- Nach $\sqrt{m}$ Phasen
  - Graph hat mindestens $\sqrt{m}$ Layer
    (Lemma 1: Jede Phase erhöht Label von s um mindestens 1)
- Durchschnittlich $m/\sqrt{m}$ Kanten pro Layer
  $\Rightarrow$ es gibt Layer i mit höchstens $\sqrt{m}$ Kanten zu Layer $i-1$
- Induziert Schnitt mit Kapazität $\leq \sqrt{m}$ (im Residualgraphen), zwischen $S = \{v | d(v) \geq i\}$ und $T = V \setminus S$
- Jede Phase erhöht Fluss um ≥ 1
  $\Rightarrow$ Zusätzlich $\leq \sqrt{m}$ Phasen



Preflow-Push

Preflow-Push

Wiederholung

Bezeichnungen

- aktiver Knoten
  - Knoten v aktiv gdw. $\text{excess}(v) = \text{inflow}(v) - \text{outflow}(v) > 0$
- gültige Kante
  - Kante $(v, w) \in G^f$ ist gültig, wenn Level $d(v) = d(w) + 1$

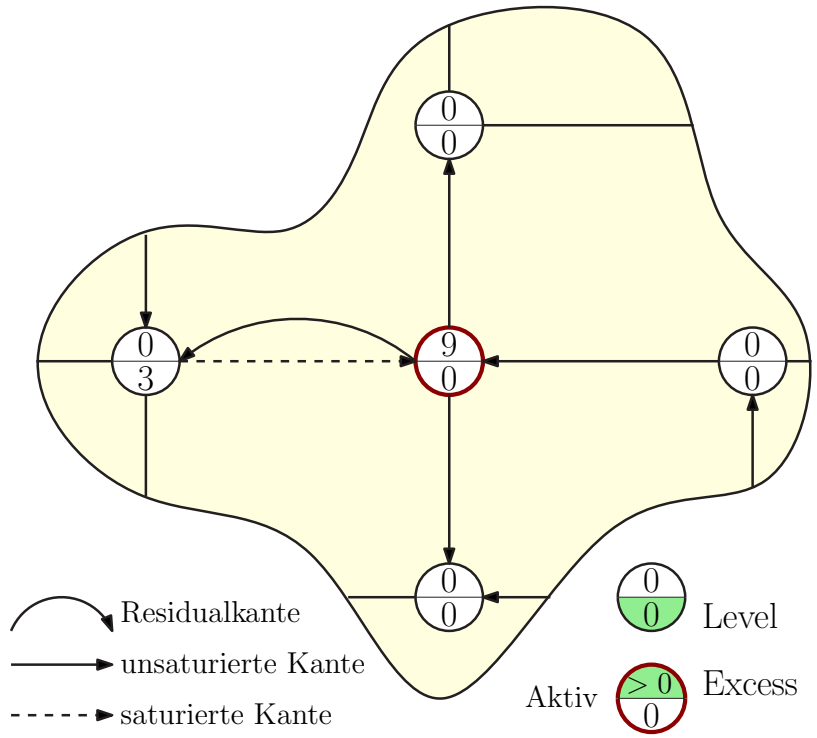
allgemeiner Ablauf

1. `wähle aktiven Knoten` v
2. `falls gültige Kante` (v, w) `existiert: push`
   - `schiebe Fluss entlang` (v, w)
3. `ansonsten: relabel`

Preflow-Push

Wiederholung

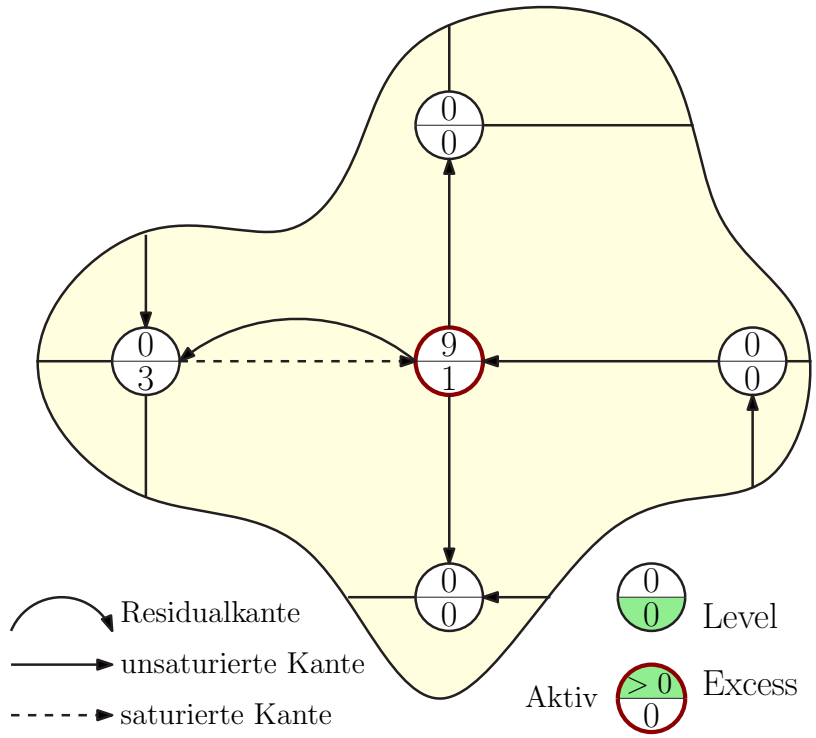
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
  inflow > outflow
- push
  - schiebe Fluss entlang Kante (u, v)
  - Vorraussetzung: $d(u) = d(v) + 1$
- **relabel**
  - erhöht Level eines Knoten
  - nur zulässig wenn *push* vom Knoten nicht möglich
    $\Rightarrow$ erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph



Preflow-Push

Wiederholung

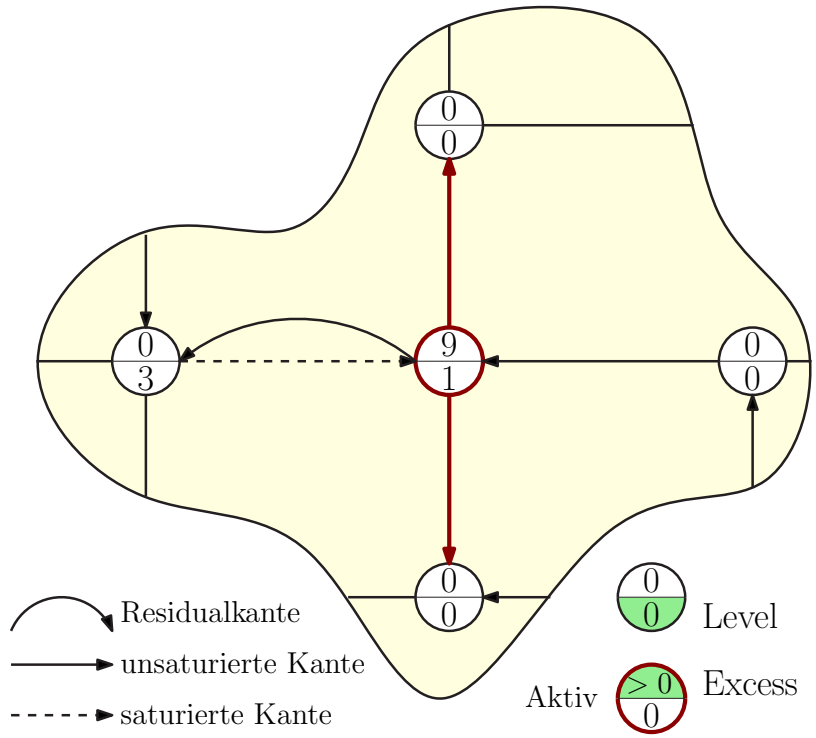
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
  inflow > outflow
- push
  - schiebe Fluss entlang Kante (u, v)
  - Vorraussetzung: $d(u) = d(v) + 1$
- **relabel**
  - erhöht Level eines Knoten
  - nur zulässig wenn *push* vom Knoten nicht möglich
    $\Rightarrow$ erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph



Preflow-Push

Wiederholung

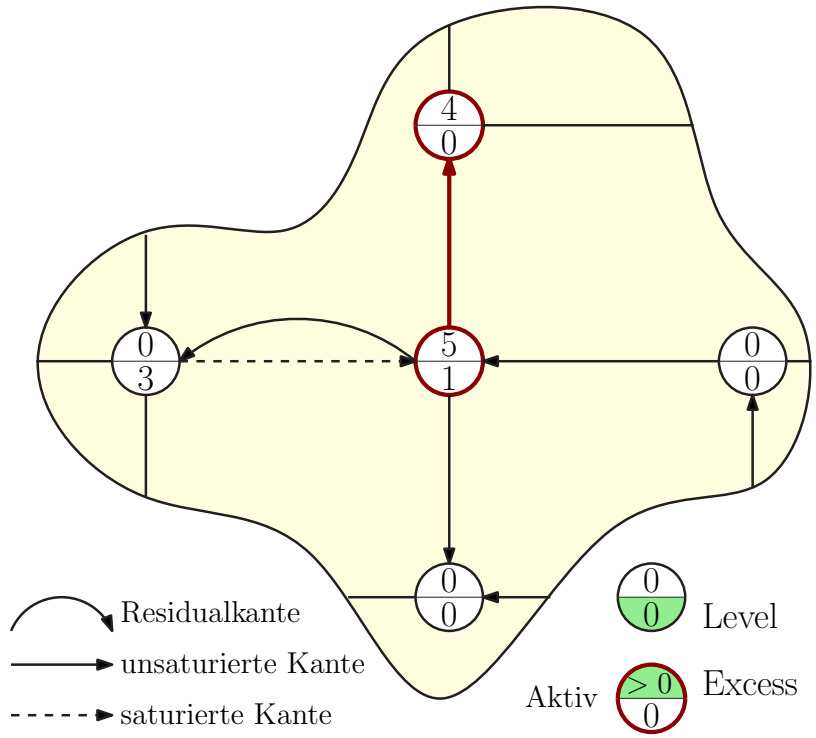
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
  inflow > outflow
- **push**
  - schiebe Fluss entlang Kante (u, v)
  - Vorraussetzung: $d(u) = d(v) + 1$
- relabel
  - erhöht Level eines Knoten
  - nur zulässig wenn *push* vom Knoten nicht möglich
    $\Rightarrow$ erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph



