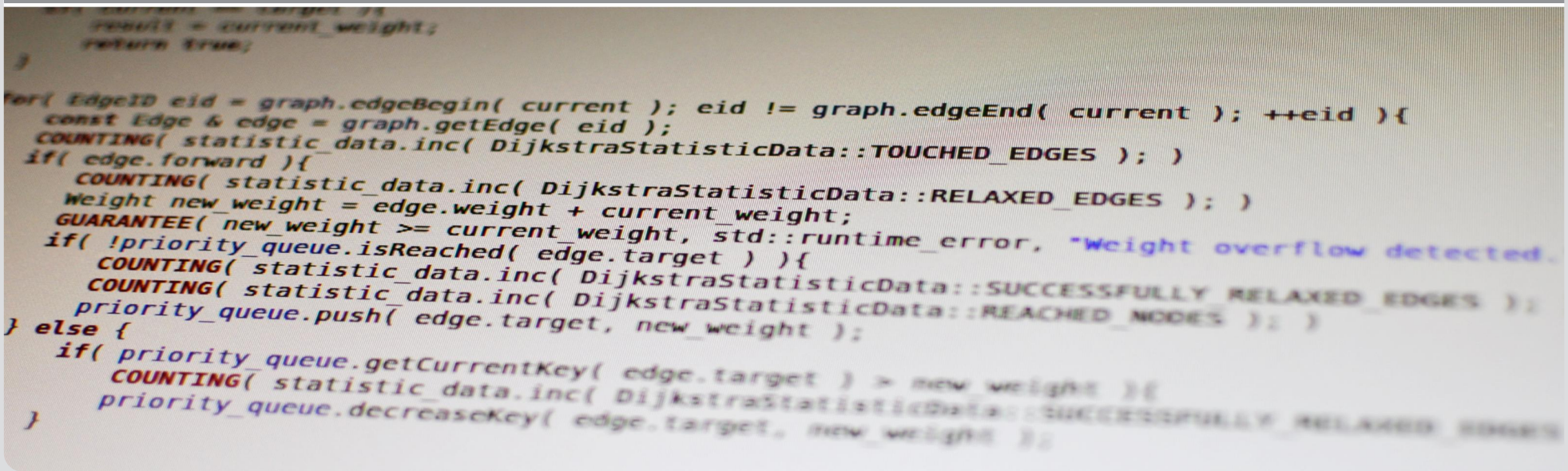
Karlsruher Institut für Technologie

Übung 5 – Algorithmen II

Tobias Heuer, Sebastian Lamm – *tobias.heuer@kit.edu, lamm@kit.edu*

`http://algo2.iti.kit.edu/AlgorithmenII_WS19.php`

Institut für Theoretische Informatik - Algorithmik II



Themenübersicht

Potentialmethode zur Algorithmenanalyse

- Stapel mit `multipop`

preflow-push Algorithmus

- Überblick
- *FIFO preflow-push*
- Heuristiken

Matching

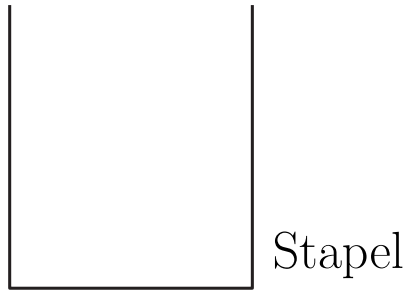
Potentialmethode

Potentialmethode

Amortisierte Analyse

- Datenstruktur Stapel mit `multipop`
  - `push(v)`: Element v oben auf Stapel legen $\mathcal{O}(1)$
  - `pop`: oberstes Element aus Stapel entfernen $\mathcal{O}(1)$
  - `multipop(k)`: oberste k Elemente aus Stapel entfernen $\mathcal{O}(n)$

  $\rightarrow$ Laufzeit von n Operationen ist im *worst-case* $\mathcal{O}(n^2)$



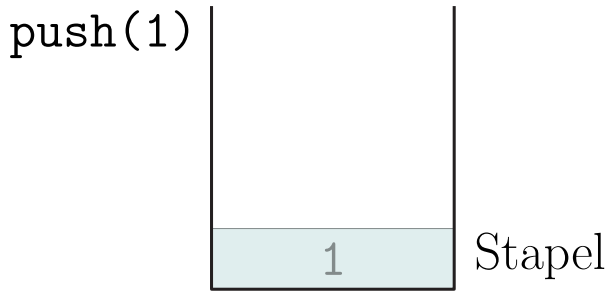
Behauptung: *Worst-case* Laufzeit für n Operationen ist amortisiert $\mathcal{O}(n)$!

Potentialmethode

Amortisierte Analyse

- Datenstruktur Stapel mit `multipop`
  - `push(v)`: Element v oben auf Stapel legen $\mathcal{O}(1)$
  - `pop`: oberstes Element aus Stapel entfernen $\mathcal{O}(1)$
  - `multipop(k)`: oberste k Elemente aus Stapel entfernen $\mathcal{O}(n)$

  $\rightarrow$ Laufzeit von n Operationen ist im *worst-case* $\mathcal{O}(n^2)$



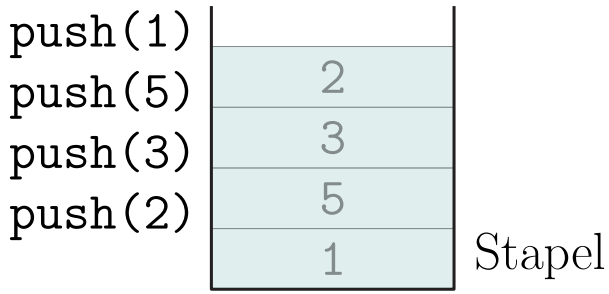
Behauptung: *Worst-case* Laufzeit für n Operationen ist amortisiert $\mathcal{O}(n)$!

Potentialmethode

Amortisierte Analyse

- Datenstruktur Stapel mit `multipop`
  - `push(v)`: Element v oben auf Stapel legen $\mathcal{O}(1)$
  - `pop`: oberstes Element aus Stapel entfernen $\mathcal{O}(1)$
  - `multipop(k)`: oberste k Elemente aus Stapel entfernen $\mathcal{O}(n)$

  $\rightarrow$ Laufzeit von n Operationen ist im *worst-case* $\mathcal{O}(n^2)$



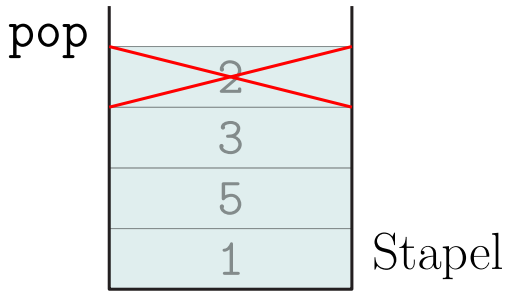
Behauptung: *Worst-case* Laufzeit für n Operationen ist amortisiert $\mathcal{O}(n)$!

Potentialmethode

Amortisierte Analyse

- Datenstruktur Stapel mit `multipop`
  - `push(v)`: Element v oben auf Stapel legen $\mathcal{O}(1)$
  - `pop`: oberstes Element aus Stapel entfernen $\mathcal{O}(1)$
  - `multipop(k)`: oberste k Elemente aus Stapel entfernen $\mathcal{O}(n)$

  $\rightarrow$ Laufzeit von n Operationen ist im *worst-case* $\mathcal{O}(n^2)$



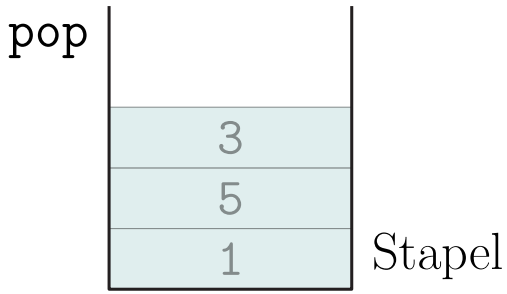
Behauptung: *Worst-case* Laufzeit für n Operationen ist amortisiert $\mathcal{O}(n)$!

Potentialmethode

Amortisierte Analyse

- Datenstruktur Stapel mit `multipop`
  - `push(v)`: Element v oben auf Stapel legen $\mathcal{O}(1)$
  - `pop`: oberstes Element aus Stapel entfernen $\mathcal{O}(1)$
  - `multipop(k)`: oberste k Elemente aus Stapel entfernen $\mathcal{O}(n)$

  $\rightarrow$ Laufzeit von n Operationen ist im *worst-case* $\mathcal{O}(n^2)$



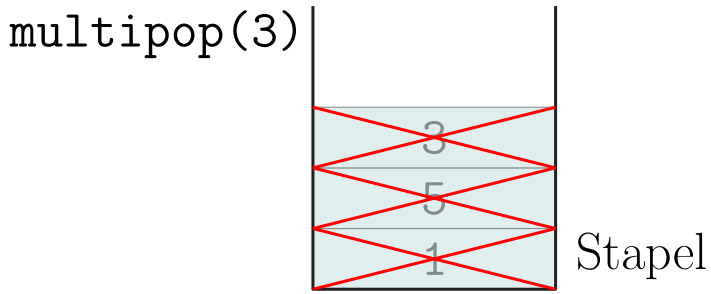
Behauptung: *Worst-case* Laufzeit für n Operationen ist amortisiert $\mathcal{O}(n)$!

Potentialmethode

Amortisierte Analyse

- Datenstruktur Stapel mit `multipop`
  - `push(v)`: Element v oben auf Stapel legen $\mathcal{O}(1)$
  - `pop`: oberstes Element aus Stapel entfernen $\mathcal{O}(1)$
  - `multipop(k)`: oberste k Elemente aus Stapel entfernen $\mathcal{O}(n)$

  $\rightarrow$ Laufzeit von n Operationen ist im *worst-case* $\mathcal{O}(n^2)$



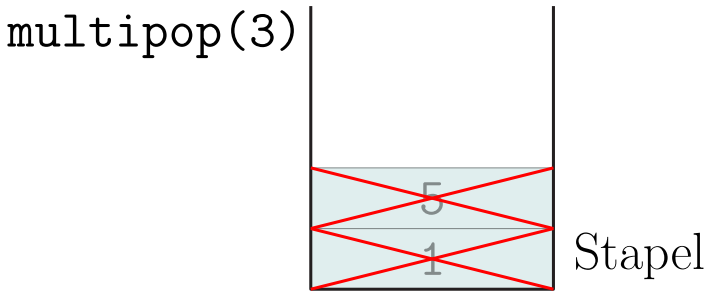
Behauptung: *Worst-case* Laufzeit für n Operationen ist amortisiert $\mathcal{O}(n)$!

Potentialmethode

Amortisierte Analyse

- Datenstruktur Stapel mit `multipop`
  - `push(v)`: Element v oben auf Stapel legen $\mathcal{O}(1)$
  - `pop`: oberstes Element aus Stapel entfernen $\mathcal{O}(1)$
  - `multipop(k)`: oberste k Elemente aus Stapel entfernen $\mathcal{O}(n)$

  $\rightarrow$ Laufzeit von n Operationen ist im *worst-case* $\mathcal{O}(n^2)$



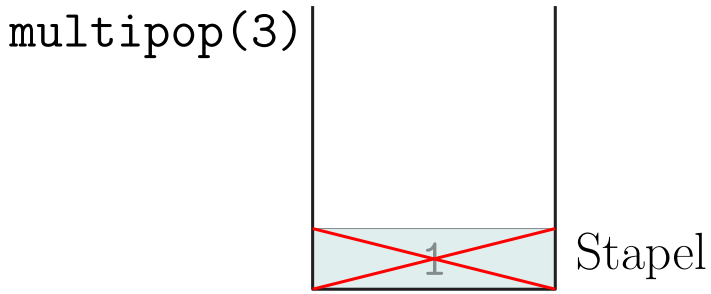
Behauptung: *Worst-case* Laufzeit für n Operationen ist amortisiert $\mathcal{O}(n)$!

Potentialmethode

Amortisierte Analyse

- Datenstruktur Stapel mit `multipop`
  - `push(v)`: Element v oben auf Stapel legen $\mathcal{O}(1)$
  - `pop`: oberstes Element aus Stapel entfernen $\mathcal{O}(1)$
  - `multipop(k)`: oberste k Elemente aus Stapel entfernen $\mathcal{O}(n)$

  $\rightarrow$ Laufzeit von n Operationen ist im *worst-case* $\mathcal{O}(n^2)$



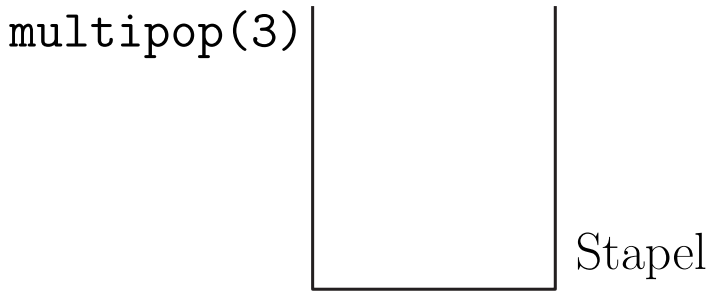
Behauptung: *Worst-case* Laufzeit für n Operationen ist amortisiert $\mathcal{O}(n)$!

Potentialmethode

Amortisierte Analyse

- Datenstruktur Stapel mit `multipop`
  - `push(v)`: Element v oben auf Stapel legen $\mathcal{O}(1)$
  - `pop`: oberstes Element aus Stapel entfernen $\mathcal{O}(1)$
  - `multipop(k)`: oberste k Elemente aus Stapel entfernen $\mathcal{O}(n)$

  $\rightarrow$ Laufzeit von n Operationen ist im *worst-case* $\mathcal{O}(n^2)$



Behauptung: *Worst-case* Laufzeit für n Operationen ist amortisiert $\mathcal{O}(n)$!

Potentialmethode

Amortisierte Analyse

- Sei $\Phi = \#_{\text{Elemente auf Stack}} \geq 0$
  - `push(v)`: $\Delta\Phi = 1$ (Erhöhung)
    `pop`: $\Delta\Phi \in \{-1, 0\}$ (Erniedrigung)
    `multipop(k)`: $\Delta\Phi \in \{-k, \ldots, 0\}$ (Erniedrigung)
  - da $\#_{push} \leq n$ (bei n Operationen)
    $\rightarrow$ maximal n Erhöhungen
    $\rightarrow$ Erniedrigungen um maximal n möglich
  - $\rightarrow$ insgesamte Änderungen um $\leq 2n$

$\rightarrow$ maximale Kosten $\mathcal{O}(n)$
(Erhöhung oder Erniedrigung um 1 kostet $\mathcal{O}(1)$)

$\rightarrow$ Was ist mit `pop` / `multipop` auf leerem Stack?

- *Worst-case* Laufzeit für n Operationen ist amortisiert $\mathcal{O}(n)$!

preflow-push Algorithmus

preflow-push Algorithmus

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 $d(v) = d(w) + 1$

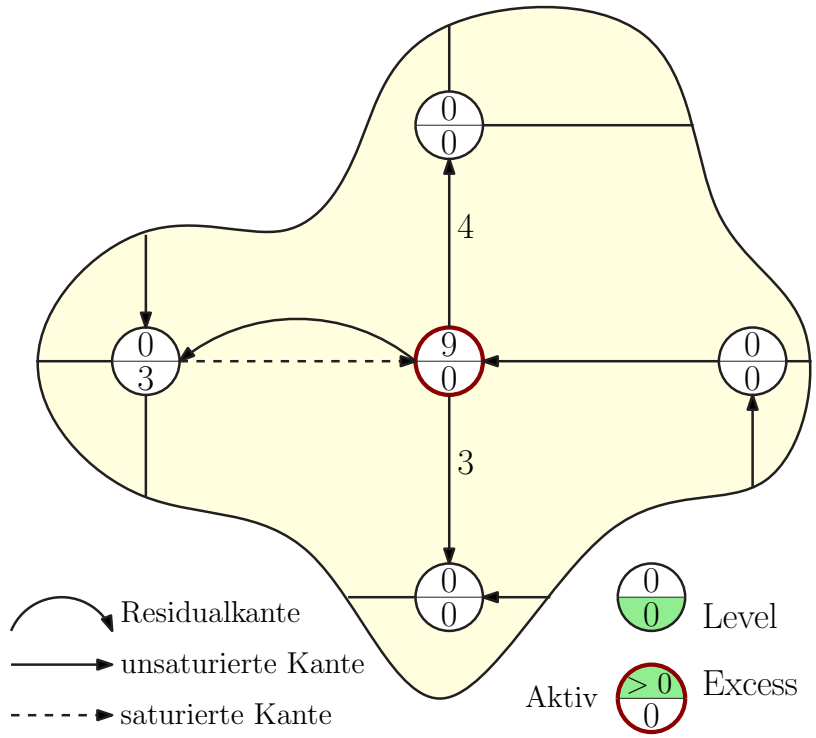
allgemeiner Ablauf

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

preflow-push Algorithmus

Wiederholung

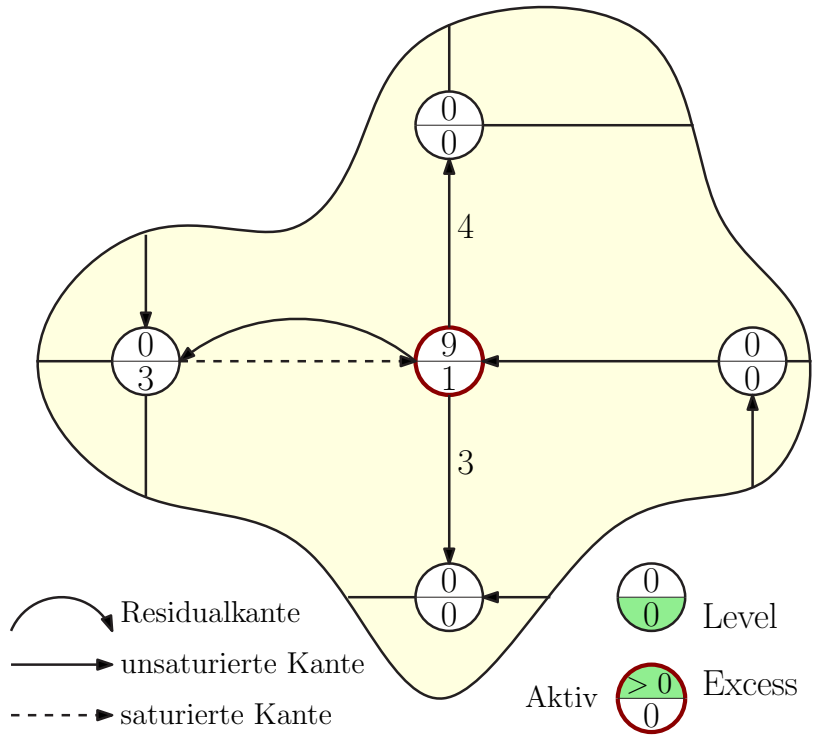
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
  (inflow > outflow)
- **relabel**
  - erhöht Level eines Knoten
  - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
  - nur zulässig wenn *push* vom Knoten nicht möglich
- push
  - schiebe Fluss entlang Kante (u, v)
  - Vorraussetzung: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