Preflow-Push

Wiederholung

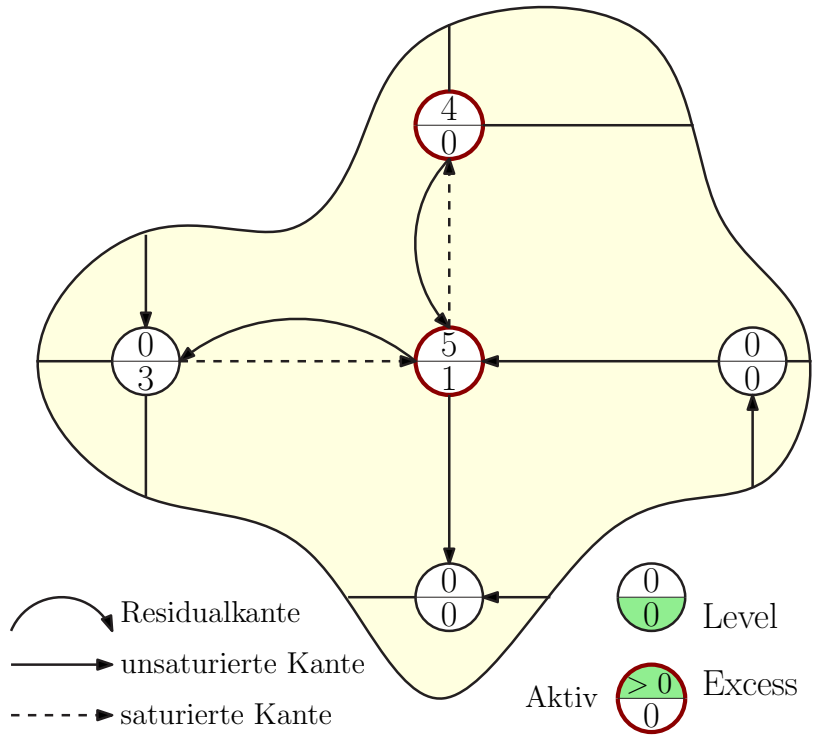
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
  inflow > outflow
- **push**
  - schiebe Fluss entlang Kante (u, v)
  - Vorraussetzung: $d(u) = d(v) + 1$
- relabel
  - erhöht Level eines Knoten
  - nur zulässig wenn *push* vom Knoten nicht möglich
    $\Rightarrow$ erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph



Preflow-Push

Wiederholung

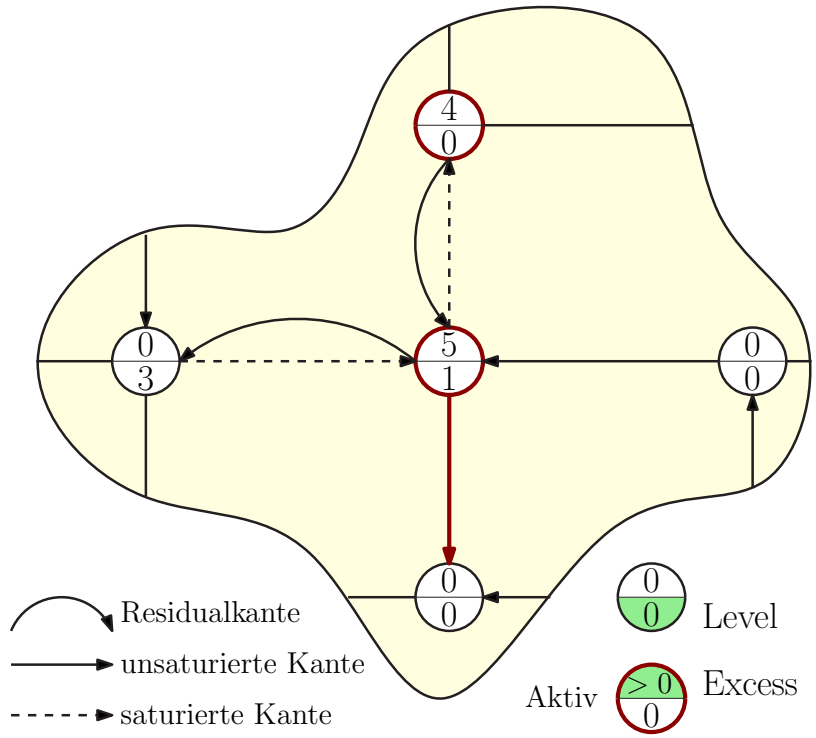
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
  inflow > outflow
- **push**
  - schiebe Fluss entlang Kante (u, v)
  - Vorraussetzung: $d(u) = d(v) + 1$
- relabel
  - erhöht Level eines Knoten
  - nur zulässig wenn *push* vom Knoten nicht möglich
    $\Rightarrow$ erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph



Preflow-Push

Wiederholung

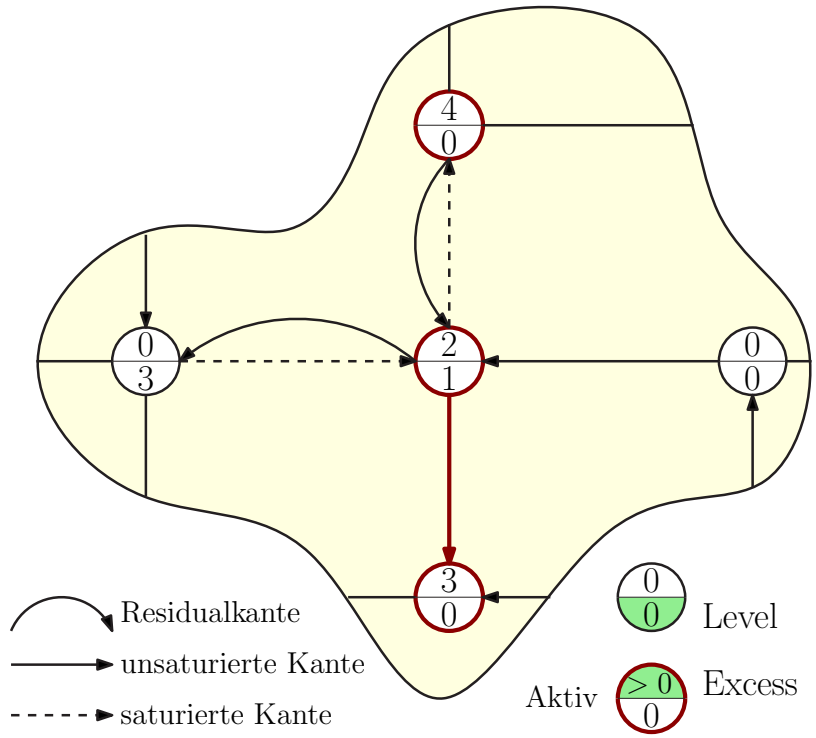
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
  inflow > outflow
- **push**
  - schiebe Fluss entlang Kante (u, v)
  - Vorraussetzung: $d(u) = d(v) + 1$
- relabel
  - erhöht Level eines Knoten
  - nur zulässig wenn *push* vom Knoten nicht möglich
    $\Rightarrow$ erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph



Preflow-Push

Wiederholung

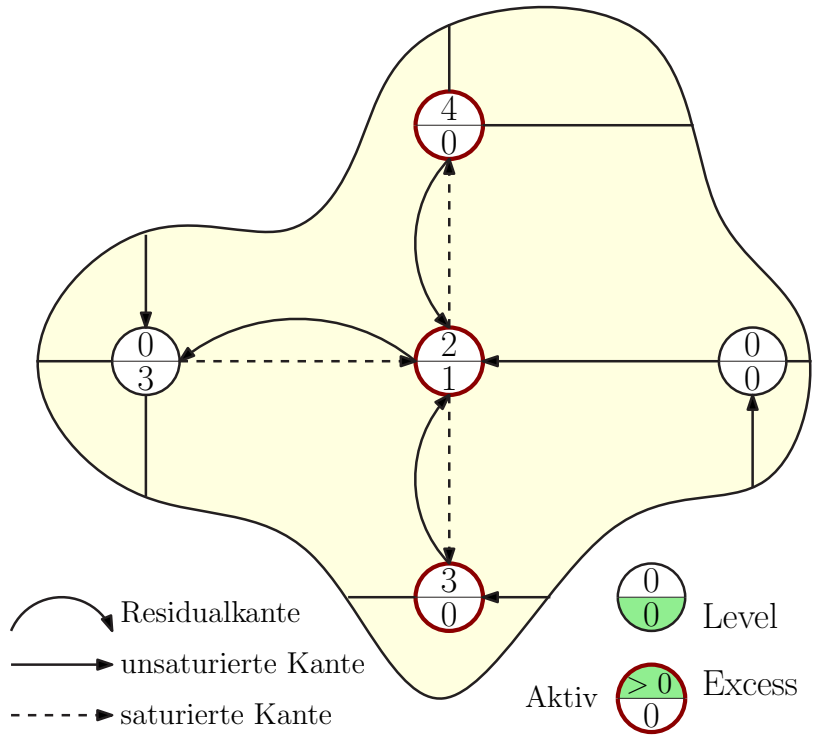
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
  inflow > outflow
- **push**
  - schiebe Fluss entlang Kante (u, v)
  - Vorraussetzung: $d(u) = d(v) + 1$
- relabel
  - erhöht Level eines Knoten
  - nur zulässig wenn *push* vom Knoten nicht möglich
    $\Rightarrow$ erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph



Preflow-Push

Wiederholung

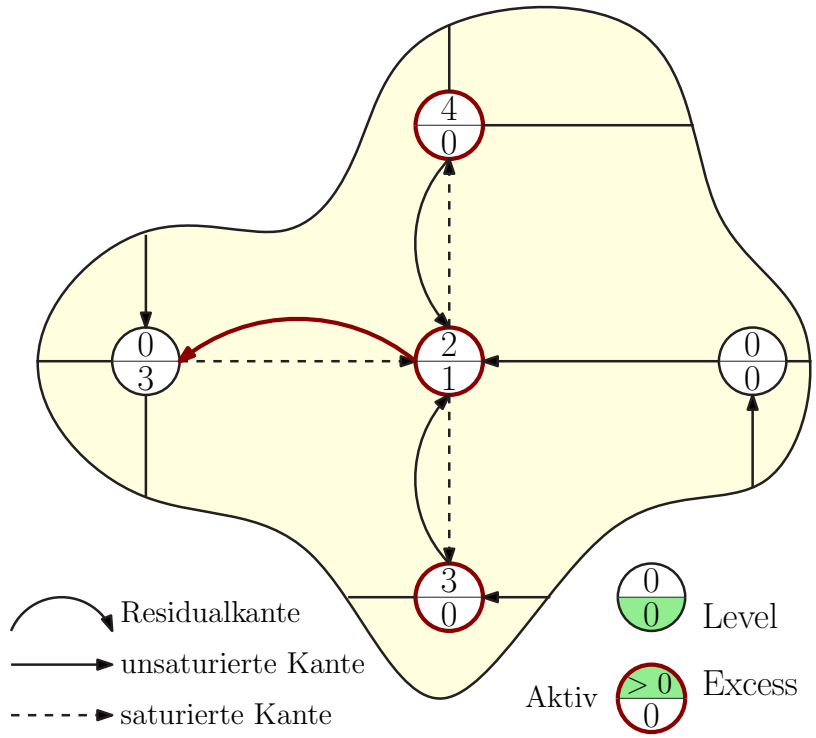
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
  inflow > outflow
- **push**
  - schiebe Fluss entlang Kante (u, v)
  - Vorraussetzung: $d(u) = d(v) + 1$
- relabel
  - erhöht Level eines Knoten
  - nur zulässig wenn *push* vom Knoten nicht möglich
    $\Rightarrow$ erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph



Preflow-Push

Wiederholung

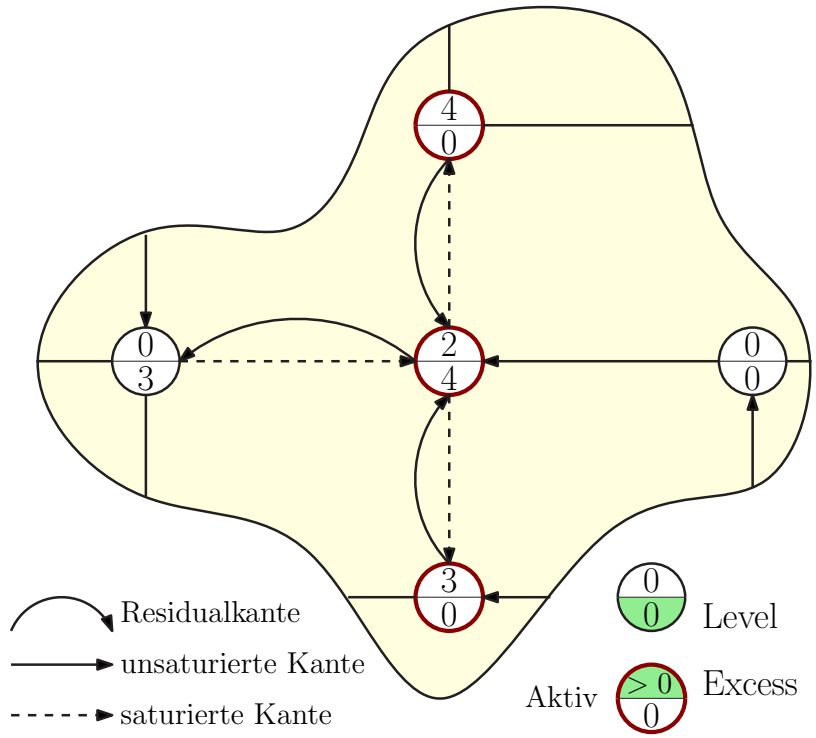
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
  inflow > outflow
- push
  - schiebe Fluss entlang Kante (u, v)
  - Vorraussetzung: $d(u) = d(v) + 1$
- **relabel**
  - erhöht Level eines Knoten
  - nur zulässig wenn *push* vom Knoten nicht möglich
    $\Rightarrow$ erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph



Preflow-Push

Wiederholung

- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
  inflow > outflow
- push
  - schiebe Fluss entlang Kante (u, v)
  - Vorraussetzung: $d(u) = d(v) + 1$
- **relabel**
  - erhöht Level eines Knoten
  - nur zulässig wenn *push* vom Knoten nicht möglich
    $\Rightarrow$ erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph



Preflow-Push

Wiederholung

Bezeichnungen

- aktiver Knoten
  - Knoten v aktiv gdw. $\text{excess}(v) = \text{inflow}(v) - \text{outflow}(v) > 0$
- gültige Kante
  - Kante $(v, w) \in G^f$ ist gültig, wenn Level $d(v) = d(w) + 1$

allgemeiner Ablauf

1. wähle aktiven Knoten v
2. falls gültige Kante (v, w) existiert: push
   - schiebe Fluss entlang (v, w)
3. ansonsten: relabel
   - erhöhe Level von v

Preflow-Push

Wiederholung

Bezeichnungen

- aktiver Knoten
  - Knoten v aktiv gdw. $\text{excess}(v) = \text{inflow}(v) - \text{outflow}(v) > 0$
- gültige Kante
  - Kante $(v, w) \in G^f$ ist gültig, wenn Level $d(v) = d(w) + 1$

allgemeiner Ablauf

1. `wähle aktiven Knoten` v — WELCHEN?
2. `falls gültige Kante` (v, w) `existiert: push` — WELCHE?
   - `schiebe Fluss entlang` (v, w) — WIEVIEL?
3. `ansonsten: relabel`
   - `erhöhe Level von` v — WIEVIEL?

Preflow-Push

Wiederholung

Bezeichnungen

- aktiver Knoten
  - Knoten v aktiv gdw. $\text{excess}(v) = \text{inflow}(v) - \text{outflow}(v) > 0$
- gültige Kante
  - Kante $(v, w) \in G^f$ ist gültig, wenn Level $d(v) = d(w) + 1$

allgemeiner Ablauf

1. `wähle aktiven Knoten` v — WELCHEN?
2. `falls gültige Kante` (v, w) `existiert: push` — WELCHE?
   - `schiebe Fluss entlang` (v, w) — WIEVIEL?
     $f_{(v,w)} = f_{(v,w)} + \min\{c^f_{(v,w)}, excess(v)\}$
3. `ansonsten: relabel`
   - `erhöhe Level von` v — WIEVIEL?
     $d(v) = d(v) + 1$

Preflow-Push

Übersicht

Unterschiedliche Auswahl des aktiven Knotens:

- *generic preflow-push* $\mathcal{O}(n^2 m)$
- *FIFO preflow-push* $\mathcal{O}(n^3)$
- *highest-level preflow-push* $\mathcal{O}(n^2 \sqrt{m})$

Unterschiedliches `relabel`:

- *aggressive local relabeling*
- *global relabeling*
- *gap heuristic*

$\Rightarrow$ nur Heuristiken, aber in der Praxis deutliche Beschleunigung!

Preflow-Push

Heuristiken

Relabeling

- *aggressive local relabeling*
- *global relabeling*
- *gap heuristic*

Knotenauswahl

- *two-phase approach*

Preflow-Push

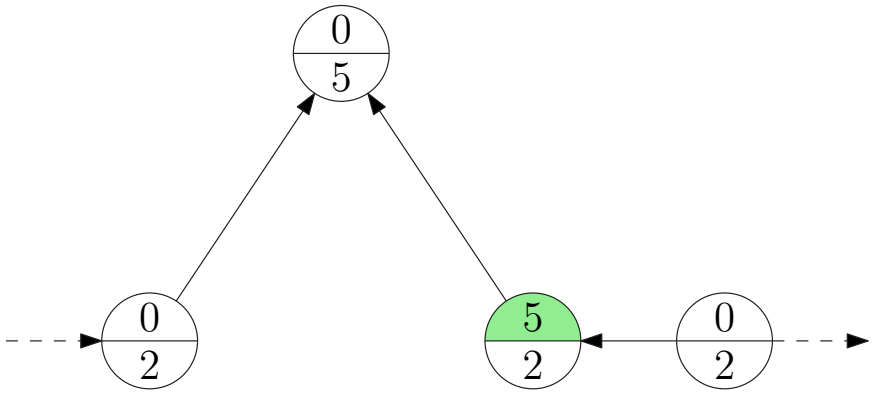
Heuristiken

- *aggressive local relabeling*
  - erhöhe Level in einem Schritt, so dass gültige Kante existiert

$$d(v) = 1 + \min_{(v,w) \in E^f} d(w)$$

$d(w) \geq d(v)$, wenn keine gültige Kante in G^f existiert!

Beispiel 1



Preflow-Push

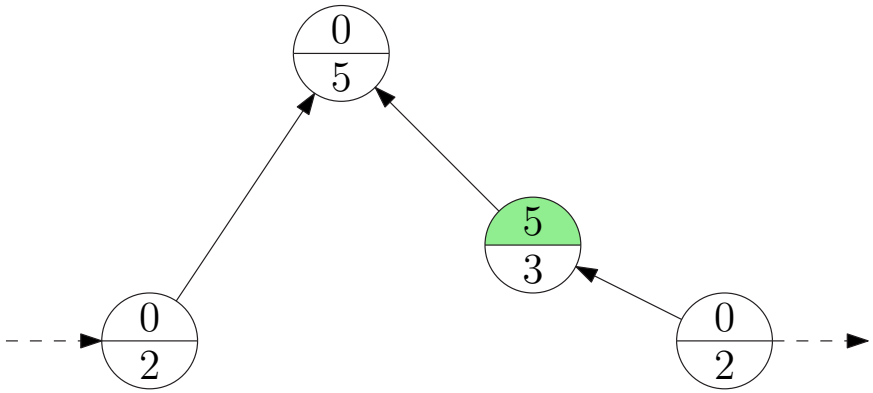
Heuristiken

- *aggressive local relabeling*
  - erhöhe Level in einem Schritt, so dass gültige Kante existiert

$$d(v) = 1 + \min_{(v,w) \in E^f} d(w)$$

$d(w) \geq d(v)$, wenn keine gültige Kante in G^f existiert!

Beispiel 1



Preflow-Push

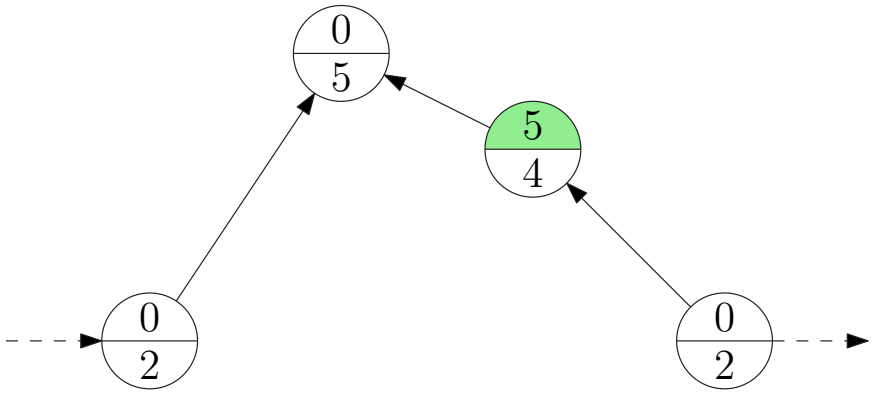
Heuristiken

- *aggressive local relabeling*
  - erhöhe Level in einem Schritt, so dass gültige Kante existiert

$$d(v) = 1 + \min_{(v,w) \in E^f} d(w)$$

$d(w) \geq d(v)$, wenn keine gültige Kante in G^f existiert!

Beispiel 1



Preflow-Push

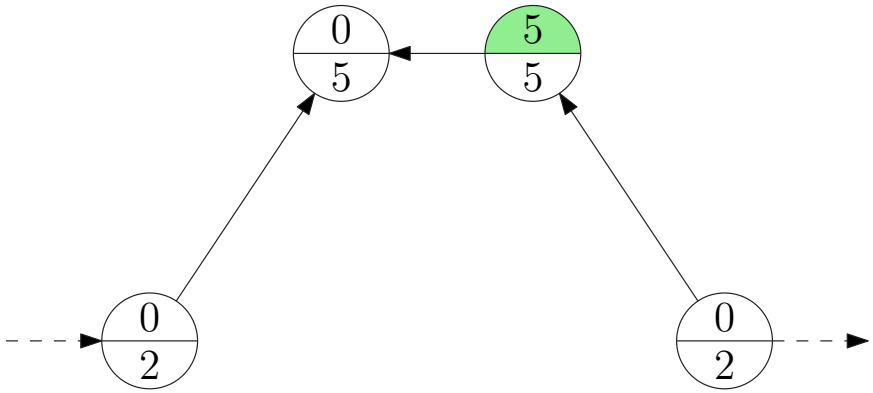
Heuristiken

- *aggressive local relabeling*
  - erhöhe Level in einem Schritt, so dass gültige Kante existiert

$$d(v) = 1 + \min_{(v,w) \in E^f} d(w)$$

$d(w) \geq d(v)$, wenn keine gültige Kante in G^f existiert!