preflow-push Algorithmus

Wiederholung

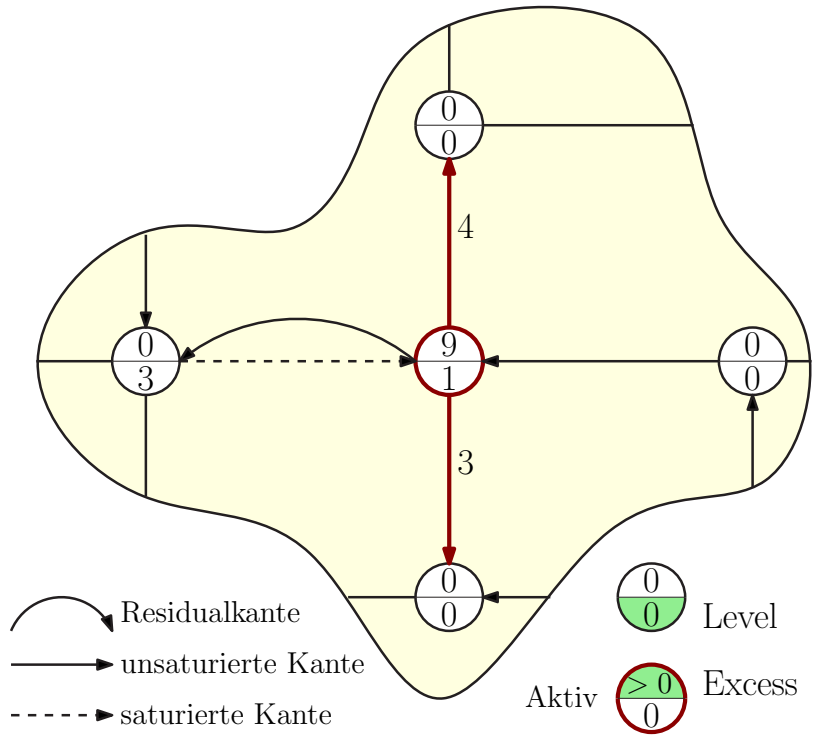
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
  (inflow > outflow)
- **relabel**
  - erhöht Level eines Knoten
  - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
  - nur zulässig wenn *push* vom Knoten nicht möglich
- push
  - schiebe Fluss entlang Kante (u, v)
  - Vorraussetzung: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



preflow-push Algorithmus

Wiederholung

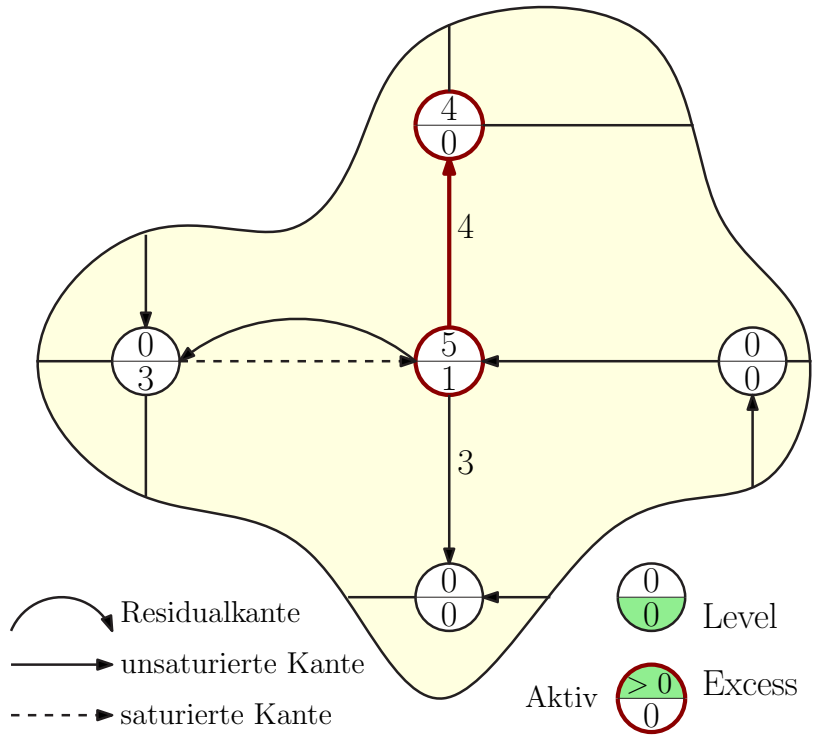
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
  (inflow > outflow)
- relabel
  - erhöht Level eines Knoten
  - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
  - nur zulässig wenn *push* vom Knoten nicht möglich
- **push**
  - schiebe Fluss entlang Kante (u, v)
  - Vorraussetzung: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



preflow-push Algorithmus

Wiederholung

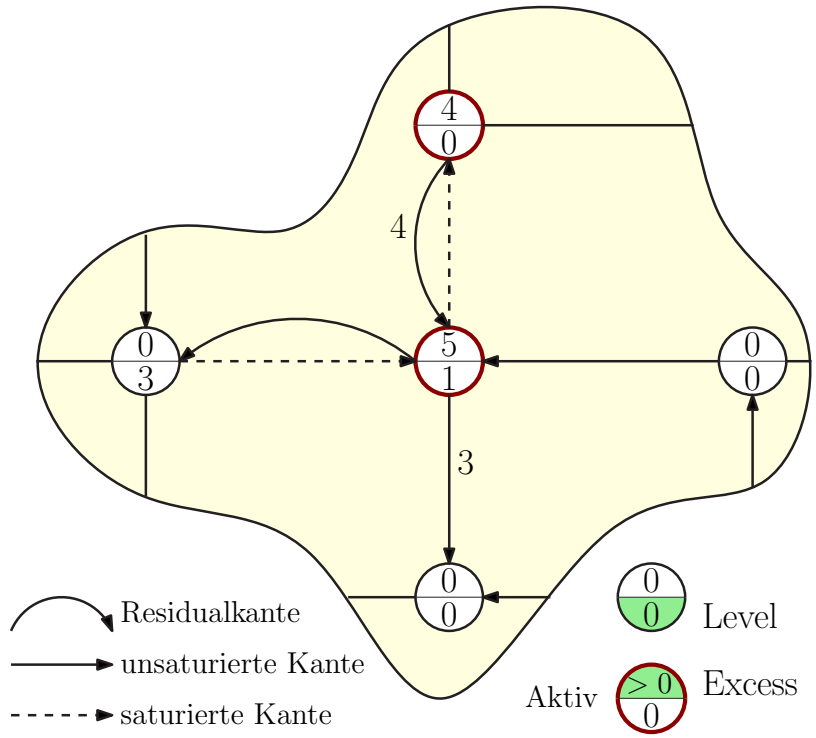
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
  (inflow > outflow)
- relabel
  - erhöht Level eines Knoten
  - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
  - nur zulässig wenn *push* vom Knoten nicht möglich
- **push**
  - schiebe Fluss entlang Kante (u, v)
  - Vorraussetzung: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



preflow-push Algorithmus

Wiederholung

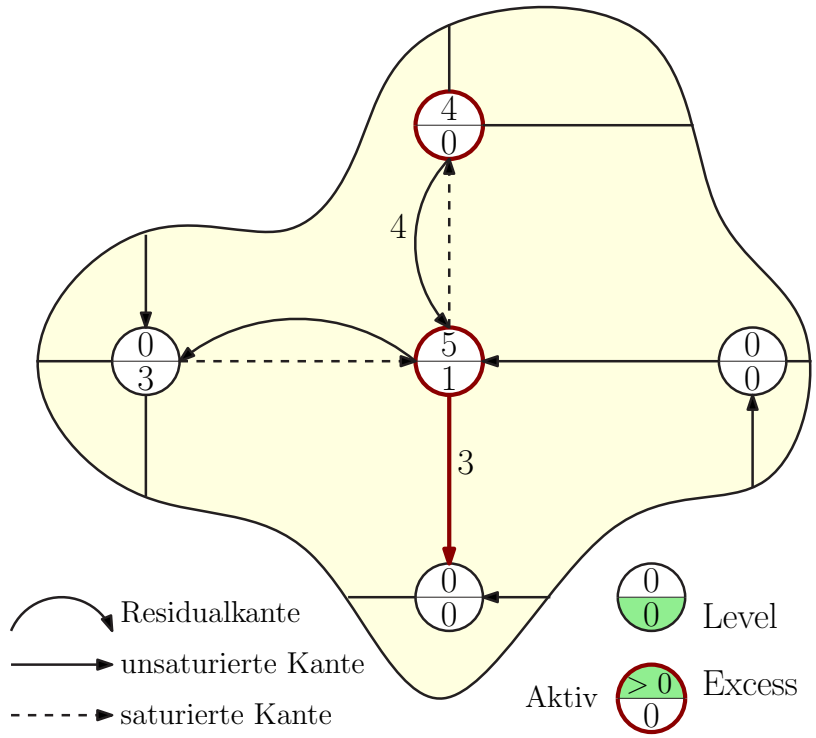
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
  (inflow > outflow)
- relabel
  - erhöht Level eines Knoten
  - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
  - nur zulässig wenn *push* vom Knoten nicht möglich
- **push**
  - schiebe Fluss entlang Kante (u, v)
  - Vorraussetzung: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



preflow-push Algorithmus

Wiederholung

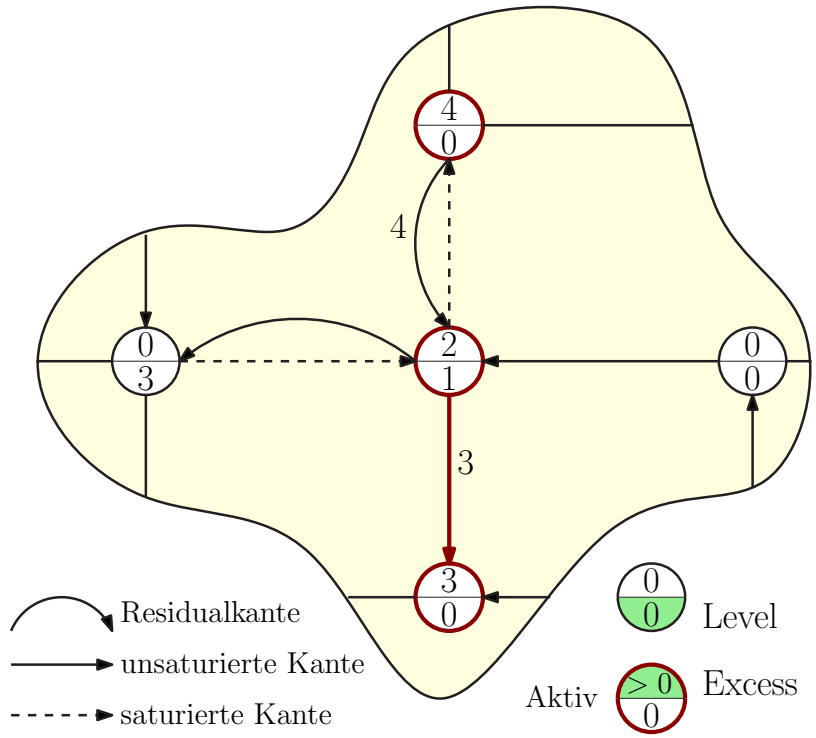
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
  (inflow > outflow)
- relabel
  - erhöht Level eines Knoten
  - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
  - nur zulässig wenn *push* vom Knoten nicht möglich
- **push**
  - schiebe Fluss entlang Kante (u, v)
  - Vorraussetzung: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



preflow-push Algorithmus

Wiederholung

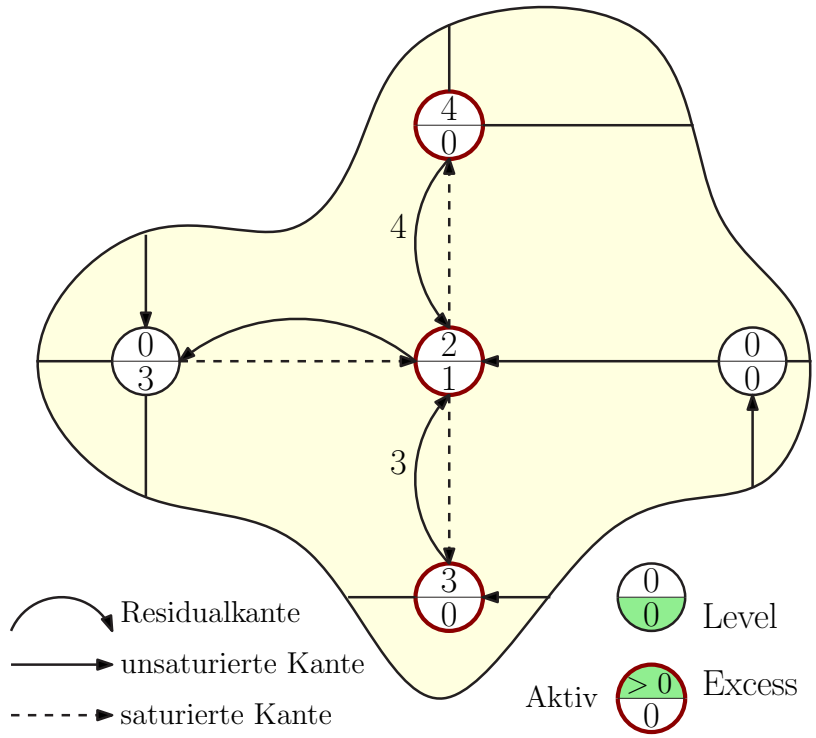
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
  (inflow > outflow)
- relabel
  - erhöht Level eines Knoten
  - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
  - nur zulässig wenn *push* vom Knoten nicht möglich
- **push**
  - schiebe Fluss entlang Kante (u, v)
  - Vorraussetzung: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



preflow-push Algorithmus

Wiederholung

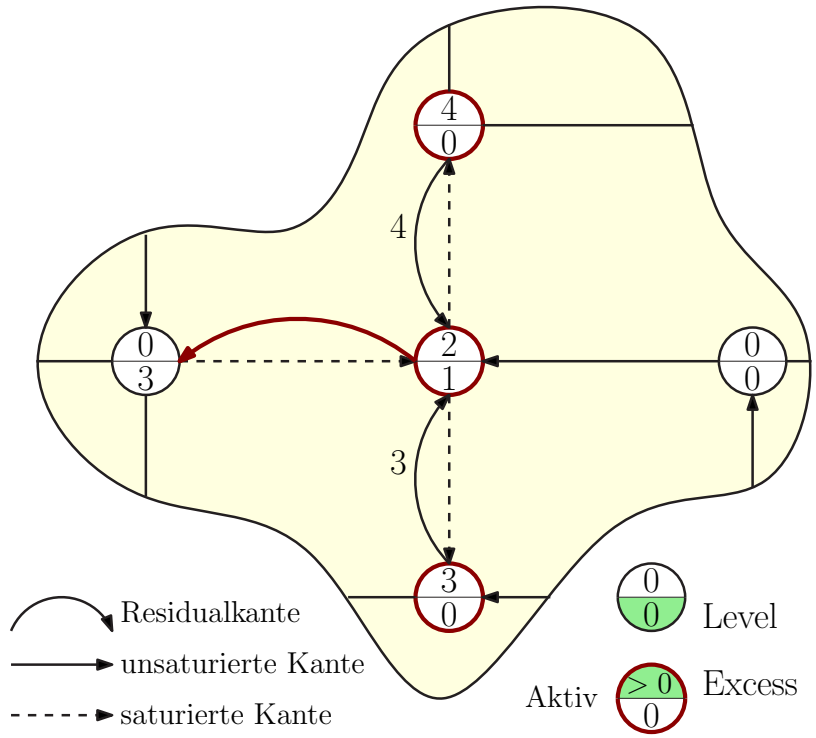
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
  (inflow > outflow)
- relabel
  - erhöht Level eines Knoten
  - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
  - nur zulässig wenn *push* vom Knoten nicht möglich
- **push**
  - schiebe Fluss entlang Kante (u, v)
  - Vorraussetzung: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



preflow-push Algorithmus

Wiederholung

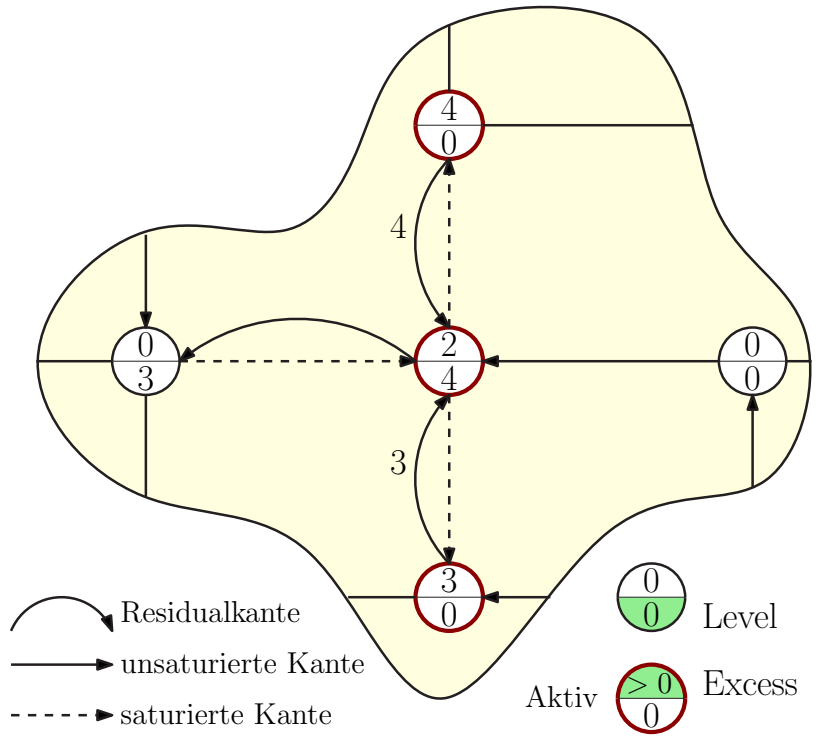
- Operationen nur auf aktiven Knoten
- aktive Knoten haben excess
  (inflow > outflow)
- **relabel**
  - erhöht Level eines Knoten
  - erhält $d(u) \leq d(v) + 1$ für alle (u, v) im Residualgraph
  - nur zulässig wenn *push* vom Knoten nicht möglich
- push
  - schiebe Fluss entlang Kante (u, v)
  - Vorraussetzung: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



preflow-push Algorithmus

Wiederholung

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



preflow-push Algorithmus

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 Algorithmus

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 Algorithmus

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 Algorithmus

Übersicht

Unterschiedliche Auswahl des aktiven Knoten:

- *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 Praxis deutliche Beschleunigung!