Beispiel 1



Preflow-Push

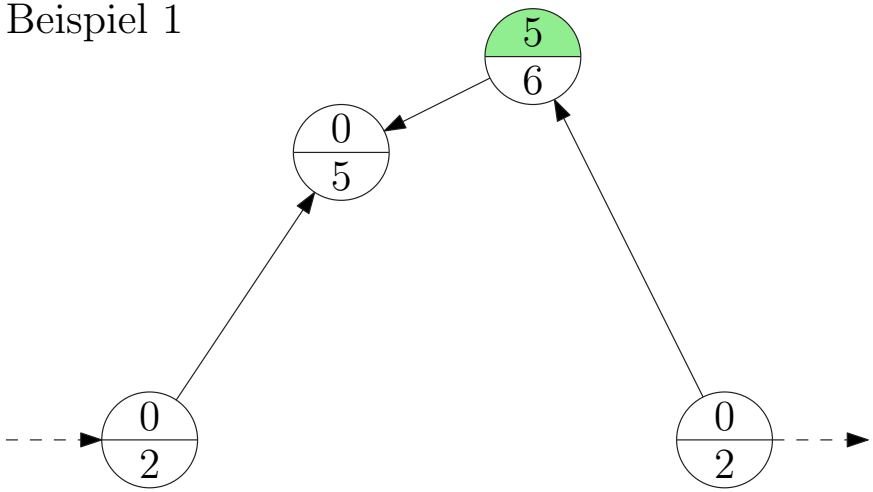
Heuristiken

- *aggressive local relabeling*
  - erhöhe Level in einem Schritt, so dass gültige Kante existiert

$$d(v) = 1 + \min_{(v,w) \in E^f} d(w)$$

$d(w) \geq d(v)$, wenn keine gültige Kante in G^f existiert!

Beispiel 1



Preflow-Push

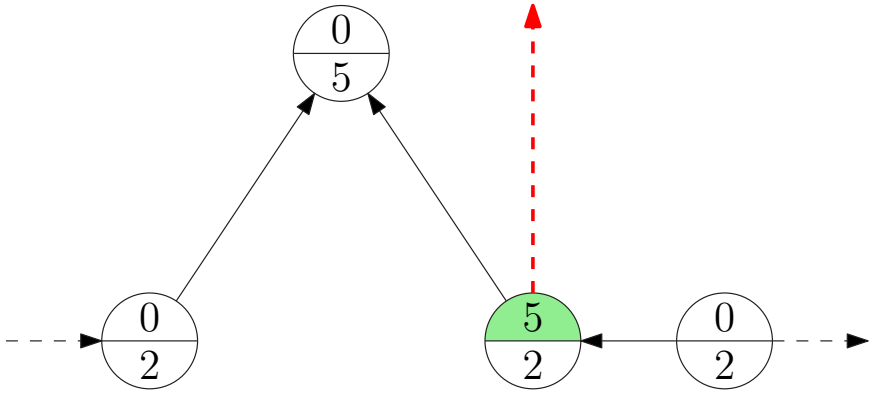
Heuristiken

- *aggressive local relabeling*
  - erhöhe Level in einem Schritt, so dass gültige Kante existiert

$$d(v) = 1 + \min_{(v,w) \in E^f} d(w)$$

$d(w) \geq d(v)$, wenn keine gültige Kante in G^f existiert!

Beispiel 1



Preflow-Push

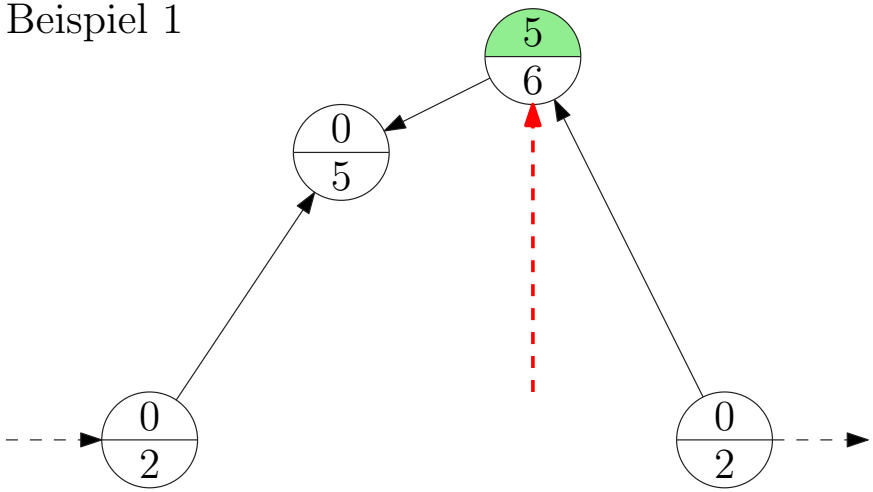
Heuristiken

- *aggressive local relabeling*
  - erhöhe Level in einem Schritt, so dass gültige Kante existiert

$$d(v) = 1 + \min_{(v,w) \in E^f} d(w)$$

$d(w) \geq d(v)$, wenn keine gültige Kante in G^f existiert!

Beispiel 1



Preflow-Push

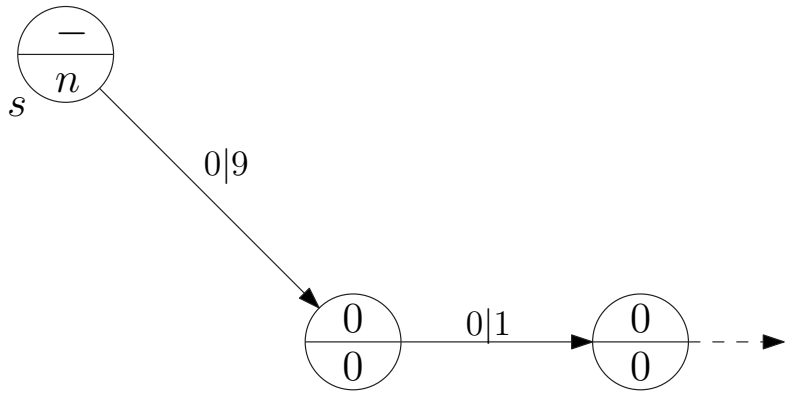
Heuristiken

- *aggressive local relabeling*
  - erhöhe Level in einem Schritt, so dass gültige Kante existiert

$$d(v) = 1 + \min_{(v,w) \in E^f} d(w)$$

$d(w) \geq d(v)$, wenn keine gültige Kante in G^f existiert!

Beispiel 2



Preflow-Push

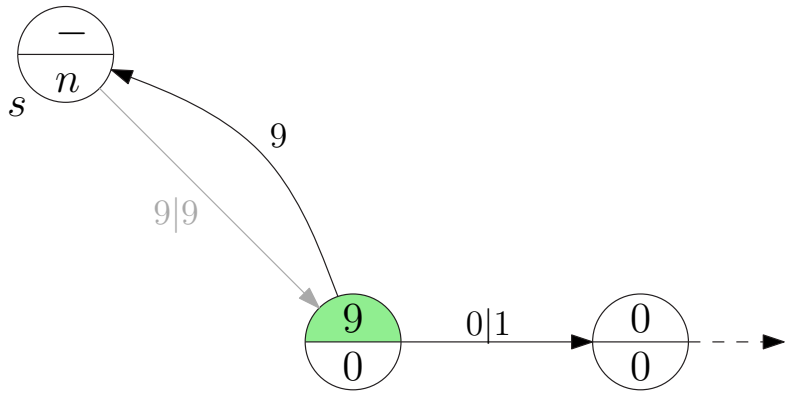
Heuristiken

- *aggressive local relabeling*
  - erhöhe Level in einem Schritt, so dass gültige Kante existiert

$$d(v) = 1 + \min_{(v,w) \in E^f} d(w)$$

$d(w) \geq d(v)$, wenn keine gültige Kante in G^f existiert!

Beispiel 2



Preflow-Push

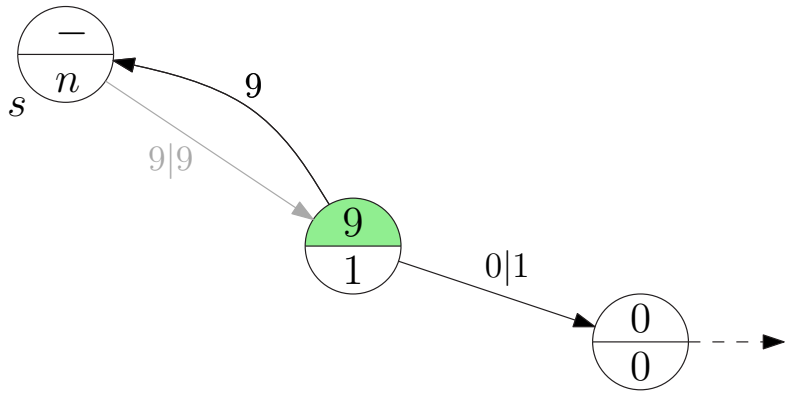
Heuristiken

- *aggressive local relabeling*
  - erhöhe Level in einem Schritt, so dass gültige Kante existiert

$$d(v) = 1 + \min_{(v,w) \in E^f} d(w)$$

$d(w) \geq d(v)$, wenn keine gültige Kante in G^f existiert!

Beispiel 2



Preflow-Push

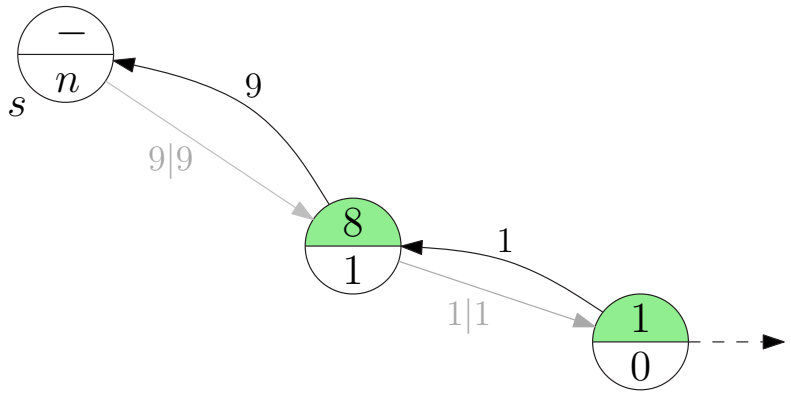
Heuristiken

- *aggressive local relabeling*
  - erhöhe Level in einem Schritt, so dass gültige Kante existiert

$$d(v) = 1 + \min_{(v,w) \in E^f} d(w)$$

$d(w) \geq d(v)$, wenn keine gültige Kante in G^f existiert!

Beispiel 2



Preflow-Push

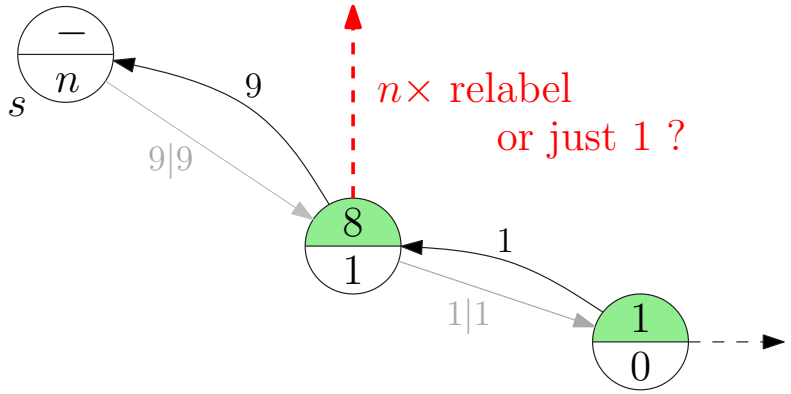
Heuristiken

- *aggressive local relabeling*
  - erhöhe Level in einem Schritt, so dass gültige Kante existiert

$$d(v) = 1 + \min_{(v,w) \in E^f} d(w)$$

$d(w) \geq d(v)$, wenn keine gültige Kante in G^f existiert!

Beispiel 2



Preflow-Push

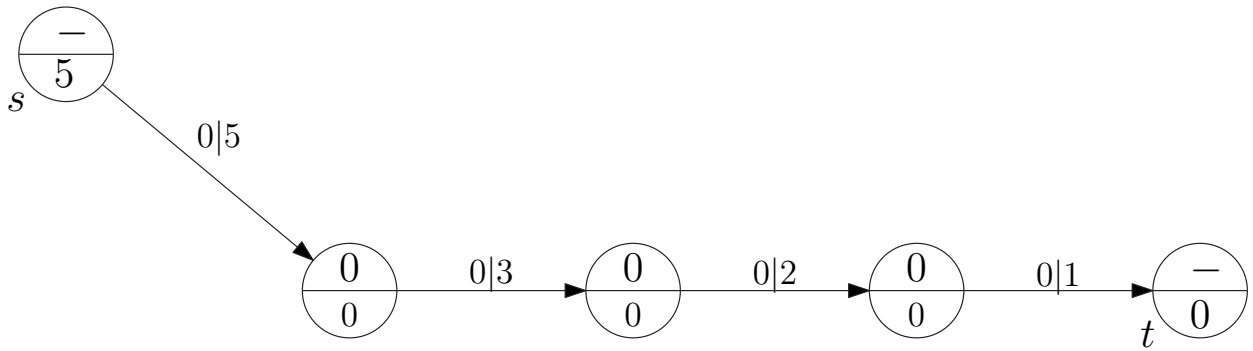
Heuristiken

- *global relabeling*
  - setze Levels auf

$$d(v) = \begin{cases} \mu(v,t) & \text{falls t von v erreichbar} \\ n + \mu(v,s) & \text{sonst, falls s von v erreichbar} \\ 2n-1 & \text{sonst} \end{cases}$$

  - Berechnung der Distanzen $\mu(v,\cdot)$ über Breitensuche in $\mathcal{O}(m)$
    nur alle $\Omega(m)$ Schritte, damit Kosten $\mathcal{O}(1)$ pro Schritt

Beispiel 1



Preflow-Push

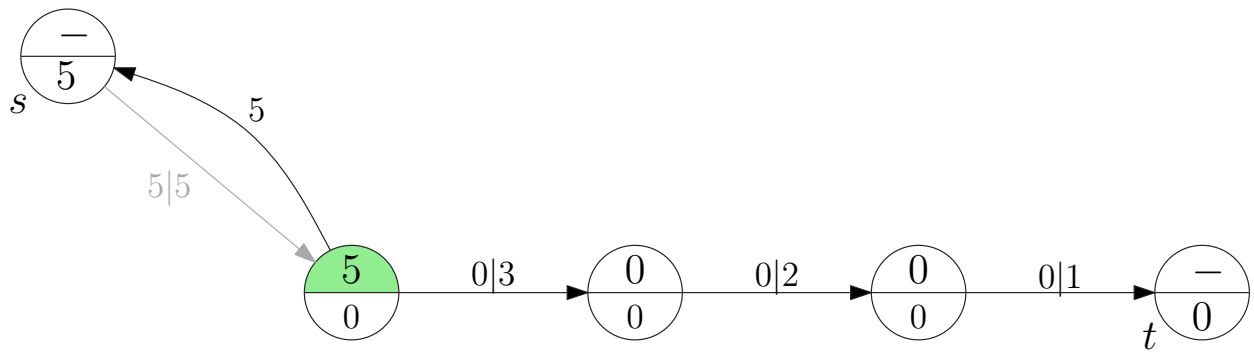
Heuristiken

- *global relabeling*
  - setze Levels auf

$$d(v) = \begin{cases} \mu(v,t) & \text{falls t von v erreichbar} \\ n + \mu(v,s) & \text{sonst, falls s von v erreichbar} \\ 2n-1 & \text{sonst} \end{cases}$$

  - Berechnung der Distanzen $\mu(v,\cdot)$ über Breitensuche in $\mathcal{O}(m)$
    nur alle $\Omega(m)$ Schritte, damit Kosten $\mathcal{O}(1)$ pro Schritt

Beispiel 1



Preflow-Push

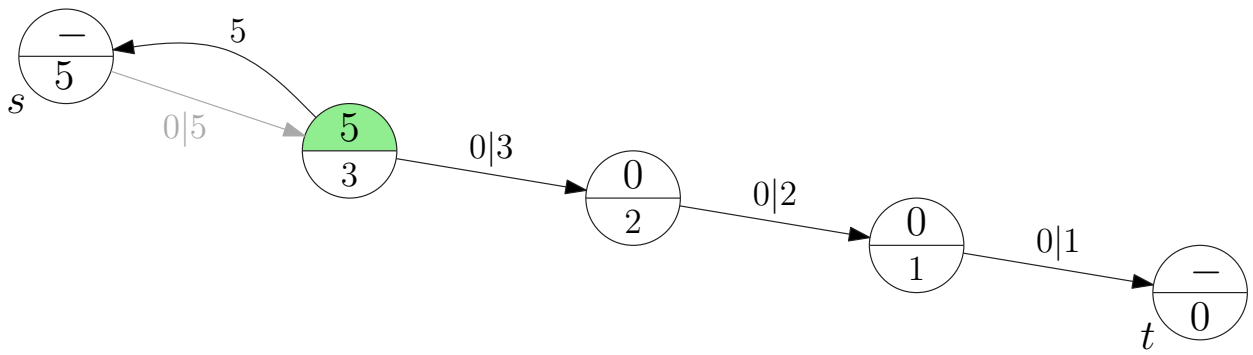
Heuristiken

- *global relabeling*
  - setze Levels auf

$$d(v) = \begin{cases} \mu(v,t) & \text{falls t von v erreichbar} \\ n + \mu(v,s) & \text{sonst, falls s von v erreichbar} \\ 2n-1 & \text{sonst} \end{cases}$$

  - Berechnung der Distanzen $\mu(v,\cdot)$ über Breitensuche in $\mathcal{O}(m)$
    nur alle $\Omega(m)$ Schritte, damit Kosten $\mathcal{O}(1)$ pro Schritt

Beispiel 1



Preflow-Push

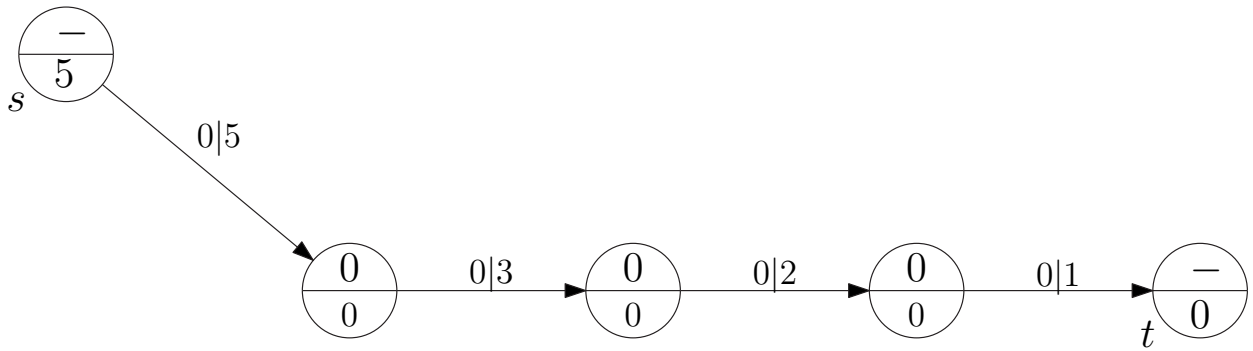
Heuristiken

- *global relabeling*
  - setze Levels auf

$$d(v) = \begin{cases} \mu(v,t) & \text{falls t von v erreichbar} \\ n + \mu(v,s) & \text{sonst, falls s von v erreichbar} \\ 2n-1 & \text{sonst} \end{cases}$$

  - Berechnung der Distanzen $\mu(v,\cdot)$ über Breitensuche in $\mathcal{O}(m)$
    nur alle $\Omega(m)$ Schritte, damit Kosten $\mathcal{O}(1)$ pro Schritt

Beispiel 2



Preflow-Push

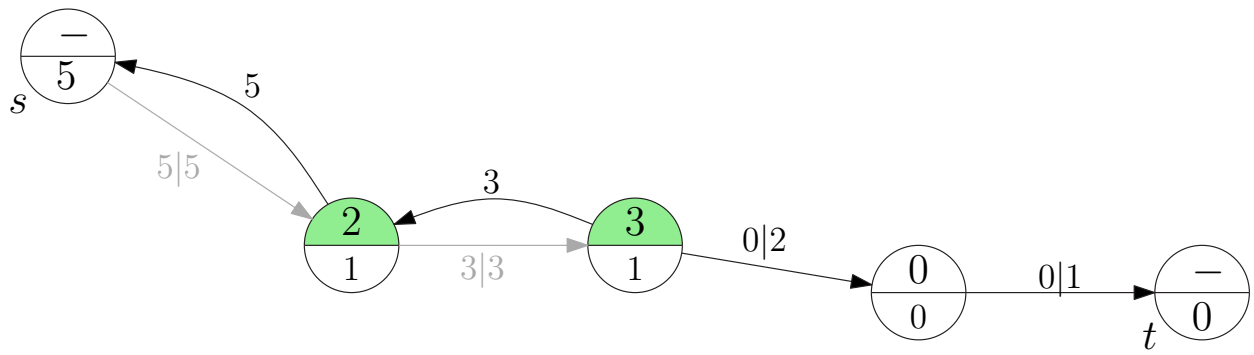
Heuristiken

- *global relabeling*
  - setze Levels auf

$$d(v) = \begin{cases} \mu(v,t) & \text{falls t von v erreichbar} \\ n + \mu(v,s) & \text{sonst, falls s von v erreichbar} \\ 2n-1 & \text{sonst} \end{cases}$$

  - Berechnung der Distanzen $\mu(v,\cdot)$ über Breitensuche in $\mathcal{O}(m)$
    nur alle $\Omega(m)$ Schritte, damit Kosten $\mathcal{O}(1)$ pro Schritt