FIFO preflow-push Algorithmus

Überblick

Unterschiede zu *generic preflow-push*

- `push` aus aktivem Knoten bis `relabel` oder *excess* abgebaut
  - typisches Vorgehen bei Ausführung per Hand
- Verwalte aktive Knoten in FIFO Liste
  - füge Knoten nach `relabel` bzw. aktiv gewordene Knoten hinten ein

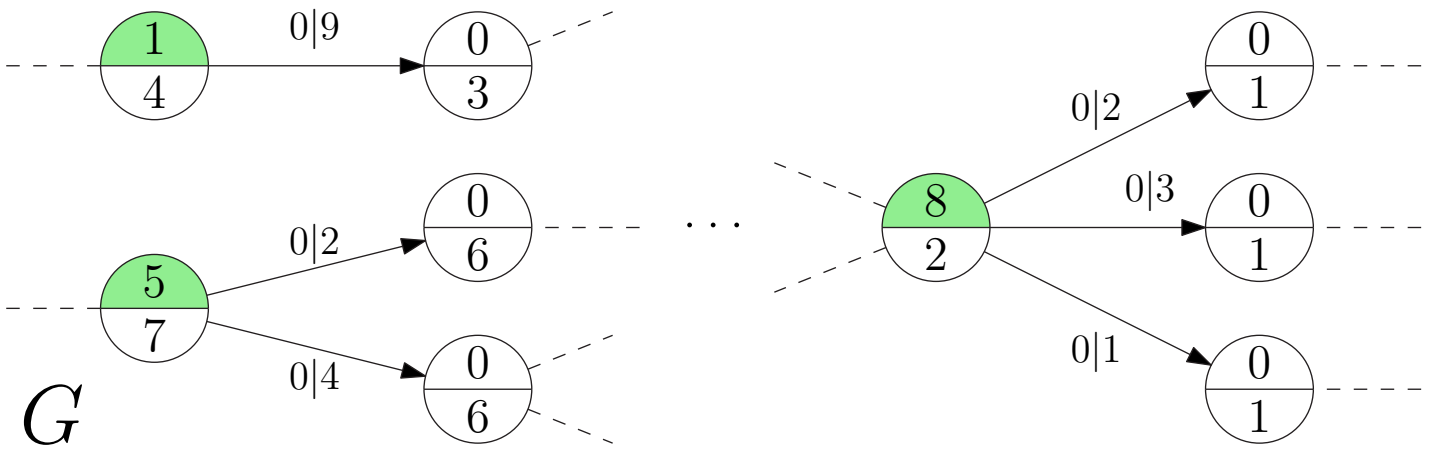
Theorem: FIFO preflow-push findet in $\mathcal{O}(n^3)$ einen maximum Fluss

FIFO preflow-push Algorithmus

Beweis Laufzeit

- Unterteile Ablauf in Phasen:
  - alle zu Phasenbeginn aktiven Knoten werden genau einmal betrachtet
    $\rightarrow$ pro Phase baut jeder Knoten max. 1x allen *excess* ab
    $\rightarrow$ pro Phase macht jeder Knoten max. 1x einen *non-saturating* `push`

$\rightarrow$ $\#_{\text{non-saturating}} \leq n \cdot \#_{\text{phases}}$

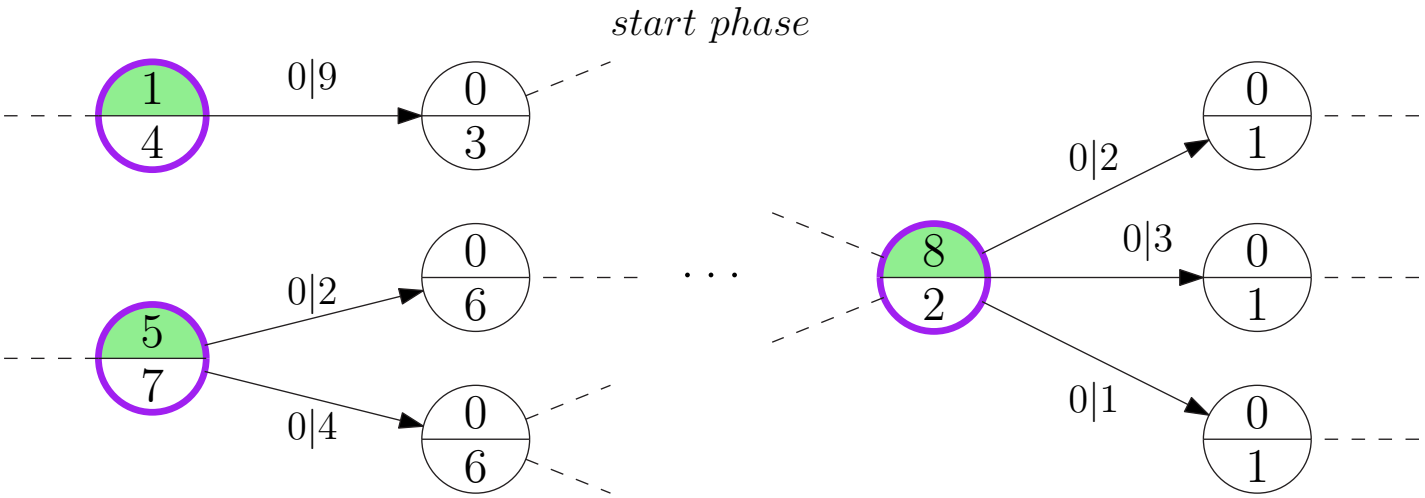


FIFO preflow-push Algorithmus

Beweis Laufzeit

- Unterteile Ablauf in Phasen:
  - alle zu Phasenbeginn aktiven Knoten werden genau einmal betrachtet
    $\rightarrow$ pro Phase baut jeder Knoten max. 1x allen *excess* ab
    $\rightarrow$ pro Phase macht jeder Knoten max. 1x einen *non-saturating* `push`

$\rightarrow$ $\#_{\text{non-saturating}} \leq n \cdot \#_{\text{phases}}$

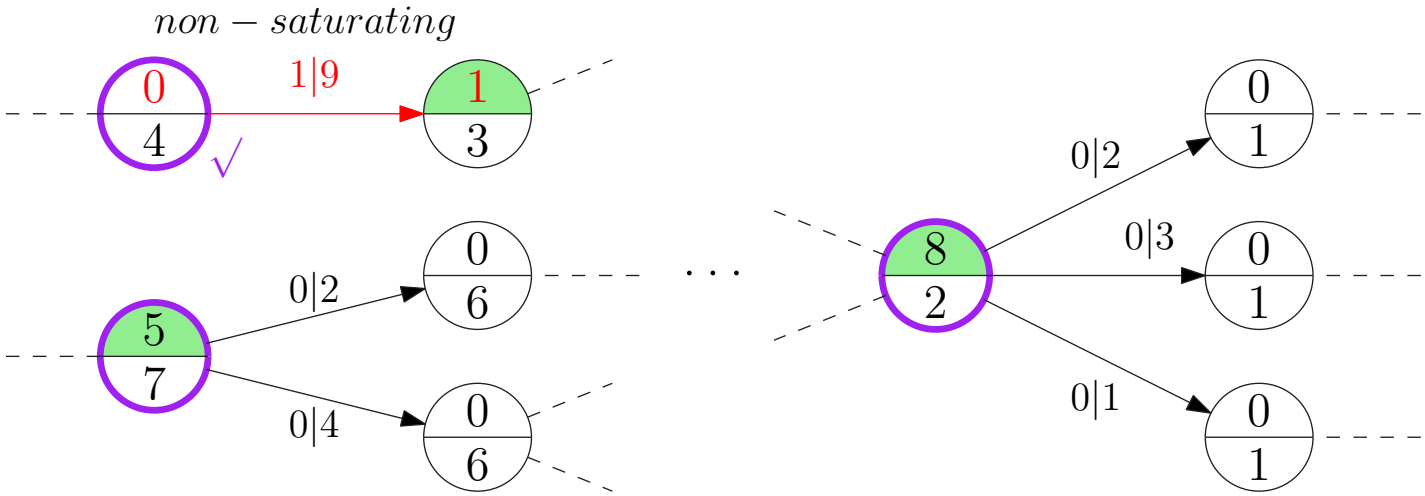


FIFO preflow-push Algorithmus

Beweis Laufzeit

- Unterteile Ablauf in Phasen:
  - alle zu Phasenbeginn aktiven Knoten werden genau einmal betrachtet
    $\rightarrow$ pro Phase baut jeder Knoten max. 1x allen *excess* ab
    $\rightarrow$ pro Phase macht jeder Knoten max. 1x einen *non-saturating* `push`

$\rightarrow$ $\#_{\text{non-saturating}} \leq n \cdot \#_{\text{phases}}$

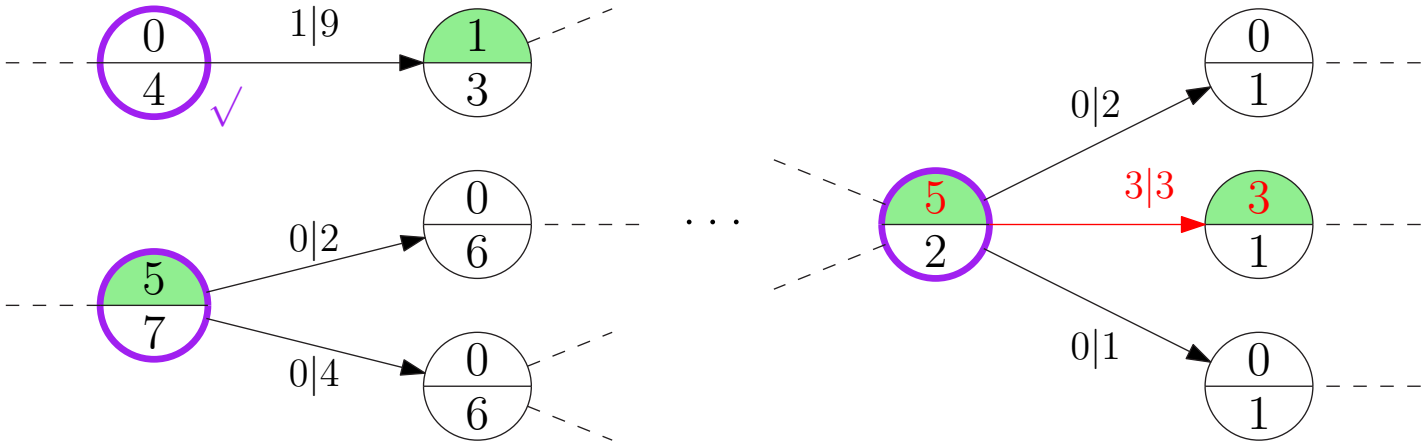


FIFO preflow-push Algorithmus

Beweis Laufzeit

- Unterteile Ablauf in Phasen:
  - alle zu Phasenbeginn aktiven Knoten werden genau einmal betrachtet
    $\rightarrow$ pro Phase baut jeder Knoten max. 1x allen *excess* ab
    $\rightarrow$ pro Phase macht jeder Knoten max. 1x einen *non-saturating* `push`

$\rightarrow$ $\#_{\text{non-saturating}} \leq n \cdot \#_{\text{phases}}$

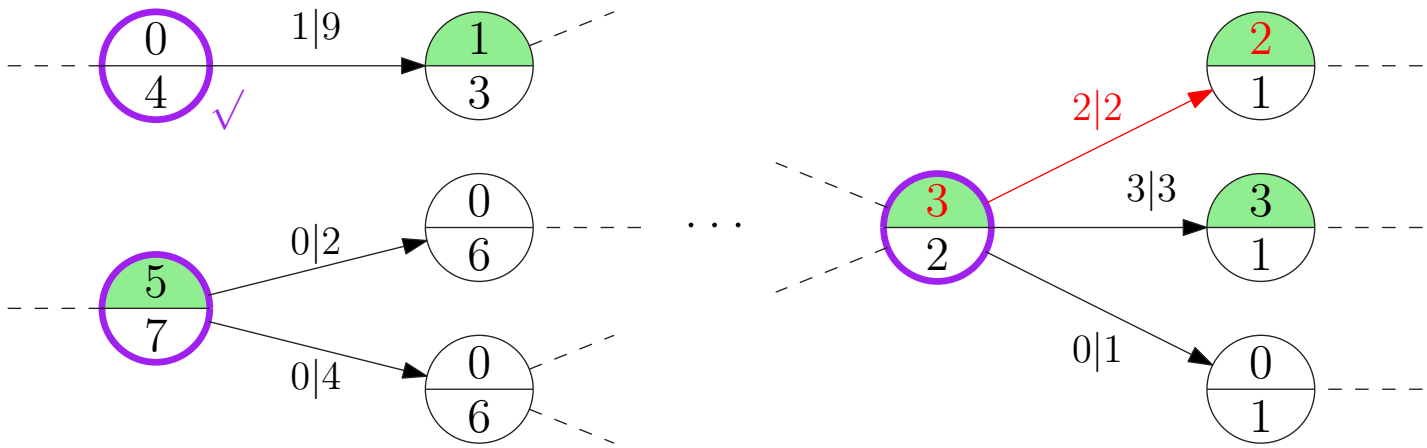


FIFO preflow-push Algorithmus

Beweis Laufzeit

- Unterteile Ablauf in Phasen:
  - alle zu Phasenbeginn aktiven Knoten werden genau einmal betrachtet
    $\rightarrow$ pro Phase baut jeder Knoten max. 1x allen *excess* ab
    $\rightarrow$ pro Phase macht jeder Knoten max. 1x einen *non-saturating* `push`

$\rightarrow$ $\#_{\text{non-saturating}} \leq n \cdot \#_{\text{phases}}$

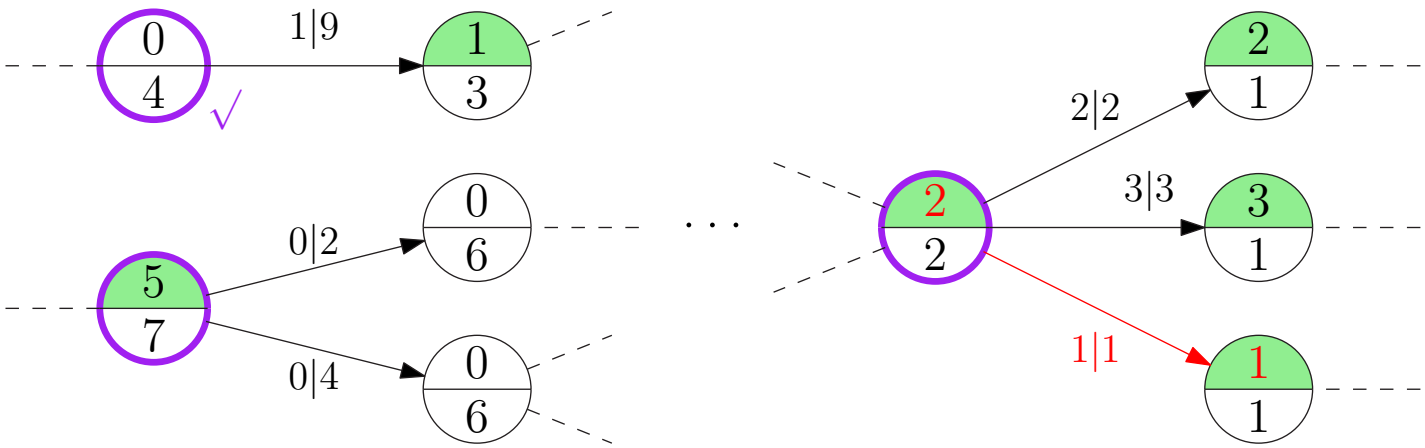


FIFO preflow-push Algorithmus

Beweis Laufzeit

- Unterteile Ablauf in Phasen:
  - alle zu Phasenbeginn aktiven Knoten werden genau einmal betrachtet
    $\rightarrow$ pro Phase baut jeder Knoten max. 1x allen *excess* ab
    $\rightarrow$ pro Phase macht jeder Knoten max. 1x einen *non-saturating* `push`

$\rightarrow$ $\#_{\text{non-saturating}} \leq n \cdot \#_{\text{phases}}$

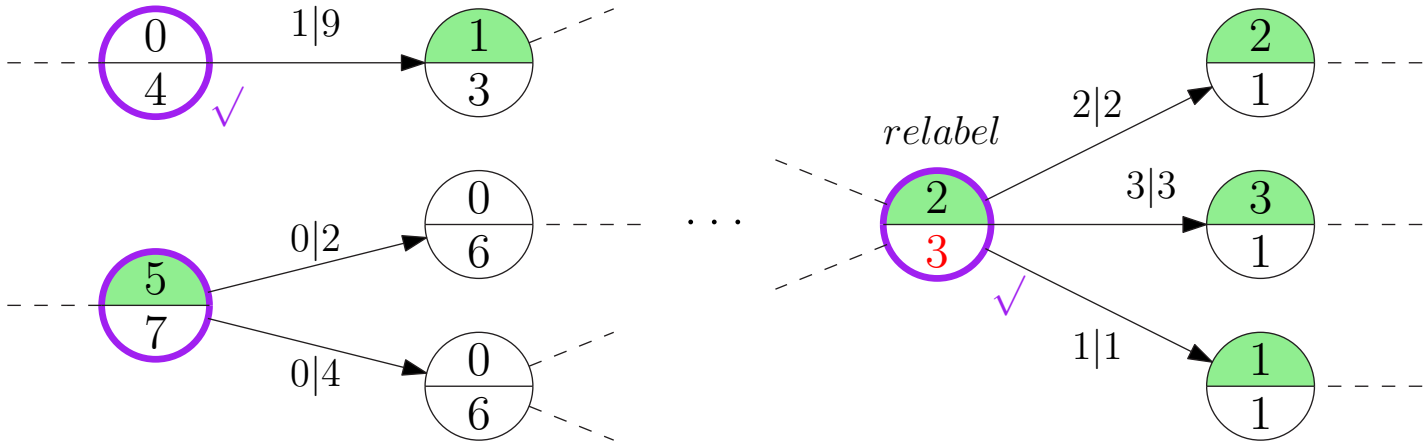


FIFO preflow-push Algorithmus

Beweis Laufzeit

- Unterteile Ablauf in Phasen:
  - alle zu Phasenbeginn aktiven Knoten werden genau einmal betrachtet
    $\rightarrow$ pro Phase baut jeder Knoten max. 1x allen *excess* ab
    $\rightarrow$ pro Phase macht jeder Knoten max. 1x einen *non-saturating* `push`