Beispiel 2



Preflow-Push

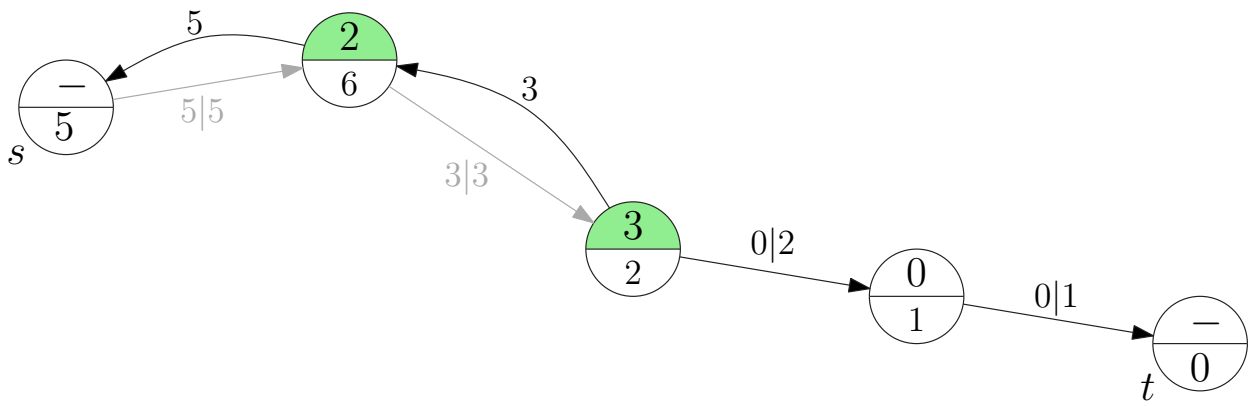
Heuristiken

- *global relabeling*
  - setze Levels auf

$$d(v) = \begin{cases} \mu(v,t) & \text{falls t von v erreichbar} \\ n + \mu(v,s) & \text{sonst, falls s von v erreichbar} \\ 2n-1 & \text{sonst} \end{cases}$$

  - Berechnung der Distanzen $\mu(v,\cdot)$ über Breitensuche in $\mathcal{O}(m)$
    nur alle $\Omega(m)$ Schritte, damit Kosten $\mathcal{O}(1)$ pro Schritt

Beispiel 2



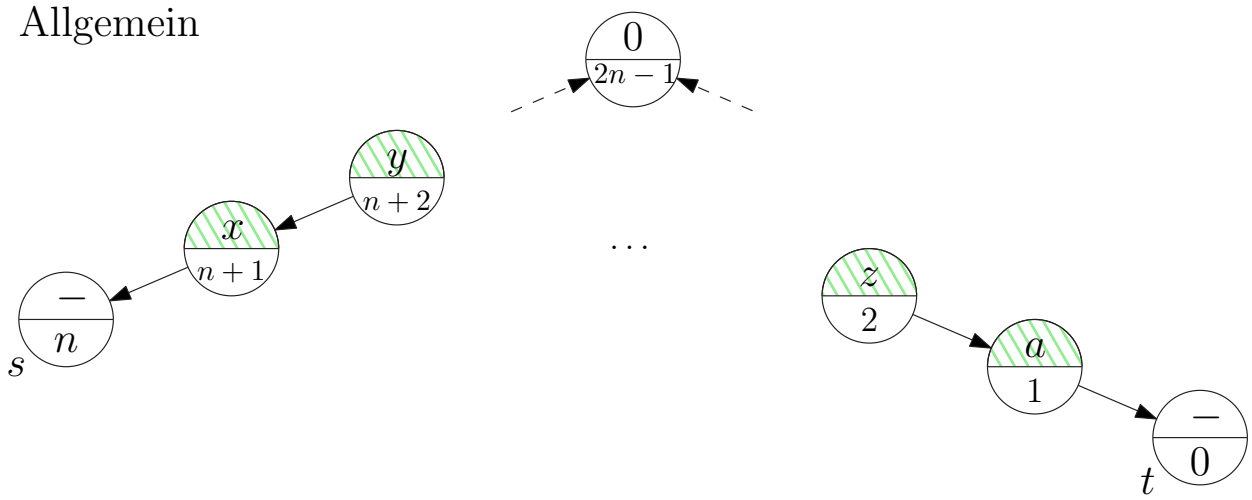
Preflow-Push

Heuristiken

- *global relabeling*
  - setze Levels auf

$$d(v) = \begin{cases} \mu(v,t) & \text{falls t von v erreichbar} \\ n + \mu(v,s) & \text{sonst, falls s von v erreichbar} \\ 2n-1 & \text{sonst} \end{cases}$$

  - Berechnung der Distanzen $\mu(v,\cdot)$ über Breitensuche in $\mathcal{O}(m)$
    nur alle $\Omega(m)$ Schritte, damit Kosten $\mathcal{O}(1)$ pro Schritt



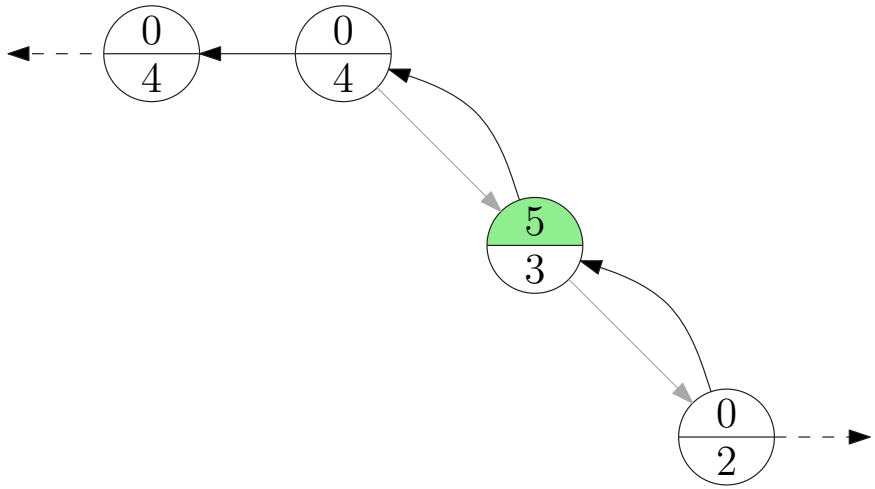
Preflow-Push

Heuristiken

- *gap heuristic*
  - wird ein Level durch `relabel(v)` leer,
    setze Level von v und aller von v erreichbaren Knoten auf

$$d(w) = \max\{d(w), n\}$$

Lücke kann nie mehr überwunden werden, Fluss nur noch zurück zu s



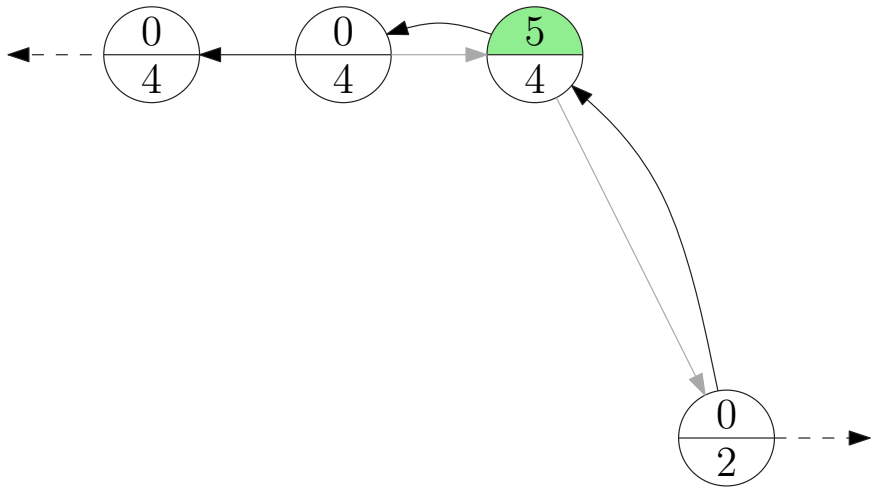
Preflow-Push

Heuristiken

- *gap heuristic*
  - wird ein Level durch `relabel(v)` leer,
    setze Level von v und aller von v erreichbaren Knoten auf

$$d(w) = \max\{d(w), n\}$$

Lücke kann nie mehr überwunden werden, Fluss nur noch zurück zu s



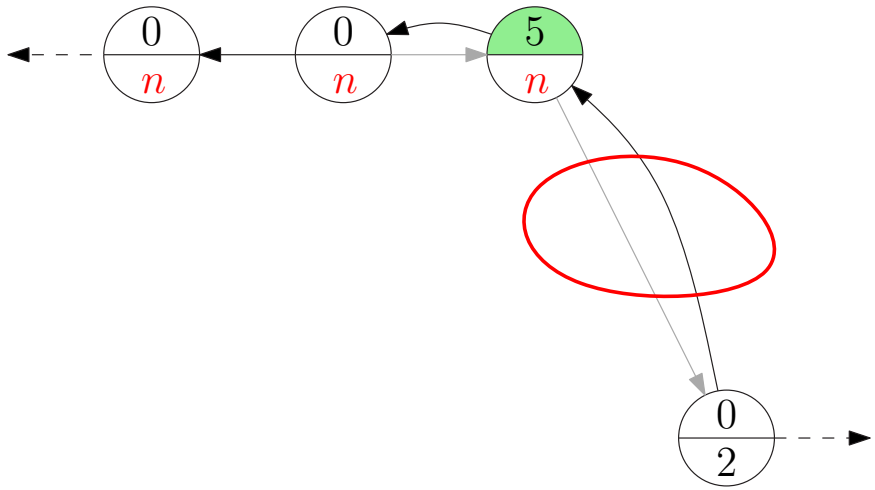
Preflow-Push

Heuristiken

- *gap heuristic*
  - wird ein Level durch `relabel(v)` leer,
    setze Level von v und aller von v erreichbaren Knoten auf

$$d(w) = \max\{d(w), n\}$$

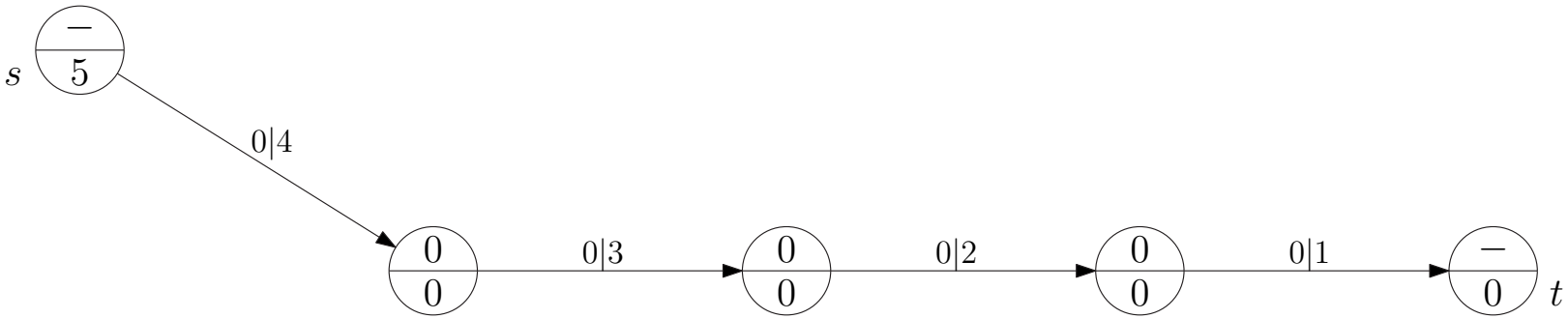
Lücke kann nie mehr überwunden werden, Fluss nur noch zurück zu s



Preflow-Push

Heuristiken

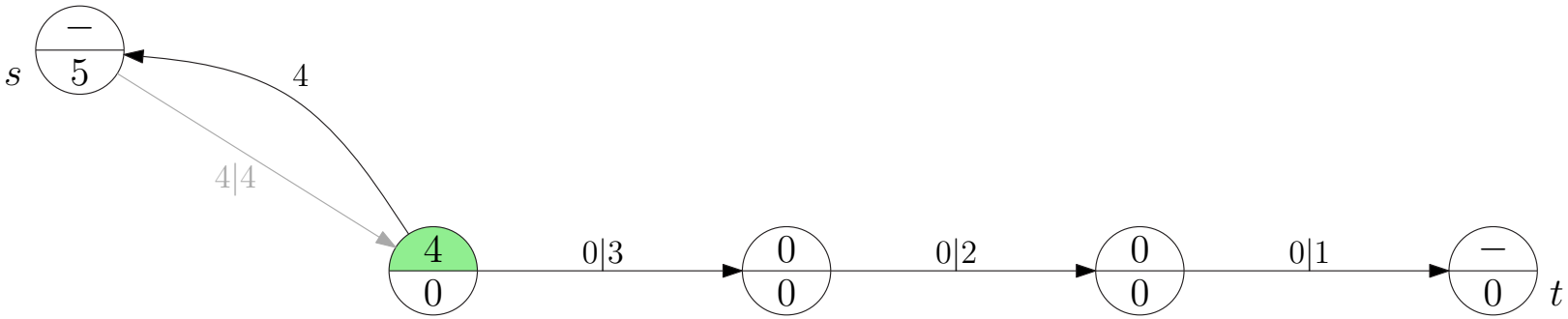
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\Rightarrow$ erzeugt *maximum preflow*
    $\Rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\Rightarrow$ Fluss nur nach s möglich



Preflow-Push

Heuristiken

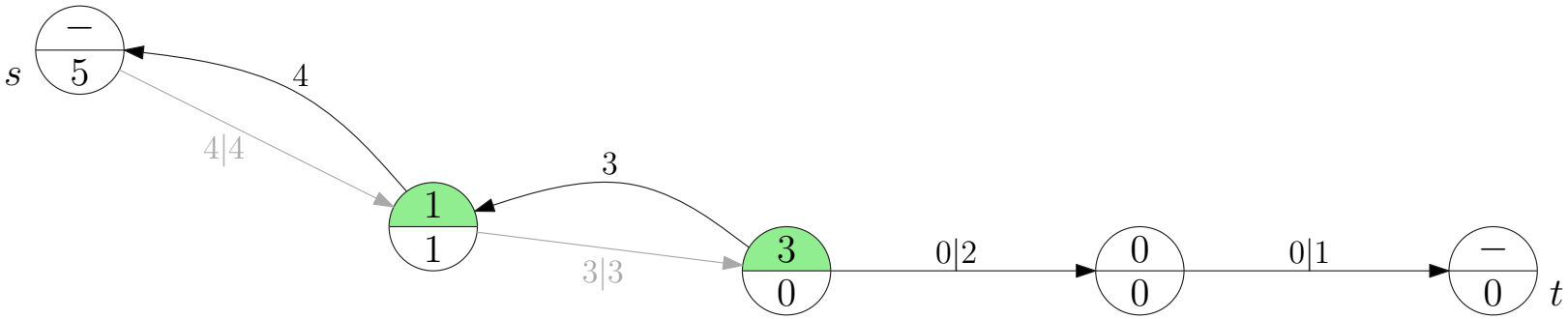
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\Rightarrow$ erzeugt *maximum preflow*
    $\Rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\Rightarrow$ Fluss nur nach s möglich



Preflow-Push

Heuristiken

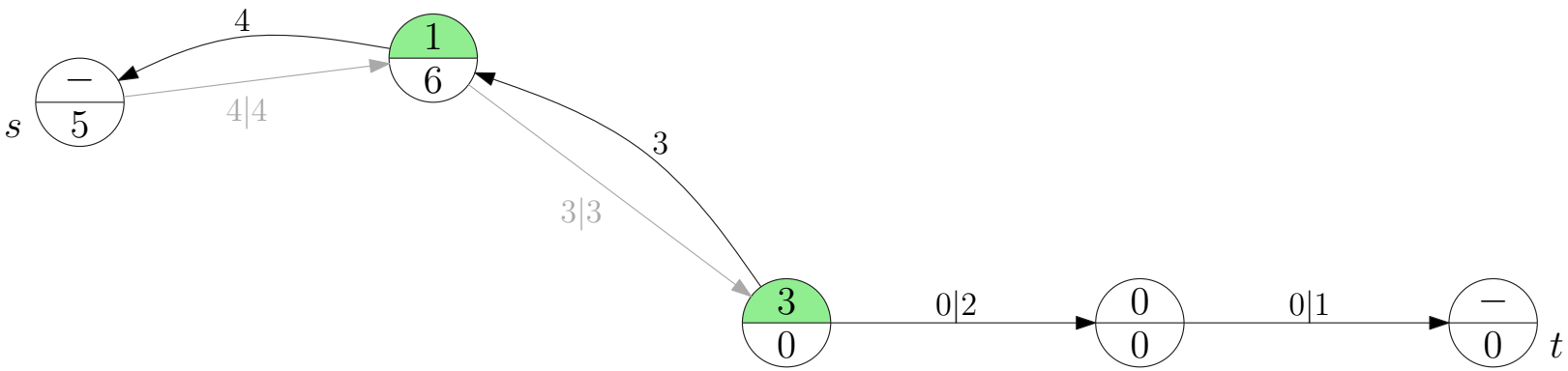
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\Rightarrow$ erzeugt *maximum preflow*
    $\Rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\Rightarrow$ Fluss nur nach s möglich



Preflow-Push

Heuristiken

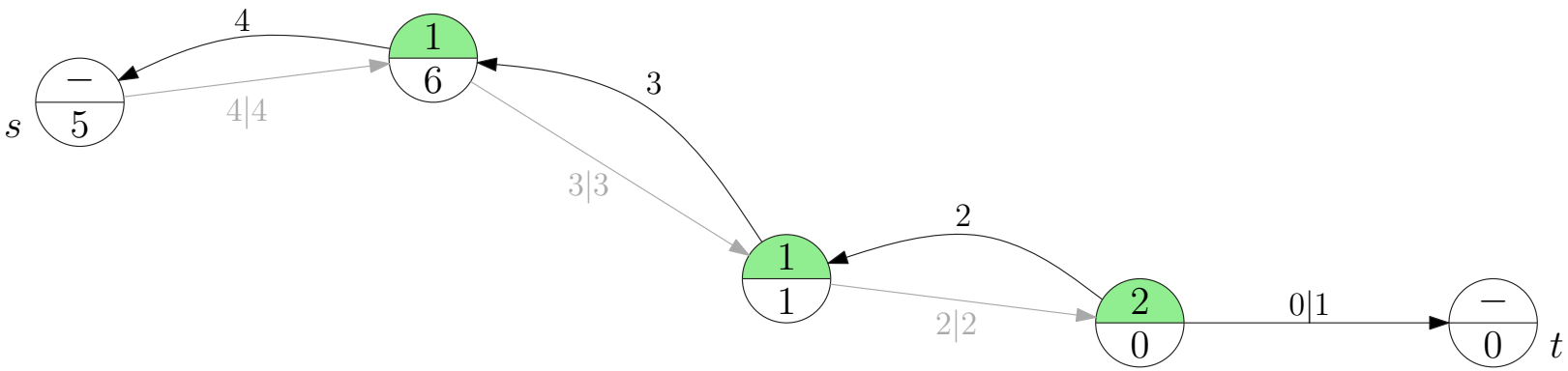
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\Rightarrow$ erzeugt *maximum preflow*
    $\Rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\Rightarrow$ Fluss nur nach s möglich



Preflow-Push

Heuristiken

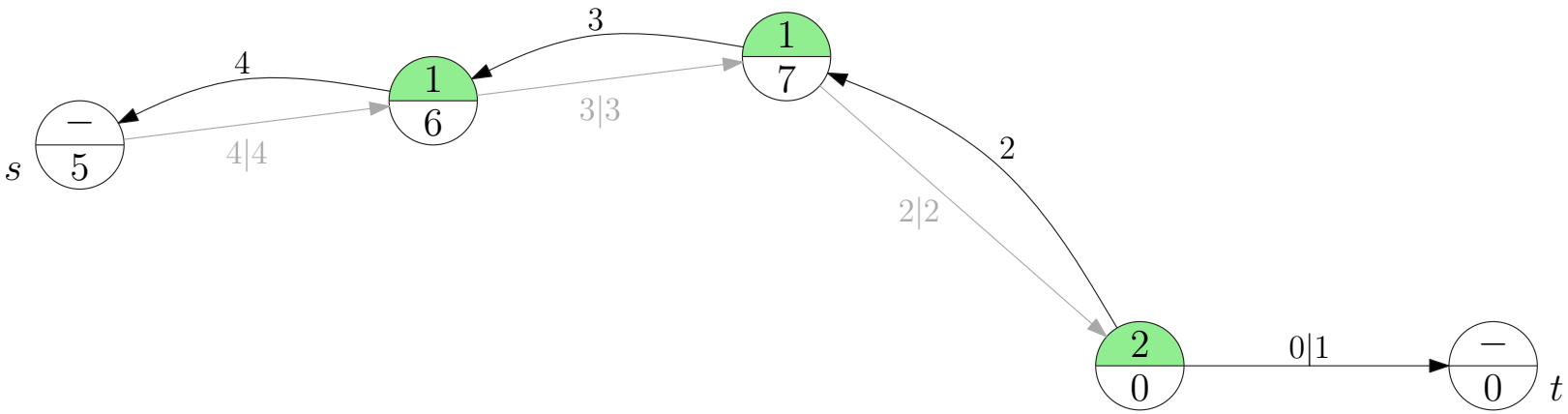
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\Rightarrow$ erzeugt *maximum preflow*
    $\Rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\Rightarrow$ Fluss nur nach s möglich



Preflow-Push

Heuristiken

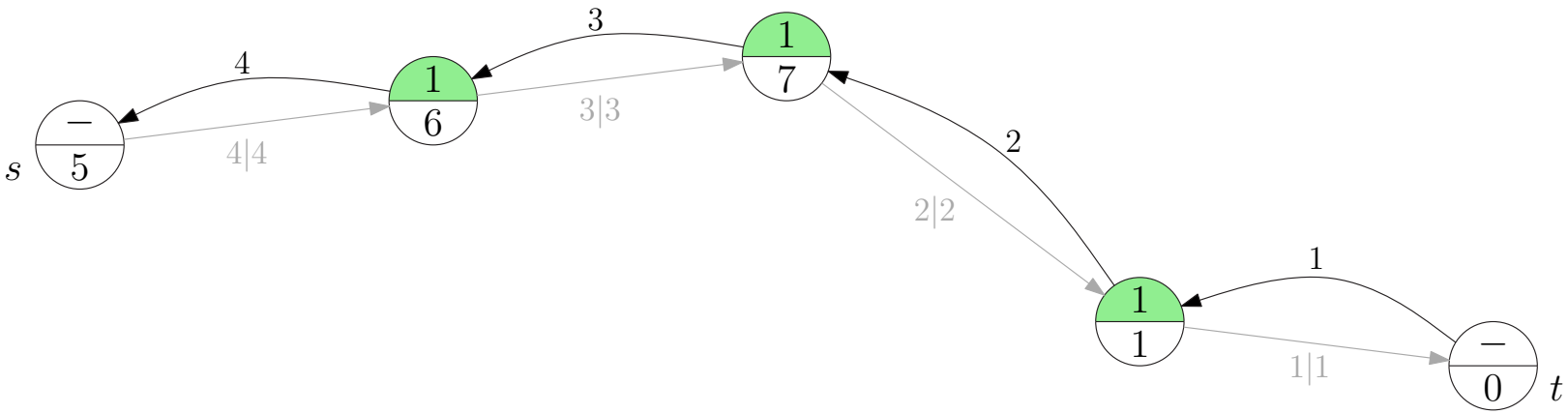
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\Rightarrow$ erzeugt *maximum preflow*
    $\Rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\Rightarrow$ Fluss nur nach s möglich