$\rightarrow$ $\#_{\text{non-saturating}} \leq n \cdot \#_{\text{phases}}$

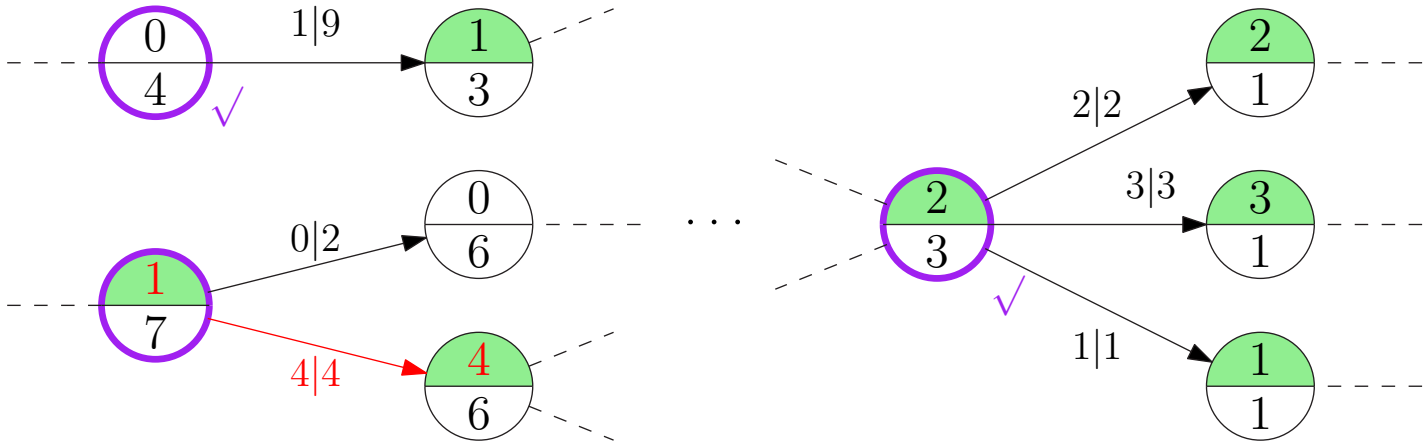


FIFO preflow-push Algorithmus

Beweis Laufzeit

- Unterteile Ablauf in Phasen:
  - alle zu Phasenbeginn aktiven Knoten werden genau einmal betrachtet
    $\rightarrow$ pro Phase baut jeder Knoten max. 1x allen *excess* ab
    $\rightarrow$ pro Phase macht jeder Knoten max. 1x einen *non-saturating* `push`

$\rightarrow$ $\#_{\text{non-saturating}} \leq n \cdot \#_{\text{phases}}$

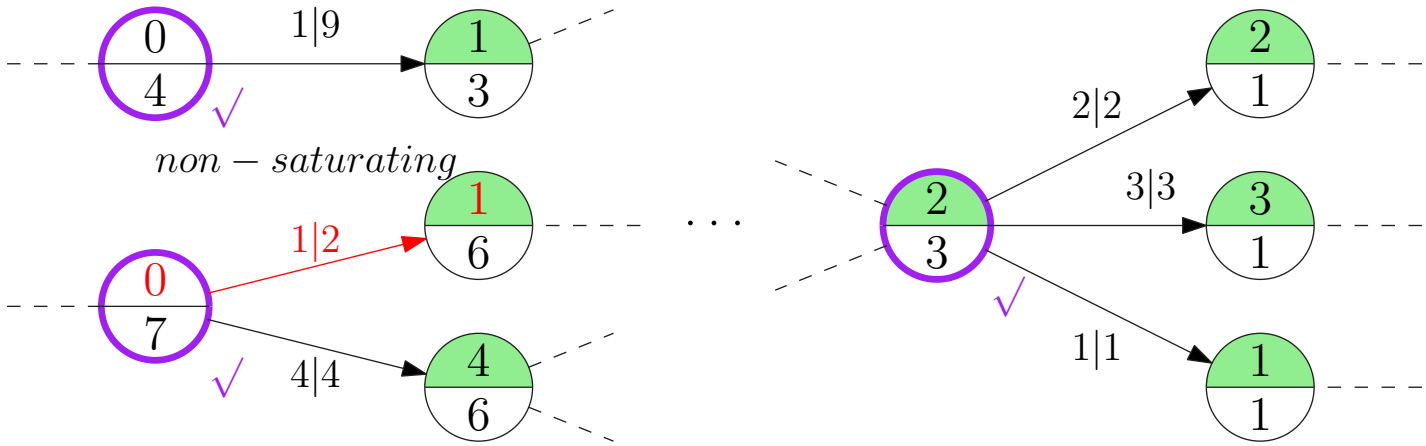


FIFO preflow-push Algorithmus

Beweis Laufzeit

- Unterteile Ablauf in Phasen:
  - alle zu Phasenbeginn aktiven Knoten werden genau einmal betrachtet
    $\rightarrow$ pro Phase baut jeder Knoten max. 1x allen *excess* ab
    $\rightarrow$ pro Phase macht jeder Knoten max. 1x einen *non-saturating* `push`

$\rightarrow$ $\#_{\text{non-saturating}} \leq n \cdot \#_{\text{phases}}$

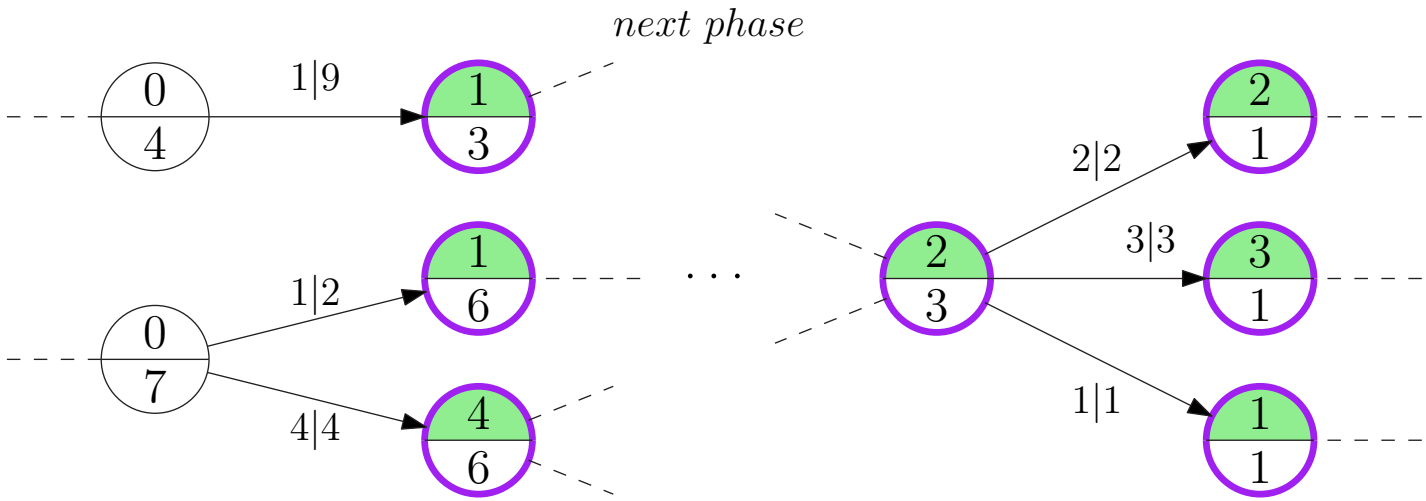


FIFO preflow-push Algorithmus

Beweis Laufzeit

- Unterteile Ablauf in Phasen:
  - alle zu Phasenbeginn aktiven Knoten werden genau einmal betrachtet
    → pro Phase baut jeder Knoten max. 1x allen *excess* ab
    → pro Phase macht jeder Knoten max. 1x einen *non-saturating* `push`

→ $\#_{\text{non-saturating}} \leq n \cdot \#_{\text{phases}}$



FIFO preflow-push Algorithmus

Beweis Laufzeit

- Sei $\Phi = \max\{d(v) \mid v \in G^f \text{ aktiv}\} \geq 0$
  (Potential, dass gleich dem höchsten aktiven Level ist)
  - Phase mit `relabel`: $\Delta\Phi \leq 1$ (Erhöhung)
    Phase ohne `relabel`: $\Delta\Phi \leq -1$ (Erniedrigung)
  - da $\#_{\text{relabel}} \leq 2n^2$ (siehe Vorlesung)
    $\rightarrow$ maximal $2n^2$ Erhöhungen
    $\rightarrow$ maximal $2n^2$ Erniedrigungen

  $\rightarrow$ $\#_{\text{phases}} \leq 4n^2$

$\rightarrow$ $\#_{\text{non-saturating}} \leq n \cdot \#_{\text{phases}} \leq n \cdot 4n^2 = 4n^3$

- restlicher Beweis wie bei *generic preflow-push*

FIFO preflow-push Algorithmus

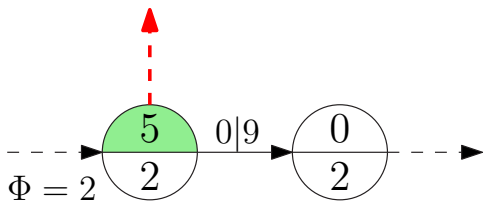
Beweis Laufzeit

- Sei $\Phi = \max\{d(v) \mid v \in G^f \text{ aktiv}\} \geq 0$
  (Potential, dass gleich dem höchsten aktiven Level ist)
  - Phase mit `relabel`: $\Delta\Phi \leq 1$ (Erhöhung)
    Phase ohne `relabel`: $\Delta\Phi \leq -1$ (Erniedrigung)
  - da $\#_{\text{relabel}} \leq 2n^2$ (siehe Vorlesung)
    $\rightarrow$ maximal $2n^2$ Erhöhungen
    $\rightarrow$ maximal $2n^2$ Erniedrigungen

  $\rightarrow$ $\#_{\text{phases}} \leq 4n^2$

$\rightarrow$ $\#_{\text{non-saturating}} \leq n \cdot \#_{\text{phases}} \leq n \cdot 4n^2 = 4n^3$

- restlicher Beweis wie bei *generic preflow-push*



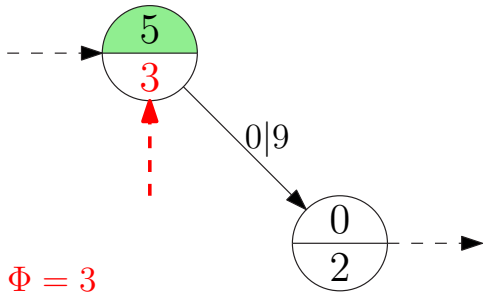
FIFO preflow-push Algorithmus

Beweis Laufzeit

- Sei $\Phi = \max\{d(v) \mid v \in G^f \text{ aktiv}\} \geq 0$
  (Potential, dass gleich dem höchsten aktiven Level ist)
  - Phase mit `relabel`: $\Delta\Phi \leq 1$ (Erhöhung)
    Phase ohne `relabel`: $\Delta\Phi \leq -1$ (Erniedrigung)
  - da $\#_{\text{relabel}} \leq 2n^2$ (siehe Vorlesung)
    $\rightarrow$ maximal $2n^2$ Erhöhungen
    $\rightarrow$ maximal $2n^2$ Erniedrigungen
  - $\rightarrow$ $\#_{\text{phases}} \leq 4n^2$

$\rightarrow$ $\#_{\text{non-saturating}} \leq n \cdot \#_{\text{phases}} \leq n \cdot 4n^2 = 4n^3$

- restlicher Beweis wie bei *generic preflow-push*



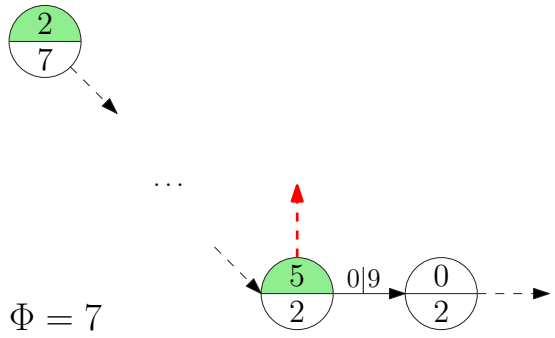
FIFO preflow-push Algorithmus

Beweis Laufzeit

- Sei $\Phi = \max\{d(v) \mid v \in G^f \text{ aktiv}\} \geq 0$
  (Potential, dass gleich dem höchsten aktiven Level ist)
  - Phase mit `relabel`: $\Delta\Phi \leq 1$ (Erhöhung)
    Phase ohne `relabel`: $\Delta\Phi \leq -1$ (Erniedrigung)
  - da $\#_{\text{relabel}} \leq 2n^2$ (siehe Vorlesung)
    $\rightarrow$ maximal $2n^2$ Erhöhungen
    $\rightarrow$ maximal $2n^2$ Erniedrigungen
  - $\rightarrow$ $\#_{\text{phases}} \leq 4n^2$

$\rightarrow$ $\#_{\text{non-saturating}} \leq n \cdot \#_{\text{phases}} \leq n \cdot 4n^2 = 4n^3$

- restlicher Beweis wie bei *generic preflow-push*



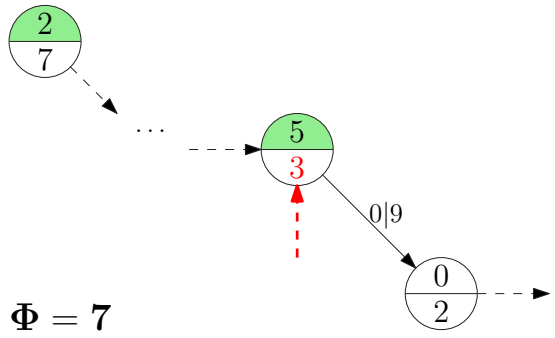
FIFO preflow-push Algorithmus

Beweis Laufzeit

- Sei $\Phi = \max\{d(v) \mid v \in G^f \text{ aktiv}\} \geq 0$
  (Potential, dass gleich dem höchsten aktiven Level ist)
  - Phase mit `relabel`: $\Delta\Phi \leq 1$ (Erhöhung)
    Phase ohne `relabel`: $\Delta\Phi \leq -1$ (Erniedrigung)
  - da $\#_{\text{relabel}} \leq 2n^2$ (siehe Vorlesung)
    $\rightarrow$ maximal $2n^2$ Erhöhungen
    $\rightarrow$ maximal $2n^2$ Erniedrigungen
  - $\rightarrow$ $\#_{\text{phases}} \leq 4n^2$

$\rightarrow$ $\#_{\text{non-saturating}} \leq n \cdot \#_{\text{phases}} \leq n \cdot 4n^2 = 4n^3$

- restlicher Beweis wie bei *generic preflow-push*



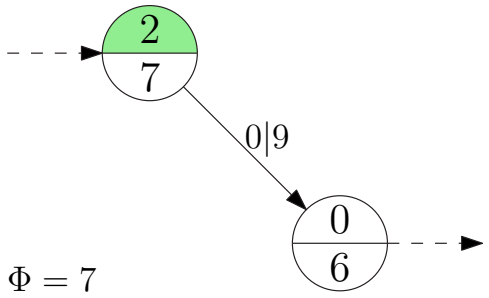
FIFO preflow-push Algorithmus

Beweis Laufzeit

- Sei $\Phi = \max\{d(v) \mid v \in G^f \text{ aktiv}\} \geq 0$
  (Potential, dass gleich dem höchsten aktiven Level ist)
  - Phase mit `relabel`: $\Delta\Phi \leq 1$ (Erhöhung)
    Phase ohne `relabel`: $\Delta\Phi \leq -1$ (Erniedrigung)
  - da $\#_{\text{relabel}} \leq 2n^2$ (siehe Vorlesung)
    $\rightarrow$ maximal $2n^2$ Erhöhungen
    $\rightarrow$ maximal $2n^2$ Erniedrigungen
  - $\rightarrow$ $\#_{\text{phases}} \leq 4n^2$

$\rightarrow$ $\#_{\text{non-saturating}} \leq n \cdot \#_{\text{phases}} \leq n \cdot 4n^2 = 4n^3$

- restlicher Beweis wie bei *generic preflow-push*



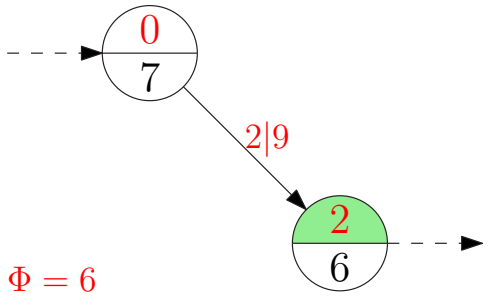
FIFO preflow-push Algorithmus

Beweis Laufzeit

- Sei $\Phi = \max\{d(v) \mid v \in G^f \text{ aktiv}\} \geq 0$
  (Potential, dass gleich dem höchsten aktiven Level ist)
  - Phase mit `relabel`: $\Delta\Phi \leq 1$ (Erhöhung)
    Phase ohne `relabel`: $\Delta\Phi \leq -1$ (Erniedrigung)
  - da $\#_{\text{relabel}} \leq 2n^2$ (siehe Vorlesung)
    $\rightarrow$ maximal $2n^2$ Erhöhungen
    $\rightarrow$ maximal $2n^2$ Erniedrigungen
  - $\rightarrow$ $\#_{\text{phases}} \leq 4n^2$

$\rightarrow$ $\#_{\text{non-saturating}} \leq n \cdot \#_{\text{phases}} \leq n \cdot 4n^2 = 4n^3$

- restlicher Beweis wie bei *generic preflow-push*



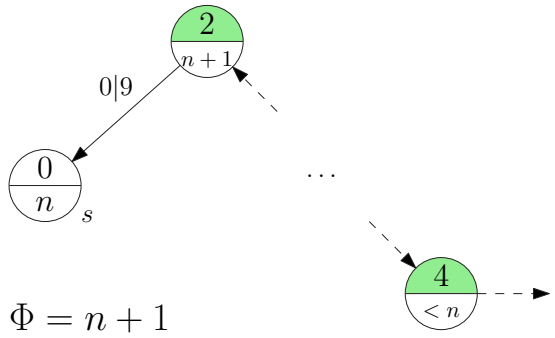
FIFO preflow-push Algorithmus

Beweis Laufzeit

- Sei $\Phi = \max\{d(v) \mid v \in G^f \text{ aktiv}\} \geq 0$
  (Potential, dass gleich dem höchsten aktiven Level ist)
  - Phase mit `relabel`: $\Delta\Phi \leq 1$ (Erhöhung)
    Phase ohne `relabel`: $\Delta\Phi \leq -1$ (Erniedrigung)
  - da $\#_{\text{relabel}} \leq 2n^2$ (siehe Vorlesung)
    $\rightarrow$ maximal $2n^2$ Erhöhungen
    $\rightarrow$ maximal $2n^2$ Erniedrigungen
  - $\rightarrow$ $\#_{\text{phases}} \leq 4n^2$

$\rightarrow$ $\#_{\text{non-saturating}} \leq n \cdot \#_{\text{phases}} \leq n \cdot 4n^2 = 4n^3$

- restlicher Beweis wie bei *generic preflow-push*



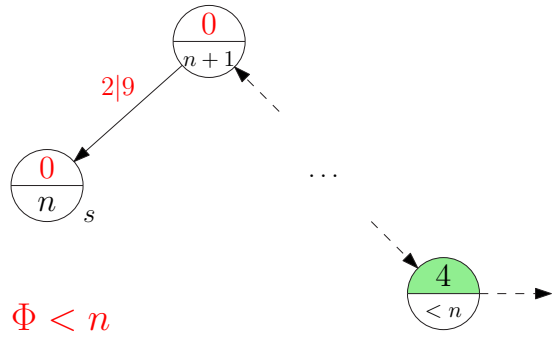
FIFO preflow-push Algorithmus

Beweis Laufzeit

- Sei $\Phi = \max\{d(v) \mid v \in G^f \text{ aktiv}\} \geq 0$
  (Potential, dass gleich dem höchsten aktiven Level ist)
  - Phase mit `relabel`: $\Delta\Phi \leq 1$ (Erhöhung)
    Phase ohne `relabel`: $\Delta\Phi \leq -1$ (Erniedrigung)
  - da $\#_{\text{relabel}} \leq 2n^2$ (siehe Vorlesung)
    $\rightarrow$ maximal $2n^2$ Erhöhungen
    $\rightarrow$ maximal $2n^2$ Erniedrigungen
  - $\rightarrow$ $\#_{\text{phases}} \leq 4n^2$

$\rightarrow$ $\#_{\text{non-saturating}} \leq n \cdot \#_{\text{phases}} \leq n \cdot 4n^2 = 4n^3$

- restlicher Beweis wie bei *generic preflow-push*



FIFO preflow-push Algorithmus

Beweis Laufzeit

- Sei $\Phi = \max\{d(v) \mid v \in G^f \text{ aktiv}\} \geq 0$
  (Potential, dass gleich dem höchsten aktiven Level ist)
  - Phase mit `relabel`: $\Delta\Phi \leq 1$ (Erhöhung)
    Phase ohne `relabel`: $\Delta\Phi \leq -1$ (Erniedrigung)
  - da $\#_{\text{relabel}} \leq 2n^2$ (siehe Vorlesung)
    $\rightarrow$ maximal $2n^2$ Erhöhungen
    $\rightarrow$ maximal $2n^2$ Erniedrigungen

  $\rightarrow$ $\#_{\text{phases}} \leq 4n^2$

$\rightarrow$ $\#_{\text{non-saturating}} \leq n \cdot \#_{\text{phases}} \leq n \cdot 4n^2 = 4n^3$

- restlicher Beweis wie bei *generic preflow-push*

FIFO preflow-push Algorithmus

Laufzeit

Lemmas

- $T_{\texttt{push}\text{ op}} = T_{\texttt{relabel}\text{ op}} = T_{\texttt{node selection}} = \mathcal{O}(1)$
- $\#_{\text{relabels}} \leq 2n^2$ (Lemma 7)
- $(\#_{\text{pushes}} + \#_{\text{relabels}}) \cdot T_{\text{edge selection}} \leq 4nm$ (Lemma 10)
- $\#_{\text{pushes}} = \#_{\text{saturating}} + \#_{\text{non-saturating}}$
- $\#_{\text{saturating}} \leq nm$ (Lemma 8)
- $\#_{\text{non-saturating}} \in \mathcal{O}(n^2 m)$ (Lemma 9)
- $T_{\text{generic preflow-push}} = T_{\text{init}} + T_{\text{pushes}} + T_{\text{relabels}} \in \mathcal{O}(n^2 m)$
  - $T_{\text{init}} = n + m$
  - $T_{\text{pushes}} = \#_{\text{pushes}} \cdot (T_{\text{node selection}} + T_{\text{edge selection}} + T_{\texttt{push}\text{ op}})$
  - $T_{\text{relabels}} = \#_{\text{relabels}} \cdot (T_{\text{node selection}} + T_{\text{edge selection}} + T_{\texttt{relabel}\text{ op}})$

FIFO preflow-push Algorithmus

Laufzeit

Lemmas

- $T_{\texttt{push op}} = T_{\texttt{relabel op}} = T_{\texttt{node selection}} = \mathcal{O}(1)$
- $\#_{\text{relabels}} \leq 2n^2$ (Lemma 7)
- $(\#_{\text{pushes}} + \#_{\text{relabels}}) \cdot T_{\text{edge selection}} \leq 4nm$ (Lemma 10)
- $\#_{\text{pushes}} = \#_{\text{saturating}} + \#_{\text{non-saturating}}$
- $\#_{\text{saturating}} \leq nm$ (Lemma 8)
- $\#_{\text{non-saturating}} \in \mathcal{O}(n^3)$ (FIFO)
- $T_{\text{FIFO preflow-push}} = T_{\text{init}} + T_{\text{pushes}} + T_{\text{relabels}} \in \mathcal{O}(n^3)$
  - $T_{\text{init}} = n + m$
  - $T_{\text{pushes}} = \#_{\text{pushes}} \cdot (T_{\text{node selection}} + T_{\text{edge selection}} + T_{\texttt{push op}})$
  - $T_{\text{relabels}} = \#_{\text{relabels}} \cdot (T_{\text{node selection}} + T_{\text{edge selection}} + T_{\texttt{relabel op}})$