Preflow-Push

Heuristiken

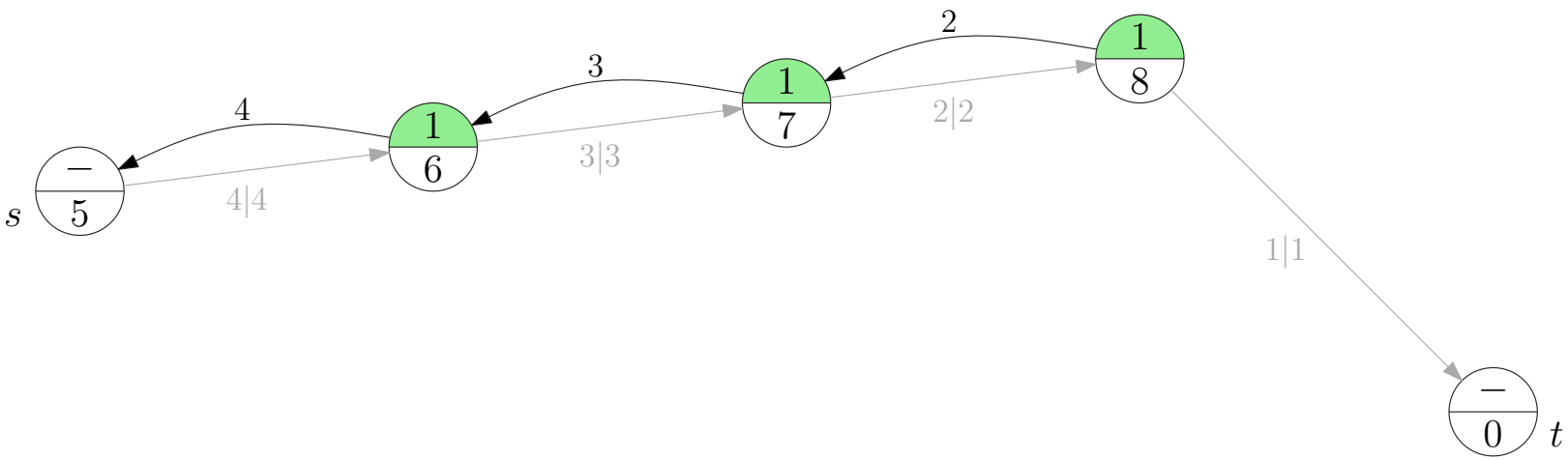
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\Rightarrow$ erzeugt *maximum preflow*
    $\Rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\Rightarrow$ Fluss nur nach s möglich



Preflow-Push

Heuristiken

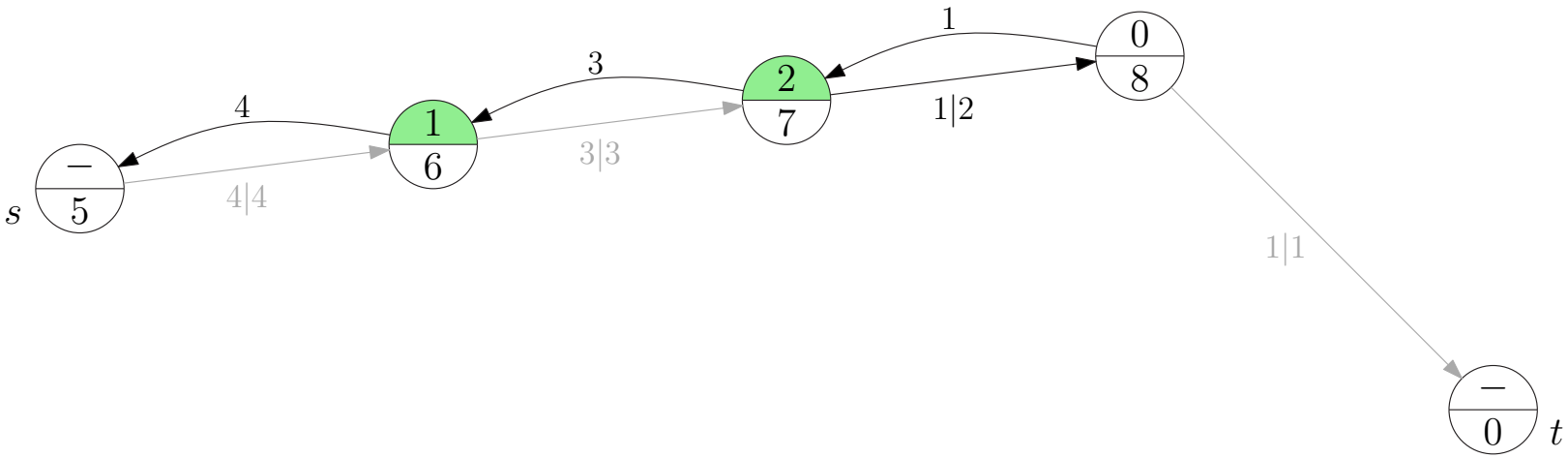
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\Rightarrow$ erzeugt *maximum preflow*
    $\Rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\Rightarrow$ Fluss nur nach s möglich



Preflow-Push

Heuristiken

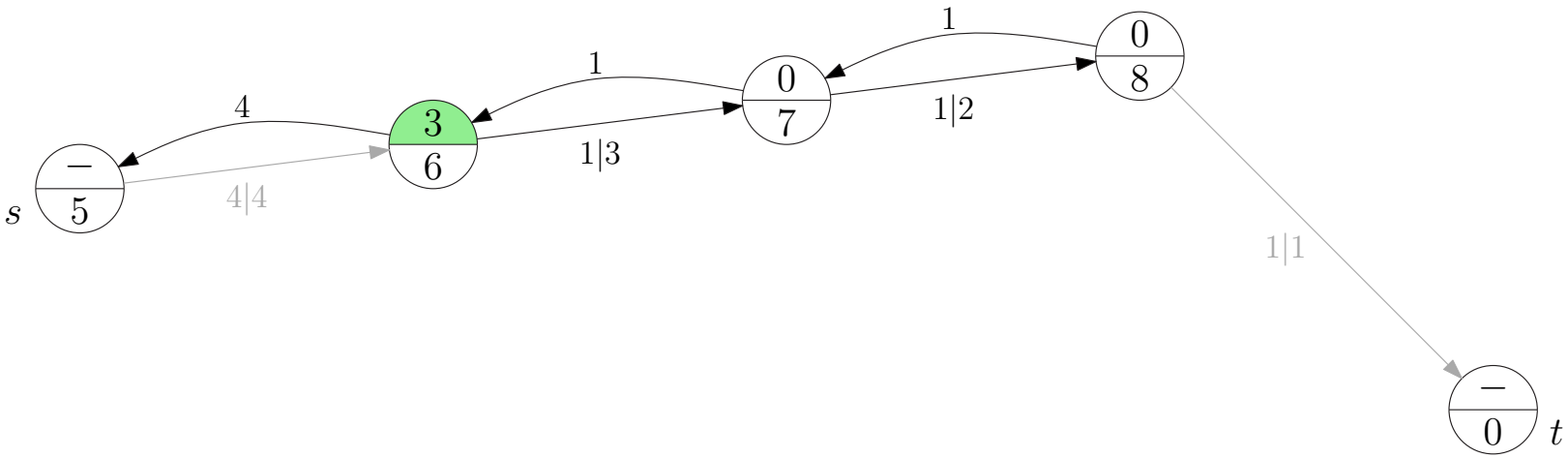
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\Rightarrow$ erzeugt *maximum preflow*
    $\Rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\Rightarrow$ Fluss nur nach s möglich



Preflow-Push

Heuristiken

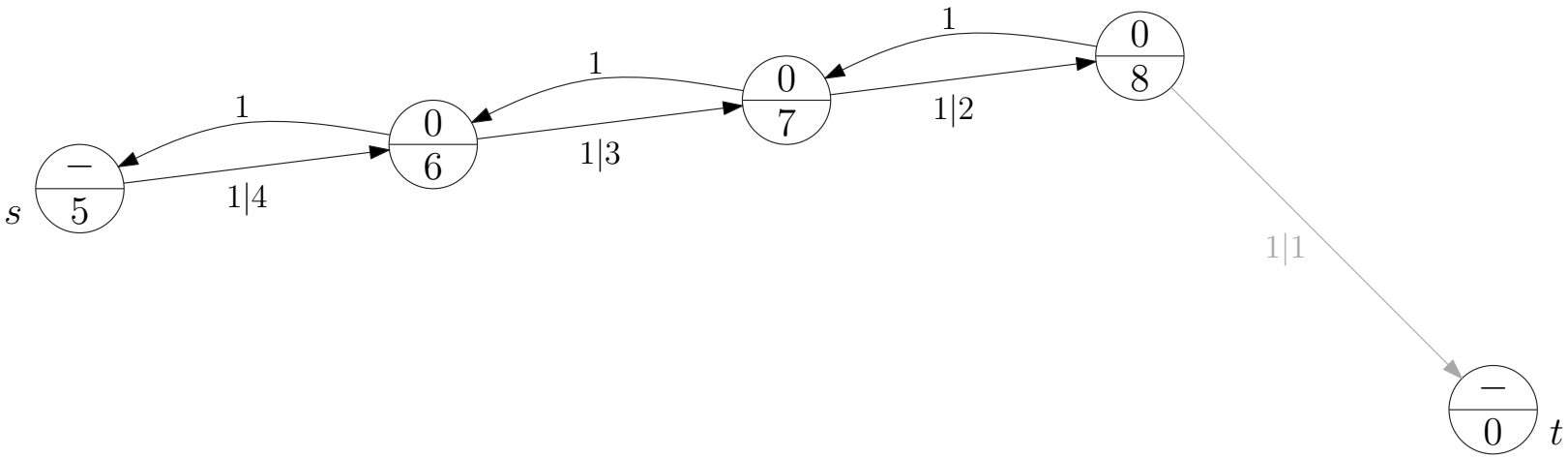
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\Rightarrow$ erzeugt *maximum preflow*
    $\Rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\Rightarrow$ Fluss nur nach s möglich



Preflow-Push

Heuristiken

- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\Rightarrow$ erzeugt *maximum preflow*
    $\Rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\Rightarrow$ Fluss nur nach s möglich



Ende!

Feierabend!