preflow-push Algorithmus

Heuristiken

Relabeling

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

Knotenauswahl

- *two-phase approach*

preflow-push Algorithmus

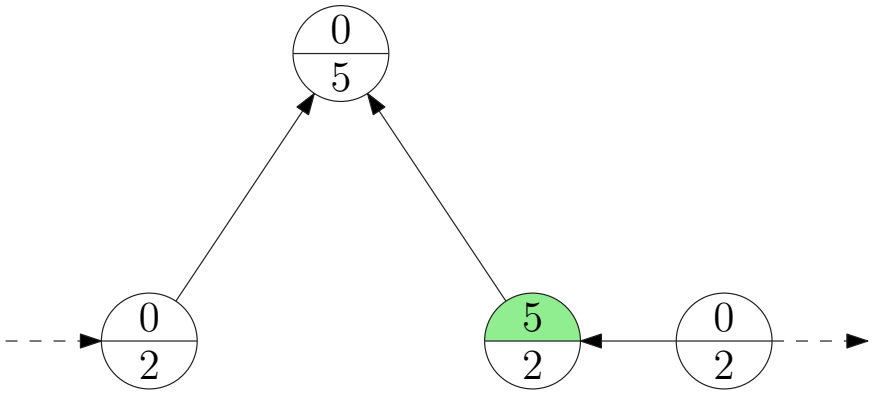
Heuristiken

- *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 Algorithmus

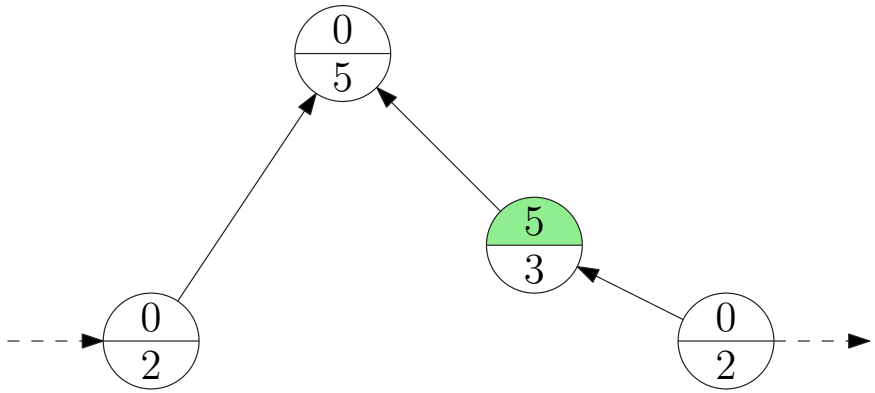
Heuristiken

- *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 Algorithmus

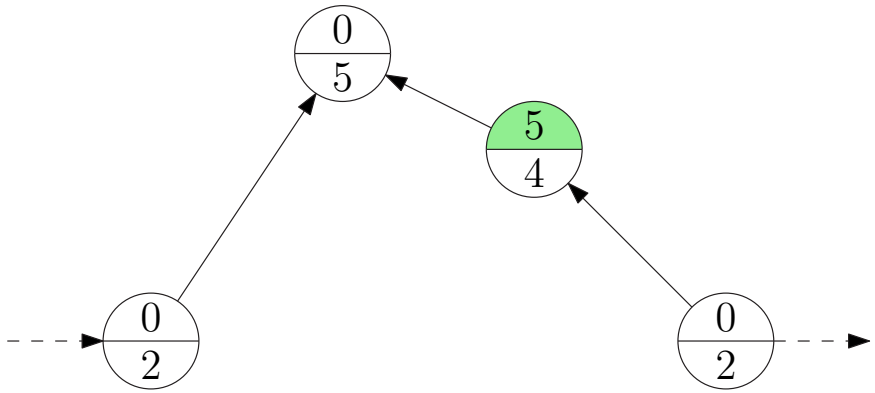
Heuristiken

- *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 Algorithmus

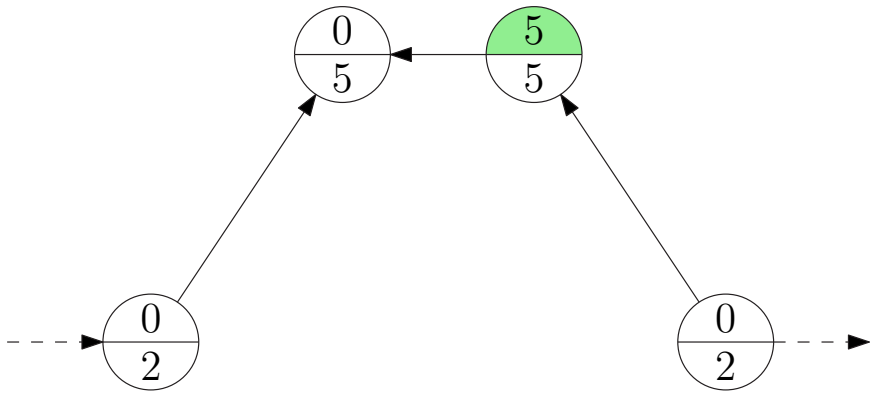
Heuristiken

- *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 Algorithmus

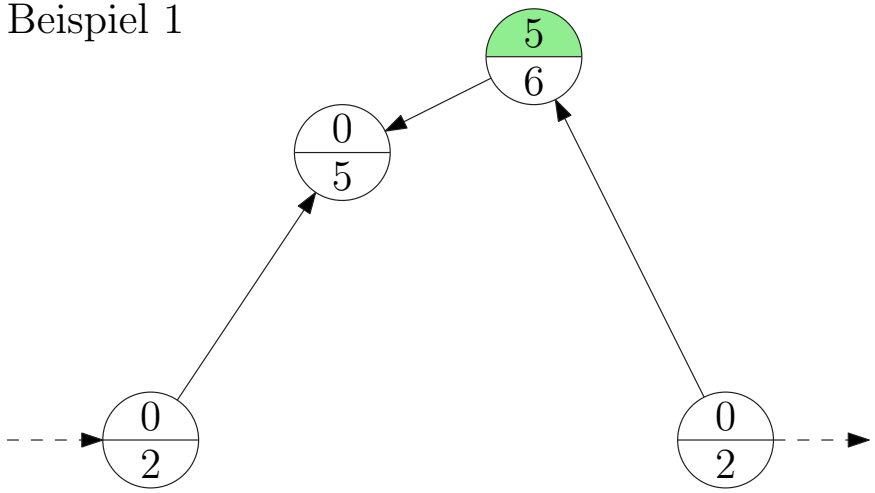
Heuristiken

- *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 Algorithmus

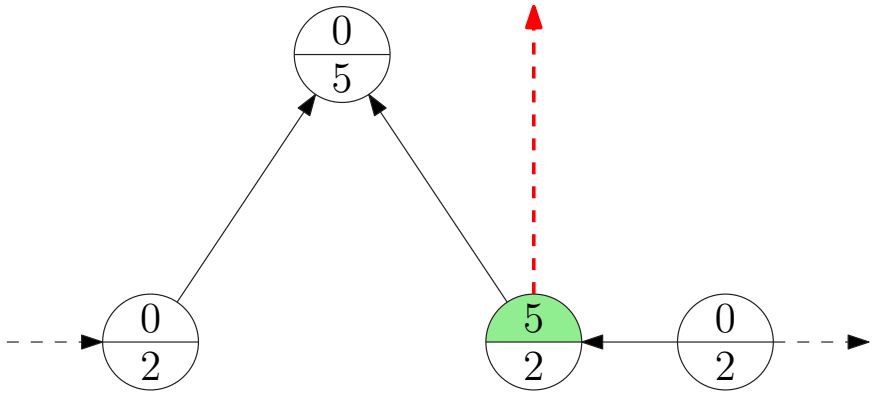
Heuristiken

- *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 Algorithmus

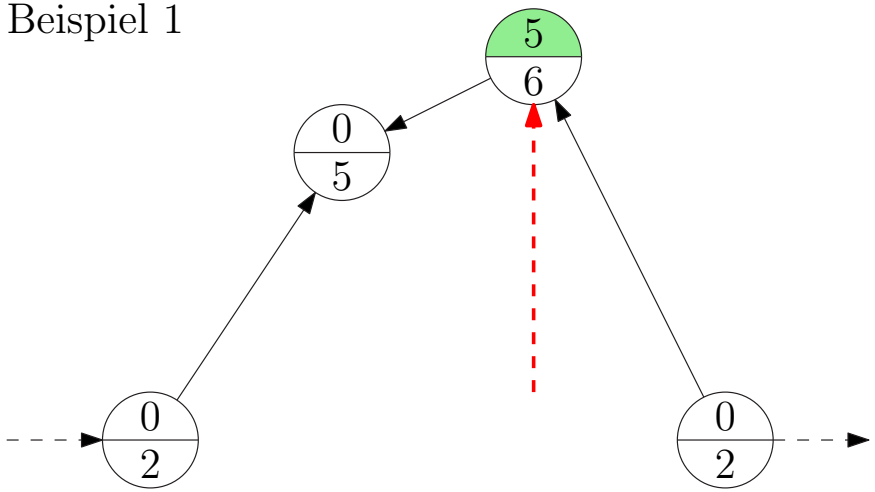
Heuristiken

- *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 Algorithmus

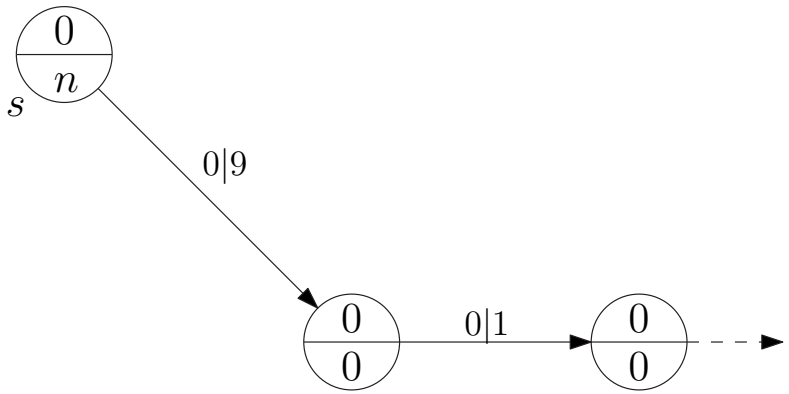
Heuristiken

- *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 Algorithmus

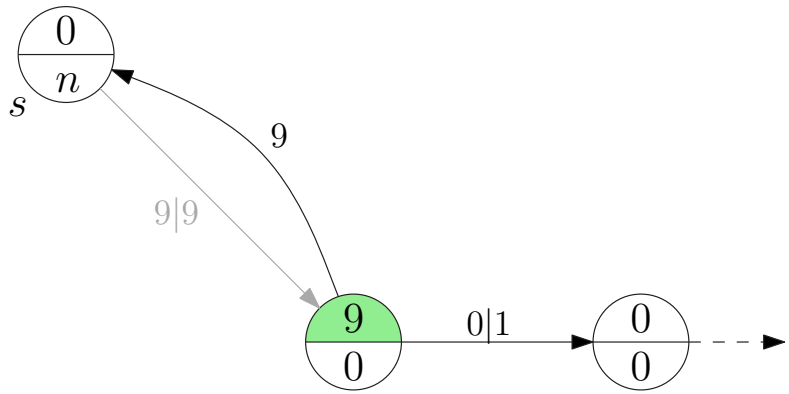
Heuristiken

- *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 Algorithmus

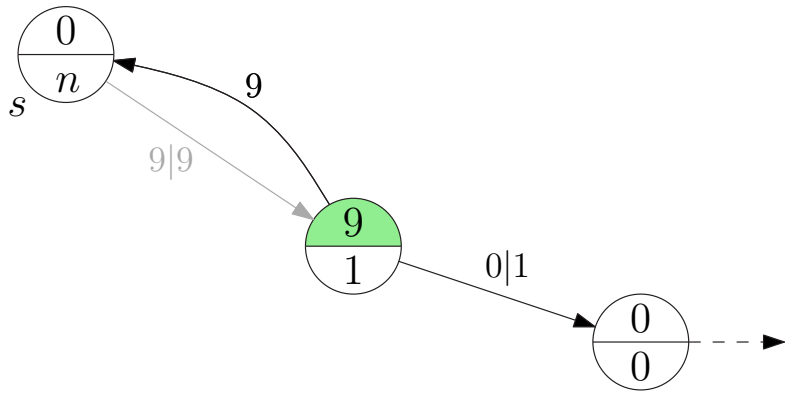
Heuristiken

- *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 Algorithmus

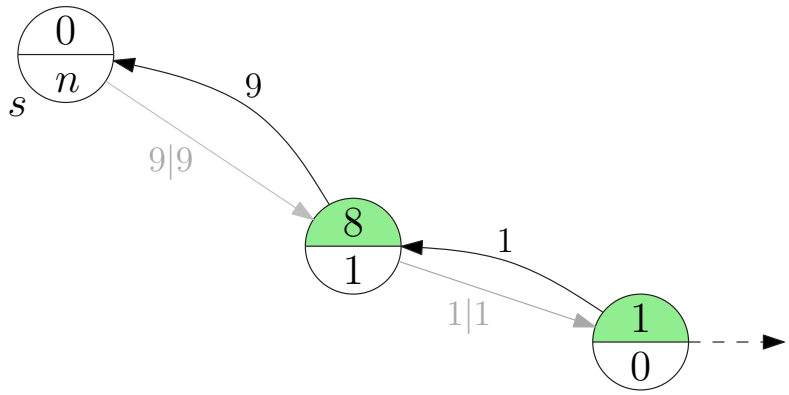
Heuristiken

- *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 Algorithmus

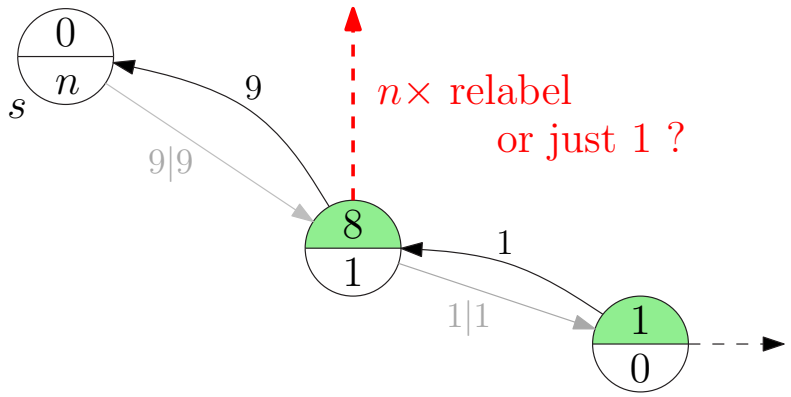
Heuristiken

- *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 Algorithmus

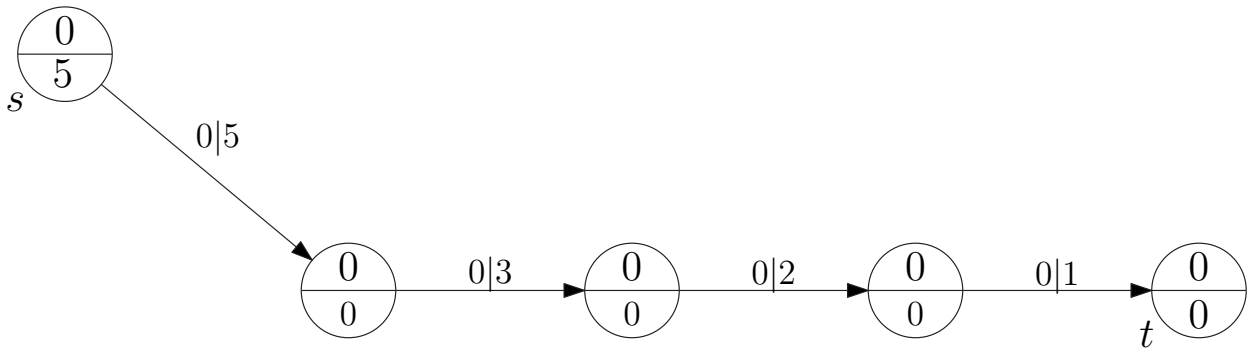
Heuristiken

- *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 Algorithmus

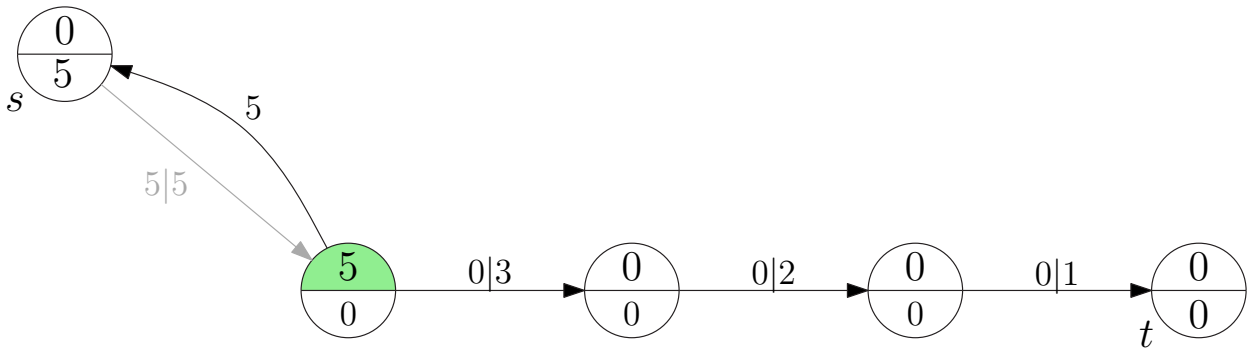
Heuristiken

- *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 Algorithmus

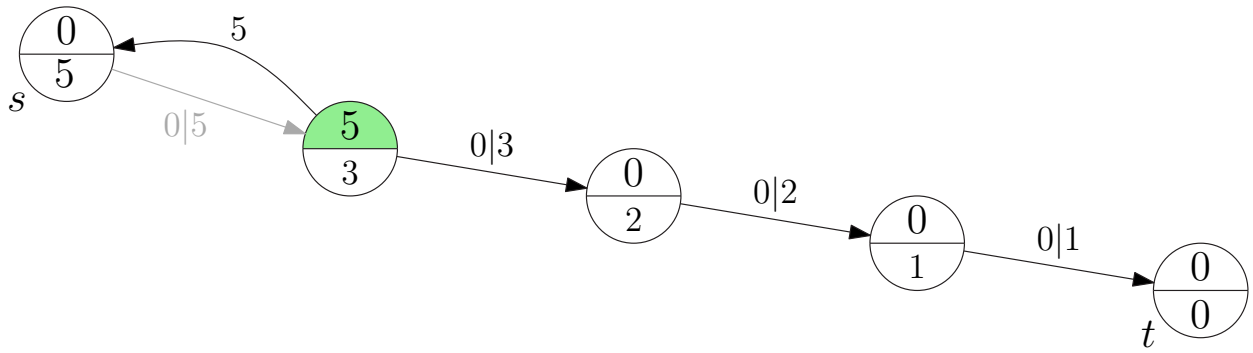
Heuristiken

- *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 Algorithmus

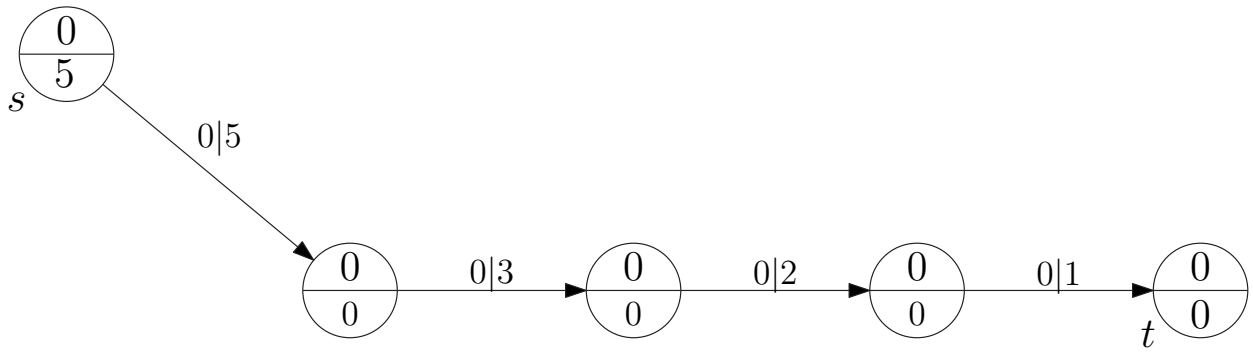
Heuristiken

- *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 Algorithmus

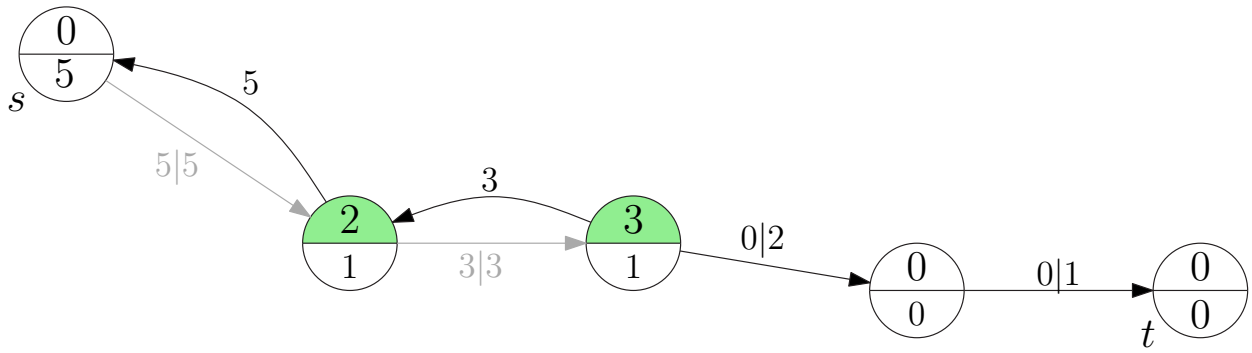
Heuristiken

- *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 Algorithmus

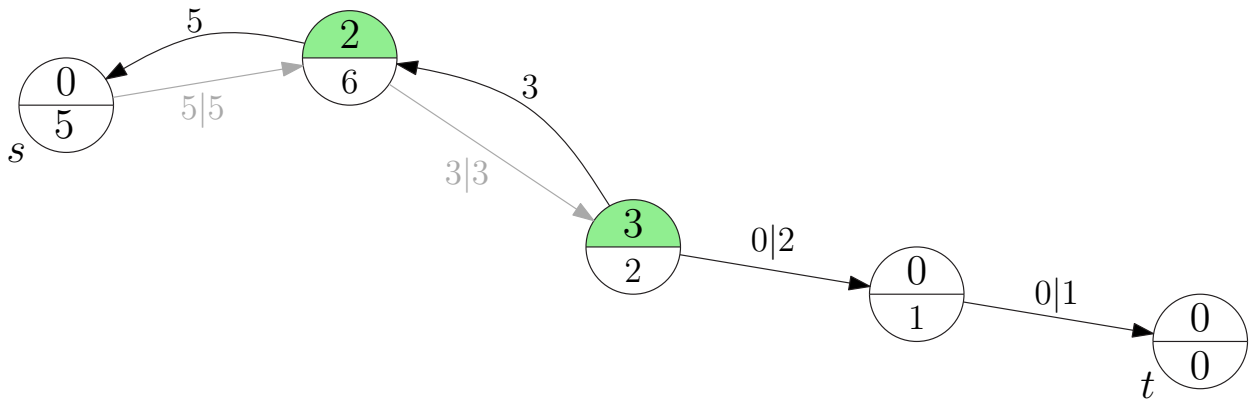
Heuristiken

- *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



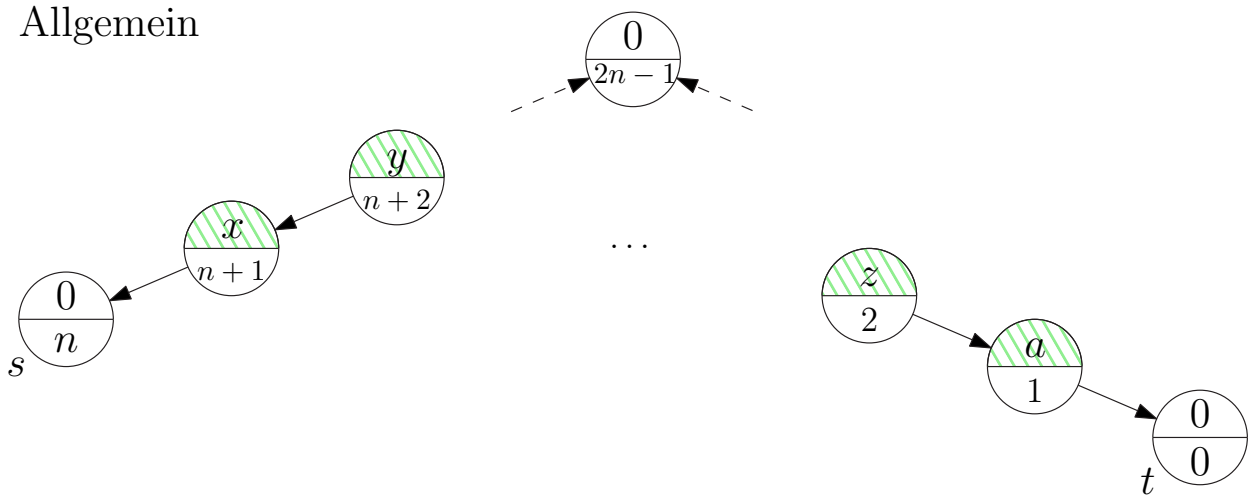
preflow-push Algorithmus

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)



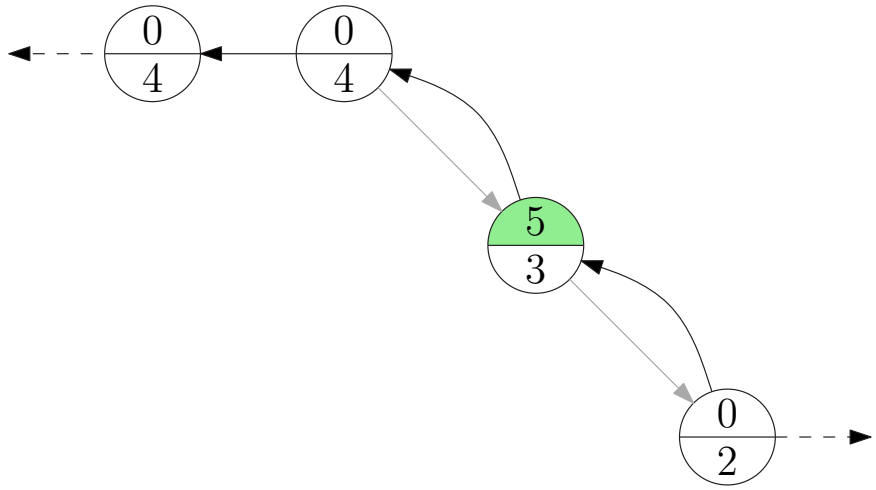
preflow-push Algorithmus

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)



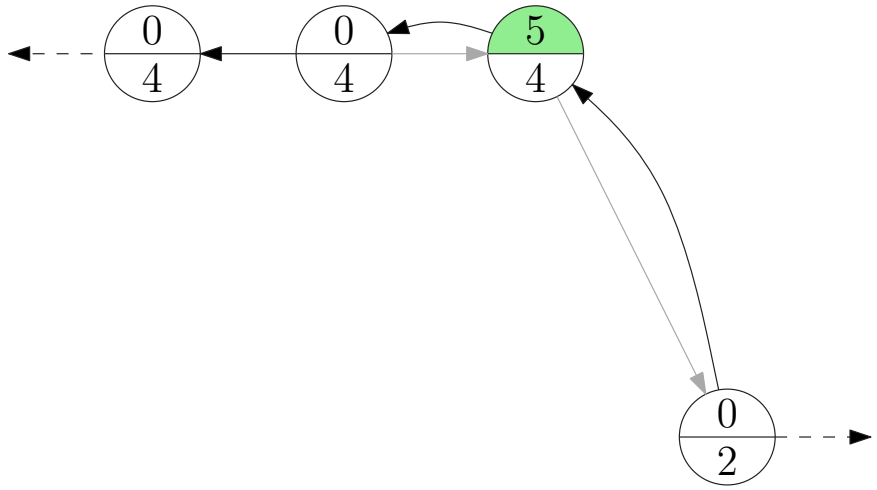
preflow-push Algorithmus

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)



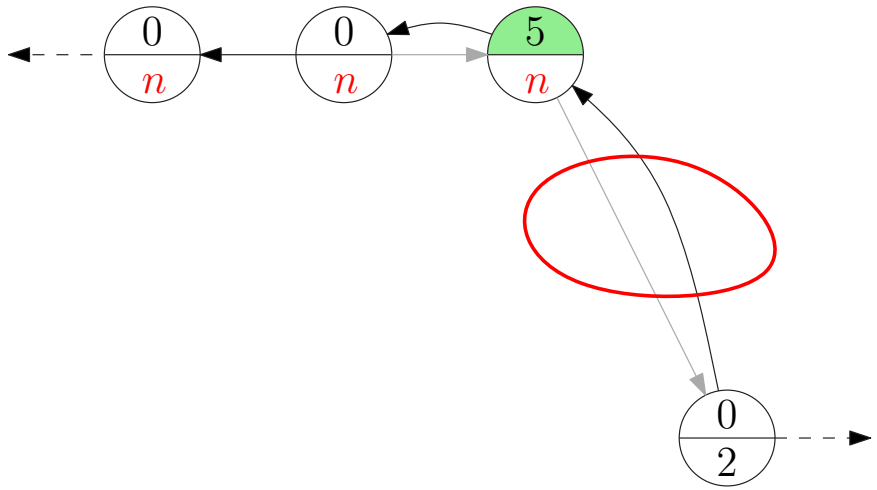
preflow-push Algorithmus

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)



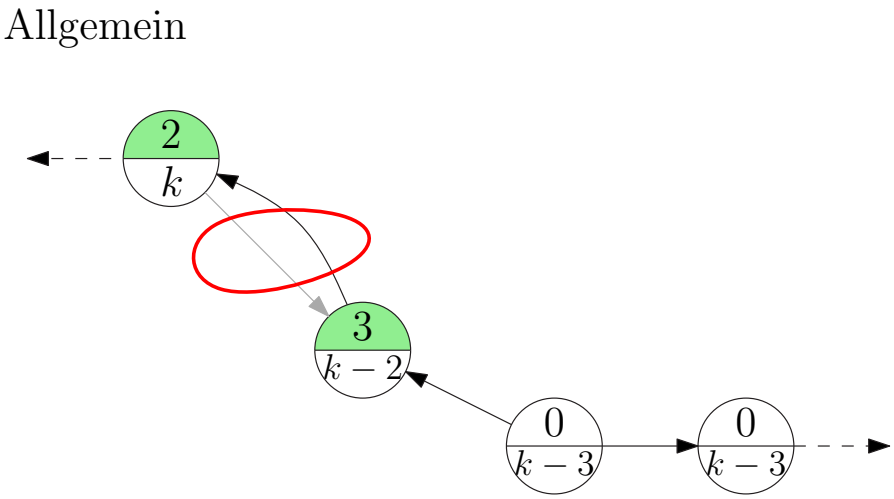
preflow-push Algorithmus

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)



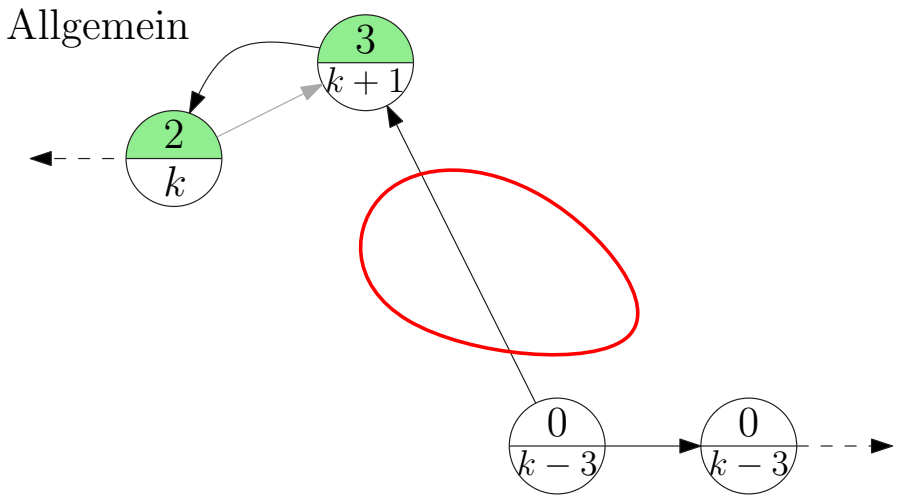
preflow-push Algorithmus

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)



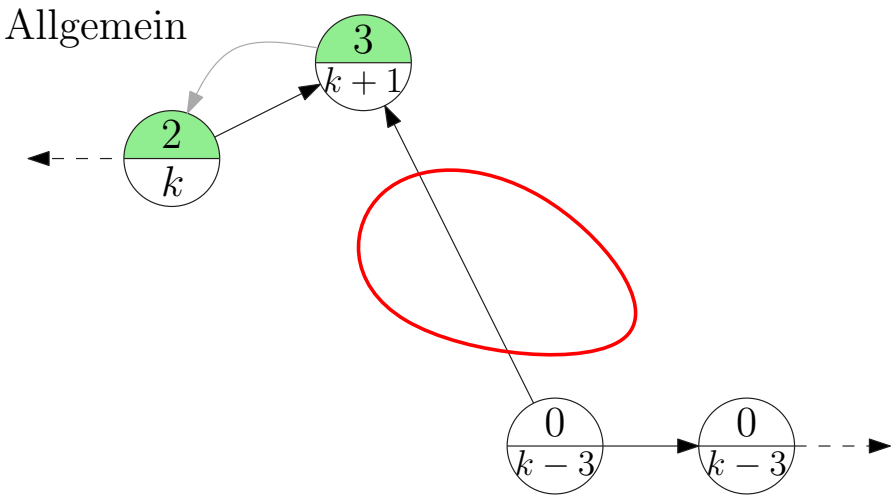
preflow-push Algorithmus

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)



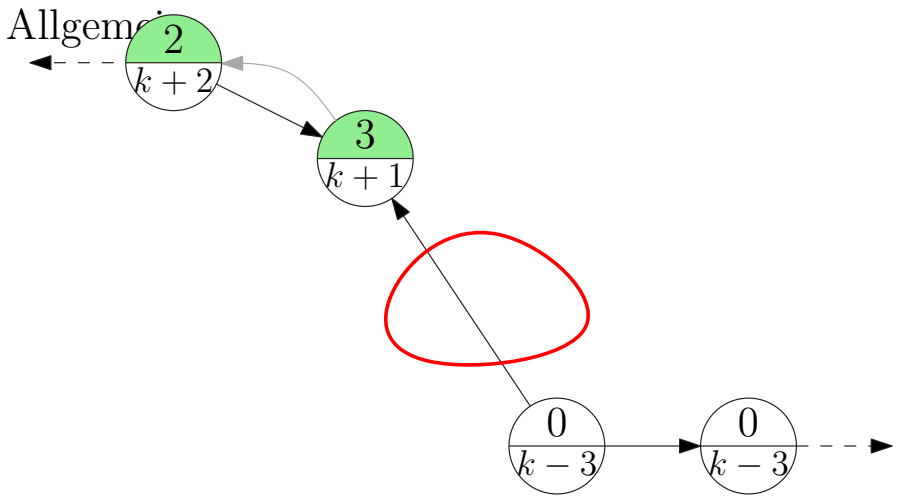
preflow-push Algorithmus

Heuristiken

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

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

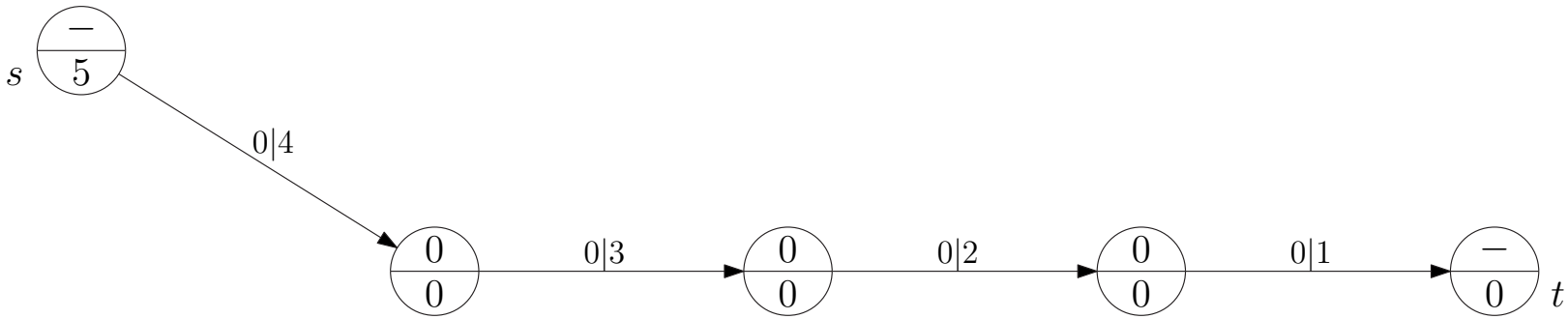
(Lücke kann nie mehr überwunden werden, Fluss nur noch zurück zu s)



preflow-push Algorithmus

Heuristiken

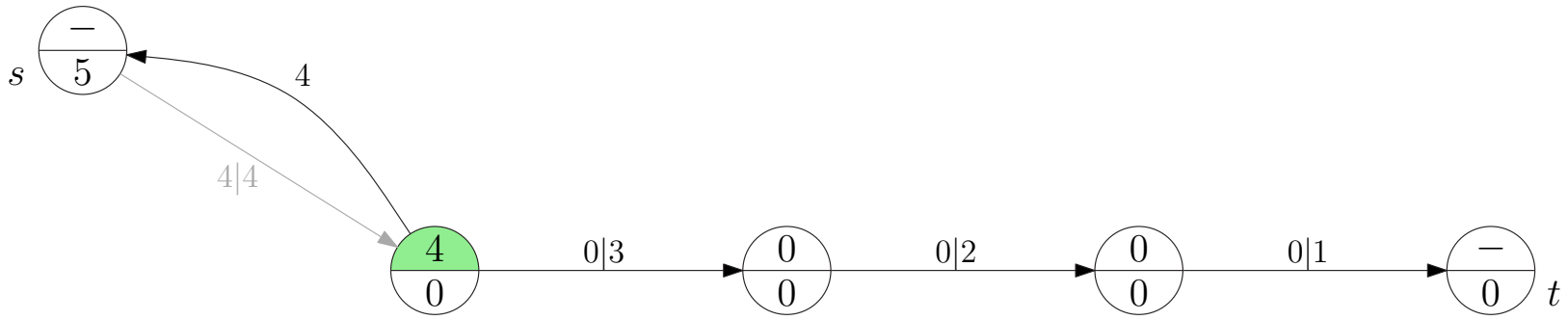
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\rightarrow$ erzeugt *maximum preflow*
    $\rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\rightarrow$ Fluss nur nach s möglich



preflow-push Algorithmus

Heuristiken

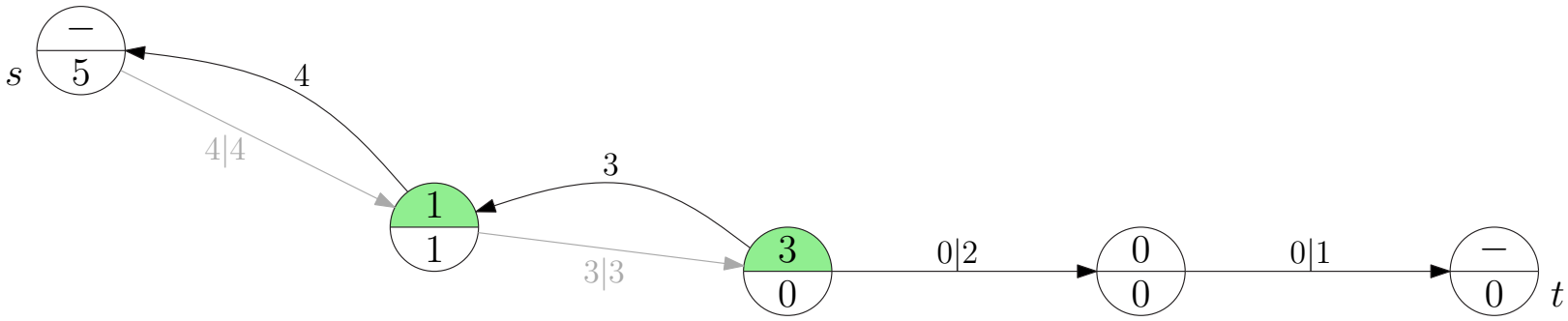
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\rightarrow$ erzeugt *maximum preflow*
    $\rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\rightarrow$ Fluss nur nach s möglich



preflow-push Algorithmus

Heuristiken

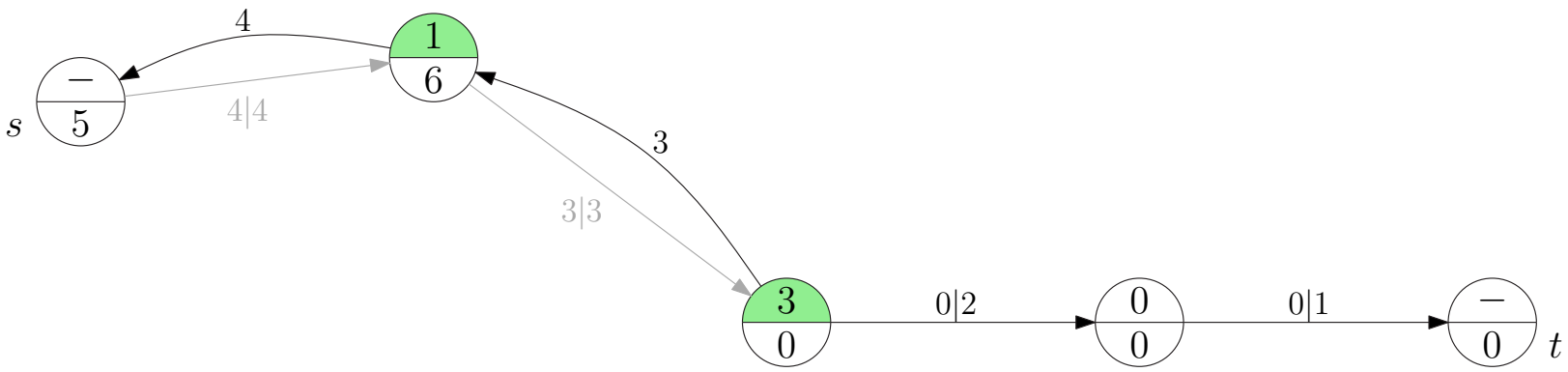
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\rightarrow$ erzeugt *maximum preflow*
    $\rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\rightarrow$ Fluss nur nach s möglich



preflow-push Algorithmus

Heuristiken

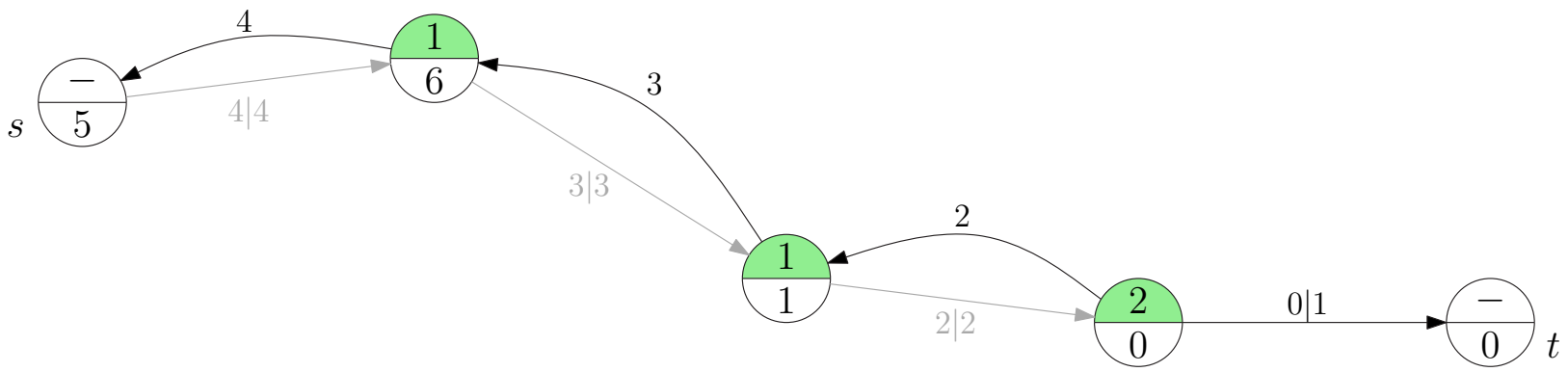
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\rightarrow$ erzeugt *maximum preflow*
    $\rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\rightarrow$ Fluss nur nach s möglich



preflow-push Algorithmus

Heuristiken

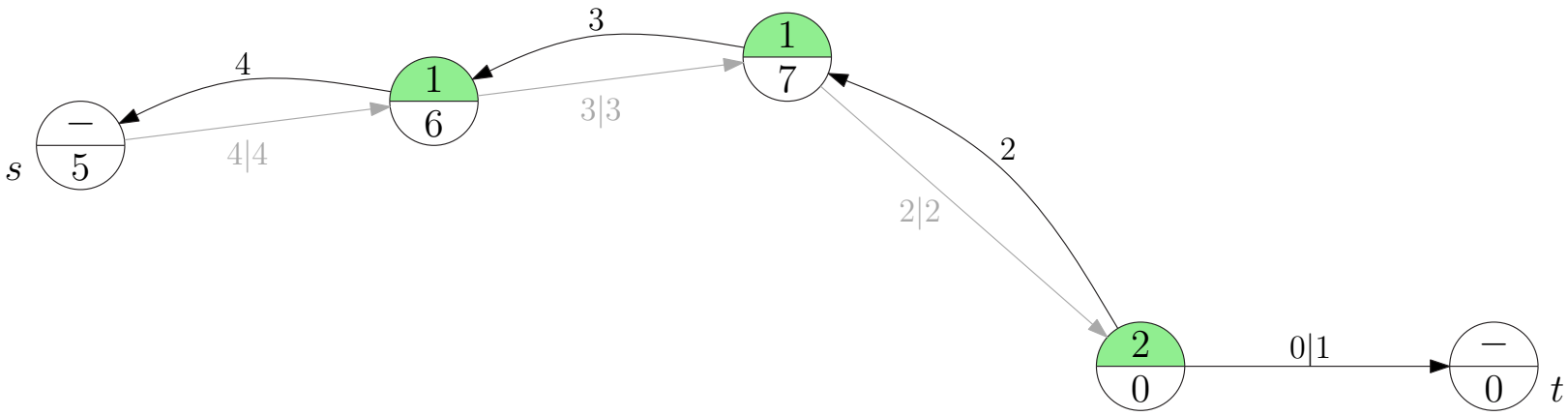
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\rightarrow$ erzeugt *maximum preflow*
    $\rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\rightarrow$ Fluss nur nach s möglich



preflow-push Algorithmus

Heuristiken

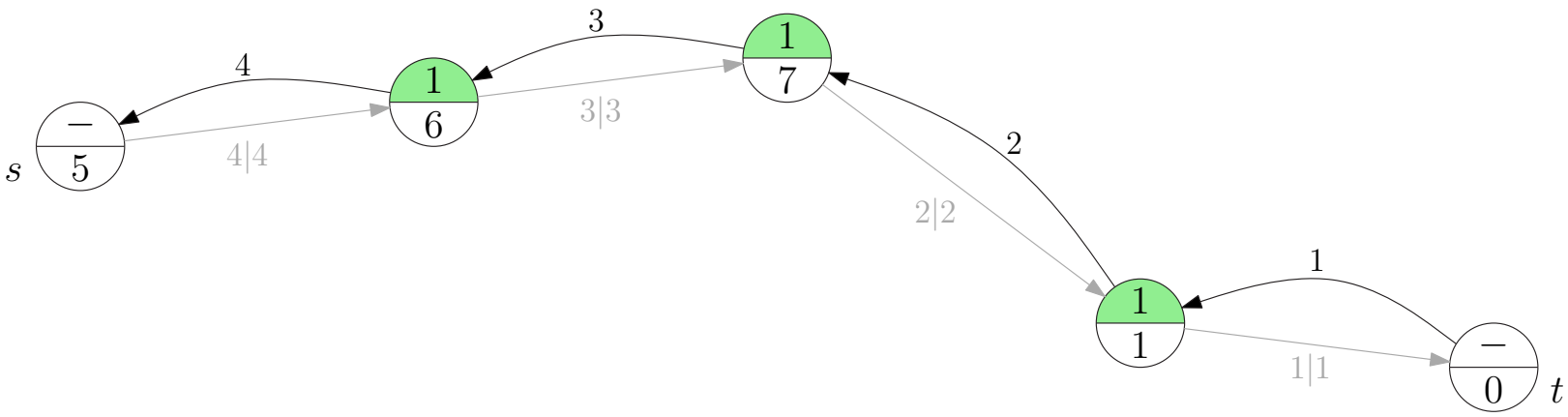
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\rightarrow$ erzeugt *maximum preflow*
    $\rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\rightarrow$ Fluss nur nach s möglich



preflow-push Algorithmus

Heuristiken

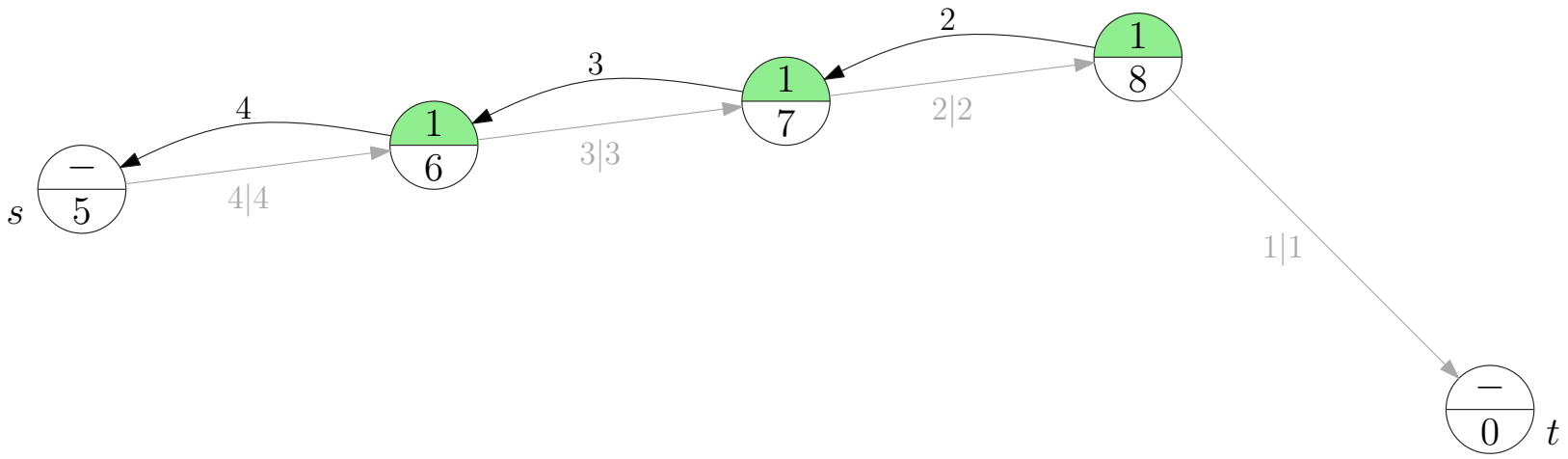
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\rightarrow$ erzeugt *maximum preflow*
    $\rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\rightarrow$ Fluss nur nach s möglich



preflow-push Algorithmus

Heuristiken

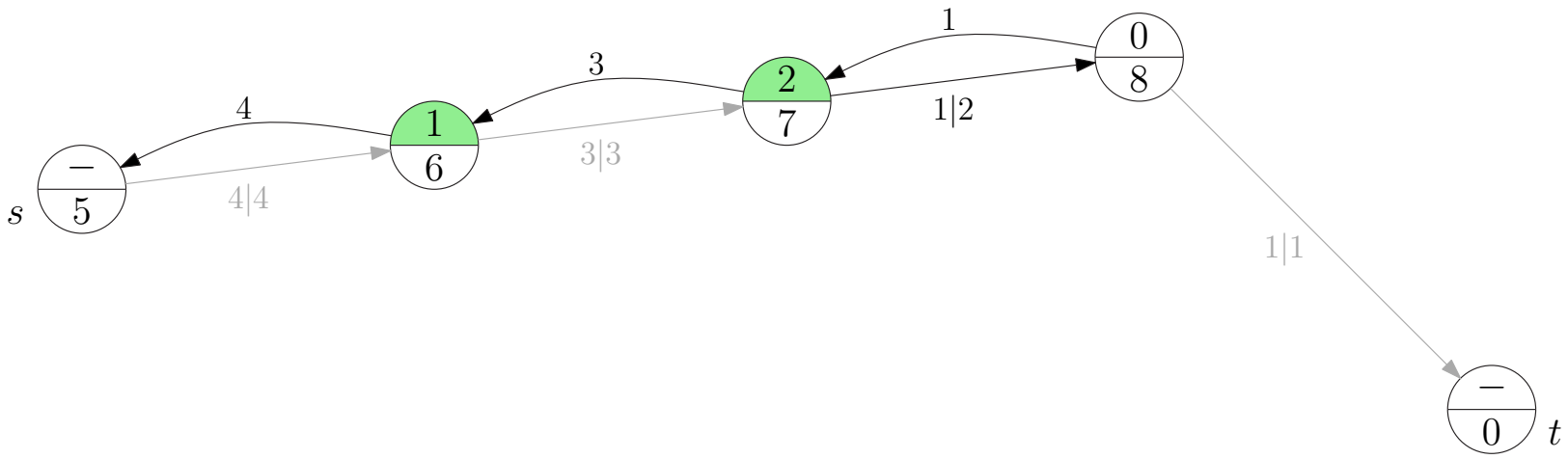
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\rightarrow$ erzeugt *maximum preflow*
    $\rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\rightarrow$ Fluss nur nach s möglich



preflow-push Algorithmus

Heuristiken

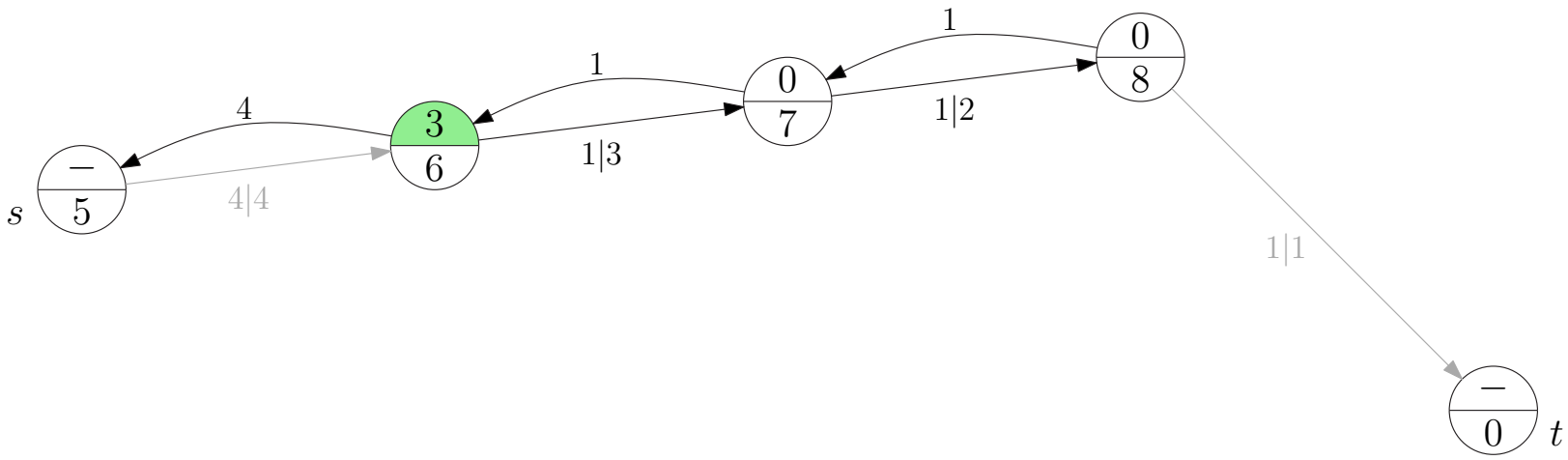
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\rightarrow$ erzeugt *maximum preflow*
    $\rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\rightarrow$ Fluss nur nach s möglich



preflow-push Algorithmus

Heuristiken

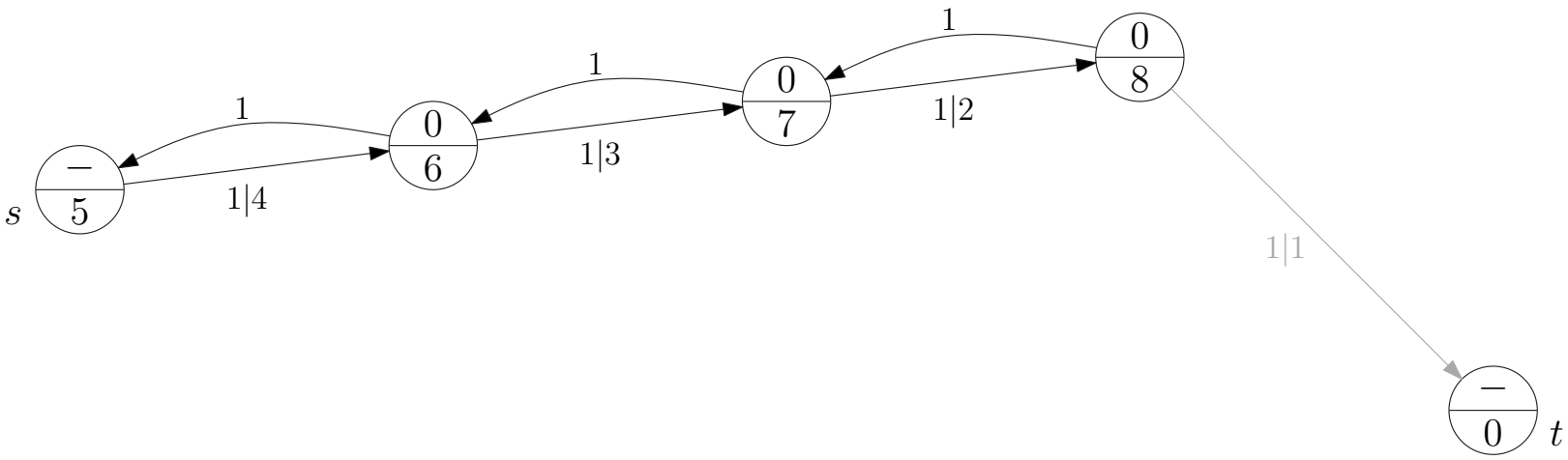
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\rightarrow$ erzeugt *maximum preflow*
    $\rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\rightarrow$ Fluss nur nach s möglich



preflow-push Algorithmus

Heuristiken

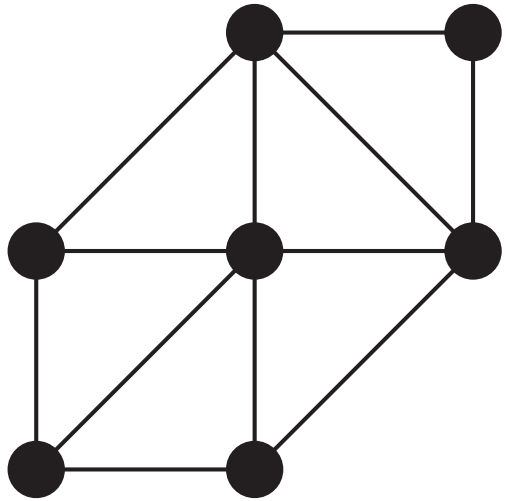
- *two-phase approach*
  - Phase 1: wähle nur Knoten Level $d(v) < n$ aus
    $\rightarrow$ erzeugt *maximum preflow*
    $\rightarrow$ korrekter Fluss in t
  - Phase 2: nur noch Knoten mit Level $d(v) \geq n$ übrig
    $\rightarrow$ Fluss nur nach s möglich



Matching

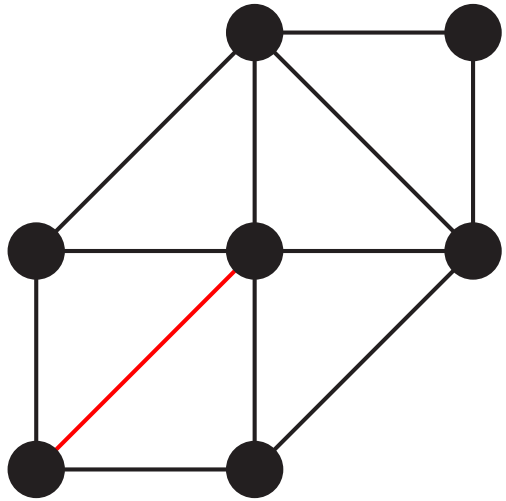
Matching

- Teilmenge von Kanten eines Graphen
- Kein Knoten inzident zu mehr als einer (gematchten) Kante
- **Maximal**: keine Kante hinzunehmbar
- **Maximum**: kein Matching hat größere Kardinalität
- **Perfekt**: jeder Knoten ist Endpunkt einer gematchten Kante
- **Kardinalität**: Zahl der Kanten im Matching
- Maximum Matching im allgemeinen über alternierende Pfade berechnet



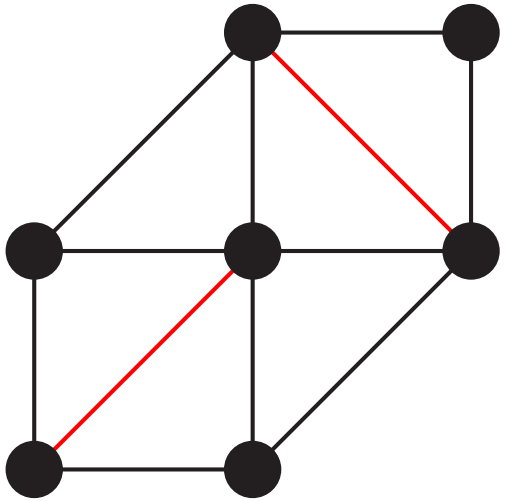
Matching

- Teilmenge von Kanten eines Graphen
- Kein Knoten inzident zu mehr als einer (gematchten) Kante
- **Maximal**: keine Kante hinzunehmbar
- **Maximum**: kein Matching hat größere Kardinalität
- **Perfekt**: jeder Knoten ist Endpunkt einer gematchten Kante
- **Kardinalität**: Zahl der Kanten im Matching
- Maximum Matching im allgemeinen über alternierende Pfade berechnet



Matching

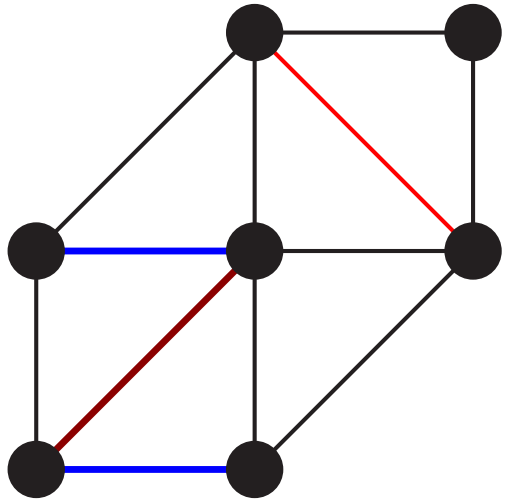
- Teilmenge von Kanten eines Graphen
- Kein Knoten inzident zu mehr als einer (gematchten) Kante
- **Maximal**: keine Kante hinzunehmbar
- **Maximum**: kein Matching hat größere Kardinalität
- **Perfekt**: jeder Knoten ist Endpunkt einer gematchten Kante
- **Kardinalität**: Zahl der Kanten im Matching
- Maximum Matching im allgemeinen über alternierende Pfade berechnet



maximal aber nicht maximum

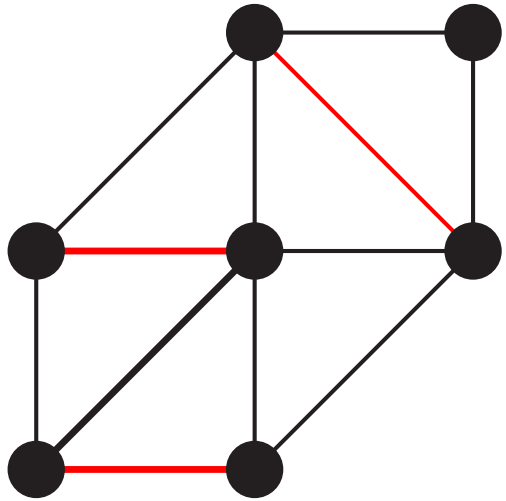
Matching

- Teilmenge von Kanten eines Graphen
- Kein Knoten inzident zu mehr als einer (gematchten) Kante
- **Maximal**: keine Kante hinzunehmbar
- **Maximum**: kein Matching hat größere Kardinalität
- **Perfekt**: jeder Knoten ist Endpunkt einer gematchten Kante
- **Kardinalität**: Zahl der Kanten im Matching
- Maximum Matching im allgemeinen über alternierende Pfade berechnet



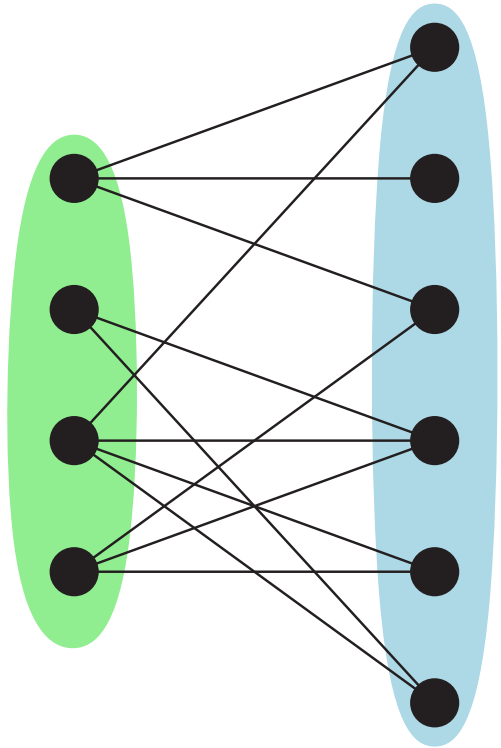
Matching

- Teilmenge von Kanten eines Graphen
- Kein Knoten inzident zu mehr als einer (gematchten) Kante
- **Maximal**: keine Kante hinzunehmbar
- **Maximum**: kein Matching hat größere Kardinalität
- **Perfekt**: jeder Knoten ist Endpunkt einer gematchten Kante
- **Kardinalität**: Zahl der Kanten im Matching
- Maximum Matching im allgemeinen über alternierende Pfade berechnet



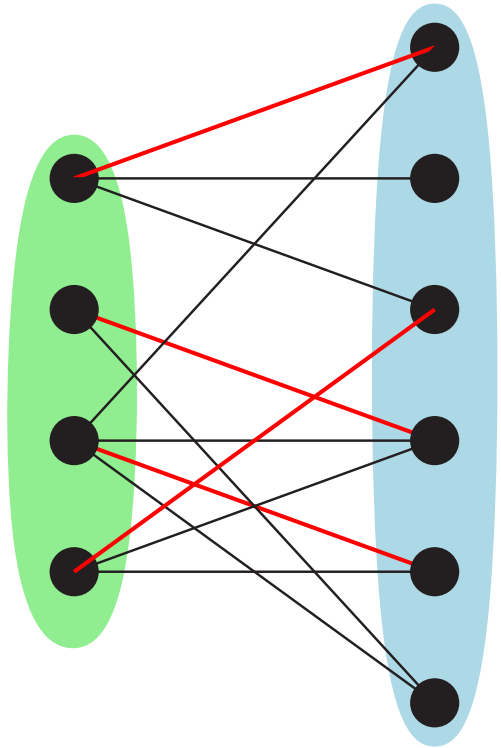
Bipartite - Matching

- Bipartiter Graph: zwei Gruppen von Knoten, Kanten nicht innerhalb einer Gruppe
- modelliere als Flussproblem mit Knotenkapazitäten
- Fluss zwischen den Teilmengen entspricht gematchten Kanten



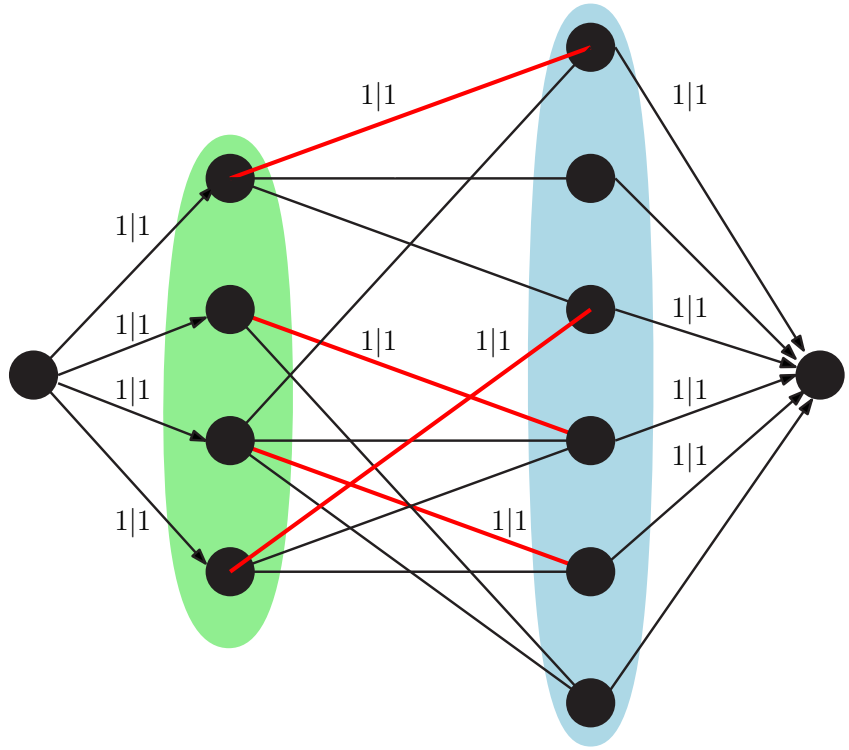
Bipartite - Matching

- Bipartiter Graph: zwei Gruppen von Knoten, Kanten nicht innerhalb einer Gruppe
- modelliere als Flussproblem mit Knotenkapazitäten
- Fluss zwischen den Teilmengen entspricht gematchten Kanten



Bipartite - Matching

- Bipartiter Graph: zwei Gruppen von Knoten, Kanten nicht innerhalb einer Gruppe
- modelliere als Flussproblem mit Knotenkapazitäten
- Fluss zwischen den Teilmengen entspricht gematchten Kanten



Matching
Anwendungen

- Anwendungen in sämtlichen Zuweisungsproblemen
- . . . Jobs auf Maschinen
- . . . Crewscheduling